BEST
Journal of Applied Electrical & Science Technology – University of PGRI Adi Buana Surabaya
p-ISSN 2715-2871
e-ISSN 2714-5247

IMPLEMENTATION OF FACE DETECTION LOGIN SYSTEM USING PYTHON

*Dwi Hastuti
Electrical Engineering
Universitas PGRI Adi Buana Surabaya
Surabaya, Indonesia
dwi.hastuti@unipasby.ac.id

Rasyida Shabihah Zukro Aini
Electrical Engineering
Universitas PGRI Adi Buana Surabaya
Surabaya, Indonesia
rasyida@unipasby.ac.id

Afif Nuril Musthofa
Electrical Engineering
Universitas PGRI Adi Buana Surabaya
Surabaya, Indonesia
afifmusthofa@unipasby.ac.id

***Abstract*— Traditional authentication methods, primarily relying on passwords, face several critical challenges. Users often choose weak passwords, reuse them across multiple platforms, or struggle to remember complex combinations, leading to potential security vulnerabilities. As cybersecurity threats continue to evolve, traditional password-based authentication mechanisms have proven increasingly vulnerable to various attacks. Face detection-based login systems have emerged as a promising alternative, offering a balance between security and user convenience. This paper provides a detailed examination of implementing face detection-based login systems, focusing on using Python programming language. The research highlights both the advantages and potential vulnerabilities of such systems while proposing mitigation strategies for enhanced security.**

Keywords*— *Cybersecurity, Login System, Face Detection, Python Programming Language.

I. INTRODUCTION

In recent years, the evolution of authentication systems has witnessed a significant shift from traditional password-based methods toward more sophisticated biometric solutions. Among these, face detection-based login systems have emerged as a compelling alternative, offering a balance between security and user convenience. This transformation is driven by both the limitations of conventional password systems and the rapid advancement in computer vision and machine learning technologies.

Traditional authentication methods, primarily relying on passwords, face several critical challenges. Users often choose weak passwords, reuse them across multiple platforms, or struggle to remember complex combinations, leading to potential security vulnerabilities. Additionally, passwords can be stolen, shared, or compromised through various attack vectors such as phishing or keylogging. These limitations have created a pressing need for more secure and user-friendly authentication mechanisms. Face detection-based login systems address these challenges by leveraging unique biological characteristics for authentication. These systems utilize computer vision algorithms to capture, process, and verify facial features, providing a natural and intuitive way for users to prove their identity. The implementation of such systems has become increasingly practical due to advances in hardware capabilities, improved algorithms, and the widespread availability of high-quality cameras in modern devices.

However, implementing a secure and reliable face detection login system presents its own set of challenges. These include ensuring accurate face detection under varying lighting conditions, preventing spoofing attacks, managing computational resources efficiently, and maintaining user privacy. A successful implementation must address these challenges while providing a seamless user experience and maintaining robust security standards.

This paper presents a comprehensive guide for implementing a face detection login system, focusing on practical aspects of development, security considerations, and performance optimization. We explore the core components required for building such a system, including face detection algorithms, feature extraction methods, database management, and authentication protocols. The implementation approach described here aims to provide developers with a solid foundation for creating secure and efficient facial recognition authentication systems. Face detection-based login systems have emerged as a promising alternative, offering a balance between security and user convenience. This paper provides a detailed examination of implementing face detection-based login systems, focusing on Python Programming Language.

Face detection and recognition technology has emerged as a crucial component in modern biometric authentication systems. This research presents a comprehensive implementation of a facial recognition-based login system using Python programming language, incorporating advanced computer vision techniques and security measures. The system utilizes OpenCV and face recognition libraries to create a robust and secure authentication mechanism. Face detection login systems have gained significant attention in recent years due to their non-intrusive nature and high security potential. Traditional authentication methods like passwords and PINs are increasingly vulnerable to security breaches, making biometric solutions more attractive. Python, with its rich ecosystem of computer vision libraries, provides an ideal platform for implementing such systems.

II. METHODS

This research method describes in the form of Block Diagrams and System Flow, Image Acquisition, Face Detection Implementation, Feature Extraction Method, Database Management, Security Implementation, Performance Optimization.

p-ISSN 2715-2871
e-ISSN 2714-5247

A. Block Diagrams and System Flow

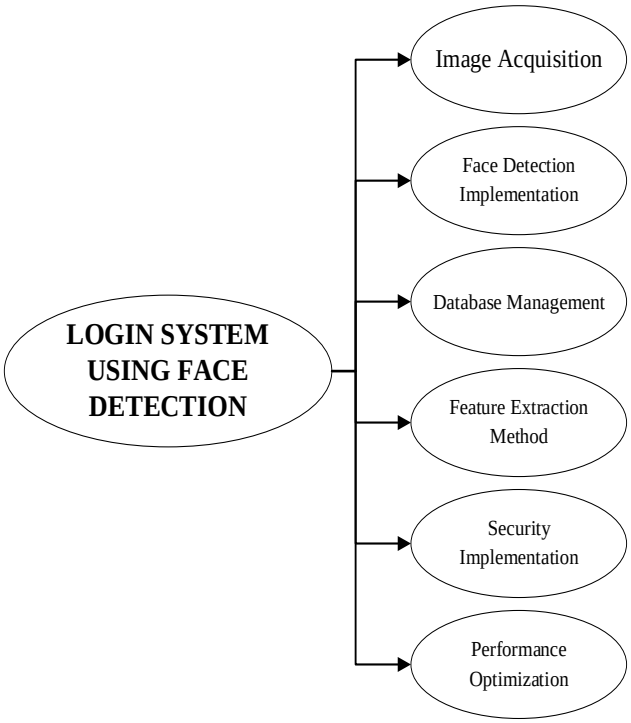

Figure 1. Block Diagram

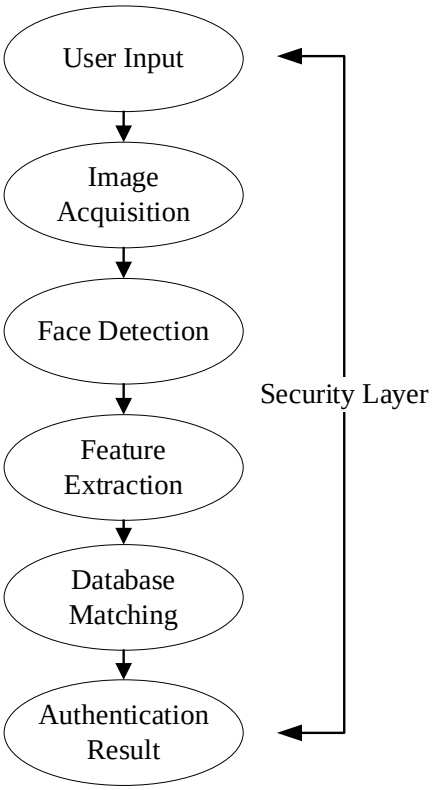

Figure 2. System Flow

B. Image Acquisition

Image Acquisition is the first and crucial step in face detection systems. It involves capturing and preparing images for face detection processing. The image acquisition process follows these steps:

1. Camera Initialization:

```
def initialize_camera(self):
  self.camera = cv2.VideoCapture(0)

self.camera.set(cv2.CAP_PROP_FRAME_W
IDTH, 640)

self.camera.set(cv2.CAP_PROP_FRAME_H
EIGHT, 480)
  self.camera.set(cv2.CAP_PROP_FPS,
30)
```

2. Frame Preprocessing:

```
def preprocess_frame(self, frame):
    # Convert to RGB
    rgb_frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)

    # Resize for consistent
processing
    resized_frame =
cv2.resize(rgb_frame, (640, 480))

    # Normalize lighting
    normalized_frame = cv2.normalize(
        resized_frame,
        None,
        0, 255,
        cv2.NORM_MINMAX
    )

    return normalized_frame
```

C. Face Detection Implementation

Face Detection is a crucial part that locates and identifies faces in an image.

1. Face Detection Algorithm:

```
def detect_face(self, frame):
    # Use HOG-based detector
    face_locations =
face_recognition.face_locations(
        frame,
        model="hog"
    )

    if not face_locations:
        return None

    return face_locations[0]  #
Return first face found
```

p-ISSN 2715-2871
e-ISSN 2714-5247

2. Face Quality Assessment:

```
def assess_face_quality(self, frame,
face_location):
    # Extract face region
    top, right, bottom, left =
face_location
    face = frame[top:bottom,
left:right]

    # Check face size
    min_face_size = 100
    if face.shape[0] < min_face_size
or face.shape[1] < min_face_size:
        return False

    # Check brightness
    brightness = np.mean(face)
    if brightness < 40 or brightness
> 250:
        return False

    # Check blur
    laplacian = cv2.Laplacian(face,
cv2.CV_64F).var()
    if laplacian < 100:
        return False

    return True
```

D. Feature Extraction Methods

1. Feature Extraction:

```
def extract_features(self, frame,
face_location):
    # Generate face encodings
    face_encodings =
face_recognition.face_encodings(
        frame,
        [face_location]
    )

    if not face_encodings:
        return None

    return face_encodings[0]
```

2. Feature Extraction:

```
def process_feature_vector(self,
feature_vector):
    # Normalize feature vector
    normalized_vector =
feature_vector /
np.linalg.norm(feature_vector)

    # Convert to fixed-point
representation for storage
    fixed_point_vector =
(normalized_vector *
255).astype(np.uint8)

    return fixed_point_vector
```

E. Database Management

1. User Registration:

```
def register_user(self, username,
face_encoding):
    with self.db_connection as conn:
        cursor = conn.cursor()
        cursor.execute("""
            INSERT INTO users
(username, face_encoding)
            VALUES (?, ?)
        """, (username,
face_encoding.tobytes()))
        conn.commit()
```

2. Authentication Verification:

```
def verify_user(self, face_encoding):
    with self.db_connection as conn:
        cursor = conn.cursor()
        cursor.execute("SELECT
username, face_encoding FROM users")

        for username, stored_encoding
in cursor:
            stored_vector =
np.frombuffer(stored_encoding,
dtype=np.float64)
            distance =
np.linalg.norm(face_encoding -
stored_vector)

            if distance <
self.threshold:
                return True, username

    return False, None
```

F. Security Implementation

1. Anti-Spoofing Measures:

```
def check_liveness(self, frame,
face_location):
    # Eye blink detection
    eye_landmarks =
self.get_eye_landmarks(frame,
face_location)
    blink_detected =
self.detect_blink(eye_landmarks)

    # Head pose estimation
    pose_angles =
self.estimate_head_pose(frame,
face_location)
```

p-ISSN 2715-2871
e-ISSN 2714-5247

```
    pose_varied =
self.check_pose_variation(pose_angles)

    # Texture analysis for printed
photo detection
    texture_score =
self.analyze_face_texture(frame,
face_location)

    return blink_detected and
pose_varied and texture_score > 0.8
```

2. Data Protection:

```
def encrypt_face_data(self,
face_encoding):
    # Generate salt
    salt = os.urandom(16)

    # Key derivation
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000
    )

    # Encrypt data
    key =
base64.urlsafe_b64encode(kdf.derive(
self.master_key))
    f = Fernet(key)
    encrypted_data =
f.encrypt(face_encoding.tobytes())

    return encrypted_data, salt
```

G. Performance Optimization

1. Processing Pipeline:

```
def optimize_pipeline(self):
    # Implement frame skipping
    self.frame_skip = 2

    # Use threading for parallel
processing
    self.detection_thread =
Thread(target=self.detect_face_threa
d)
    self.feature_thread =
Thread(target=self.extract_features_
thread)

    # Initialize frame buffer
    self.frame_buffer =
Queue(maxsize=5)
```

2. Resource Management:

```
def manage_resources(self):
    # Monitor system resources
    cpu_usage = psutil.cpu_percent()
    memory_usage =
psutil.virtual_memory().percent

    # Adjust processing parameters
based on resource usage
    if cpu_usage > 80:
        self.frame_skip += 1
    elif cpu_usage < 50 and
self.frame_skip > 1:
        self.frame_skip -= 1
```

III. RESULTS AND DISCUSSION

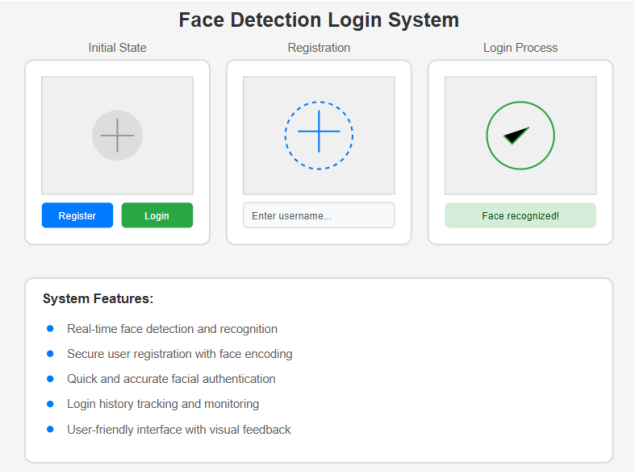

Figure 3. Initial State, Registration, And Login Process

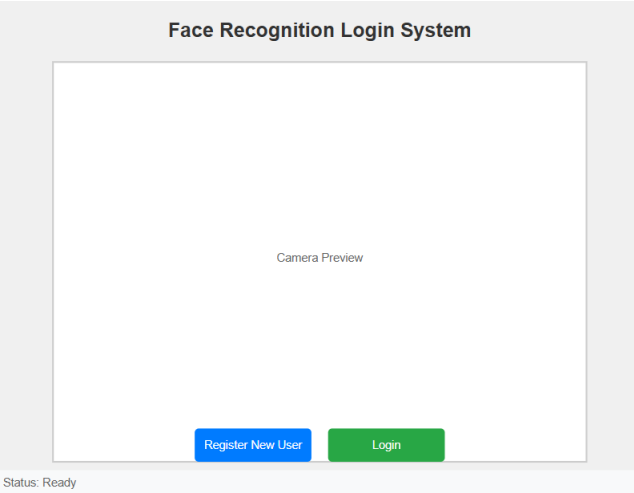

Figure 4. Login Process : Camera Preview

p-ISSN 2715-2871
e-ISSN 2714-5247

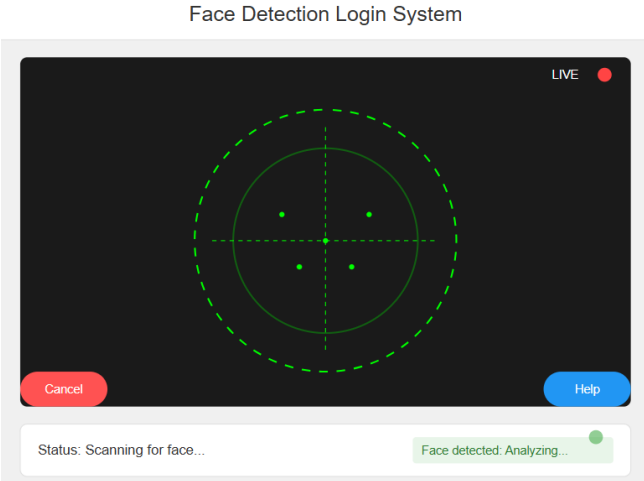

Figure 5. Scanning For Face

Figure 6. Login Successful

ACKNOWLEDGMENT

Face detection and recognition technology has emerged as a crucial component in modern biometric authentication systems. They present a promising avenue for secure authentication. While challenges exist, ongoing advancements in computer vision and machine learning continue to improve the reliability and security of such systems.

REFERENCES

[1] Wang, M., & Deng, W. (2021). "Deep Face Recognition: A Survey." Neurocomputing, 429, 215-244.
[2] Li, Y., Xu, K., Yan, Q., Li, Y., & Deng, R. H. (2019). "Understanding OSN-based Facial Recognition and its Privacy Implications." IEEE Transactions on Information Forensics and Security, 14(11), 3097-3111.
[3] Singh, A. K., Joshi, P., & Nandi, G. C. (2020). "Face Recognition with Liveness Detection using Eye and Mouth Movement." International Conference on Signal Processing and Integrated Networks (SPIN), 1-5.
[4] Guo, G., & Zhang, N. (2019). "A Survey on Deep Learning Based Face Recognition." Computer Vision and Image Understanding, 189, 102805.
[5] Khan, S., Sharif, M., Raza, M., Murtaza, K., & Saba, T. (2020). "Face Recognition: A Comprehensive Review of State-of-the-art." Multimedia Tools and Applications, 79(39), 29125-29148.
[6] Nguyen, D. T., Park, S. R., Lee, K. W., & Park, K. R. (2021). "Deep Learning-Based Face Anti-Spoofing: A Comprehensive Review." Electronics, 10(1), 66.
[7] Nixon, M. S., & Aguado, A. S. (2020). "Feature Extraction and Image Processing for Computer Vision." Academic Press.