2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

DOS®: A Deployment Operating System for Robots

Guo Ye[1] Qinjie Lin[1] Zening Luo[1] Han Liu[1]

***Abstract*— We propose a new system named DOS®(Deployment Operating System for Robots) for reliably deploying any data-driven robots in both production and simulation environments. Compared to existing systems, DOS® features a unique CI/CD (continuous integration and continuous deployment) architecture which allows us to seamlessly integrate agile development and reliable operation in a fully automated fashion. With this CI/CD architecture, this paper mainly introduces three essential components that uniquely differentiate DOS® from existing robotic systems: (i) An environment adapter that provides a systematic and robust approach to handle the deployment complexity in real world environments; (ii) A data replay reservoir that provides a unified data model supporting arbitrary robotic decision models; (iii) An analytical profiler that collects any set of user-defined performance metrics for system optimization. DOS® significantly increases the reliability and maintainability of the deployed robotic systems. To illustrate this point, we compare DOS® with more traditional approaches on deploying a navigational robot in a challenging working environment with many new corner case scenarios. Our results show that DOS® outperforms traditional approach in great magnitudes in terms of deployment time and operational robustness.**

I. INTRODUCTION

We propose DOS®, a system designed for reliably deploying modern AI-enhanced robotics. The system is lightweight but powerful to run on the cloud for simulation and on a real-world production level. In recent years, machine learning models are applied and run on the backend as a service every day on various domains, e.g., recommendation system, language translation and image recognition. Ubiquitous application leads to a large team with more developers involved. How to improve the efficiency of cooperation encourages lots of large cloud service like AWS, GCP and Azure to provide complicated toolchains to address large-scale deployment. A great many of pipeline orchestrators also develop tools for CI/CD for machine learning applications. On robotic area, there are platforms [1]–[6] researchers designed for managing and interacting with deep learning models. The key problem here is that components of these heterogeneous platforms are deeply bound to cloud services and their architect. It also leads to the relatively heavy configurations and learning cost for a new application. This motivates us to develop a lightweight but fully functional system specially designed for robotic deployment. With the clear division of tasks and modular design, the system can help the team to minimize crunch between developers and get fast feedback.

While facing the deployment of robotic applications, we extract three essential components based on our robotic deployment experience. Firstly, the environment is volatile and complex, most of the platforms whether from industry or academic research hardly consider the environmental adaption of models. The change in the environment is always delivered as non-consecutive events. It has an essential difference compared to continuous and unified query data. These events can either be images, text or environment observations.

Fig. 1: This figure shows DOS® deploy a collision-free navigation algorithm to a swarm of robots. (Turtlebot Burger). The right-top figure is the Operating page where viewing from the coach robot (Pyrobot). User can manually move the robot by the joystick the page provides.

The system should be able to change the company's model with the external context. We first need a model reservoir to manage the pre-trained policies to achieve this. While the model is running, we also want to know how everything is going. Another module collects metrics like runtime CPU, Memory and GPU usages for showing the system's state. These metrics are valuable for later system optimization. For example, if the user finds the system is under high-intensity operation, the system can automatically calculate how many extra computational resources it needs to balance the workload. On the other hand, users have customized criteria for the model's performance. The system can return these evaluations and do further optimization based on that.

We argue that deployment and delivery for robotics need custom software abstractions and toolsets. To address the above challenges, we propose DOS®(Deployment Operating System), a production-level system for deploying the modern data-driven robot applications. DOS® contains programming abstractions, shown in Figure 2. (i) an environmental adapter following the unified interface. We acknowledge the complexity of events, so one single instance taking care of all problems facilitates the maintainability and scalability of the system. In this instance, we devise multiple handlers, each with expertise in processing one specific problem. (ii) a data replay reservoir encapsulates the data model into one unified structure. For a

[1]Department of Computer Science, Northwestern University, Evanston, IL, USA 60201 guoye2018, qinjielin2018; hanliu@northwestern.edu

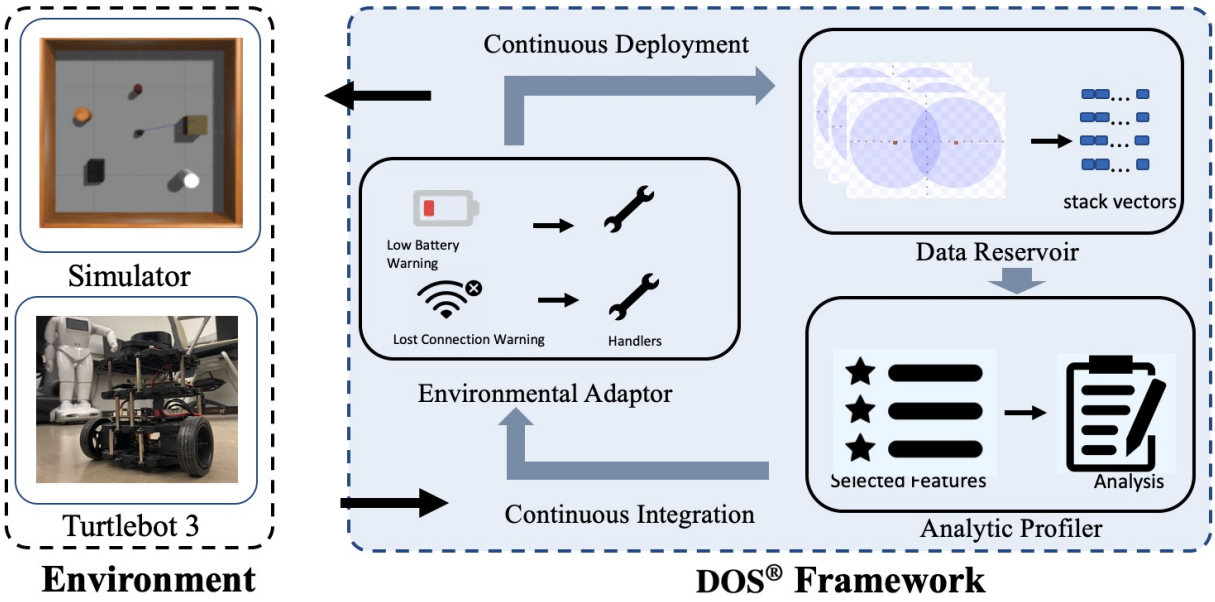


Fig. 2: The overall architecture of the system. DOS® system can interact with both simulators and the real-world. Starting from the environmental adaptor, the system first detect is there any event happened, then feeds the observation to the model and executor, after that it saves the data to the data replay reservoir. This first partition is continuous deployment. The continuous integration part is that Analytic Profiler will examine the performance of the model and system, then make modifications based on the analytic report.

large variety of robotic applications, the models are different in many aspects depending on users' cases. DOS® offers a simple and unified model API supporting easy plugins of users' customized models. (iii) An analytical profiler that collects runtime metrics for optimizations. DOS® also provides a user-friendly UI to faciliate robotic development and deployment.

The rest of the paper is organized as follows. Section 2 introduces recent machine learning deployment system both in academic and industry. Section 3 presents the concept, implementation and components of the DOS® system. Section 4 evaluates DOS® in the context of a participant study. We shows the reliability and convenience of our system. Finally, section 5 provides conclusions and future work.

II. RELATED WORK

Continuous Development / Continuous Deployment. Continuous deployment is a software application that can automatically deploy the software in production. The incremental updates of the model are efficiently released and tested [7]. CD extensively interacts and builds upon continuous integration. Humble et al. [8] introduced some CD examples with guidelines. A common CD includes several stages. Each stage will have a specific tool to finish the job [9]. Canete et al. [10] proposed an energy-efficient deployment in edge-based infrastructure. Ahmadighohandizi et al. [11] present a framework for IoT applications. They design the workflow under the continuous deployment principle. Sokolowski et al. proposed an IaC (Infrastructure as Code) system automating deployment coordination [12]. Mysari et al. proposed a CI/CD pipeline using Jenkins [13]. Amazon Web Services (AWS) provides several specialized commercial services for CI/CD for robotic applications. AWS RoboMaker is a development studio integrating Gazebo simulator [14]. It mitigates the toughness of robotic development by containing preconfigured tools and simulators. To deploy the application to hardware, AWS IoT Greengrass 2.0 [15] provide an example of deploying the robotic applications to a fleet using their modular, self-contained units. Teixeira et al. investigate CI/CD techniques in robotic repositories (ROS packages) [16].

Machine Learning DevOps. Traditional software DevOps aims to improve the efficiency, quality and robustness of the development lifecycle [17]. MLOps is this process specifically applied to Machine Learning area. Compared to traditional DevOps, MLOps is more data-centric. Development stages are orchestrated around a data center [18]. Multiple works have been made to different aspects of data preparation and processing [19]–[24]. Currently, there are a large number of ML DevOps tools on the market. Most of them provide training and inferencing the ML jobs, which leads to that they are deeply interwoven with cloud service vendors. Argo CD [25] is an application deployment and lifecycle management tool based on Kubernetes. MLFlow [26] is an open-sourced ML platform that provides the whole lifecycle of ML pipeline. Vertex AI is an artificial intelligence platform running on GCP (Google Cloud Service) providing the building and deploying of ML models. KServe [27] is an inference platform on Kubernetes previously called KubeServing. As we can see, due to the nature of their full-cycle functionalities, all these methods are deeply binding to at least one cloud service. For robotic applications such as patrolling in the supermarket

or navigation task in our case study, cloud-to-edge transport cost considerable latency. Thus, once the model is trained, it needs to be deployed on the robot rapidly. Our lightweight DOS® is specifically designed for it. In academia, Ease.ML [28] is a management system specially designed for MLops, it divides the development of machine learning application into eight sub-process that covers the whole lifecycle of CI/CD. MLbase [29] provides declarative abstractions for ML tasks and the ability to adapt the learning algorithm.

III. System Design

DOS® contains three key components:(i) an environmental adaptor that handles all kinds of triggered events, (ii) a replay reservoir that stores the data points for incremental modification of the deployed models, and (iii) an analytical profiler that monitors and save the system-wise and model-wise user-defined metrics. The CI/CD paradigm is a methodology that automatically develops and deploys the applications in every stage. We pre-trained the deploying model, but that doesn't mean it never change ever since running. The complex environment leads us to adjust the model using methods like transfer learning [30] or online reinforcement learning [31]. To achieve this goal, we implement analytical profiler for extracting model performance and data reservoir that buffers the most recent data for continuous online development. Furthermore, the analytical profiler also has the functionality of system analysis because the workload system is burdening has a non-negligible influence on model performance. It is also reponsible for recording and optimize the model's performance.

In the following sections, we use a mobile robot navigation task to illustrate each component of DOS®. We suppose the model is reinforcement learning based. The model generates a velocity command and passes it through ROS.

A. Environmental Adaptor

Robots encounter a plethora of unexpected accidents every day. We don't want them to get stuck or disfunctional when doing their job. We hope to implement this adaptor that not only acts as an extensive collection of event handlers but also has an intelligent scheduler that can assign the event to a specialized handler. It will send a notification asking for human intervention if it can't find an appropriate one among all handlers. In such case, human operator needs to write a new handler for the unseen event. The handler's structure is unified, it only takes a few definitions to create a new instance. Our environment adapter considers contingencies during deployment. It sets up a specific handler for each type of event. The adapter's initialization and calls are shown in Figure 3. Here we consider three environmental emergencies: warnings of being too close to the walls, out of battery and broken internet connection. DOS® is implemented as a library, with a definition of name, message type and on_call function, it can quickly bring the robot back into operation. There is also another kind of event caused by multiple potential reasons. For example, the robot spins in one place. There could be no input feeding to model, unexpected value in

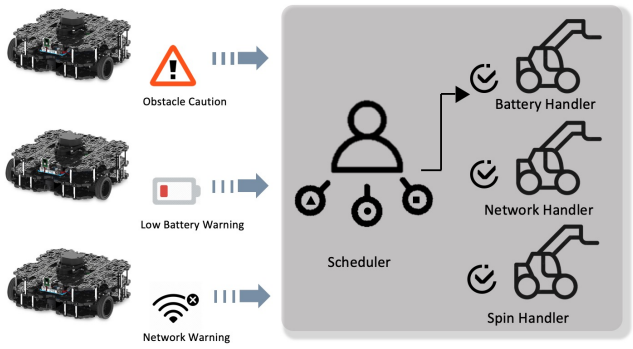


Fig. 3: Three example events the robot may encounter. The environmental adapter scheduler will subscribe to the event and assign it to the corresponding handler.

command or hardware problem. Then the scheduler needs to figure out the problem by using information from the analytic profiler or loop over all the handlers to see if anyone has solved it. We scale our adaptor to different kinds of message tools.

B. Data Reservoir

Experience plays a crucial role in modern robotics learning methods. The learners consume a large amount of data for each time of training. Lots of reinforcement learning libraries provide replay buffer like OpenAI Baselines [32], Ray RLlib [33] and ChainerRL [34]. These libraries mainly provide algorithm implementations and do not put particular focus on replay buffers. It exists only as a necessary part of the ecosystem. Our data replay reservoir is a critical part of CI/CD system. We introduce our data replay reservoir for its high capacity and flexibility. It can efficiently store the customized data generated during deployment. We take the same mobile robot navigation task to go through the efficient functionality of this module. The navigation task takes in acquired sensor data like odometry and lidar sensor data. The lidar data consists of several consecutive frames. Each frame is represented by a list of beam distances. User reconstruct all these raw data into unified `obs_t`. Data replay reservoir store the transition by calling `replay_buffer.add(obs_t,action,reward, obs_tp1,done)`. When moving to the training stage, use `replay_buffer.sample( batch_size,mode)` to retrieve the data points. Batch size is the number of transitions required by this epoch of training. To use the samples stored in replay reservoir, we offer three modes: random, recent, and prioritize. Random mode retrieves samples of batch size using a randomly generated index of all data points. The recent mode only returns the last samples. Prioritize mode returns a tuple of batch size transition with weights added.

C. Analytic Profiler

Analytic Profiler consists of two main modules for a different purpose. The first is the module designed for the evaluation of the model. We can use the loss, accuracy and

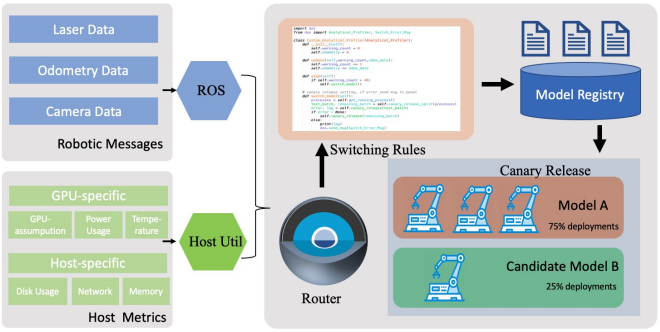


Fig. 4: Analytical profiler first retrieves required messages from middleware like ROS and host utilities provided by DOS® system. Based on user-defined rules, it then picks a candidate model from model registry and use canary release to update new policy to the whole fleet.

total rewards as criteria depending on the algorithm user deploy. These metrics are indispensable in hyperparameter tuning [35]. Tuning methods like bayesian optimization, reinforcement learning-based optimization and grid search consume and make decisions based on these metrics. A user-defined metric like the warnings of too close to the wall also belongs to the model performance category. The second aspect we tend to profile is the system-wise metrics including GPU-specific and host-specific [36]. We list the metrics in Figure 4. Under the context of mobile navigation tasks, since it's equipped with onboard computers with limited computational resources. If the CPU and memory usage are too high, we need to switch to a small model or concatenate the observations to one laser frame etc. Since DOS® is supposed to support massive deployments, we also implement canary release which will update a small group of robots to check the availability before rollout to the whole fleet.

IV. Evaluation

To evaluate the usability and reliability of DOS® 's interface which can help developers efficiently deploy and monitor their programs, we first conduct a user study with 50 participants from various backgrounds. We ask participants to deploy a navigation policy to a mobile robot with customized environmental adapter and analytical profiler. Once finished, the whole system can be deployed with one click from the GUI provides as 7a shows. Participants can also monitor the status and receive the run-time summary. We further investigate each modules's functionality by plotting some metrics as the robot run in Section IV-C. As a robotic deployment operating system, it is a common demand to have the ability to apply to a massive number of robots. So in Section IV-D, we deploy policy to 6 `Turtlebot3 burger` robots to show system's scalability.

A. Environment Setup

We prepare several mobile robots, one server hosting DOS® and three collision-free navigation policies in advance. The details are listed below.

Real Robot. Specifically, we install our system on a mobile robot platform `Turtlebot3 burger`. This robot is equipped with LDS-01, a 2D LIDAR sensing 360 degrees with 0.12 to 3.5 meter range and a scan rate of 300 $\pm$ 10 rpm. We limit the robot's maximum speed to 0.1m/s. The on-board computer of `burger` is `Raspberry Pi 3`. The robot has all the drivers and ROS Kinetic installed and configured, so the participants do not need to make any additional changes to it. The bot publishes the following necessary topics for complete deployment `/scan`,`/odom`,`/cmd_vel`,`/battery_status`, `/is_crashed`. `/scan` data is the lidar sensor data, it's an array consisting 360 float numbers, each number represents the beam range of a certain degree. It's the most valuable input for collision-free decision model. `/odom` is the odometry data recording the position and velocity compared with the start location. `/cmd_vel` is the action command controls the robot's linear and angular velocities. It's also the only output of the model. `/battery_status` and `/is_crashed` are the event messages for environmental adapter. The handlers subscribe to its corresponding topics and deal with policies set in advance. The robot connects to the same local network as server, so the information can also pass to it.

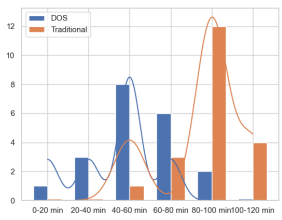
Server. The computer used as a server is an Intel NUC 11 mini with Linux 20.0 installed. DOS® system including backend and frontend server is running and linked using roslibjs [37]. We also installed ROS and Stage simulator [38]. The ROS master node is running on server and the messages can be transferred from the real robot. Utilizing the GUI DOS® provides, participants can complete their deployment and monitor the status of robots from their remote laptops.

Navigation Algorithms. In order to test the generality and robustness of our system, we consider the following three reinforcement learning-based navigation methods: (i) CFNDRL [39], a collision-free navigation policy trained by path-following integration of the environment difficulty. (ii) CADRL [40], an agent-level decentralized method with each agent accesses precise state information. (iii) DRLMACA [41], a fully decentralized method in which the same policy is trained for all agents in several multi-agent environments. All these three algorithms take the laser data and odometry information as input. DOS® has a clear interface for users to replace the models.

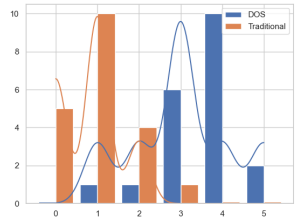
Participants. We expect our participants have some backgrounds either in having used machine learning models or having hand in robotic development. We pick up 20 participants, 14 from Computer Science, 4 from Computer Engineering and 2 from Mechanical Engineering Department. We divide them into two groups, one group utilizes DOS® to finish a deployment task, and another uses python scripts to wrap up the functions required by themselves.

B. RQ1: Does DOS® effectively support CI/CD process for robotic application?

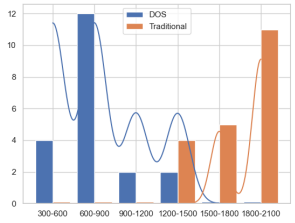
To answer RQ1, our study mainly focuses on measuring agile CI/CD and the user-friendly interaction of our system. Although all the required information is published using ROS



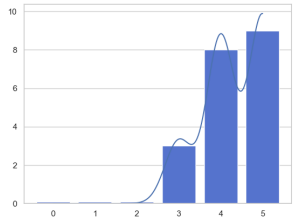
(a) How much time you spend to deploy using DOS®?



(b) How many successful times you deploy among 5 tries?



(c) How many lines of code you write to deploy the model?



(d) What rating you would like to recommend the system?

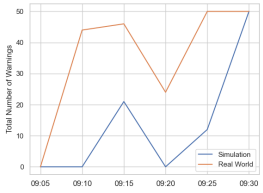
Fig. 5: The blue bar represents the group using DOS®, the orange one is the group who needs to wrap everything by themselves. We use kernel density estimation (KDE) to show the main distribution of the two groups' evaluation. Participants' ratings on preference degree. 1 is least recommended, and 5 is strongly recommended.

topics, we do not expect participants have rich experience in ROS. The instruction gives detailed and precise DOS® API of how to subscribe and publish specific messages. We set up a computational environment and prepare the navigation model in advance. Participants' job is to write the three modules in separate files. Users need to inherit the superclass we provide for environmental adapter and customize their own handlers. The robot's states are transported from the robot by ROS. Users must process it to the observation that fits the replay reservoir's data model. The instruction also provides an example of writing switching rules based on too-close warning quantity. DOS® will take the rest of the work to get the data and utilize them to successfully deploy this navigation algorithm. We ask participants to finish the task within two hours. After the experiment, we collected feedback with evaluation metrics designed to show the accessibility of our system.

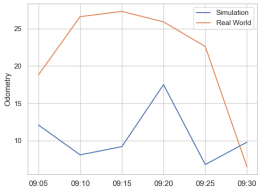
We collect several objective questions in the following: 1) How much time do you spend deploying using DOS®? 2) How many successful times you deploy among 5 tries? 3) How many lines of code do you write to deploy the model? and subjective feedback like 4) What rating you would like to recommend the system from $1-5$, 5 is strongly recommended. 5) What improvements do you think DOS® can make?

As Figure 5a shows, using DOS® can greatly save time. Average time participants use DOS® locates in 40-60 min zone while the group using traditional method takes around one more hour to deploy. We found that most time traditional method group spend is to integrate the different modules into one operable program. We give each participant 5 opportunities to run their deployment on the real robot. We count how many times they successfully move the bot using the policy we provide. Resorting to the modularity of our system, as long as the participant strictly follows our defined API, the whole system is robust and error-free. We use Lines of Code (LoC) to represent the conciseness of programs that aim to complete the same task. We found the most significant difference is that group using the traditional method writes a large part of code to construct redundant events handlers. And the other challenging part is to develop a mechanism using relatively less code that can seamlessly run the program for a day.

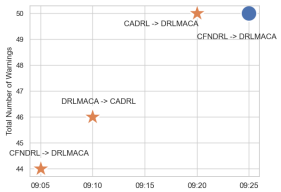
The remaining questions are about the scoring of DOS® and one open question of suggestions participants may have. Some advise further support of ROS2 or another language like C++ for acceleration.



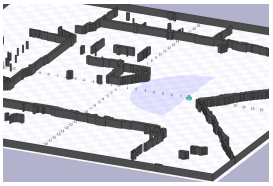
(a) Total number of Warnings



(b) Odometry



(c) Model Switch Time

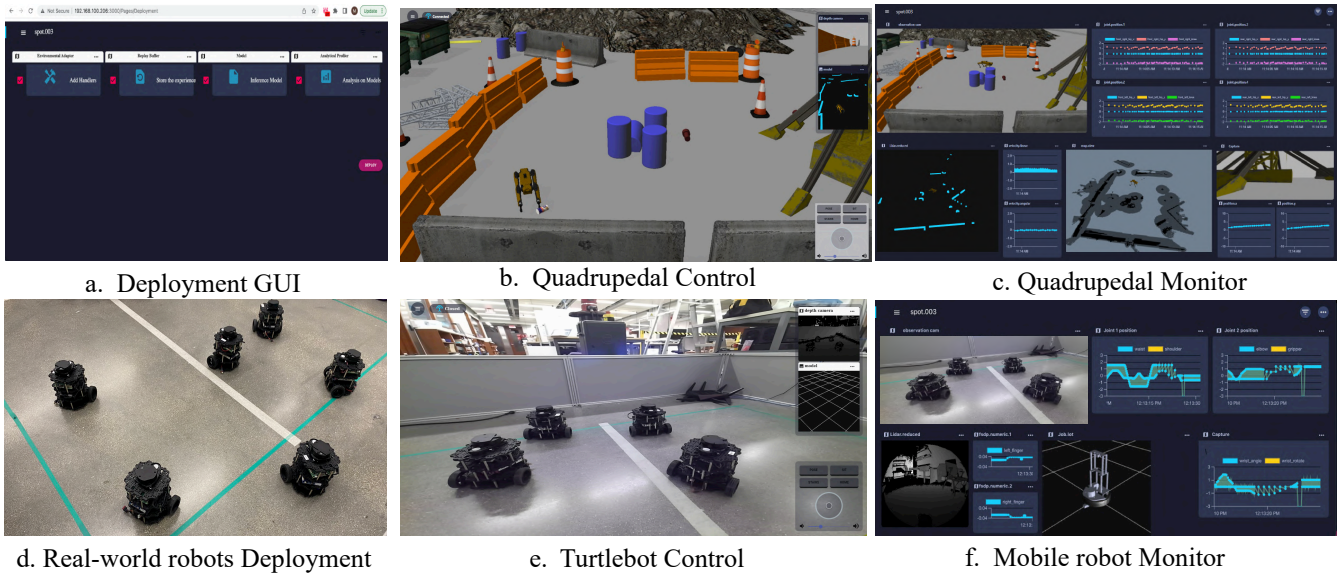


(d) Simulation Environment

Fig. 6: After the successful deployment of the models, we test the system's performance based on the total number of warnings and odometry data. In our navigation algorithm, we count a warning number while the minimum value in one frame of laser scan is less than 0.2m. The odometry data is the distance robot traveled in 5 minutes. Using the analytical profiler in DOS®, the different models can be seamlessly switched by certain rules. (c) shows the exact moment and the total number of warnings when system switches model. The orange star is the real-world deployment and the blue circle is the simulation deployment. Map in (d) is reconstructed from the real test environment. The green box represents the turtlebot with the same range of scan.

C. RQ2: How do modules of DOS® facilitate the users after deployment?

As a complete deployment system, besides assisting in developing the algorithms, it is also essential to gather feedback on the running policy. It can switch models based on user-defined metrics. In this case study, we deploy the policies on the simulation environment and real-world turtlebot. Our system is designed for $7 * 24$ deployment, but here we only deploy the same task for half an hour due to the battery limitation of our mobile platform. Figure 6d shows the



a. Deployment GUI b. Quadrupedal Control c. Quadrupedal Monitor

d. Real-world robots Deployment e. Turtlebot Control f. Mobile robot Monitor

Fig. 7: (a) shows the deployment page. Once users finish the customized modules, use the red button shown at the right-bottom to deploy the policy automatically. (b) and (c) are the operating and monitoring pages of a simulated quadrupedal robot deployment. The view occupying most region on operating page is from the camera on bot, there are also depth camera view and control joysticks on the right side. The monitoring page displays data from analytical profiler like odometry and joint positions. (d) is the deployment to a swarm of real world turtlebots. (e) and (f) are the operating and monitoring page from a pyrobot [42] view.

simulated environment in Stage simulator reconstructed from the laser scan of our laboratory.

For environmental adapter, we set too-close warning, low battery and crash handlers. When receiving the latter two signals from the corresponding ROS topic, the GUI will pop out a window to notice the user. These kinds of events need user's manual management. The too-close warnings is that when the robot navigates too close to the obstacles, i.e, the minimum value in one frame of laser scan is less than 0.2m. The callback function for too-close handler is to add one value of a counter from analytical profiler.

Analytical profiler provides advanced analytics that can deliver useful insights based on the running summary during a certain period. These key attributes can be applied to predict future behaviors and optimize the model. For this example, we record every 5 minutes odometry data (travel distance) and the total number of warnings and use them as indicators of the model's performance. The rule of when to switch the model is specified as if the total number of warnings in the past 5 minutes is higher than a preset average value (We set it as 40 in this experiment), DOS® will randomly pick a new one from the model pool.

Figure 6 shows the changing number of warnings along the operating time and the corresponding Figure 6c marks the time and number of warnings when the analytic profiler decides to switch the model. The label on each star shows the exact transfer between models. Real-world deployment has more warnings and odometry than simulation. We found that's because in simulation, some relatively thin obstacles like table legs are not reconstructed.

D. Can DOS® help to deploy model to massive robots?

For massive deployments, we choose the same navigation model and deploy them to 6 `Turtlebot3 Burger` robots, see Figure 7d. To make the whole system more user-friendly, we also provide operating page and monitoring page see Figure 7b and Figure 7c. On operating page, the main screen is the robot's camera view. There is a joystick where user can use to manually control the robot out of stuck. Users can also use the menu button on the top left to switch among robots. The monitoring page visualizes users' interested topics and plots them over time. While switching to new policy, DOS® provide canary deployment as Figure 4 illustrates. The system will update 3 out of 7 robots, if there is no error coming out, the deployments will continue to the rest.

V. Conclusion

The deployment of machine learning-based robotic models has been under rapid growth recently. However, the complexity also becomes increasingly sophisticated in terms of a plethora of frameworks and libraries. In this paper, we propose DOS®, a robotic deployment system that introduces three modules. Each address one of the key challenges developers commonly encounter. Explicit division brings benefits and efficiency on reliability and robustness. Using such a system is a considerable strength to help cooperation and future development among the team. We demonstrate the power of our system by using it to deploy navigation algorithms to one and a swarm of real-world mobile robots. For future work, we plan to deploy multiple tasks on heterogeneous platforms and scale the scope to more robotic applications.

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*. IEEE, 2010, pp. 3084–3089.

[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)*. IEEE, 2017, pp. 0809–0814.

[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)*. IEEE, 2018, pp. 287–296.

[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)*. IEEE, 2016, pp. 1039–1043.

[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*, 2020, pp. 1–13.

[7] P. Rodríguez, A. Haghighatkhah, L. E. Lwakatare, S. Teppola, T. Suomalainen, J. Eskeli, T. Karvonen, P. Kuvaja, J. M. Verner, and M. Oivo, "Continuous deployment of software intensive products and services: A systematic mapping study," *Journal of Systems and Software*, vol. 123, pp. 263–291, 2017.

[8] J. Humble, C. Read, and D. North, "The deployment production line," in *AGILE 2006 (AGILE'06)*. IEEE, 2006, pp. 6–pp.

[9] Y. Zhang, B. Vasilescu, H. Wang, and V. Filkov, "One size does not fit all: an empirical study of containerized continuous deployment workflows," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 295–306.

[10] A. Cañete, M. Amor, and L. Fuentes, "Energy-efficient deployment of iot applications in edge-based infrastructures: A software product line approach," *IEEE Internet of Things Journal*, vol. 8, no. 22, pp. 16 427–16 439, 2020.

[11] F. Ahmadighohandizi and K. Systä, "Application development and deployment for iot devices," in *European Conference on Service-Oriented and Cloud Computing*. Springer, 2016, pp. 74–85.

[12] D. Sokolowski, P. Weisenburger, and G. Salvaneschi, "Automating serverless deployments for devops organizations," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 57–69.

[13] S. Mysari and V. Bejgam, "Continuous integration and continuous deployment pipeline automation using jenkins ansible," in *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. IEEE, 2020, pp. 1–4.

[14] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3. IEEE, 2004, pp. 2149–2154.

[15] J. Wallace, "Deploy and Manage ROS Robots with AWS IoT Greengrass 2.0 and Docker," https://aws.amazon.com/blogs/robotics/deploy-and-manage-ros-robots-with-aws-iot-greengrass-2-0-and-docker/, 2021.

[16] S. Teixeira, R. Arrais, and G. Veiga, "Cloud simulation for continuous integration and deployment in robotics," in *2021 IEEE 19th International Conference on Industrial Informatics (INDIN)*. IEEE, 2021, pp. 1–8.

[17] C. Renggli, L. Rimanic, N. M. Gürel, B. Karlaš, W. Wu, and C. Zhang, "A data quality-driven view of mlops," *arXiv preprint arXiv:2102.07750*, 2021.

[18] Q. Lin, G. Ye, J. Wang, and H. Liu, "Roboflow: a data-centric workflow management system for developing ai-enhanced robots," in *Conference on Robot Learning*. PMLR, 2022, pp. 1789–1794.

[19] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision," in *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, vol. 11, no. 3. NIH Public Access, 2017, p. 269.

[20] K. Katsiapis and K. Haas, "Towards ml engineering with tensorflow extended (tfx)," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 3182–3182.

[21] T. Kraska, "Northstar: An interactive data science system," 2021.

[22] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg, "Activeclean: Interactive data cleaning for statistical modeling," *Proceedings of the VLDB Endowment*, vol. 9, no. 12, pp. 948–959, 2016.

[23] S. Schelter, S. Grafberger, P. Schmidt, T. Rukat, M. Kiessling, A. Taptunov, F. Biessmann, and D. Lange, "Differential data quality verification on partitioned data," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1940–1945.

[24] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger, "Automating large-scale data quality verification," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 1781–1794, 2018.

[25] https://argoproj.github.io/cd/.

[26] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[27] https://github.com/kserve/kserve.

[28] T. Li, J. Zhong, J. Liu, W. Wu, and C. Zhang, "Ease. ml: Towards multi-tenant resource sharing for machine learning workloads," *Proceedings of the VLDB Endowment*, vol. 11, no. 5, pp. 607–620, 2018.

[29] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, "Mlbase: A distributed machine-learning system." in *Cidr*, vol. 1, 2013, pp. 2–1.

[30] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, no. 1, pp. 1–40, 2016.

[31] C.-Y. Wei, Y.-T. Hong, and C.-J. Lu, "Online reinforcement learning in stochastic games," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[32] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[33] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, "Ray rllib: A composable and scalable reinforcement learning library," *arXiv preprint arXiv:1712.09381*, vol. 85, 2017.

[34] Y. Fujita, P. Nagarajan, T. Kataoka, and T. Ishikawa, "Chainerrl: A deep reinforcement learning library," *Journal of Machine Learning Research*, vol. 22, no. 77, pp. 1–14, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-376.html

[35] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," *arXiv preprint arXiv:1807.05118*, 2018.

[36] A. Bhattacharjee, Y. Barve, S. Khare, S. Bao, Z. Kang, A. Gokhale, and T. Damiano, "Stratum: A bigdata-as-a-service for lifecycle management of iot analytics applications," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 1607–1612.

[37] R. Toris and J. Lee, "roslibjs," https://github.com/RobotWebTools/roslibjs.git, 2014.

[38] R. Vaughan, "Massively multi-robot simulation in stage," *Swarm intelligence*, vol. 2, no. 2-4, pp. 189–208, 2008.

[39] G. Ye, Q. Lin, T.-H. Juang, and H. Liu, "Collision-free navigation of human-centered robots via markov games," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 11 338–11 344.

[40] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3052–3059.

[41] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6252–6259.

[42] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta, "Pyrobot: An open-source robotics framework for research and benchmarking," *arXiv preprint arXiv:1906.08236*, 2019.