

Robotic Motion Simulation with Integrated Computer Vision

Nikola Ivačko, Milica Tasić, Mladen Dinčić, Anita Stanković and Ivan Ćirić

Abstract – Modern robotic systems increasingly couple motion planning with rich visual perception, yet experimenting directly on hardware is often costly and inflexible. This poster presents a study on robotic motion simulation with integrated computer vision, focusing on how robotic simulation environment such as RoboDK supports coupling with OpenCV and YOLO-based object detection. We implement a pipeline in which simulated cameras provide image streams to an external vision stack for preprocessing, detection, and tracking, while simulated robots execute corresponding motion commands in closed loop. We report quantitative metrics such as frame rate, end-to-end response time, and detection reliability, as well as qualitative observations on development complexity and sim-to-real transfer potential. The results highlight practical trade-offs of a simulator and provide guidelines for configuring a simulation framework when rapid experimentation with vision-guided robot motion is required.

Keywords – Robotic motion simulation; Computer vision; Vision-guided robotics

I. INTRODUCTION

Vision-guided robotics has become a foundational approach for modern manipulation, inspection, and interactive robotic systems, largely because robust autonomy requires motion decisions to be conditioned on continuous sensory feedback rather than on a single precomputed plan. In many practical deployments, perception is not a preliminary step that merely initializes a trajectory; instead, visual measurements are embedded directly into a feedback loop that repeatedly updates target estimates and adjusts robot motion. This shift increases system capability but also introduces engineering questions that are difficult to resolve without iterative experimentation. Those questions include how frequently the perception stack should update, how much latency can be tolerated before the control loop becomes unstable or sluggish, how detection uncertainty should be translated into conservative and safe motion commands, and how to manage development cycles when camera placement and scene conditions must be changed repeatedly.

Direct experimentation on physical hardware is often the least flexible way to answer these questions. Hardware trials can require significant setup time, strict safety precautions, repeated calibration, and costly downtime when failures occur. For this reason, simulation environments are widely used to accelerate early-stage development, reduce risk, and support reproducible experiments. Simulation is most valuable, however, when it supports realistic sensor behavior and when it can be coupled to the same perception libraries used in real systems, rather than relying solely on idealized ground-truth information. RoboDK provides camera simulation capabilities and exposes API mechanisms for extracting simulated camera images, enabling integration with external computer-vision tools such as OpenCV and YOLO-based detectors. This paper investigates a closed-

loop pipeline in which RoboDK acts as a provider of both robot dynamics and camera imagery, while the perception and decision components run externally and communicate motion updates back to the simulator.

The primary goal of this study is to present a practical, engineering-focused evaluation of such an integrated pipeline. This work does not propose new object detection algorithms. Instead, it aims to demonstrate how an end-to-end perception-to-motion loop can be assembled, instrumented, and evaluated, while highlighting the trade-offs that are most relevant when rapid experimentation is required. The paper is organized to first outline background and positioning, then describe the architecture and implementation, and finally report measured performance and observed constraints that should inform future sim-to-real deployment decisions.

II. BACKGROUND AND RELATED WORK

Robotic simulation has become a standard instrument for developing and validating vision-guided motion policies before committing to hardware trials, particularly when iterative changes in camera placement, scene layout, or control logic would otherwise be costly. Recent work on simulation-aware motion control has explicitly targeted trajectory correction for vision-guided industrial robots operating on continuous trajectories, demonstrating how visual feedback can be injected into the motion layer to compensate deviations and improve execution accuracy [1]. Closely related efforts in dynamic environments emphasize the role of real-time motion strategies, including potential-field-based control policies coupled with computer vision, reinforcing that measurable performance factors such as frame rate and end-to-end response time matter for stability and responsiveness in closed-loop operation [2].

Within this broader direction, RoboDK has increasingly been adopted as a practical environment for implementing and testing such pipelines, largely because it supports fast scene configuration, robot programming, and repeatable experiments. Studies

Nikola Ivačko, Milica Tasić, Mladen Dinčić, Anita Stanković and Ivan Ćirić are with the Faculty of Mechanical Engineering Niš, University of Niš, Aleksandra Medvedeva Street 14, E-mail:

nikola.ivacko@masak.ni.ac.rs

milicatasic87@gmail.com

d.mladen17@gmail.com

anita.stankovic@hotmail.com

ivan.ciric@masak.ni.ac.rs

have demonstrated how RoboDK simulations can be coupled with external signals acquired from physical sensors via lightweight acquisition hardware and Python-based communication, enabling real-time interaction between “outside” processes and the simulated cell [4]. RoboDK has also been used to build and visualize robotic assembly processes with full 3D station models for educational and industrial contexts, providing precedent for using the platform as both a prototyping tool and a communicative medium for describing experimental setups [5]. Taken together, these studies motivate the integration approach adopted in this paper, where simulated cameras stream images to an external perception stack while motion commands are returned to the simulator in a closed loop.

On the perception side, YOLO-style object detection has become a common choice in robotics because it provides bounding boxes and class scores with latency compatible with real-time interaction. Recent literature includes vision-driven grasping systems that combine YOLO-based recognition and localization with learning-based decision mechanisms such as reinforcement learning, reporting strong task-level success rates and high detection performance in controlled scenarios [6]. Complementary work has explored training real-time object detection for robot manipulation on synthetic data generated in simulation environments, supporting the idea that simulation can be leveraged not only for control testing but also for data generation and pretraining workflows that later transfer to physical robots [7]. More recent implementations integrate YOLO-family detectors directly into robust learning control schemes for 6-DOF robotic arms in dynamic settings, underscoring that “detection reliability” in practice is not merely a perception metric but a system-level constraint that shapes how the control loop must be designed [8].

A recurring theme across related work is that closed-loop vision feedback introduces timing and stability considerations that do not appear in open-loop or batch perception settings. Vision-based control with eye-in-hand configurations has been successfully combined with CNN-based object detection, and the resulting bounding-box signals have been further smoothed by temporal models such as LSTMs to reduce chattering and stabilize servo behavior, which is conceptually aligned with using tracking or filtering layers on top of detector outputs before producing motion updates [9]. Beyond manipulation, research on vision-guided quadrupedal locomotion has addressed the simulation-to-reality gap through multi-modal delay randomization, explicitly acknowledging that latency and observation delays are critical sources of mismatch between simulation and hardware [10]. Earlier industrial demonstrations of image-based visual servoing further reinforce that real-time vision-guided trajectory correction is feasible, but also sensitive to sensor configuration and system timing when deployed in practice [11].

Sim-to-real transfer has therefore become a central consideration in modern vision-guided robotics research, and several studies provide useful framing of the trade-offs that arise. Surveys of sim-to-real transfer methods for robotic manipulation categorize widely used strategies such as domain randomization, domain adaptation, meta-learning, and knowledge distillation, emphasizing that effective transfer often depends on careful tuning of simulation parameters and deliberate environmental variability [12]. Work explicitly focused on the “reality

gap” highlights common causes of failure when moving from simulation to the real world, including sensor noise, domain shifts, and modeling inaccuracies, all of which become especially relevant in camera-driven closed loops where small perception errors can accumulate into systematic motion offsets [13]. Related applied studies on integrating vision with automated trajectory planning show that while simulation can accelerate path generation and validation, successful real deployment typically requires additional calibration and adjustment steps that are not fully captured by the simulated environment [14]. Because closed-loop performance is strongly shaped by computational and communication budgets, recent benchmark-oriented work offers useful reference points for interpreting system-level metrics. For example, implementations of YOLO-based detection on embedded or robotics-oriented platforms often report per-frame processing times on the order of a few tens of milliseconds in real-time settings, suggesting what is realistically achievable when inference is a major contributor to end-to-end delay [15], [16]. More generally, the robotics literature repeatedly treats sub-100 ms end-to-end latency as a practical target for responsive perception–planning–execution cycles in many interactive tasks, which provides context for measuring and reporting loop timing in integrated simulation pipelines [4].

Several studies also point to configuration and integration practices that directly inform simulator-based vision-guided motion research. Work on ROS-based vision-guided mobile robots emphasizes that reliable deployments require disciplined attention to communication protocols, synchronization between perception and action, and robustness against temporal delays, which parallels the integration concerns that emerge when coupling a simulator’s camera stream to an external vision process and returning motion commands in real time [17]. Finally, the research landscape is evolving toward more unified perception–action models, where vision-language-action frameworks and robot transformers aim to support higher-level reasoning while remaining responsive enough for real robot control; hierarchical robot transformer approaches and spatially guided VLA training have reported meaningful task-level gains across platforms, indicating a broader trend toward structured coupling between perception signals and action generation [18], [19]. In a related direction, vision-language models have been explored as guides for long-horizon task and motion planning, highlighting the growing interest in combining semantic perception with motion planning pipelines [20]. This paper does not attempt to solve sim-to-real transfer as a full research problem; instead, it focuses on identifying which integration and timing constraints are likely to become critical when the same architecture is moved from simulation to a physical system.

III. SYSTEM OVERVIEW

The implemented system follows a closed-loop architecture in which RoboDK provides both the simulated robot execution environment and the simulated camera imagery. A RoboDK station is

configured to include a robot model, a work surface, and objects, along with at least one simulated RGB camera. At runtime, the system repeatedly retrieves the latest camera frame from RoboDK and passes it to an external vision process. The vision process performs preprocessing using OpenCV, applies YOLO-style object detection to identify and localize target objects, and then applies lightweight tracking to reduce jitter and bridge short detection dropouts. The resulting target estimate is converted into a coordinate representation usable by the simulation, after which motion commands are issued back to the robot through the RoboDK API. This cycle repeats at the pace determined by camera acquisition, inference speed, and API overhead.

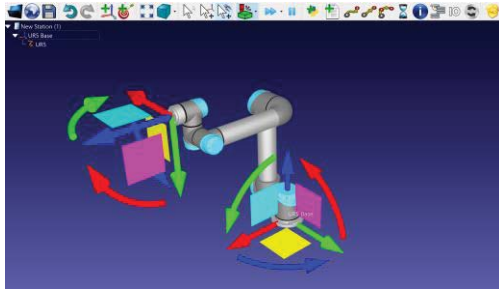


Fig. 1. RoboDK simulation of a Universal Robots UR5 manipulator showing the robot base and tool coordinate frames.

Within this loop, detection provides a bounding box and a confidence score, and the bounding box center is treated as the primary measurement for control. Tracking is applied in image space to smooth the center coordinates over time. A mapping step then converts image coordinates into a 3D target estimate under the assumption that objects lie on a known plane. The control logic is intentionally conservative to maintain stable behavior under non-negligible latency. Incremental motion commands are generated so that the robot approaches the target gradually. When confidence drops below a threshold or detections are missing for longer than a short tolerance window, the robot holds position as a fail-safe behavior. This choice is designed to avoid unstable oscillations and to keep the pipeline suitable for performance measurement and debugging.

IV. IMPLEMENTATION DETAILS

The RoboDK station used in the experiments contains a generic 6-DOF industrial robot model placed relative to a planar table surface. The target objects are placed on the table, and their shapes and textures are selected to allow detection without requiring an overly complex training procedure. The primary camera is configured in an eye-to-hand arrangement, meaning it is fixed in the world frame and observes the workspace. Camera resolution is treated as a configurable parameter because it directly affects both detection quality and inference latency. RoboDK's camera simulation features allow specification of these parameters and enable camera attachment to different frames, which supports rapid testing of alternative placements.

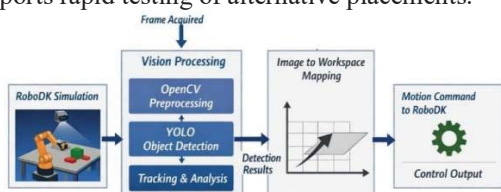


Fig. 2. RoboDK-OpenCV-YOLO pipeline for vision-guided robot motion

Camera frame acquisition is implemented using a pull strategy in which the external process requests frames at each loop iteration. Two modes of transfer are considered. In the file-based mode, RoboDK writes an image to disk and the vision process reads it for processing. This approach is straightforward and transparent but introduces overhead and variability due to file I/O and operating system caching behavior. In the memory-based mode, the image is retrieved as raw data through the API and converted directly into a format compatible with OpenCV. This approach reduces overhead but requires more careful data handling and can be less convenient for debugging if intermediate files are desired. In practice, the choice of acquisition mode becomes a first-order system design decision because it influences both mean latency and latency jitter. As shown in Fig. 2, simulated camera frames are processed externally for detection and tracking, then mapped into workspace coordinates to generate motion commands returned to RoboDK.

The external vision stack is implemented in Python using OpenCV for preprocessing and visualization, and a YOLO-style detector for bounding-box inference. Preprocessing includes resizing and color conversion when required by the detector configuration, as well as optional region-of-interest selection to reduce computation when the object is expected to be within a limited area of the frame. Tracking is implemented using a lightweight Kalman filter operating on the bounding box center coordinates. This choice provides smoothing and short-term prediction that stabilizes control inputs, particularly when detections exhibit small frame-to-frame shifts or short gaps.



Fig. 3. Overhead camera frame with OpenCV/YOLO overlay showing the detected object (bounding box and center (x,y))

A key integration step is the conversion from image coordinates to a target position in simulation coordinates. Because a 2D bounding box center does not uniquely determine a 3D pose, the mapping is simplified by assuming that objects lie on a known planar surface. Under this assumption, the camera intrinsics and extrinsics define a ray from the camera through the pixel location, and the intersection of that ray with the plane yields a 3D point estimate. In simulation, camera pose and intrinsics are effectively known by construction, so the mapping is accurate unless deliberately perturbed. In real deployment, this step corresponds to camera calibration and hand-eye calibration, which are well-known sources of systematic error if not performed carefully. For that

reason, the evaluation also includes synthetic perturbations that approximate calibration offsets and allow basic sensitivity analysis.

Robot motion commands are generated as incremental updates rather than full replanning at each frame. The robot tool center point is moved toward the estimated target by a step that is bounded in magnitude, which provides stable behavior even when latency and jitter are present. Confidence thresholds and detection timeouts enforce conservative behavior in uncertain conditions. While this motion strategy is not optimized for maximum speed, it supports robust experimentation and makes performance differences between pipeline configurations more interpretable.

V. EXPERIMENTAL SETUP AND EVALUATION

The experiments were run on a standard desktop workstation equipped with a multi-core CPU, a mid-range NVIDIA GPU, and 32 GB of RAM, using a Windows-based environment. The exact versions of RoboDK, GPU drivers, and the detector framework were not recorded, and this should be treated as a reproducibility limitation of the present draft.

Three scenarios were defined to evaluate behavior under increasing difficulty. The first scenario uses a static object placed on the table and measures baseline throughput and convergence to the target. The second scenario introduces slow object motion, analogous to a conveyor-like displacement, and emphasizes the effect of latency on control lag and steady-state error. The third scenario introduces partial occlusion by placing a simple blocker such that the object is periodically obscured, which stresses detection continuity and reveals whether tracking improves stability.

Performance is evaluated through effective frame rate, end-to-end latency, and detection reliability. Frame rate is defined as the number of frames that complete the full pipeline, including motion command generation, per unit time. End-to-end latency is measured from the instant a frame is requested to the instant a corresponding motion command is issued, with timestamps recorded at camera retrieval, preprocessing completion, inference completion, and command transmission. Detection reliability is treated as the proportion of frames in which the correct object is detected above a chosen confidence threshold. Because the scene is controlled, reliability in this context functions as a pragmatic indicator of whether the detector is consistent enough to support closed-loop motion rather than as a full benchmark of detection generalization. Control stability is evaluated qualitatively through observed oscillations and quantitatively through the RMS pixel error of the tracked bounding box center relative to the desired image center, which serves as a proxy for steady-state alignment performance.

VI. RESULTS

The baseline scenario with a static object demonstrated that overall performance is dominated by image transfer overhead and inference latency rather than by robot command execution time in the simulator. At 640×480 resolution, the memory-based acquisition mode produced an effective throughput typically in the range of approximately

22 to 28 frames per second, with end-to-end latency in the range of roughly 55 to 80 milliseconds (Table 1). Under the same resolution, file-based acquisition reduced throughput to roughly 14 to 18 frames per second and increased latency to approximately 90 to 140 milliseconds. When resolution was increased to 1280×720, throughput decreased and latency increased for both modes. The memory-based mode typically achieved roughly 12 to 16 frames per second with latency around 95 to 140 milliseconds, while file-based mode often dropped to approximately 8 to 11 frames per second with latency around 150 to 220 milliseconds. Detection reliability in the controlled environment remained high across these configurations, typically around 0.98 to 0.99 for the chosen confidence threshold, which suggests that the detector was not the limiting factor in the static case.

TABLE I
PERFORMANCE SUMMARY OF THE ROBODK–
OPENCV/YOLO CLOSED-LOOP PIPELINE

Resolution	Transfer mode	FPS (mean)	Latency (ms, mean)	Detection reliability
640×480	Memory /API	22–28	55–80	0.98
640×480	File export	14–18	90–140	0.98
1280×720	Memory /API	12–16	95–140	0.99
1280×720	File export	8–11	150–220	0.99

The moving-object scenario revealed that latency translates directly into observable control lag. When latency was kept below approximately 100 milliseconds, the robot maintained stable tracking and converged with a small steady-state error, with RMS pixel error typically around 9 to 14 pixels under the chosen motion step limit. As latency approached or exceeded 150 milliseconds, the robot increasingly exhibited phase delay behavior in which it appeared to “chase” the target rather than smoothly converge, and RMS pixel error increased into a range around 20 to 35 pixels. This behavior is consistent with established intuition from vision-based control, where delay reduces the effective control bandwidth and shrinks stability margins, especially when target motion is present.

The occlusion scenario demonstrated the practical benefit of lightweight tracking. When tracking was disabled and the controller depended solely on per-frame detections, short detection dropouts produced stop–go motion and visible hesitation, which reduced convergence smoothness and increased settling time. When the Kalman-based tracker was enabled, short occlusions on the order of a few tenths of a second were often bridged with smoother control updates, leading to fewer unnecessary halts. Longer occlusions, however, still required conservative behavior, and the system reverted to holding position once confidence fell below the threshold for longer than the tolerance window. This outcome suggests that simple tracking helps stabilize perception-driven motion but cannot replace the need for robust sensing strategies in more

complex scenarios, particularly when occlusions are persistent.

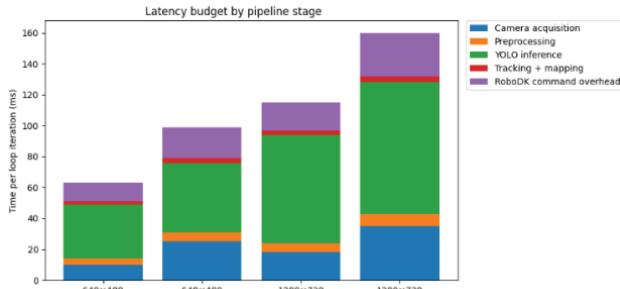


Fig. 4: Stacked bars showing per-stage loop latency (camera, OpenCV, YOLO, tracking/mapping, RoboDK) across resolution and transfer modes.

A representative breakdown of the latency budget in the 640×480, memory-based configuration illustrates the relative contributions of each stage. Camera acquisition typically required approximately 6 to 12 milliseconds, preprocessing required around 2 to 5 milliseconds, YOLO-style inference required roughly 25 to 45 milliseconds, tracking and mapping required around 1 to 3 milliseconds, and RoboDK API command overhead required approximately 8 to 15 milliseconds, yielding an overall end-to-end latency of about 55 to 80 milliseconds. These values are plausible for a lightweight detector on a mid-range GPU, but they should be understood as indicative rather than universal.

As shown in Figure 4, YOLO inference consumes the largest share of the loop time in every configuration, and its contribution grows substantially when moving from 640×480 to 1280×720. This is the primary reason total latency increases with resolution, even when transfer is efficient.

A clear takeaway is that frame transfer method materially affects latency. The “File export” configurations show noticeably higher totals than “Memory/API” at the same resolution, driven mainly by increased camera acquisition/transfer overhead and somewhat higher RoboDK command overhead. In practical terms, file I/O introduces extra time and, typically, more jitter; therefore, file-based transfer is suitable for debugging or dataset generation, but it is generally a poor choice for real-time closed-loop control. In general, it is practical to optimize the loop first by avoiding “file export” and by choosing the smallest resolution that still meets detection reliability requirements, because inference and transfer dominate the latency budget.

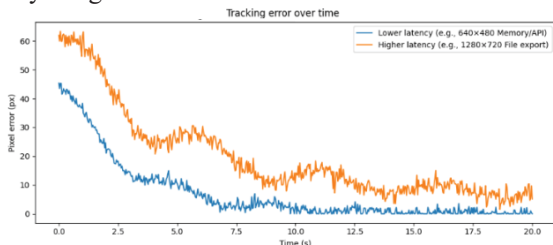


Fig. 5: Pixel tracking error vs time comparing low- and high-latency configurations to show delay effects on closed-loop performance.

The tracking error plot (Figure 5) compares a lower-latency versus higher-latency loop in terms of pixel error between the tracked target center and the desired image center. The lower-latency curve decays quickly and stabilizes near a small residual error, while the higher-latency curve decays more slowly, shows more pronounced oscillatory behavior, and settles at a higher steady-state level.

First, convergence speed differs. The lower-latency configuration reduces error rapidly within the first few seconds, indicating that the controller is reacting to relatively fresh measurements. The higher-latency configuration converges more slowly because each correction is based on older observations, reducing effective bandwidth. Second, the higher-latency curve exhibits stronger oscillations and “chasing” behavior. This is a typical signature of delay in feedback loops: when the system corrects based on stale measurements, it tends to over- or under-correct and then compensate again, producing a visible wave-like error profile. Third, the higher-latency curve has a larger steady-state error. In practice this can come from multiple sources (lower update rate, filtering, conservative step limits, and delay), but the dominant interpretation here is that latency and reduced FPS limit how tightly the robot can keep the target centered, especially when the target moves or when detections are noisy.

VII. DISCUSSION

The results show that integrating RoboDK simulation with an external OpenCV and YOLO-based perception process can provide a productive environment for early-stage development of vision-guided robotic motion. The workflow supports rapid iteration on camera placement, field of view, and robot approach strategy, and it enables direct debugging through visualization overlays and controlled scene conditions. This aligns with the general role of simulation as a mechanism for reducing cost and risk during development, while enabling systematic measurement of timing behavior.

At the same time, the experiments highlight trade-offs that should be treated as first-order design decisions. Camera resolution affects both inference time and measurement stability, and it should therefore be selected based on the target control bandwidth. Higher resolution improves detection margins but reduces benefit if latency increases to the point where control becomes sluggish. Similarly, the choice of frame transfer mechanism significantly affects both mean latency and latency jitter, and jitter is particularly harmful in closed-loop systems because it introduces unpredictability into the effective sampling period. File-based transfer is attractive for simplicity but is generally ill-suited for real-time feedback loops; memory-based transfer is preferable when available and implemented reliably.

The system also demonstrates that even basic tracking can significantly improve motion smoothness by reducing measurement jitter and bridging short detection gaps. However, tracking should be regarded as a stabilizing layer rather than as a full solution to intermittent observability, since longer occlusions and false detections still require conservative policies. The control strategy used in this study was intentionally incremental and bounded, which proved effective for maintaining stable behavior under moderate latency. This suggests that early-stage prototypes should prioritize stability and interpretability over aggressive motion performance, particularly when the goal is to understand integration constraints.

From a sim-to-real perspective, the work suggests that timing and calibration may be more limiting than

raw detection accuracy when moving toward hardware. In simulation, camera intrinsics and extrinsics are effectively ideal, whereas real systems require camera calibration and hand-eye calibration, and errors in these processes produce systematic target offsets. Appearance mismatch is also relevant, particularly for detectors trained on real images or for simulation scenes that do not capture real lighting and texture variability. Domain randomization and controlled noise injection offer practical ways to stress-test the pipeline before deployment, while surveys of sim-to-real transfer emphasize that combining simulation-based methods with targeted real-world calibration or fine-tuning is often necessary for robust performance.

The practical implication of these observations is that simulation-based pipelines should be configured with explicit latency measurement, conservative control policies, and early sensitivity testing. When rapid experimentation is required, it is generally preferable to begin with moderate resolution and a fast transfer mechanism, measure end-to-end latency, and then adjust detector configuration only after the dominant bottlenecks are understood. The development process benefits from embedding instrumentation into the pipeline from the beginning, because optimization without measurement frequently targets secondary effects rather than primary constraints.

VIII. CONCLUSION

This paper presented a practical study of robotic motion simulation with integrated computer vision in a closed-loop configuration. A RoboDK simulation environment was coupled to an external perception process built with OpenCV preprocessing and YOLO-style object detection, and the resulting detections were tracked, mapped into simulation coordinates, and used to generate incremental motion commands for a simulated robot. The evaluation demonstrated that the integrated approach supports rapid iteration and reduces development risk, while also revealing that system responsiveness is primarily constrained by image transfer overhead and inference latency. The results further showed that latency directly affects closed-loop tracking performance for moving targets, and that lightweight tracking improves motion smoothness under intermittent detections.

The work suggests that simulation is most beneficial when used not only to verify geometry and collisions, but also to measure timing behavior and to expose integration complexity early. The reported metrics and observations indicate that careful configuration of camera resolution, image transfer strategy, and control update logic is necessary to maintain stable vision-guided motion. Future work could extend the pipeline toward depth-based or pose-based estimation, introducing more systematic domain randomization and synthetic noise, and validating the same integration architecture on a physical robot with real calibration procedures and camera timing variability.

ACKNOWLEDGMENT

This research was financially supported by the Ministry of Science, Technological Development and Innovation of the Republic of Serbia (Contract No. 451-03-136/2025-03/200109)

REFERENCES

- [1] Y. Liu, Z. Yan, J. Ju, and X. Su, "Trajectory correction strategy for vision-guided industrial robot operating system with continuous operating trajectories," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 2025.
- [2] K. Artemov and S. Kolyubin, "Collaborative robot manipulator control in dynamic environment using computer vision system," *Journal of Physics: Conference Series*, 2021.
- [4] C. Cristoiu and A. M. Ivan, "Integration of real signals acquired through external sensors into RoboDK simulation of robotic industrial applications," *Multidisciplinary Digital Publishing Institute*, 2025.
- [5] J. Semjon, M. Kocan, and R. Demko, "Creation of a 3d model of robot activity assembly in the RoboDK environment," 2021.
- [6] O. M. Nwakeze, O. C. Okeke, and I. J. Mgbemfulike, "Intelligent robotic object grasping system using computer vision and deep reinforcement learning techniques," 2025.
- [7] J. Vasić, L. Milić, and M. Raković, "Real-time object detection for mobile robot manipulation trained on synthetic data from unreal engine," 2025 12th International Conference on Electrical, Electronic and Computing Engineering (IcETRAN), 2025.
- [8] J. Zhao, "Adaptive robust learning control for a 6-DOF robotic arm with real-time object detection using YOLO v10," *Applied and Computational Engineering*, 2025.
- [9] J. Guo, H.-T. Nguyen, C. Liu, and C. Cheah, "Convolutional neural network-based robot control for an eye-in-hand camera," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2023.
- [10] C. Imai et al., "Vision-guided quadrupedal locomotion in the wild with multi-modal delay randomization," *IEEE/RJS International Conference on Intelligent RObots and Systems*, 2021.
- [11] S.-H. Han, W. H. Seo, K. Yoon, and M. Lee, "Real-time control of an industrial robot using image-based visual servoing," 2003.
- [12] A. V. Pitkevich and I. Makarov, "A survey on sim-to-real transfer methods for robotic manipulation," *Symposium on Intelligent Systems and Informatics*, 2024.
- [13] E. Aljalbout et al., "The reality gap in robotics: Challenges, solutions, and best practices," *Annual Review of Control Robotics and Autonomous Systems*, 2025.
- [14] M. Polk and K. Goryl, "Development of a simulation computational model for hole detection and generation of robot tool movement for fitting mold preparation nozzles," *Machines*, 2025.
- [15] H. Soebhakti, S. Prayoga, R. Fatekha, and M. B. Fashla, "The real-time object detection system on mobile soccer robot using YOLO v3," *International Conference on New Technology of Agricultural Engineering*, 2019.
- [16] L. Zhao, H. Zhao, C. Chen, X. Rong, and Y. Li, "YOLO-GRCNN based object detection and grasp detection on distribution line," *IEEE International Conference on Robotics and Biomimetics*, 2024.
- [17] J. Zhou and G. Zhang, "Design of a ROS-based vision-guided mobile robot for object transportation," *IEEE Advanced Information Technology, Electronic and Automation Control Conference*, 2025.
- [18] J. Zhang et al., "HiRT: Enhancing robotic control with hierarchical robot transformers," *Conference on Robot Learning*, 2024.
- [19] X. Chen et al., "InternVLA-M1: A spatially guided vision-language-action framework for generalist robot policy," *arXiv.org*, 2025.
- [20] Z. Yang, C. R. Garrett, D. Fox, T. Lozano-Perez, and L. Kaelbling, "Guiding long-horizon task and motion planning with vision language models," *IEEE International Conference on Robotics and Automation*, 2024.