Evolved Control of Natural Plants: Crossing the Reality Gap for User-defined Steering of Growth and Motion

Daniel Nicolas Hofstadler, University of Graz
Mostafa Wahby, University of Lübeck
Mary Katherine Heinrich, Centre for IT and Architecture
Heiko Hamann, University of Lübeck
Payam Zahadat, University of Graz
Phil Ayres, Centre for IT and Architecture
Thomas Schmickl, University of Graz

1 Abstract

Mixing societies of natural and artificial systems can provide interesting and potentially fruitful research targets. Here we mix robotic setups and natural plants in order to steer the motion behavior of plants while growing. The robotic setup uses a camera to observe the plant and uses a pair of light sources to trigger phototropic response, steering the plant to user-defined targets. An evolutionary robotic approach is used to design a controller for the setup. Initially, preliminary experiments are performed with a simple predetermined controller and a growing bean plant. The plant behavior in response to the simple controller is captured by image processing and a model of the plant tip dynamics is developed. The model is used in simulation to evolve a robot controller that steers the plant tip such that it follows a number of randomly generated target points. Finally, we test the simulation evolved controller in the real setup controlling a natural bean plant. The results demonstrate a successful crossing of the reality gap in the setup. The success of the approach allows for future extensions to more complex tasks including control of the shape of plants and pattern formation in multiple plant setups.

2 Introduction

The concept of mixed societies of living systems and autonomous robots self-organizing with each other and forming bio-hybrid systems has attracted a lot of interest over the last years. While the large concentration of the work in this area deals with combining animals and robots, e.g., [13, 3, 7, 12, 26], we are

interested in mixed societies of plants and robots. Our long-term objective in the context of the EU-funded project *flora robotica* [10, 14] is to bring together the best aspects of both worlds and to generate synergies between them. Plants are efficient in growing. They sense their dynamic environment and react to it by changing their shape in a way that the overall structure is benefited [11]. Robots on the other hand are programmable. We can build robots and program them such that they influence the growth of plants by imposing desired stimuli. They can also extend the plants' sensing and decision-making capabilities.

One of the challenges in this mixed-society approach is the extremely different time scale of plant growth control that differs in several magnitudes from motion control of mobile robots. In comparison to many other living systems, plants are slow in many of their activities including, of course, their growth. For example, the common bean plant (*Phaseolus vulgaris*), which is considered to be a fast growing plant, grows on average 3 cm per day [4]. In addition to growth, plants also show motion, which is often ignored due to its low speed. Bean shoots' intrinsic motion (circumnutation [22]) allows the plant tips to explore their local environment and – together with phototropism (i.e., directed growth towards or away from light [6]) – influence growth to approach more preferable regions. Plant motion seems underestimated by many, likely because their speed is very slow in relation to time scales of human perception. However, on these slow time scales of plants activities, the speed of motion is still considerably faster than the speed of growth. For example, according to our preliminary experiments, bean plants (longer than 20 cm) bend towards a light source with a velocity of up to 4.4 mm/min. Angular velocities of the intrinsic circumnutation reported in literature are even larger [17].

Other challenges of this bio-hybrid approach concern the tasks of sensing and actuation by the robotic side of the system. The robot needs to detect the plant's position to allow for closed-loop control. The robot also needs to impose appropriate stimuli at appropriate times to influence the plant in the desired way.

In this paper, we investigate how the motion and growth of a plant can be influenced by a robotic hardware setup that uses light as an attractive stimulus. The light sources in the setup are controlled by a closed-loop controller detecting the plant's tip and reacting to its position such that the tip reaches a number of target points. For that, we first create an appropriate simulator that addresses relevant features of the plant growth based on the data collected from a set of preliminary experiments with the plant and the light sources controlled by a predefined open-loop controller.

There is vast variety of plant models in the literature. Most models from the field of plant science focus on partial aspects of plant systems or are too detailed and too complex for use in robot controllers (e.g., [1]). In the context of research in self-organization, e.g., in artificial life, a number of usefully low-complexity abstract models of plant growth have been reported. In L-systems [16], a set of rules are iteratively applied to a string of symbols (grammars). The rules process symbols and expand the string whereas certain symbols are interpreted as geometrical structures. The L-system is extended, for example in swarm gram-

mars [23], such that the reaction of the plants to their environmental stimuli is included in the model. The individual nodes then act as autonomous reactive agents that can be attracted to light sources. Bending of plants (motion) is approximated by considering the stiffness of the connected stem elements and the attraction by the light source in the environment [9]. In [28] and [21] abstract branching trees are derived from the inverse computation based on polygon meshes of the geometry of an actual tree and its variations. The plant growth is also modeled focusing on the self-organization and competition between branches and it is used as distributed controller driving the development of artificial structures [27].

Despite the variety of available models, we follow a purpose-specific data-driven approach that is based on acquired data of the plant behavior in our actual setup. We find that creating a model specific to our purpose is very efficient and successful as reported in the following. Hence, our approach could serve as a positive example for future applications in similar contexts. First, we build a simple model of the combined growth and motion behaviors of our plant, based on data extracted from processed images taken in our experimental setup. Then, we use the model to simulate plant behavior and apply evolutionary computation methods to generate our closed-loop controller of the bio-hybrid.

Evolutionary methods have been successfully applied to generate controllers in many robotic applications [2]. The evaluation of an individual controller's fitness can be a costly and especially time-consuming task where embodied evolution [25] is applied and the controller is directly evolved on the actual hardware. Hence, we simplify the process by evolving the controllers based on the derived models in order to achieve a considerable speed-up. The drawback is the so-called reality gap problem [15]. It refers to the often experienced problem that controllers developed in simulation may perform poorly in reality due to limitations of the simulation and unknown features in reality.

In the following[1], we apply evolutionary methods to evolve a closed-loop controller directing the tip of a bean plant by using a pair of light sources. The task is to have the plant tip approach a set of different user-defined target points in space by switching the light sources on and off with appropriate order and timing. The target points are generated independently during run time and the controller of the light sources acts according to the position of the current target point and the current plant tip position. Controllers are evolved based on a model of tip-motion that we obtain from processing the data collected by image sampling from a preliminary experiment setup using real plants and a trivial open-loop controller. The position of the plant tip and the current status of the light sources are collected while the light sources are alternated in a regular pattern controlled by the trivial non-reactive controller. From the collected set of plant tip positions we build the simple model of the combined

[1] This paper is an extended version of [24]. The previous paper is extended by: 1) extending the controller such that it can adaptively direct the plant to reach unforeseen and randomly positioned targets instead of fixed predetermined targets, 2) changing the plant model in order to achieve faster, more memory-efficient, and more accurate results, and 3) improving the image processing method for tip-detection.

growth and motion behavior of a bean plant's tip in response to the light sources. The model is then used to evolve a closed-loop controller for the light sources for directing the bean's tip such that the tip reaches any randomly generated target that is in the reach of the plant over the time of the experiment. We use these evolved controllers in a real experimental setup, and successfully control the real plant's tip to reach arbitrary targets. The results show that the reality gap in this setup is crossed with our applied approach.

Having crossed the reality gap for this task, we discuss future work based on this approach. We discuss extending our image processing method from a single point description of the plant tip to a 10-point description of the full plant stem geometry. This could allow the *tip-motion model* to be extended to a full stem-dynamics model. In the future, we will combine this stem model with cameras and image sampling in two axes. This could allow us to evolve robot controllers for more complicated tasks, such as 3D target patterns or obstacle avoidance by the growing plants.

3 Methods

Our evolutionary robotics approach to the task of steering plant growth and motion follows the methods described in this section. First, the bio-hybrid system setup shown in Fig. 1 is used in both preliminary data-gathering experiments and reality gap experiments. The preliminary experiments record the growth patterns of plants exposed to light sources from trivial manually-defined controllers, and a sampling method interprets the images into a raw data set of plant tip positions. Then, this data is used to build a purpose-specific model of tip-motion that enables simulation of tip trajectories in our setup under any given light source sequence. Finally, controllers are evolved in simulation using the *tip-motion model*, for the task of steering a plant tip to reach sequences of arbitrary user-defined targets. The method used for generation of user-defined targets is described, along with the two types of fitness functions used for evolution.

3.1 Bio-hybrid setup

The bio-hybrid system contains one biological plant that has a simple symbiotic relationship with centrally controlled robotic elements. The robot influences the plant by triggering directional light sources in its environment, and the plant influences the robot through on-board image sampling that detects plant dynamics.

As in the previous work [24], the plant in this setup is the common bean plant (*Phaseolus vulgaris L. var. nanus cf. Saxa*, a bush bean[2]), germinated in commercial soil intended for growing vegetables[3]. The bean is planted in a 1.5 liter pot, with a top diameter of 15 cm and soil level of 12 cm from the base.

[2] https://shop.nebelung.de/gemuesesamen/bohnen/buschbohnen-saxa.html
[3] FloraSelf Gemüse- und Tomatenerde ohne Torf (Floragard Vertriebs-GmbH)

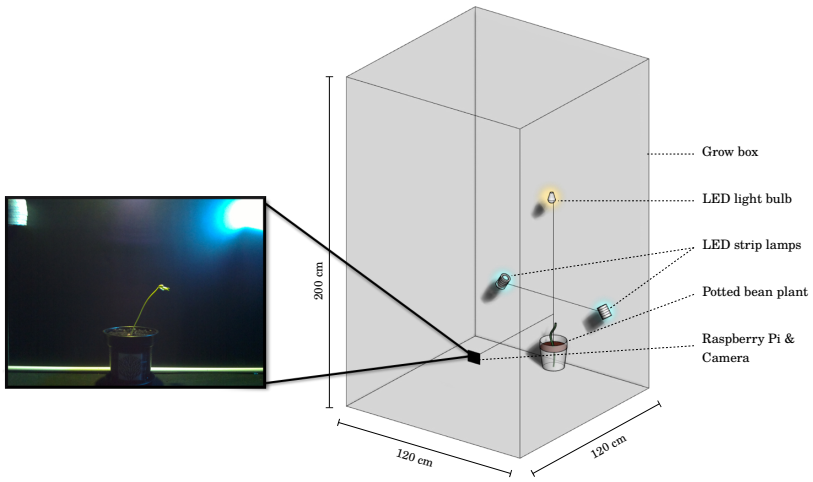


Figure 1: Bio-hybrid system setup. The setup is housed in a grow box, with the plant near the back wall and the camera near the front wall, facing the plant. A pair of light sources that provide stimuli are mounted above the plant on either side, right and left, and a flash for the camera hangs directly above the plant.

The robotic portion of the system is subject to a centralized controller, and consists of the following elements: two NeoPixel LED strip lamps[4] for providing stimuli, a camera module[5] to record the growth and motion of the plant by capturing photographs, an LED light-bulb[6] which provides flash for capturing photographs, and a Raspberry Pi[7] as a central control and processing unit. A NeoPixel strip contains 144 individually controllable integrated RGB LEDs[8], with peak-emission at wavelengths $\lambda_{\max}$ 630, 530, and 475 nm respectively. Each LED consumes 0.24 W when emitting white light at full power, giving 18 lumens. As we only trigger the two NeoPixel strips individually (never simultaneously), we can expect a total power consumption of up to approximately 35 W.

Various scripts were developed to detect the plant tip, run the evolved artificial neural networks (ANNs), control the light sources, and regularly upload the captured photos and log files to a Network-attached storage device (NAS). The scripts run on the Raspberry Pi as background processes managed by systemd[9]. The ZeroMQ[10] library is used to allow the necessary communication among

[4] Adafruit NeoPixel RGB LED strips (https://www.adafruit.com/products/1506)

[5] Raspberry Pi camera module (https://www.raspberrypi.org/products/camera-module/)

[6] Philips LED bulb 8718696490860 (http://www.philips.co.uk/c-p/8718696490860/)

[7] Raspberry Pi 3 Model B (https://www.raspberrypi.org/products/raspberry-pi-3-model-b/)

[8] WS2812 integrated light source (https://cdn-shop.adafruit.com/datasheets/WS2812.pdf)

[9] Systemd is a system and service manager for Linux operating systems

[10] http://zeromq.org/

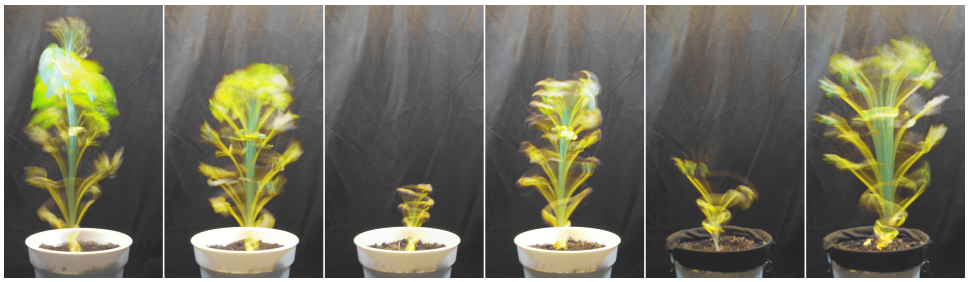


Figure 2: Compiled time-lapse photographs of experiments. From left to right, experiments 1, 2, 3, 4, 5, and 6. Photographs originally published in prior work [24].

these processes (e.g., sending a flash-light request before capturing a photo).

The bio-hybrid system is contained in a commercial grow box of dimensions $120 \times 120 \times 200$ cm in width, depth and height (see Fig. 1). The standard grow box is modified, with the interior walls clad in matte black foam board to reduce light reflections and provide a smooth, consistent background that is in high contrast to the plant. The potted bean plant is positioned at the center of the back of the grow box, such that the plant's root-shoot transition (i.e., the location where the germinated bean protrudes from the soil) is at about 12 cm height, 8 cm from the back wall, and 60 cm from each side wall.

The camera module is positioned near the front wall of the grow box, at a height of 32 cm. It faces the plant, with the focal plane parallel to the back wall, as diagrammed in Fig. 1. This placement positions the camera approximately 74 cm from the plant and 82 cm from the back wall. The flash-light used when recording photographs hangs at a height of 80 cm from the base of the potted plant, and is centered over the pot. The two NeoPixel strips are coiled in cylindrical shapes to form two LED strip lamps, and affixed to the back wall of the grow box at a height 30 cm above the root-shoot transition and 35 cm to either side (see Fig. 1).

3.2 Model setup

The simple data-driven model of plant growth and motion, through which we project plant tip trajectories in simulation and inform the evolution of our bio-hybrid controllers, is constructed according to the method described in this section. Our purpose-specific *tip-motion model* is based on time-lapse photographs of preliminary plant growth experiments. These images are sampled to detect the xy-coordinates of the plant tip, building a data set of timestamped tip positions under one of two triggered light sources. We use this data to calculate the time-normalized tip-motion vectors of a subsequent time step according to the current bio-hybrid system configuration. These motion vectors are used, in conjunction with statistical functions incorporating the stochasticity appropriate

to plant growth, to construct our model of plant tip dynamics.

3.2.1 Preliminary plant experiments

Our *tip-motion model* builds on our prior work [24] by using the same group of photographed preliminary experiments to create a data set of plant growth and motion. These preliminary experiments (described here according to [24]) use a predetermined open-loop controller which simplistically alternates the two light sources over regular time intervals. There are six repetitions of the experiment. In each repetition, the light source alternates every six hours, and the experiment is photographed at five minute intervals. The plants' tip position and stem geometry over time (see compiled images for each experiment in Fig. 2) show that their dynamics are substantially influenced by both growth and motion[11]. Despite the consistent setup conditions in each repetition, the compiled images show variety in the plants' patterns of horizontal motion and overall height. In plant science, variance in experiments is an expected phenomenon and is dealt with by conducting large quantities of repetitions. In our work, we instead take an engineering approach, and deal with plant variance by testing whether our method succeeds in controlling a plant despite certain unpredictability in behavior.

3.2.2 Tip detection

The images described above provide time-stamped raw data documenting plant responses to the predetermined open-loop controllers. The images are sampled[12] at 1/8 resolution and further processed by using the following method for detecting the plant tips. The tip-detection method works based on two sets of photographs. The first set contains the background photographs showing the setup without a plant. The second set contains the photographs from the preliminary experiments.

For the background photographs, many examples of each controller state are included in the set, as there can be slight variations in lighting conditions cast on the background. The green RGB channel value at each pixel position (i, j) is isolated and remapped onto the domain $(0.0,\ 1.0)$, resulting in a matrix $S_k = \{s_{i,j,k}\}$ for all images k. To represent the range of green values possible at every pixel in the background, matrices $A = \{a_{i,j}\}$ and $B = \{b_{i,j}\}$ are constructed with the same shape as S_n matrices. Every entry of A and B are the minimum and maximum values of the corresponding entries in all the S matrices:

$$\forall a_{i,j}, b_{i,j} : \quad a_{i,j} = \min_k(s_{i,j,k}), \quad b_{i,j} = \max_k(s_{i,j,k}). \tag{1}$$

[11] Find a video at: https://youtu.be/r4PknIwgTyo

[12] The tip-detection was implemented in two separate programming platforms. First, we implemented the method in Iron-Python and native libraries of the VPL Grasshopper pertaining to image sampling for processing the data in simulation. Then, we implemented the method in standard Python, utilizing the OpenCV library. This was then used for the reality gap experiments detailed in Sec. 4.2.

After constructing these matrices representing the background setup, the photographs from the preliminary experiments containing plants are processed. The green channel value is again isolated for each pixel (i, j) and remapped to the domain (0.0, 1.0), then is saved into the matrices $G_k = \{g_{i,j,k}\}$ for every image k of the preliminary experiment. The G_k matrices are then compared against the range matrices A and B from the background setup in order to detect pixels containing plant material. A pixel (i, j) inside a certain cropped window is identified as containing plant material if its value is external to the corresponding range, with respect to threshold $\theta = 0.2$,

$$P_k = \{(i,j) \mid \quad \forall i,j: \quad g_{i,j,k} < (a_{i,j} - \theta) \vee g_{i,j,k} > (b_{i,j} + \theta)\}. \tag{2}$$

Each identified plant pixel is extracted to set P, and their $\mathbf{x}_p = (x_p, y_p)$ coordinate positions are used to identify two possible locations of the plant's tip. In order to locate the tip $\mathbf{x}_t$, plant pixels are compared to the globally defined anchor $\mathbf{x}_a = (x_a, y_a)$, representing the position where the plant stem emerges from the soil. We identify two possible tip positions (corner point $\mathbf{x}_c = (x_c, y_c)$ and high point $\mathbf{x}_h = (x_h, y_h)$),

$$\mathbf{x}_c = \underset{\mathbf{x}_p \in P}{\arg\max} \, |x_a - x_p| + |y_a - y_p|, \tag{3}$$

$$\mathbf{x}_h = \underset{\mathbf{x}_p \in P}{\arg\max} \, |y_a - y_p|. \tag{4}$$

From the two points $\mathbf{x}_c$ and $\mathbf{x}_h$, the one closer in Euclidean distance to the previously detected tip $\mathbf{x}_{t-1}$ is selected as the current tip $\mathbf{x}_t$,

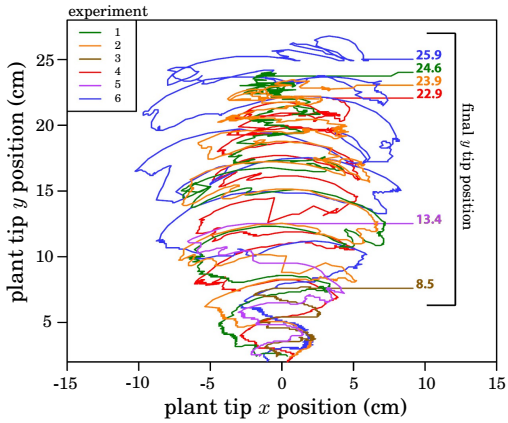
$$\mathbf{x}_t = \begin{cases} \mathbf{x}_h, & \text{if } \|\mathbf{x}_{t-1} - \mathbf{x}_h\| < \|\mathbf{x}_{t-1} - \mathbf{x}_c\| \\ \mathbf{x}_c, & \text{else} \end{cases}. \tag{5}$$

The plant tip positions $\mathbf{x}_t$ are not based on physical measurement units but on pixels. We make use of this position data in our purpose-specific model, described next.

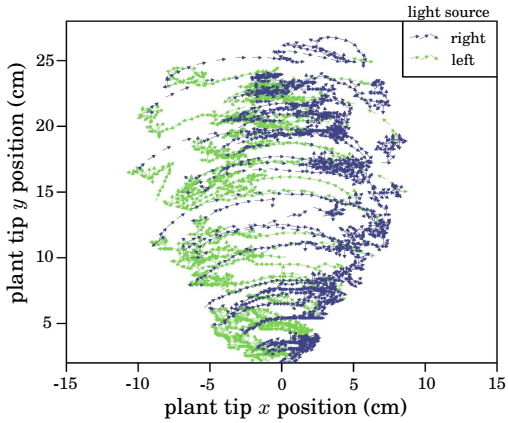
The trajectories of tip positions are shown in Fig. 3(a), with the six experiments indicated by color. Fig. 3(b) shows the 2D vector direction of tip-motion at each time step in the experiments, categorized according to active light source (right light as blue vectors and left light as green vectors).

3.2.3 Tip-motion model

We use the data from our preliminary experiments and image processing to create a simple *tip-motion model*. The model used in this work is an extension of our prior work [24]. In the previous approach, we defined the system configuration at time step t as $(\mathbf{x}, L, C)_t$, where $\mathbf{x}_t$ is the plant's tip position, L_t is the plant's length, and C_t is the lighting condition (Boolean value indicating whether the right light is on). The model simply predicted the next plant's tip position $\hat{\mathbf{x}}_{t+1}$, given the current system configuration $(\mathbf{x}, L, C)_t$. This was achieved by windowing into the recorded data during run time of the simulations



(a) Trajectories of plant tip position in the six preliminary experiments, with experiment number indicated by color (refer to Fig.2) and height in the final time step in each experiment indicated on the right-hand side.



(b) Vector direction of Δx and Δy of plant tip trajectories at each time step, in all six preliminary experiments, with color indicating right or left light source.

Figure 3: Plant tip trajectories from preliminary experiments (only includes real data; does not include mirrored). Figures originally published in prior work [24].

and, from this window, calculating average position-changes that are corrected for consistent plant-lengths using standard trigonometry [24].

In this work, in order to greatly improve the speed of the plant trajectory simulations, our extension relies on an array of precomputed model statistics for each pixel location where a tip could be detected. During run time, the position of the detected tip is used to index into this array, retrieving the values needed for simulation. Because the model statistics are precalculated and therefore do not require short computing times, we use more sophisticated methods of aggregating the data. This allows the plant length parameter L_t to be removed from the model, simplifying the full description of the system configuration to $(\mathbf{x}, C)_t$.

In summary, we first aggregate the tip detection data into a four-column 2D array storing normalized and mirrored x, y, Δx, and Δy values from all six preliminary experiments. We then window into this new data to calculate the averages and standard deviations of the Δx and Δy values contained in the area around each pixel. We finally create two 3D arrays (one for the left light source, one for the right light source) that both have the same xy measurements of the experiment images (where the rows and columns correspond to those of an image), and have four layers in the third dimension, containing the mean Δx, mean Δy, standard deviation of Δx, and standard deviation of Δy. These two arrays comprise the *tip-motion model* and are used to compute the next tip position during simulation of growth and motion dynamics. A schematic overview of the process is shown in Fig. 4. Following is a more detailed description of the procedures.

Aggregating the tip data Each tip position change from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$ occurs under the influence of Boolean light condition C_{t+1}. For each active light source, we first construct sets L (left light) and R (right light) containing tip positions $\mathbf{x}_t = (x_t, y_t)$,

$$L = \{\mathbf{x}_t | \quad C_{t+1} = 0\}, \quad R = \{\mathbf{x}_t | \quad C_{t+1} = 1\}. \tag{6}$$

We assume our setup to be symmetric and the plant to lack directional bias, therefore we mirror the data by transforming all $\Delta\mathbf{x}_t$ values associated with the left light source ($C_{t+1} = 0$) such that they provide additional data for the right light source. We calculate the tip position changes for all time steps t (excepting the final time step for each experiment) and time-normalize it by $\Delta\mathbf{x}_t = (\mathbf{x}_{t+1} - \mathbf{x}_t)/\Delta u_t$, where $\Delta u_t = u_{t+1} - u_t$ is the duration in minutes between every time step t. These vectors, along with positional data, are mirrored according to the width w of the processed image, and the data recorded in both light conditions is pooled together. In this way, all tip-motion data is aggregated into a single data set cast relative to the right light source. It is later reinterpreted to apply to both light sources, during the step of building the *tip-motion model*. Finally, the time-normalized and light-mirrored $x_i, y_i, \Delta x_i, \Delta y_i$

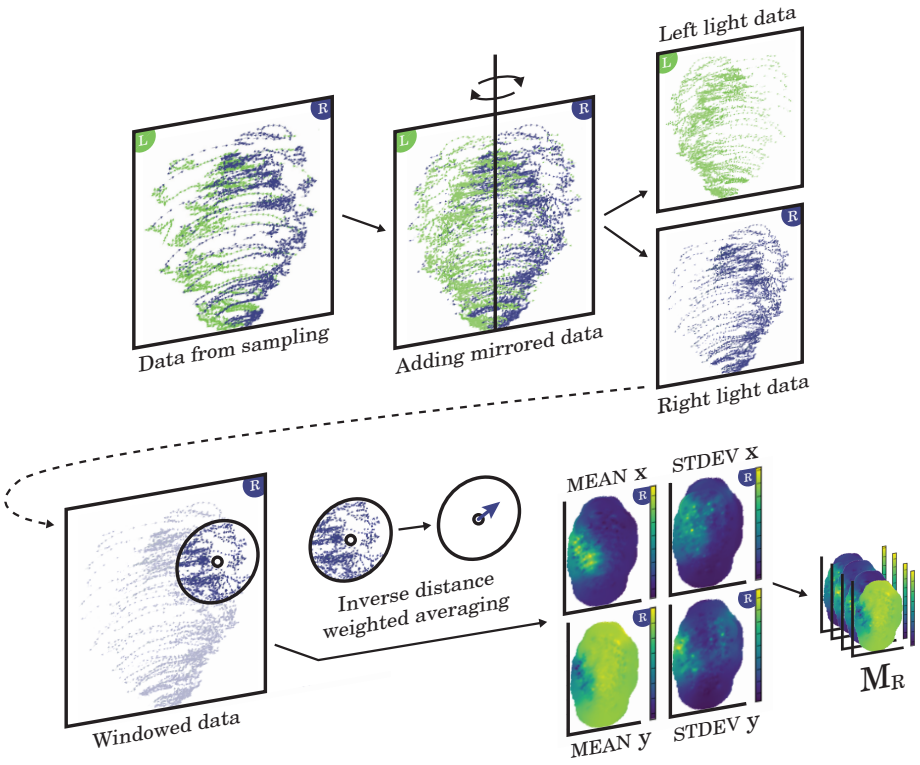
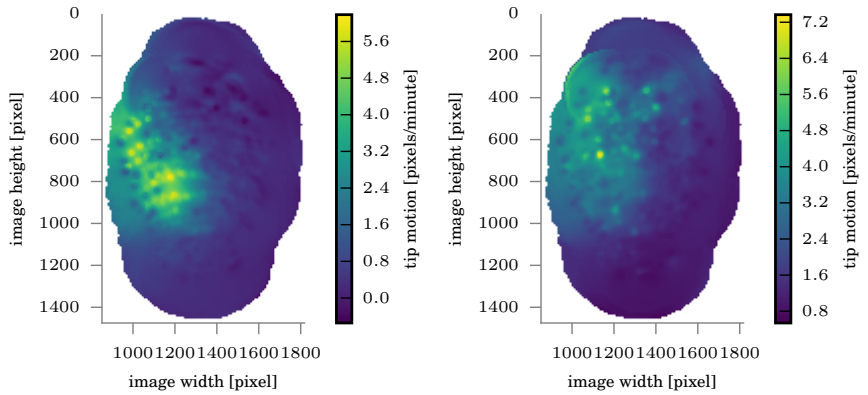


Figure 4: Schematic for building M_R, one of the two arrays comprising our tip-motion model. In the top part of the schematic, once vector data is extracted through sampling, the data set is doubled by mirroring around the central axis, with the data of the right and left light sources separated. In the bottom part of the schematic, the model array M_R for the right light source is built. The data for the right light is interpolated and smoothened through inverse distance weighted averaging, saving the mean and standard deviation of each location for both the x and y directions.



(a) $\mu(\Delta x)$ (b) $\sigma(\Delta x)$

(c) $\mu(\Delta y)$ (d) $\sigma(\Delta y)$

Figure 5: The four layers of the *tip-motion model* array M_R, describing tip-motion when the right light source is triggered. The colormaps indicate the distribution of the IDW means μ and standard deviations σ of the Δx and Δy values at each xy position. The data is mirrored, time-normalized, smoothened, interpolated, and extrapolated according to the method described in Section 3.2.3. White patches indicate absence of data. Axes are given in image pixel coordinates. 37.5 pixels correspond to 1 cm in the camera focal plane occurring at the plant. (a) and (b) show the statistical description of the tip-motion on the x-axis, while (c) and (d) describe motion on the y-axis. Because the y-axis origin is placed at the top of the image (above the plant), negative values in colormap (c) indicate the plant tip's upward motion. In (a), positive values indicate motion towards the right.

data for all experiments is pooled into a 4-column 2D array D_R, where

$$D_R = \{(x_i, y_i, \Delta x_i, \Delta y_i) | (x_i, y_i) \in R\} \cup \{(w - x_i, y_i, -\Delta x_i, \Delta y_i) | (w - x_i, y_i) \in L\}. \tag{7}$$

Building the tip-motion model Our purpose-specific *tip-motion model* consists of two parts, each part for one of the two light conditions (left light source or right light source). The two parts are each structured as a 3D array, matching the x-axis and y-axis pixel count of the sampled images in two dimensions, and layering four sets of values in the third dimension. The first array of the model, for the right light source, is calculated from the $(x_i, y_i, \Delta x_i, \Delta y_i) \in D_R$ described above. The second array is later calculated from the first. We utilize inverse distance weighting (IDW) [19] in calculating the first array, to interpolate and smoothen the data. The four values calculated are the IDW averages of x-axis and y-axis tip-motion vectors, along with their respective standard deviations.

In contrast to the method in prior work [24], which windows into the aggregated tip-motion data using small rectangles (37.5 pixels × 75 pixels in width and height), here we window into the data using large circles of radius $r = 200$ pixels. Also, while the prior work [24] models tip-motion differently along the x-axis and y-axis, here we model them jointly, taking the IDW average and standard deviation of the windowed data for both axes.

To calculate the model for the right light source, we first window into the aggregated data D_R and construct array $W_{\mathbf{x}_p}$, which has dimensions matching the sampled images. For each position $\mathbf{x}_p = (x_p, y_p)$, the data points of D_R where $\mathbf{x_i} = (x_i, y_i)$ falls into window radius $r = 200$ is collected as

$$W_{\mathbf{x}_p} = \{(x_i, y_i, \Delta x_i, \Delta y_i) \in D_R \quad | \quad \|\mathbf{x}_i - \mathbf{x}_p\| \leq r\}. \tag{8}$$

This data is used to compute the IDW averages and standard deviations of Δx_i and Δy_i at each $\mathbf{x}_p$. A weight w_i is assigned, based on a simple IDW function with exponent two, to every data point $\mathbf{x_i} = (x_i, y_i)$ where $(x_i, y_i, \Delta x_i, \Delta y_i) \in W_{\mathbf{x}_p}$, such that

$$w_i = \begin{cases} 1/\|\mathbf{x}_p - \mathbf{x}_i\|^2 & \text{if } \|\mathbf{x}_p - \mathbf{x}_i\| \geq 12 \\ 1/12^2 & \text{else} \end{cases}. \tag{9}$$

If the distance is under 12 pixels in this operation, we fix it at 12 to avoid overweighting and division by zero, as our tip detection occurs at 1/8 resolution. The IDW averages for each position $\mathbf{x}_p = (x_p, y_p)$ are computed according to $(x_i, y_i, \Delta x_i, \Delta y_i) \in W_{\mathbf{x}_p}$ as

$$\mu(\Delta x_p) = \frac{1}{|W_{\mathbf{x}_p}|} \times \sum_{i=1}^{|W_{\mathbf{x}_p}|} w_i \times \Delta x_i, \tag{10}$$

$$\mu(\Delta y_p) = \frac{1}{|W_{\mathbf{x}_p}|} \sum_{i=1}^{|W_{\mathbf{x}_p}|} w_i \times \Delta y_i, \tag{11}$$

and the IDW standard deviations are computed as

$$\sigma^2(\Delta x_p) = \frac{1}{|W_{\mathbf{x}_p}|} \times \sum_{i=1}^{|W_{\mathbf{x}_p}|} (w_i \times (\Delta x_i - \mu(\Delta x_p))^2), \tag{12}$$

$$\sigma^2(\Delta y_p) = \frac{1}{|W_{\mathbf{x}_p}|} \times \sum_{i=1}^{|W_{\mathbf{x}_p}|} (w_i \times (\Delta y_i - \mu(\Delta y_p))^2), \tag{13}$$

unless $|\mathbf{x}_i \in \mathbf{x}_p| < 30$ (i.e., there are fewer than 30 data points in the circular window around $\mathbf{x}_p$). In such cases, we discard the entries and cast that $\mathbf{x}_p$ position as empty, because there is not enough tip-motion data to calculate the model values.

A new 3D array M_R is constructed with two dimensions matching the sampled images, and with the values $\mu(\Delta x_p)$, $\mu(\Delta y_p)$, $\sigma(\Delta x_p)$, and $\sigma(\Delta y_p)$ for each $\mathbf{x}_p$ assigned to four layers in the 3rd dimension. This array M_R contains all the information of the model, but is cast under the right light source, as described above. To calculate the second array M_L of the model, we mirror M_R by flipping the four layers over the x-axis and by multiplying the $\mu(\Delta x_p)$ values by -1. This yields the final *tip-motion model* arrays, M_L and M_R.

The values contained in the *tip-motion model* array M_R (i.e., for the right light source) can be seen in Fig. 5. Fig. 5(a) and 5(c) represent the mean tip-motion vectors $\mu(\Delta x)$ and $\mu(\Delta y)$, Fig. 5(b) and Fig. 5(d) show the standard deviations $\sigma(\Delta x)$ and $\sigma(\Delta y)$. The standard deviation values are usually larger than their respective means, showing the large variance in the dataset. In Fig. 5(c), there is a general trend of growth upwards (negative values on the colorscale indicate upward motion, as $y = 0$ is the uppermost pixel) that is especially pronounced on the far side from the light source. Towards the top-right corner of the data is a concentration of motion downwards. Both of these features in the data reveal the plant stem bending towards the light source. Fig. 5(a) indicates that, when the tip is farther from the light source, it typically moves more quickly towards it. The features visible in these colormaps of M_R would be mirrored horizontally in colormaps of M_L. These *tip-motion model* arrays, M_L and M_R, allow for significantly improved and sped up plant simulations, as compared to prior work [24].

3.2.4 Simulation of tip behavior

We begin simulations of plant tip trajectories at a manually fixed origin (x_0, y_0). The origin x_0 is placed at the horizontal center of the image ($x_0 = 1296$ pixels), and y_0 is placed at the lowest point in the photographs that occurs higher than the edge of the pot in all experiments ($y_0 = 1250$ pixels, where $y = 0$ occurs at the uppermost row of the image, above the plant). We use this (x_0, y_0) origin as our first simulated plant tip position $\mathbf{x}_t = (x_t, y_t)$ in each simulation run. At each time step of the simulation, we index into the appropriate array M_L or M_R depending on the light condition output by the controller (see Section 3.3 for

controller details), and according to the current simulated tip position $\mathbf{x}_t$ retrieve the stored $\mu(\Delta x_i)$, $\mu(\Delta y_i)$, $\sigma(\Delta x_i)$, and $\sigma(\Delta y_i)$. We compute the next tip position $\mathbf{x}_{t+1}$ by drawing from random distributions according to the retrieved values, such that

$$x_{t+1} = x_t + \mathcal{N}(\mu(\Delta x_i), \sigma(\Delta x_i)), \tag{14}$$

$$y_{t+1} = y_t + \mathcal{N}(\mu(\Delta y_i), \sigma(\Delta y_i)). \tag{15}$$

3.3 Controller setup

Our controller is an ANN with four inputs. The inputs are the xy coordinates of the current plant tip position and the current target position at each time step. These time steps correspond with five minute discrete intervals in reality. The network has only one output, which is a binary decision indicating whether the left or right light source will be triggered in the next time step. When the controller runs in reality gap experiments, an image of the plant is captured and processed at each time step, acquiring the current position of the real plant tip. When the controller runs in simulation, the *tip-motion model* is used to project the current tip position from the previous.

3.3.1 Definition of the task

In our previous work [24], the controller performed the task of steering the plant tip serially towards three specific, unchanging targets in xy space. Here, we extend the work by generalizing the controller, such that the next arbitrary xy target is provided as an input. This enhancement allows the controller to direct a wide range of growth patterns for various applications (e.g., plant shaping, braiding, see Sec. 6). A new target input to the controller is triggered once the current target $\mathbf{x}_i^*$ is reached by the plant tip $\mathbf{x}_t$, such that

$$|x_t - x_i^*| \leq 12 \vee |y_t - y_i^*| \leq 12 \vee |y_i^* - y_t| \geq 40. \tag{16}$$

If the tip's horizontal or vertical distance to the target is smaller than 12 pixels, we count the target as reached. Else, when the tip is 40 pixels above the target (to account for the possible downward bending of the plant stem), we switch to the next target anyway. Using this approach of analyzing target proximity separately along the two axes eliminates the speed problem posed by calculation of Euclidean distance, at a time-critical step. When triggered, a new target is generated according to the method described below.

3.3.2 Target generation

The task of the evolved controllers is to steer the plant tip to arbitrary targets that can change at run time. Targets would be user-defined in an application, hence, for benchmarking we automatically generate a varying number of arbitrary targets for every individual controller and evaluate its behavior according to those target sets. Besides the objective of targets being reachable by the

plant in principle, we have two competing priorities in our generation method. On the one hand, because we evaluate each controller with multiple simulation runs (each with varying numbers of targets), the target sets should have comparable statistical properties, such that the evaluation of controller performance is not biased by differences in targets. On the other hand, the generated targets should be diverse enough to ensure that the evolved controllers will be general. We approach these objectives by using a method similar to our simulation of tip-motion, but instead of drawing random values from a normal distribution parameterized from the *tip-motion model* array, we select values corresponding to a given cumulative probability at each pixel. As such, we employ a data-driven probabilistic target generation method, described below. A new target is generated once a previous target is reached. The position of the new target is defined based on a given probability for reachability of the point and the desired relative distance along the trajectory from the tip position to the edge of data in the array.

The function for target generation (see Algorithm 1) receives the position of the plant tip at the time of target generation (tip_x, tip_y), as well as the *tip-motion model* arrays M_L for left light source and M_R for right light source. The function also uses a set of parameters that can be changed by the user according to the desired level of reachability of the targets, p_{reach}, the level of desire for switching direction when moving from one target to the next, p_{switch}, and the relative distance d_{traverse} along the projected trajectory where we fix the target, the maximum y position $y_{\max}$ of any target in the system, and lastly, the direction r_{last} in which the prior target was generated. In the implementation used here, we draw uniform random values for p_{reach} from the half-open interval $[0.58,\ 0.72)$ and for p_{traverse} from $[0.5,\ 0.7)$ every time a new target is generated. The value of p_{switch} is fixed at 0.75.

In order to generate a trajectory in a certain direction, we use the $\mu(\Delta y_i)$ and $\sigma(\Delta y_i)$ from the *tip-motion model* array of the respective light source C (left light, $C = 0$, right light, $C = 1$). Based on that, we calculate a trajectory of positions that are reachable with the desired probability p_{reach}. The set of positions are traversed in order to choose a target with the desired distance from the previous one based on d_{traverse}. Starting from the last target, we iteratively generate a set of potential target positions $\mathbf{x}_p$ where each point is adjacent to the previous one, such that

$$x_{p+1} = \begin{cases} x_p - 1, \text{ if } C = 0 \\ x_p + 1, \text{ else} \end{cases}, \tag{17}$$

$$y_{p+1} = y_p + \mu(\Delta y_i) - z \times \sigma(\Delta y_i). \tag{18}$$

where z is the multiplier for the standard deviation $\sigma(\Delta y_i)$ associated to position (x_p, y_p). The probit[13] function Φ^{-1} is used to convert the desired cumulative probability p_{reach} into the corresponding value $z = \Phi^{-1}(p_{\text{reach}})$ from a

[13]The Scipy Python library is used to apply the probit function (the quantile function associated with the standard normal distribution. It is the inverse of the cumulative distribution function (CDF)).

standard normally distributed variable, that we can then use to get the according value for any normal distribution (by multiplying with its σ).

The resulting list of pixel coordinates is defined as the trajectory from which a target will be selected (cast as border trajectory b in Algorithm 2). The target is chosen by traversing this series of pixels to reach the distance d_{traverse}, which gives the relative distance from the previous target (see Algorithms 1 and 2 for further details).

3.3.3 Evolutionary approach

We evolve ANN controllers using the portable Python library MultiNEAT [5], which is based on NeuroEvolution of Augmenting Topologies (NEAT) [20]. NEAT is an efficient evolutionary algorithm that begins with a random population of ANNs with minimal structure (i.e., no hidden layers), then applies complexifying methods to modify the weights and the network structure. We use here the set of NEAT parameters in Table 1, based on the parameters that have shown successful performance in our previous work [24].

At each time step t, the xy coordinates of the current plant tip position $\mathbf{x}_t = (x_t, y_t)$ and current target position $\mathbf{x}_i^* = (x_i^*, y_i^*)$ are input to the ANN. The ANN then outputs the binary light source state C_t, determining whether the left light source ($C_t = 0$) or the right light source ($C_t = 1$) is activated. The current system configuration $(\mathbf{x}, C)_t$ influences the plant's behavior (growth and motion) during that time step. Therefore, in the case of simulation, the *tip-motion model* is used to project the next tip position $\mathbf{x}_{t+1}$ from the current system configuration. In reality gap experiments, an image of the plant is captured after the time step is complete, and the processing method described in Section 3.2.2 is used to detect the plant's new tip position. This process is repeated at every time step until the tip's y_t value is equivalent to roughly 20 cm in height. This allows the plant enough growth space to reach between two and six arbitrary targets generated by the method described above. It also allows the reality gap experiments to be performed in a relatively short period of time (approximately 72 hours each), such that the overhead is manageable for an engineering task. In order to evolve ANN controllers to steer the plant tip to generalized targets, we define two alternative fitness functions, F_1 (a behavioral fitness function) and F_2 (an aggregate fitness function), according to the classification in [18].

First, we define the behavioral fitness function F_1. Since the motion control acts mainly along the x-axis while the change along the y-axis is mostly due to growth, we measure the distances traversed towards every new target from the previous target along the x-axis. For that, for every target $i \in \{1, 2, ..., n\}$ we define the instant rewards $r(t)$, based on the number N of total targets and the distance Δx_t traversed in every time step t:

$$\Delta x_t = x_t - x_{t-1}, \quad t \in \left\{ t \mid y_{i-1}^* \leq y_t < y_i^* \right\}, \tag{19}$$

ALGORITHM 1: Target Generation

Data: tip position (tip_x, tip_y), *tip-motion model* arrays M_L, M_R, given probability for reaching the target p_{reach}, given probability to switch direction of target generation (left or right of tip) p_{switch}, the given desired relative distance d_{traverse} between the old and new target positions, the Boolean r_{last} that is true when the previous target was generated towards the right, the maximum height for target $y_{\max}$

Result: target position $(targ_x, targ_y)$, Boolean c_r (true if the new target was generated towards the right)

$r \leftarrow$ uniform random number in $[0, 1)$;
$b \leftarrow$ empty list of border pixel coordinates;

if $r < p_{\text{switch}}$; `/* i.e., we switch direction */`
then
  if *not* r_{last}; `/* i.e., last target was generated towards the left */`
  then
    $c_r \leftarrow True$;
    $M_C \leftarrow M_R$; `/* pick the array for right light */`
  else
    $c_r \leftarrow False$;
    $M_C \leftarrow M_L$;
  end
else
  if r_{last} **then**
    $c_r \leftarrow True$;
    $M_C \leftarrow M_R$;
  else
    $c_r \leftarrow False$;
    $M_C \leftarrow M_L$;
  end
end

$b \leftarrow getTrajectory(tip_x, tip_y, M_C, p_{\text{reach}})$; `/* call function getTrajectory() */`

$b_{\text{len}} \leftarrow$ length of b; `/* number of tuples (`b_x, b_y`) in list */`

if $b_{\text{len}} > 0$; `/* i.e., plant tip was at valid entry of model array */`
then
  $b_{\text{index}} \leftarrow floor(b_{\text{len}} \cdot d_{\text{traverse}})$; `/* index of target tuple according to` d_{traverse} `*/`
  $(targ_x, targ_y) \leftarrow b[b_{\text{index}}]$;
  if $targ_y < y_{max}$; `/* if target is higher than allowed */`
  then
    $(targ_x, targ_y) \leftarrow$ the first tuple from b that is lower than $y_{\max}$;
  end
else
  generate a random target in the central data area (with $targ_y$ lying above the tip_y and below $y_{\max}$);
end

return tuple $(targ_x, targ_y)$ and Boolean c_r

ALGORITHM 2: Function getTrajectory()

Data: tip position (tip_x, tip_y); $\mu(\Delta y_i)$ and $\sigma(\Delta y_i)$ from the *tip-motion model* array M_C for light condition C (Boolean value, equal to 1 for right light); the probability for reaching a target p_{reach}

Result: list of tuples (b_x, b_y) border trajectory b

$z \leftarrow \Phi^{-1}(p_{\text{reach}})$; `/* probit function */`
$b \leftarrow$ empty list of border pixel coordinates;

if *C is True;* `/* i.e., right light is lit */`
then
  $m \leftarrow 1$; `/* next target right */`
else
  $m \leftarrow -1$; `/* next target left */`
end

$x_p \leftarrow tip_x$;
$y_p \leftarrow tip_y$;
while *not* ($\mu(\Delta y_i) ==$ *'NaN');* `/* i.e., we have data for that location */`
do
  append the tuple (x_p, y_p) to the list b;
  $x_p \leftarrow x_p + m$;
  $y_p \leftarrow y_p + \mu(\Delta y_i) - z \cdot \sigma(\Delta y_i)$; `/* using the quantile on` σ `*/`
  $[\mu(\Delta y_i), \sigma(\Delta y_i)] \leftarrow$ aisle of M_c at row $m = round(y_p)$ and column $n = round(x_p)$;
end

return the list of tuples b constituting all points approximately reachable with probability p_{reach} under the given light condition C

Table 1: Used NEAT parameters.

Parameter	**Value**	**Parameter**	**Value**
PopulationSize	50	*CrossoverRate*	0.5
DynamicCompatibility	True	*MutateWeightsProb*	0.9
YoungAgeTreshold	15	*YoungAgeFitnessBoost*	1.0
OverallMutationRate	0.5	*WeightReplacementMax*	5.0
MinSpecies	5	*WeightMutationRate*	0.75
MaxSpecies	25	*Elitism*	0.1
SurvivalRate	0.6	*MutateAddNeuronProb*	0.04

$$r(t) = \begin{cases} \Delta x_t, & \text{if } x_t < x_i^* \\ -\Delta x_t, & \text{if } x_t > x_i^* \\ |\Delta x_t|, & \text{if } x_t = x_i^* \end{cases}, \tag{20}$$

where (x_t, y_t) is the position of the tip at time step t and (x_i^*, y_i^*) is the position of the target i. (x_0^*, y_0^*) is the starting position of the tip. The sum of the rewards for each target is defined as

$$R_i = \sum_t r(t), \quad t \in \left\{ t \mid y_{i-1}^* \leq y_t < y_i^* \right\}. \tag{21}$$

The controller is rewarded R_i when the tip transitions between the old and the new targets. If the tip starts out of this region we make a correction in the value of R_i by decreasing it by $x_i^* - x_t$ where x_t is the starting point of the tip for the corresponding target. The reason for this correction is to prevent solutions where the tip compensates for missing a target by getting extra reward for the next target, through starting its motion towards the new target at a farther distance. The behavioral fitness F_1 is then computed as follows:

$$F_1 = \frac{\sum_{i=1}^{N} R_i}{\sum_{i=1}^{N} R_i^{\max}}, \tag{22}$$

where $R_i^{\max} = |x_i^* - x_{i-1}^*|$ is the maximal reward that a controller can theoretically achieve for every target i, and N is the number of targets. During every epoch of the artificial evolution, we evaluate each controller in 15 independent plant growth simulations (with distinct generated targets) and select the minimum fitness as the controller's final fitness.

F_1 is a behavioral fitness function; namely, it has prior knowledge about possible useful behavior leading to a potential solution. It uses this knowledge to continuously reward/punish the controller according to its behavior.

To define the second fitness function F_2, we use the Euclidean distance between the current plant tip position (x_t, y_t) and the considered target i, where

$$\text{dist}_i(t) = \sqrt{(x_i^* - x_t)^2 + (y_i^* - y_t)^2}. \tag{23}$$

A controller receives a reward of $R = 1$ if it reaches the vicinity r of the target, at any time step $t \in T$ when the plant tip is positioned between the heights of the current and former target, such that

$$T_i = \left\{ t \mid y_{i-1}^* \leq y_t < y_i^* \right\}, \tag{24}$$

$$R_i = \begin{cases} 1, & \text{if } \exists t \in T_i, \text{dist}_i(t) \leq r \\ 0, & \text{else} \end{cases}. \tag{25}$$

F_2 is then defined according to the reward values, as

$$F_2 = \frac{1}{N} \sum_{i=1}^{N} R_i \ . \tag{26}$$

Similarly to F_1, for the controllers evolved using F_2, we evaluate each controller according to 15 independent target sets and plant growth simulations. However, F_2 is an aggregate fitness function, meaning that the final step of task completion is the only metric for evaluation, regardless of the preceding steps leading up to its solution. Because of the stochasticity in our simulated growth and motion, analytically good controllers can still slightly miss targets, thus ending up with a fitness of zero according to the aggregate fitness function. Using a minimum of 15 repetitive runs greatly increases this probability. By instead taking the average, we can ensure steadier evolution of the controllers.

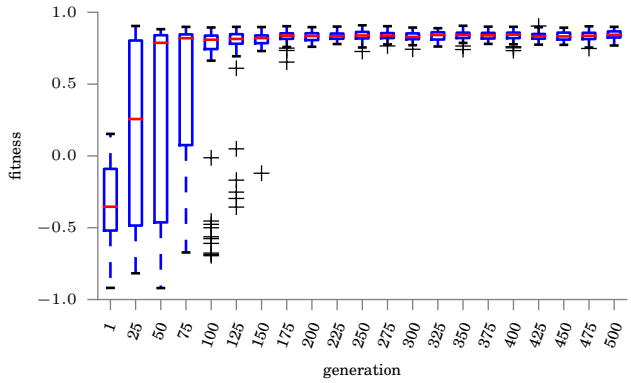
As F_2 only rewards the controller when the tip reaches a target (see Eq. 25), F_2 is less complex than the behavioral F_1. However, the lack of guidance in F_2 could lead to bootstrapping problems, slowing down the evolutionary process [8]. We do not encounter this potential slowness, by virtue of the array approach to modeling and simulating growth (see Section 3.2.3), which greatly speeds our evolutionary process. This allows us to investigate the aggregate fitness function approach seen in F_2.

4 Results

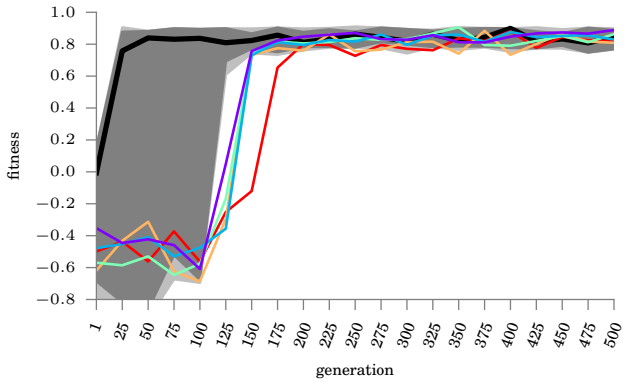
First, we report the results of evolving controllers using the *tip-motion model* (see Sec. 3.2.3) in simulation. We test our previously mentioned two fitness functions (F_1 and F_2) in two sets of 50 evolutionary runs, 500 generations each. Second, we report the performance of the evolved controllers in plant experiments (i.e., discuss the existence of reality gap).

4.1 Evolution of controllers in simulation

Compared to [24], the optimized procedure allowed, but also required, the stricter evaluation scheme of testing each individual controller on 15 simulation runs. The current model includes substantially more (and locally inhomogeneous) stochasticity, better reflecting the plants' behavior. First, we show the results for the behavioral fitness function F_1. Fig. 6 shows boxplots and functional boxplots of 50 independent runs that each contain 500 generations. We can clearly see that, unlike in the previous work [24], we can much better guarantee that the NEAT process finds a solution to this more complex problem. By generation 200, we have reached convergence, while the majority of populations evolved a solution within the first 50 generations. Counterintuitively, feeding the ANN the current target as an additional input parameter makes this task much easier, as the network can learn the correlation between the input and output parameters. Fig. 7(a) shows the worst performance of 15 plant simulations of the controller we selected to guide the plants across the reality gap (see Sec. 4.2). We chose it because it had the highest fitness of all controllers, even though we are aware that there is a lot of stochasticity involved. The strategy is straightforward: trigger the light source that leaves the target in between the light source and the plant tip. This leads to the plant being guided below the

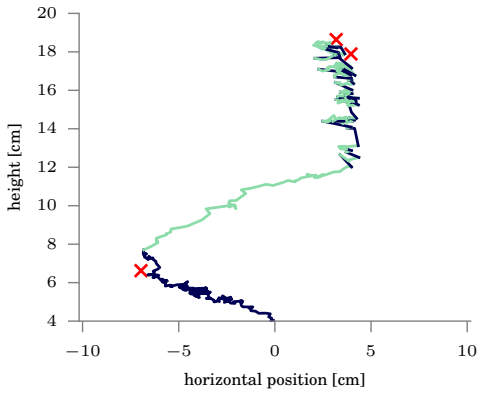


(a) Fitness of best controller per generation.

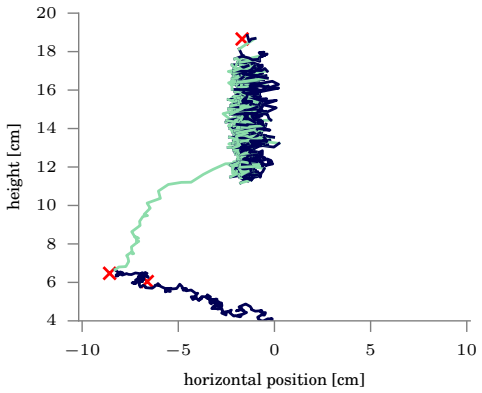


(b) Functional boxplot.

Figure 6: Evolution of controllers selected by fitness function F_1. Shown here are 50 independent runs. (a) Boxplots give the first quartile, the median, and the third quartile, while the whiskers are 1.5 times the interquartile distance added to the box. Outliers are shown spearately. (b) Because the generation-fitnesses of a single run are independent of each other, we can use functional boxplots. The figure shows the "median-function" in black, while the "quartile" and "whisker functions" color areas dark and light gray respectively. The "whisker functions" are 3 times the interquartile distance added. A function is considered an outlier and drawn separately, if it is outside the range at any single generation.



(a) Evolved controller using F_1, success = 92.3%



(b) Evolved controller using F_1, success = 91.0%

Figure 7: Trajectories of the simulated plant tip for two successful controllers using F_1

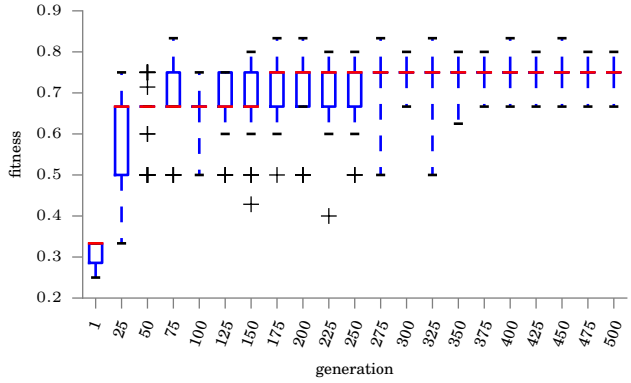
target as fast as possible, and then kept there (by alternating the lights) until it grows to reach the target. This behavior is particularly pronounced in Fig. 7(b), where the tip sometimes moved down, giving a thicket of trajectory very similar to one of our reality-gap experiments[14] (see Sec. 4.2, for more details explaining this behavior).

Second, we show the results for the aggregate fitness function F_2. The results from 50 independent runs are shown in Fig. 8. Following the median in Fig. 8(a), the fitness increases steadily and saturation is achieved after 225 generations. Notice that the median shows step-wise increase/decrease behavior which reflects the properties of an aggregate fitness function (explicit reward when a target is achieved).

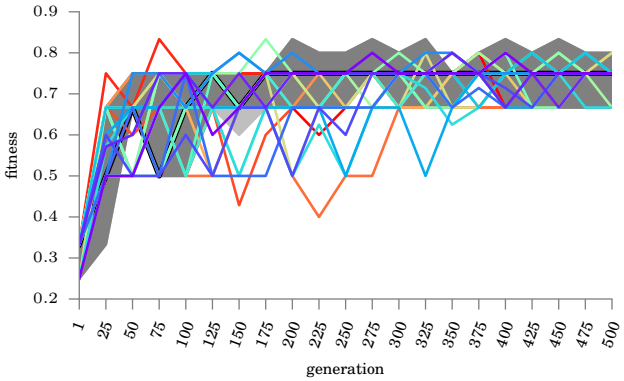
In Fig. 9(a), five targets were generated and the controller could score 85.7% success. Here, the controller could steer the plant tip towards four targets successfully, but fails to approach the last target due to stochasticity in simulated plant behavior. In Fig. 9(b), another four targets were generated and the controller successfully guided the simulated tip towards all of them, giving a score of 100%.

To further assess our results, we compare the performance of controllers evolved for each fitness function with the performance of an equivalent quantity of randomly generated controllers. We generate more than 1,250,000 (50 runs $\times$ 500 generations $\times$ population size 50, yielding the same number of evaluations as in the evolutions) random neural networks per fitness function, and evaluate them in the same way as their evolved counterparts. For each individual controller, which undergoes 15 independent simulation runs, we look at the minimal fitness for the behavioral F_1, and at the average fitness for the aggregate F_2. When evaluating with F_1, fitness can range from large negative values corresponding with the distance of growth away from the targets, to a fitness of 100% when perfectly reaching all targets. For random controllers evaluated by F_1, we observe a mean fitness of -229% with a standard deviation of 122%, with the best controller achieving a fitness of 54% (compared to 92% fitness achieved through evolution). Out of 1,327,605 random controllers, only five of the controllers (i.e., 3.8×10^{-6}%) achieved a fitness over 50%. When evaluating with F_2, the combined fitness values are discrete and dependent on the number of targets, and the overall fitness is mapped to the interval of 0% to 100%. For random controllers evaluated with F_2, the mean fitness is at 20% with a standard deviation of 8%, with the best controller achieving a fitness of 67% (compared to 100% fitness achieved through evolution). Out of 1,379,379 random controllers, only three of them (i.e., 2.2×10^{-6}%) achieved a fitness better than 50%. Thus, in both cases, the evolutionary process substantially outperforms the blind controllers.

[14] Find a video of dynamic-targets experiment #1, at: https://vimeo.com/205469308

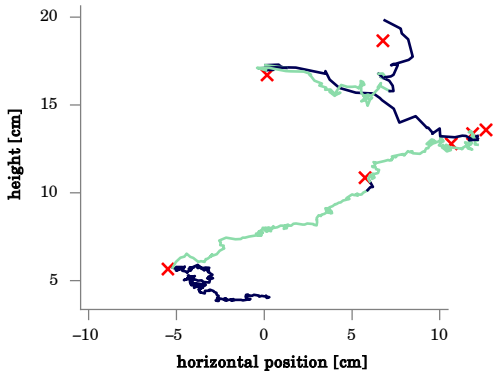


(a) Boxplot of best fitness per generation.

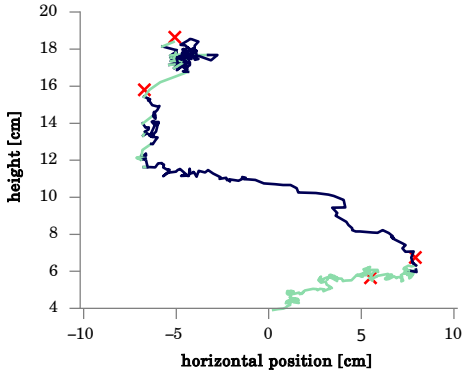


(b) Functional boxplot of best fitness per generation.

Figure 8: Performance of the evolutionary process over generations for 50 evolutionary runs.



(a) Evolved controller using F_2, success = 85.7%



(b) Evolved controller using F_2, success = 100%

Figure 9: Trajectories of the simulated plant tip for two successful controllers using F_2

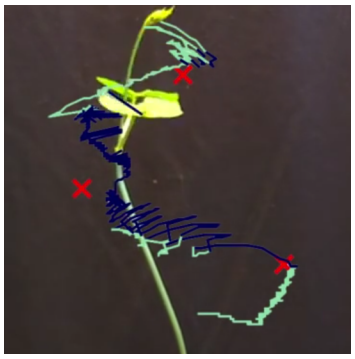
4.2 Performance of controllers in plant experiments

In a final set of plant experiments, one of our controllers (see Fig. 7(a)), that was successfully evolved in simulation, is tested in reality with actual bean plants. It is one of the controllers evolved based on the behavioral fitness function F_1. This is a typical attempt to investigate the reality gap problem [15]. In our previous work [24], we confirmed the possibility to bridge the reality gap for a task with three predefined target points ($\mathbf{x}_1^* = (3, 6)$, $\mathbf{x}_2^* = (-5, 9)$ and $\mathbf{x}_3^* = (-1, 13.5)$). Here, we test the performance of our evolved controller first in a similar scenario (fixed targets experiments). Second, we extend our reality gap investigation by including plant experiments with dynamically generated targets (dynamic-targets experiments) as described in Sec. 3.3.2. One of these plant growth experiments takes up to 72 hours, therefore, we parallelized our experiments and tested the controller concurrently in two separate bio-hybrid experiment setups.

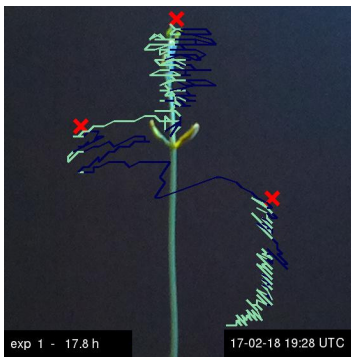
On the one hand, the fixed-targets experiment was repeated three times, scoring a fitness of 95.17% on average. Hence, we observe an average 28.7% increase in performance in comparison to our previous work. In Fig. 10, we show the trajectories of the guided plant tip from our previous work (Fig. 10(a)) side by side with the trajectories from our current work which scored a fitness of 95.17% (Fig. 10(b)). On the other hand, the dynamic-targets experiment was repeated two times, scoring a fitness of 91.25% in average. In Figs. 11(a) and (b) we show the trajectories of the guided plant tip from the two dynamic-targets experiments scoring fitness values of 92.6% and 89.9% respectively. In comparison to the experiments in simulation, we notice similar behaviors of the actual plant steered by the controller. The controller makes the tip of the plant approach every target precisely from below ($x_t = x_i^*$). Then it maintains the horizontal coordinate x_t during the plant's growth by alternating between the two light sources until the target is reached. This generates a series of back and forth movements (e.g., in Fig. 11(b) between the second and third target[15]. When the target is finally approached and switched, the controller rotates the plant appropriately to the opposite side (where required) beneath the next target.

Both the superficial observation of the plant behavior, that is, the effects of the control and the measured fitness in the plant experiments indicate that the controllers transferred successfully from simulations to reality without being changed. Hence, we have successfully bridged the reality gap in this specific setup. This is a very encouraging result because it demonstrates possible future pathways for this research. While the plant experiments are slow and expensive, our modeling approach allows to quickly evolve controllers with high transferability. Whether this feature scales up to scenarios of higher complexity needs to be shown.

[15] Find a video of fixed-targets and dynamic-targets experiments, at: https://vimeo.com/205469308

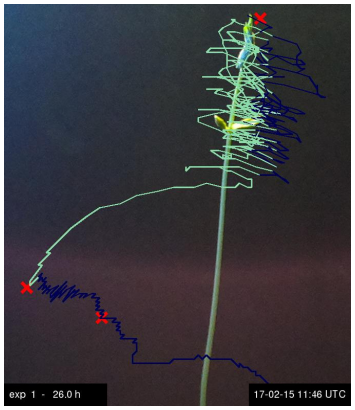


(a) Trajectories from previous work, fitness = 72.6%

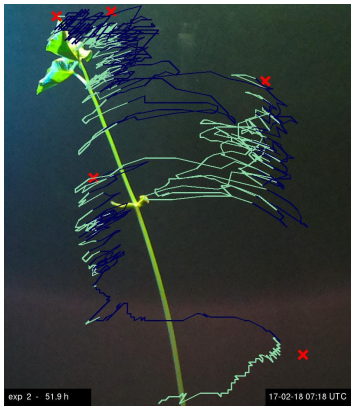


(b) Trajectories from current work, fitness = 95.17%

Figure 10: Trajectories of the detected plant tips in fixed targets experiments.



(a) Dynamic-targets experiment #1, fitness = 92.6%



(b) Dynamic-targets experiment #2, fitness = 89.9%

Figure 11: Trajectories of the detected plant tips in dynamic-targets experiments.

5 Discussion

This is a pioneering study into a new domain of using robots to control natural plants. Naturally, we started our endeavor of developing a novel methodology with a task that is arguably not overly complex. There are not many degrees of freedom in the actuator control values as the controller is limited to switch between the left and right light. Still, the task contains all important challenges that we will also face in later, more complex studies. These challenges are correct timing, the plant's reactive motion behavior, the plant's medium-term growth, and the user-defined target points that introduce considerable issues in steering the plant. For example, the stiffening process of the plant's stem requires sometimes to steer the plant tip in trajectories that overshoot the intuitive intermediate waypoints. Only the overshoot ensures later that the plant finally reaches the desired target point.

The contribution of the proposed data-driven plant model is a relevant byproduct of this work. Although there is of course a large body of literature on plant models, the large majority of these cannot be applied in this work. Most research in plant science is focused on specific details, for example, in the physiology and biochemistry of the plant. What was required here, however, was a rather holistic model of plant behavior. Developing a generic holistic model of plant growth and motion may prove to be extremely challenging or even infeasible, while a domain-specific model may not only be feasible but even simple. Our work shows that a data-driven model can be defined for a specific domain and task. Maybe this result can encourage developing more of such models, that would be required for engineering approaches as in the combination of plants and robots.

The successful crossing of the reality gap as reported here is a significant result. This success may seem surprising given the degree of complexity in the involved plant behavior compared to regular scenarios in evolutionary robotics, such as maze exploration or legged locomotion.

With the data at hand, we can only speculate about the causes. On the one hand, the relatively low degree of freedom in the control values (switching two lights) may limit the possibilities of how a controller, that performed high in the simulation, can behave in unfortunate ways in the real experiment. On the other hand, the easy crossing of the reality gap may also hint to the potentially high quality of our data-driven plant model. The development of the model came with a considerable cost in the form of preliminary plant experiments and a certain amount of theoretical work. Hence, the easy crossing was a result of an upfront investment and is probably not a good role model for similar efforts in evolutionary robotics in general. Also within this work, it is unclear how our modeling approach would scale up to more complex experimental setups and tasks. Each additional dimension (e.g., spatial dimension, third light, second plant, plant branching) would increase the complexity of the model and lead to a combinatorial explosion if the method is not improved qualitatively.

The task of controlling a single plant's growth and motion with two lights should be seen as a microscopic problem instance of what we develop and in-

vestigate in the project *flora robotica*. The vision is to use a distributed robot system to control plant growth on architectural scales. In ongoing work (unpublished), we experiment with eight robots in a 2D setup to control plant growth on a bigger scale including robot-robot interactions but limited to regularly programmed controllers (not evolved). Tasks of interest include growing a green wall with user-defined properties. For example, a user can specify to have no growth into a certain area and particularly much biomass in another area. The distributed robot system then detects the current condition and coverage of the plants on the wall, steers individual plant tips and coordinates the growth control among the robots. These will be the tasks that we will investigate with the methodology introduced in this paper.

The novel paradigm underlying this work is to use natural plants for engineering tasks. Once the step to 3D setups is done, steering growth can be used similarly to additive manufacturing (e.g., 3D printing) with the advantage of almost zero material costs but with the challenge of low speed. Our vision is to enable plants to grow into desired forms, shapes, and functions supported and guided by robots. For example, plants could be used to connect building material and a long-term vision could be to enable them to enter the domain of actual robot tasks, such as grasping or monitoring.

6 Conclusion and future work

In this paper, we presented our approach to a bio-hybrid of robots and natural plants. Here, the robotic setup consists of a camera as a sensor for the plant's behavior and two LEDs as actuators providing stimuli for the plant to be controlled. First, we developed a data-driven model of the interaction between the robotic setup and the plant's tip. For that, we carried out a set of initial experiments with a predetermined open-loop robotic controller that generated a regular pattern of on-off command for the LEDs. An image processing method was used to collect information about the behavior of the plant's tip in reaction to the LEDs. The model was used to simulate the tip behavior for evolving controllers that steer a plant's tip to approach a set of randomly generated targets. We tested two different types of fitness functions that require different levels of a priori knowledge in their design, and therefore different speeds of convergence. The evolved controllers were tested on the setups with natural plants. The successful control of the plant approaching several different targets demonstrated the successful bridging of the reality gap in this particular setup.

In the future, we will extend the method by modeling the whole stem and aiming for control over the shape of the stem. The model can be also extended to 3D dynamics by adding another camera direction to the setup. With the extended setup we can define more complex tasks, for example reaching target points in 3D space or growing a spiral shape around an obstacle. We will extend the setup with additional stimuli, for example to repel the plants. We also plan to simultaneously control several independent plants in order to steer plant material into a greater variety of patterns and shapes. We are extending our

approach to a decentralized setup where autonomous nodes containing sensors (e.g., infra-red sensors for proximity) and actuators will self-organize, reacting to and influencing the plants, reaching a synergistic behavior in the bio-hybrid.

Acknowledgments

Project '*flora robotica*' has received funding from the European Union's Horizon 2020 research and innovation program under the FET grant agreement, no. 640959.

References

[1] Renaud Bastien, Stéphane Douady, and Bruno Moulia. A unified model of shoot tropism in plants: Photo-, gravi- and propio-ception. *PLOS Computational Biology*, 11(2):1–30, 2015.

[2] Josh C. Bongard. Evolutionary robotics. *Communications of the ACM*, 56(8):74–83, 2013.

[3] Gilles Caprari, Alexandre Colot, Roland Siegwart, José Halloy, and Jean-Louis Deneubourg. Animal and robot mixed societies: building cooperation between microrobots and cockroaches. *IEEE Robotics & Automation Magazine*, 2005.

[4] Oscar E. Checa and Matthew W. Blair. Mapping QTL for climbing ability and component traits in common bean (phaseolus vulgaris l.). *Molecular Breeding*, 22(2):201–215, 2008.

[5] Peter Chervenski and Shane Ryan. MultiNEAT, project website, 2017.

[6] John M. Christie and Angus S. Murphy. Shoot phototropism in higher plants: New light through old concepts. *American Journal of Botany*, 100(1):35–46, 2013.

[7] Rodrigo da Silva Guerra, Hitoshi Aonuma, Koh Hosoda, and Minoru Asada. Behavior change of crickets in a robot-mixed society. *Journal of Robotics and Mechatronics*, 22(4):526–531, 2010.

[8] Mohammad Divband Soorati and Heiko Hamann. The effect of fitness function design on performance in evolutionary robotics: The influence of a priori knowledge. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 153–160. ACM, 2015.

[9] Roy Featherstone and David Orin. Robot dynamics: equations and algorithms. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 826–834, 2000.

[10] *flora robotica*. project website, 2017. http://www.florarobotica.eu.

[11] Paco Calvo Garzón and Fred Keijzer. Plants: Adaptive behavior, root-brains, and minimal cognition. *Adaptive Behavior*, 19(3):155–171, 2011.

[12] Alexey Gribovskiy, José Halloy, Jean-Louis Deneubourg, Hannes Bleuler, and Francesco Mondada. Towards mixed societies of chickens and robots. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4722–4728, Oct 2010.

[13] José Halloy, Gregory Sempo, Gilles Caprari, Colette Rivault, Mahdi Asadpour, Fabien Tâche, Imen Saïd, Virginie Durier, Stephane Canonge, Jean-Marc Amé, Claire Detrain, Nikolaus Correll, Alcherio Martinoli, Francesco Mondada, Roland Siegwart, and Jean-Louis Deneubourg. Social integration of robots into groups of cockroaches to control self-organized choices. *Science*, 318(5853):1155–1158, November 2007.

[14] Heiko Hamann, Mostafa Wahby, Thomas Schmickl, Payam Zahadat, Daniel Hofstadler, Kasper Stoy, Sebastian Risi, Andres Faina, Frank Veenstra, Serge Kernbach, Igor Kuksin, Olga Kernbach, Phil Ayres, and Przemyslaw Wojtaszek. *flora robotica* – mixed societies of symbiotic robot-plant biohybrids. In *Proc. of IEEE Symposium on Computational Intelligence (IEEE SSCI 2015)*, pages 1102–1109. IEEE, 2015.

[15] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, 17(1):122–145, 2013.

[16] Aristid Lindenmayer. Developmental algorithms for multicellular organisms: A survey of L-systems. *Journal of Theoretical Biology*, 54(1):3–22, 1975.

[17] Bernard Millet and Pierre-Marie Badot. The revolving movement mechanism in phaseolus: new approaches to old questions. *Vistas on Biorhythmicity*, 1:77–98, 1996.

[18] Andrew L. Nelson, Gregory J. Barlow, and Lefteris Doitsidis. Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, 57:345–370, 2009.

[19] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, ACM '68, pages 517–524, New York, NY, USA, 1968. ACM.

[20] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21(1):63–100, January 2004.

[21] Ondrej Stava, Soren Pirk, Julian Kratt, Baoquan Chen, Radomir Mech, Oliver Deussen, and Bedrich Benes. Inverse procedural modelling of trees. *Computer Graphics Forum*, 33(6):118–131, 2014.

[22] Maria Stolarz. Circumnutation as a visible plant action and reaction: physiological, cellular and molecular basis for circumnutations. *Plant Signaling & Behavior*, 4(5):380–387, 2009.

[23] Sebastian von Mammen and Christian Jacob. The evolution of swarm grammars – growing trees, crafting art, and bottom-up design. *IEEE Computational Intelligence Magazine*, 4(3):10–19, 2009.

[24] Mostafa Wahby, Daniel N. Hofstadler, Mary Katherine Heinrich, Payam Zahadat, and Heiko Hamann. An evolutionary robotics approach to the control of plant growth and motion: Modeling plants and crossing the reality gap. In *Self-Adaptive and Self-Organizing Systems (SASO), 2016 IEEE 10th International Conference on*, pages 21–30. IEEE, 2016.

[25] Richard A. Watson, Sevan G. Ficici, and Jordan B. Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, 2002.

[26] Payam Zahadat, Michael Bodi, Ziad Salem, Frank Bonnet, Marcelo E. d. Oliveira, Francesco Mondada, Karlo Griparic, Tomislav Haus, Stjepan Bogdan, Stjepan Mills, Pedro Mariano, Luis Correia, Olga Kernbach, Serge Kernbach, and Thomas Schmickl. Social adaptation of robots for modulating self-organization in animal societies. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2014 IEEE Eighth International Conference on*, pages 55–60, 2014.

[27] Payam Zahadat, Daniel N. Hofstadler, and Thomas Schmickl. Vascular morphogenesis controller: A generative model for developing morphology of artificial structures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 163–170, New York, NY, USA, 2017. ACM.

[28] Aleš Zamuda and Janez Brest. Vectorized procedural models for animated trees reconstruction using differential evolution. *Information Sciences*, 278:1–21, 2014.