



OPEN MOTT modular optical tool tracking framework enabling efficient benchmarking

Federico Salerno , Alessandro Contenti , Luca Ulrich , Giorgia Marullo , Sandro Moos  & Enrico Vezzetti 

Optical tool tracking is the process of determining the 6DoF pose of an object in real time using visual sensor streams and image processing algorithms. It enables spatial localization in applications such as robotics, medical imaging, augmented reality, and precision manufacturing. However, existing solutions often involve tight coupling between hardware and software, complicating the management and benchmarking of different tracking systems. This paper presents the Modular Optical Tool Tracking (MOTT) framework, a unified platform for implementing, integrating, and benchmarking optical tracking solutions. A requirement-based design approach was adopted, using Quality Function Deployment (QFD) to systematically derive technical features and identify design drivers that guided the architecture of the framework. The resulting software framework standardizes the concept of an optical tracking method, featuring a flexible and extensible architecture based on object-oriented principles. Two marker-based tracking methods using an RGB camera as video source were evaluated through the developed framework. The presented case studies showcased the use of the framework for method implementation and comparison within a unified pipeline, reporting computational metrics such as frame rate, CPU and memory usage, and providing pose visualizations. The proposed framework enables standardized evaluation and benchmarking of optical tracking systems, and provides a foundation for future extensions involving non-optical tracking modalities and large-scale comparative studies. The implementation is openly available at <https://github.com/tooltip-optical-tracking/mott-framework>.

Keywords Optical tracking, Pose estimation, Modular framework, Benchmarking, QFD, Tool tracking

Optical tool tracking refers to the task of localizing and following an object over time using visual sensor data and image processing techniques^{1,2}. Within this domain, pose estimation is an algorithmic component^{3,4} that computes the spatial parameters of the object with respect to a reference frame, typically expressed as its three-dimensional position and orientation (6DoF).

More specifically, it is referred to as 6DoF pose estimation when the position of the object's centroid, i.e. the three coordinates x , y , and z , is calculated along with roll, pitch, and yaw, thus covering the six degrees of freedom of a rigid body. It is therefore possible to perform tracking without pose estimation, but not vice versa.

Optical tool tracking enables real-time determination of object position and orientation using visual sensor streams and image processing algorithms. It is widely used in various domains where spatial awareness is critical, including medical imaging⁵, precision manufacturing^{6–8}, and robotic systems⁹.

In the industrial domain, optical tracking monitors tool position and orientation in automated processes, ensuring millimeter-level accuracy in tasks like assembly and quality control^{10,11}. It is also applied in industrial maintenance and calibration, where AR enhances technician precision^{12,13}.

Tracking methods can be classified along two complementary dimensions: the camera setup and the detection strategy for the tool, with various approaches available depending on the application. The camera setup encompasses the type of camera used, such as RGB, grayscale, or infrared, along with more specialized options like multispectral cameras, which capture images across multiple discrete spectral bands¹⁴, hyperspectral cameras, which record a continuous spectrum for each pixel¹⁵, or event-based cameras, which detect changes in the scene asynchronously rather than capturing frames at fixed rates¹⁶.

Department of Production and Management Engineering, Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129 Turin, Italy. ✉email: federico.salerno@polito.it

Furthermore, some cameras provide depth estimation capabilities¹⁷, achieved through various techniques including passive and active stereoscopy, structured light, Time-of-Flight (ToF), and depth from defocus (DFD)¹⁸. The detection strategy can be broadly categorized into markerless and marker-based approaches. In markerless tracking, the tool is recognized and followed directly based on its intrinsic features, without relying on external markers^{19,20}. Recent advancements in this area exploit machine learning and deep learning methods to improve robustness and generalization capabilities^{2,21}. In contrast, marker-based tracking uses predefined visual references to facilitate detection and pose estimation, offering higher reliability in controlled environments. These markers may include pattern-based markers such as QR codes, ArUco markers, or image targets^{22–24}, as well as localization markers that employ visible light or infrared LEDs. Additionally, passive markers, including retroreflective spheres or colored markers, are commonly utilized in these systems^{25,26}.

Optical tracking methods are widely applied in various fields^{9,27,28}, but their implementations are often tightly coupled with specific hardware and software configurations²⁹. This tight integration makes it difficult to upgrade hardware, experiment with alternative tracking algorithms³⁰, or combine different methods within a single system, largely due to the lack of standardization³¹.

Although several tracking methods exist, there is no widely adopted framework that allows for seamless combination or swapping of these methods without significant customization^{32–35}. Existing toolkits and prototypes highlight useful contributions but remain limited in scope and adoption. This gap in flexibility and interoperability suggests the need for a more generalized solution.

This work aims to address these limitations by proposing the MOTT (Modular Optical Tool Tracking) framework, which abstracts the execution flow of various tracking methods. The framework is designed to be adaptable to different hardware and algorithms, allowing developers to experiment with and compare multiple tracking techniques without the need for extensive software modifications. This flexibility makes it suitable for both research, where rapid prototyping and testing are essential, and industrial applications, which require robust and scalable solutions.

The key contributions of this work are:

- The application of Quality Function Deployment (QFD) to systematically derive requirements for an optical tracking framework, enabling alignment between user needs and technical features through structured analysis by domain experts.
- The definition of a software framework that standardizes the concept of an optical tracking method, designed with a flexible and extensible architecture based on object-oriented principles.
- The demonstration of the framework's applicability through two case studies showcasing two state-of-the-art marker-based tracking methods using an RGB camera.
- The introduction of a modular structure that supports benchmarking of optical tracking methods, providing a foundation for the comparison and evaluation of different algorithms and hardware configurations.

The remainder of this paper is structured as follows: Section “[Related works](#)” reviews related work and existing tracking techniques; Section “[Materials and methods](#)” introduces the proposed framework and outlines its implementation; Section “[Case studies](#)” presents two case studies demonstrating the framework's capabilities; finally, section “[Discussion](#)” provides a discussion of the results, and section “[Conclusions](#)” concludes the paper with suggestions for future research directions.

Related works

Research on optical tracking spans hardware testbeds, methodological surveys, commercial and custom-built systems, and integration middleware. Across these strands, prior work highlights the need for standardized evaluation while revealing fragmentation that hinders comparability and reuse.

Benchmarking platforms have commonly employed physical artifacts to quantify accuracy and stability. Khadem et al.³⁶ introduced a mechanical setup to measure jitter in commercial trackers, and Wiles et al.³⁷ formalized protocols, showing that vendor procedures may conceal non-uniform error distributions. Herregodts et al.³⁸ emphasized dynamic evaluation with moving targets, indicating that static-only tests underestimate errors encountered in practice. These contributions provide concrete procedures but remain tied to specific hardware and do not generalize to modular, algorithm-level benchmarking.

Methodological reviews point to heterogeneity in definitions, datasets, and metrics. Bouget et al.² analyzed vision-based tool tracking in surgery, documenting variability that complicates cross-paper comparison. Hoque et al.²¹ surveyed 6DoF pose estimation and noted challenges in reproducibility and evaluation design. Fritz et al.³⁵ discussed AR/VR benchmarking and the scarcity of unified protocols. Collectively, these surveys indicate that algorithmic progress has often outpaced the standardization of evaluation pipelines.

Commercial navigation systems and in-house platforms demonstrate clinical feasibility but add constraints for research. Studies such as Koivukangas et al.³⁹ and Kral et al.⁴⁰ compared optical and electromagnetic trackers in surgical contexts, reporting trade-offs shaped by line-of-sight and environment. Proprietary ecosystems offer accuracy and robustness yet restrict extensibility, while custom-built solutions are typically tailored to specific experiments and are difficult to reproduce or share across groups.

General-purpose tracking benchmarks advance evaluation in adjacent domains but differ in scope from optical tool tracking. Dendorfer et al.⁴¹ provide standardized metrics and datasets for multi-object tracking, primarily targeting 2D object trajectories rather than 6DoF tool pose. Li et al.⁴² introduced LasHeR for RGB–thermal tracking, focusing on cross-modality robustness. Pouw et al.⁴³ proposed a pedestrian-tracking benchmark emphasizing flow, density, and trajectory accuracy. These resources strengthen dataset-driven comparisons but do not offer a reconfigurable software framework for optical tool pipelines.

Several integration frameworks address interoperability. OpenIGTLink⁴⁴ enables vendor-neutral communication between devices and navigation software, and the PLUS toolkit⁴⁵ connects ultrasound and trackers to 3D Slicer, facilitating acquisition and streaming. Such tools are effective for device integration but are not designed as benchmarking harnesses for evaluating interchangeable detection and pose-estimation modules under consistent runtime conditions.

Within robotics, ROS and ROS 2 are widely adopted middleware for modular system design^{46,47}. They provide message passing, node abstractions, time synchronization, and simulation tooling that support orchestration across perception, planning, and control^{48,49}. Their objective, however, is system-level coordination rather than controlled assessment of individual tracking algorithms. ROS-based pipelines depend on distributed runtimes and layered configuration, which can complicate fine-grained benchmarking. By contrast, dedicated frameworks for optical tool tracking emphasize lightweight, single-process execution with stable input–output contracts and built-in computational profiling; such specialization complements ROS/ROS 2 and can be wrapped as nodes when broader orchestration is needed.

Taken together, prior work provides valuable protocols, surveys, benchmarks, and integration layers, yet leaves gaps for a domain-focused, modular framework that standardizes method-level evaluation for optical tool tracking while remaining lightweight and adaptable.

Materials and methods

In this section, we outline the approach and processes used for the development and implementation of the proposed framework. In section “[Requirements for framework design](#)” we first identify the requirements necessary for the framework design, followed by the construction and interpretation of the QFD. Subsequently, the design of the framework is detailed in section “[Framework design](#)”, including its structure and operational flow. Source management, tracking management, and feedback management are illustrated in sections “[Source management](#)”, “[Tracking management](#)”, and “[Feedback management](#)”, respectively. In section “[Framework implementation](#)” we describe the software architecture of the framework.

Requirements for framework design

To guide the development of the framework, a requirement-based design methodology was adopted. Among the available approaches for capturing and structuring requirements, Quality Function Deployment (QFD)⁵⁰ was selected. QFD has been widely applied in engineering design because it establishes a direct mapping between stakeholder needs and technical features, ensuring traceability throughout the design process. Compared to diagrammatic modeling approaches such as UML use-case diagrams⁵¹ or agile requirement techniques⁵², QFD offers a structured matrix representation that is particularly effective when multiple technical solutions must be weighed against heterogeneous user demands.

The QFD in this study was developed by a focus group composed of the authors of this work, plus two researchers in computer engineering with expertise in computer vision and software architectures and two researchers in management engineering with experience in requirements engineering and design methodologies, all affiliated with Politecnico di Torino. This composition was chosen to balance domain-specific technical expertise with methodological perspectives, reducing the risk of overlooking either practical constraints or higher-level design goals.

The outcome of this process was the identification of guiding principles, referred to as design drivers, and a set of key framework features that collectively shaped the architecture of the proposed system. These drivers and features ensured that the framework design remained aligned with both user needs and technical feasibility, providing a foundation for the implementation described in the following sections.

QFD development

The QFD integrates two orthogonal dimensions: the horizontal dimension represents the user or system requirements, while the vertical dimension represents the technical features or engineering characteristics that must be implemented to meet these requirements. By mapping these two dimensions, the QFD enables a systematic analysis of how well the technical features align with the identified requirements, ensuring a design that reflects both functional and technical objectives.

Regarding the design of the QFD, the requirements were first collected and then refined through discussion within the focus group. Each requirement was individually assigned an importance weight from 1 (low) to 5 (high), reflecting its relative priority for the framework. After individual assignments, the weights were compared and discussed collectively until consensus was reached, so that the final scores reflected agreement among all members.

Once the requirements were fixed, the focus group used a brainstorming session to define the set of technical features that could fulfill them. The number of features was intentionally kept within a manageable range to preserve the clarity and interpretability of the matrix, in line with recommendations from established QFD methodology^{50,53}. Each feature was included only if the group agreed on its direct relevance to at least one of the main requirements.

The relationship between requirements and features was then scored using the standard QFD convention (1 = weak, 3 = medium, 9 = strong). Each member first proposed correlation values independently, then the group converged on shared values through open discussion. This process ensured that both expert judgment and consensus were represented in the final matrix. The weighted correlations derived from multiplying requirement importance by correlation strength highlighted which features contributed most strongly to satisfying the requirements.

The focus group also considered interdependencies within requirements and technical features, summarizing them in the QFD roof of correlations

Figure 1 illustrates the resulting QFD. For the sake of compactness, the full lists of requirements and technical features, together with their detailed descriptions, are reported in Online Appendix.

QFD interpretation

In this work, we define *design drivers* as high-level technical principles derived from the QFD analysis⁵⁴. These drivers group together related technical features and provide guidance for the architectural, functional, and implementation choices made during the framework’s development. This approach is consistent with requirement-driven design methodologies⁵⁵, and aligns with the use of QFD in software engineering⁵⁶.

The highest-rated technical features are used to define a set of design drivers. Figure 2 presents a visual mapping between technical features, the corresponding design drivers, and the resulting framework features.

The most influential technical feature, *Modular architecture*, together with *Configurable system*, defines the primary design driver, *Framework design*, which governs the fundamental characteristics of the framework, including its architectural and operational structure.

Subsequently, additional design drivers related to the framework’s functionalities were identified from the QFD’s technical features: (i) *Multi-sensor support* and *Public API* define the design driver *Source management*. (ii) *Configurable system* also contributes to the design driver *Tracking management*. (iii) *Visual feedback* and *Tracking error measurements* define the design driver *Feedback management*.

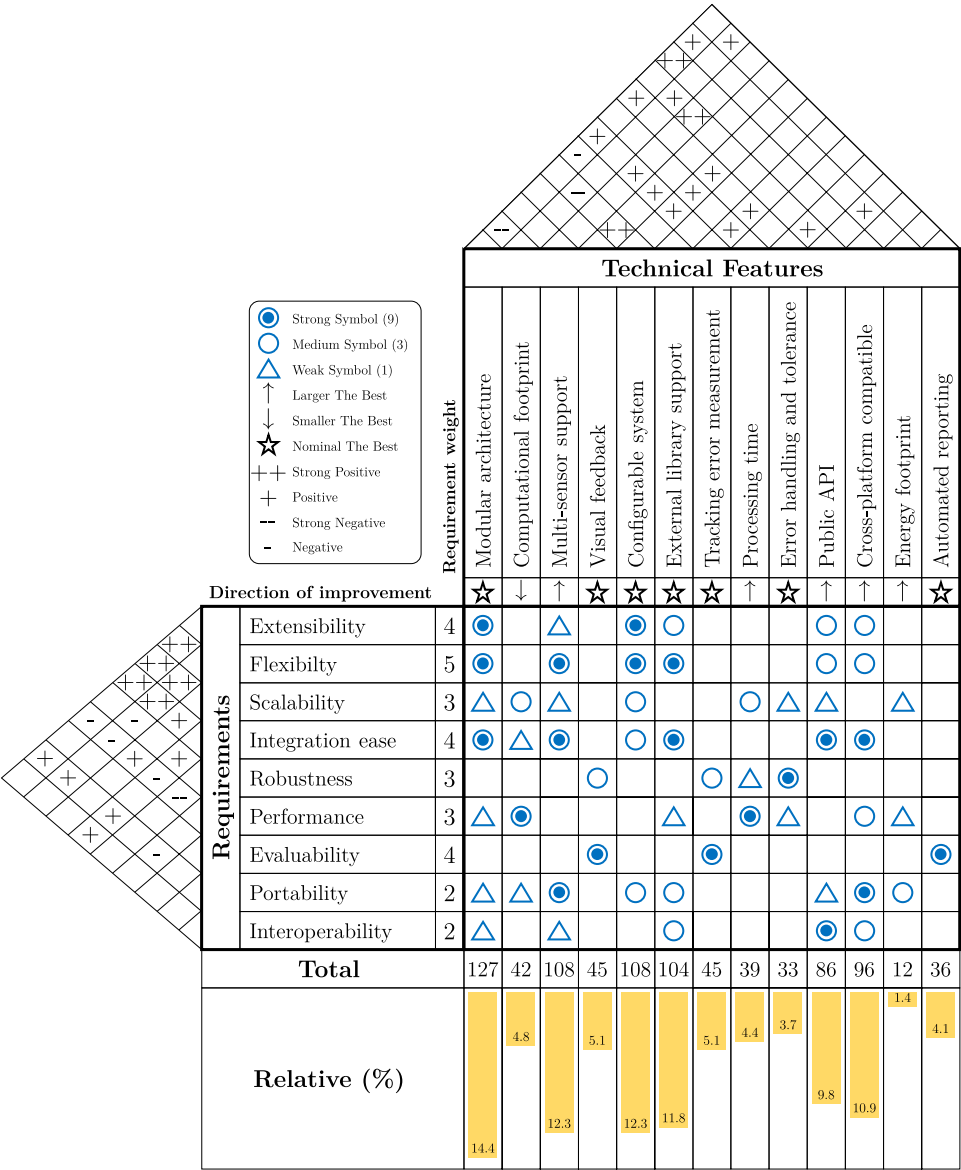


Fig. 1. Quality Function Deployment (QFD) showing the alignment between the requirements and technical features of the tracking framework. The left side details the system requirements, while the top section presents the technical features.

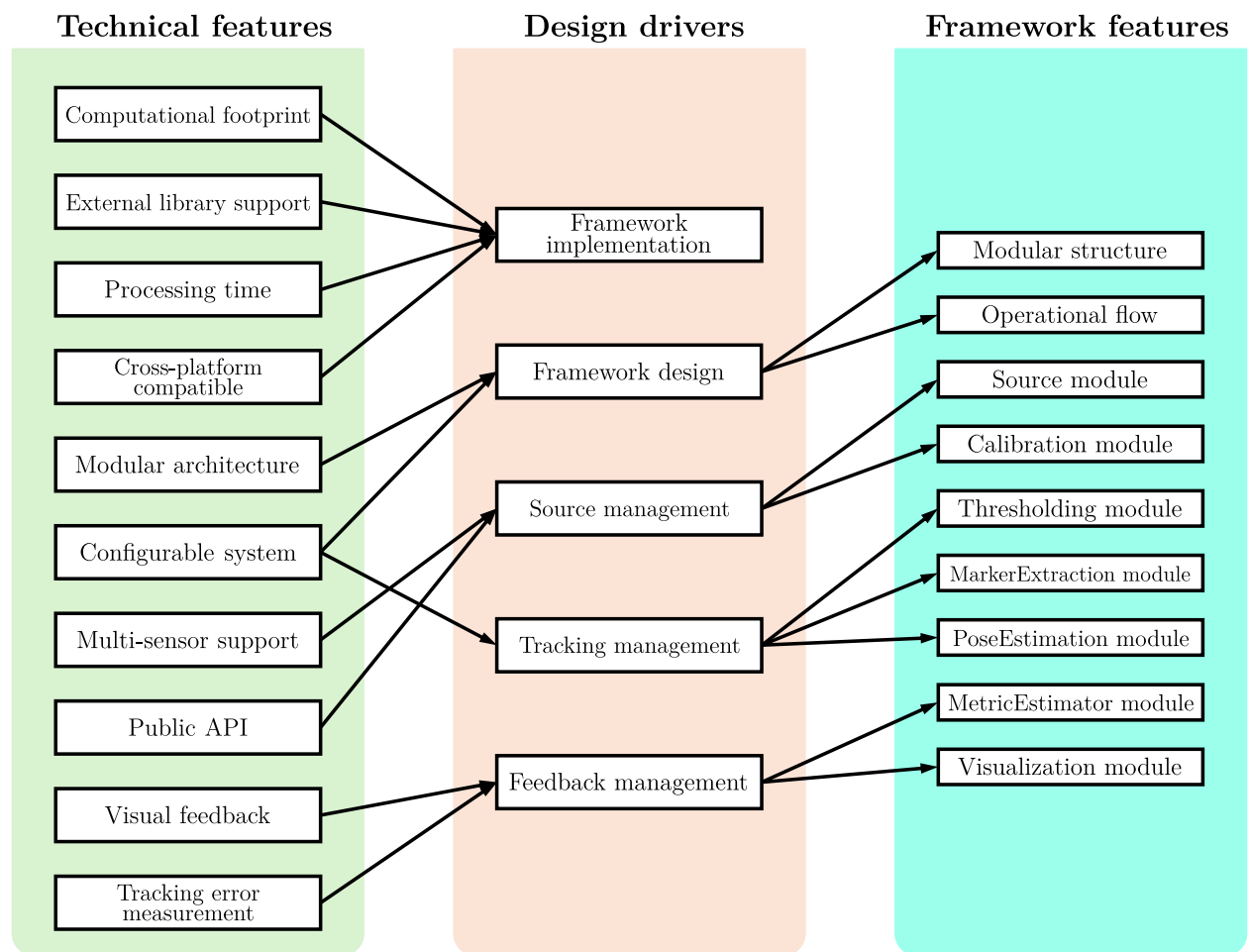


Fig. 2. Mapping between technical features, design drivers, and framework features resulting from the QFD analysis. This diagram was specifically developed as part of the present work.

Finally, a comprehensive design driver addressing the *Framework implementation* was derived from the following technical features: *Computational footprint*, *External library support*, *Processing time*, and *Cross-platform compatible*.

These design drivers collectively guided the decision-making process throughout the framework's development. Each driver provided a structured approach to define key architectural, functional, and implementation aspects, ensuring that the framework meets the identified technical requirements.

Consequently, from the identified design drivers, specific *framework features* have been derived to translate the high-level design principles into concrete characteristics that the framework must have. These features define the practical aspects that the framework must meet to ensure compliance with the established design drivers. They encompass key operational characteristics, such as modularity, structured data flow, and support for essential processes including source management, calibration, thresholding, marker extraction, pose estimation, metric computation, and data visualization. These elements collectively contribute to the achievement of a coherent and efficient framework architecture.

Framework design

The *Framework Design* design driver led to the definition of two key *framework features* that define the system's architecture: a modular structure and a clear operational workflow.

The framework is organized into independent modules, each responsible for a specific task within the optical tracking process, such as image acquisition, preprocessing, marker tracking, or performance evaluation. This structure improves maintainability and allows components to be added, modified, or replaced without impacting the rest of the system, enabling straightforward adaptation to different tracking algorithms and configurations. By abstracting each functionality, the framework eases seamless interchangeability and adaptability to various tracking applications and hardware setups, and ensures scalability for extending with new tracking methods, external libraries, or additional sensors. Furthermore, it simplifies development through clear module boundaries, promotes code reusability across different scenarios, and enhances debugging and optimization due to the independence of each module.

The operational flow of the framework is structured around a central entity, the *Pipeline*, which orchestrates the interaction between different processing modules, ensuring that each step in the tracking process is executed in the correct sequence. The operational flow is divided into three phases: *initialization*, *continuous execution*, and *finalization*. During the initialization phase, the framework configures all necessary resources and sets up the computational environment, loading parameters, establishing hardware connections, and performing essential preprocessing steps to prepare input data for subsequent tracking operations. This phase also includes preliminary setup tasks that may require human intervention, whether they need to be performed at every execution or only once.

Once initialized, the system enters the continuous execution phase, where data is processed in real time. At each iteration, new input is acquired, relevant tracking information is extracted, processed according to the chosen methodology, and output is generated for visualization or further analysis. Execution can terminate under different conditions, such as a user request, the exhaustion of available data, or the completion of a predefined time limit. Finally, during the finalization phase, all allocated resources are released, connections with external devices are closed, and any necessary data is stored before shutdown.

A schematic representation of the operational flow is depicted in Fig. 3.

Source management

This design driver deals with handling data acquisition from camera devices and camera calibration and configuration. It supports arbitrary camera configurations, from single-camera to stereo and multi-camera setups. In such scenarios, the framework does not directly handle time synchronization, which must instead be managed within the implementation of the specific *Source*. It led to the introduction of two framework features that correspond to framework modules:

- *Source module*: it is responsible for managing video input streams. It is designed to support a generic video source, whether from cameras, prerecorded videos, or synthetic data sources. It accommodates any camera configuration, supporting an arbitrary number of source feeds in various video formats, such as single-camera, stereo, and multi-camera setups, as well as RGB, black-and-white, and depth cameras. The modular design allows for switching between different input sources without modifying the core framework logic.
- *Calibration module*: given that accurate tracking often requires precise knowledge of the intrinsic and extrinsic parameters of the cameras used, this module performs this task by computing these parameters, for example, focal length, optical center, distortion coefficients, and relative camera poses in multi-camera setups.

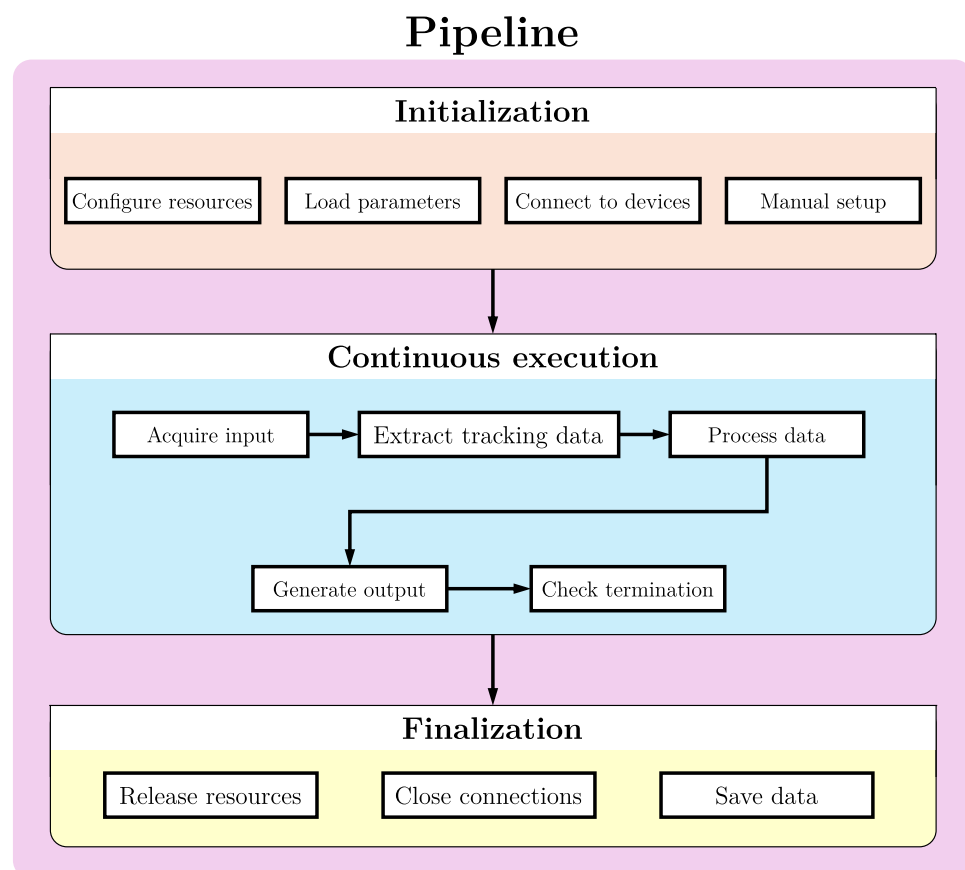


Fig. 3. Operational flow diagram.

Tracking management

The present design driver is responsible for detecting markers, estimating their spatial positions, and computing the tool's pose. To maintain a structured and adaptable approach, we identified a series of framework features and corresponding modules, each performing a specific function:

- *Thresholding*: this module preprocesses acquired frames by applying image segmentation techniques to isolate markers from the background, for methods that require this preliminary step. The thresholding parameters are configurable, allowing adaptation to different environments and tracking scenarios.
- *MarkerExtraction*: this module detects and localizes markers within preprocessed images, extracting their relevant features for further processing. It supports tracking algorithms that determine marker coordinates either in the 2D image space or in the 3D world space, ensuring compatibility with diverse tracking methodologies.
- *PoseEstimation*: once markers are identified, this module computes the tracked object's six degrees of freedom (6DOF) pose, i.e., its position and orientation in the 3D world space. Its modular implementation allows different estimation methods to be integrated, providing flexibility to balance accuracy and computational efficiency.

Feedback management

The present design driver facilitates real-time feedback and quantitative performance evaluation, enabling continuous system monitoring and data-driven adjustments to enhance tracking precision and stability. This led to the inclusion of framework features and respective modules dedicated to visualization and metrics estimation:

- *Visualization*: this module renders tracking results in real-time, providing users with visual feedback, such as marker positions, estimated poses, and tracking trajectories. This module may be used for debugging, system calibration, and applications requiring immediate feedback.
- *MetricEstimator*: its role is to provide an evaluation of the tracking process through quantitative metrics. This module is particularly important for benchmarking different tracking configurations and methodologies.

Framework implementation

In this section, we describe the design driver that governs the framework implementation, detailing how it translates into the concrete development of the system. The MOTT framework has been developed using an object-oriented approach in C++, leveraging the OpenCV library, and is available as an open-source library⁵⁷. It follows a modular design, enabling adaptability across different tracking methods by decomposing the execution flow into distinct, reusable components.

The software architecture of the framework is outlined next, with a focus on its modular components and execution flow.

Building on the ideas presented in section “[Framework design](#)”, the *Pipeline* class implements the functionality of the *Pipeline* entity, managing the execution of the tracking process and orchestrating a set of modular components. The execution follows a structured approach, where each module is responsible for a specific task, ensuring a clear separation of concerns and facilitating configurability.

At the core of this design is the *Pipeline* class, which implements the execution flow of the tracking process by orchestrating a set of modules. The execution is structured into three phases, *Initialization*, *Continuous execution*, and *Finalization*, ensuring a consistent operational flow. This structure is visually summarized in the activity diagram shown in Fig. 4, which outlines the sequential execution of modules and their interaction within the pipeline. Each module is responsible for a specific task within the pipeline and follows a standardized structure to facilitate integration and adaptability.

To enforce modularity, the framework defines a generic interface, *Module*, which serves as a common blueprint for all functional blocks. This interface establishes three fundamental methods: *initialize* for resource allocation and configuration, *execute* for performing the module's primary function, and *finalize* for resource deallocation. These methods align with the execution phases of the pipeline, ensuring coherence in module interactions.

A set of abstract classes inherit from *Module*, defining the essential steps of a generic tracking method. These abstract classes act as templates for concrete implementations, promoting reusability and allowing different tracking methods to be assembled through configurable module combinations. The *Pipeline* aggregates instances of these modules and sequentially invokes their respective *initialize*, *execute*, and *finalize* methods.

Through the use of multiple instances of *Pipeline*, the framework supports multiple tracking methods within the same execution environment. This capability enables comparative evaluations of different configurations without requiring significant modifications to the software structure.

Supplementary Figure 1 illustrates the hierarchy of classes and the relationships between the components of the framework.

Case studies

The case studies presented here are intended as proof-of-concept demonstrations, to illustrate the modularity of the framework and how different components can be integrated in a unified pipeline. Reference implementations aligned with these examples are also available in the public repository.

The Intel® RealSense™ D435i camera⁵⁸ was employed due to its array of sensors, enabling the implementation of diverse tracking techniques (Fig. 5). The camera is equipped with the OmniVision OV2740 RGB sensor,

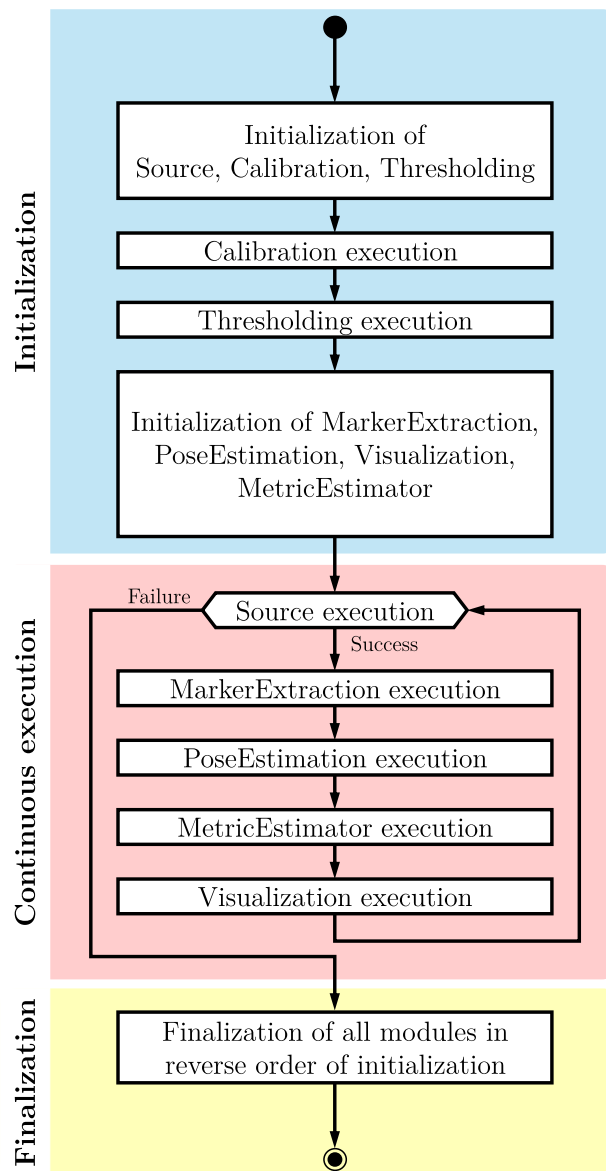


Fig. 4. Activity diagram of the execute method in the Pipeline class.

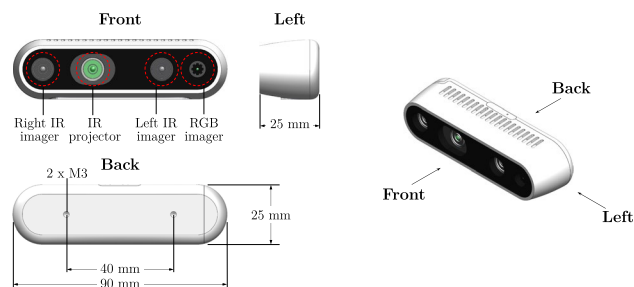


Fig. 5. Dimensional specifications and integrated imagers of the Intel® RealSense™ D435i camera, based on data from⁵⁸.

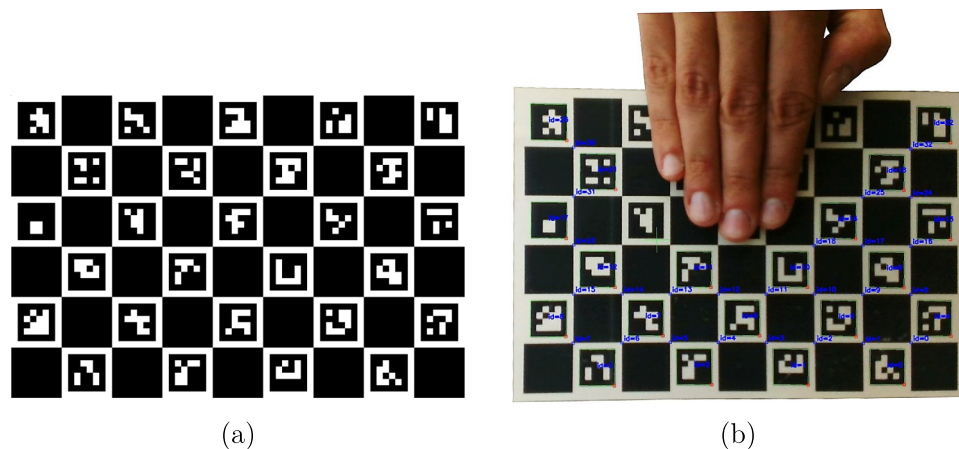


Fig. 6. (a) 6×9 calibration ChArUco board; (b) partially occluded ChArUco board during calibration process.

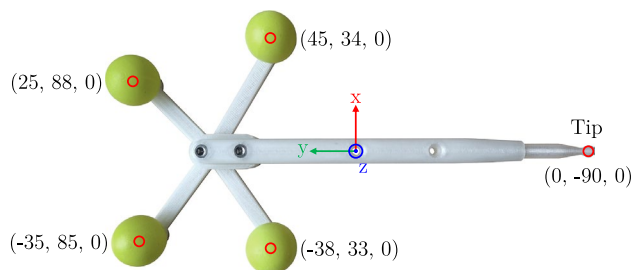


Fig. 7. Tool equipped with passive spherical markers.

which provides a resolution of 1920×1080 pixels with a 16:9 aspect ratio at a frame rate of 30 FPS, and it supports both 10-bit RAW RGB and YUY2 compressed stream formats.

To integrate the aforementioned camera with our framework, we implemented the `RGBMonoSource` class, extending the abstract `Source` class, which supports the use of monocular RGB cameras through the OpenCV API.

The intrinsic parameters of the camera, essential for pose estimation, are computed through a calibration process using the `CharucoCalibration` class, which extends the abstract `Calibration` class, and employs ChArUco boards⁵⁹ with the OpenCV library to achieve accurate and robust calibration. These boards consist of a checkerboard pattern where each white square contains an ArUco marker. The markers help uniquely identify the board's position and orientation, while additional calibration points enable accurate calibration even when parts of the board are occluded. Figure 6 shows the ChArUco board used for the calibration of both tracking methods.

The calibration process involves capturing multiple images of the board from different angles using the target camera. The `CharucoCalibration` class then computes the camera's intrinsic parameters through the `cv::calibrateCamera` function⁶⁰. In the case of stereo camera setups, which this class also supports, the process continues with the estimation of extrinsic parameters to model the spatial relationship between the cameras.

Passive spherical marker tracking

In the first method, a tool equipped with four fiducial spherical markers, coated with fluorescent matte yellow paint to enhance contrast, and measuring 20 mm in diameter, was utilized to improve detectability against the surrounding environment (Fig. 7).

The tool also featured a tip, which was the target to be localized after the pose estimation. In this case, tracking this type of marker requires isolating it within the frames through a segmentation process using the `HSVThresholding` class, a subclass of `Thresholding`. This class enables the isolation of the spherical markers on the tool by generating a binary mask that excludes all other elements. The `HSVThresholding` class allows for manual adjustment of the binary mask parameters by fine-tuning the minimum and maximum Hue, Saturation, and Value (HSV) values. This ensures that only the colors of interest are retained in the frames during the thresholding phase, which occurs after calibration and before the *continuous execution* phase (Fig. 8).

Marker detection is performed by the `SpheresMonoExtraction` class, which extends the abstract `MarkerExtraction` class. Following segmentation with the thresholding module, a blob detection algorithm

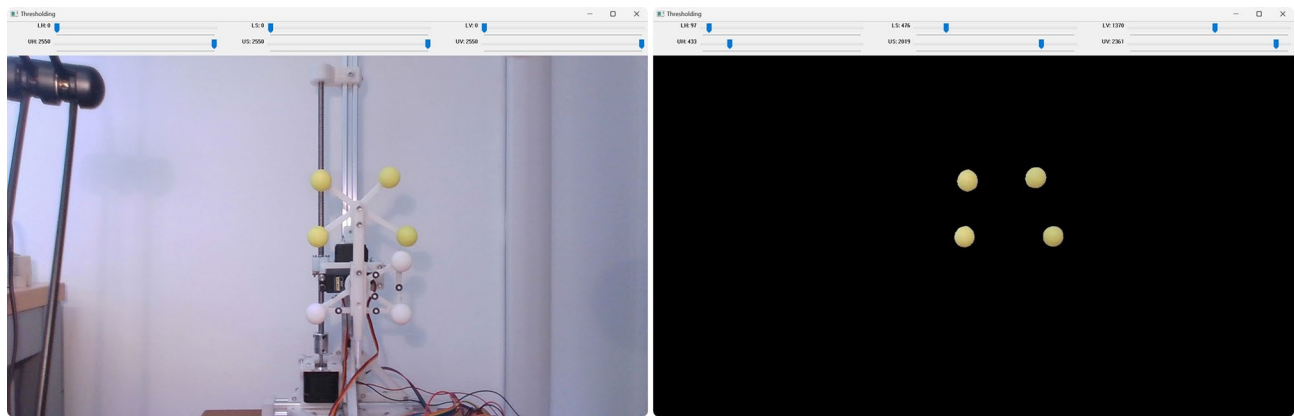


Fig. 8. HSVThresholding user interface: before (left) and after (right) thresholding.

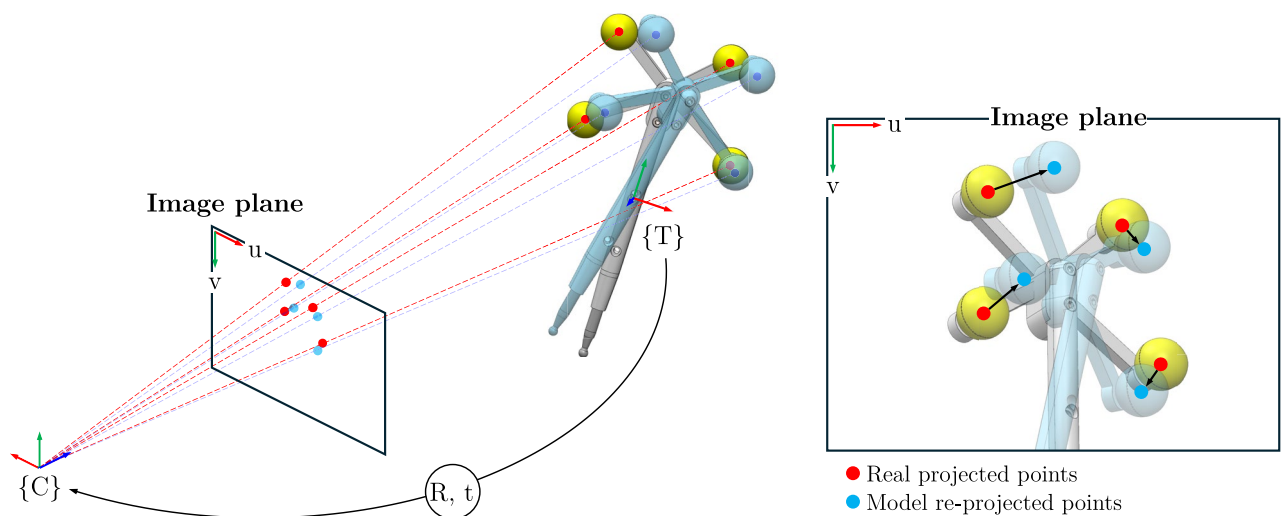


Fig. 9. Schematic representation of the Perspective-n-Point (PnP) problem for the tool with passive spherical markers. The real tool's markers are detected and projected onto the image plane (red dots). Knowing the 3D coordinates of the model's markers, Equation 1 is iteratively solved for $[R \mid t]$. The set of markers, transformed by R and t , representing the transformation from the tool's frame T to the camera's frame C , is reprojected onto the image plane (blue dots). The reprojection error, shown by black arrows, is iteratively minimized by recalculating $[R \mid t]$.

is applied to the source frame to identify 2D regions corresponding to potential tool markers. The centers of the minimum enclosing circles of the detected blobs, representing the marker positions, are then determined in the image-space (u, v) coordinates, with outliers filtered out to improve accuracy. Since the geometry of the tool is known, the 3D coordinates of the spherical marker centers with respect to the tool reference frame, along with their 2D coordinates in the uv image space, are available. This allows the pose estimation problem⁴ to be formulated as a Perspective-n-Point (PnP) problem. The PnP problem, coined by Fischler and Bolles⁶¹, is defined as follows: given a set of 3D points $P_i = (X_i, Y_i, Z_i)$ in the world reference frame and their corresponding 2D projections $p_i = (u_i, v_i)$ on the image plane, the goal is to find the extrinsic camera parameters, rotation R and translation t , that minimize the reprojection error between the 3D points projected into the image plane and their corresponding 2D points (Fig. 9):

$$s \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = K[R \mid t] \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \quad (1)$$

where K is the intrinsic camera matrix and s is a scaling factor, with both 3D and 2D points expressed in homogeneous coordinates.

The matrix R and vector t transform the 3D points from the world reference frame to the camera reference frame.

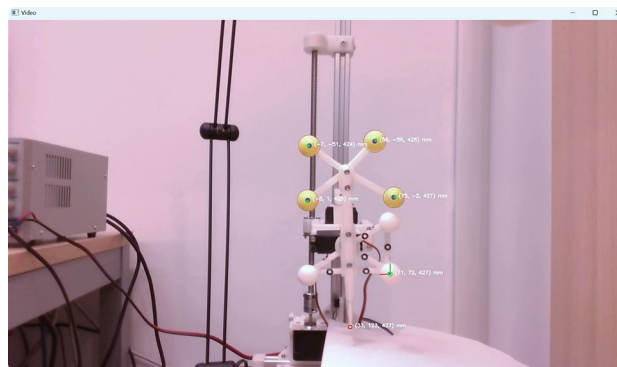


Fig. 10. ScreenView's user interface where circles indicate the positions of the detected markers, along with the reference frame used to define their 3D coordinates.

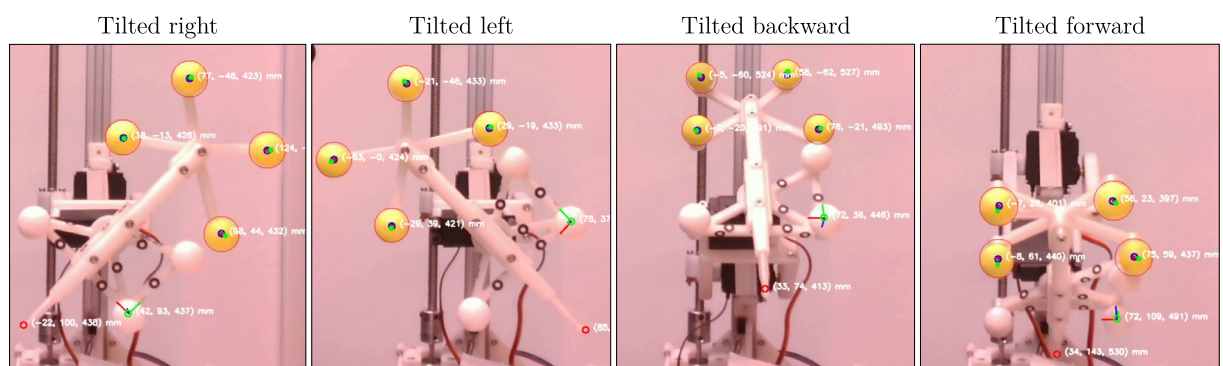


Fig. 11. Visualization of the estimated poses in four different spatial configurations of the instrument.

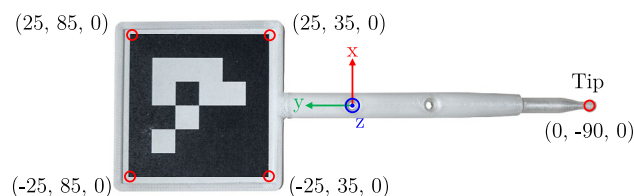


Fig. 12. Tool equipped with ArUco planar marker.

The `PnPAlignment` class extends the abstract `PoseEstimation` class to solve the PnP problem using the `cv::solvePnP` function from OpenCV. This function applies the RANdom Sample Consensus (RANSAC) method⁶¹ to robustly estimate the rotation matrix R and translation vector t , even in the presence of outliers and noise, while keeping the scale factor fixed at $s = 1$. To enhance tracking stability, the `PnPAlignment` class employs the `KalmanFilter` class, which inherits from `Filter`. Separate Kalman filters⁶² are applied to the translational and rotational components, reducing noise and improving robustness in real-time applications.

For visualization purposes, the tracking method utilizes the `ScreenView` class, which inherits from the `Visualization` abstract class. This class renders the live video feed with annotations from various modules and displays it in a dedicated window, as depicted in Fig. 10.

The tool was positioned and oriented in different configurations to qualitatively assess the correctness of the pose estimation and verify that the obtained values were consistent (Fig. 11).

Planar ArUco marker tracking

The second method uses a planar binary ArUco marker⁶³, measuring 50×50 mm, attached to a tool with a target tip (Fig. 12).

After converting the source frame to grayscale, the tracking process relies on detecting the marker's corners in the image. This is handled by the `ArUcoExtraction` class, which implements the `MarkerExtraction` abstract class leveraging the `ArucoDetector` class from the `aruco` module of the OpenCV library. Therefore,

this method employs the `NoThresholding` class, which performs no operations. In this case, no additional segmentation of the source image is required beyond what is performed by OpenCV APIs. Detecting the corners provides their coordinates in the image space (u, v). When combined with the known 3D coordinates of the marker's corners and the tool tip in the tool reference frame, obtained from the tool model, the pose estimation problem can again be formulated as a PnP problem. Consequently, the `PnPAlignment` class is used in this method as well.

As in the previous method, this approach also employs the `ScreenView` class, derived from the `Visualization` abstract class, for displaying the tracking output. The class overlays annotations on the live video feed and shows the result in a dedicated window, as illustrated in Fig. 13.

Evaluation

To evaluate the computational impact of the two tested methods, two subclasses of `MetricEstimator` were used: `FPSMonitor` and `ResourcesUsage`. These classes measure, respectively, the achieved frame rate (FPS) and the computational load in terms of CPU and RAM usage by the process.

Performance metrics were recorded while each tool remained stationary in front of the camera at a distance of 30 cm. The system used for these measurements was equipped with an Intel(R) Core(TM) i9-9900X processor, featuring 10 cores and a clock speed of 3.50 GHz, along with 32 GB of DDR4 RAM operating at 2666 MT/s.

Data were recorded over a 10 second interval, measuring Frames Per Second (FPS), CPU usage, and RAM load (Fig. 14). The red line represents the tracking method using the tool with passive spherical markers, while the blue line corresponds to the method with the tool equipped with ArUco planar markers. Both tracking methods demonstrate similar performance in terms of frame rate, maintaining a relatively stable FPS between 29 and 33 Hz, with occasional spikes reaching approximately 34 Hz. Neither method shows a clear advantage in terms of frame rate stability or average value. Regarding CPU usage, both methods show a consistent pattern of fluctuation between approximately 2% and 4%, with occasional spikes reaching 6%. The fluctuation patterns appear to be comparable between the two methods, with no substantial differences in average CPU utilization. However, the RAM load profiles show distinct differences between the two methods. The tool equipped with ArUco markers (blue line) initially starts at around 60 MB and steadily increases until approximately the 3-second mark, after which it fluctuates between approximately 68 MB and 75 MB for the remainder of the measurement period. In contrast, the tool equipped with passive spherical markers (red line) maintains a more consistent load at approximately 67 MB throughout the 10-second interval, with minimal variation. The higher memory usage of the ArUco method may stem from its pipeline, which includes contour extraction, corner refinement, and pattern decoding, unlike the simpler blob-based segmentation of spherical markers.

Figure 15 shows a diagram of the classes and the instances of two case studies.

Discussion

This work introduced the Modular Optical Tool Tracking (MOTT) framework as a modular environment for implementing and benchmarking optical tracking methods. The framework was designed through a requirement-based process, where Quality Function Deployment (QFD) was applied to identify, with the support of a focus group, the fundamental technical features that would guide its development. These features were then organized into five design drivers, which represent the functional building blocks of the framework. The implementation aimed to demonstrate that a lightweight and extensible architecture can facilitate the integration of diverse tracking modules and support preliminary comparative evaluations. This was illustrated through two case studies based on marker-based tracking methods using an RGB camera, which served as proof-of-concept demonstrations to validate the feasibility of the approach in real scenarios.

The requirements identified through the QFD tool were overall satisfied by the design drivers defined in the framework. The requirement of “Extensibility” and “Flexibility” was supported by the “Framework design” driver, which introduced a modular, object-oriented pipeline where modules can be added, replaced, or reconfigured with limited adjustments. This enabled different tracking methods to be instantiated within a common pipeline, as illustrated in the case studies. The “Source management” driver allowed the framework

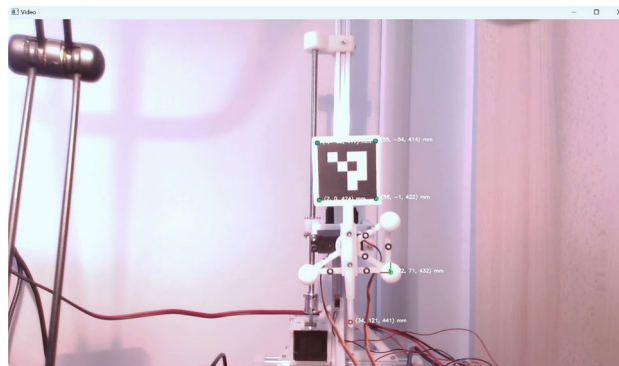


Fig. 13. ScreenView's user interface where circles indicate the positions of the detected corners, along with the reference frame used to define their 3D coordinates.

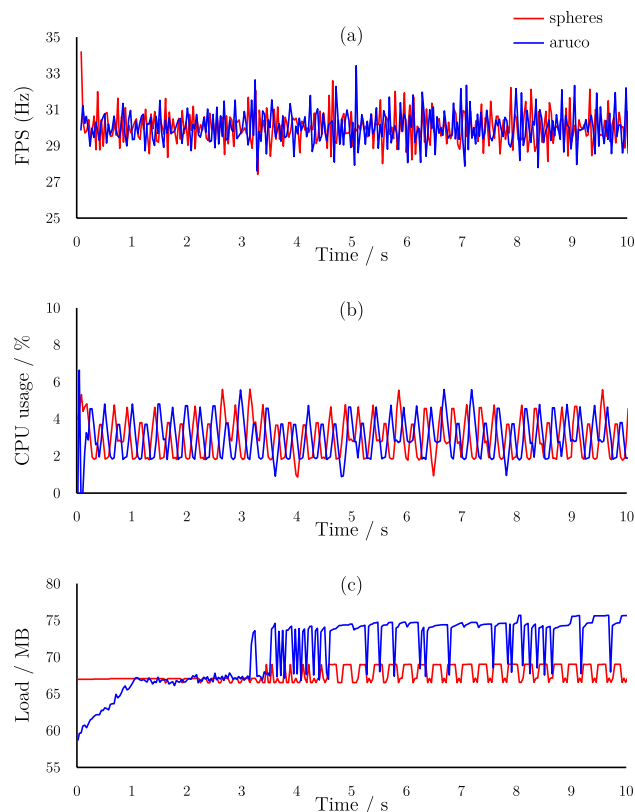


Fig. 14. Performance evaluation of both tracking methods while keeping the respective tool stationary in front of the camera at a distance of 30 cm: recorded values over 10 seconds for (a) FPS, (b) CPU usage, and (c) RAM load. The red and blue lines represent the tracking methods for the tools with passive spherical markers and ArUco planar markers, respectively.

to meet the requirement of “Integration ease”, by accommodating different video input configurations ranging from single- to multi-camera setups, while maintaining a unified data flow. The “Framework implementation” driver addressed the requirement of “Performance”, as it included optimization of computational footprint and processing time; this aspect was evaluated in the case studies through metrics such as frame rate, CPU usage, and memory load. The requirement of “Evaluability” was also addressed by the “Feedback management” driver, which provided visualization modules and metric reporting to support method comparison.

Previous studies on optical tracking largely focused on accuracy and hardware-dependent protocols. Khadem et al.³⁶ quantified jitter⁶⁴ using a mechanical platform, while Wiles et al.³⁷ highlighted how vendor accuracy assessment protocols may obscure non-uniform error distributions. These works contributed accuracy benchmarks, developed within the context of specific physical setups. In contrast, the present framework does not aim to establish intrinsic accuracy values for tracking methods. Accuracy evaluation in MOTT is instead left to the experimental metrics defined by the user, allowing benchmarks to be designed around accuracy when needed, or extended to computational and architectural aspects. Koivukangas et al.³⁹ and Kral et al.⁴⁰ compared optical and electromagnetic tracking in surgical contexts, emphasizing trade-offs between accuracy and robustness to line-of-sight occlusion. Their focus was modality comparison, whereas the present framework provides a software structure for integrating different methods under consistent runtime conditions. Similarly, Herregodts et al.³⁸ advocated dynamic benchmarking with moving targets to address limitations of static evaluations, whereas the present framework introduces a modular structure that could be reused to replicate benchmarking under different experimental conditions. Reviews and benchmarks in adjacent domains^{2,21,35,41–43} have underscored the need for standardized and reproducible pipelines. The present work is consistent with these observations but focuses specifically on optical tool tracking, where modular evaluation environments are still limited. Integration frameworks such as OpenIGTLink⁴⁴, PLUS⁴⁵, and middleware like ROS/ROS 2^{46,47} address interoperability and system-level orchestration. By contrast, the scope of this framework is confined to method-level evaluation, where interchangeable tracking modules can be implemented and compared within a unified software pipeline.

The study nonetheless presents some limitations. Although this was not the primary aim of the work, the presented case studies provide only a partial coverage of the wide variety of state-of-the-art optical tracking solutions, as they focused on marker-based methods relying on a similar pose estimation logic, leaving other approaches unexplored. In addition, the evaluation considered only computational performance (FPS, CPU, and memory usage), without addressing other important aspects such as static and dynamic accuracy, which places the study at an early stage of development. A further limitation concerns scalability: while the framework was

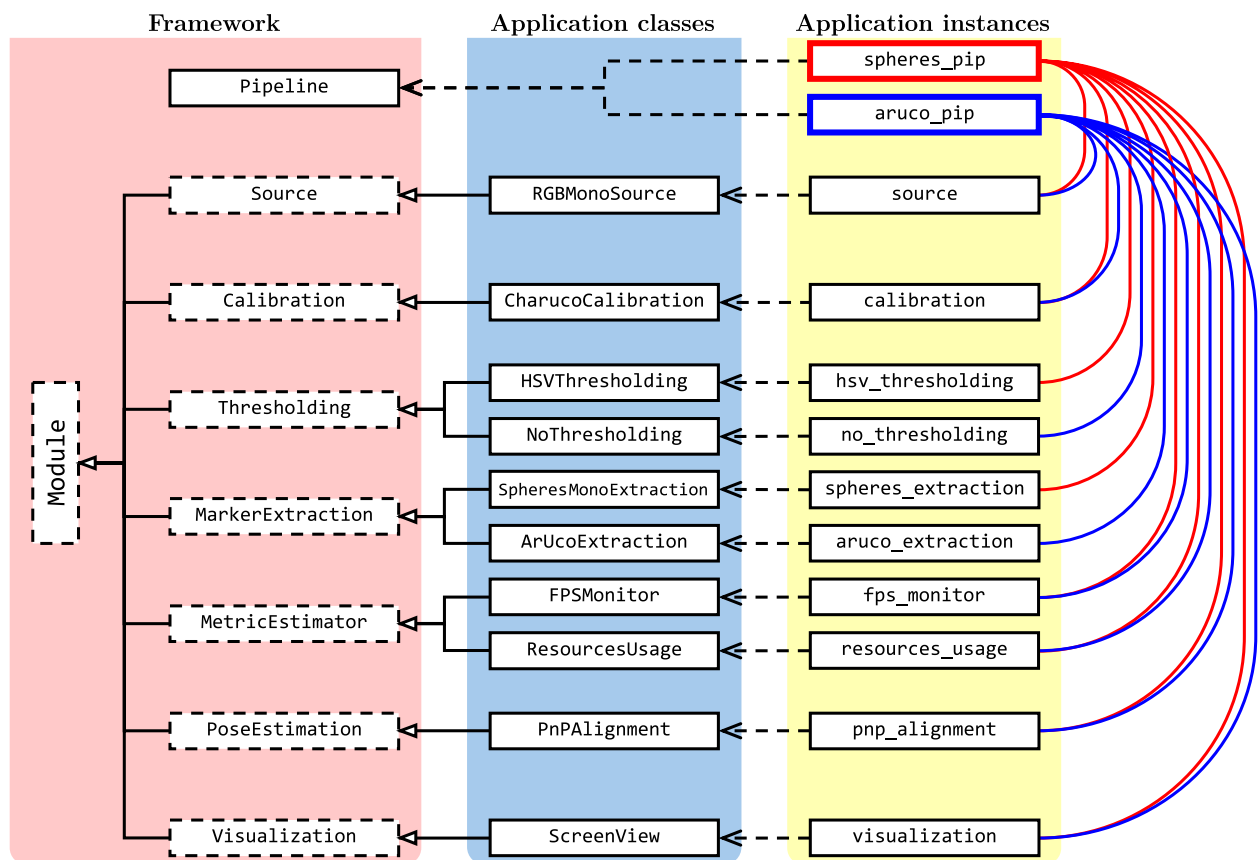


Fig. 15. Class and instance diagram related to the two case studies presented.

designed to support multi-sensor setups, synchronization and large-scale deployment remain untested. Finally, despite the cross-disciplinary expertise and effort of the focus group, it is acknowledged that not all requirements may have been fully captured in the initial QFD analysis. These factors may restrict the generalizability of the present findings and underline the preliminary nature of the contribution.

Future work will include the integration of additional optical tracking solutions based on different pose estimation logics, as well as markerless approaches. Validation across heterogeneous hardware platforms is also required, together with experiments designed to establish ground truth and to assess both static and dynamic accuracy. These investigations would help verify the framework's ability to adapt to diverse experimental scenarios with complex and varied evaluation metrics. A further line of development concerns multi-sensor configurations, where synchronization and scalability need to be implemented and evaluated to enable application in more demanding settings.

Conclusions

This work presented the Modular Optical Tool Tracking (MOTT) framework as a proof of concept for a modular environment dedicated to benchmarking optical tracking methods. The framework was designed through a requirement-based approach and tested in two case studies, confirming its feasibility for integrating and comparing alternative tracking modules. As a preliminary contribution, it establishes a foundation on which future work can test a broader range of optical tracking solutions and pursue more extensive validation studies.

Data availability

The source code supporting the findings of this study has been deposited at <https://github.com/tooltip-optical-tracking/mott-framework>. Additional details supporting this work are provided in the Supplementary Material.

Received: 7 June 2025; Accepted: 18 September 2025

Published online: 24 October 2025

References

1. Smeulders, A. W. M. et al. Visual tracking: An experimental survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **36**(7), 1442–1468. <https://doi.org/10.1109/TPAMI.2013.230> (2014).
2. Bouget, D. et al. Vision-based and marker-less surgical tool detection and tracking: A review of the literature. *Med. Image Anal.* **35**, 633–654. <https://doi.org/10.1016/j.media.2016.09.003> (2017).

3. Guan, J., Wang, Y., Song, Z., Chen, Q. & Li, B. A survey of 6dof object pose estimation methods. *Sensors* **24**(4), 1076. <https://doi.org/10.3390/s24041076> (2024).
4. Marchand, E., Uchiyama, H. & Spindler, F. Pose estimation for augmented reality: A hands-on survey. *IEEE Trans. Visual Comput. Graphics* **22**(12), 2633–2651. <https://doi.org/10.1109/TVCG.2015.2513408> (2016).
5. Ma, L., Huang, T., Wang, J. & Liao, H. Visualization, registration and tracking techniques for augmented reality guided surgery: A review. *Phys. Med. Biol.* <https://doi.org/10.1088/1361-6560/acaf23> (2023).
6. Zhang, Q., Yang, S., Liu, H., Xie, C., Cao, Y., & Wang, Y. Real-time implementation of a joint tracking system in robotic laser welding based on optical camera. In *Transactions on Intelligent Welding Manufacturing*, Vol. 1, 99–111 (Springer, Singapore, 2018). https://doi.org/10.1007/978-981-10-8330-3_6.
7. Salerno, F., Moos, S., Ulrich, L., Novaresio, A., & Vezzetti, E. A methodology for the dimensional and mechanical analysis of surgical guides. In *International Conference of the Italian Association of Design Methods and Tools for Industrial Engineering* 184–193 (Springer, Cham, 2023). https://doi.org/10.1007/978-3-031-52075-4_22.
8. Catalucci, S., Thompson, A., Piano, S., Branson, D. T. & Leach, R. Optical metrology for digital manufacturing: A review. *Int. J. Adv. Manuf. Technol.* **120**(7), 4271–4290. <https://doi.org/10.1007/s00170-022-09084-5> (2022).
9. Liu, Y., Li, Y., Zhuang, Z. & Song, T. Improvement of robot accuracy with an optical tracking system. *Sensors* **20**(21), 6341. <https://doi.org/10.3390/s20216341> (2020).
10. Peh, S. et al. Accuracy of augmented reality surgical navigation for minimally invasive pedicle screw insertion in the thoracic and lumbar spine with a new tracking device. *Spine J.* **20**(4), 629–637. <https://doi.org/10.1016/j.spinee.2019.12.009> (2020).
11. Müller, F. et al. Augmented reality navigation for spinal pedicle screw instrumentation using intraoperative 3d imaging. *Spine J.* **20**(4), 621–628. <https://doi.org/10.1016/j.spinee.2019.10.012> (2020).
12. Feudis, I. et al. Evaluation of vision-based hand tool tracking methods for quality assessment and training in human-centered industry 4.0. *Appl. Sci.* **12**(4), 1796. <https://doi.org/10.3390/app12041796> (2022).
13. Han, Z. et al. Collaborative human-robot surgery for mandibular angle split osteotomy: Optical tracking based approach. *Biomed. Signal Process. Control* **93**, 106173. <https://doi.org/10.1016/j.bspc.2024.106173> (2024).
14. Chen, L. et al. Object tracking in hyperspectral-oriented video with fast spatial-spectral features. *Remote Sens.* **13**(10), 1922. <https://doi.org/10.3390/rs13101922> (2021).
15. Nguyen, H. V., Banerjee, A., & Chellappa, R. Tracking via object reflectance using a hyperspectral video camera. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition—Workshops*, 44–51 (IEEE Computer Society, Los Alamitos, 2010). <https://doi.org/10.1109/CVPRW.2010.5543780>.
16. Ramesh, B., Zhang, S., Lee, Z. W., Gao, Z., Orchard, G., & Xiang, C. Long-term object tracking with a moving event camera. In *Proceedings of the British Machine Vision Conference (BMVC)*, 241 (2018).
17. Revuelta Sanz, P., Ruiz Mezcuca, B., & Sánchez Pena, J. M. Depth estimation—An introduction. In *Current Advancements in Stereo Vision* 93–118 (InTech, Rijeka. Chap. 5, 2012). <https://doi.org/10.5772/45904>.
18. Watanabe, M., Nayar, S. K., & Noguchi, M. N. Real-time computation of depth from defocus. In *Three-Dimensional and Unconventional Imaging for Industrial Inspection and Metrology*, Vol. 2599, 14–25 (SPIE, Bellingham, 1996). <https://doi.org/10.1117/12.230388>.
19. Park, K., Patten, T., & Vincze, M. Neural object learning for 6d pose estimation using a few cluttered images. In *European Conference on Computer Vision* 656–673 (Springer, Cham, 2020). https://doi.org/10.1007/978-3-030-58592-1_39.
20. Labbé, Y., Manuelli, L., Mousavian, A., Tyree, S., Birchfield, S., Tremblay, J., Carpentier, J., Aubry, M., Fox, D., & Sivic, J. MegaPose: 6D pose estimation of novel objects via render & compare. arXiv preprint [arXiv:2212.06870](https://arxiv.org/abs/2212.06870), <https://doi.org/10.48550/arXiv.2212.06870> (2022).
21. Hoque, S., Arafat, M. Y., Xu, S., Maiti, A. & Wei, Y. A comprehensive review on 3D object detection and 6d pose estimation with deep learning. *IEEE Access* **9**, 143746–143770. <https://doi.org/10.1109/ACCESS.2021.3114399> (2021).
22. Xiao, C., & Zhang, L. Implementation of mobile augmented reality based on Vuforia and Rawajali. In *2014 IEEE 5th International Conference on Software Engineering and Service Science*, 912–915 (IEEE, Piscataway, 2014). <https://doi.org/10.1109/ICSESS.2014.6933681>.
23. Ulrich, L., Salerno, F., Moos, S. & Vezzetti, E. How to exploit augmented reality (AR) technology in patient customized surgical tools: A focus on osteotomies. *Multimedia Tools Appl.* <https://doi.org/10.1007/s11042-023-18058-y> (2024).
24. Simon, J. Augmented reality application development using unity and vuforia. *Interdiscip. Descr. Complex Syst. INDECS* **21**(1), 69–77. <https://doi.org/10.7906/indexs.21.1.6> (2023).
25. Wu, H. et al. An accurate recognition of infrared retro-reflective markers in surgical navigation. *J. Med. Syst.* **43**(6), 153. <https://doi.org/10.1007/s10916-019-1257-x> (2019).
26. Nakazato, Y., Kanbara, M., & Yokoya, N. A localization system using invisible retro-reflective markers. In *Proceedings of the 2005 Machine Vision Applications Conference*, 140–143 (MVA Organization, Tokyo, 2005).
27. Olesen, O. V., Paulsen, R. R., Højgaard, L., Roed, B. & Larsen, R. Motion tracking for medical imaging: A nonvisible structured light tracking approach. *IEEE Trans. Med. Imaging* **31**(1), 79–87. <https://doi.org/10.1109/TMI.2011.2165157> (2012).
28. Wang, S., Cheung, B. & Ren, M. Uncertainty analysis of a fiducial-aided calibration and positioning system for precision manufacturing of optical freeform optics. *Meas. Sci. Technol.* **31**(6), 065012. <https://doi.org/10.1088/1361-6501/ab7a0f> (2020).
29. Azagury, D. E. et al. Image-guided surgery. *Curr. Probl. Surg.* **52**(12), 476–520. <https://doi.org/10.1067/j.cpsurg.2015.10.001> (2015).
30. Geisbüsch, A., Auer, C., Dickhaus, H., Niklasch, M. & Dreher, T. Electromagnetic bone segment tracking to control femoral derotation osteotomy—a saw bone study. *J. Orthop. Res.* **35**(5), 1106–1112. <https://doi.org/10.1002/jor.23348> (2017).
31. Rabbi, I. & Ullah, S. A survey on augmented reality challenges and tracking. *Acta Graphica: znanstveni časopis za tiskarstvo i grafičke komunikacije* **24**(1–2), 29–46. <https://doi.org/10.5555/102430> (2013).
32. Marchand, E., Spindler, F. & Chaumette, F. Visp for visual servoing: A generic software platform with a wide class of robot control skills. *IEEE Robot. Autom. Mag.* **12**(4), 40–52. <https://doi.org/10.1109/MRA.2005.1577023> (2005).
33. Singh, M., & Jagersand, M. Modular tracking framework: A software design for interactive object tracking. In: *2016 13th Conference on Computer and Robot Vision (CRV)*, 273–280 (IEEE, 2016). <https://doi.org/10.1109/CRV.2016.12>.
34. Collet, A., Martinez, M. & Srinivasa, S. S. The moped framework: Object recognition and pose estimation for multiple objects. *Int. J. Comput. Vis.* **112**(2), 172–193. <https://doi.org/10.1007/s11263-014-0783-4> (2015).
35. Fritz, J., Braun, J., Tönnis, M. & Klinker, G. Benchmarking and evaluation of tracking systems for augmented reality. *Comput. Graphics* **89**, 128–141. <https://doi.org/10.1016/j.cag.2020.04.010> (2020).
36. Khadem, R. et al. Comparative tracking error analysis of five different optical tracking systems. *Comput. Aided Surg.* **5**(2), 98–107. <https://doi.org/10.3109/10929080009148876> (2000).
37. Wiles, A. D., Thompson, D. G., & Fritsch, D. D. Accuracy assessment and interpretation for optical tracking systems. In *Medical Imaging 2004: Visualization, Image-Guided Procedures, and Display*, Vol. 5367, 421–432 (SPIE, 2004). <https://doi.org/10.1117/12.536128>.
38. Herregodts, S. et al. An improved method for assessing the technical accuracy of optical tracking systems for orthopaedic surgical navigation. *Int. J. Med. Robot. Comput. Assist. Surg.* **17**(4), 2285. <https://doi.org/10.1002/rcs.2285> (2021).
39. Koivukangas, T., Katisko, J. P. A. & Koivukangas, J. P. Technical accuracy of optical and the electromagnetic tracking systems. *Springerplus* **2**, 1–7. <https://doi.org/10.1186/2193-1801-2-90> (2013).
40. Kral, F., Puschban, E. J., Riechelmann, H. & Freysinger, W. Comparison of optical and electromagnetic tracking for navigated lateral skull base surgery. *Int. J. Med. Robot. Comput. Assist. Surg.* **9**(2), 247–252. <https://doi.org/10.1002/rcs.1502> (2013).

41. Dendorfer, P. et al. Motchallenge: A benchmark for single-camera multiple target tracking. *Int. J. Comput. Vision* **129**, 845–881. <https://doi.org/10.1007/s11263-020-01393-0> (2021).
42. Li, C., Xue, W., Jia, Y., Qu, Z. & Luo, B. Lasher: A large-scale high-diversity benchmark for RGBT tracking. *IEEE Trans. Image Process.* **30**, 5920–5935. <https://doi.org/10.1109/TIP.2021.3079712> (2021).
43. Pouw, C. et al. Benchmarking high-fidelity pedestrian tracking systems for research, real-time monitoring and crowd control. *Collect. Dyn.* **6**, 1. <https://doi.org/10.17815/cd.2021.134> (2022).
44. Tokuda, J. et al. Openigtlinc: An open network protocol for image-guided therapy environment. *Int. J. Med. Robot. Comput. Assist. Surg.* **5**(4), 423–434. <https://doi.org/10.1002/rcs.274> (2009).
45. Lasso, A. et al. Plus: Open-source toolkit for ultrasound-guided intervention systems. *IEEE Trans. Biomed. Eng.* **61**(10), 2527–2537. <https://doi.org/10.1109/TBME.2014.2322864> (2014).
46. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. Ros: An open-source robot operating system. In *ICRA Workshop on Open Source Software*, Vol. 3, 5 (2009).
47. Macenski, S., Foote, T., Gerkey, B., Lalancette, C. & Woodall, W. Robot operating system 2: Design, architecture, and uses in the wild. *Sci. Robot.* **7**(66), 6074. <https://doi.org/10.1126/scirobotics.abm6074> (2022).
48. Baklouti, S. On the improvement of ros-based control for enhanced robot simulation. *Appl. Sci.* **11**(16), 7190. <https://doi.org/10.3390/app11167190> (2021).
49. Rahbar, M. D. Toward intraoperative visual intelligence: Real-time surgical instrument segmentation for enhanced surgical monitoring. *Front. Med. Technol.* **6**, 10 (2024).
50. Akao, Y. Development history of quality function deployment. In *The Customer Driven Approach to Quality Planning and Deployment*, Vol. 339, 90 (1994).
51. Rumbaugh, J., Jacobson, I. & Booch, G. *The Unified Modeling Language Reference Manual* 2nd edn. (Addison-Wesley Professional, Boston, 2004).
52. Cohn, M. User Stories Applied: For Agile Software Development. Addison-Wesley Professional, Boston, MA (2004). Requirements process in agile software development.
53. Cornell University: QFD Guide - CESYS525. https://bpb-us-w2.wpmucdn.com/sites.coecis.cornell.edu/dist/0/221/files/2018/09/CESYS525_QFD-Guide-1mi5wud.pdf. Accessed: 2025-08-22 (2018).
54. Dasanayake, S., Markkula, J., Aaramaa, S., & Oivo, M. Software architecture decision-making practices and challenges: An industrial case study. In *2015 24th Australasian Software Engineering Conference (ASWEC)* 88–97 (EEE, Piscataway, 2015). <https://doi.org/10.1109/ASWEC.2015.20>.
55. Hofmeister, C. et al. A general model of software architecture design derived from five industrial approaches. *J. Syst. Softw.* **80**(1), 106–126. <https://doi.org/10.1016/j.jss.2006.05.024> (2007).
56. Herzwurm, G., Schockert, S., & Tauterat, T. Quality function deployment in software development—state-of-the-art. In *Proceedings of the 21th International Symposium on Quality Function Deployment (ISQFD)* 1–18 (2015).
57. Contenti, A., & Salerno, F. MOTT Framework: Modular Optical Tracking Toolkit (2025). <https://github.com/tooltip-optical-tracking/mott-framework>.
58. Corporation, I. Intel RealSense D400 Series Datasheet (2017). <https://www.intel.com/content/www/us/en/content-details/841984/intel-realsense-d400-series-product-family-datasheet.html>.
59. OpenCV: Detection of ChArUco Corners using OpenCV. Accessed: 2024-09-11 (2018). https://docs.opencv.org/3.4/df/d4a/tutorial_charuco_detection.html.
60. OpenCV: Camera Calibration and 3D Reconstruction. Accessed: 2024-09-11 (2024). https://docs.opencv.org/4.10.0/d9/d0c/group_calib3d.html.
61. Fischler, M. A. & Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* **24**(6), 381–395. <https://doi.org/10.1145/358669.358692> (1981).
62. Dorfmueller-Ulhaas, K. Robust optical user motion tracking using a Kalman filter. In *Intelligent Motion Tracking by Combining Specialized Algorithms* 173–182 (Springer, Berlin, Heidelberg, 2009). https://doi.org/10.1007/978-3-642-02809-0_19.
63. Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J. & Marin-Jiménez, M. J. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recogn.* **47**(6), 2280–2292. <https://doi.org/10.1016/j.patcog.2014.01.005> (2014).
64. Teather, R. J., Pavlovych, A., Stuerzlinger, W., & MacKenzie, I. S. Effects of tracking technology, latency, and spatial jitter on object movement. In *2009 IEEE Symposium on 3D User Interfaces*, 43–50 (2009). <https://doi.org/10.1109/3DUI.2009.4811204>.

Author contributions

F.S. conceptualized the work, developed the methodology, implemented the software, performed the validation, conducted the formal analysis and investigation, curated the data, handled the visualizations, and wrote the original draft. A.C. contributed to conceptualization, methodology, software development, formal analysis, validation, and original draft writing. L.U. contributed to conceptualization, methodology, validation, visualization, and original draft writing. G.M. performed formal analysis and validation, supervised the work, and contributed to review and editing. S.M. contributed to conceptualization, supervision, funding acquisition, and review and editing. E.V. provided supervision, project administration, funding acquisition, and contributed to review and editing. All authors reviewed and approved the final manuscript.

Declarations

Competing interests

The authors declare no competing interests.

Additional information

Supplementary Information The online version contains supplementary material available at <https://doi.org/10.1038/s41598-025-21044-z>.

Correspondence and requests for materials should be addressed to F.S.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2025

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).

Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.

These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.

We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.

While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval, sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.

These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.

To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.

Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.

If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com