Published by :
http://www.ijert.org

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

A Computer Vision Approach to Gesture-Based Game Control and Fatigue Tracking in Multiplayer Environments

Prof. Suvarna Thakur
Dept of Information Technology
DBATU, Lonere, Maharashtra

Shreya More
Dept of Information Technology
DBATU, Lonere, Maharashtra

Atharva Shinde
Dept of Information Technology
DBATU, Lonere, Maharashtra

Rajratna Kamble
Dept of Information Technology
DBATU, Lonere,Maharashtra

Sahil Kamble
Dept of Information Technology
DBATU,Lonere, Maharashtra

Dr. Shivajirao Jadhav
Dept of Information Technology
DBATU, Lonere, Maharashtra

***Abstract*—This paper introduces a new system that uses OpenCV and Python to control the well-known game Flappy Bird through natural user interfaces, including hand gestures, head movements (both single and multiplayer), and fatigue cues. Using a webcam, the system records live video, recognizes pertinent user movements (such as hand raises and head tilts), tracks indications of weariness, and converts these signals into commands for the game. Pygame is used to implement the Flappy Bird game, which enables the bird to flap in response to user gestures. This method shows how gaming experiences can be improved by vision-based inputs. We outline the OpenCV package integration, implementation modules, modeling and analysis of facial cues and gestures, and system architecture. Benefits, drawbacks, and future research—such as more reliable detection and user adaptation—are covered in the conclusions.**

I. INTRODUCTION

It has been demonstrated that vision-based inputs, like head movements or gestures, can supplement or even replace traditional controllers. Our goal is to use human behavior interpretation to control the popular simple game Flappy Bird. We use OpenCV for computer vision processing and the Pygame library for game implementation. The system can identify signs of weariness like yawning or drowsiness, as well as hand gestures like raising a hand and head positions like tilting or nodding. Each player can use their own gestures to control their own bird in both single-player and splitscreen multiplayer modes. Inspired by studies that identify eye closure and yawning as indicators of drowsiness, the motivation includes investigating new game interfaces and implementing a fatigue warning system to guarantee player safety.

a) Conventional computer games mainly use touch-basedcontrols, keyboards, and mice. But thanks to developments in computer vision, hands-free interaction and increased user engagement are now possible with natural user interfaces that use gestures and facial expressions. The popular game Flappy Bird has a vision-based control system presented in this paper that uses real-time webcam input to control the bird rather than manual input.:

b) The suggested system recognizes hand gestures, head movements, and fatigue indicators using OpenCV and MediaPipe. Intuitive gameplay is made possible by mapping these inputs to in-game actions like ”flap” and ”pause.” The game engine, which was developed with Pygame, simulates the motion of the bird by interpreting these signals in real time.

II. LITERATURE SURVEY

1. Google’s MediaPipe[2] framework introduced lightweight, real-time hand and facial landmark tracking. MediaPipe Hands outputs 21 3D landmarks per hand with high accuracy, making it ideal for gesture detection and classification. MediaPipe Face Mesh detects 468 facial landmarks, allowing for advanced facial analysis such as head pose estimation and fatigue tracking.
2. In the context of gesture recognition, research bySaponas[5]. demonstrated how hand movements could be effectively recognized using surface electromyography and computer vision techniques. While electromyography offers precision, vision-based approaches using hand landmarks (such as those provided by MediaPipe) are more accessible and hardware-independent.
3. Fatigue detection systems have been extensively studied in driver monitoring scenarios. According to Parvez[4], prolonged eye closure, frequent yawning, and slow blink rates are strong indicators of drowsiness. They proposed a hybrid model integrating OpenCV’s Deep Neural Network (DNN) modules with convolutional neural networks for realtime fatigue detection. Although their work targets automotive safety, the principles can be adapted to game environments to promote healthy usage behavior.
4. In gesture-controlled games, Pygame has been frequentlyused to implement prototypes due to its simplicity and support for custom input events. Integrating OpenCV-based gesture detectors with Pygame allows for hands-free gameplay, as demonstrated in works like “Vision-Based Game Control using Webcam” by Ghosh[6], where players used hand gestures to control a spaceship.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

5. Multiplayer vision-based control is relatively underexplored. Most systems focus on single-user input, but splitscreen or dual-camera systems can extend gesture detection to multiple players. Accurate face tracking and pose estimation for each user enables separate control channels for multiplayer modes.
6. From the reviewed literature, it is evident that whilecomponents like gesture recognition, head pose tracking, and fatigue detection are well-researched individually, there is limited work integrating all these modules into a cohesive, real-time interactive system for gaming. Our work aims to fill this gap by combining hand, head and fatigue detection into a single architecture for automating Flappy Bird gameplay.

III. CONCEPTUAL OVERVIEW

Mapping human actions captured by a live camera to game controls is the main concept. Certain gestures in our system are equivalent to Flappy Bird's "flap" action. For instance, a head tilt or a raised hand can cause a jump. Additionally, we track head drooping and facial cues like yawning or frequent eye blinks to keep an eye on user fatigue. A Pygame loop that shows pipes and a bird sprite is run by the game engine. The system sends a command to cause the bird to fly upward whenever it detects a gesture. The bird is controlled by a single player in single-player mode. In a multiplayer mode, the video frame is split or two cameras are used so that two players can simultaneously use gestures to control two birds. Conceptually, this pipeline follows the pattern: *Webcam Input → Feature Detection → Command Generation → Game Engine*. Modern libraries like OpenCV provide pretrained classifiers for detecting faces, eyes, and hands, and even landmark-based tracking via MediaPipe that can run real-time on consumer hardware. These technologies enable our system to achieve robust real-time tracking. The system is designed to be user-adaptive: before gameplay, each user can calibrate gesture sensitivity and define control modes (e.g. enable/disable trigger, adjust head-tilt thresholds).

IV. MODELING AND ANALYSIS

We model each control modality as follows. Hand Gesture Detection: We detect open palm or hand raises using either color-based segmentation or machine learning models like MediaPipe Hands. Specific gestures (e.g. open palm vs fist) can be mapped to actions. Head Position Tracking: We detect the face region (using Haar cascades or a DNN face detector) and estimate head pose. A sufficient upward nod or tilt above a threshold is treated as a "flap" command. For multiplayer, two faces are tracked separately. Fatigue Detection: We track things like how often you blink, how long you keep your eyes closed, and how open your mouth is (yawning). Prolonged eye closure, forward head nodding, and yawning are recognized signs of drowsiness. The game may pause or display a warning if these surpass predetermined thresholds. We also take into account game adaptation; for example, the game may slow down or recommend a break if it detects player fatigue. To prevent spurious triggers, each input channel in the analysis generates a binary signal (flap/no-flap or alert), which is then filtered over a brief period of time.

V. USER ADAPTATION

The ability of natural user interfaces to adjust to the physical characteristics, preferences, and interaction styles of various users is a critical component. To ensure accessibility and responsiveness, our system incorporates several user adaptation mechanisms across the input pipeline.

A. Gesture Calibration

Before gameplay, the system allows a brief calibration phase in which the user performs sample gestures (e.g., hand raise, head tilt). During this phase, the system records baseline positions and dynamic ranges specific to the user's posture, hand height, and facial proportions. For example, the Ycoordinate threshold for detecting a hand raise can be adjusted based on the user's height and camera angle.

B. Control Mode Selection

Users can select from multiple input modalities depending on their comfort or environment:

- Hand Gesture Mode: Uses hand raises or waves to trigger the flap.
- Head Movement Mode: Detects head nods or tilts as input.
- Multimodal Mode: Accepts input from any enabled gesture source.

This flexibility ensures that users with varying levels of mobility, expression range, or environmental constraints (e.g., low light) can still interact effectively.

C. Fatigue-Aware Gameplay

The system continuously tracks user fatigue indicators like blink frequency, mouth openness (a sign of yawning), and eye closure duration. If persistent drowsiness is observed, the game either:

- Slows down obstacle speed to reduce cognitive load,
- Displays an alert encouraging the user to take a break, or
- Pauses the game temporarily to avoid strain.

These adaptive responses not only enhance user experience but also promote healthier gameplay habits.

D. Multiplayer Adaptation

In multiplayer mode, the system distinguishes between two players by segmenting the webcam feed or using multiple cameras. Each user undergoes individual calibration to personalize gesture thresholds. The system maintains separate gesture detection pipelines to ensure input separation and fairness during simultaneous play.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

E. Feedback and Responsiveness

To improve user experience, the system provides real-time visual feedback (e.g., on-screen cues or debug markers) during gesture detection. These cues help users adjust their posture or expression and confirm when an input has been successfully registered.

Overall, the user adaptation features of the system make it robust, inclusive, and more enjoyable for a diverse set of players in both casual and experimental settings.

VI. SYSTEM ARCHITECTURE

The system follows a modular architecture. The Video Capture Module acquires frames from the webcam. Each frame passes through parallel Detection Modules:

- Hand Gesture Module: Processes the frame to detect and recognize hand gestures using skin-color segmentation or a neural network (MediaPipe).
- Face: Uses OpenCV's Haar cascade or a CNN-based face detector to locate the face.
- Head Pose Module: Estimates the head orientation from facial landmarks (MediaPipe Face Mesh with 468 landmarks) or using solvePnP with detected facial features to compute pitch and yaw.
- Fatigue Module: Tracks temporal facial features (eye aspect ratio, mouth opening) to detect blinks or yawns.
- Game Interface Module: Receives binary commands (flap) from the detectors and sends inputs to the Flappy Bird game loop implemented in Pygame.

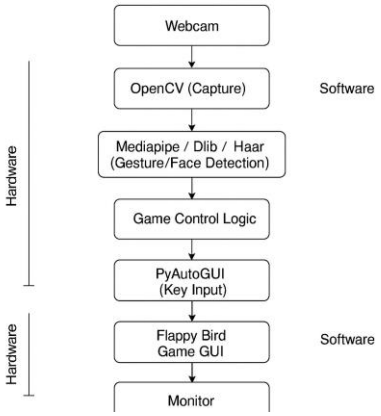

Fig. 1. System Architecture

Shared data structures allow these modules to communicate: detected events set flags that the game loop looks at every frame. Real-time processing is prioritized in the architecture, with detections optimized to operate at least 15–30 frames per second. Every pertinent module makes use of OpenCV features like detectMultiScale (for faces). The location of the bird and the surroundings are continuously updated by the Game Engine module (Pygame). Two instances of the Detection Modules operate simultaneously in multiplayer mode on separate input regions or with different cameras, each of which transmits commands to the corresponding game instance.

VII. MODULES

The software is organized into the following key modules:

- Hand Gesture Recognition: Captures the hand contour or landmarks. For example, an open-palm detected above the shoulder height indicates a flap command. We use MediaPipe Hands or a simple color threshold plus contour logic to identify hand raise. The landmark model provides 21 points per hand in real-time.
- Face Detection: Utilizes OpenCV's pre-trained Haar cascades for face detection. When a face is detected, the mouth region is analyzed. This is a discrete check each frame (binary yes/no).
- Head Pose Tracking: Once the face is detected, we estimate head tilt. We compute the slope of the line between eyes or use a PnP algorithm on facial landmarks. If the head tilts upward beyond a threshold angle, a flap is issued. In multiplayer, separate face detections produce separate head pose outputs.
- Fatigue Monitoring: We assess signs of fatigue on a regular basis. We count blinks or eye closures using the eye aspect ratio or the location of the eyelids at landmarks. A mouth aperture is measured by a yawning detector. A fatigue score is derived from these metrics. For instance, the system detects fatigue if the number of yawns per minute or the eye closure duration X seconds surpasses a threshold. This data can be utilized to modify gameplay or remind the user to take a break.
- Flappy Bird Game (Pygame): The main game logic is implemented in Python using the Pygame library. The bird sprite falls under gravity and ascends when a command arrives. Pipes scroll from right to left, and collision detection ends the game on hit. The score increments each time a pipe is passed. This module runs continuously and polls flags set by the input modules to perform jumps.

VIII. ALGORITHMS USED

This system utilizes a combination of classical computer vision algorithms and real-time landmark detection models to interpret user gestures and expressions for gameplay control. The following algorithms and techniques were integrated:

A. Haar Cascade Classifier

The module uses OpenCV's pre-trained Haar cascade classifier. It applies the Viola–Jones object detection framework to detect facial regions and identify a mouth curvature and feature positions. The classifier is fast and efficient, making it suitable for real-time applications.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

B. Eye Aspect Ratio (EAR) for Fatigue Detection

Fatigue is monitored by calculating the Eye Aspect Ratio (EAR), which measures the vertical and horizontal distances between eye landmarks:

$$EAR = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2 \cdot ||p_1 - p_4||}$$

If the EAR falls below a predefined threshold for a sustained period, it is interpreted as eye closure or drowsiness. This helps in pausing the game or issuing fatigue warnings.

C. MediaPipe Hand Landmark Detection

For hand gesture recognition, Google's MediaPipe Hands solution is used. It detects 21 hand landmarks in real time and classifies the gesture based on the position and relative distance of fingertips. For example, if the index finger tip is significantly raised above the wrist, it is interpreted as a flap command.

D. Head Pose Estimation using Facial Landmarks

Head movements (nods or tilts) are interpreted using 3D head pose estimation. The algorithm fits a 3D model to 2D facial landmarks (e.g., nose tip, chin, eyes) and estimates rotation vectors using the SolvePnP method in OpenCV. A tilt exceeding a calibrated angle threshold is mapped to a flap command.

E. MediaPipe Face Mesh for Landmark Extraction

The system uses MediaPipe Face Mesh to obtain 468 highprecision facial landmarks. These landmarks form the basis for both fatigue detection (EAR, mouth openness) and head pose estimation. Its lightweight and fast performance makes it suitable for real-time gesture-based control.

F. Gesture-to-Game Event Mapping

The detected gestures (hand raised, head tilt, fatigue) are translated into game events using a rule-based mapping logic. Each frame is analyzed and associated flags are raised, which are then read by the game engine to trigger corresponding in-game actions like jump, pause, or reset.

These combined algorithms enable accurate, real-time interpretation of user inputs for gesture-controlled gameplay, offering a natural and immersive interaction model.

IX. ADAPTIVE CONTROL AND USER PERSONALIZATION

Our system allows users to adjust settings for better performance. Prior to playing, a brief calibration can be done: the user can set the baseline head position and the threshold for detecting a tilt or raise. The system also offers different control modes: e.g., *Hand-Only*, *Head-Only*, or *Face* mode, depending on user preference. This adaptation ensures that players with different heights, hand sizes, or movement ranges can comfortably use the interface. Additionally, the game difficulty can adapt to fatigue signals: for instance, if a player shows signs of drowsiness, the game can slow down or pause, encouraging a break. Such user-centered design improves accessibility and comfort.

X. PACKAGES AND TOOLS

The implementation uses several key open-source packages:

- Python with OpenCV: We use OpenCV (version 4.x) for all computer vision tasks. OpenCV is a comprehensive open-source computer vision library that supports many functionalities like image filtering, feature detection, and machine learning. It includes pre-trained classifiers for face, eye detection.
- MediaPipe: Google's MediaPipe provides lightweight real-time hand and face landmark models. MediaPipe Hands yields 21 3D hand landmarks in real time, even on mobile processors. MediaPipe Face Mesh provides 468 facial landmarks, enabling precise head pose and expression analysis.
- Pygame: A Python game framework used to recreate Flappy Bird. Pygame simplifies rendering graphics and handling keyboard/mouse events.
- NumPy: For numerical operations on image arrays.
- Dlib (optional): Could be used for more accurate face landmark detection, though MediaPipe suffices.

These packages together form a toolchain for realtime vision-based game control. For example, OpenCV's detectMultiScale function can identify faces in each frame, with parameters tuned for speed.

XI. METHODOLOGY

The methodology consists of several steps:

1. Video Capture: We capture live video frames using OpenCV. Each frame is resized (e.g., to 640x480) to balance speed and accuracy.
2. Preprocessing: The frame is converted to grayscale for cascade detectors and a blurred version is optionally used to reduce noise.
3. Face Detection & Landmark Extraction: We apply a Haar cascade to find the face region in the frame. Within the detected face, we locate eyes and mouth. Alternatively, we use MediaPipe Face Mesh to get 468 landmarks and compute head pose via solvePnP. The normalized mouth opening determines eye aspect ratio computes blink.
4. Hand Detection: In parallel, the image is processed to find hands. Using MediaPipe Hands, we detect up to two hands and extract keypoints. A quick heuristic (e.g. palm orientation or vertical position) decides if a "raise" gesture occurred.
5. Gesture Classification: The outputs of detection are combined. For example, if hand landmarks indicate an open palm above a certain Y-coordinate, we set flap_flag=1. Similarly, if the user's (mouth landmarks show a wide grin) or nods head upward beyond a threshold angle, we set flap_flag=1.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

6. Fatigue Monitoring: Independently, eye closure over multiple frames increments a fatigue counter. Yawn detection (large mouth opening for several frames) triggers a drowsiness flag. If fatigue crosses a limit, we may reduce game speed or notify the user.

7. Game Control: The Flappy Bird Pygame loop checks the flap_flag each frame. If set, it makes the bird jump and resets the flag. Gravity then pulls the bird down in subsequent frames. Pygame updates the display and checks for collisions or pipe passage.

This process repeats at interactive speed. Critical steps (face/hand detection) use optimized OpenCV/MediaPipe implementations, achieving real-time performance (tens of FPS) on typical PCs. An example pipeline flow.

XII. RESULTS AND DISCUSSION

We implemented a prototype on a standard PC with a webcam. In single-player tests, the system reliably mapped gestures to game actions. Players could make the bird fly by either raising their hand or simply smiling, whichever was more intuitive. The face detection operated robustly under normal indoor lighting (OpenCV Haar cascades detect frontal faces in real time). Head tilt controls also worked, though with slightly more calibration needed per user. In multiplayer mode, two users placed side-by-side each controlled a separate bird by their own gestures; we achieved near-simultaneous tracking using OpenCV or two webcams. The latency from gesture to flap was typically under 200ms, providing smooth gameplay.

Fatigue detection added a useful safety feature. For instance, when a user yawned continuously, the system flagged drowsiness (consistent with findings that yawning frequency correlates with fatigue). Future work could refine this by, e.g., counting yawns over time as suggested by Parvez.

One limitation is environmental sensitivity: varying lighting or background can affect detection accuracy. Also, competing gestures (e.g. waving vs raising) require careful gesture definitions. Nonetheless, the modular approach allows replacing classifiers with deep-learning models if needed. Overall, the results indicate that hand and head gestures can effectively replace keyboard input for Flappy Bird, and the additional modules (fatigue) enrich the interaction.

Fig. 2. System Interface Of Flappy Bird Game

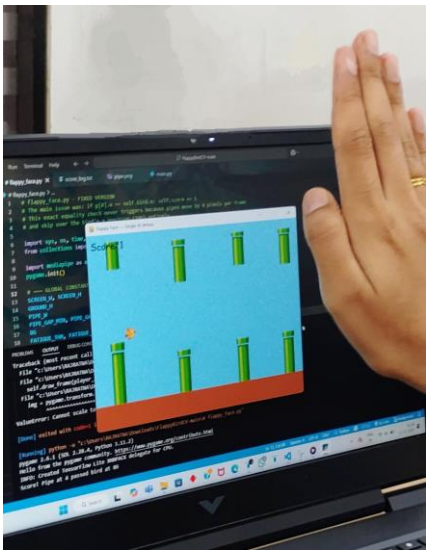

Fig. 3. Hand Gesture Detection Triggering Flap

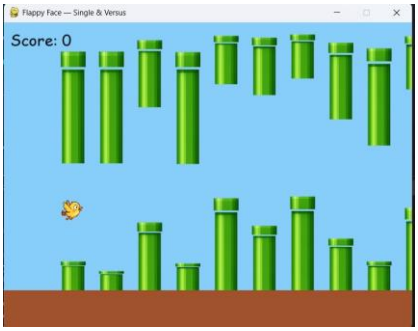

Fig. 4. Head Detection in Single Player Mode

Fig. 5. Head Detection in Multiplayer Mode P1

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 07, July - 2025

Fig. 6. Head Detection in Multiplayer Mode P2

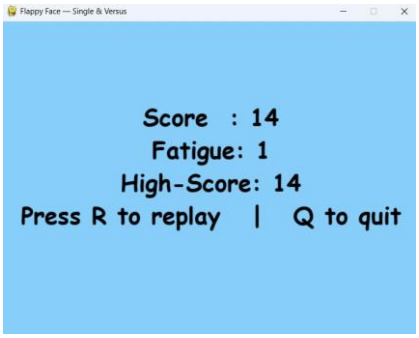

Fig. 7. Fatigue Detection In Flappy Bird Game

XIII. CONCLUSION

This research successfully study the effective illustration of how real-time computer vision techniques can automate the Flappy Bird game, providing a novel touchless control experience. The suggested framework offers a fresh take on interactive gaming that transcends traditional keyboard or touch interfaces by combining hand gesture recognition, head pose detection, and fatigue monitoring into a single system. For high-performance landmark detection, the system makes use of MediaPipe and OpenCV, allowing for responsive and fluid control over in-game actions. Intuitive control mechanisms are provided by head and hand movements. Furthermore, yawn and eye aspect ratio (EAR) fatigue detection adds a safety feature that is frequently overlooked in casual games: it pauses or modifies gameplay in response to user drowsiness.

The multimodal control scheme improves accessibility, enabling users with motor impairments or those in hands-busy environments to play the game, according to experimental implementation. Additionally, split-camera or segmented feed techniques were successfully used to demonstrate multiplayer support, enabling two-player interaction without the need for input devices.

XIV. FUTURE SCOPE

Using computer vision, the current system effectively illustrates a different control method for the Flappy Bird game. Nonetheless, there are numerous chances to improve and broaden the gaming experience. Using deep learning to apply sophisticated gesture recognition techniques is one important avenue to increase the precision of hand and facial input detection and lower false positives while playing games.Adaptive difficulty, in which the game modifies obstacle speed or gap size according to the player's current performance or degree of fatigue, may also be made possible by integrating reinforcement learning.

With the use of sophisticated face tracking and split-screen logic, the system can be extended to accommodate more than two players in multiplayer mode. Gameplay could become even more inclusive and interactive by implementing voice commands in addition to gestures, resulting in a multimodal control experience.In order to help players fine-tune their movements more precisely, the game may also offer visual or aural feedback when a gesture is detected.

XV. ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the Department of Information Technology, Dr. Babasaheb Ambedkar Technological University, Lonere, for providing the necessary infrastructure and academic support throughout the duration of this research.

Finally, we extend our appreciation to the developers and open-source communities behind OpenCV, MediaPipe, and Pygame for providing powerful and accessible tools that made the practical execution of this vision-based automation system possible.

REFERENCES

[1] OpenCV Documentation, "OpenCV: Open Source Computer Vision Library," https://opencv.org/, Accessed: July 2025.
[2] Google, "MediaPipe Hands: Real-time Hand Tracking," https://google.github.io/mediapipe/solutions/hands.html, Accessed: July 2025.
[3] P. Lu, Y. Chen, X. Zeng, and Y. Wang, "A Vision Based Game Control Method," in Computer Vision in Human–Computer Interaction, LNCS vol. 3766, pp. 70–78, Springer, 2005.
[4] M. M. Parvez, R. Yadav, and S. Das, "Advanced Driver Fatigue Detection using OpenCV and Deep Learning," in Engineering Proceedings, vol. 34, no. 1, p. 15, 2023.
[5] T. S. Saponas, D. Tan, and D. Morris, "Demonstrating the feasibility of using bio-acoustic sensing to control games," in Proceedings of the ACM CHI Conference on Human Factors in Computing Systems, 2009.
[6] S. Ghosh and A. Kumar, "Vision-Based Game Control using Webcam and Hand Gestures," in International Journal of Computer Applications, vol. 179, no. 44, pp. 15–19, 2018.
[7] T. Soukupova and J.´ Cech, "Real-Time Eye Blink Detection Using Facial˘ Landmarks," in Computer Vision Winter Workshop, 2016.