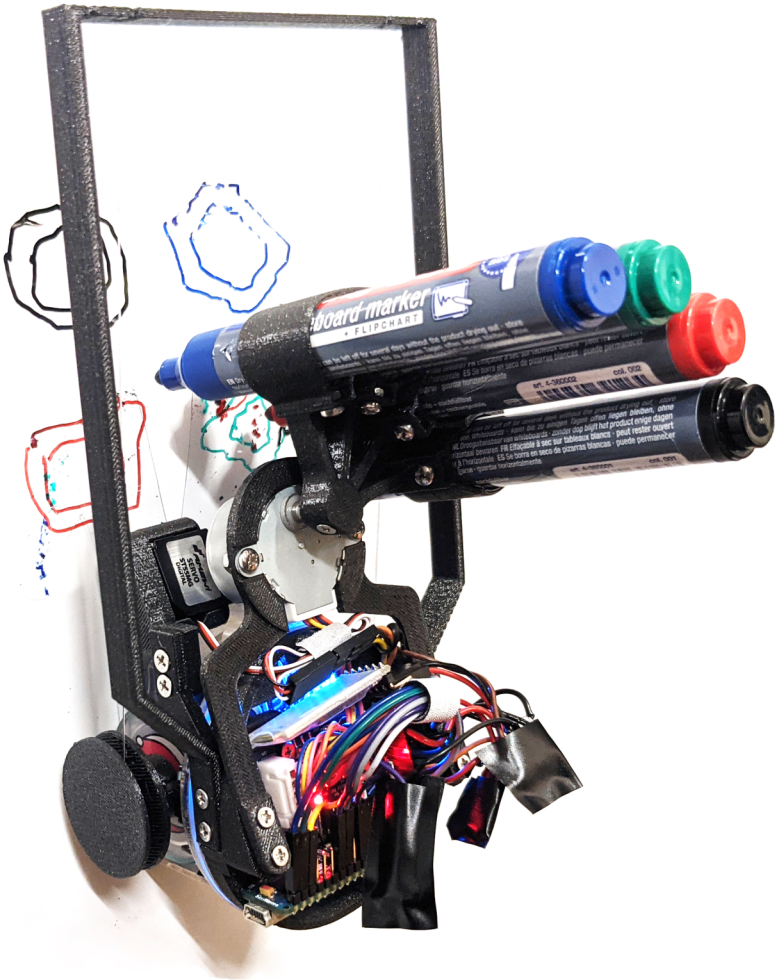
École polytechnique fédérale de Lausanne

Embedded Systems and Robotics
BA-6 Semester project
May 2021

epuck–artist

An edge processing, 4-color equipped wall plotter

Nguyen Thanh Vinh Vincent
Bakkali Yassine



Contents

1 Introduction

This project consists in a robot that can take a picture of any subject, process the image to generate a 4-color vectorized path of the picture, and finally reproduce the generated picture on a wall. This project was driven by the desire to not only follow the mandatory constraints, i.e. the use of the two stepper motors, a distance sensor and another sensor, but to also come up with a creative project idea that also makes use of other skills, such as 3D printing, communication protocols, mechanism design and electronics.
Source code, 3D printed files and more can be accessed on the **GitHub repository**.

2 General operation

2.1 Communication

The user can directly send commands to the e-puck or the arduino via Bluetooth with a Python script. Possible commands will be discussed in the following sections.

2.2 Setting starting point

At first, the user can send a command to manually move the e-puck to the desired drawing position. This is done by approaching one's hand close to the front, side and back IR sensors, to move forward, backward, left or right respectively. Once the user is satisfied with the starting position of the robot, a confirmation command can be sent, which will set the current position to the top center of the canvas.

2.3 Initial height calibration

Then, the user can calibrate the initial height of the robot. This height corresponds to the perpendicular distance between the robot's initial position and the segment relying the two thread attach points (see figure 2). It is crucial to get a good estimate of this starting position to get more accurate results, which should not be too stretched or too distorted. The calibration goes as follows:

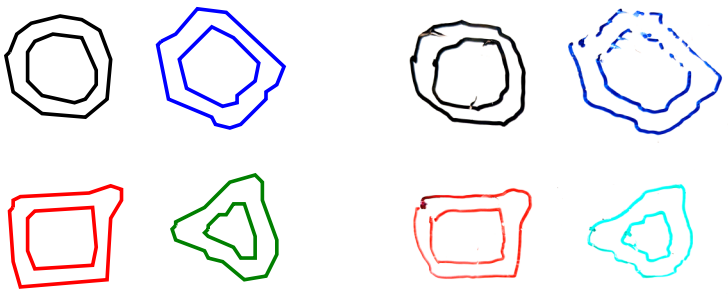
1. The robot draws two calibration points, spaced horizontally.
2. The user can then physically measure the distance between the two points and send it to the e-puck.
3. A flat surface is then placed under the robot, and if it stays steady for a certain amount of time, the robot goes down by the distance sent previously and measures the number of motor steps needed to reach this goal distance.
4. Once the goal distance is reached, it goes back to its starting position. From the previous measures, the e-puck can then determine its starting position.

2.4 Image capture

Once the robot is properly calibrated, the user can send a command to take a picture of the desired subject. The e-puck then processes the image and sends back intermediate images that illustrate the different steps of the image processing process. The images that are sent back can be seen in figure 7. The predicted path computed by the robot is visualized in figure 1a.

2.5 Drawing

Finally, when the user is satisfied with the predicted path sent back by the robot, a command to start drawing can be sent. The robot will then move across the canvas, change colors and lift the pens if necessary to draw the requested picture. An example picture is presented in figure 1b. More details about the communication aspect of this process are discussed in section 4.5.



(a) *Predicted path computed by the e-puck (converted to vector* `.svg` *file).*

(b) *Actual path traveled by the e-puck.* 10×10 cm.

Figure 1: *Comparison between computed path and actual path. Dotted line effect at the upper right is caused by the e-puck wobbling.*

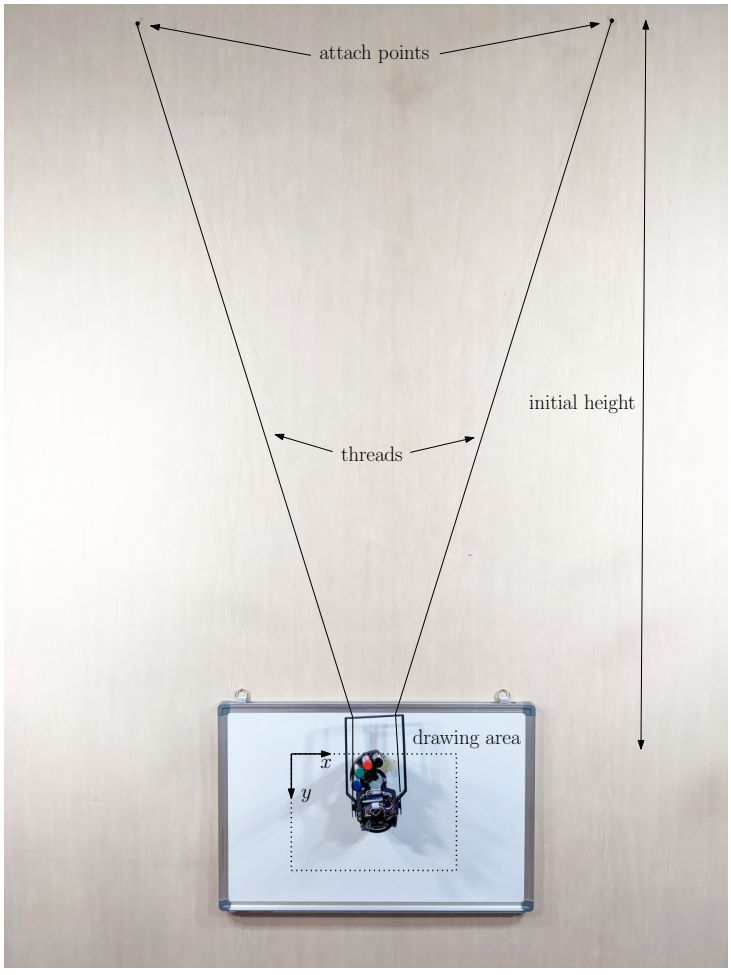


Figure 2: *Drawing setup.*

3 Extensions

3.1 External peripherals and Arduino

We started this project with the idea to attach a single pen to the e-puck. However, after completing this first task, we thought that it would be more interesting to interface the e-puck with an external microcontroller, which would allow us to control other peripherals. Figure 3 illustrates how external components were mounted. In our case, we have chosen an **Arduino Nano ATmega328** [5] (3), which controls the following peripherals: a servomotor (**ST55MG**) (10), a stepper motor (**CX28BYJ48-08** [4]) (2) and its driver (**Seeedstudio Stepper Motor Driver**) (6), and finally a Bluetooth module (**HC-05** [1]) (5) to enable wireless communication. See figure 9 for more details.

Of course, we were aware that this project should mainly be focused on the e-puck and its memory constraints, which is why no heavy calculations nor complex processing are done on the Arduino side of the project. The Arduino only receives requests from the e-puck (to change colors), and sends back a confirmation message when the color has successfully changed.

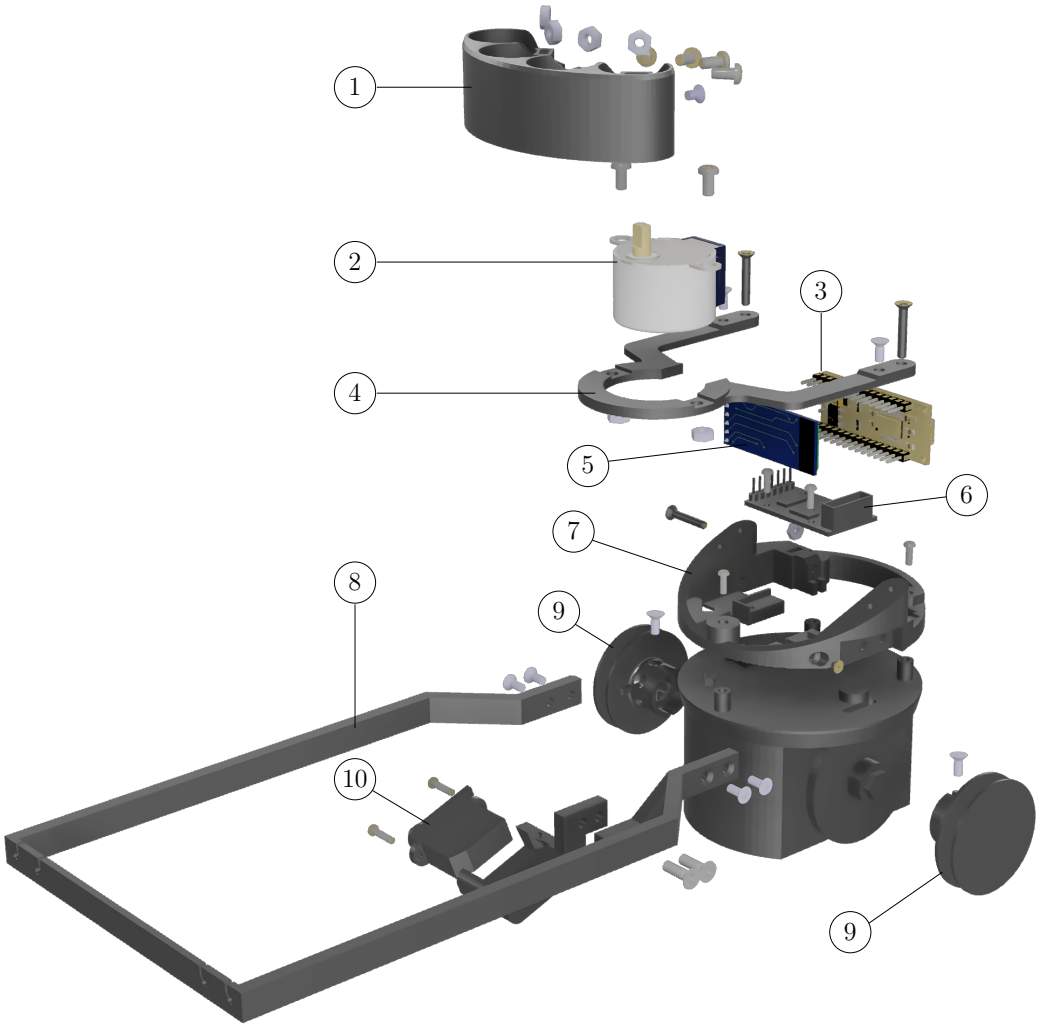


Figure 3: *Exploded view of the final mechanism and electronics. Wiring is not shown.*

3.2 3D printed parts and mechanism

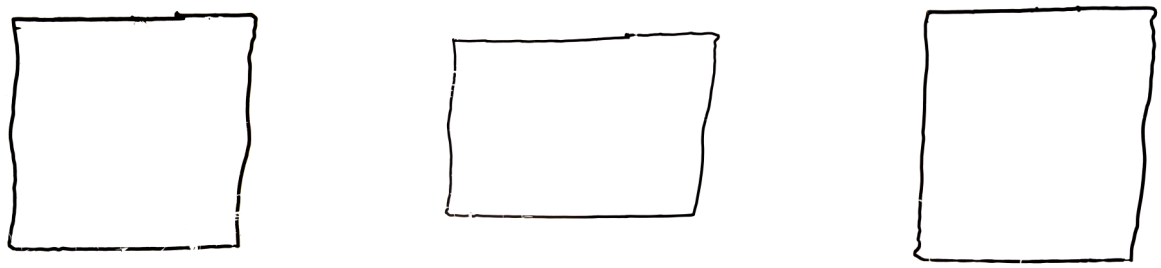
The mechanical parts are shown in figure 3. The pens are attached to a **tilted carriage** (1), mounted on a stepper motor (2), itself mounted on a **flexible pair of blades** (4). The slight angle of the carriage is here so that the pen that is directly in line with the robot's axis is lower compared to other pens. The flexible blades act as a spring, pushing down the pen on the drawing surface. When pen lifting is needed, a servomotor (10) placed under the stepper motor and mounted on its **support** rotates, lifting the whole pen carriage up so as not to touch the canvas. A transparent sheet of plastic is taped under the robot to avoid unwanted blemishes when the robot wobbles. The robot can move vertically and horizontally in any direction thanks to two threads attached around **pulleys** (9) specifically designed to be fixed to the wheels. The diameter of the pulley was chosen to be relatively small to avoid large torques. An elongated piece (**thread aligner**) (8) straightens the threads in front of the robot to minimize tilting when the robot moves or when the center of mass shifts due to color changes. The whole mechanism is mounted on the **main support** (7) that holds other electronic components.

4 Modules

The most important modules of this project will be briefly presented in the following subsections. Initial tests and choices will also be discussed.

4.1 `mod_calibration`

The goal of this module is to give to the user the ability to move the robot to a desired starting position and to calibrate the initial y-coordinate of the robot. As mentioned in section 2.3, it is important to have an accurate estimate of the starting position to get the best possible results. Examples of well/under/over-calibrated initial heights can be seen in figures 4a to 4c.



(a) *Well-calibrated.* (b) *Under-calibrated.* (c) *Over-calibrated.*

Figure 4: *Examples of a well/under/over-calibrated system. The waviness on the vertical lines is caused by nonuniform winding of the thread around the wheel pulleys.*

4.1.1 Initial tests

For the e-puck to reach the goal distance with precision, it is important to read very stable values, which is why a filter was implemented to reduce signal covariance. Multiple filters were tested and the results can be seen in figure 5, for which filter parameters were chosen to have the same response time. The best performing filter was found to be a single variable Kalman filter [6], chosen for its high stability, fast response time, easy implementation and finally because it is relatively not too computation heavy.

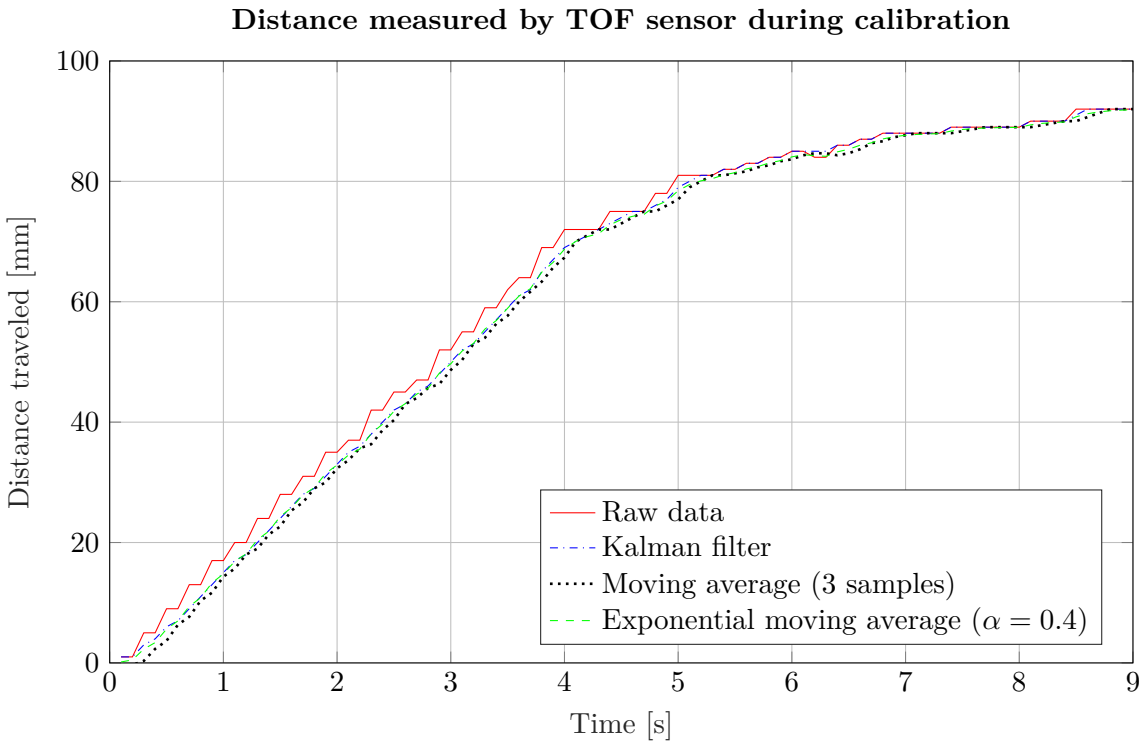


Figure 5: *Raw and filtered distance traveled during calibration, measured by TOF sensor.*

For the initial height calculation, we opted for an empirical, two variables approach (x_{mm}, horizontal distance in mm between the 2 calibration points and y_{st}, vertical distance in steps traveled to match x_{mm}), which requires one physical measurement (x_{mm}) from the user (y_{st} is measured by counting the number of steps, and the robot goes down by $y_{mm} = x_{mm}$, measured by TOF sensor). This is because an analytical method based only on the TOF measurements results in a highly non-linear initial height estimation, which, because of sensor noise, is not ideal. In the case of a two variable approach, a linear regression allowed us to find a relationship between x_{mm}, y_{st} and the initial length. The linear regression can be seen in figure 6.

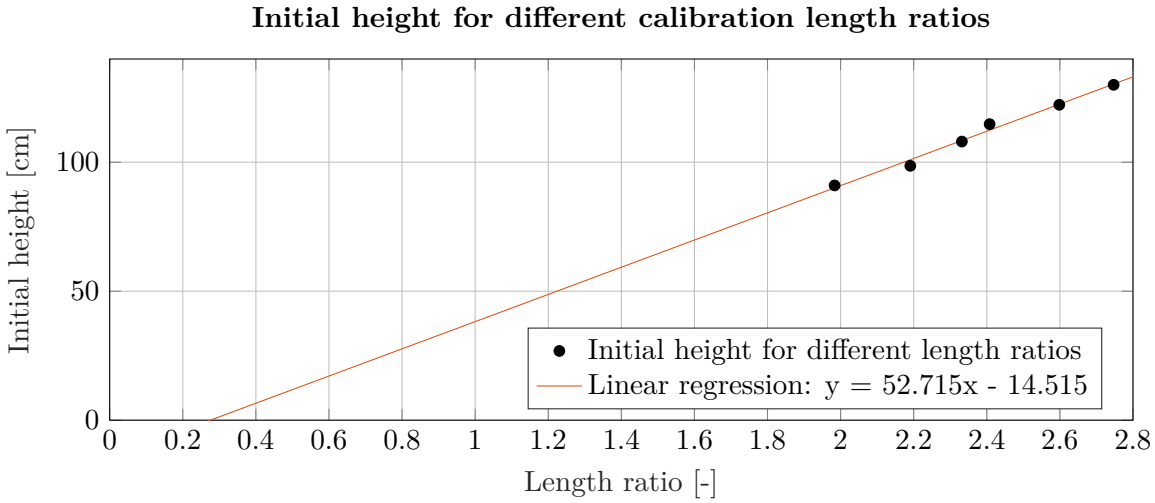


Figure 6: *Linear regression between initial height against calibration length ratio. The length ratio is determined by the ratio between y_{st} and $y_{st,est}$, where $y_{st,est}$ is an estimation of y_{st} calculated from x_{mm} for an arbitrary initial height (see `mod_calibration.c` for more details).*

4.2 `mod_communication`

This module ensures the communication between the e-puck and the computer. It enables image, path and message sending to the computer. This module also allows the computer to directly send movement and color data to the e-puck for debugging purposes.

4.2.1 Python script

A Python script was written to give to the user a way to directly interact with the e-puck and the Arduino. This script ensures the communication between the e-puck and the Arduino, relaying color change requests and color change confirmation messages. It also interprets image/path data from the e-puck. Results of these interpretations can be seen on figure 7 and 1a.

4.3 `mod_img_processing`

The image to be drawn is first captured and processed in this module with the call of a thread. A Canny edge detection algorithm [3] was implemented to detect both the edges of an image and their respective color. An illustration of the process can be seen on figure 7.

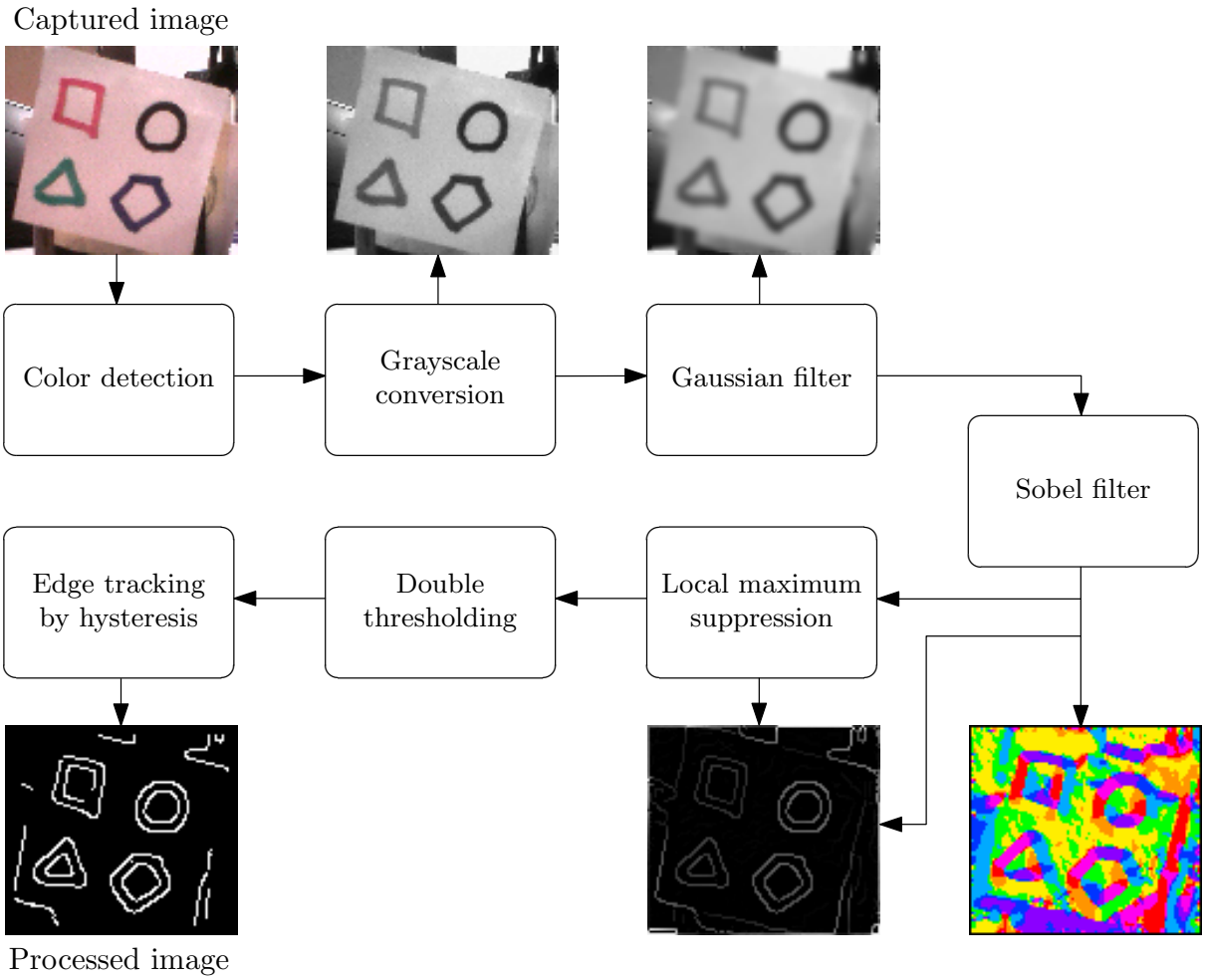


Figure 7: *Image processing steps. The different colors on the Sobel filter output gives an indication about the angle of a shape (see `mod_img_processing.c` for more details).*

4.4 `mod_path`

This module has two main purposes: path vectorization by ordering pixels and optimizing the calculated path, [2] and path color classification. The process is as follows :

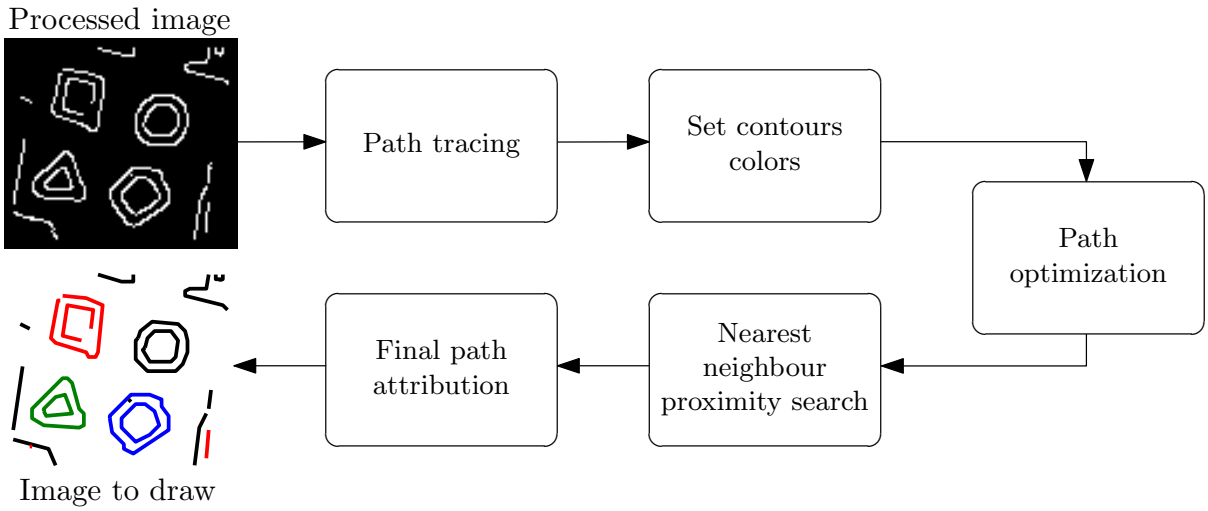


Figure 8: *Path vectorization and colorization from processed image (see `mod_path.c` for more details).*

Path optimization is the process in which redundant points are removed, to get a smoother output path.

4.5 `mod_draw`

This module contains the thread that controls the motors according to the given coordinates of the contours processed in the path vectorization module. It calculates the thread length required to reach a certain XY coordinate. When a color change or when pen lifting is needed, a message is sent from the e-puck to the computer which relays the message to the Arduino. The e-puck then waits for a confirmation message from the Arduino that indicates a successful completion of the requested action.

5 Conclusion

This project revealed itself to be an enriching learning experience. As we delved deeply into its realization, we managed to realize what we initially intended with a satisfying outcome despite the various limitations that we were confronted to. The limitations that we had to overcome were mainly: the torque limit on the stepper motors, sensor noise, power consumption and memory management. This has encouraged us to find creative solutions to bypass these problems (weight minimization, filters, low-power components,...). Furthermore, working with the e-puck's limited on-board memory made us realize how programming on an embedded system required high vigilance about memory allocation, data-types and optimizations. From an organisation standpoint, we are very satisfied about how we used version-managing tools such as Git, which allowed us to organize the project in multiple branches, and merge tools, which helped us easily merge new features in the final product.

References

[1] Paul Jan Armas. *HC-05*. [online; accessed 30 april 2021]. 2019. URL: https://grabcad.com/library/hc-05-4.

[2] Wikipedia contributors. *Ramer–Douglas–Peucker algorithm*. [online; accessed 21 april 2021]. URL: https://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm.

[3] Noah Kuntz. *Canny Tutorial*. [online; accessed 15 april 2021]. 2006. URL: http://www.pages.drexel.edu/~nk752/Research/cannyTut2.html.

[4] Achim Pieters. *28BYJ-48*. [online; accessed 30 april 2021]. 2021. URL: https://grabcad.com/library/28byj-48-5.

[5] Andrew Whitham. *Arduino Nano*. [online; accessed 30 april 2021]. 2013. URL: https://grabcad.com/library/arduino-nano--1.

[6] Wikipedia contributors. *Kalman filter*. [online; accessed 10 april 2021]. URL: https://en.wikipedia.org/wiki/Kalman_filter.

Appendix

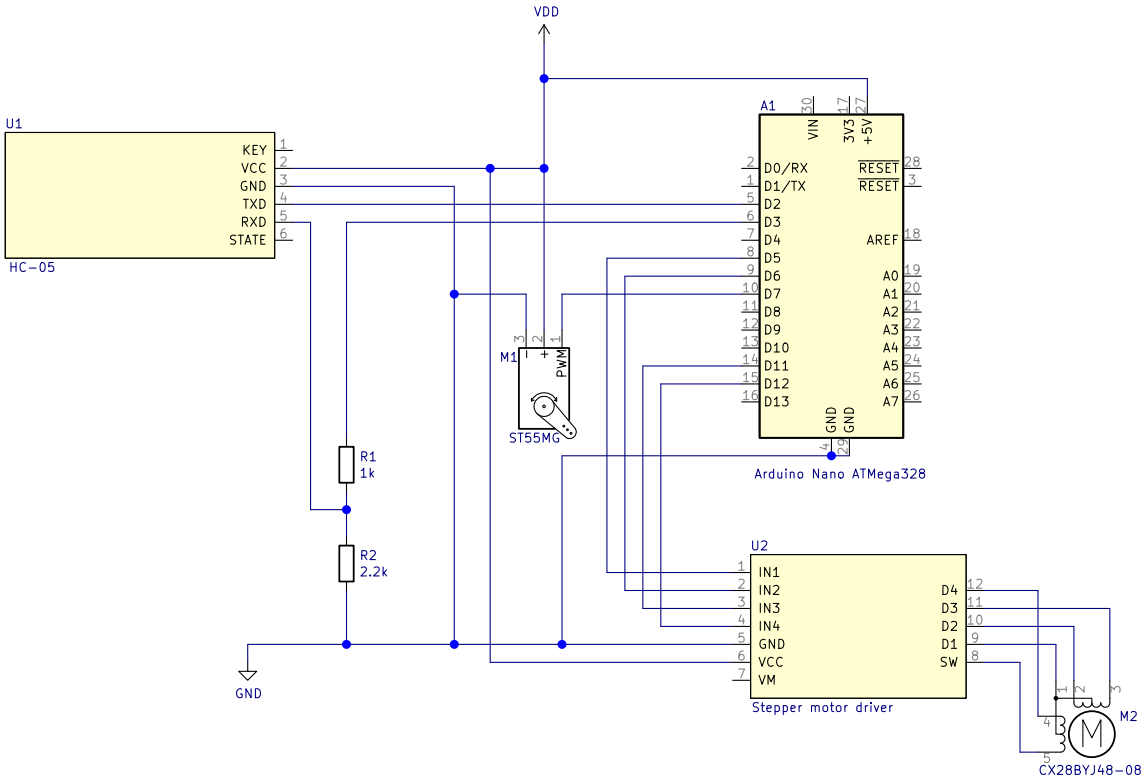


Figure 9: *Wiring for Arduino Nano and its peripherals.*