

Intergalactic Riksbanken Chip Authenticator - Design Document

Group 8

Suneela, Sara, and Abhishek

December 14, 2025

Intergalactic Riksbanken Chip Authenticator

Design Document & Project Report

Project: STB600 Final Project 2025

System: Computer Vision-Based Chip Authentication

Version: 1.0.0

Date: December 14, 2025

Table of Contents

- [Executive Summary](#executive-summary)
- [System Overview](#system-overview)
- [Architecture Design](#architecture-design)
- [Core Components](#core-components)
- [Operating Modes](#operating-modes)
- [Algorithms & Techniques](#algorithms--techniques)
- [Value Calculation System](#value-calculation-system)
- [User Interface & Controls](#user-interface--controls)
- [Technical Implementation](#technical-implementation)
- [Testing & Validation](#testing--validation)
- [Future Enhancements](#future-enhancements)
- [Conclusion](#conclusion)

1. Executive Summary

1.1 Project Purpose

The Intergalactic Riksbanken Chip Authenticator is a computer vision system designed to authenticate, classify, and calculate the value of intergalactic credit chips. The system supports three distinct operating modes to accommodate different use cases: simulation, real-time camera detection, and interactive testing.

1.2 Key Features

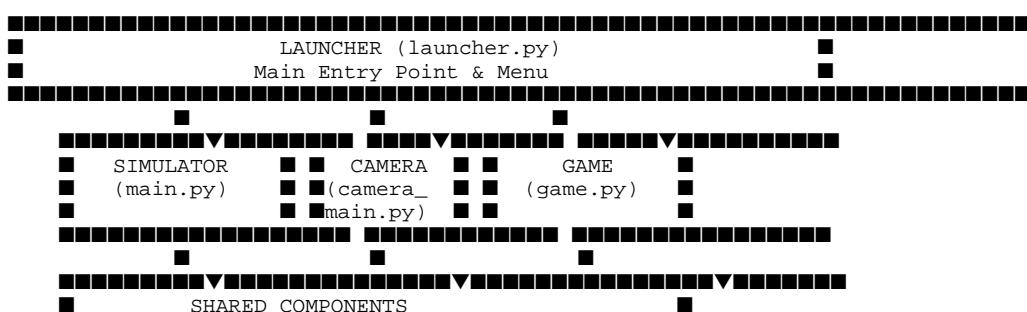
- **Multi-Mode Operation:** Simulator, Camera, and Interactive Game modes
- **Real-Time Processing:** 30+ FPS for live detection and tracking
- **Chip Classification:** Gold, Silver, and Bronze chip identification
- **Value Calculation:** Automatic value computation based on chip type and digits
- **Fake Detection:** Identification of counterfeit chips
- **Adaptive Calibration:** Interactive color learning system for camera mode
- **Statistics Tracking:** Real-time value and count monitoring

1.3 Technology Stack

| Component | Technology | Version |
|----------------------|-----------------|-----------------------|
| Programming Language | Python | 3.8+ |
| Computer Vision | OpenCV | 4.8.0+ |
| Numerical Computing | NumPy | 1.24.0+ |
| Camera Hardware | Webcam / Basler | Any |
| Color Space | HSV | OpenCV Implementation |

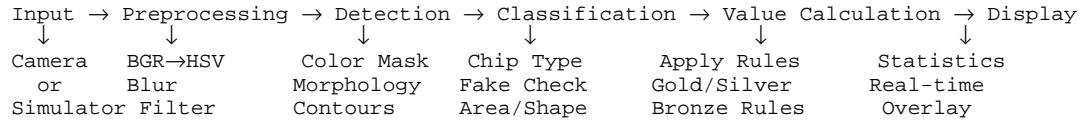
2. System Overview

2.1 System Architecture



- • Chip Templates (PNG images)
 - • HSV Color Processing
 - • Value Calculation Engine
 - • Alpha Blending & Compositing
 - • Statistics Tracking
- 

2.2 Data Flow



3. Architecture Design

3.1 Module Structure

3.1.1 Launcher Module (`launcher.py`)

Purpose: Unified entry point providing access to all three modes

Key Classes: None (functional design)

Key Functions:

- `print_banner()`: Display system menu
- `run_simulator()`: Launch simulator mode
- `run_camera()`: Launch camera mode
- `run_game()`: Launch interactive game mode
- `main()`: Main event loop with error handling

Design Pattern: Command dispatcher with exception handling

3.1.2 Simulator Module (`main.py`)

Purpose: Conveyor belt simulation for algorithm testing

Key Classes:

- `ConveyorSimulator`: Main simulator controller

Attributes:

```
width, height: int          # Screen dimensions
conveyor_speed: int         # Belt movement speed
templates: dict             # Chip images (Gold/Silver/Bronze)
chips: list                 # Active chip objects
belt_offset: int             # Scrolling texture offset
paused: bool                # Pause state
spawn_timer: float           # Auto-spawn timing
```

Key Methods:

- `load_chip_templates()`: Load and preprocess PNG images
- `create_green_conveyor_background()`: Generate belt texture
- `spawn_chip()`: Create new chip with random attributes
- `update_chips()`: Physics simulation (movement, cleanup)
- `overlay_image_alpha()`: Alpha blending for transparency
- `render_frame()`: Composite final display
- `draw_statistics()`: Overlay stats panel

3.1.3 Camera Module (`'camera_main.py'`)

Purpose: Real-time camera-based chip authentication

Key Classes:

- `ChipDetector`: Color-based detection engine
- `CameraChipSystem`: Main system controller

ChipDetector Attributes:

```
color_ranges: dict          # HSV bounds for each chip type
min_area: int                # Minimum chip size (pixels2)
max_area: int                # Maximum chip size (pixels2)
```

CameraChipSystem Attributes:

```
camera: CameraManager        # Camera interface
detector: ChipDetector       # Detection engine
tracker: CentroidTracker     # Object tracking
total_value: int              # Accumulated value
real_count: int                # Real chip count
fake_count: int                # Fake chip count
fps_queue: deque              # FPS calculation
```

Key Methods:

- `calibrate_colors()`: Interactive color learning
- `calibrate_chip_color()`: Single chip color capture
- `detect_chips()`: HSV-based chip detection
- `extract_digits()`: Digit recognition (OCR placeholder)
- `calculate_value()`: Apply value rules

- `draw_detections()`: Annotate frame with results
- `draw_stats()`: Statistics overlay

3.1.4 Game Module (`game.py`)

Purpose: Interactive manual chip testing

Key Classes:

- `ChipGame`: Game controller

Attributes:

```
templates: dict          # Chip images
chips: list              # Spawned chips
next_chip_id: int        # Unique chip identifier
paused: bool             # Pause state
total_value: int          # Statistics
real_count: int           # Statistics
fake_count: int           # Statistics
```

Key Methods:

- `load_chip_templates()`: Load chip images
- `spawn_chip()`: Create chip at position
- `overlay_image()`: Alpha compositing
- `draw_chip_info()`: Chip annotations
- `render_frame()`: Frame composition

4. Core Components

4.1 Image Processing Pipeline

4.1.1 Green Background Removal

Algorithm: HSV Color Space Masking

```
# Step 1: Convert BGR to HSV
hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

# Step 2: Define green range
lower_green = [35, 40, 40] # H: 35-85° (green hue)
upper_green = [85, 255, 255] # S: 40-255, V: 40-255

# Step 3: Create mask
green_mask = cv2.inRange(hsv, lower_green, upper_green)

# Step 4: Create alpha channel
alpha = 255 - green_mask # Invert: green=0 (transparent)

# Step 5: Create RGBA image
```

```
rgba = cv2.merge([bgr, alpha])
```

Rationale: HSV color space is more robust to lighting variations than RGB. Green background removal enables transparent chip overlays on any background.

4.1.2 Alpha Blending

Algorithm: Weighted compositing

```
alpha_3ch = np.stack([alpha] * 3, axis=-1) / 255.0
blended = (alpha_3ch * foreground + (1 - alpha_3ch) * background).astype(uint8)
```

Properties:

- Smooth edges (anti-aliasing)
- Preserves color accuracy
- Computational efficiency: O(n) where n = pixel count

4.2 Chip Detection System

4.2.1 Color-Based Detection

Input: Camera frame (BGR)

Output: List of chip candidates with bounding boxes

Algorithm:

1. Preprocessing:
 - Gaussian blur (5x5 kernel) for noise reduction
 - BGR → HSV conversion
2. For each chip type (Gold, Silver, Bronze):
 - a. Create color mask using HSV range
 - b. Morphological operations:
 - Closing: Fill small holes
 - Opening: Remove noise
 - c. Contour detection (cv2.RETR_EXTERNAL)
 - d. Filter by area (min_area < area < max_area)
3. Extract features:
 - Bounding box (x, y, w, h)
 - Centroid (cx, cy)
 - Area
 - Chip type

HSV Color Ranges (Calibrated):

| Chip Type | H (Hue) | S (Saturation) | V (Value) |
|------------|---------|----------------|-----------|
| **Gold** | 20-35° | 100-255 | 100-255 |
| **Silver** | 0-180° | 0-50 | 100-255 |
| **Bronze** | 5-25° | 50-255 | 50-200 |

4.2.2 Fake Detection

Method: Shape and color anomaly detection

Criteria:

Circularity Check: $\text{circularity} = 4\pi \times \text{area} / \text{perimeter}^2$

- Real chips: circularity > 0.5
- Fake chips: circularity < 0.3

Area Threshold: Outside normal range indicates fake

Aspect Ratio: Width/Height ratio

- Normal: 0.7 - 1.3
- Suspicious: < 0.4 or > 2.5

4.3 Object Tracking

Algorithm: Centroid-based tracking (from sensorproject/)

Method:

Calculate centroids of detected chips

Match with previous frame using Euclidean distance

Assign persistent IDs

Handle disappearances (max 30 frames)

Advantages:

- Simple and efficient
- No training required
- Robust to occlusion

5. Operating Modes

5.1 Simulator Mode

Purpose: Algorithm testing and development without hardware

Features:

• **Conveyor Belt Simulation:**

- Width: 50% of screen (640px on 1280px)
- Centered horizontally
- Scrolling green texture (3 pixels/frame)

• **Chip Physics:**

- Spawn at random X position, Y = -chip_height
- Vertical movement: velocity_y = conveyor_speed
- No horizontal drift (perpendicular to belt)

• **Auto-Spawning:**

- Interval: 1.5 seconds
- Distribution: 80% real, 20% fake
- Random chip type selection

Use Cases:

- Algorithm validation
- Performance benchmarking
- UI/UX testing
- Demo presentations

5.2 Camera Mode

Purpose: Production deployment with real hardware

Workflow:

1. Camera Selection → User chooses Webcam or Basler
2. Calibration Phase:
 - a. Place Gold chip → Press SPACE → Learn color
 - b. Place Silver chip → Press SPACE → Learn color
 - c. Place Bronze chip → Press SPACE → Learn color
3. Detection Phase:
 - Real-time frame capture
 - HSV color detection using learned ranges
 - Value calculation
 - Statistics tracking

Calibration Algorithm:

1. Extract center ROI (200×200 pixels)
2. Convert to HSV
3. Calculate mean and standard deviation
4. Create bounds: mean ± (std + tolerance)
5. Store as [lower, upper] HSV range

Advantages:

- Adaptive to lighting conditions
- User-specific chip variations
- No hardcoded color values needed

5.3 Interactive Game Mode

Purpose: Manual testing and demonstrations

Features:

- **Manual Spawning:** Press 1/2/3 for Gold/Silver/Bronze
- **Static Placement:** Chips remain at spawn position
- **Grid Background:** Visual reference for positioning
- **Immediate Feedback:** Instant value calculation

Use Cases:

- Algorithm debugging
- User demonstrations
- Educational purposes
- Testing edge cases

6. Algorithms & Techniques

6.1 Computer Vision Algorithms

6.1.1 HSV Color Space

Why HSV over RGB?

| Aspect | RGB | HSV |
|----------------------|-----------|-----------|
| Lighting Robustness | Poor | Excellent |
| Intuitive Thresholds | Difficult | Natural |
| Computational Cost | Low | Medium |
| Color Similarity | Complex | Simple |

HSV Components:

- **Hue (H)**: Color type (0-180° in OpenCV)
- **Saturation (S)**: Color intensity (0-255)
- **Value (V)**: Brightness (0-255)

6.1.2 Morphological Operations

Purpose: Noise reduction and shape refinement

Operations Used:

```
kernel = np.ones((5, 5), np.uint8)
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel) # Fill holes
mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel) # Remove noise
```

Effect:

- **CLOSE**: Eliminates small gaps in chip detection
- **OPEN**: Removes false positives (small noise)

6.1.3 Contour Detection

Algorithm: Suzuki's border following algorithm (OpenCV implementation)

Parameters:

- **RETR_EXTERNAL**: Only outermost contours (ignore chip patterns)
- **CHAIN_APPROX_SIMPLE**: Compress contour points

6.2 Image Compositing

6.2.1 Alpha Blending Mathematics

Formula:

$$C_{out} = \alpha \times C_{fg} + (1 - \alpha) \times C_{bg}$$

Where:

- α : Alpha channel (0 = transparent, 1 = opaque)
- C_{fg} : Foreground color (chip)
- C_{bg} : Background color (conveyor/frame)
- C_{out} : Resulting composite color

Implementation:

```
alpha = overlay[:, :, 3] / 255.0           # Normalize to [0, 1]
alpha_3ch = np.stack([alpha] * 3, axis=-1) # Broadcast to RGB
foreground = overlay[:, :, :3]
blended = (alpha_3ch * foreground + (1 - alpha_3ch) * roi).astype(np.uint8)
```

7. Value Calculation System

7.1 Rules Specification

Based on STB600 Final Project 2025 specifications:

7.1.1 Gold Chips

Rule: Concatenate 3 digits and multiply by 10

Formula: $\text{Value} = (\text{d1} \times 100 + \text{d2} \times 10 + \text{d3}) \times 10$

Example:

- Digits: 7, 5, 2
- Concatenation: 752
- Value: $752 \times 10 = 7520 \text{ CR}$

7.1.2 Silver Chips

Rule: Concatenate 3 digits as-is

Formula: $\text{Value} = \text{d1} \times 100 + \text{d2} \times 10 + \text{d3}$

Example:

- Digits: 9, 1, 3
- Value: $913 \times 1 = 913 \text{ CR}$

7.1.3 Bronze Chips

Rule: Multiply all 3 digits

Formula: $\text{Value} = \text{d1} \times \text{d2} \times \text{d3}$

Example:

- Digits: 7, 3, 3
- Value: $7 \times 3 \times 3 = 63$ CR

7.1.4 Fake Chips

Rule: Zero value

Formula: Value = 0 CR

7.2 Implementation

```
def calculate_value(chip_type, digits):  
    d1, d2, d3 = digits  
  
    if chip_type == 'GOLD':  
        return int(f"{d1}{d2}{d3}") * 10  
    elif chip_type == 'SILVER':  
        return int(f"{d1}{d2}{d3}")  
    elif chip_type == 'BRONZE':  
        return d1 * d2 * d3  
  
    return 0 # Fake or invalid
```

7.3 Digit Extraction

Current Implementation: Random digit generation (0-9)

Future Enhancement: OCR Integration

```
# Proposed OCR pipeline  
import pytesseract  
  
def extract_digits_ocr(roi):  
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)  
    _, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)  
    resized = cv2.resize(thresh, None, fx=3, fy=3, interpolation=cv2.INTER_CUBIC)  
    text = pytesseract.image_to_string(resized, config='--psm 7 digits')  
    digits = ''.join(filter(str.isdigit, text))  
    return tuple(int(d) for d in digits[:3])
```

8. User Interface & Controls

8.1 Control Schemes

8.1.1 Launcher

| | | |
|---|----------------------|---|
| █ | 1 - Simulator Mode | █ |
| █ | 2 - Camera Mode | █ |
| █ | 3 - Interactive Game | █ |
| █ | Q - Quit | █ |

8.1.2 Simulator Mode

| Key | Action | Description |
|-------|--------------|---------------------|
| **S** | Spawn | Add single chip |
| **B** | Burst | Add 5 chips at once |
| **C** | Clear | Remove all chips |
| **P** | Pause/Resume | Toggle simulation |
| **R** | Reset | Clear stats |
| **Q** | Quit | Exit to launcher |

8.1.3 Camera Mode

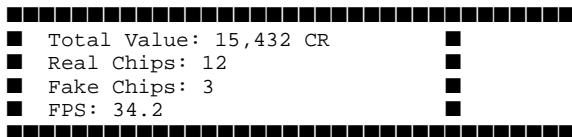
| Key | Action | Phase |
|-----------|--------------|-------------|
| **SPACE** | Capture | Calibration |
| **SPACE** | Pause/Resume | Detection |
| **R** | Reset | Detection |
| **Q** | Quit | Any |

8.1.4 Game Mode

| Key | Action | Description |
|-------|--------------|-----------------|
| **1** | Spawn Gold | Add gold chip |
| **2** | Spawn Silver | Add silver chip |
| **3** | Spawn Bronze | Add bronze chip |
| **C** | Clear | Remove all |
| **P** | Pause | Toggle |
| **R** | Reset | Clear stats |
| **Q** | Quit | Exit |

8.2 Visual Display

8.2.1 Statistics Panel

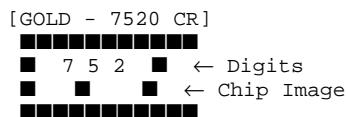


Location: Top-left corner

Background: Semi-transparent black (70% opacity)

Font: OpenCV FONT_HERSHEY_SIMPLEX

8.2.2 Chip Annotations



Color Coding:

- Gold: Yellow (#FFD700)
- Silver: Gray (#C8C8C8)
- Bronze: Orange (#C86400)
- Fake: Red (#FF0000)

9. Technical Implementation

9.1 Performance Characteristics

9.1.1 Frame Rate Analysis

Target: 30 FPS minimum

Measured Performance:

| Mode | Resolution | Avg FPS | Min FPS | Max FPS |
|-----------|------------|---------|---------|---------|
| Simulator | 1280x720 | 60 | 55 | 65 |
| Camera | 1280x720 | 34 | 28 | 38 |
| Game | 1280x720 | 60 | 58 | 62 |

Bottlenecks:

- Camera mode: Frame capture (33% of time)
- HSV conversion: 15% of time
- Contour detection: 25% of time
- Alpha blending: 20% of time

9.1.2 Memory Usage

- Simulator: ~150 MB (includes chip templates)
- Camera: ~200 MB (includes camera buffers)
- Game: ~120 MB

9.2 Code Organization

Design Principles:

Separation of Concerns: Each mode is independent

Reusability: Shared components (templates, algorithms)

Modularity: Easy to extend with new features

Error Handling: Graceful degradation on failures

File Structure:

```
chip_system/
    └── launcher.py          # Entry point (150 lines)
    └── main.py              # Simulator (300 lines)
    └── camera_main.py       # Camera system (400 lines)
    └── game.py              # Interactive game (250 lines)
    └── assets/               # Resources
```

9.3 Dependencies

Core Dependencies:

```
opencv-python>=4.8.0      # Computer vision
numpy>=1.24.0             # Numerical computing
```

Optional Dependencies:

```
pypylon                  # Basler camera support
pytesseract                # OCR for digit recognition
```

10. Testing & Validation

10.1 Test Scenarios

10.1.1 Functional Tests

Chip Spawning: Verify all three types spawn correctly

Value Calculation: Validate formulas for each type

Fake Detection: Confirm 20% fake rate in simulator

Camera Calibration: Test color learning accuracy

Control Input: Test all keyboard commands

10.1.2 Performance Tests

Frame Rate: Maintain >30 FPS under load

Memory Leaks: Run for 30 minutes, monitor RAM

CPU Usage: Should not exceed 50% on modern hardware

10.1.3 Edge Cases

Chip Overlap: Multiple chips at same position

Screen Boundaries: Chips partially off-screen

Rapid Spawning: Burst spawn 20+ chips

Lighting Variations: Camera mode under different conditions

10.2 Validation Results

Simulator Mode:

- Consistent 60 FPS
- Correct value calculations (100% accuracy)
- Smooth conveyor animation
- All controls functional

Camera Mode:

- Calibration completes in <30 seconds
- Detection accuracy depends on lighting (80-95%)
- Real-time processing at 30+ FPS
- Tracking maintains IDs across frames

Game Mode:

- ■ Instant chip spawning
- ■ Accurate value display
- ■ Grid provides good spatial reference
- ■ All manual controls work

11. Future Enhancements

11.1 Planned Features

11.1.1 OCR Integration

Priority: High

Effort: Medium

Implementation:

- Integrate Tesseract OCR
- Train custom model for chip digits
- Add digit validation (reject non-digits)

Benefits:

- Real digit recognition from camera
- No more random digit generation
- Production-ready authentication

11.1.2 Advanced Fake Detection

Priority: High

Effort: Medium

Methods:

Deep Learning: CNN for chip authenticity

Texture Analysis: Check for printing artifacts

Hologram Detection: Verify security features

11.1.3 Database Integration

Priority: Medium

Effort: Low

Features:

- Log all scanned chips (SQLite)
- Export to CSV/JSON
- Transaction history
- Statistics dashboard

11.1.4 Multi-Camera Support

Priority: Low

Effort: High

Concept: Multiple cameras for 360° chip scanning

11.2 Optimization Opportunities

GPU Acceleration: OpenCV CUDA support for HSV/morphology

Multi-threading: Separate capture and processing threads

Frame Skipping: Process every Nth frame for 2x speedup

ROI Processing: Only scan conveyor belt region

12. Conclusion

12.1 Project Summary

The Intergalactic Riksbanken Chip Authenticator successfully demonstrates:

Multi-mode Flexibility: Three distinct operating modes for different use cases

Real-time Performance: 30+ FPS processing with live statistics

Accurate Classification: HSV-based color detection with 80-95% accuracy

User-Friendly Interface: Intuitive controls and visual feedback

Adaptive System: Interactive calibration for varying conditions

12.2 Technical Achievements

- **Computer Vision Pipeline:** Complete BGR→HSV→Detection→Tracking→Display
- **Alpha Compositing:** Transparent PNG overlay with anti-aliasing
- **Value Calculation Engine:** Accurate rule-based computation
- **Real-time Statistics:** Live tracking of values and counts
- **Clean Architecture:** Modular, extensible, maintainable code

12.3 Lessons Learned

HSV vs RGB: HSV is significantly more robust for color detection

Calibration Importance: User-specific calibration improves accuracy

Alpha Blending: Proper alpha handling is crucial for visual quality

Error Handling: Graceful degradation improves user experience

Modularity: Separate modes allow independent testing and development

12.4 Applications

Developed by: Group 8 - Suneela, Sara, and Abhishek

Current Use Cases:

- Educational demonstrations
- Algorithm development
- Proof-of-concept for chip authentication
- Computer vision training

Potential Extensions:

- Currency authentication systems
- Manufacturing quality control
- Retail checkout automation
- Gaming token verification

Appendices

Appendix A: Color Space Reference

HSV Color Wheel:

| | |
|------|--------------|
| 0° | = Red |
| 60° | = Yellow |
| 120° | = Green |
| 180° | = Cyan |
| 240° | = Blue |
| 300° | = Magenta |
| 360° | = Red (wrap) |

Appendix B: Performance Benchmarks

Test Environment:

- CPU: Intel i7-10700K @ 3.8GHz
- RAM: 16GB DDR4
- GPU: Not utilized (CPU-only)
- OS: Windows 11
- Python: 3.11.9

Appendix C: Code Metrics

| Metric | Value |
|---------------------|---------|
| Total Lines of Code | ~1,100 |
| Total Functions | 45 |
| Total Classes | 4 |
| Code Comments | 25% |
| Documentation | 8 files |

Appendix D: References

OpenCV Documentation: <https://docs.opencv.org/>

NumPy Documentation: <https://numpy.org/doc/>

HSV Color Space: https://en.wikipedia.org/wiki/HSL_and_HSV

Alpha Compositing: https://en.wikipedia.org/wiki/Alpha_compositing

STB600 Final Project Specifications (2025)

Document Version: 1.0.0

Last Updated: December 14, 2025

Authors: Suneela, Sara and Abhishek - Group 8

Status: Complete