phenopype: a phenotyping pipeline for Python

Moritz D. Lürig[1,2]

[1]Lund University, Department of Biology, Sölvegatan 37, 22362 Lund, Sweden

[2]Eawag, Department of Fish Ecology and Evolution, Seestr. 79, 6047 Kastanienbaum, Switzerland

Abstract

1. Digital images are a ubiquitous way to represent phenotypes. More and more ecologists and evolutionary biologists are using images to capture and analyze high dimensional phenotypic data to understand complex developmental and evolutionary processes. Therefore, images are being collected at ever increasing rates, already outpacing our abilities for processing and analysis of the contained phenotypic information.
2. phenopype is a high throughput phenotyping pipeline for the programming language Python to support ecologists and evolutionary biologists in extracting high dimensional phenotypic data from digital images. phenopype integrates existing state-of-the-art computer vision functions (using the OpenCV library as a backend), GUI-based interactions, and a project management ecosystem to facilitate rapid data collection and reproducibility.
3. phenopype offers two different workflows that support users during different stages of scientific image analysis. In the “low throughput workflow”, computer vision functions are used directly from a Python interpreter, which is suitable for prototyping or analysis of a few images. The “high-throughput workflow” stores image specific functions and settings in human-readable YAML format along with the processed images and results, which makes the acquired phenotypic information highly reproducible.
4. Building upon Python’s cutting-edge computer vision, array manipulation and data handling libraries, phenopype empowers users with a background in biology to come up

with their own solutions to image analysis problems. Overall, phenopype is aiming to augment, rather than replace the utility of existing Python computer vision libraries, allowing biologists to focus on rapid and reproducible data collection.

1 Introduction

Digital imaging has become one of the most widespread methods to study phenotypes in ecological and evolutionary research (Roeder, Cunha, Burl, & Meyerowitz, 2012; Pennekamp & Schtickzelle, 2013). Produced by a myriad of devices such as digital cameras, scanners and microscopes, high resolution images capture a variety of internal, external or behavioral characteristics of organisms. In some cases, analysis of digital images replaced manual tasks, e.g. when using landmarks instead of manual calipers. Additionally, biologists are using images to draw a more complete picture of organismal phenotypes, e.g. through the analysis of colour (Ezray, Wham, Hill, & Hines, 2019) and shape patterns (Church, Donoughe, de Medeiros, & Extavour, 2019), by tracing complex behavior (Dell et al., 2014; Manoukis & Collier, 2019), or by capturing microbial diversity (Zackrisson et al., 2016; French, Coutts, & Brown, 2018). Such data driven approaches bring the field closer to characterizing the phenome, i.e. the phenotype as a whole (Soulé, 1967), and to tracing causal links between phenotypes, genotypes and environmental conditions (Houle, Govindaraju, & Omholt, 2010). In consequence, digital images are being collected at ever increasing rates, which has already outpaced our abilities to efficiently process and analyze the contained phenotypic information.

Computer vision (CV), the automated extraction and processing of information from digital images, has great potential to reduce image analysis as an analytical bottleneck in ecological and evolutionary research (Lürig, Donoughe, Svensson, Porto, & Tsuboi, 2021). CV has already transformed data collection in various fields of research, such as medicine (Gao, Yang, Lin, & Park, 2018), remote sensing (Marmanis & Wegner, 2016), or material sciences (DeCost & Holm, 2015). In ecological and evolutionary research, however, software for CV is still only infrequently used to analyze image datasets (Weinstein, 2018;

Lürig et al., 2021). Potential hurdles may lie in tradeoffs between flexibility and scalability on the one hand, and ease of use on the other hand. For instance, standalone toolboxes with a graphical user interfaces (GUI) such ImageJ (Schneider, Rasband, & Eliceiri, 2012) are often preferred, because they offer a relatively broad range of CV-functions while being user friendly. However, batch analysis of larger image datasets is typically more convenient and efficient from within a high level programming language, such as Python, which has native file handling and array computing capabilities, as well as state-of-the art CV libraries. Yet, implementing an image analysis workflow from scratch often requires considerable programming knowledge and typically comes without even basic GUI functionality.

Here I introduce the Python package *phenopype*, which is a high throughput image analysis pipeline that allows ecologists and evolutionary biologists to extract high dimensional phenotypic data from digital images. phenopype resolves the aforementioned trade-off by combining a rich (and further expandable) collection of computer vision functions for high throughput image analysis with basic GUI functionality, allowing users to step into the automatic workflow when necessary. phenopype also comes with an efficient file and project management ecosystem to facilitate rapid data collection: all settings for the image analysis are stored along with processed images and results, so that the acquired phenotypic information becomes highly reproducible. Overall, phenopype is aiming to augment, rather than replace the utility of existing Python CV libraries. Put differently, phenopype does neither intend to be an exhaustive library of granular image processing functions, like OpenCV or scikit-image, nor a purely GUI-based toolkit like ImageJ, but instead, it is a set of wrappers and convenient management tools that allows biologists to focus on rapid and reproducible data collection.

2 Using phenopype

2.1 Implementation and installation

phenopype (v2.0.1) is written in Python (v3.7), and has passed peer review at the pyOpenSci consortium ((Lürig, 2021); also see “Data availability statement”). Today, Python is the most widespread high level programming language (Robinson, 2017; Kambayashi, 2018) due to its intuitive syntax and flexibility. Python is the ideal programming-environment for image analysis, due to its vast ecosystem of state-of-the-art image processing, image analysis and machine learning libraries with prolific communities of users and developers. phenopype uses the OpenCV (“open computer vision”) library as the main backend for image analysis ((Bradski, 2000), implemented through the *opencv-python* package). OpenCV has a comprehensive catalogue of extremely performant CV functions that are typically used for real time applications (Pulli, Baksheev, Kornyakov, & Eruhimov, 2012) and deep learning based image analysis (Raschka & Mirjalili, 2017). Furthermore, phenopype uses *pyradiomics (van Griethuysen et al., 2017)* for the extraction of texture features, *numpy* for array manipulations (Virtanen et al., 2020), and *pandas* for data handling *(McKinney & Others, 2011)*.

phenopype is primarily available through the Python Package Index (PyPI) and can be directly installed via *pip,* the default Python package manager (*pip install phenopype*). New phenopype users and Python beginners should consult the phenopype documentation (https://mluerig.github.io/phenopype/; (Lürig, 2021)) , where extended installation instructions, the full API reference, and tutorials and example applications of phenopype are available.

2.2 Range of computer vision functions

Depending on image quality and the complexity of the trait or species of interest, the extraction of phenotypic data via CV can involve many different processing steps (Lürig et

al., 2021). In phenopype, these steps include “preprocessing”, “segmentation”, “measurement”, “visualization”, and “export”, which constitute the five core modules containing the functions relevant for those steps (Fig. 1). Preprocessing prepares images for the CV routines to follow, e.g. by reducing or adding noise, labelling areas of interest, or detecting reference cards. The central step in phenotyping-related CV is the segmentation of images into pixels that contain the desired trait or specimen (the foreground) and all other pixels (background). In its current form, phenopype offers a selection of signal processing algorithms (threshold, watershed, foreground-background subtraction), but future versions will include the possibility to train machine learning classifiers for segmentation. After segmentation, characteristics of the foreground can be measured (e.g. the area, shape or texture of a feature), but there are, of course, also characteristics that can be measured without prior segmentation, such as manually placed landmarks or segments. Finally, segmentation results can be visualized, and the measurements exported as comma-separated value files for statistical analysis.

Some of the functions in the preprocessing, segmentation and measurement modules feature GUI user interactions. Examples are the creation of masks during preprocessing to include or exclude certain parts of the image from the following steps, editing of imperfect segmentation, or manual measurements such as landmarks. GUI interactions are reduced to the bare minimum to facilitate high throughput. Furthermore, although there is a certain degree of sequential dependency between these modules and their functions (e.g. pixels need to be segmented and measured before data can be exported), they don’t necessarily have to be all executed as a linear, one directional workflow, but can run iteratively (e.g. preprocessing needs to be adjusted to segmentation outcomes). Finally, phenopype also includes a video analysis module which allows segmentation of video frames through foreground-background subtraction. This algorithm first determines what constitutes the background by detecting static pixels over a range of frames, and then determines foreground by determining pixels that - through moving - deviate from the background model. The video module, although fully functional and with

complete documentation, is currently only weakly integrated into the different phenopype workflows and is listed here for completeness.

2.3 Low- vs. high-throughput

Especially early during an experiment or survey when images are first evaluated on the computer screen, scientific image analysis can be a highly iterative process that may require frequent user input to preprocess and segment images, and then evaluate the obtained results. Later, when the best functions and appropriate settings are found and rapid data collection has priority, image analysis should be seamless and with minimal user input for maximal throughput, but also with maximal scientific reproducibility. In phenopype, users can choose between “low-throughput”, and “high-throughput” workflow that accommodate the different stages in the process of scientific image analysis.

In the low-throughput workflow, functions from the five core modules have to be explicitly coded in a Python script (Fig. 1B). This allows users to easily build and interact with the functions, and their input and output. However, this also means that settings need to be modified for each image separately, which makes this workflow inconvenient for larger datasets. The high-throughput workflow has a radically different principle: instead of supplying the image analysis function-stack through a single Python script, it is supplied in the human readable YAML format (Fig. 1C). For large projects, initially a single YAML template with appropriate settings for the average image is created, copied and stored alongside each image. Then, through a special function, the *pype,* these configurations are automatically parsed back into Python code, executed, and results stored together with each image. By passing on the YAML-configuration and the raw images, anyone with a phenopype installation can recreate the exact CV workflow used to collect data with two lines of Python code.

3 High throughput phenotyping

To facilitate efficient analysis of larger datasets (100s - 1000s of images), phenopype has built-in management functions to store and organize raw image data, YAML configurations, and results. These management functions, although in principle also applicable to the low-throughput workflow, unfold their full potential when used in conjunction with the *pype* function (Fig. 1C, 2). To start the high-throughput process, first, a *project* object needs to be initialized, which is a Python class that contains methods to add files (using *add_files*) and create a directory tree where copies of raw images, image-attributes, *pype*-configurations and results are stored (Fig 2A). Then users need to create a *pype*-configuration in YAML format and store it within each folder of the phenopype directory tree (Fig. 2B). This can be done automatically using *add_config,* using one of several configuration templates for different cases (e.g. landmarking, colour scoring, or shape measurement, and more). Refer to the documentation for a complete overview (https://mluerig.github.io/phenopype/; (Lürig, 2021)). The next step is the actual image analysis. Loading a project folder that contains the image and configuration files with the *pype* function will trigger three events (Fig. 2C):

i. the *pype* will open the contained yaml configuration with the default OS text editor
ii. the *pype* will parse and execute the contained functions from top to bottom
iii. the *pype* will open a GUI window showing the processed image

Once the *pype* has finished running all functions contained in the YAML configuration file from top to bottom, the results are visualized. Now users can decide to either modify the opened configuration file (e.g. either change function parameters or add new functions), which will trigger to run the *pype* again, or to close the GUI window, which will terminate the *pype* and save all results to the folder. By including the *pype* function in a simple Python *for* loop over all image directories, users can continue with this procedure throughout the entire project dataset, eliminating the need to manually open, close or save any of the images - all is managed from within the YAML configuration file. Furthermore, by providing unique names, users can store different *pype* configurations and the associated results side by side

without having to create new directories (Fig. 2D). When all images have been analyzed, users can use *collect_results* to extract the obtained results from each project subfolder, and store them in a new folder in the project root directory for further processing in the users' favourite statistical environment.

4 Feature demonstration

Here I demonstrate some of phenopype's key features that allow rapid phenotyping, and give examples of different types of trait data that can be extracted. The following paragraphs correspond to the panels in Fig. 3. More extensive examples with code can be found in the online documentation of phenopype (https://mluerig.github.io/phenopype/; (Lürig, 2021)).

A) *Counting snails:* Counting small organisms is a typical image analysis and relatively easy to solve with computer vision. Here, phenopype was used to count freshwater snails (*Potamopyrgus antipodarum*), and measure their body size, shape and pigmentation. This was achieved through consecutive use of threshold (to segment black snails from white background), and watershed algorithms (to separate snails that are touching each other). Small fecal pellets are filtered from the detected foreground (the green contours) by setting an area limit, and the bounding box around each specimen is used to correct for the slight gradient in background brightness.

B) *Stickleback landmarks and armour plating*: Functional morphology of organisms is often measured by placing landmarks at specific points, but variation in continuous phenotypic traits like shape or area are difficult to quantify manually because they are too complex or have no underlying assumption of homology. In this example, the traits of interest are contained in 22 morphometric landmarks that were set with phenopype across the anterior half of a threespine stickleback (*Gasterosteus aculeatus*), which was stained with alizarin red. In addition, the number, area and extension of armour plating was measured as a continuous trait: first, a mask was set

around the posterior region that contains the plates, then the red colour channel of the image was thresholded (because it has the highest signal-to-noise ratio).

C) *Venation patterns in damselfly wings*: Here, phenopype was used to measure the spacing, angles, and connectivities within the wing tissue of an adult damselfly (*Ischnura elegans*). Because the venation within the wing has an extremely high signal-to-noise ratio, a single thresholding algorithm could be used without further preprocessing. The full hierarchy of foreground contours was resolved: the outer perimeter of the whole wing (in green) and all domains within the wing (in red).

D) *Isopod movement in response to predation:* In the given example, the goal was to track the movement of freshwater isopods (*Asellus aquaticus*) in response to presence, absence and activity of a predator (*G. aculeatus*). The output of the phenopype's video tracking module (foreground-background subtractor) is a time series of coordinates that correspond to the trajectories of predator and prey. In addition, the FGBG algorithm produces a binary segmentation mask for every frame. This allows the detection of pigmentation and body size of every isopod, and thus the sequence in which the fish consumed differently pigmented and sized specimens.

E) *Phytoplankton cell shape and fluorescence:* A plate reader produced bright-field and fluorescence emission images of freshwater phytoplankton communities (shown here is *Pediastrum sp.*). Here the goal was to measure the shape (cell morphology) and texture (fluorescence intensity) of cells. This was done in a two step process: first, the brightfield image was thresholded to detect the boundary contours of the cell. Then, the contours were passed on to the fluorescence image, where intensities were measured within the contour of the cell.

F) *Automatic reference card detection*: Even in highly standardized laboratory environments, there are often slight variations in camera exposure and zoom that may affect the outcome of one CV-based measurement. To automate the time-consuming task of correcting for size ratio and brightness variation, phenopype uses image registration algorithms (Tareen & Saleem, 2018) to detect reference cards

within images, and automatically correct size and colour histograms of that image. This can be done on a project wide scale, where first a template is created, and then detected in every image analyzed with any of the three workflows.

5 Discussion and outlook

With *phenopype* I have presented a novel approach for the efficient extraction of phenotypic data from digital images in Python through the combination of efficient CV-functions, GUI interactions, project management, and convenience function for a seamless workflow. Users can assemble the provided processing functions, segmentation algorithms and measurement routines in manifold ways to create a customized workflow that addresses the specifics of their respective study organism, while maintaining full scientific reproducibility. phenopype has already proven its effectiveness by producing large amounts of phenotypic data for two peer reviewed publications (Lürig, Best, Svitok, Jokela, & Matthews, 2019; Lürig & Matthews, 2021). Currently *phenopype* provides CV functions that are based on signal processing, which work instantaneously (i.e., they don't require any training), have lower technical hurdles (no GPU required), and are less complex to implement. This limits the use of phenotype to images with high signal-to-noise ratio, as produced in a standardized laboratory environment where conditions can be controlled.

In the long term, *phenopype* may serve as a stepping stone towards technically more involved machine learning approaches. Beside the phenotypic data, phenopype produces foreground masks as a "by-product" and thus provides labelled training datasets, which are an increasing bottleneck for deep learning approaches (Sun, Shrivastava, Singh, & Gupta, 2017). Furthermore, Python, specifically the OpenCV package that phenopype is using, already comes with a suite of state-of-the-art deep learning implementations, which are frequently updated with the most cutting-edge algorithms (e.g. Mask-RCNN (Abdulla, 2017)). phenopype could therefore be easily extended to include current and future advances within the highly dynamic field of scientific CV.

Data availability statement

A snapshot of phenopype in version 2.0.1, as well as the full documentation, including installation instructions, API reference and tutorials can be found on Zenodo (https://zenodo.org/record/4761417 (Lürig, 2021)) and on GitHub (master branch: https://github.com/mluerig/phenopype; documentation branch: https://mluerig.github.io/phenopype/).

Acknowledgements

I would like to thank Blake Matthews and the Eawag directorate for funding my work on the package. I conceived phenopype in its current form during a lab retreat in Vna (Graubünden, Switzerland) that was sponsored by Jukka Jokela. I thank Kim Kaltenbach for being a patient alpha tester, and Cam Hudson, Ryan Greenway, Nare Ngoepe, Anja Merz and Irene Gallego for being helpful beta testers. I would also express my gratitude to Arthur Porto and Seth Donoughe whose review at *pyOpenSci* greatly improved the presentation and documentation of the package. Finally, This package might not exist without Matt McGee, who encouraged me to learn Python for computer vision - thank you Matt!

References

Abdulla, W. (2017). Mask r-cnn for object detection and instance segmentation on keras and tensorflow. Retrieved from https://github.com/matterport/Mask_RCNN

Bradski, G. (2000). The opencv library. *Doctor Dobbs Journal*, *25*(11), 120–126.

Church, S. H., Donoughe, S., de Medeiros, B. A. S., & Extavour, C. G. (2019, July). Insect egg size and shape evolve with ecology but not developmental rate. *Nature*. doi:10.1038/s41586-019-1302-4

DeCost, B. L., & Holm, E. A. (2015). A computer vision approach for automated analysis and classification of microstructural image data. *Computational Materials Science*, *110*, 126–133. doi:10.1016/j.commatsci.2015.08.011

Dell, A. I., Bender, J. A., Branson, K., Couzin, I. D., de Polavieja, G. G., Noldus, L. P. J. J., … Brose, U. (2014). Automated image-based tracking and its application in ecology. *Trends in Ecology & Evolution*, *29*(7), 417–428. doi:10.1016/j.tree.2014.05.004

Ezray, B. D., Wham, D. C., Hill, C. E., & Hines, H. M. (2019). Unsupervised machine learning reveals mimicry complexes in bumblebees occur along a perceptual continuum. *Proceedings. Biological Sciences / The Royal Society*, *286*(1910), 20191501. doi:10.1098/rspb.2019.1501

French, S., Coutts, B. E., & Brown, E. D. (2018). Open-Source High-Throughput Phenomics of Bacterial Promoter-Reporter Strains. *Cell Systems*, *7*(3), 339–346.e3. doi:10.1016/j.cels.2018.07.004

Gao, J., Yang, Y., Lin, P., & Park, D. S. (2018). Computer Vision in Healthcare Applications. *Journal of Healthcare Engineering*, *2018*, 5157020. doi:10.1155/2018/5157020

Houle, D., Govindaraju, D. R., & Omholt, S. (2010). Phenomics: the next challenge. *Nature Reviews. Genetics*, *11*(12), 855–866. doi:10.1038/nrg2897

Kambayashi, S. (2018, July 26). Python is becoming the world's most popular coding language. Retrieved 15 March 2020, from https://www.economist.com/graphic-detail/2018/07/26/python-is-becoming-the-worlds-most-popular-coding-language

Lürig, M. D. (2021). *Data from: phenopype: a phenotyping pipeline for Python*. doi:10.5281/zenodo.4609990

Lürig, M. D., Best, R. J., Svitok, M., Jokela, J., & Matthews, B. (2019). The role of plasticity in the evolution of cryptic pigmentation in a freshwater isopod. *The Journal of Animal Ecology*, *88*(4), 612–623. doi:10.1111/1365-2656.12950

Lürig, M. D., Donoughe, S., Svensson, E. I., Porto, A., & Tsuboi, M. (2021). Computer Vision, Machine Learning, and the Promise of Phenomics in Ecology and Evolutionary Biology. *Frontiers in Ecology and Evolution*, *9*, 148. doi:10.3389/fevo.2021.642774

Lürig, M. D., & Matthews, B. (2021). Dietary-based developmental plasticity affects juvenile survival in an aquatic detritivore. *Proceedings of the Royal Society B: Biological Sciences*, *288*(1945), 20203136. doi:10.1098/rspb.2020.3136

Manoukis, N. C., & Collier, T. C. (2019). Computer Vision to Enhance Behavioral Research on Insects. *Annals of the Entomological Society of America*. doi:10.1093/aesa/say062

Marmanis, D., & Wegner, J. D. (2016). Semantic segmentation of aerial images with an ensemble of CNNs. *Annals of the …*. Retrieved from https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/III-3/473/2016/isprs-annals-III-3-473-2016.pdf

McKinney, W., & Others. (2011). pandas: a foundational Python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, *14*(9).

Pennekamp, F., & Schtickzelle, N. (2013). Implementing image analysis in laboratory-based experimental systems for ecology and evolution: a hands-on guide. *Methods in Ecology and Evolution / British Ecological Society*, *4*(5), 483–492. doi:10.1111/2041-210X.12036

Pulli, K., Baksheev, A., Kornyakov, K., & Eruhimov, V. (2012). Realtime Computer Vision with OpenCV: Mobile computer-vision technology will soon become as ubiquitous as touch interfaces. *Queueing Systems. Theory and Applications*, *10*(4), 40–56. doi:10.1145/2181796.2206309

Raschka, S., & Mirjalili, V. (2017). Python Machine Learning: Machine Learning and Deep Learning with Python. *Scikit-Learn, and TensorFlow. Second Edition Ed*. Retrieved from https://www.igi-global.com/pdf.aspx?tid%3D267132%26ptid%3D254262%26ctid%3D17%26t%3Dpython+machine+learning%3A+machine+learning+and+deep+learning+with+python%2C+scikit-learn%2C+and+tensorflow+2%2C+third+edition%26isxn%3D

Robinson, D. (2017, September 6). The Incredible Growth of Python | Stack Overflow. Retrieved 21 June 2019, from https://stackoverflow.blog/2017/09/06/incredible-growth-python/

Roeder, A. H. K., Cunha, A., Burl, M. C., & Meyerowitz, E. M. (2012). A computational image analysis glossary for biologists. *Development* , *139*(17), 3071–3080. doi:10.1242/dev.076414

Schneider, C. A., Rasband, W. S., & Eliceiri, K. W. (2012). NIH Image to ImageJ: 25 years of image analysis. *Nature Methods*, *9*(7), 671–675. doi:10.1038/nmeth.2089

Soulé, M. (1967). PHENETICS OF NATURAL POPULATIONS I. PHENETIC RELATIONSHIPS OF INSULAR POPULATIONS OF THE SIDE-BLOTCHED LIZARD. *Evolution; International Journal of Organic Evolution*, *21*(3), 584–591. doi:10.1111/j.1558-5646.1967.tb03413.x

Sun, C., Shrivastava, A., Singh, S., & Gupta, A. (2017, July 10). *Revisiting Unreasonable Effectiveness of Data in Deep Learning Era*. *arXiv [cs.CV]*. Retrieved from http://arxiv.org/abs/1707.02968

Tareen, S. A. K., & Saleem, Z. (2018). A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In *ResearchGate*. doi:10.1109/ICOMET.2018.8346440

van Griethuysen, J. J. M., Fedorov, A., Parmar, C., Hosny, A., Aucoin, N., Narayan, V., … Aerts, H. J. W. L. (2017). Computational Radiomics System to Decode the Radiographic Phenotype. *Cancer Research*, *77*(21), e104–e107. doi:10.1158/0008-5472.CAN-17-0339

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. doi:10.1038/s41592-019-0686-2

Weinstein, B. G. (2018). A computer vision for animal ecology. *The Journal of Animal Ecology*, *87*(3), 533–545. doi:10.1111/1365-2656.12780

Zackrisson, M., Hallin, J., Ottosson, L.-G., Dahl, P., Fernandez-Parada, E., Ländström, E., … Blomberg, A. (2016). Scan-o-matic: High-Resolution Microbial Phenomics at a Massive Scale. *G3* , *6*(9), 3003–3014. doi:10.1534/g3.116.032342

Figures

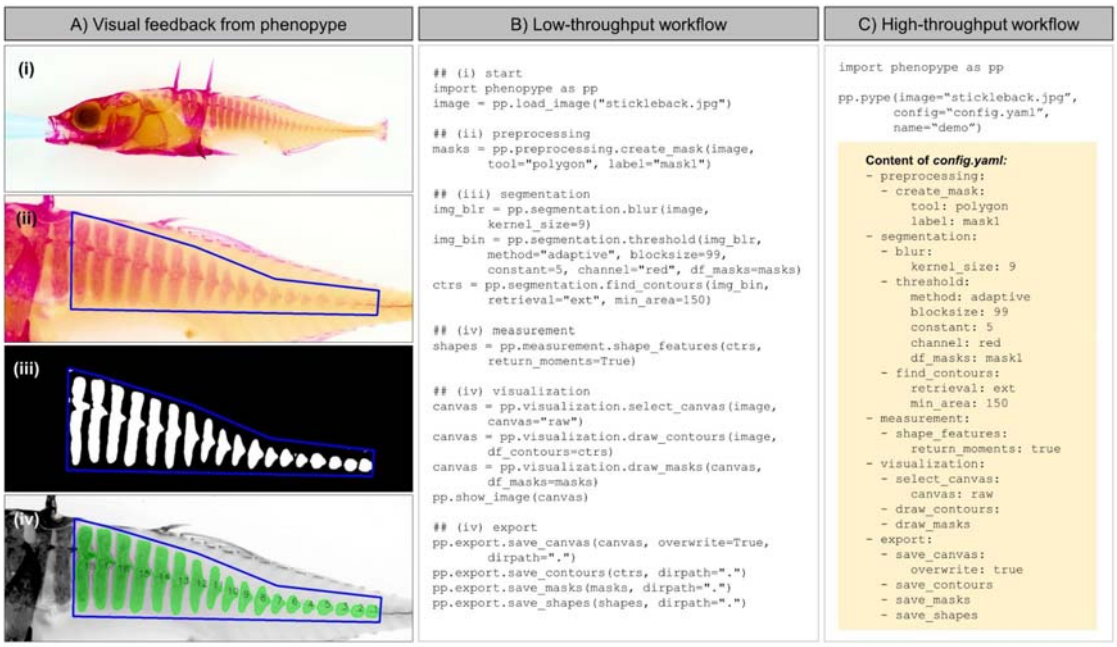


Figure 1 - A typical computer vision workflow in phenopype. A) The goal is to measure the number, area and extension of bony armor plating in a stained specimen of threespine stickleback (*Gasterosteus aculeatus*). Shown are (B) low-throughput and (C) high-throughput workflow, which use the same functions, but differ in the degree of Python code that is required. While in low-throughput all functions have to be explicitly coded in Python, the high-throughput routine parses the functions from human readable YAML files.

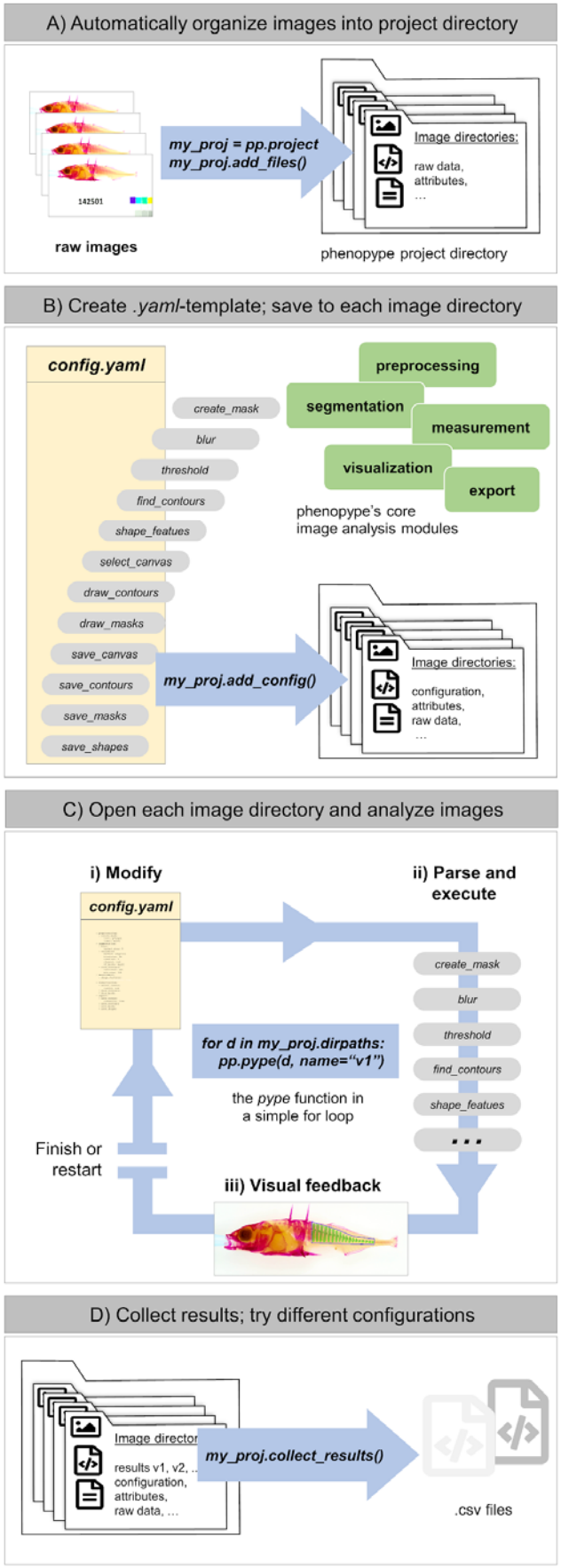

A) Automatically organize images into project directory
142501
raw images
my_proj = pp.project
my_proj.add_files()
Image directories:
raw data,
attributes,
...
phenopype project directory
B) Create .yaml-template; save to each image directory
config.yaml
create_mask
blur
threshold
find_contours
shape_featues
select_canvas
draw_contours
draw_masks
save_canvas
save_contours
save_masks
save_shapes
preprocessing
segmentation
measurement
visualization
export
phenopype's core
image analysis modules
my_proj.add_config()
Image directories:
configuration,
attributes,
raw data,
...
C) Open each image directory and analyze images
i) Modify
config.yaml
ii) Parse and
execute
create_mask
blur
threshold
find_contours
shape_featues
...
for d in my_proj.dirpaths:
pp.pype(d, name="v1")
the pype function in
a simple for loop
Finish or
restart
iii) Visual feedback
D) Collect results; try different configurations
Image director
results v1, v2, ...
configuration,
attributes,
raw data, ...
my_proj.collect_results()
.csv files

Figure 2 - High throughput phenotyping in phenopype. A) First, users create a project directory with the raw images that are organized into separate folders where all relevant data, attributes, configurations and results will be stored. B) A configuration file is created by the user with functions from phenopype's five core modules. The functions are saved in human-readable YAML format and stored in the image folders. C) The *pype* function will load both the images and configuration, execute the functions in sequence, and visualize the results, which can be repeated until the results are satisfactory. By continuing the *pype* function in a for loop, users can seamlessly process the entire image dataset. D) When all images have been analyzed, users can collect obtained results from each project subfolder, or try a different *pype* configuration.

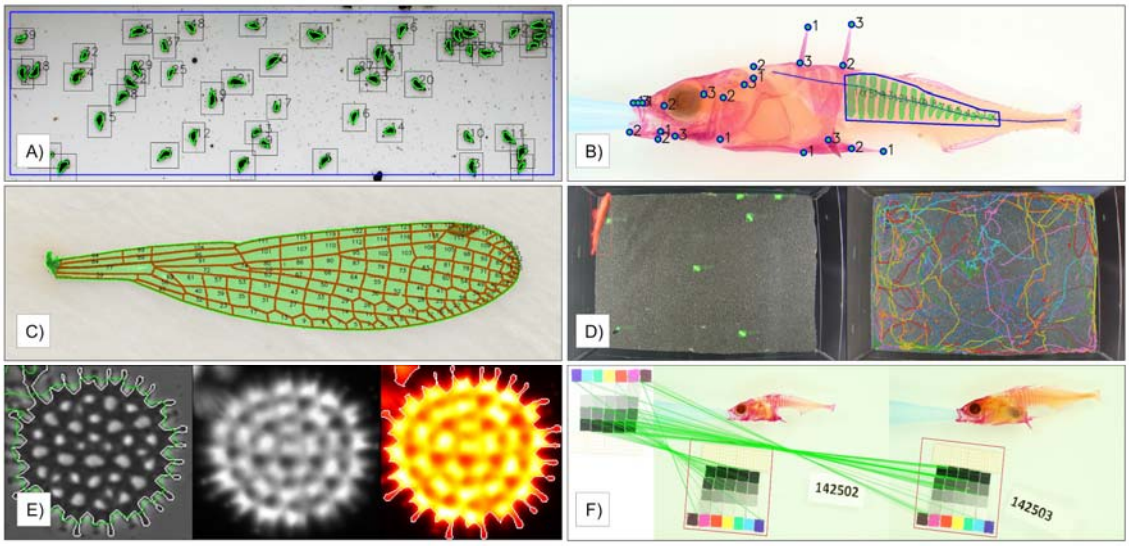
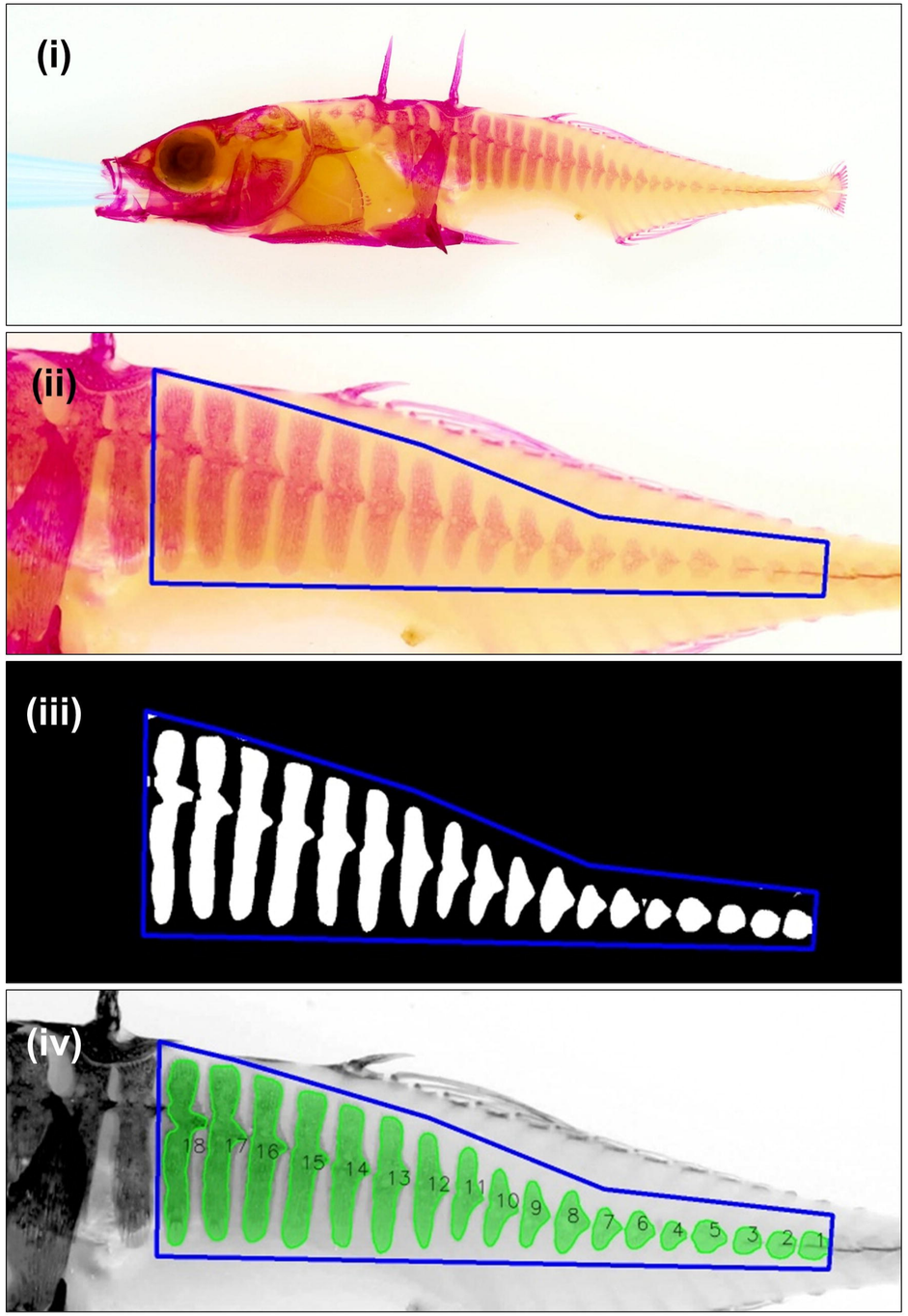


Figure 3 - Demonstration of major phenopype features (the panels correspond to the paragraphs in section 4). A) Counting and measuring freshwater snails (*Potamopyrgus antipodarum;* image by Jukka Jokela*)*. B) Landmarking and armor plate measurement in threespine stickleback (*Gasterosteus aculeatus*; image by Dominique Stalder). C) Quantifying venation patterns in damselfly wings (*Ischnura elegans*; image by Masahito Tsuboi). D) Tracking predator prey interactions between stickleback (*G. aculeatus)* and freshwater isopods (*Asellus aquaticus;* video shot by Kim Kaltenbach). E) Extracting shape and fluorescence traits from phytoplankton cells (*Pediastrum sp.*; image by Irene Gallego). F) Automatic detection of reference cards in images using phenopype (image by Dominique Stalder).

A) Visual feedback from phenopype



B) Low-throughput workflow

```
## (i) start
import phenopype as pp
image = pp.load_image("stickleback.jpg")

## (ii) preprocessing
masks = pp.preprocessing.create_mask(image,
       tool="polygon", label="mask1")

## (iii) segmentation
img_blr = pp.segmentation.blur(image,
       kernel_size=9)
img_bin = pp.segmentation.threshold(img_blr,
       method="adaptive", blocksize=99,
       constant=5, channel="red", df_masks=masks)
ctrs = pp.segmentation.find_contours(img_bin,
       retrieval="ext", min_area=150)

## (iv) measurement
shapes = pp.measurement.shape_features(ctrs,
       return_moments=True)

## (iv) visualization
canvas = pp.visualization.select_canvas(image,
       canvas="raw")
canvas = pp.visualization.draw_contours(image,
       df_contours=ctrs)
canvas = pp.visualization.draw_masks(canvas,
       df_masks=masks)
pp.show_image(canvas)

## (iv) export
pp.export.save_canvas(canvas, overwrite=True,
       dirpath=".")
pp.export.save_contours(ctrs, dirpath=".")
pp.export.save_masks(masks, dirpath=".")
pp.export.save_shapes(shapes, dirpath=".")
```

C) High-throughput workflow

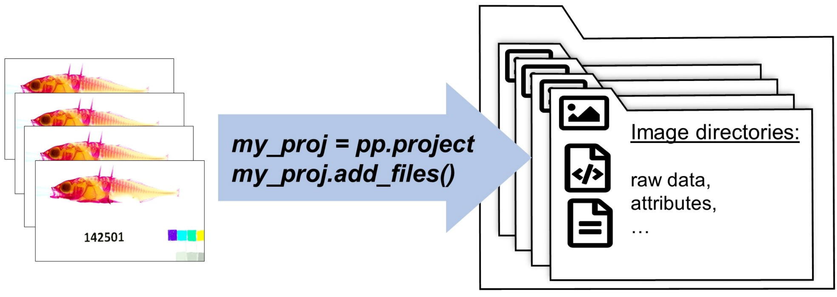
```
import phenopype as pp

pp.pype(image="stickleback.jpg",
       config="config.yaml",
       name="demo")
```

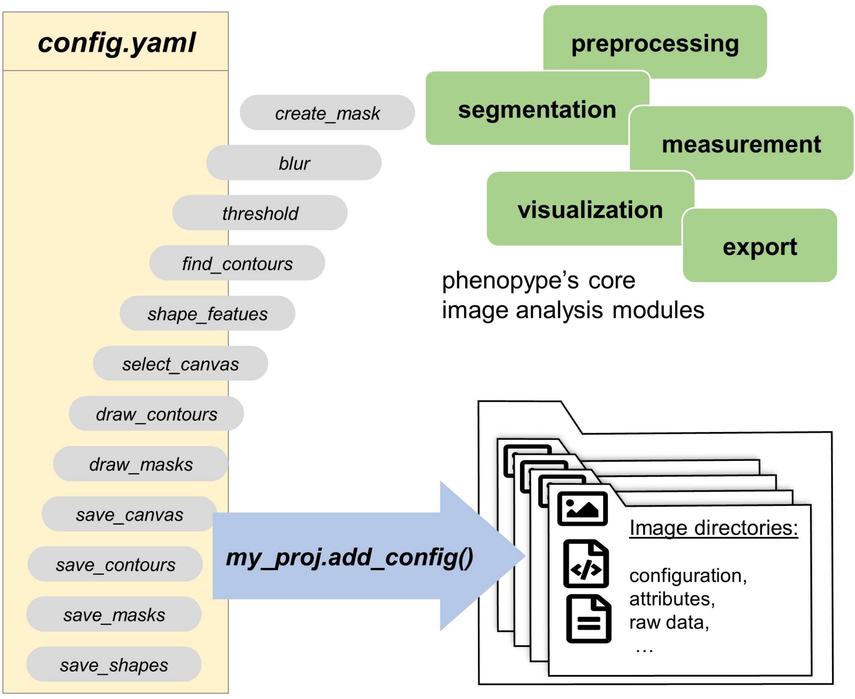
Content of *config.yaml:*

```
- preprocessing:
  - create_mask:
      tool: polygon
      label: mask1
- segmentation:
  - blur:
      kernel_size: 9
  - threshold:
      method: adaptive
      blocksize: 99
      constant: 5
      channel: red
      df_masks: mask1
  - find_contours:
      retrieval: ext
      min_area: 150
- measurement:
  - shape_features:
      return_moments: true
- visualization:
  - select_canvas:
      canvas: raw
  - draw_contours:
  - draw_masks
- export:
  - save_canvas:
      overwrite: true
  - save_contours
  - save_masks
  - save_shapes
```

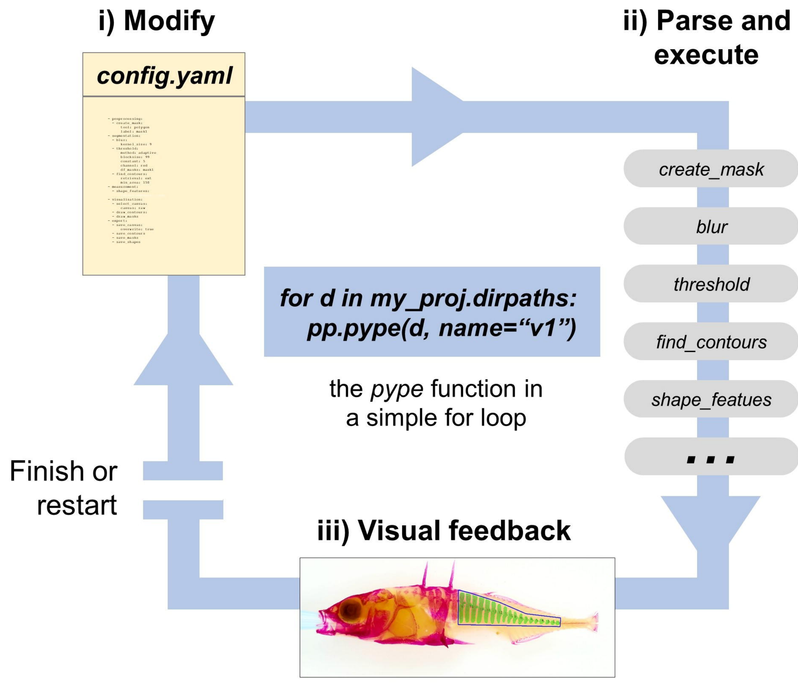
A) Automatically organize images into project directory



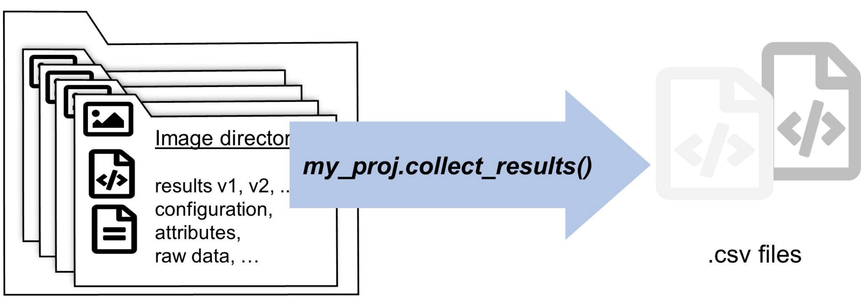
B) Create *.yaml*-template; save to each image directory

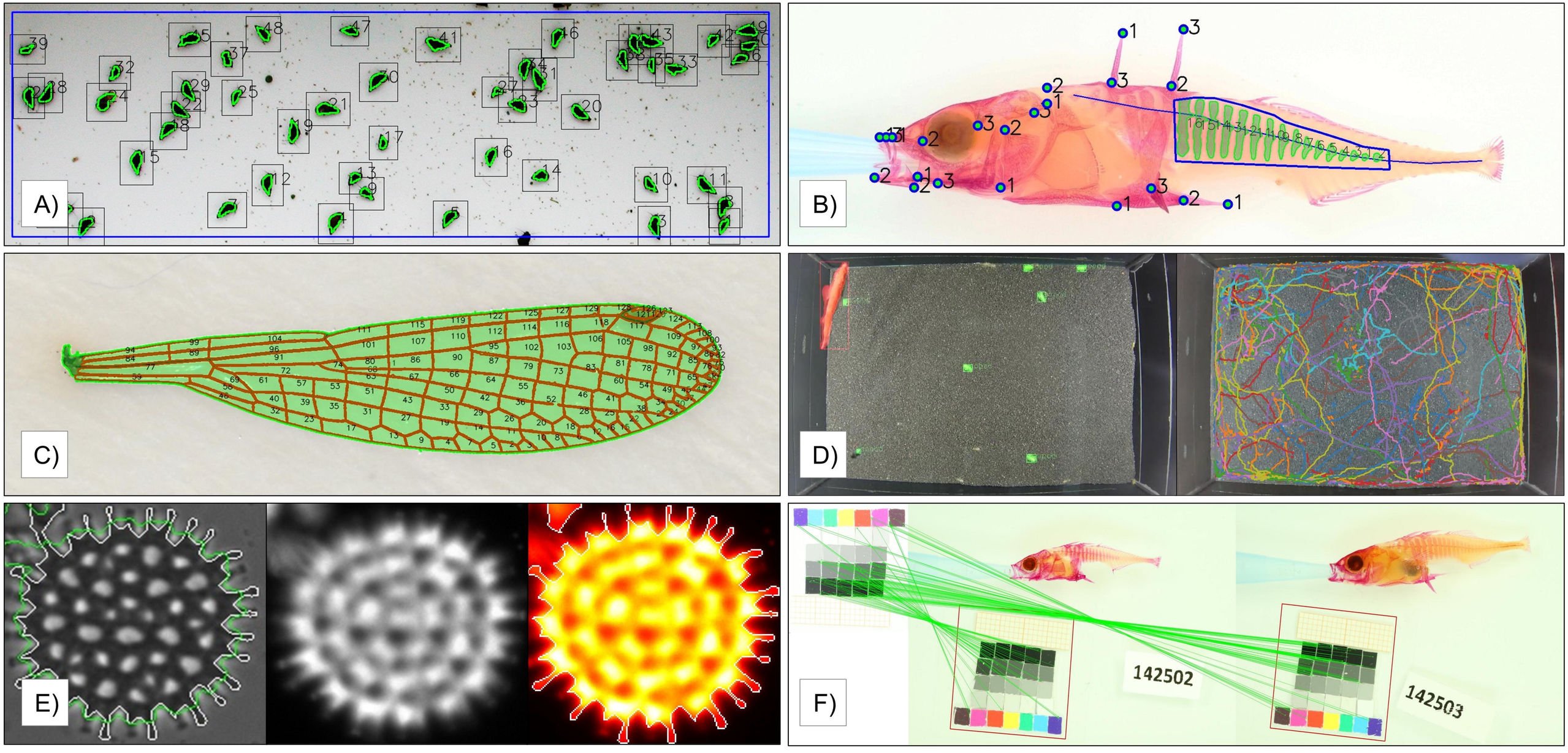


C) Open each image directory and analyze images



D) Collect results; try different configurations




A)
B)
C)
D)
E)
F)
142502
142503