RoboFlow: a Data-centric Workflow Management System for Developing AI-enhanced Robots

Anonymous Author(s)
Affiliation
Address
email

Abstract: We propose RoboFlow, a cloud-based workflow management system orchestrating the pipelines of developing AI-enhanced robots. Unlike most traditional robotic development processes that are essentially process-centric, RoboFlow is data-centric. This striking property makes it especially suitable for developing AI-enhanced robots in which data play a central role. More specifically, RoboFlow models the whole robotic development process into 4 building modules (1. data processing, 2. algorithmic development, 3. back testing and 4. application adaptation) interacting with a centralized data engine. All these building modules are containerized and orchestrated under a unified interfacing framework. Such an architectural design greatly increases the maintainability and re-usability of all the building modules and enables us to develop them in a fully parallel fashion. To demonstrate the efficacy of the developed system, we exploit it to develop two prototype systems named "Egomobility" and "Egoplan". Egomobility provides general-purpose navigation functionalities for a wide variety of mobile robots and Egoplan solves path planning problems in high dimensional continuous state and action spaces for robot arms. Our result shows that RoboFlow can significantly streamline the whole development lifecycle and the same workflow is applicable to numerous intelligent robotic applications.

Keywords: AI-enhanced robots, robotic development workflow management, data-centric development, cloud-based robotic development

1 Introduction

We propose RoboFlow, a cloud-based workflow management system for developing data-centric and AI-enhanced robots. This work is done in the context that significant progresses have been made in robotics development and a paradigm shift from process-centric development to data-centric development is being witnessed, especially for learning robots. Specifically, traditional robot development workflow [1, 2, 3, 4, 5, 6] is essentially process-centric, which emphasizes more on designing, developing and integrating different "processing modules" interacting and inter-operating with each other in a complex fashion. Such a process-centric robotic development model, though natural for humans, is not suitable for developing modern AI-enhanced robotic systems (aka., learning robots) that are essentially data-centric [7, 8, 9, 10, 11, 12]. Some key reasons are that the development processes of AI-enhanced robots generally involve managing and interacting with massive amounts of data, and even after the systems have been deployed, continuous modification and improvement are still needed when more data get acquired. Such an extra "data-centric" dimension of learning robots causes a dramatic increase in both coding complexity and maintaining complexity of the traditional process-centric robotic development workflow, thus a new data-centric robotic development workflow is crucially is needed. To bridge this gap, we propose RoboFlow.

A high-level overview of RoboFlow is illustrated in Figure 1. In the most abstract fashion, the RoboFlow system divides the whole pipeline of developing AI-enhanced robots into 4 building modules (1. data processing, 2. algorithmic development, 3. back testing and 4. application adaptation) interacting with a centralized data engine. Specifically, the data engine can be viewed as an "oracle" that abstracts out all the data management details and is interacting (e.g., being queried or manipulated) with all the 4 building modules in an asynchronous fashion. Centered around the

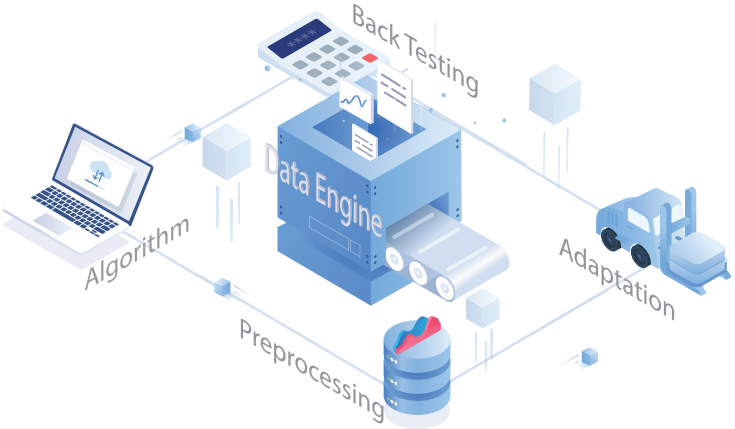


Figure 1: An overview of RoboFlow. The robot development pipeline consists of 4 modules interacting with a centralized data engine. The data engine manages the large-scale dataset, and publishes data from robot. The data preprocess module encodes raw input to data that algorithm development module can easily parse. The algorithms development module develops customized control policy. The back testing module tests the policy in various environments. The application adaption module deploys the learned policy in real world.

data engine, the 4 building modules follow an iterative spiral model. Unlike the classical spiral model[13] for software development, these building blocks mainly interact with data engine, thus can be developed in a fully parallel fashion and each module could have different "versions". Such a data-centric design dramatically decreases the developing and maintaining complexities, making it especially suitable for developing AI-enhanced robots. The framework in Figure 1 is quite generic and described in a fully abstract way. To understand this framework better, we put it in a concrete context of developing learning robots. From a learning perspective, the data processing module transforms and encodes data into a state that other modules can easily parse. The algorithm module represents the planning policy of the robots, which is in charge of the robots' actions. The back testing module evaluates the policy of the learning robot based on large-scale collected or simulated data. The application adaptation module puts the developed policy into real-world environments once the back testing meets a designed criterion. More details about these modules will be provided in Section 3.

This paper has 3 major contributions: (i) we proposed a novel data-centric robot development model which improves upon the traditional routines in terms of development flexibility and maintainability. (ii) We implement a first prototype system using containerization techniques (specifically, Docker[14] and Kubernetes [15]). (iii) Using this system, we develop two platforms: Egomobility and Egoplan. Egomobility exploits deep reinforcement learning to provide navigation ability for turtlebot and Egoplan uses imitation learning to solve complex motion planning problems for sawyer robots. These studies demonstrate the efficacy of RoboFlow in various intelligent robotics applications. The rest of this paper is organized as below. Section 2 introduces related work. Section 3 describes implementation of RoboFlow. Section 4 presents two platforms to showcase the usefulness of RoboFlow.

2 Related Work

The two most relevant lines of work related to RoboFlow are the workflow management system and data management engine for data-centric robot systems. For the workflow management system, earlier robot development pipelines try to simplify the workflow of robotics development and reduce re-programming efforts. For example, Fetch robotics[16] launches a Workflow Builder which allows customers to design, implement, and redesign their own workflows. But their available tools are not designed for large-scale dataset processing. In the machine learning field, frameworks like Kubeflow[17] and MLFlow[18] have been developed to manage the workflow of model development but they are not designed for robot development. For the data management engine, researchers in bioinformatics community developed system tools to manage data workflows in an end-to-end fashion[19, 20]. Nevertheless, in robotics, few works emphasize on the data-centric aspect of the development processes except some proposals related to cyber-physical systems which view robots as data-gathering nodes [21, 22, 23, 24, 25, 26, 21]. In recent work, Farzad et al. [27] proposed a cloud framework aiming to facilitate the development of IoT applications. but it is not straightforward on how to apply it to the more heavy-weighted data-centric robotic systems.

3 System Architecture and Software Implementation

The RoboFlow architecture builds upon the containerization and container orchestration techniques. More specifically, a container platform (e.g., the Docker) packages applications so that they can access a specific set of resources on a physical or virtual host. The main benefit, especially for developers, is that containers isolate different applications and are elastic, i.e., come and go as demanded by need. This is particularly useful for developing massive robotic systems in which developers may contribute code in different programming languages and application frameworks. In such scenarios, we could exploit a container platform to establish many containers to isolate and manage all the developed applications. To manage containers at scale, we can utilize a container orchestrate system (e.g., Kubernetes, Docker Swarm) to automate the deployment, management, scaling, networking, and availability of all the containers. In the rest of this section, we describe the system architecture and software implementation of the RoboFlow system.

System Architecture. Figure 2 illustrates the system architecture of RoboFlow. It has 4 essential modules(1.Data Preprocess 2.Algorithm Development 3.Back Testing 4.Application Adaptation) interacting with a centralized data engine. Each module is employed into a containerized environment by bundling it together with all related configuration files, libraries and dependencies. These module containers run isolated processes on the system, thus enabling RoboFlow to be developed in a fully parallel fashion. In addition, any change in these module containers is recorded, making version control easy to implement. These containers exploit DDS (Data Distribution Service) to manage real-time communication between them and the data engine. DDS implements a publish-subscribe pattern for sending and receiving data and each process running in RoboFlow are considered as DDS nodes to communicate data with other process. Also, a networked filesystem named Glusterfs is also ultilized for sharing large-scale files (e.g., large neural network models or training datasets) between modules. Such integration of DDS and Glusterfs make RoboFlow suitable for developing data-driven methods on robotics. RoboFlow also provides a web-based frontend to ease developers to monitor, analyze, and manage the robotic development process.

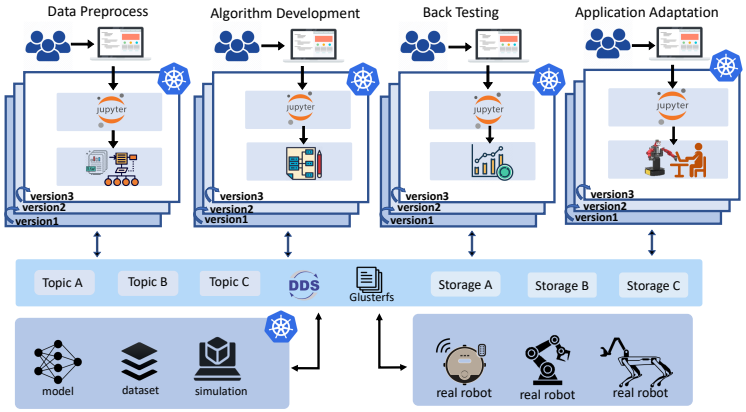
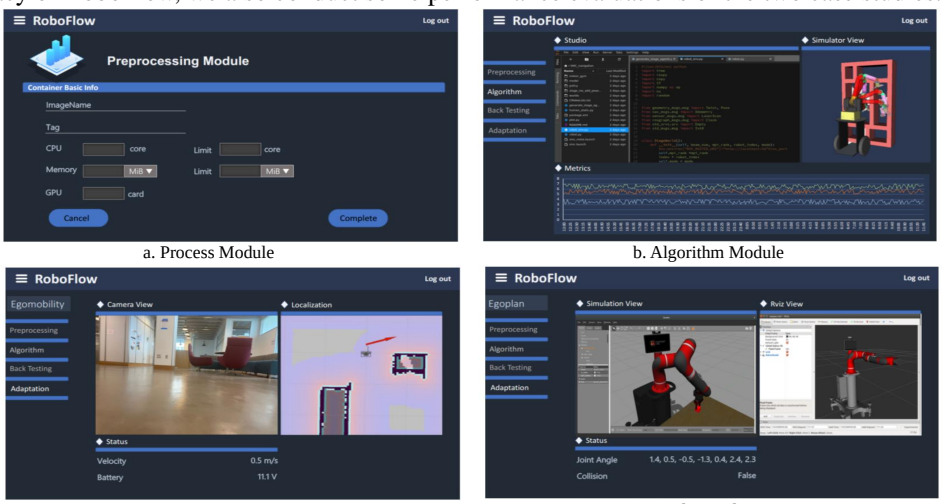


Figure 2: The architecture of RoboFlow. RoboFlow provides a graphical user interface for developers to access all containerized modules. These modules interact with the data engine through shared storage and DDS topics. The data engine consists of large-scale datasets and model data stored in the shared folder, simulation containers and robots connected to the RoboFlow system through DDS topics.

Software Implementation. To implement RoboFlow, we wrap each module as a `docker` image. For this, we specify the software environment (e.g., the operating system distribution and pre-installed packages) of an image in a `Dockerfile`. The obtained module images are installed with `juypter lab` and ROS2. Some modules may need additional packages. For example, the algorithm development module is equipped with the deep learning libraries `tensorflow` and `pytorch`, while the data engine image utilizes several robot simulators like `Gazebo`, `Stage` and `OpenRave`.The successfully built images are stored in a cloud-based storage space named `cargo`, which can be viewed as on-premise `dockerhub` of the RoboFlow system. During the run time, each instance of a built image is deployed as a container and RoboFlow utilizes `Kubernetes` to deploy and manage these containers. We exploit the `React` javascript library to implement a graphical user interface (GUI). Through this GUI users can choose desired versions of the docker images and allocate computational resources (e.g., CPUs or GPUs) to the container being created. Once such configuration information is submitted, it is turned into a `YAML` file which will be serialized and deployed by `Kubernetes`.

4 Two Case Studies and Performance Evalaution

In this section, we exploit RoboFlow to develop two prototype systems named "Egomobility" and "Egoplan". Egomobility provides a general-purpose navigation platform for managing a wide variety of mobile robots and Egoplan is a motion planning platform for robot arms. To demonstrate the efficacy of RoboFlow, we also conduct some performance evaluations of the two case studies.



a. Process Module b. Algorithm Module

c. Egomobility d. Egoplan

Figure 3: Graphical User interface of RoboFlow. Figure a illustrates the edit page of preprocess module, which allows developers to specify the image name, version, required CPU number, memory size, and GPU number. Figure b illustrates the algorithm module, enabling users to access the code editor and simulator view, monitor the progress of a learning robot training process. Figure c illustrates the Egomobility platform. This adaptation page provides real-time camera stream and current position, velocity and battery usage. Figure d illustrate the Egoplan, which provides the simulation views, rviz view and status panel.

Case Study 1: The Egomobility platform for Mobile Robots. In this study, we exploit RoboFlow to develop a mobile robot navigation platform named Egomobility, which is a data-centric AI-enhanced robot system using data from a `Stage` simulator [28] as training data. Within this environment, a mobile robot takes 3 raw laser frames and its velocity as input. Our goal is to train a reinforcement learning policy that outputs a velocity guiding the robot avoiding dynamic obstacles. The deployment page of Egomobility is provided in Figure 3.

Case Study 2: The Egoplan Platform for Arm Robots. In this study, we exploit RoboFlow to develop a robot arm motion planning Platform named EgoPlan. More specifically, we exploit the NEXT (Neural Exploration-Exploitation Trees) algorithm proposed in[29] to learn a motion plan policy for solving path planning problems in 7-dimensional state space and 7-dimensional action space. In each planning task, we simulate a robot arm and a shelf in the `openrave` [30] simulator in our data engine. Each level of the shelf is horizontally divided into multiple bins. The task for the robot arm is to plan a collision-free path from an initial pose to grab an object placed in a shelf. The main strategy is to exploit BIT* [31] to solve the planning problem in a brutal force way but collect the data to learn a smarter motion planning policy.

Performance Evaluation. Through the above two case studies, the benefit of RoboFlow for managing a data-centric robotic system development pipeline is quite obvious: RoboFlow enables developers to develop all the component modules in a fully-parallel fashion. For example, in developing Egomobility, algorithm developers can join in the RoboFlow to develop the navigation algorithm independently without interfering with each other. The developers for the application adaptation module can simultaneously test many learned policies without the need of complex communication processes. For a more quantitative comparative analysis, we let the same team of developers to build the Egomobility and Egoplan platforms with and without using RoboFlow and document. During the development lifecyle, we carefully record different progress metrics. We observe at least 10 times improvement (3 months vs 1 week) in terms of development speed. One thing that needs to be careful is that the same team-first developed these two platforms without using RoboFlow and some significant experience has been acquired. However, even adjusted by this confounding factor, all the members in the team agree that at least 5 times more productivity should be achieved by using RoboFlow. In addition, due to the usage of sophisticated container-orchestration techniques, the resulting systems developed by RoboFlow is much more reliable and maintainable.

References

[1] R. Arumugam, V. R. Enti, L. Bingbing, W. Xiaojun, K. Baskaran, F. F. Kong, A. S. Kumar, K. D. Meng, and G. W. Kit. Davinci: A cloud computing framework for service robots. In *2010 IEEE international conference on robotics and automation*, pages 3084–3089. IEEE, 2010.

[2] R. Bouziane, L. S. Terrissa, S. Ayad, J.-F. Brethe, and O. Kazar. A web services based solution for the nao robot in cloud robotics environment. In *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 0809–0814. IEEE, 2017.

[3] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pages 287–296. IEEE, 2018.

[4] Y. Lei, Z. Fengyu, W. Yugang, Y. Xianfeng, Z. Yang, and C. Zhumin. Design of a cloud robotics visual platform. In *2016 Sixth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, pages 1039–1043. IEEE, 2016.

[5] S. Dennis, L. Alex, L. Matthias, and S. Christian. The smartmdsd toolchain: An integrated mdsd workflow and integrated development environment (ide) for robotics software. 2016.

[6] G. Huang, P. S. Rao, M.-H. Wu, X. Qian, S. Y. Nof, K. Ramani, and A. J. Quinn. Vipo: Spatial-visual programming with functions for robot-iot workflows. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[7] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

[8] S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

[9] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research*, 35 (11):1289–1307, 2016.

[10] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4238–4245. IEEE, 2018.

[11] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[12] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[13] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5): 61–72, 1988.

[14] Docker. https://www.docker.com/.

[15] Kubernetes. https://kubernetes.io/.

[16] F. Robotics. Fetch research robot.

[17] Kubeflow. https://www.kubeflow.org/.

[18] Mlflow. https://mlflow.org/.

[19] A. M. Comeau, G. M. Douglas, and M. G. Langille. Microbiome helper: a custom and streamlined workflow for microbiome research. *MSystems*, 2(1), 2017.

[20] T. W. Backman and T. Girke. systempiper: Ngs workflow and report generation environment. *BMC bioinformatics*, 17(1):388, 2016.

[21] P. Gil, I. Maza, A. Ollero, and P. Marrón. Data centric middleware for the integration of wireless sensor networks and mobile robots. In *proc. 7th Conference on Mobile Robots and Competitions, ROBOTICA*. Citeseer, 2007.

[22] K. Römer, O. Kasten, and F. Mattern. Middleware challenges for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):59–61, 2002.

[23] Y. Yu, B. Krishnamachari, and V. K. Prasanna. Issues in designing middleware for wireless sensor networks. *IEEE network*, 18(1):15–21, 2004.

[24] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, et al. Envirotrack: Towards an environmental computing paradigm for distributed sensor networks. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 582–589. IEEE, 2004.

[25] S. L. Remy and M. B. Blake. Distributed service-oriented robotics. *IEEE Internet Computing*, 15(2):70–74, 2011.

[26] N. Mohamed, J. Al-Jaroodi, and I. Jawhar. Middleware for robotics: A survey. In *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 736–742. Ieee, 2008.

[27] F. Khodadadi, R. N. Calheiros, and R. Buyya. A data-centric framework for development and deployment of internet of things applications in clouds. In *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6. IEEE, 2015.

[28] R. Vaughan. Massively multi-robot simulation in stage. *Swarm intelligence*, 2(2-4):189–208, 2008.

[29] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rJgJDAVKvB.

[30] R. Diankov and J. Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 79, 2008.

[31] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Bit*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. *arXiv preprint arXiv:1405.5848*, 2014.