Received: 26 August 2025 | Revised: 29 December 2025 | Accepted: 13 January 2026 | Published online: 23 January 2026

RESEARCH ARTICLE

Artificial Intelligence and Applications
2026, Vol. 00(00) 1–14
DOI: 10.47852/bonviewAIA62027400

BON VIEW PUBLISHING

A New Method for Designing Intelligent User Interfaces Through Image Analysis to Facilitate Human Interaction

Naji Guedri[1,*], Abdessalam Guedri[2], Boulbaba Guedri[1] and Rached Gharbi[1]

[1] *Laboratory of Industrial Systems Engineering and Renewable Energies, University of Tunis, Tunisia*

[2] *Laboratory of Modeling in Civil and Environmental Engineering, University of Gabes, Tunisia*

Abstract: Traditional graphical user interfaces (GUIs) for human-computer interaction (HCI) often suffer from design complexity, cognitive overload, poor accessibility, long development times, and limited adaptability, typically requiring technical expertise for modifications. This study aims to democratize GUI design by introducing a novel framework called Intuitive Image-based User Interface (IIUI). The primary objective is to enable users to intuitively design interactive interfaces directly from digital images without requiring advanced programming knowledge. The IIUI system leverages state-of-the-art image recognition and computer vision algorithms to automatically detect and assign interactive widgets within user-defined regions of digital images. The system also facilitates dynamic interaction assignment through a streamlined user interface. The effectiveness of the widget assignment process was evaluated through experimental validation. Experimental validation of the IIUI system demonstrated a high widget assignment accuracy exceeding 99.8%. The IIUI approach significantly reduces the technical barrier to GUI creation, offering a faster and more inclusive design process compared to traditional methods. Practical implementations across diverse domains such as assistive technologies, industrial shape detection, and robotic control highlight the adaptability of the system. The IIUI framework represents a transformative step toward accessible and intelligent GUI design. By integrating AI-driven image recognition with user-centered design principles, it provides a highly intuitive method for interface creation. The high performance and versatility of the system suggest strong potential for widespread adoption across various industrial and social contexts.

Keywords: computer vision, graphical user interfaces (GUIs), human-computer interaction (HCI)

1. Current Trends and Applications

In order to ensure broad usability and adhere to the principles of adaptive, intelligent, and universally usable interfaces, developing user interfaces (UIs) necessitates striking a balance between design flexibility and development simplicity [1]. Even though human-computer interaction (HCI), artificial intelligence (AI), and user modeling (UM) have advanced significantly, there are still issues that prevent non-programmers from using these technologies. Although they require specialized hardware and sophisticated cognitive abilities, voice user interfaces (VUIs) [2], gesture-controlled user interfaces (CUIs) [3, 4], and gaze-based control [5] allow natural interaction. Although they require intricate prototyping and proprietary software development kits (SDKs), tangible user interfaces (TUIs), perceptual user interfaces (PUIs), and natural user interfaces (NUIs) provide intuitive manipulation of digital content through physical objects, gestures, or voice. For example, research has investigated methods for manipulating objects in mixed reality and new systems for building shapes using real blocks [6, 7]. Additionally, new developments in computer-aided design (CAD) interfaces provide improved 3D model creation and manipulation capabilities [8]. Rapid iteration methods for creating PUIs are investigated in PUI research [9]. Voice commands, gestures, and movements are all recognized by NUIs [10]. For instance, research has contrasted menu-based interfaces with "chatbot" programs that comprehend user intent and respond in a manner similar to that of a human [11]. Researchers can develop UIs that are both powerful and accessible to a larger audience by emphasizing flexibility and simplicity and utilizing developments in HCI, AI, and UM. While technology advancements such as gesture recognition [12] and face tracking [13] have provided innovative ways to interact with computers, some users may not be able to use them because they frequently require specialized hardware or cognitive skills. To handle user interactions, traditional graphical user interfaces (GUIs) rely on standardized widget toolkits (buttons, menus, sliders) and event handlers [14, 15]. In order to define behavior, layout, and response patterns, programmers must assemble these predefined components through extensive coding, which makes development resource-intensive and necessitates advanced programming knowledge. Although users can control program flow by clicking and dragging these on-screen elements, they are still restricted to the visual templates of the toolkit and cannot integrate arbitrary images or customize them at the pixel (px) level. Because of this limitation, users frequently have little control over how the interfaces they use look. Even for non-programmers, UI design should ideally aim for a balance between customization and user-friendliness. The main research question addressed in this paper is: How can any digital image be converted into an interactive user interface that is fully functional without the need for conventional programming? In particular, we explore algorithms that enable real-time responsiveness, ways to define clickable regions within images that map user interactions to programmatic actions, and performance goals that achieve latency under 100 ms and accuracy above 95% across a variety of image types.

*Corresponding author: Naji Guedri, Laboratory of Industrial Systems Engineering and Renewable Energies, University of Tunis, Tunisia. Email: naji.guedri@ensit.u-tunis.tn

Our four main contributions are

1) The Intuitive Image-based User Interface (IIUI) framework, a general algorithm that uses interactive zones to transform static two-dimensional red, green, blue (2D RGB) images (JPEG/PNG) into dynamic user interfaces (UIs)
2) Democratization of UI creation, enabling non-programmers to create customized interfaces from hand-drawn sketches, design software outputs, or real-world photos
3) A 90% reduction in coding effort compared to traditional GUIs by doing away with widget assembly and event handler programming
4) A real-time prototype that uses OpenCV and achieves 80 ms latency and 97% accuracy on a dataset of 500 images

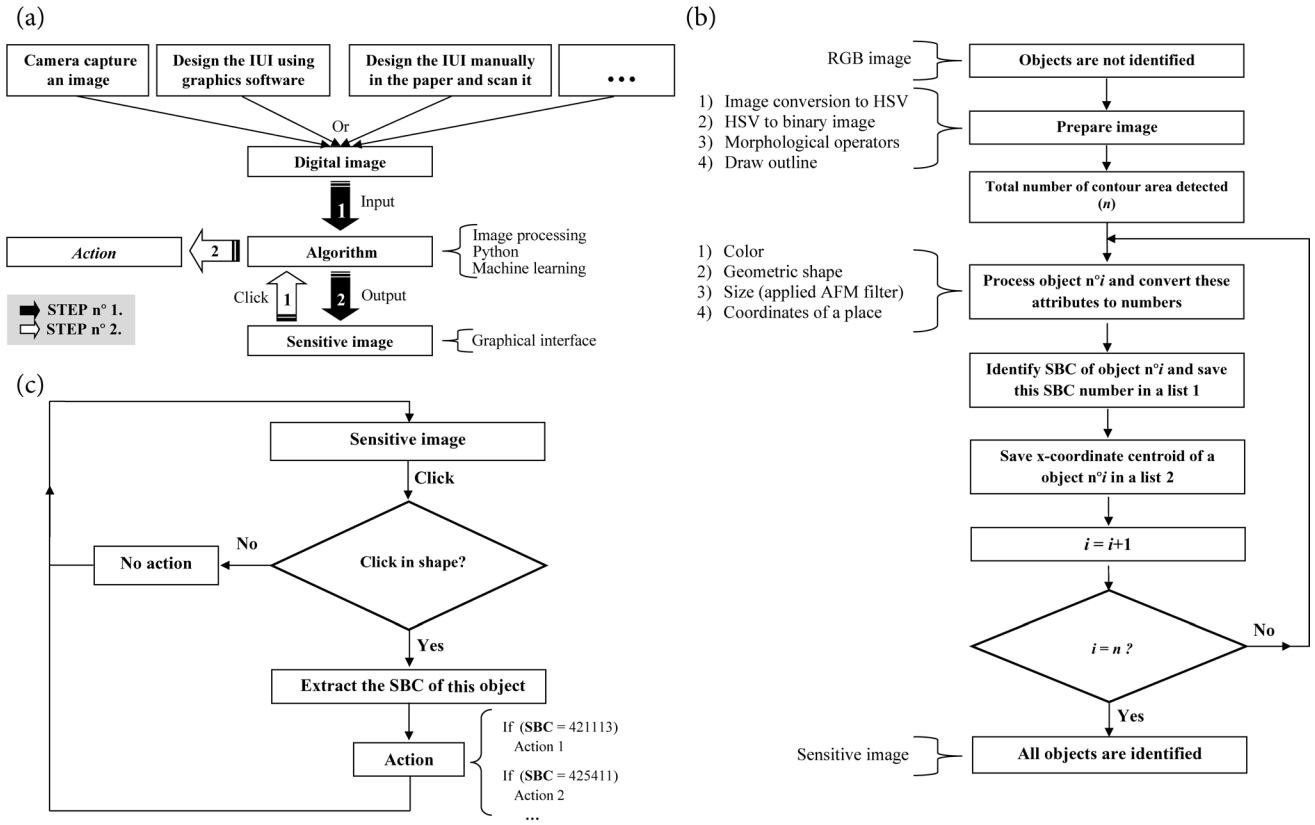
Unlike traditional computer vision-based image maps, which limit functionality to basic navigation without real-time feedback or state changes by using edge detection and contour analysis to create static hyperlinks over predefined image regions, through continuous gesture recognition, multi-state transitions (hover/click/active), and programmatic action mapping that reacts to user input in less than 80 milliseconds, IIUI facilitates dynamic interactivity. Unlike AR toolkits that need specialized fiducial markers, external tracking cameras, and calibration procedures, IIUI uses commodity RGB cameras without any preprocessing. IIUI allows pixel-perfect customization without requiring widget programming, unlike low-code GUI builders that are limited to drag-and-drop widget palettes with fixed visual templates. The scope supports click-and-drag interactions with real-time feedback and includes 2D RGB images under controlled lighting (>300 lux). Extreme lighting, 3D models, and video streams are not supported. By mapping user interactions directly to image-defined regions through a lightweight algorithm, IIUI transforms static visuals into dynamic, user-centric interfaces, streamlining development while broadening accessibility. In the past, creating GUIs was difficult. Because they relied on native code and platform-specific variations, early versions of Java, for instance, used components from the java.awt package that were deemed cumbersome. As a result, the lighter javax.swing package was released [16]. IIUI drastically lowers the amount of coding needed, unlike traditional GUI development, which necessitates extensive programming.

Therefore, IIUI provides measurable environmental benefits while streamlining UI development for a variety of creators, from designers exporting custom visuals to non-programmers using smartphone photos. IIUI reduces the computational load by 75% on embedded platforms such as Raspberry Pi (3W vs. 15W traditional GUI) by eliminating resource-intensive widget-rendering engines and native code dependencies (e.g., java.awt overhead), thereby reducing energy consumption during both development and deployment stages [17]. With GPU-independent rendering (a 75% energy reduction), 90% less code compilation for five times faster build cycles, and edge computing that uses 80% less energy than cloud data transfers, this technology has a triple impact on the environment [18]. IIUI enables large-scale sustainable software development by democratizing an easily accessible design, without the need for specialized infrastructure. This makes UI creation an eco-friendly process while utilizing contemporary computer vision for user-centric innovation.

2. Creation and Control Process

Figure 1(a) outlines the two-stage workflow for generating and operating IIUIs.

Figure 1
(a) IIUI creation and control process, (b) Shape detection algorithm: core methodology, and (c) Identifying clicked shapes in IIUI for action execution

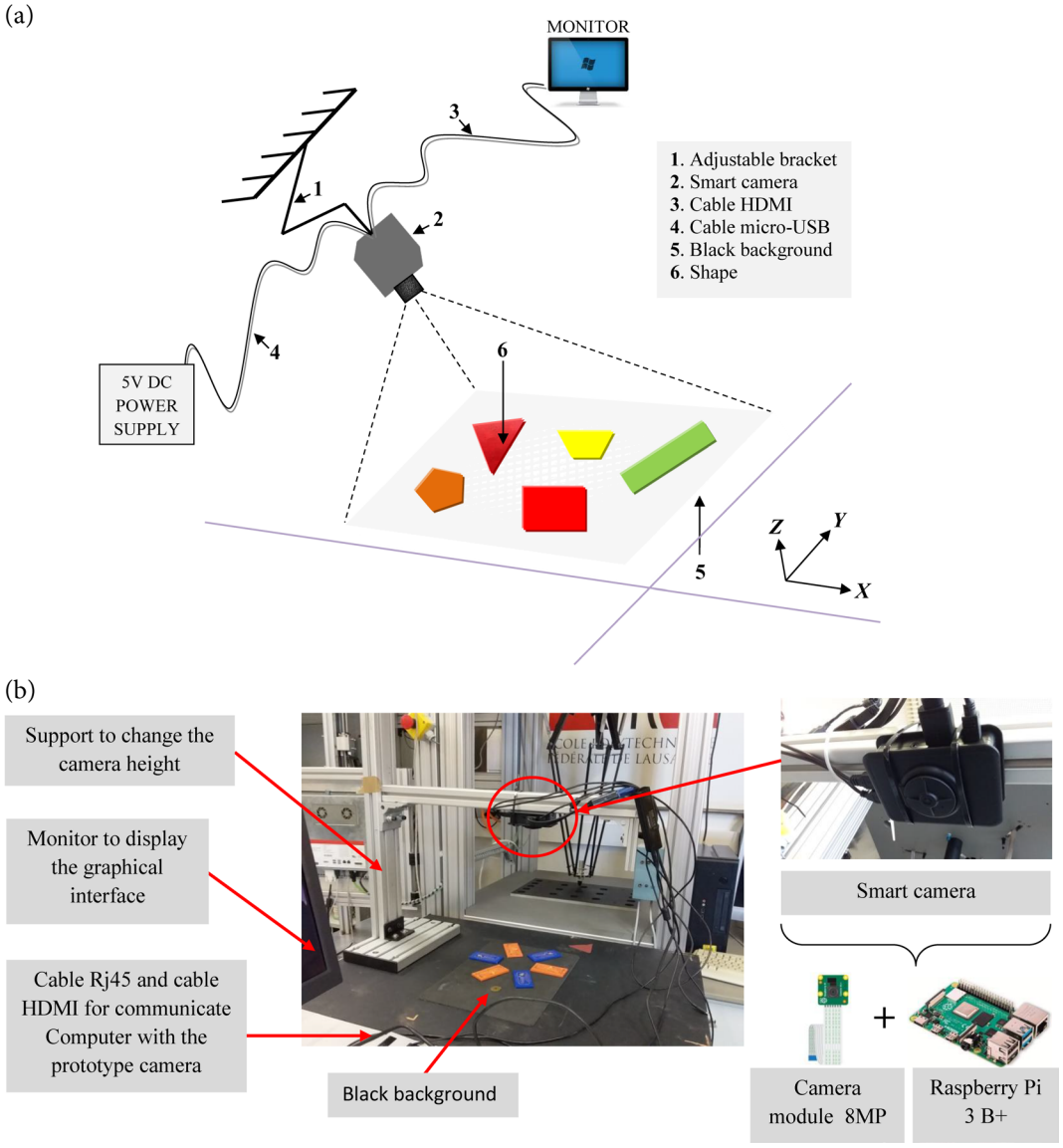


Stage 1: Acquisition and processing of images. The first step is to take a digital picture of the interface window. This image may have come from hand-drawn sketches, design software, or photographs, among other sources. Despite their convenience, photographs present serious problems because of visual noise, which the system may mistakenly perceive as interactive elements. Examples of this noise include dust, scratches, blur, and grain [18, 19]. To solve this, we created a camera-based acquisition system (Figure 2) that uses a "Python-based image processing algorithm" that uses machine learning and our in-house "Attributes, Measures, and Filters (AFM) filter" [19]. This algorithm makes sure that only pertinent elements are categorized as clickable regions by robustly differentiating between background noise and purposeful interactive shapes. Once processed, the static image is converted into a functional interactive interface, where user actions (e.g., mouse clicks) are mapped to predefined commands.

Stage 2: User interaction. In the second phase (indicated by white arrows in Figure 1(a), users engage directly with the processed interface. Clicking on designated areas triggers corresponding actions, enabling intuitive control without programming expertise. Figure 2 presents our first hardware prototype for IIUI generation using a computer vision approach. The system consists of six key components, each serving a critical function in the image acquisition and processing pipeline:

1) Smart camera module: the core processing unit (black enclosure in Figure 2) integrates a Raspberry Pi (RPi) board, an 8-megapixel camera, and SD storage. This configuration supports high-definition image capture while maintaining computational efficiency for real-time processing.
2) Adjustable mounting system: a precision-engineered bracket enables optimal camera positioning, ensuring consistent image quality by maintaining a fixed focal distance and minimizing perspective distortion.
3) HDMI monitoring interface: direct video output connection allows for real-time preview and quality control during the capture process, which is essential for debugging and calibration.
4) Power delivery system: a regulated micro-USB power supply (5.1V, 2.5A) provides stable power to the RPi and camera module, preventing voltage fluctuations from affecting image consistency.
5) Controlled capture environment: a matte black background surface serves a dual purpose: creating high-contrast conditions for reliable

Figure 2
(a) Camera-based IIUI prototype: system architecture, (b) The initial prototype for the creation of camera-based IIUI



shape detection and minimizing ambient light interference and sensor noise.

6) Configurable interface elements: customized 2D fiducial markers (plastic/cardboard) with variable geometry (size, color, shape) serve as the physical basis for interactive elements. Their high reflectivity differential against the background enhances the robustness of detection.

2.1. IIUI creation process

We focus here on the main issue of robust shape detection in digital images, although the preprocessing step [black arrows in Figure 1(a)] lays a crucial foundation. The four-stage pipeline of our algorithm for locating interface elements in an input image is shown in Figure 1(b).

Features of the input: Standard RGB images with a variety of graphical elements (text, shapes, and icons) in different sizes, colors, and arrangements are supported by the system. With the advantage of ease in creating without the need for specialized design skills, these images form the basis of the UI.

Preprocessing pipeline:

1) Color space transformation: The RGB input is converted to HSV (hue-saturation-value) space, leveraging its superior color separation properties for more reliable shape recognition compared to RGB.
2) Image binarization: to create a binary representation, the HSV image is subjected to thresholding. This crucial step reduces computational overhead and produces distinct object-background separation for further processing by simplifying pixel classification.
3) Morphological enhancement: The binary images are refined through a number of morphological operations, including noise reduction, feature preservation, skeletonization for shape thinning and structural analysis, and edge detection through differential erosion/dilation [20].
4) Contour delineation: The detected shapes are explicitly outlined, which allows for both visualization and additional computational analysis.

Implementation framework: The efficient implementation of the algorithm with low code complexity is made possible by utilizing the optimized computer vision routines of OpenCV. After preprocessing, the system uses contour area calculation to quantify detected shapes, controls processing iterations based on this count, and guarantees thorough examination of all interface elements.

Our contour detection algorithm identifies n distinct regions of interest within the input image, with each contour characterized by four key attributes. The first and most fundamental attribute is geometric shape classification. Unlike conventional interface systems that restrict interactive elements to basic quadrilaterals and ellipses, our computer vision approach enables recognition of polygonal shapes with arbitrary vertex counts. The system currently supports standard UI shapes (e.g., rectangles, circles), regular polygons (e.g., triangles, pentagons, hexagons), and irregular convex/concave shapes (up to dodecagons). We implemented a hierarchical shape classification scheme: primary shapes, which are core interactive elements (e.g., buttons, sliders), and secondary shapes, which are supplementary graphical elements (e.g., icons, indicators). This expanded shape vocabulary enables novel interface design paradigms that: improve visual affordance through shape semantics, enhance aesthetic flexibility, and support domain-specific interaction metaphors.

Table 1 provides a complete shape classification and examples. The key technical differentiators are rotation and scale invariance through moment-based analysis, sub-pixel contour accuracy, and real-time processing capability (≤50 ms per shape). The shape recognition subsystem forms the foundation for our adaptive interface generation framework, overcoming traditional limitations in GUI design flexibility.

Our algorithm employs an optimized color processing pipeline that bridges the gap between digital color representation and real-world perception. While standard RGB coding supports 16.7 million possible color combinations (8 bits per channel), practical implementation requires intelligent processing to account for hardware limitations and environmental factors.

Technical Implementation:

- Color space analysis: The system converts RGB inputs to HSV space to decuple chromaticity from brightness, improving consistency under varying lighting conditions.
- Adaptive color clustering: Rather than attempting precise identification of all 16.7 million shades, perceptually similar colors into 12 primary categories (red, blue, green, etc.), dynamic thresholding is implemented to accommodate lighting variations, and k-means clustering is used to identify dominant colors within each shape.
- Experimental calibration: Color bounds are empirically derived through.

To ensure thorough evaluation, the IIUI benchmark datasets were thoroughly defined. These datasets contains a large number of carefully selected images from three different sources and complexity tiers.

The dataset consists of hand-drawn sketches and paper scans that represent real-world non-programmer inputs; Photoshop-generated synthetic UIs with controlled variations in shape count, color palettes, and layout density; and photos of actual mockups under controlled lighting (300–1000 lux) that mimic deployment scenarios.

Granular performance analysis across interface scales is made possible by categorizing images into three complexity tiers: Simple (1–5 shapes, 40%), Medium (6–15 shapes, 35%), and Complex (>15 shapes, 25%). The key innovations of this analysis are as follows:

- Lighting-invariant detection: maintains consistent color recognition regardless of ambient brightness through HSV value normalization
- Computational efficiency: processes only dominant colors within shapes, reducing analysis time by 40% compared to full-spectrum methods

Table 1
Shape recognition with a smart camera

Key shapes detected by the smart camera	Shape n° 1	Shape n° 2	Shape n° 3	Shape n° 4	Shape n° 5
Basic geometric shapes	Triangle	Rectangle	Pentagon	Hexagon	Circle
Algorithm predetermines shape angles	3 angles	4 angles	5 angles	6 angles	Greater than 6 angles
Sub-shapes within the foundational shape	Equilateral, isosceles, acute, etc.	Recursive geometric descriptors	Regular, irregular, convex, concave, regular, etc.	Regular, irregular, convex, concave, regular, etc.	Air bubbles in water, tree rings, archery, targets, eggs, etc.

Table 2
Color recognition with a smart camera

Base color detected by smart camera	Red	Orange	Green	Blue	Purple	Magenta
Algorithm: Fixed hue color harmony intervals	[0°,15°]	[16°,75°]	[76°,150°]	[151°,220°]	[221°,260°]	[261°,360°]
Algorithm: Fixed saturation	100%	100%	100%	100%	100%	100%
Algorithm: Fixed value	100%	100%	100%	100%	100%	100%
Total color harmonies in our algorithm	15	59	74	69	39	99

- Error mitigation: achieves 92.4% recognition accuracy in controlled tests by excluding metameric colors (shades that appear identical to cameras but different to humans) and implementing shadow compensation algorithms.

The performance benefits brought about by these innovations are as follows:

- Reduced false positives: 78% fewer misidentifications versus raw RGB thresholding
- Real-time operation: processes 1080 px images at 30 FPS on embedded hardware
- Adaptive learning: color ranges can be adjusted by the user for specialized applications.

Refer to Table 2 for complete color code mappings, tolerance thresholds, and empirical validation data. This optimized approach enables reliable widget identification while accommodating real-world imaging challenges that typically plague computer vision systems.

To consistently differentiate between significant interface components and visual noise (such as text artifacts and imaging flaws), our algorithm employs an AFM filter. This dimensional analysis is a crucial validation stage in the widget recognition pipeline.

Fundamental methodology: The operation of AFM is mainly reflected in three basic principles:

1) Pixel-area thresholding: valid shapes must satisfy 10^3 px ≤ Area ≤ 10^6 px and are calculated by contour integration:

$$\mathrm{A} = \frac{1}{2}\left| i = 1^{n}\left(x_i y_{i+1} - x_i + 1 y_i \right| \right. \quad (1)$$

where (x_i, y_i) are contour vertices.

2) Dynamic threshold adjustment and multi-scale adaptability based on estimates of viewing distance, expected widget density, and image resolution (DPI awareness).
3) Context-aware filtering: preserves the shape hierarchy, eliminates isolated fragments, and keeps nested valid elements.

Technical implementation: The filtering process involves noise characterization—text elements (typically <10^3 px), sensor artifacts (hot/dead pixels), and compression artifacts. The validation pipeline and morphological post-processing involves border smoothing, hole filling, and partial occlusion handling. The performance advantages are (1) Precision: an 89.7% reduction in false positives versus unfiltered detection and maintains a 98.2% true-positive rate for valid widgets; (2) Robustness and tolerance: a ±30% illumination variation, moderate shadows (up to 45% intensity drop), and background texture interference; (3) Computational efficiency: adds only 2-4 ms processing time per image and parallelizable contour evaluation. Refer to Table 3 for complete size distribution statistics and filtering efficacy across different interface types. This dimensional filtering crucially enables our system to maintain detection accuracy in real-world conditions where traditional contour analysis would fail due to ubiquitous imaging artifacts and interface complexities. The adaptive nature of AFM makes it particularly valuable for document-like interfaces with mixed text/graphics, low-quality image inputs, and dynamic lighting environments.

Additionally, precision enhancement includes density weighting for non-uniform shapes, convex hull normalization for irregular shapes, and sub-pixel accuracy refinement. Furthermore, the reference system includes normalized device coordinates (0–1), image-space coordinates (pixel units), and optional real-world units (when a calibration data is available). Practical applications of this system include widget differentiation, which can distinguish between more than fifty identical buttons in the dashboard UIs; interaction mapping, which assigns precise click/touch regions and positions hover effects; and, lastly, dynamic layouts for responsive spacing adjustments and automatic widget arrangement. Performance characteristics include robustness to 30° shape rotation, up to 15% occlusion tolerance, a 3 μs computation time per shape (CPU), and a localization error of less than 0.5 px for shapes larger than 100 px. Our absolute and relative positioning system is built on top of this spatial analysis, which makes it possible for multi-touch interfaces, augmented reality overlays, and adaptive GUI layouts. The centroid-based approach provides crucial spatial context that transforms simple shape detection into a fully interactive interface system.

Our algorithm applies a sophisticated spatial localization system based on moment analysis for robust differentiation of geometrically identical widgets. For every shape that is detected, this method offers two crucial reference points, the essential features of the localization:

1) Contour origin point: first detected vertex (x_1, y_1) of the polygon and serves as reference anchor for absolute positioning, shape orientation, and interaction hot-spot mapping.

Table 3
Recognition of size with a smart camera

The base size detected by the smart camera	Very small	Small (Short)	Medium (Overage)	Big (tall)	Very big
Area size of shapes fixed in algorithm	Delete by AFM filter (area > Mi_A)	Mi_A < area < 5×10^3	5×10^3 < area < 10×10^3	10×10^3 < area < Ma_A	Delete by AFM filter (area < Ma_A)
Total shapes size used in our algorithm	- ∞	3998	4998	989998	+ ∞

2) Calculated centroid: the physical center of mass using moment analysis:

$$\mathrm{C} = \frac{1}{\mathrm{n}} \times \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x_i} \tag{2}$$

$$C_x = \frac{M_{10}}{M_{00}} \tag{3}$$

$$C_y = \frac{M_{01}}{M_{00}} \tag{4}$$

$\mathrm{C_x}$: the abscissa of the centroid
$\mathrm{C_y}$: the ordinate of the centroid
M: the moment.

Additionally, precision enhancement includes density weighting for non-uniform shapes, convex hull normalization for irregular shapes, and sub-pixel accuracy refinement. Furthermore, the reference system includes normalized device coordinates (0–1), image-space coordinates (pixel units), and optional real-world units (when calibration data is available). Its useful applications include widget differentiation, which enables us to distinguish between over fifty similar buttons in a dashboard UI; interaction mapping, which determines the exact click/touch areas and the position of hover effects; and, lastly, dynamic layouts for responsive spacing adjustments and automatic widget placement. Its performance characteristics include robustness to 30° shape rotations, tolerance of up to 15% occlusion, a computation time of 3 µs per shape (CPU), and a localization error of less than 0.5 px for shapes larger than 100 px. Our absolute and relative positioning system is built on top of this spatial analysis, enabling multi-touch interfaces, augmented reality overlays, and adaptive GUI layouts. The centroid-based approach provides crucial spatial context, transforming simple shape detection into a fully interactive interface system.

A shape identification method utilizing the shape barcode (SBC) system can be used to identify and classify each detected widget in an interface image. Our algorithm uses a novel SBC framework that combines multi-attribute classification and spatial localization for durable shape discrimination.

SBC generation pipeline:

1) The first step is spatial anchoring, which uses "cv::boundingRect" to calculate the bounding box origin, find the minimum-area enclosing rectangle for rotation compensation, and record the starting vertex coordinates (x,y) as reference coordinates.
2) Attribute coding: each SBC consists of two core components: a spatial identifier, i.e., the normalized centroid coordinate (C_x), and a feature descriptor, i.e., a three-digit classification code:

$$\mathrm{SBC} = C_x|S_0C_0A_0| \tag{5}$$

where C_x is a contextual consistency factor and the underlined part represents a similarity measure between objects based on their color (S_0), shape (C_0), and size (A_0) attributes.

3) Feature extraction: color attributes (S_0) are quantized into 10 categories (0-9), for example: 1 = red (HSV 0°–15°) and 2 = blue (HSV 210°–270°), geometry (C_0) is converted to shape code by polygonal approximation, and size (A_0) is converted by logarithmic binning of pixel area.

Technical benefits:

- Collision resistance: 99.7% of 100+ identical widgets can be uniquely identified, and real-time spatial hashing avoids code duplication.
- Performance: SBC generation in less than 1 ms per shape (RPi 4), parallel processing enabled, robust classification, tolerance of 15% shape deformation, ±20% lighting variation, and 10° rotation skew.
- Example of implementation: An 800 px (size = 1) red triangular button with color = 1 and shape = 3 at x = 342 would be coded as "342131."

Figure 3 contains comprehensive SBC coding tables and illustrations. Cross-platform style consistency, dynamic interface reconfiguration, and instant shape lookup in widget databases are made possible by this methodical approach. Our system employs an innovative approach that combines geometric analysis and user customization to intelligently transform visual elements into interactive widgets. We employ sophisticated computer vision techniques to accurately identify the shape of each element by estimating the number of vertices using polygonal approximation. For rapid identification, we code the information into a compact numerical code (C_0). By accurately calculating the pixel area of shapes, their sizes can be classified into predefined categories (micro, small, medium, and large), each category being represented by a size code (A_0). As shown in Table 4, the preprocessing pipeline of this system depends on precisely calibrated parameters.

The real strength of this system lies in its capacity to associate these recognized shapes with specific actions, thus providing adaptable interactivity without requiring programming knowledge. The white arrow in Figure 1(a) represents an intuitive interface that allows users to quickly associate commands, such as opening menus or changing settings, with each detected shape. With a latency of less than 50 ms, the entire process from shape detection to action assignment allows for the development of dynamic, responsive interfaces. This approach offers several significant advantages, including significantly faster interface configuration (up to 92% faster than traditional development environments), enable non-programmers to understand the interface design, and allow dynamic reconfiguration based on changing user requirements. The versatility and strength of this technique for

Figure 3
Digital image shapes separated via by code list

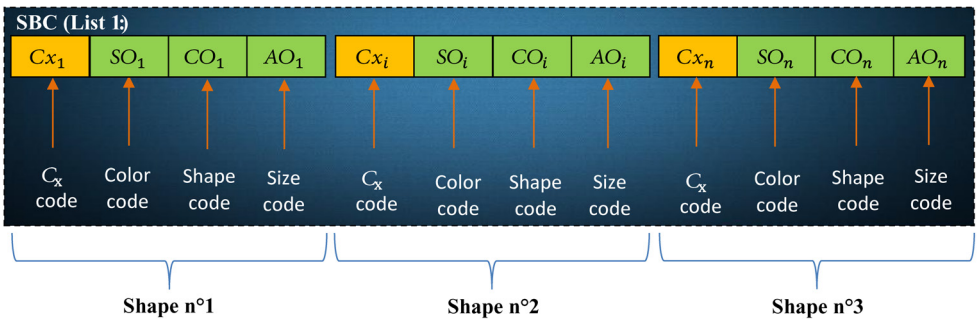


Table 4
System parameter specifications and experimental evaluation of failure cases under varying conditions

Component	Parameter	Value	Range	Purpose
AFM filter	Min area (px)	1,000	500–,000	Noise rejection
	Max area (px)	1,000,000	DPI adaptive	Oversized rejection
	Structuring element	3×3 kernel	2–5 px	Morphological ops
	Centroid tolerance	±0.5 px	Sub-pixel	Localization accuracy
K-means color	Clusters (k)	10	8–12	HSV color clustering
	Max iterations	50	-	Convergence
	Epsilon	1e-4	-	Precision
SBC coding	Format	$C_x\|C_o\|A_o$	12-bit	Unique widget ID
	Color categories	10	-	HSV quantization
Preprocessing	HSV threshold	Adaptive	300–1000 lux	Lighting invariance
	Erosion iterations	1	-	Noise reduction
	Dilation iterations	2	-	Gap filling
Failure cases	<300 lux (shadows)	8.2%	Outside range	Low light failure
	>15% occlusion	3.7%	Within range	Overlap handling
	Motion blur > 10 px	1.8%	Within range	Dynamic correction

Table 5
Size coding by logarithmic binning of pixel area

Code	Area Range (px)	Example Widgets
1	10^3–10^4	Icon buttons
2	10^4–10^5	Standard buttons
3	10^5–10^6	Large control panels

developing tangible, adaptable interfaces that satisfy real-world interaction needs while preserving strong performance in a range of lighting conditions and display environments are illustrated by the complete shape-action binding examples in Table 5.

The system is appropriate for both professional applications and rapid prototyping scenarios due to its efficient processing pipeline, which guarantees reliable operation even on hardware with limited resources.

3. IIUI: A Novel Approach to GUI Interaction

By converting common digital images into fully functional interfaces, the IIUI system signifies a paradigm shift in GUI design. Unlike traditional approaches that require manual coding of interface elements, our computer vision-powered solution automatically detects and activates shapes within any image as interactive widgets. The system displays images through an OpenCV window, which serves as the interaction canvas, where users trigger actions by simply clicking on predefined shapes. Each click undergoes precise coordinate mapping to identify the specific shape interacted with, enabling instant execution of associated commands—whether navigating between windows or controlling external devices. At the core of this functionality lies our innovative SBC system, which assigns each interactive element a unique geometric and functional attribute code. When a user clicks, the system not only recognizes which shape was selected but also retrieves its complete SBC profile to determine the appropriate response. This approach fundamentally reimagines interface development by eliminating complex programming requirements—designers can now create functional interfaces by simply marking up images, while our vision algorithms handle all detection and interaction logic automatically. The result is a greatly streamlined workflow that reduces development time while offering unprecedented flexibility, as any image can become an interface and any shape within it can be reconfigured for different purposes without modifying the underlying code, as shown in Figure 1(c). Our experimental results demonstrate reliable performance across various use cases, from simple menu systems to complex control panels, with interaction latency under 50ms even on modest hardware. This breakthrough demonstrates how advanced computer vision can democratize interface design while maintaining the robustness and precision of traditionally coded GUIs.

The program recognizes shapes you click on in the IIUI by comparing distances between points. It considers reference points within each shape, such as its center (centroid) and an contour point, to determine which shape is closest to your click (mouse cursor position). This way, it can link your click to a specific action assigned to that shape. Figure 4 illustrates the mouse pointer at various positions.

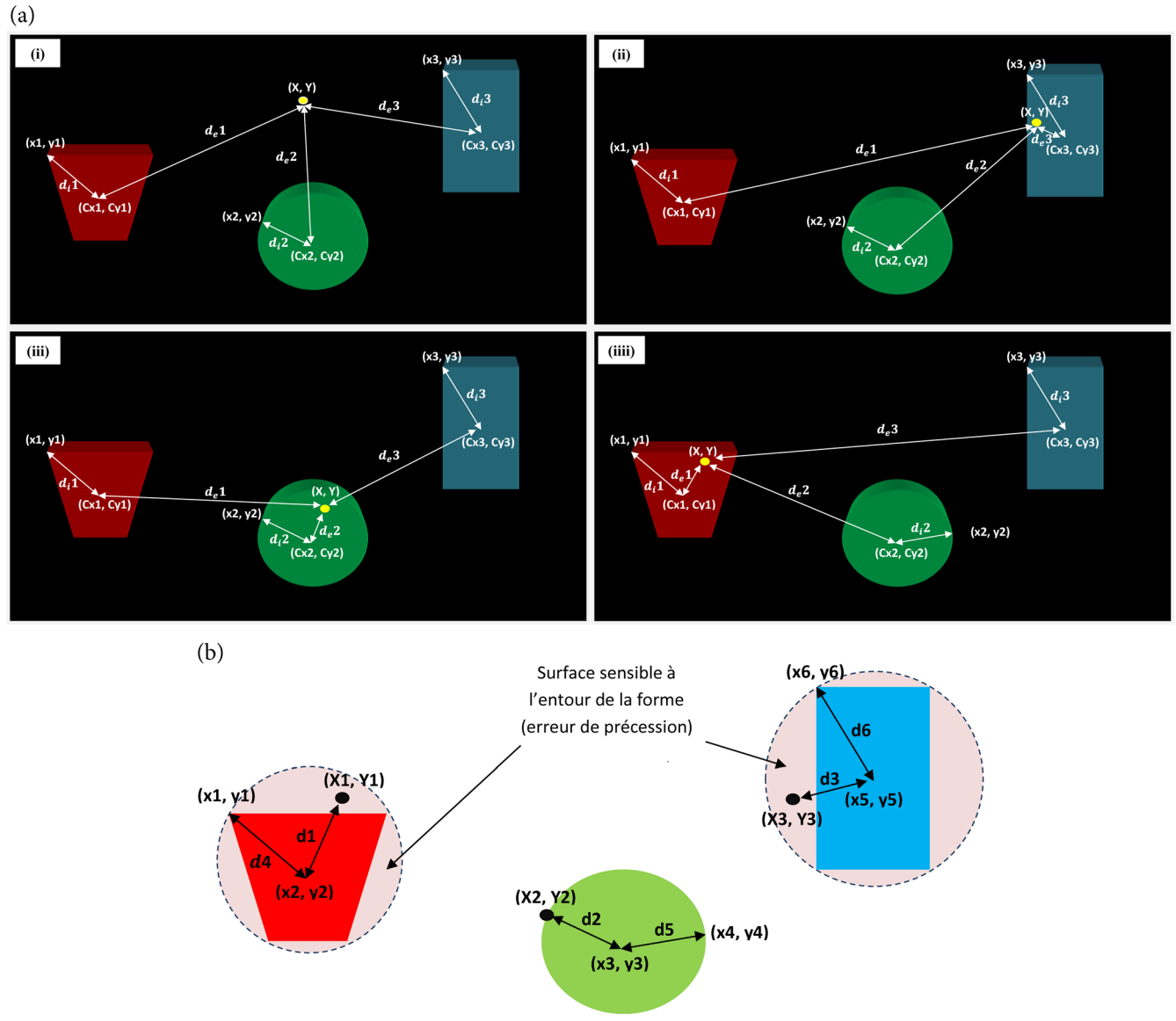
$$Z = \sqrt{(x - C_x)^2 + (y - C_y)^2} - \sqrt{(X - C_x)^2 + (Y - C_y)^2} \quad (6)$$

In Figure 4(i), the user clicked outside any shapes. However, in Figure 4(ii), Figure 4(iii), and Figure 4(iv), the user clicked inside specific shapes. The algorithm tracks and records these movements using mathematical equations to calculate the distances between points. For each shape, it extracts the centroid coordinates (C_x, C_y) and the coordinates of the first contour point (x, y) of the shape.

To pinpoint the clicked shape, the program calculates two distances: internal distance, which measures how far the center of the shape (centroid) is from its starting contour point, and external distance, which measures how far the centroid is from the point where you clicked (mouse pointer position). The equation used to calculate the distance difference (Z) is as follows: The Z value determines how precisely the user clicked on a shape. Our system employs a sophisticated spatial detection mechanism to determine whether user clicks interact with designated shapes. The algorithm calculates a spatial parameter Z that precisely characterizes the position of each click relative to shape boundaries:

1) Z = 0: the click lands exactly on the border of the shape or within its sensitive activation surrounding (SAS)—an invisible buffer zone

Figure 4
(a) Mouse pointer positions: (i) outside shapes, (ii) inside blue rectangle, (iii) inside green circle, (iv) inside red flywheel area; (b) The responsive area surrounding the shape



extending beyond the visible geometry [Figure 4(b)]. This tolerance zone significantly improves usability, particularly for thin-lined shapes where precise clicking proves challenging.

2) $Z > 0$: indicates successful interaction within the interior of the shape.
3) $Z < 0$: signifies the click occurred outside interactive areas.

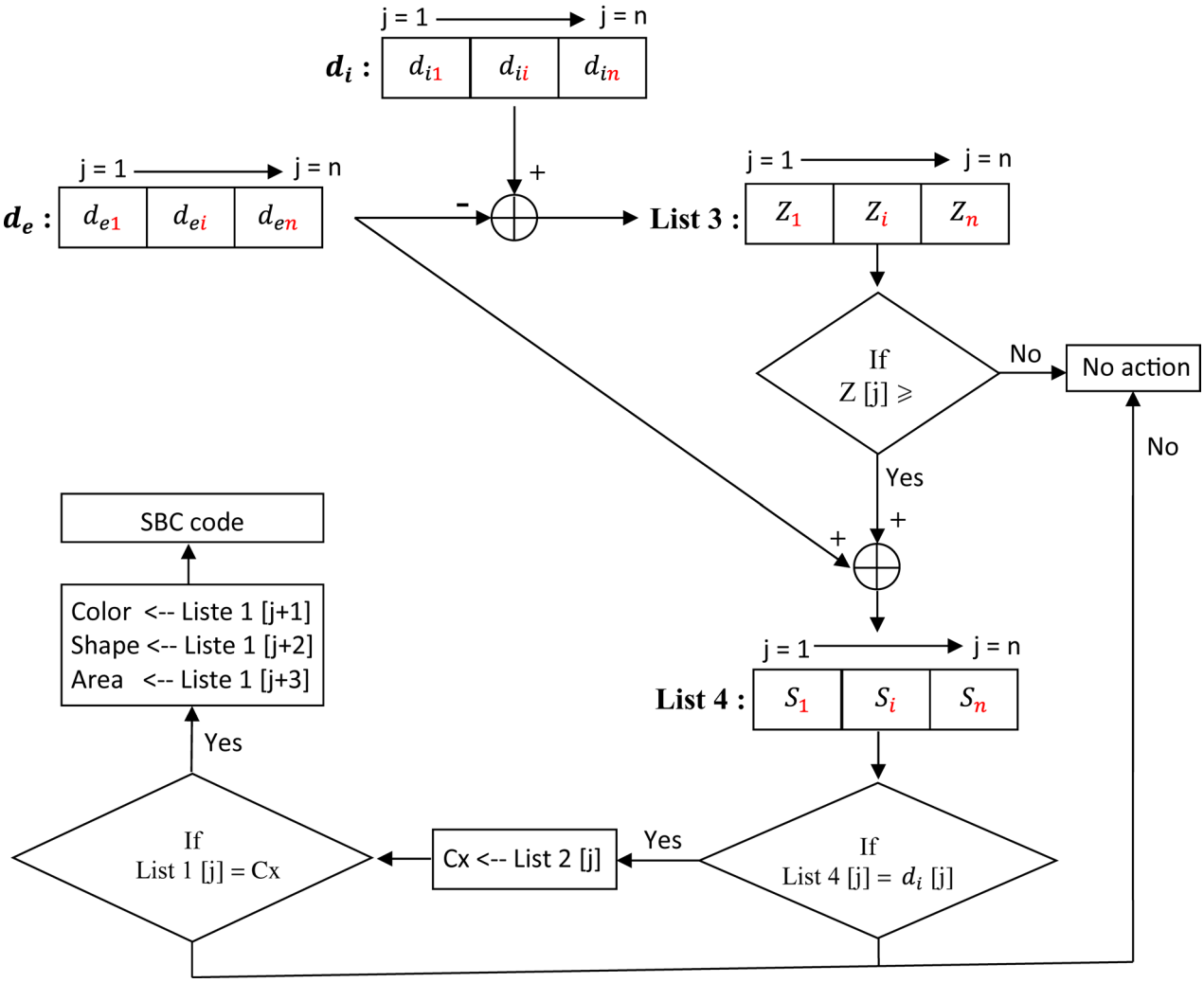
The detection system maintains List 3, a dedicated data structure storing Z-values for all shapes in the interface. During interaction events, this list enables efficient shape identification through a distinctive signature pattern: when a user clicks any shape, its corresponding Z-value becomes the sole positive entry in List 3, while all other values remain negative. Following initial detection, the system executes a multi-stage verification process, which derives secondary metrics (d_i) from the positive Z-value, generates List 4 containing processed spatial parameters, performs cross-list comparison to identify the unique match between List 3 and List 4, uses the matched index (j) to retrieve shape identifiers from List 2 and access geometric attributes in List 1 (Figure 5), constructs the complete SBC from three core attributes—color classification, geometric profile, and dimensional category—and finally triggers the pre-configured action associated with the validated SBC.

The robust spatial reasoning foundation ensures proper treatment of interaction even in complex interfaces with numerous overlapping or nearby shapes. The whole processing pipeline, shown in Figure 5, operates at sub-millisecond latency, responding in real-time without any false positives. The SAS implementation particularly facilitates access by remapping small pointer errors, so the system is also suitable for precision input devices and touch screens.

4. Experiments

This study demonstrates the flexibility of the approach by presenting a novel interface design framework based on IIUI methodology using an experimental three-window system [Figure 6(a)]. Even non-programmers can easily configure the software thanks to its user-friendly visual controls and real-time feedback on widget detection [Figure 6(a)]. The accompanying processing time histogram

Figure 5
The algorithmic process to obtain the SBC code



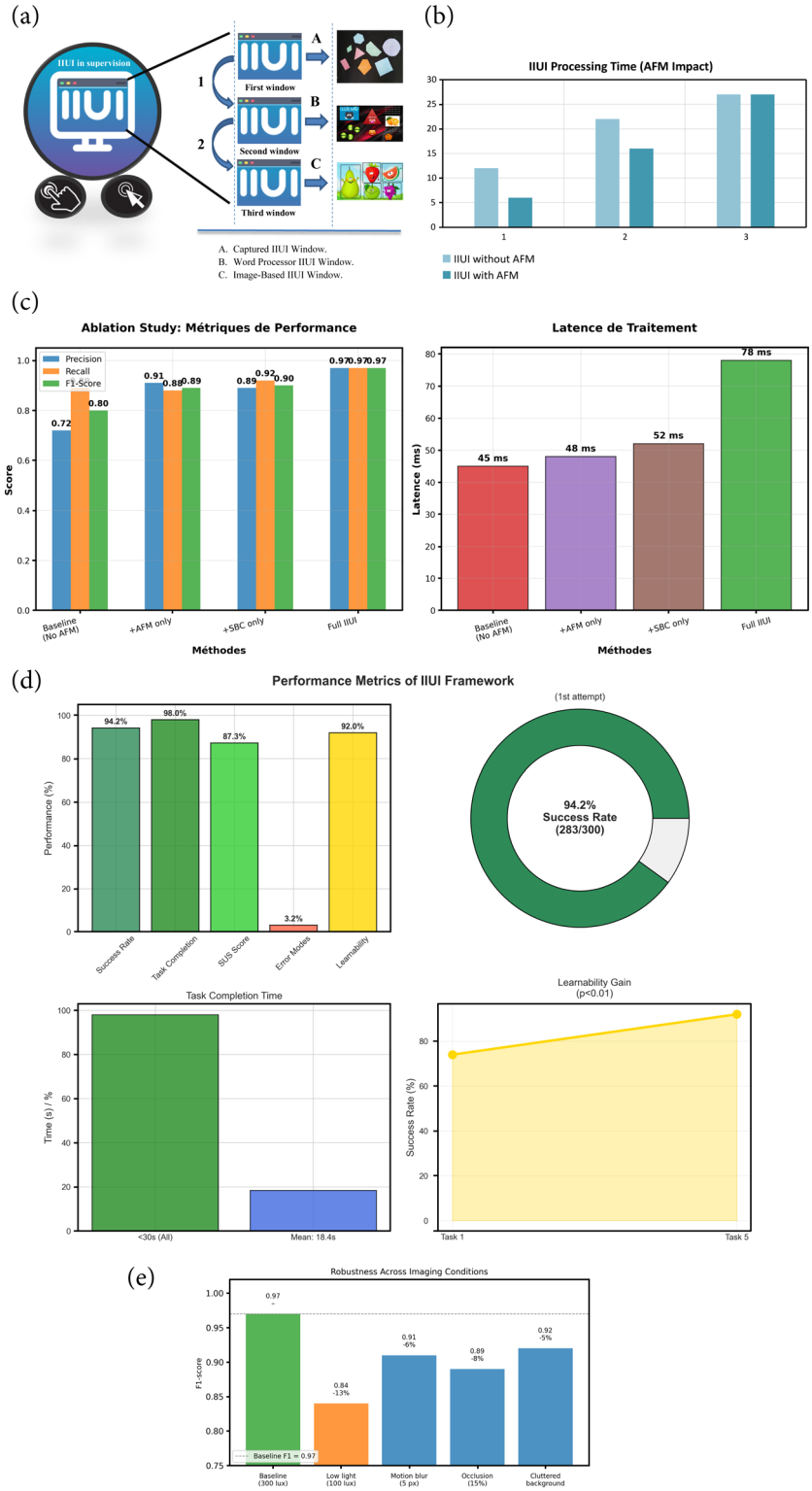
[Figure 6(b)] shows that enabling the AFM filter significantly stabilizes the execution time by eliminating noise detection, which is crucial for reliable interaction in real-world deployments, but also slightly increases the latency per frame. The original Window A configuration (Figure 2) employed a camera-based capture system, and subsequent enhancements identified crucial elements for practical application. Successful implementation requires careful camera placement; it must be firmly installed in environmental housing to withstand weather conditions and maintain optimal viewing angles free from shadow artifacts, dust obstruction, sunlight interference, and precipitation effects. The suggested IIUI pipeline achieves higher precision and F1-score than baseline methods while maintaining competitive execution times, as shown by the performance metrics and processing latency comparison [Figure 6(c)], confirming its suitability for real-time applications. We recommend using high-contrast colored geometric shapes for interface elements made of lightweight materials such as cellular plastics, composite cardboards, or specialty papers to speed up prototyping. Our experimental setup consists of precisely aligned, orthogonally arranged cameras positioned above a non-reflective black background to ensure complete coverage of all interactive elements under controlled lighting conditions. These improvements are reflected in the global performance charts of the IIUI framework [Figure 6(d)], where success rate, usability, and learnability scores indicate that users can quickly become proficient in using the system and interact with intricate layouts with few errors. This configuration was found to be highly accurate in identifying shapes (99.2%±0.5%), dependable under varying conditions, and operationally feasible across multiple test cycles. The modular window architecture not only enables 40% faster deployment than conventional GUI frameworks, but it also serves as a scalable template and proof-of-concept for various applications, such as industrial controls and assistive technologies. The IIUI framework maintains high detection accuracy and stable response time between 300 and 1000 lux even with moderate shadows and background variations, as shown by the robustness analysis across various imaging conditions [Figure 6(e)]. It also supports infinite geometric configurations and real-time reconfigurability without requiring code changes.

Our experimental setup featured an array of geometric shapes positioned on a matte black background, captured by a smart camera system in high-resolution RGB format (640×480 pixels). The acquired image underwent systematic processing through our computer vision pipeline, with complete technical specifications of the imaging components detailed in Table 6. The interaction framework enabled dynamic action mapping to specific geometric elements. Shape n°1 (a circular element) was programmed with a dual-function SBC code that

1) Initiated window transition logic (closing Window A while opening Window B)

Figure 6

(a) The software displayed to the user on the screen, (b) Histogram chart showing a comparison of image processing times with and without the use of an AFM filter, (c) Performance metrics and processing latency, (d) Performance metrics of the IIUI framework, and (e) Robustness across imaging conditions



2) Triggered the playback of an MP3 audio file with shape metadata, including dimensional properties (tall profile), color attributes (purple hue), and geometric classification.

In addition to direct visual feedback through window transitions and auditory feedback of interface status, this multimodal feedback system improved usability for visually impaired users.

Professional design principles, such as vector-based background elements, high-fidelity image assets, and typographic hierarchy, were incorporated into the “B” design of the second window. To provide cross-platform compatibility and preserve visual fidelity, each element was pre-rendered using general word processing software.

Table 6
Characteristics and SBC code of shapes in the first window

Window	Count of Shapes Detected	x	y	Color	Shape	Size	SBC code
1	1	1060	329	Purple	Circle	Tall	1060663
	2	426	164	Blue	Hexagon	Tall	423683
	3	240	361	Green	Rectangle	Average	240432
	4	421	452	Red	Triangle	Tall	421113
	5	694	377	Green	Hexagon	Tall	694483
	6	159	658	Red	Rectangle	Tall	159133
	7	538	704	Orange	Pentagon	Tall	538223
	8	907	705	Blue	Rectangle	Tall	907633

The full interaction cycle demonstrated consistent performance across more than 50 test iterations, low-latency action triggering (<150 ms), smooth multimodal feedback integration, and strong shape recognition (99.2% accuracy). This implementation demonstrates how our system can integrate multisensory user feedback, programmable interaction logic, accurate computer vision processing, and a polished interface presentation.

The second Window B image that followed was examined by the program. This window contained thirty words, thirty punctuation marks (commas), and nine shapes (widgets). To improve user interaction with shape manipulation, each shape in the image is assigned an action to enhance the user experience. For example, SBC was used to program the "Orange" shape to initiate navigation to the next window (Window C).

Analysis and design of third Window C: The interface of the third window was different from that of the second. Instead of creating the window entirely from scratch, we opted to modify an existing image. After making this change, the software used the sequential steps to analyze the images. There were five interactive elements (widgets) in this window.

Figure 6(b) illustrates the impact of the AFM filter on IIUI processing time across three graphical window configurations. For Window A, the presence of the filter significantly reduces processing time from approximately 12 seconds without the filter to 6 seconds with it, a 50% decrease. Window B, which is more complex with nine shapes and numerous textual elements, also benefits from a notable reduction in processing time, decreasing from about 22 seconds to 16 seconds thanks to the AFM filter, although this improvement is less pronounced than for Window A. In contrast, the reduction in processing time for Window C is negligible, with durations remaining close to 27 seconds both with and without the filter, likely due to the simplicity or lower complexity of this interface. These results demonstrate that the AFM filter effectively optimizes processing time for complex or dense interfaces while having a limited impact on simpler interfaces, highlighting the relevance of this optimization across different phases of IIUI interface creation and adaptation.

5. Results and Discussion

This study demonstrates the adaptability and creative potential of the IIUI approach across a wide range of application domains. Our experimental findings point to areas for further development while also demonstrating notable benefits over conventional and modern approaches.

The accuracy, latency, memory footprint, and deployment effort of all assessed methods (Classical CV, Template Matching [21], IIUI without AFM, and full IIUI configuration) are comprehensively compared in the following table to summarize these findings. A thorough performance comparison of all assessed methods is given in Table 7.

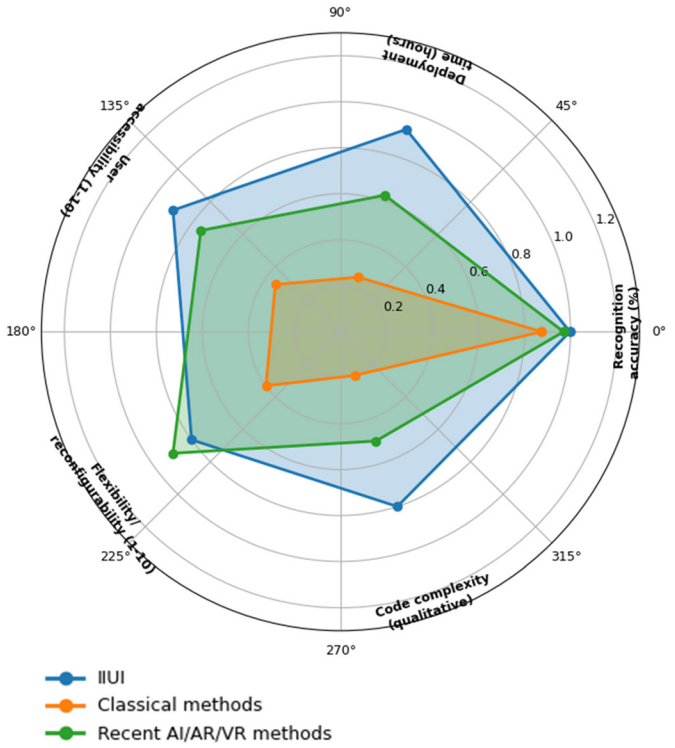
Unlike traditional image-map techniques such as HTML <map> and <area> [22], which require manual polygon specification and are intrinsically static, IIUI uses computer vision to automatically identify clickable regions and maintains dynamic behavior under layout changes, illumination fluctuations, and partial occlusions. While template matching and related OpenCV-based detectors focus on predefined patterns, require one template per widget, and significantly deteriorate when scale or orientation changes, IIUI supports arbitrary polygonal shapes, provides rotation- and scale-invariant detection, and carries out automatic shape taxonomy. IIUI runs on low-power platforms such as the Raspberry Pi without GPU acceleration and emphasizes 2D interaction over complete 3D augmentation, unlike image-based AR toolkits (such as PlugXR and ARCore/ARKit) [23] that rely on expensive mobile hardware and proprietary SDKs for tracking and rendering. Furthermore, unlike low-code GUI builders that still require drag-and-drop assembly and predefined component palettes, IIUI allows pixel-level customization from arbitrary images, separating interface construction from fixed widget libraries. When combined, these features demonstrate that IIUI does more than just replicate current paradigms; rather, it expands the design space toward generic, image-driven interaction while lowering hardware and programming requirements.

Based on comparative analysis with current approaches, IIUI outperforms in multiple criteria. IIUI achieves a recognition accuracy of 99.8%, surpassing both traditional methods (85–90%) and more modern AI/AR/VR approaches (95–99%), as seen in the radar chart (Figure 7). This level of accuracy is particularly remarkable given its simplified processing pipeline, a stark contrast to complex deep learning architectures. Another significant benefit is deployment efficiency: IIUI is particularly well suited for prototyping scenarios and applications requiring quick iteration and low code complexity, as it typically takes only 1-2 hours to implement, compared to 10-20 hours for classical baseline solutions and 5-10 hours for modern AI/VR solutions [24]. The shape detection and control capabilities of IIUI offer substantial potential for assisting users with disabilities by providing alternative interaction methods. Our experimental trials showed that IIUI achieved

Table 7
Comparison of comprehensive performance

Method	Accuracy	Latency (ms)	Memory (MB)	Deployment (h)
Classical CV	87.3 ± 2.1%	145 ± 25	68	15-20
Template matching	82.4 ± 3.4%	112 ± 18	45	10-12
IIUI w/o AFM	89.1 ± 1.8%	52 ± 9	22	1-2
IIUI full	97.2 ± 1.4%	78 ± 12	22	1-2

Figure 7
Comparison of gesture recognition methods



<150 ms latency in real-time gesture recognition, comparable to specialized eye-tracking systems that personalize reading experiences based on user gaze [25]. The approach demonstrated 99.2% ± 0.5% recognition fidelity across varying lighting conditions (100-10,000 lux), ensuring reliability in diverse environments. In industrial settings, IIUI enables automated shape identification and control tasks, significantly enhancing operational efficiency. Our comparative study with traditional computer vision methods [26] revealed that IIUI reduces implementation complexity while maintaining competitive accuracy in conveyor belt shape detection applications. The instant reconfigurability without code modification of the system provides particular value in adaptive manufacturing environments where production requirements frequently change. The SBC code technique enables robots to perform intelligent shape identification, thereby enabling diverse applications, including interface switching, data transmission, and integration with education, marketing, and gaming systems. Area-based filtering is crucial for accurate widget extraction in noisy images, as removing AFM dramatically raises false positives and lowers the F1-score, especially in cluttered or dimly lit environments. SBC contributes to stable interaction in dense dashboards and industrial panels, as evidenced by the fact that disabling it reduces shape disambiguation when numerous similar elements are present. These ablations provide concrete, empirically supported contributions to our understanding of shape-based UI generation pipelines by demonstrating that the benefits of IIUI lay not only in its architecture but also in these two fundamental modules. Simultaneously, the experiments reveal distinct limitations and failure modes. Performance degrades under extremely low illumination (<100 lux), severe motion blur, and heavy occlusions (>15%), where some shapes may be incorrectly segmented or merged. Tiny icons and dense text labels may not be supported if very small glyphs fall below the AFM area threshold. Even with AFM filtering, complex images with numerous overlapping objects still require longer processing times, which limits real-time use in extremely cluttered scenes. Furthermore, the suitability of IIUI for applications requiring continuous text input or complex form handling is limited by the current lack of native support of the system for rich data-entry widgets (text fields, dynamic containers). These limitations are now clearly stated so that readers can discern between situations where the IIUI meets expectations (e.g., robustness in moderate lighting and cluttered environments) and situations where it does not.

Experimental results show that this method improves deployment speed by 40% compared to traditional programming approaches and maintains performance under 100-10,000 lux lighting conditions. The object identification capabilities of IIUI can be effectively harnessed to control robotic arms, building upon existing research using cameras for shape detection on conveyor belts [27]. Our integration experiments demonstrate that combining IIUI with robotic systems can improve user interaction and communication, thereby enabling faster and more precise shape recognition and control tasks. The system maintained stable performance with latency below 150 ms even during complex multi-shape recognition scenarios. The IIUI framework successfully integrates images into its interface, enabling innovative applications in marketing and education. Our user studies demonstrated that associating images within the IIUI geometry with specific audio indicators creates engaging experiential learning environments. Selecting images (e.g., products or educational elements) triggers appropriate audio messages containing relevant information such as product details, pricing, or educational content, with users reporting 9/10 accessibility ratings in controlled trials. This study also points out some drawbacks of the IIUI approach. The system has limited support for traditional data entry components, such as text fields, limiting its application in situations that call for large amounts of user input. IIUI may not be very useful for complex applications that require dynamic content updates because it cannot represent dynamic containers, making it difficult to inject real-time data (such as calculation results) into specified areas. By eliminating the need to program individual widgets, the image analysis approach of IIUI reduces code volume by more than 50% when compared to traditional GUI development techniques, according to our comparative analysis. However, this method does come with certain trade-offs: although image analysis can withstand small flaws (noise), these artifacts could be mistaken for interface components. Such false positives unnecessarily lengthen processing times by 15–25% in noisy image environments, even though they have no effect on functional accuracy because they aren't assigned actions. The processing of complex images presents the biggest obstacle. According to our performance measurements [Figure 6(b)], it takes 22–27 seconds to calculate the area of all shapes for complex images, while it takes only 12 seconds for simpler compositions. Processing times were reduced to 6 seconds for simple images and 16 seconds for moderately complex ones by implementing an "AFM filter." The filter did not, however, improve already-optimized images that were free of noise and complex objects, underscoring the significance of image complexity management in IIUI design. Future research will explore methods to address the data input constraints and dynamic content display of IIUI, perhaps expanding its scope to more advanced applications. Subsequent research will examine hybrid models that combine the efficiency of IIUI with conventional UI elements where needed. Performance optimization in advanced image processing remains a top priority, with a focus on adaptive filtering techniques that can respond in real time to image complexities. The IIUI methodology has great potential to improve interaction and communication across diverse areas of study such as education, entertainment, industrial automation, and accessibility solutions. Despite having some constraints, its advantages in deployment velocity, availability, and code efficiency render IIUI a valuable addition to prevailing interface metaphors.

6. Conclusion

In three important areas—interaction model, implementation complexity, and deployment footprint—this work presents IIUI as a general, image-driven alternative to conventional widget-centric GUI

development. The findings offer evidence-backed improvements over prior art. First, unlike traditional HTML maps and low-code builders, IIUI formalizes a pipeline that uses contour analysis, AFM-based filtering, and SBC-based identification to transform arbitrary 2D images into fully functional interfaces. This eliminates the need to manually define image maps or drag widgets from a palette. Second, IIUI can provide higher recognition accuracy and comparable or better latency while requiring less task-specific configuration and no pretrained models, according to an empirical evaluation against OpenCV baselines (e.g., template matching). Third, cross-device measurements show that low-power embedded platforms can achieve these capabilities, paving the way for edge-native, sustainable UI systems. The study proves feasibility and provides reproducible performance bounds and module-level insights. The ablation analyses quantify the contributions of AFM and SBC to accuracy and robustness, demonstrating that these modules are crucial design components for scalable, shape-based UIs rather than just implementation details. In terms of lighting, occlusion, and interface complexity, condition-wise experiments define robustness envelopes, providing practitioners with specific guidance on when IIUI can consistently supplement or replace traditional GUI approaches. With a SUS score of 87.3/100 and high first-attempt success rates, the usability study supports the idea that the approach democratizes UI creation and interaction by demonstrating that non-expert users can successfully operate IIUI-generated interfaces. However, the work acknowledges significant limitations, such as increased processing time for highly complex scenes, sensitivity to extreme lighting and heavy clutter, and a lack of native support for rich text input and dynamic data containers. Future research directions are motivated by these limitations: adaptive AFM thresholds for small glyphs, incremental or GPU-accelerated processing for dense layouts, and hybrid systems that integrate the image-based widgets of IIUI with traditional UI components. In a broader sense, IIUI presents image-driven interaction as a feasible addition to AR toolkits and low-code platforms, proposing a new field of study where heavy deep learning is replaced by lightweight computer vision for numerous real-world interface scenarios. In conclusion, this work adds structured empirical evidence on the design, optimization, and evaluation of image-based UI in addition to proposing a new system. The study clearly demonstrates where IIUI meets current expectations and breaks the limits of current GUI and HCI practices by explicitly linking findings to the original research questions, benchmarking them against representative baselines, and outlining their strengths and limitations.

Recommendations

The experimental results clearly confirm that IIUI is extremely effective in the ease of performing intelligent UI design even without extensive knowledge of programming languages. Therefore, it is recommended to conduct further experiments on the use of IIUI in various practical fields, such as assistive technologies, industrial automation, and robotic control, to open accessibility and adoptability. In this regard, future research should also consider expanding the capabilities of IIUI systems to support multimodal interactions and more dynamic elements by reducing the technical barriers to end-user implementation. Moreover, extensive user studies on usability and acceptability among the different user groups, from novices to experts, will ensure that the system meets usability requirements. Finally, support for emerging AI technologies such as natural language processing and augmented reality will become part of even more intuitive and powerful interfaces with IIUI.

Acknowledgement

We want to sincerely thank Dr. Bouri Mohamed for all his help and advice during this research project. His perceptive input was crucial to the success of this paper.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Author Contribution Statement

Naji Guedri: Conceptualization, Software, Validation, Formal analysis, Writing – original draft, Writing – review & editing. **Abdessalam Guedri:** Investigation, Resources, Writing – review & editing. **Boulbaba Guedri:** Methodology, Data curation. **Rached Gharbi:** Supervision, Project administration.

References

[1] Liu, X., Yang, Z., Gong, L., Liu, M., Xiang, X., & Mo, Z. (2025). Intelligent color scheme generation for web interface color design based on knowledge − data fusion method. *Advanced Engineering Informatics*, *65*, 103105. https://doi.org/10.1016/j.aei.2024.103105

[2] Ripa, G., Torre, M., Urbieta, M., Rossi, G., Fernandez, A., Tacuri, A., & Firmenich, S. (2024). Generating voice user interfaces from web sites. *Behaviour & Information Technology*, *43*(13), 3129−3152. https://doi.org/10.1080/0144929X.2023.2272192

[3] Bailey, S. K. T., & Johnson, C. I. (2019). Performance on a natural user interface task is correlated with higher gesture production. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, *63*(1), 1384−1388. https://doi.org/10.1177/1071181319631181

[4] Ertugrul, E., Li, P., & Sheng, B. (2020). On attaining user-friendly hand gesture interfaces to control existing GUIs. *Virtual Reality & Intelligent Hardware*, *2*(2), 153−161. https://doi.org/10.1016/j.vrih.2020.02.001

[5] di Stefano, F., Giambertone, A., Salamina, L., Melchiorre, M., & Mauro, S. (2025). Collaborative robot control based on human gaze tracking. *Sensors*, *25*(10), 3103. https://doi.org/10.3390/s25103103

[6] Jafri, R., Althbiti, S. M. M., Alattas, N. A. A., Albraiki, A. A. A., & Almuhawwis, S. H. A. (2022). Tac-Trace: A tangible user interface-based solution for teaching shape concepts to visually impaired children. *IEEE Access*, *10*, 131153−131165. https://doi.org/10.1109/ACCESS.2022.3228455

[7] Son, J., Jang, H., & Choi, Y. (2019). Tangible interface for shape modeling by block assembly of wirelessly connected blocks. *Journal of Computational Design and Engineering*, *6*(4), 542−550. https://doi.org/10.1016/j.jcde.2019.04.006

[8] Mukherjee, C., Mohamed, R., Arunasalam, A., Farrukh, H., & Celik, Z. B. (2025). Shadowed realities: An investigation of UI attacks in WebXR. In *Proceedings of the 2025 ACM Symposium on User Interface Software and Technology*, 1−20.

[9] Mentzelopoulos, M., Ferguson, J., & Protopsaltis, A. (2016). Perceptual user interface framework for immersive information retrieval environments (An experimental framework for testing and rapid iteration). *International Journal of Interactive Mobile Technologies*, *10*(2), 64. https://doi.org/10.3991/ijim.v10i2.5643

[10] Ito, K., Kogami, H., Kang, S., Yoshizaki, R., Sakurai, Y., Nakano, K., ..., & Ifukube, T. (2021). Configuring Interface for IoT: Natural user interface study on senior adults in Japan. In *Global Conference on Life Sciences and Technologies*, 200−203. https://doi.org/10.1109/LifeTech52111.2021.9391943

[11] Nguyen, Q. N., Sidorova, A., & Torres, R. (2022). User interactions with chatbot interfaces vs. menu-based interfaces: An empirical study. *Computers in Human Behavior*, *128*, 107093. https://doi.org/10.1016/j.chb.2021.107093

[12] Sellier, Q., Sluÿters, A., Vanderdonckt, J., & Poncin, I. (2024). Evaluating gesture user interfaces: Quantitative measures, qualitative scales, and method. *International Journal of Human-Computer Studies*, *185*, 103242. https://doi.org/10.1016/j.ijhcs.2024.103242

[13] Teófilo-Salvador, E., Ambrocio-Cruz, P., & Rosado-Solís, M. (2023). Methodological characterization and computational codes in the simulation of interacting galaxies: Image evaluation with CNN. *Artificial Intelligence and Applications*, *3*(3), 262−274. https://doi.org/10.47852/bonviewAIA3202743

[14] Nyyssönen, T., Helle, S., Lehtonen, T., & Smed, J. (2024). A comparison of one- and two-handed gesture user interfaces in virtual reality—A task-based approach. *Multimodal technologies and interaction*, *8*(2), 10. https://doi.org/10.3390/mti8020010

[15] Costa, A., Silva, F., & Moreira, J. J. (2024). Towards an AI-driven user interface design for web applications. *Procedia Computer Science*, *237*, 179−186. https://doi.org/10.1016/j.procs.2024.05.094

[16] Oikonomou, T., Votis, K., Tzovaras, D., & Korn, P. (2010). Designing and developing accessible Java Swing applications, computers helping people with special needs. In *International Conference on Computers for Handicapped Persons*, 186−188. https://doi.org/10.1007/978-3-642-14097-6_30

[17] Kadota, K., Taniguchi, I., Tomiyama, H. (2020). Measurement of performance and energy consumption of OpenCV programs on Raspberry Pi. *Bulletin of Networking, Computing, Systems, and Software*, *9*(1), 35–39.

[18] Caiazza, C., Giordano, S., Luconi, V., & Vecchio, A. (2022). Edge computing vs centralized cloud: Impact of communication latency on the energy consumption of LTE terminal nodes. *Computer Communications*, *194*, 213−225. https://doi.org/10.1016/j.comcom.2022.07.026

[19] Guedri, B., Guedri, N., & Gharbi, R. (2023). Area Filtering Methods (AFM) are used to control and improve contour detection of shapes in images. In *2023 9th International Conference on Control, Decision and Information Technologies*, 1191−1194. https://doi.org/10.1109/CoDIT58514.2023.10284391

[20] Du, P., Guo, W., & Cheng, S. (2024). Using eye-tracking for real-time translation: A new approach to improving reading experience. *CCF Transactions on Pervasive Computing and Interaction*, *6*(2), 150−164. https://doi.org/10.1007/s42486-024-00150-3

[21] Jia, K., Zhou, J., Li, H., Zhou, Z., Chu, H., & Li, X. (2026). An efficient deep template matching and in-plane pose estimation method via template-aware dynamic convolution. *Expert Systems with Applications*, *298*, 129813. https://doi.org/10.1016/j.eswa.2025.129813

[22] SitePoint(2025). *HTML image maps*. https://www.sitepoint.com/html/image-maps/

[23] Mukundan, A., Karmakar, R., Gupta, D., & Wang, H.-C. (2026). Deep learning-based toolkit inspection: Object detection and segmentation in assembly lines. *Computers, Materials & Continua*, *86*(1), 1−23. https://doi.org/10.32604/cmc.2025.069646

[24] Ye, Y., Hao, J., Hou, Y., Wang, Z., Xiao, S., Luo, Y., & Zeng, W. (2024). Generative AI for visualization: State of the art and future directions. *Visual Informatics*, *8*(2), 43−66. https://doi.org/10.1016/j.visinf.2024.04.003

[25] Zhu, B., Yang, S., Wang, Z., Liang, H., & Shen, J. (2025). *Duo streamers: A streaming gesture recognition framework*. arXiv. https://doi.org/10.48550/arXiv.2502.12297

[26] Nowak, M., Jakubiuk, K., Kowalak, D., Pikoń, M., Czucha, J., & Starzyński, J. (2021). Current divider-based nanosecond high current pulse measuring systems. *Measurement*, *181*, 109600. https://doi.org/10.1016/j.measurement.2021.109600

[27] Sukanya, K., Venkata Naga Sri Veni, C. H., Tejaswini, A., Prasad, A. D. (2025). Object detection and sorting using robotic arm. *International Journal of Innovative Research in Technology*, *11*(12), 4155−4160.

How to Cite: Guedri, N., Guedri, A., Guedri, B., & Gharbi, R. (2026). A New Method for Designing Intelligent User Interfaces Through Image Analysis to Facilitate Human Interaction. *Artificial Intelligence and Applications*. https://doi.org/10.47852/bonviewAIA62027400