ROS 2 and EGM-Driven Sketching Pipeline for ABB Collaborative Robots

Bartek Łukawski*[1], Daniel G. Cubillo[2], Edwin Daniel Oña[1], Juan G. Victores[1], Carlos Balaguer[1], Alberto Jardón[1]
Departamento de Ingeniería de Sistemas y Automática, RoboticsLab
Universidad Carlos III de Madrid, ROR: 03ths8210
Avenida de la Universidad, 30 (edificio Sabatini), 28911, Leganés (Madrid), España
[1]{blukawsk*, eona, jcgvicto, balaguer, ajardon}@ing.uc3m.es
[2]{daniel.g.cubillo}@alumnos.uc3m.es

***Abstract*—This paper presents an open-source, end-to-end pipeline that automates robot sketching on an ABB CRB 15000-5 "GoFa" cobot from user-supplied images. The software integrates ROS 2 with ABB's Externally Guided Motion (EGM) interface and features a graphical Python application for image acquisition, edge processing (via classical Canny edge detection with OpenCV, local AI models, or cloud APIs), vectorization, trajectory generation, and real-time execution preview. We provide an extensive empirical evaluation comparing Deep Reinforcement Learning (DQN, PPO, SAC) against a deterministic geometric approach. Results demonstrate that while discrete DQN achieves a high in-distribution similarity (Jaccard index of 0.601), it degrades significantly out-of-distribution. In contrast, the geometric method generalizes reliably across unseen targets and reaches a 149-times reduction in total sketch processing time (21.8 ms versus 3248 ms). The key contributions are an open-source ROS 2 driver interfacing EGM, a user-friendly GUI workflow, and a comparative benchmark of RL versus geometric trajectory generation for robot sketching.**

***Index Terms*—ABB Externally Guided Motion, Collaborative robot, Image processing, Robot sketching, ROS 2**

I. Introduction

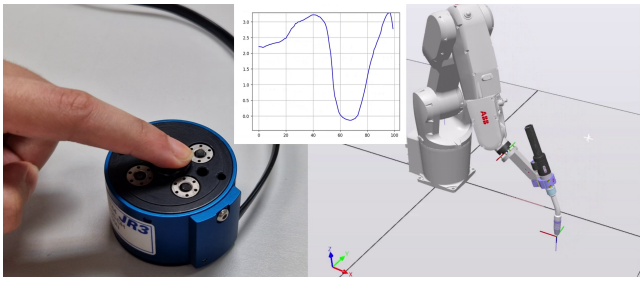
Collaborative robots (i.e., "cobots") helped breach the gap between the factory and research laboratories, allowing to introduce industry-grade manipulators in broader contexts such as academia. Their inherent safety and distinctive programming characteristics (such as kinesthetic teaching) create new learning opportunities. It has been proven that hands-on programming and handling of cobots helped increase student satisfaction and usability in laboratory sessions [1].

Considering these advantages over traditional robots, novel high-level applications can be explored by bridging vendor-specific tooling (such as ABB's EGM feature) and well-established community-maintained software stacks (such as ROS 2) leveraged by flexible programming languages (such as Python). The authors have expanded their previous experience with ABB cobots to propose a fully automated drawing task in a user-friendly fashion through graphical interfaces: a picture is to be taken (e.g. via webcam or mobile phone), processed to the liking of the user, and reproduced by the real robot. This work also covers a comparative study on selected reinforcement learning algorithms, discussing their suitability for generic scenarios beyond their training base.

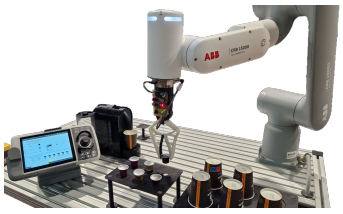
II. Background

This work depends or expands on previous developments by the RoboticsLab UC3M team; some have been depicted in Figure 1. Most notably, the Externally Guided Motion (EGM) feature of ABB controllers has reached a mature adoption status in said team following the introduction of a collection of high-level applications and use cases [2]. For instance, a path correction controller relying on force-torque sensor input was integrated with EGM and a custom embedded application, bridging individual modules via ROS 2 nodes (1a) [3].

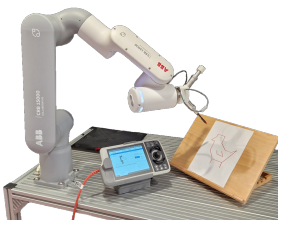
On the other hand, there is also a focus on expanding the range of user-facing applications to boost student engagement in graduate and postgraduate studies. Demonstrators help expose our results, and pose an opportunity for integration in practical sessions: a coffee-making station was built with that purpose in mind, showcasing new techniques for building graphical applications on the robot's teach pendant and on mobile devices (1b) [4]. Our studies show that even simple robot drawing experiments are well received by students (1c) [1].



(a) Path correction using live sensor data and EGM-based control [2], [3].



(b) Coffee-making demonstrator [4].



(c) Drawing application [1].

Fig. 1: Relevant previous works by RoboticsLab UC3M.

III. Materials

The hardware setup, both real and simulated in ABB RobotStudio, is depicted in Figure 2. The robot platform is the ABB CRB 15000-5 "GoFa" collaborative manipulator arm having six axes and a custom pen tool attached to its end effector, optionally equipped with a six-axis force-torque sensor to apply corrections to the axial displacement in real time while exerting a constant, preconfigured pressure on the paper. The robot is placed on a table and in front of an easel- or lectern-like object to which a paper sheet is attached. Inherent safety measures are in place throughout the task thanks to the integrated safety controller; in consequence, several kinematic and dynamic parameters (such as torques exerted on each joint) are monitored in runtime to automatically act upon collisions or excess force applied onto the environment.



(a) Real robot (b) RobotStudio model

Fig. 2: ABB CRB 15000-5 "GoFa" with a custom pen tool.

IV. Architecture

The proposed system architecture relies on the ROS 2 middleware to enable a network of distributed components leveraging hardware and software resources [5]. Figure 3 depicts said architecture including three main computing actors: the robot controller provided by the vendor (ABB); a PC communicating with the latter either remotely via Ethernet or locally through a virtual controller in RobotStudio; and a user PC or mobile device that interfaces with the previous through a graphical application. Lastly, live or static frames are grabbed from a camera device to initiate the pipeline's process.

Communication with the robot is handled by the Externally Guided Motion (EGM) feature, allowing to send and process high-frequency motion commands (250 Hz) via UDP packets. A custom ROS 2 package was developed to enable and facilitate the communication between the PCs and the robot.[1]

V. Discussion on Algorithm Candidates

Existing solutions have been considered for the sketching process. Previous works by the group also include a deep Q-learning neural network (DQN), adopting a reinforcement learning (RL) technique to enable the TEO humanoid to perform art drawings via policy-learned motions [6]. This method is contrasted in this section with other well-established RL algorithms: Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC).

[1]https://github.com/roboticslab-uc3m/abb_egm_driver

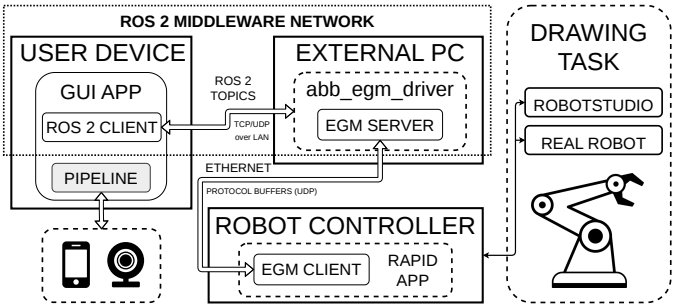


Fig. 3: Diagram of the proposed architecture.

A. Reinforcement Learning: SAC vs. PPO vs. DQN

Model evaluation was conducted on the *Quick, Draw!* dataset, a collection of 14 400 training and 3 600 test images (84×84 pixels) representing 13 semantic categories of sketches [7], [8]. Samples of 12 sketches of the test set were chosen randomly with a fixed seed to ensure uniformity. Each episode consists of 150 decision steps. Selected metrics include the mean square error (MSE), the Hausdorff distance, the Chamfer distance, the Jaccard index (IoU), a specificity ratio introduced to assess how strong the dependency of the policy is on the input (values lower than the unit imply a weak dependency), kinematic smoothness (as a measure of the mean curvature and jerk), compute time and success rate. A null policy (meaning that nothing is drawn) is defined to detect whether an agent has actually learned.

The results have been summarized in Table I. It is important to note that SAC produced no valid drawing. This off-policy algorithm optimizes the expected reward and the policy entropy simultaneously, using the temperature parameter α to adjust their weights accordingly. On a training phase of 500 000 environment steps and 2667 episodes, Figures 4 and 5 show a tendency to reducing the return while increasing the mean similarity. However, Table II shows that the temperature saturates at 180 000 training steps, dropping the dispersion terms σ_{dx} and σ_{pen}; thus, the policy becomes deterministic and the algorithm no longer tries to press the pen against the paper, effectively drawing nothing. There was a later attempt to automatically re-adjust, although the saturation of the outputs canceled the effect of the temperature increase.

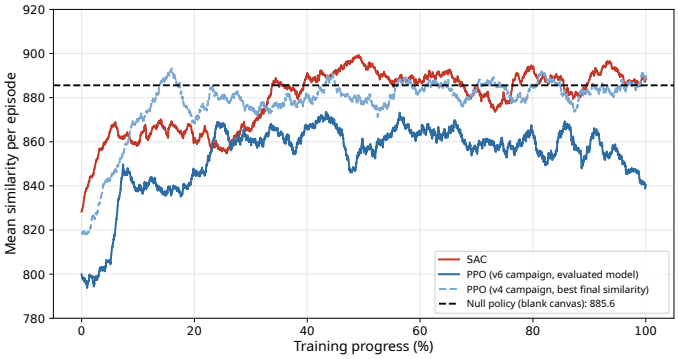


Fig. 4: Evolution of mean similarity in SAC and PPO (moving average of 101 episodes, scale $\lambda_{sim} = 1000$).

TABLE I: Comparison of RL agents evaluated on the *Quick, Draw!* dataset (sample: $n = 12$ sketches). SAC was omitted.

Concept	**DQN (discrete)**	**PPO (continuous)**
Action space	242 indexes	$[-1, 1]^3$
Distinct actions per ep.	17.8 of 242	—
Action σ	—	(0.098; 0.094; 0.006)
Steps with pen on paper	86.7%	99.3%
Pen-ups	$\mathbf{10.75 \pm 7.54}$	1
Stroke length (px)	$\mathbf{890.97 \pm 60.89}$	58.9 ± 47.6
Specificity ratio	**5.25**	0.08
Jaccard index (IoU) ↑	$\mathbf{0.6013 \pm 0.0377}$	0.0085 ± 0.0048
Chamfer distance (px) ↓	$\mathbf{0.571 \pm 0.445}$	11.60 ± 3.23
Hausdorff distance (px) ↓	$\mathbf{11.00 \pm 7.17}$	40.45 ± 3.62
Precision ↑	**0.7995**	0.3555
Completeness ↑	**0.7129**	0.0087
Similarity	**942.82**	884.56
Latency per step (ms)	21.79	26.49
Time per sketch (ms)	3248 ± 115	3986 ± 110
Success (Chamfer < 3 px)	**100.0%**	0.0%

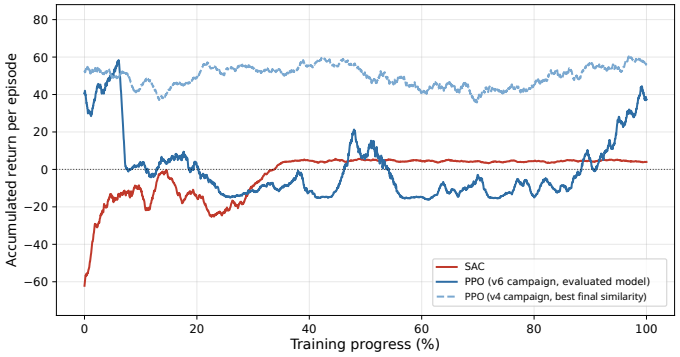


Fig. 5: Evolution of episodic return in SAC and PPO during training (moving average of 51 episodes). Scale not comparable, each implementation weighs reward terms differently.

TABLE II: Evolution of SAC entropy collapse (training).

Steps	Temperature α	σ_{dx}	σ_{pen}	pen on paper
20 000	0.127	0.115	0.076	96.7%
100 000	0.056	0.136	0.070	0.0%
180 000	0.010	0.054	0.002	0.0%
300 000	0.010	0.033	0.001	0.0%
420 000	0.111	0.000	0.000	99.3%
500 000	0.050	0.000	0.000	0.0%

In contrast to SAC, PPO did succeed in performing a drawing. Six versions were iteratively developed, reaching a high similarity ratio. However, it presents a reduced total stroke length, resulting in incomplete sketches; its low specificity of 0.08 supports this, revealing that each stroke resembles twelve times more closely the strokes generated for other sketches other than the one currently being drawn.

Lastly, DQN has a discrete space of 242 actions (11×11, that is, 121 motions, and two tool states: drawing or pen-up). It yields better results than continuous PPO and SAC algorithms, also improving upon the trivial null policy. Its high specificity ratio of 5.25 reveals that each stroke has a five times stronger resemblance to its own goal than to those previously generated.

B. DQN vs. purely geometric method

A non-RL, purely deterministic geometric method is compared against DQN. It consists of four processing stages: 1. Gaussian blur and Canny edge detection; 2. contouring and area filtering for noise reduction; 3. contour simplification via the Ramer-Douglas-Peucker algorithm [9], [10]; 4. segment discretization based on arc lengths to ensure constant waypoint spacing. The results have been summarized in Table III.

TABLE III: Comparing DQN against our geometric method.

Metric	DQN (best RL)	Geometric method
Chamfer distance (px) ↓	$\mathbf{0.571 \pm 0.445}$	0.898 ± 0.132
Hausdorff distance (px) ↓	11.00 ± 7.17	$\mathbf{4.65 \pm 1.12}$
HD_{95} (px) ↓	6.08 ± 6.80	$\mathbf{2.87 \pm 0.77}$
Jaccard index (IoU) ↑	**0.6013**	0.1857
Precision ↑	**0.7995**	0.5001
Completeness ↑	**0.7129**	0.2296
Stroke length (px)	890.97 ± 60.89	969.77 ± 345.78
Specificity ratio	**5.25**	3.42
Mean quadratic jerk ↓	$\mathbf{0.529 \pm 0.051}$	0.670 ± 0.062
Mean curvature (px^{-1})	0.349	**0.317**
Mean angular variation (°)	**12.73**	17.23
Time per sketch (ms) ↓	3248 ± 115	$\mathbf{21.8 \pm 6.7}$
Latency per step (ms)	21.79	—
Generated waypoints	151 (fixed)	1170 ± 389
Pen-ups	10.75 ± 7.54	18.00 ± 10.13
Success (Chamfer < 3 px) ↑	**100.0%**	**100.0%**

While the results are comparable, the geometric method is selected due to the great reduction in processing time (149 times). This is caused by the RL technique being autoregressive, i.e., it must evaluate the neural network once per decision step having a latency of 21.79 ms per inference; this cannot be parallelized since each state depends on the previous one. Moreover, trajectories generated through the geometric method can be visually inspected by the user and validated prior to sending them to the controller. To summarize, Figure 6 depicts a qualitative comparison of all methods (excluding SAC).

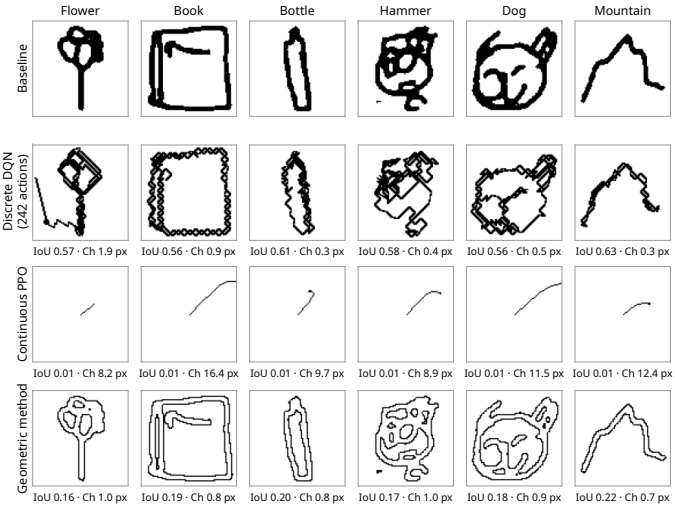


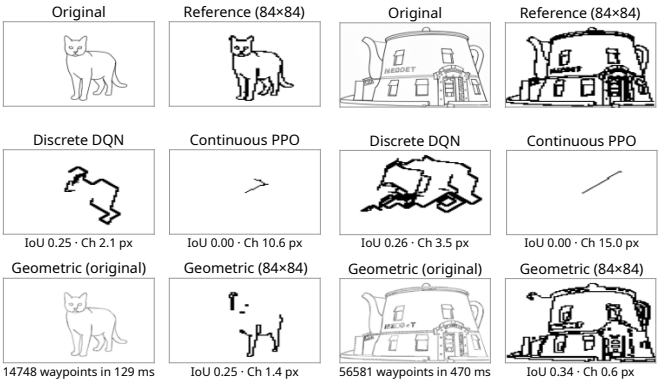
Fig. 6: Comparison across different test categories.

C. Generalization

After validating the aforementioned methods against the sketches they have been trained against, it is interesting to analyze the behavior in the out-of-distribution scenario in order to assess their capability of generalization. Two test images have been selected: a cat and a teahouse. The differences are summarized in Table IV and depicted in Figures 7 and 8.

TABLE IV: Quantitative metrics of all methods in-distribution (QD!) and out-of-distribution (cat, TH – teahouse).

Metric	Set	Method			
		DQN	PPO	SAC	Geometric
IoU ↑	QD!	**0.601**	0.008	0.000	0.186
	cat	0.250	0.002	0.000	0.248
	TH	0.255	0.002	0.000	**0.344**
Chamfer (px) ↓	QD!	**0.571**	11.60	N/A	0.898
	cat	2.06	10.62	N/A	1.38
	TH	3.49	14.98	N/A	**0.64**
HD_{95} (px) ↓	QD!	**6.08**	35.69	N/A	2.87
	cat	12.94	32.57	N/A	7.52
	TH	21.02	46.62	N/A	**2.83**
Coverage	QD!	0.144	0.005	0.000	0.074
	cat	0.040	0.003	0.000	0.030
	TH	0.127	0.003	0.000	0.232
Null policy Δ ↑	QD!	**+56.36**	−1.89	+0.00	+0.05
	cat	+0.14	−2.69	+0.00	**+5.81**
	TH	+22.11	−2.41	+0.00	**+9.35**



(a) Cat: 1024×688 px, 10 contours (b) TH: 864×580 px, 143 contours

Fig. 7: Out-of-distribution sketching results (TH – teahouse).

The results show that DQN no longer yields high similarity values ("null policy Δ") in an out-of-distribution scenario. Similarly, its Chamfer distance and Jaccard index degrade if compared against an in-distribution test. The DQN outputs shown in Figure 7 suggest that the algorithm is unable to correctly follow the contour of the represented object. The deterministic, purely geometric method is the only one devoid of degradation. On a final note, whereas RL algorithms have a fixed waypoint threshold, this is the only approach that can scale up to the original resolution (at the cost of greatly increasing the number of waypoints) and restore all details lost at the lower 84×84 dataset image size.

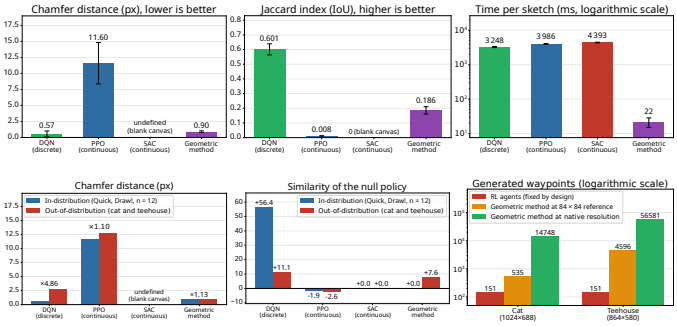


Fig. 8: Relevant differences across all methods.

VI. Implementation

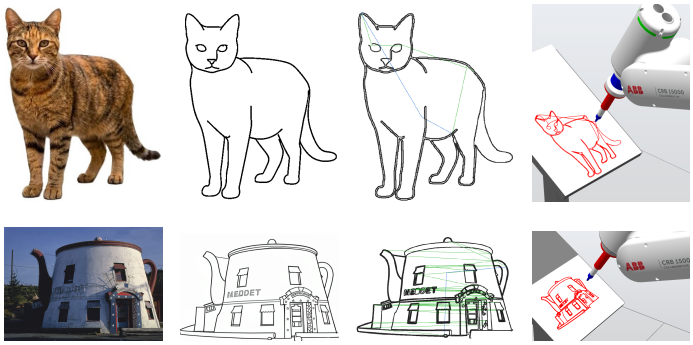
A. Proposed pipeline

An end-to-end workflow was implemented in a single application to cover all necessary phases, from the image acquisition through sketch generation to robot motion execution, and integrated within a ROS 2 ecosystem for flexibility and abstraction: the EGM feature is leveraged and wrapped in a ROS node to deal with any ABB-compatible robots while adhering to standard ROS messages (*geometry_msgs/Pose*). A pipeline is introduced in said workflow to streamline parts of the algorithm that do not require or would not particularly benefit from further modularization in the usual ROS-driven fashion. The pipeline is organized as a sequence of steps:

1) Image acquisition either via a live USB camera stream or from an image saved on disk. An optional background removal option is presented to the user, applying the GrabCut algorithm [11].
2) Conversion of the base image to a simplified black and white sketch representation using simple traces (lines, arcs). Several algorithms are to be chosen from following either a local (on-disk) or remote pipeline (via server connection and API token). Modern LLM-based models are leveraged in order to produce high quality results, or to assist the user in regenerating them on-demand. This step includes adaptive histogram equalization via CLAHE [12], thresholding and noise filtering.
3) Vectorization of the resulting sketch in the SVG format. A Canny edge detection step is run on the PNG sketch, contours are extracted with OpenCV's `findContours` function. An area filter is applied to discard high-frequency noise, and the Douglas-Peucker algorithm is used to simplify the contours.
4) Conversion of the vectorized image to the robot space using an external open source library.[2] Paths are discretized and simplified with the provided resolution. Pen up/down motions are interwoven where necessary, and corners are smoothened.
5) Execution of the drawing task through EGM commands issued and sent by means of ROS 2-based messages. A dry-run mode is provided to check the trajectory's feasibility beforehand.

[2]https://github.com/regebro/svg.path

Figure 9 illustrates some of these steps. Figure 9a represents a free stock image to be represented by the robot. Figure 9b depicts the resulting simplified monochrome sketch obtained through an edge-detection algorithm. Any outliers have been filtered out to maximize longer, continuous lines. Figure 9c shows the result of the trajectory generation as a collection of points to be reached by the TCP during the task. Green lines represent transitions between paths during which the pen tool has been lifted. Blue lines correspond to transitions at the beginning and at the end of the task. Finally, Figure 9d portrays the simulated robot in the RobotStudio environment executing the programmed motions.



(a) Original image (b) Resulting sketch (c) Path generation (d) Execution

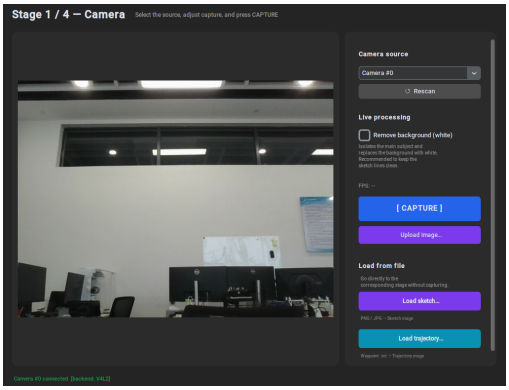
Fig. 9: Example data processing pipeline resulting in the trajectory generation and execution on the simulated robot.

B. Graphical interface

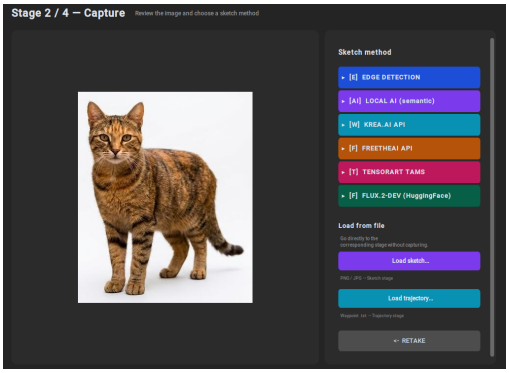
The application is accompanied by a graphical interface to aid the user throughout the aforementioned steps, allowing for algorithm selection, parameter configuration and inspection, and live supervision of the executed task.

This graphical user interface (GUI) is depicted in Figure 10. Figure 10a illustrates the main menu with a preview of the target image loaded from on disk. Alternatively, live camera stream would be displayed in a previous step enabling to grab a frame thereof; the interface detects all available USB cameras and features a selection menu to pick the desired one. On the right side, several options are presented to choose the preferred algorithm; among them, a classical OpenCV-based Canny edge detection, and several LLM-based services relying on a public API and a token to be supplied through the interface. It is also possible to run a local AI model. A prepared sketch can be loaded from disk to override this step; similarly, a text file can be read from to also skip the next phase and load a trajectory to be commanded to the robot.

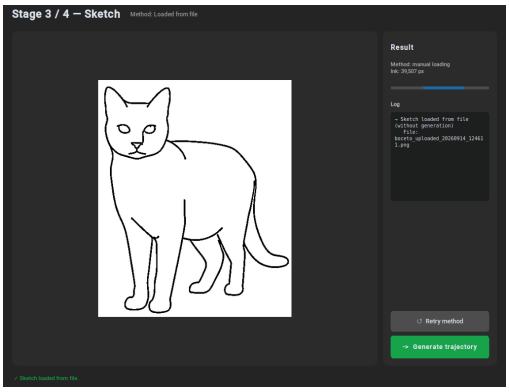
Figure 10d illustrates the last step of the pipeline before sending any motion commands. At this stage, motion and trajectory parameters can be adjusted, such as: dimensions of the resulting sketch, pen-up height, and linear speed of the pen's tip. A summary is presented including additional information such as estimated execution time. Prior to sending EGM commands to the robot, the user is given the option to run a preview of the task (Figure 9c).



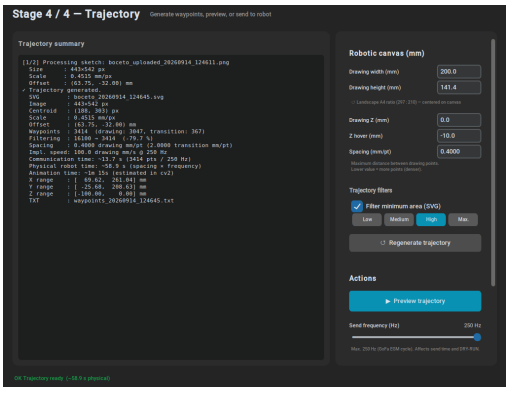
(a) Camera frame acquisition or image loaded from disk



(b) Image preview and sketching algorithm selection step



(c) Sketch preview screen



(d) Selection and summary of the trajectory parameters

Fig. 10: Graphical user interface (GUI) of the application.



Fig. 11: Drawing task executed by the simulated robot controller using the trajectory generated by the proposed application.

VII. Experiments

Experiments have been carried out on the real robot following validation on the RobotStudio simulator. Coordinate systems have been adjusted using ABB's concept of "work objects" so that the same paths are referred to a configurable frame defined at one corner of the paper, regardless of the environment (real or simulated). An initial point is established above the center of the paper sheet, which is also the point at which the trajectory ends.

Figure 11 shows three consecutive snapshots during an execution of the algorithm on the sample cat image displayed in Figure 9a. It has been noted that the intrinsic smoothing factor at the ROS node side needs to be significantly reduced or removed altogether to prevent the built-in low-pass filter from influencing the result.

VIII. Conclusions

In this work, an end-to-end robot sketching pipeline was implemented and validated on a real robot setup. Vendor-specific (ABB's EGM) and community-backed open source (ROS 2) tools have been leveraged to introduce a Python graphical application that assists the user during the process. Reinforcement learning techniques have been studied and compared against a geometric method, which has been ultimately selected for the final implementation. This work can be suited for demonstrating an use case of collaborative robotics both in technical stands and in the classroom. Students can learn how to integrate popular tools with a robot to fulfill an engaging task: what you photograph is what you get.

Future work includes support for mobile devices through either native or web applications. For increased usability, the user could be able to initiate, control and monitor the whole process on a modern mobile phone, only requiring an external PC to act as the server machine interfacing with the robot through Ethernet, and with the phone through a local wireless network. On a side note, the drawing task itself would benefit from the introduction of an external force-torque sensor in order to adjust the pressure applied by the pen tool on the paper in real time [3].

The code of the application has been open-sourced and published on a GitHub repository.[3]

[3]https://github.com/roboticslab-uc3m/gofa-sketching

Acknowledgments

This research has been financed by "iRoboCity2030-CM, Robótica Inteligente para Ciudades Sostenibles" (TEC-2024/TEC-62) and "FotoArt5.0-CM, Laboratorios inteligentes para la ciencia del futuro" (TEC-2024/TEC-308), both funded by "Programas de Actividades I+D en Tecnologías de la Comunidad de Madrid"; "ROBOASSET: Intelligent robotic systems for assessment and rehabilitation in upper limb therapies" (PID2020-113508RB-I00, financed by AEI/10.13039/501100011033); "iREHAB: AI-powered Robotic Personalized Rehabilitation" (DTS22/00105, financed by Instituto de Salud Carlos III (ISCIII)); and EU structural funds.

References

[1] B. Łukawski, E. D. Oña, J. G. Victores, C. Balaguer, and A. Jardón, "Assessing student satisfaction and system usability in industrial robotics education," Simposios del Comité Español de Automática, CEA, 2(1), 2026. DOI: 10.64117/simposioscea.v2i1.150

[2] B. Łukawski, E. D. Oña, A. Jardón, J. G. Victores, and C. Balaguer, "Development of educational applications with ABB GoFa collaborative robot using Externally Guided Motion," Int. Conf. on Control, Automation and Diagnosis (ICCAD), IEEE, 2025. DOI: 10.1109/ICCAD64771.2025.11099395

[3] B. Łukawski, A. Olano Díaz, A. González Villasante, E. D. Oña, J. G. Victores, and A. Jardón, "External force-torque sensor feedback on the ABB GoFa collaborative robot," Simposio de Robótica, Bioingeniería, Visión Artificial y Automática Marina, CEA, 2(2), 2026. DOI: 10.64117/simposioscea.v2i2.215

[4] B. Łukawski, M. Martín-Cervera, C. Menchén-Pascual, E. D. Oña, J. G. Victores, and A. Jardón, "Espresso macchiato, por favore: collaborative robotic coffee-making for education," Jornadas de Automática, CEA, 46, 2025. DOI: 10.17979/ja-cea.2025.46.12274

[5] M. Quigley et al., "ROS: an open-source Robot Operating System," ICRA Workshop on Open Source Software, 3, 2009. Corpus ID: 6324125

[6] R. Fernandez-Fernandez, J. G. Victores, and C. Balaguer, "Deep robot sketching: an application of deep Q-learning networks for human-like sketching," Cognitive Systems Research, Elsevier, 81, 2023, pp. 57–63. DOI: 10.1016/j.cogsys.2023.05.004

[7] R. Fernandez-Fernandez, J. G. Victores, D. Estevez, and C. Balaguer, "Quick, Stat!: A statistical analysis of the Quick, Draw! dataset," EUROSIM, 2019. DOI: 10.48550/arXiv.1907.06417

[8] D. Ha, and D. Eck, "A neural representation of sketch drawings," ArXiv pre-print, 2017. DOI: 10.48550/arXiv.1704.03477

[9] D. H. Douglas, and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," The Canadian Cartographer, 10, 1973, pp. 112–122. DOI: 10.3138/FM57-6770-U75U-7727

[10] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," Computer Graphics and Image Processing, 1(3), 1972, pp. 244–256. DOI: 10.1016/S0146-664X(72)80017-0

[11] C. Rother, V. Kolmogorov, and A. Blake, "'GrabCut': Interactive foreground extraction using iterated graph cuts," ACM Transactions on Graphics, 23(3), 2004, pp. 309–314. DOI: 10.1145/1015706.1015720

[12] K. Zuiderveld, "Contrast limited adaptive histogram equalization," Graphics Gems, Academic Press Professional, 1994, pp. 474–485. DOI: 10.1016/B978-0-12-336156-1.50061-6