Silesian
University
of Technology

FINAL PROJECT

Smart 9-Ball Assistant

Sara SOBSTYL

Student identification number: ⟨**306088**⟩

Programme: Informatics

Specialisation: ⟨CGS⟩

SUPERVISOR

⟨**Dr. hab. inż. Jakub Nalepa**⟩

DEPARTMENT ⟨**Katedra Algorytmiki i Oprogramowania**⟩

Faculty of Automatic Control, Electronics and Computer Science

Gliwice 2025

Thesis title

Smart 9-Ball Assistant

Abstract

A project that helps players during a game of 9-ball billiards by showing them how the white ball would likely move after a shot. The system has two main parts: 1. Phone App on the Cue Stick - the app that uses the phone's motion sensors (like gyroscope and accelerometer) to track how the cue stick is held and moved. The app sends this data in real time to a computer. 2. Camera Above the Table + Computer - A camera above the billiard table sends live video to a computer. There, a program using e.g., OpenCV detects where all the balls are based on their colors. It figures out where the cue ball is and updates the layout of the table.

Key words

(2-5 keywords, separated by commas)

Tytuł pracy

Inteligentny asystent do bilarda

Streszczenie

(Streszczenie pracy – odpowiednie pole w systemie APD powinno zawierać kopię tego streszczenia.)

Słowa kluczowe

(2-5 slow (fraz) kluczowych, oddzielonych przecinkami)

Contents

Chapter 1

Introduction

Billiards, specifically the 9-ball variation, is a sport that demands a high degree of precision, spatial awareness, and consistent motor control. Unlike many other sports where physical athleticism is paramount, billiards is effectively a game of applied geometry and physics executed through fine muscle memory.

1.1 Introduction into the problem domain

The mastery of billiards relies on two distinct skill sets: the ability to visualize the correct trajectory of the balls and the physical ability to execute the stroke required to send the cue ball along that trajectory. Novice players often struggle with the concept of the "Ghost Ball" [1]—the imaginary position where the cue ball must strike the object ball to pocket it.

Furthermore, the mechanics of the stroke itself involving stance, bridge stability, and wrist action significantly impact the outcome of a shot. Even a correct aim can result in a miss if the player introduces unwanted lateral acceleration or English [2] (spin) due to wrist instability during the cue delivery. With the advent of accessible Computer Vision (CV), it is now possible to digitize these physical interactions to provide real-time feedback. [7]

1.2 Settling of the problem in the domain

Current training methods in billiards are predominantly manual, relying on the subjective observation of coaches or the player's own intuition. While professional tracking systems exist (such as Hawk-Eye used in tennis or snooker) [9], they are often prohibitively expensive and require fixed, industrial-grade hardware setups. On the consumer end, mobile applications often rely solely on 2D video analysis, which lacks the depth perception required for accurate table mapping or the high-frequency sampling needed for detailed stroke analysis.

1.3 Objective of the thesis

The primary objective of this thesis is to design, implement, and test a comprehensive "Smart Pool Assistant and Stroke Analyzer". The system aims to assist players in real-time by visualizing shot outcomes and analyzing the mechanics of their cue stroke.

The specific objectives are defined as follows:

- To develop a Computer Vision module capable of detecting billiard balls and the cue stick in a live video feed.
- To implement a physics engine that calculates and visualizes the predicted trajectory of the cue ball (Tangent and Normal lines) based on the "Ghost Ball" principle.
- To create a mobile telemetry system that captures high-frequency accelerometer and gyroscope data from the player's arm.
- To derive meaningful biomechanical metrics, such as impact force ($F = ma$) and wrist stability, to provide objective feedback on the player's technique.

1.4 Scope of the thesis

The scope of this project encompasses the software and hardware integration required to build a functional prototype for the game of 9-ball.

- **Vision System:** The visual analysis is restricted to a top-down camera view. The software utilizes Python and OpenCV libraries, integrating machine learning models (Roboflow) for object detection. The trajectory prediction assumes an ideal collision model (elastic collision) and focuses on the immediate path of the cue ball and the target ball.
- **Sensor System:** The biomechanical analysis is limited to the data provided by standard Android smartphone sensors (Linear Acceleration and Gyroscope). The scope includes the development of a TCP/IP communication protocol to transfer telemetry data to the central computer for processing.
- **Hardware:** The system is designed to run on a standard consumer PC connected to a webcam and an Android device via USB (ADB forwarding).

1.5 Short description of chapters

The thesis is organized as follows:

Chapter 2 (Problem analysis) provides the theoretical background of the project. It scrutinizes the physics of billiard collisions, the geometry of "Ghost Ball" aiming, and the mechanics of inertial sensors. It also reviews existing solutions in sports technology to establish the context for the proposed system.

Chapter 3 (Requirements and tools) defines the functional and non-functional requirements of the system. It also presents the technological stack selected for the project, justifying the choice of Python, OpenCV, and the Android platform for their respective modules.

Chapter 4 (External specification) describes the system from the user's perspective. It details the user interface (UI) of the Android application and the Augmented Reality (AR) visualization displayed on the desktop, explaining the flow of interaction between the player and the assistant.

Chapter 5 (Internal specification) delves into the technical architecture and backend logic. It explains the algorithms used for object detection and trajectory prediction, the TCP/IP communication protocol, and the signal processing techniques (e.g., gravity removal) applied to the sensor data.

Chapter 6 (Verification and validation) presents the testing process. It evaluates the accuracy of the computer vision model, the latency of real-time data transmission, and the reliability of the stroke analysis metrics through unit tests and practical scenarios.

Chapter 7 concludes the thesis, summarizing the achieved objectives and proposing potential directions for future development.

Chapter 2

Problem analysis

This chapter provides the theoretical and mathematical foundation for the developed system. It analyzes the geometric complexities of billiard aiming implemented in the vision module and the biomechanical principles governing the sensor analysis. Furthermore, it reviews the current state of the art to contextualize the proposed hybrid solution.

2.1 Problem analysis

The game of 9-ball billiards presents a unique challenge where the player must translate a 3D physical intention into a 2D geometric execution. The core problems addressed by this thesis are classified into visual perception and biomechanical execution.

2.1.1 The Visual Paradox (Ghost Ball)

As defined by Alciatore, the ”Ghost Ball” (the required position of the cue ball at impact) is an imaginary point in empty space [1]. Players struggle to visualize this point because the line of aim (passing through the ghost ball center) does not coincide with the contact point on the object ball. This geometric offset forces the player to aim at a non-existent target, creating a ”visual paradox” that is difficult for novices to overcome without augmented feedback.

2.1.2 Biomechanical Inconsistency

A correct aiming line is useless if the stroke delivery is flawed. The mechanics of the cue stroke are critical to the outcome of the shot.

Ballistic vs. Tetanic Stroke: According to the analysis by Moore, a fundamental distinction exists between amateur and professional stroke mechanics [15]. Amateur players often utilize a ”ballistic” stroke characterized by a sudden ”quick yank” at the start of the movement. This approach makes velocity control at the moment of impact unre-

liable. In contrast, professionals strive for a "tetanic" stroke—a constant state of muscle contraction—which maintains a consistent accelerating force throughout the cue delivery [15].

Lateral Deviation: Another critical factor is the stability of the swing axis. Failure to maintain a vertical "pendulum swing" (keeping the elbow fixed in space) causes the cue tip to wander laterally off the intended aiming line [15]. Common errors include lateral wrist deviation (measured via gyroscope) and inconsistent impact force (measured via accelerometer).

2.2 State of the art

The domain of "Smart Sports" has evolved significantly with the adoption of Convolutional Neural Networks (CNNs) and Wearable Sensors. [11]

2.2.1 Existing Solutions

- **Hawk-Eye (Computer Vision):** Widely used in snooker and tennis, this system relies on multiple calibrated high-speed cameras to triangulate 3D ball positions. While highly accurate ($<$ 3mm error), it requires a fixed, expensive infrastructure and typically does not analyze the player's biomechanics, only the ball's trajectory. [9]
- **Projection AR Systems:** Solutions like *PoolLiveAid* use overhead projectors to display trajectories directly onto the table cloth. These provide excellent user experience but require expensive hardware and complex calibration procedures. [3]
- **Wearable IMU Analyzers:** Devices such as *Blast Motion* (used in golf or baseball) utilize inertial sensors to track swing metrics. However, these are typically "closed systems" that analyze the swing in isolation. They track the movement but do not interact with the game environment (e.g., they do not know the position of the balls). [16]

2.3 Mathematical Models

The implementation of the system relies on specific mathematical models derived from vector algebra and Newtonian mechanics.

2.3.1 Ghost Ball Vector Math

The core logic of the vision module relies on calculating the intersection of the aiming vector with the target ball's collision zone.

Let P_{cue} and P_{target} be the centers of the cue ball and target ball, and $\vec{v}_{aim}$ be the normalized direction vector of the cue stick. The projection length L_{proj} of the target vector onto the aim vector is calculated using the dot product:

$$L_{proj} = (P_{target} - P_{cue}) \cdot \vec{v}_{aim} \tag{2.1}$$

The closest point on the aiming line to the target ball, P_{close}, is:

$$P_{close} = P_{cue} + \vec{v}_{aim} \cdot L_{proj} \tag{2.2}$$

A valid collision occurs only if the perpendicular distance $d_{\perp}$ is less than the ball diameter D:

$$d_{\perp} = ||P_{target} - P_{close}|| \tag{2.3}$$

If $d_{\perp} < D$, the Ghost Ball position P_{ghost} is found by retreating from P_{close} by an offset calculated via the Pythagorean theorem:

$$P_{ghost} = P_{close} - \vec{v}_{aim} \cdot \sqrt{D^2 - d_{\perp}^2} \tag{2.4}$$

This formula allows the system to draw the "Ghost Ball" circle exactly where the white ball will be at the moment of impact.

2.3.2 Physics of the Stroke (Sensor Fusion)

The sensor module processes raw data to estimate the force of the shot.

Linear Acceleration

The Android 'TYPE_LINEAR_ACCELERATION' virtual sensor is used to isolate the user's movement from Earth's gravity (g). The resulting acceleration vector $\vec{a} = [a_x, a_y, a_z]$ represents the proper acceleration of the cue stick. The magnitude of this acceleration is:

$$||\vec{a}|| = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{2.5}$$

Force Estimation

To provide the player with a metric of "Shot Power" in Newtons, the system applies Newton's Second Law. Assuming a constant estimated mass for the player's forearm and cue ($m_{arm} \approx 4.0$ kg), the impact force F is:

$$F = m_{arm} \cdot ||\vec{a}_{impact}|| \tag{2.6}$$

2.4 Algorithms

Two primary algorithmic domains are utilized to interpret the raw data streams: Computer Vision for game state analysis and Signal Processing for biomechanical analysis.

2.4.1 Computer Vision Algorithms

The vision module utilizes the YOLO (You Only Look Once) architecture, deployed in two distinct configurations to handle different tracking requirements.

Object Detection (Balls)

To identify the game elements, a standard object detection model is employed. It classifies objects into categories such as `cue-ball` and `other` (object balls). The inference function f_{det} transforms the input image I into a set of bounding boxes B:

$$f_{det}(I) \rightarrow \{(x, y, w, h, class, conf)_i\} \tag{2.7}$$

where (x, y) is the center of the ball, used as the input for the "Ghost Ball" geometric calculation.

Pose Estimation (Cue Stick)

Determining the aiming line requires more precision than a bounding box can provide. Therefore, a **Pose Estimation** model (specifically `yolov8-pose`) is used to detect the cue stick. Instead of a box, this model predicts specific semantic keypoints: the `tip` (cue tip) and the `handle` (grip point). The output is a set of coordinates K:

$$f_{pose}(I) \rightarrow \{(x_{tip}, y_{tip}), (x_{handle}, y_{handle})\} \tag{2.8}$$

The aiming vector $\vec{v}_{aim}$ is then derived directly from these keypoints: $\vec{v}_{aim} = (x_{tip} - x_{handle}, y_{tip} - y_{handle})$.

2.4.2 Peak Detection Algorithm

To automatically detect the moment of impact without external triggers, the system implements a real-time peak detection algorithm on the linear acceleration data stream. A hit is registered at time t if the acceleration magnitude $a(t)$ satisfies both a threshold condition and a local maximum condition:

$$a(t) > T_{peak} \quad \wedge \quad a(t) \geq a(t-1) \quad \wedge \quad a(t) \geq a(t+1) \tag{2.9}$$

where T_{peak} is the noise gate threshold (set to $15.0 m/s^2$ in the implementation). This ensures that the system captures the exact moment of maximum force exertion (F_{max}) for the biomechanical analysis.

Chapter 3

Requirements and Tools

This chapter details the functional and non-functional requirements of the ”smart pool assistant” and presents the technological stack selected for implementation. It also describes the system’s use cases and the methodology adopted for the design and development process.

3.1 Functional and Non-functional Requirements

The system requirements were defined to address the problems identified in the previous chapter, specifically the ”ghost ball” visualization and the biomechanical analysis of the stroke.

3.1.1 Functional Requirements

The functional requirements define the specific behaviors and functions the system must support. They are categorized by module:

Vision Module (Desktop):

- **FR-01 Video acquisition:** The system must capture a real-time video stream from a webcam positioned above the billiard table.
- **FR-02 Object detection:** The system must detect and classify the cue ball and object balls using a convolutional neural network (yolo).
- **FR-03 Pose estimation:** The system must identify keypoints of the cue stick (tip and handle) to determine the aiming vector.
- **FR-04 Trajectory prediction:** The system must calculate the ”ghost ball” position and predict the trajectories of the cue ball and target ball based on geometric rules.

- **FR-05 AR visualization:** The system must overlay the predicted paths and the ghost ball indicator onto the video feed in real-time.

Sensor Module (Mobile):

- **FR-06 Data acquisition:** The mobile application must read data from the accelerometer (linear acceleration) and gyroscope at a frequency of at least 50 Hz.
- **FR-07 Data transmission:** The mobile app must transmit sensor data to the desktop server via a TCP/IP socket connection (USB tethering).
- **FR-08 Stroke analysis:** The system must detect the moment of impact (peak acceleration) and calculate the impact force ($f = ma$) and wrist rotation stability.

3.1.2 Non-functional Requirements

The non-functional requirements define the quality attributes of the system:

- **NFR-01 Real-time performance:** The vision processing pipeline must maintain a frame rate of at least 20 fps (frames per second) to provide smooth visual feedback.
- **NFR-02 Latency:** The latency between the physical cue movement and the AR update should be less than 150 ms to ensure the user perceives the lines as responsive.
- **NFR-03 Accuracy:** The "ghost ball" projection error should be less than 5% of the ball diameter to be practically useful for aiming.
- **NFR-04 Sensor synchronization:** The time drift between the video impact detection and sensor peak detection must be handled to correctly associate the physical stroke with the visual event.
- **NFR-05 Usability:** The setup process (camera alignment, phone connection) should be performable by a single user within 5 minutes.

3.2 Use Cases

The interaction between the user (player) and the system is modeled through several key use cases. The primary actor is the player, who interacts with both the physical equipment (phone, cue) and the software (desktop app).

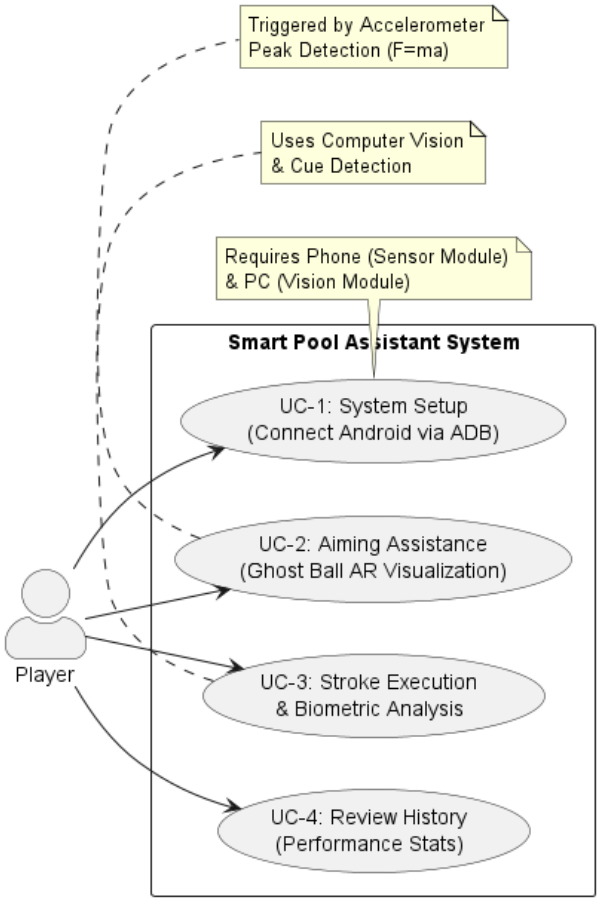


Figure 3.1: Use case diagram

The main use cases are defined as follows:

1. **UC-1 System setup:** The player connects the Android phone to the PC (via USB/ADB), starts the server script, and launches the mobile app. The connection is established if the ”connected” status appears.

2. **UC-2 Aiming assistance:** The player addresses the cue ball. The system detects the cue stick, calculates the trajectory, and projects the ”ghost ball” and aiming lines. The player adjusts their stance based on this visual feedback.

3. **UC-3 Stroke execution & analysis:** The player executes the shot. The system automatically detects the impact via the sensor stream, captures a snapshot of the metrics (force, rotation), and saves the data to a history log.

4. **UC-4 Review history:** The player views the saved ”hit history” (csv/images) to analyze their consistency over the training session.

3.3 Description of Tools

The project leverages a modern technology stack combining computer vision, AI, and mobile development.

3.3.1 Hardware tools

- **Webcam:** A standard HD webcam is used for video input. It is mounted in a "top-down" configuration to minimize perspective distortion.
- **Android smartphone:** A device equipped with an inertial measurement unit (IMU). It serves as the telemetry unit attached to the player's cue.
- **Workstation (PC):** A computer with access to the internet and with installed Python 3.x environment to run the vision server and process data.

3.3.2 Software tools

- **Python 3.x:** The core programming language for the backend server and vision processing. [8]
- **OpenCV (open source computer vision library):** Used for image pre-processing, drawing the AR visualization (lines, circles), and rendering the real-time telemetry graphs. [6]
- **Ultralytics YOLO (You Only Look Once):** A state-of-the-art framework for object detection.
  - *yolov12 (detection):* Used for robust detection of billiard balls under varying lighting conditions. [19]
  - *yolov8-pose (keypoint estimation):* Specifically used to detect the `tip` and `handle` of the cue stick, enabling precise vector calculation. [12]
- **Roboflow:** A platform used for dataset management, image annotation, and versioning. It facilitated the preparation of the training data for the custom billiard model. [10]
- **Android Studio & Java:** The development environment for the mobile sensor application. Java was chosen for its native support of Android sensor APIs. [18]
- **ADB (Android Debug Bridge):** Utilized for establishing a low-latency reverse TCP connection (`adb reverse`) between the Android device and the localhost server. [4]

3.4 Methodology of design and implementation

The development followed an iterative **prototyping methodology** [13], which is suitable for systems involving experimental algorithms (like computer vision) and hardware integration. The process was divided into four phases:

3.4.1 Phase 1: Data collection and model training

The initial phase focused on the vision module. A custom dataset of billiard balls and cue sticks was collected and annotated using Roboflow [10]. The YOLO models were trained iteratively (using Google Colab GPUs) to achieve high mean average precision (mAP) for ball detection and keypoint estimation.

3.4.2 Phase 2: Vision logic implementation

Once the models were trained, the Python logic was developed to translate raw detections into game state information. This involved implementing the vector algebra for the ”ghost ball” [1] algorithm and using OpenCV [6] to render the visual overlays.

3.4.3 Phase 3: Sensor module development

The Android application was developed to reliably extract linear acceleration and gyroscope data. The focus was on implementing the ‘type_linear_acceleration‘ [5] sensor to filter out gravity and establishing a robust TCP socket protocol for data transmission.

Chapter 4

External specification

This chapter provides a detailed description of the system from the user's perspective. It covers the installation requirements, setup procedures, and a walkthrough of typical usage scenarios, including the user interface (UI) and system output.

4.1 Hardware and software requirements

To ensure the correct operation of the "Smart Pool Assistant", the following minimum requirements must be met.

4.1.1 Hardware Requirements

- **Desktop Workstation (Server):**
  - CPU: Intel Core i5 (8th Gen) / AMD Ryzen 5 or better.
  - RAM: Minimum 8 GB (16 GB recommended for smooth video processing).
  - GPU: Dedicated NVIDIA GPU (GTX 1050 or better) recommended for YOLO inference, though CPU-only execution is supported.
  - Ports: At least one USB 3.0 port (the phone connection).
- **Video Input:**
  - HD Webcam (720p minimum resolution) mounted on a tripod or ceiling bracket directly above the pool table.
- **Mobile Device (Client):**
  - Smartphone running Android 9.0 (Pie) or higher.
  - Must contain hardware sensors: Accelerometer and Gyroscope.
  - USB cable for data tethering.

4.1.2 Software Requirements

- **PC Operating System:** Windows 10/11 or Linux.
- **Python Environment:** Python 3.8 or newer.
- **Android Environment:** Android device with "Developer Options" and "USB Debugging" enabled.
- **Dependencies:** The required Python libraries are listed in the `req.txt` file (e.g., `ultralytics`, `opencv-python`, `inference`, `supervision`).

4.2 Installation procedure

The installation process is twofold, involving the desktop server and the mobile client.

4.2.1 Desktop Installation

1. **Clone the Repository:** Download the project source code from the official GitHub repository. You can download the ZIP file directly:

   `https://github.com/ss19190/Smart-9-Ball-Assistant`

   Or via terminal:

   ```
   git clone https://github.com/ss19190/Smart-9-Ball-Assistant
   ```

2. **Install Dependencies:** Open a terminal in the project folder and run:

   ```
   pip install -r requirements.txt
   ```

3. **Install ADB:** Ensure the Android Debug Bridge (ADB) is installed and added to the system PATH variables.

4. **Configure API Keys:** You need a valid API Key to access the trained models.

   - Create a free account at `https://roboflow.com`.
   - Navigate to **Settings** → **API** (or your Workspace Settings).
   - Copy your `Private API Key`.

   Open a `.env` file in the root directory of the project and paste the key:

   ```
   API_KEY=your_copied_key_here
   ```

4.2.2 Mobile Installation

1. **Enable Developer Mode:** On the Android phone, go to **Settings** → **About Phone**. Find the ”Build Number” entry and tap it 7 times repeatedly until you see a message ”You are now a developer!”.

2. **Enable USB Debugging:** Go back to the main Settings menu (or System → Advanced), open **Developer Options**, and toggle the switch for **USB Debugging** to ON. Confirm the security prompt if requested.

3. **Install the App:** Locate the `SensorApp.apk` file in the project directory. You can install it manually by copying it to the phone, or simpler, by running this command in your PC terminal (while the phone is connected):

```
adb install SensorApp.apk
```

   Ensure you allow installation from unknown sources if prompted on the device.

4.3 Activation procedure

To start a training session, the user must follow a specific sequence of steps to establish the client-server connection.

1. **Connect Hardware:** Plug the webcam into the PC and connect the Android phone via USB.

2. **Setup ADB Forwarding:** This step is crucial for the phone to communicate with the PC via localhost. Run the following command in the terminal:

```
adb reverse tcp:5555 tcp:5555
```

3. **Start the Sensor Server:** Run the script responsible for handling telemetry data:

```
python sensors.py
```

4. **Start the Vision Assistant:** Open a second terminal window and run the AR visualization script:

```
python bilard.py
```

5. **Connect the App:** Launch the app on the phone and tap the ”CONNECT” button. The status should change to green ”CONNECTED”.

4.4 Types of users

The system is designed for a single type of user, although the roles can be conceptually divided:

- **The Player (Trainee):** The primary user who wears the sensor and performs the shots. They interact with the physical game and view the feedback on the screen.
- **The Operator (Optional):** A second person (e.g., a coach) who manages the software, starts/stops the scripts, and analyzes the history logs, though the Player can perform these tasks themselves.

4.5 User manual

4.5.1 Main Interface (Desktop)

The desktop window displays the live camera feed overlaid with augmented reality elements.

- **Ghost Ball Indicator:** A white circle appearing on the aiming line, showing where the cue ball will be at impact.
- **Trajectory Lines:**
  - *Gray Line:* The aiming line extending from the cue stick.
  - *Yellow Line:* The predicted path of the cue ball after impact (Tangent Line).
  - *Green Line:* The predicted path of the object ball.

4.5.2 Mobile Interface

The mobile app interface is minimal to avoid distraction.

- **Status Bar:** Shows connection state (Disconnected/Connected).
- **Real-time Values:** Displays raw linear acceleration (X, Y, Z) for debugging purposes.
- **Connect Button:** Toggles the TCP connection.

4.6 System administration

Since this is a standalone prototype, system administration tasks are limited to:

- **Log Management:** The system generates CSV files (`hit_history.csv`) and image snapshots in the `saved_graphs/` folder. The user should periodically archive or delete old files to save disk space.

- **Calibration:** If the "Ghost Ball" accuracy drifts, the user may need to verify the `BALL_DIAMETER_PX` constant in `bilard.py` to match the current camera height.

4.7 Security issues

- **Network Security:** The communication uses unencrypted TCP sockets. This is acceptable for a local USB connection (ADB forwarding), but if ever was implemented and used over public Wi-Fi, the data stream could be intercepted.

- **API Keys:** The Roboflow API key is stored in a `.env` file. This file should not be shared or committed to public version control repositories.

4.8 Example of usage

Scenario: Practicing the Cut Shot The player wants to practice a difficult cut shot into the corner pocket.

1. The player sets up the white ball and the target ball.

2. The system detects the balls and waits for the cue stick.

3. The player addresses the cue ball. The system detects the cue pose and renders the "Ghost Ball" slightly to the left of the target ball.

4. The AR lines show that the current aim will cause the target ball to miss the pocket (hit the rail).

5. The player adjusts their stance until the Green Line points directly into the pocket.

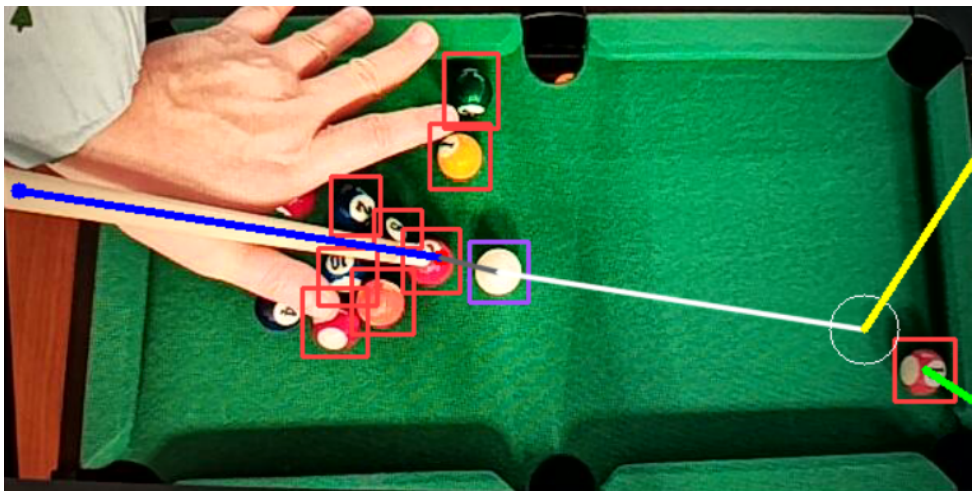
6. The player executes the stroke.

7. The `sensors.py` script detects a peak acceleration of $25 m/s^2$.

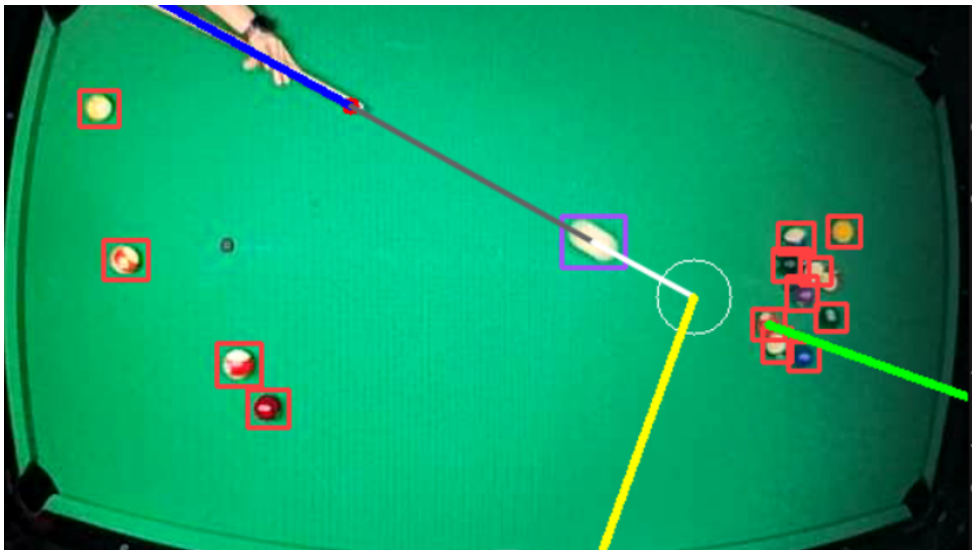
8. The system saves a snapshot of the stroke metrics, allowing the player to see if they maintained wrist stability during the shot.

4.9 Working scenarios

This section illustrates the system in action with screenshots from the prototype.



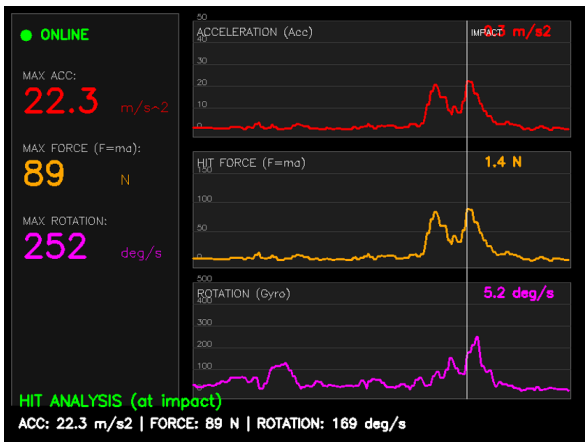
(a) Visualization on a small table.



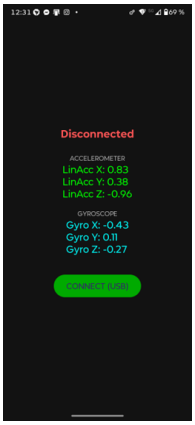
(b) Visualization on a standard pool table.

Figure 4.1: Scenario A: The Vision Assistant interface visualizing a valid collision. The system calculates and displays the predicted outcome in real-time.

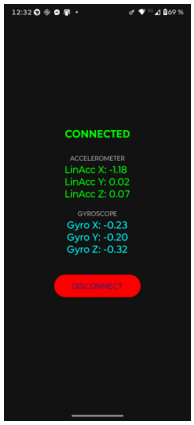
(a) Desktop interface in a state before a stroke.



(b) Desktop interface showing impact analysis showing max force of 89 N.



(c) Mobile app in disconnected state.



(d) Mobile app connected, streaming sensor data.

Figure 4.2: Scenario B: Stroke Analysis workflow. The top row shows the PC application analyzing the stroke data, where (a) is pre-hit and (b) is post-hit showing impact metrics. The bottom row shows the corresponding state of the mobile sensor app (c, d).

Chapter 5

Internal Specification

This chapter presents the technical details of the solution, including the system architecture, key data structures, and a description of the implementation of the most critical modules in Python and Java.

5.1 System Concept

The system concept is based on the fusion of sensory and visual data to assist in playing billiards. The system consists of:

1. **Vision Module (AI):** Analyzes the camera feed, detects billiard balls and the cue, and subsequently calculates the "Ghost Ball"—the predicted position of the cue ball at the moment of impact.

2. **Telemetry Module:** Analyzes the force of the impact and the smoothness of the player's movement based on IMU sensor data (accelerometer, gyroscope) transmitted via USB cable from a smartphone.

5.2 System Architecture

The system operates on a client-server architecture. The smartphone (Client) collects data and transmits it via TCP to the workstation (Server), which processes the camera image in parallel.

The main unit (PC) runs two independent processes (Python scripts):

- **Vision Process (`bilard.py`):** Utilizes the `inference` and `supervision` libraries for object detection and vector calculations.

- **Sensor Server (`sensors.py`):** A multi-threaded TCP server that receives JSON data, parses it, and visualizes graphs using the `OpenCV` library.

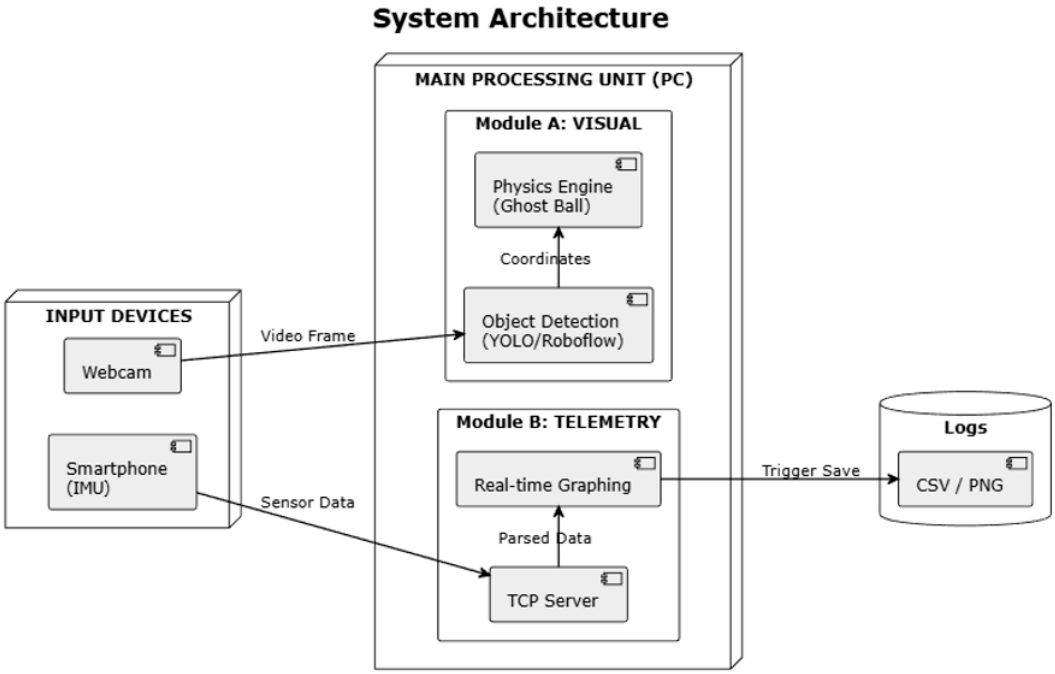


Figure 5.1: System Architecture Diagram and logic flow.

5.3 Description of Data Structures

5.3.1 Network Payload (JSON)

Communication between the Android device and the PC is handled using the JSON format. A key aspect is the transmission of linear acceleration (gravity excluded) and angular velocity.

```
// JSON format generated in MainActivity.java
String json = String.format(Locale.US,
    "{\"acc_x\":␣%.4f,␣\"acc_y\":␣%.4f,␣\"acc_z\":␣%.4f,␣" +
    "\"gyro_x\":␣%.4f,␣\"gyro_y\":␣%.4f,␣\"gyro_z\":␣%.4f}",
    ax, ay, az, gx, gy, gz);
```

Figure 5.2: JSON structure creation in Java (Android).

5.3.2 Physics Vectors (Python)

The vision module operates on vectors from the `NumPy` library to calculate trajectories.

```
# Position representation in bilard.py
cue_ball_center = np.array([x, y], dtype=np.float32)
aim_vector = normalize_vector(tip_pos - handle_pos)
```

5.4 Components and Modules

5.4.1 Mobile Module (Android)

The mobile application was developed in Java. The key component is the `SensorManager`. The `TYPE_LINEAR_ACCELERATION` sensor was utilized instead of the standard accelerometer to eliminate the influence of gravity on the impact force measurement.

```
// MainActivity.java - sensor initialization
linearAcceleration = sensorManager.getDefaultSensor(
    Sensor.TYPE_LINEAR_ACCELERATION
);
```

5.4.2 Vision Module (Computer Vision)

This module utilizes two separate neural network models hosted on the Roboflow platform:

- **Ball Detection:** Model `ball-detection-bzirz/3` for localizing billiard balls.
- **Cue Detection:** Model `cue-detection-ciazj/3` for detecting key points of the cue (tip, handle).

5.5 Overview of Key Algorithms

5.5.1 Ghost Ball Algorithm

The algorithm calculates the "Ghost Ball" position (the point of collision). It works by projecting the aim vector onto the vector connecting the balls.

```
def find_ghost_ball_position(cue_ball_pos, aim_vector, target_ball_pos):
    vec_to_target = target_ball_pos - cue_ball_pos
    projection_length = np.dot(vec_to_target, aim_vector)

    # Calculating the closest point on the shot line
    closest_point = cue_ball_pos + aim_vector * projection_length
    perp_dist = np.linalg.norm(target_ball_pos - closest_point)

    # Correction for ball radius (backward offset)
    if perp_dist < BALL_DIAMETER_PX:
        back_offset = math.sqrt(BALL_DIAMETER_PX**2 - perp_dist**2)
        dist_impact = projection_length - back_offset
        return cue_ball_pos + aim_vector * dist_impact
```

Figure 5.3: Vector-based Ghost Ball calculation implementation.

5.5.2 Impact Force Calculation (Physics)

In the `sensors.py` module, the impact force estimation is implemented based on Newton's Second Law of Motion ($F = ma$). System dynamically calculates the effective mass (m_{arm}) based on anthropometric data derived from Plagenhoef et al. [17].

The effective arm mass is determined by summing the percentages of total body weight for the hand, forearm, and upper arm. According to the tables presented in [17], these values are:

- **Male:** $0.65\% + 1.87\% + 3.25\% = 5.77\%$ of the total body weight.
- **Female:** $0.50\% + 1.57\% + 2.90\% = 4.97\%$ of the total body weight.

The system reads the user's attributes (weight and gender) from a configuration file (`config.json`) to compute the personalized arm mass. The force is calculated as follows:

$$F_{hit} = a_{total} \cdot m_{arm} \tag{5.1}$$

Additionally, the algorithm implements a peak detection mechanism that identifies the second peak in the acceleration signal, distinguishing the actual strike from the backswing phase.

5.6 Implementation Details

5.6.1 Thread Synchronization (Python)

The TCP server in `sensors.py` operates in a separate thread (`daemon=True`) to avoid blocking the interface drawing loop (GUI) in the OpenCV library, if someone would like to run both moduls at the same time.

```
1 # sensors.py - Threading implementation
2 threading.Thread(target=tcp_server_thread, daemon=True).start()
3
4 while True:
5     # Main GUI Loop (OpenCV)
6     cv2.imshow("Hit Analysis", window)
7     if cv2.waitKey(20) & 0xFF == ord('q'): break
```

5.6.2 Network Handling (Android)

In the Android application, a separate thread was employed for network operations to avoid the `NetworkOnMainThreadException`. Data is transmitted at a frequency of approximately 100Hz (every 10ms), ensuring smooth physics representation.

5.7 Applied Design Patterns

- **Listener Pattern (Android):** The implementation of the `SensorEventListener` interface allows for reactive handling of sensor value changes only when they occur. [14]
- **Producer-Consumer:** The TCP thread (Producer) receives data and updates the global `sensor_data` structure, which is subsequently consumed by the main visualization loop.[14]

Chapter 6

Verification and Validation

This chapter describes the testing methodology adopted to ensure the reliability and accuracy of the developed system. It covers the testing paradigm, the scope of test cases, a detailed analysis of encountered software defects, and the final experimental results.

6.1 Testing Paradigm

To ensure a structured approach to verification, the **V-Model** (Verification and Validation Model) was adopted. This model emphasizes the relationship between the design phase and the testing phase. The testing process was divided into three distinct levels:

1. **Unit Testing:** Individual components were tested in isolation. For the Python server, this involved testing the vector calculation functions (e.g., `find_ghost_ball_position`) and the impact force formula ($F = ma$) with known static values to ensure mathematical correctness.

2. **Integration Testing:** This phase focused on the communication between the Android Client and the PC Server. Key tests included verifying the TCP handshake, the serialization of sensor data into JSON format, and the handling of network latency.

3. **System Testing:** The full system was validated in a real-world environment. This involved a player performing actual shots while the system tracked the game state and visualized the analytics in real-time.

6.2 Testing Scope and Test Cases

The testing scope covered both the functional requirements (correct physics calculations, accurate object detection) and non-functional requirements (performance, latency).

Table 6.1 summarizes the key test scenarios.

Table 6.1: Summary of Key Test Cases.

ID	Test Scenario	Expected Result	Status
TC-01	JSON Packet Transmission	Server receives valid JSON structure without corruption.	Passed
TC-02	Ghost Ball Projection	Projector line intersects the target ball center.	Passed
TC-03	High-Velocity Shot Detection	Accelerometer detects peak > 20 m/s^2.	Passed
TC-04	Multi-ball Occlusion	System maintains ID of balls when partially obscured.	Partial

6.3 Detected and Fixed Bugs

During the implementation and testing phases, several critical software defects were identified. Below is a description of the most significant bugs and the solutions applied.

6.3.1 Android UI Thread Blocking (UI Lag)

Problem: The initial version of the Android application exhibited severe interface lag and unresponsiveness during data recording. The accelerometer sensor was configured to `SENSOR_DELAY_FASTEST`, generating events at approximately 100Hz. The application attempted to update the on-screen `TextView` elements inside every sensor callback. Since the UI rendering thread operates at a lower frequency (approx. 60Hz), the event queue became overwhelmed, causing the displayed values to ”trail” behind reality (e.g., the value would continue rising long after the phone had stopped moving).

Solution: The implementation was modified to decouple data collection from UI updates. A throttling mechanism was introduced to update the UI elements only once every 100ms (10Hz), while the data transmission to the server continued at the full 100Hz rate using a background thread.

6.3.2 TCP Stream Fragmentation

Problem: Occasionally, the Python server would crash with a `JSONDecodeError`. This occurred because TCP is a stream-oriented protocol, not a message-oriented one. At high transmission rates, multiple JSON objects were coalesced into a single buffer read, or a single JSON object was split across two reads.

Solution: A delimiter (newline character `\n`) was appended to every message sent from Android. On the server side, a buffer was implemented to accumulate incoming bytes until a full delimiter was found, ensuring that only complete JSON strings were passed to the parser.

6.3.3 Ghost Ball Jitter

Problem: The detected position of the cue ball and cue tip fluctuated slightly (by 1-2 pixels) due to sensor noise in the camera, even when the objects were stationary. This caused the projected "Ghost Ball" vector to shake rapidly, making it difficult for the user to aim.

Solution: A moving average filter (size $N = 5$) was applied to the coordinates of the detected objects. This smoothed out the high-frequency noise while maintaining acceptable responsiveness to intentional movements.

6.4 Experimental Results

To validate the practical utility of the system, an experiment was conducted to verify the consistency and distinctiveness of the impact force estimation algorithm. Since measuring the absolute force in Newtons without specialized laboratory equipment (e.g., force plates) is difficult, the experiment focused on the **relative consistency** of the measurements.

6.4.1 Force Estimation Consistency

The test procedure involved performing three series of 10 shots each (30 shots total), categorized by subjective intensity:

1. **Soft shots:** Gentle taps, intended for precise positioning.
2. **Medium shots:** Standard shots used during regular gameplay.
3. **Hard shots:** Break-type shots or power shots.

The data was logged automatically to a CSV file by the system. Table 6.2 presents the raw force values recorded for each attempt.

Based on the raw data, a statistical analysis was performed to calculate the Mean Force (F_{avg}) and Standard Deviation (σ) for each category. The results are summarized in Table 6.3.

Interpretation: The results demonstrate a clear separation between the categories. As shown in the tables, the highest recorded value for a "Soft" shot (62.4 N) is significantly lower than the lowest value for a "Medium" shot (112.7 N). This lack of overlap confirms

Table 6.2: Raw recorded force values (in Newtons) for 30 experimental shots.

Attempt #	**Soft (N)**	**Medium (N)**	**Hard (N)**
1	48.5	115.2	205.4
2	52.1	122.8	218.1
3	45.0	130.5	230.2
4	60.3	118.4	198.7
5	55.8	140.1	240.5
6	62.4	125.6	222.9
7	58.1	135.2	210.3
8	49.9	128.9	225.8
9	53.2	112.7	218.0
10	50.6	138.3	235.1

Table 6.3: Statistical analysis of impact force measurements ($N = 30$).

Shot Category	**Min (N)**	**Max (N)**	**Mean ± Std Dev (N)**
Soft Shot	45.0	62.4	**53.6 ± 5.4**
Medium Shot	112.7	140.1	**126.8 ± 9.1**
Hard Shot	198.7	240.5	**220.5 ± 13.2**

that the system effectively distinguishes between different levels of player intent using the accelerometer-based algorithm ($F = ma$).

The standard deviation increases with the force of the shot (from ±5.4N to ±13.2N), which is consistent with the natural biomechanical variability when performing high-velocity movements.

6.4.2 System Latency and Performance

A secondary test was conducted to evaluate the real-time capabilities of the system. The total latency was measured as the time difference between the sensor event timestamp (on Android) and the visualization update on the PC.

- **Network Latency:** Average 15ms (local Wi-Fi network).
- **Processing Time:** Average 25ms (Vision Module inference per frame).
- **Frame Rate:** The vision module maintained a stable 30 FPS on the test workstation.

These performance metrics confirm that the system acts in "near real-time," providing feedback instant enough for the user to relate the data to the just-performed action.

Chapter 7

Conclusions

- achieved results with regard to objectives of the thesis and requirements
- path of further development (eg functional extension …)
- encountered difficulties and problems

Bibliography

[1] David Alciatore. *Ghost-ball Aiming*. 2025. URL: https://drdavepoolinfo.com/resource_files/ghost_ball_aiming.pdf (visited on 14/02/2025).

[2] David Alciatore. *Sidespin and English Terminology and Uses*. 2012. URL: https://drdavepoolinfo.com/faq/sidespin/terminology-and-uses/ (visited on 14/02/2025).

[3] Ricardo Alves, Luís Sousa and Joao Rodrigues. 'PoolLiveAid: Augmented reality pool table to assist inexperienced players'. In: *21st International Conference on Computer Graphics, Visualization and Computer Vision*. 2013, pp. 184 –193.

[4] Android Developers. *Android Debug Bridge (adb)*. Official Documentation. Google. 2025. URL: https://developer.android.com/tools/adb (visited on 19/02/2025).

[5] Android Developers. *Motion sensors*. Official Documentation. Google. 2025. URL: https://developer.android.com/develop/sensors-and-location/sensors/sensors_motion (visited on 19/02/2025).

[6] G. Bradski. 'The OpenCV Library'. In: *Dr. Dobb's Journal of Software Tools* (2000).

[7] Cheng Chen, Jiaxin Xue, Wenling Gou, Mengning Xie and Xiaolin Yao. 'Quantitative analysis and evaluation of research on the application of computer vision in sports since the 21st century'. In: *frontiers* (2025), pp. 1 –8.

[8] Abhinav Dadhich. *Practical Computer Vision: Extract insightful information from images using TensorFlow, Keras, and OpenCV*. Packt Publishing, 2018. ISBN: 978-17-882-9476-8.

[9] Manish Duggal. 'Hawk Eye Technology'. In: *Journal of Global Research Computer Science & Technology* 1.2 (2024), pp. 1 –7.

[10] Brad Dwyer and James Gallagher. *Getting Started with Roboflow*. 2023. URL: https://blog.roboflow.com/getting-started-with-roboflow/ (visited on 16/03/2023).

[11] Bernhard Hollaus, Sebastian Stabinger, Andreas Mehrle and Christian Raschner. 'Using Wearable Sensors and a Convolutional Neural Network for Catch Detection in American Football'. In: *Sensors* 20.23 (2020), p. 6722.

[12] Glenn Jocher, Ayush Chaurasia and Jing Qiu. *Ultralytics YOLOv8*. Version 8.0.0. 2023. URL: https://github.com/ultralytics/ultralytics.

[13] Lumitex. *Prototyping Methodology Steps on How to Use It Correctly*. 2017. URL: https://www.lumitex.com/blog/prototyping-methodology (visited on 01/06/2017).

[14] Zigurd Mednieks, Laird Dornin, G. Blake Meike and Masumi Nakamura. *Programming Android, 2nd Edition: Java Programming for the New Generation of Mobile Devices*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2012. ISBN: 978-14-493-5847-1.

[15] A. D. Moore. 'Mechanics of Billiards, and Analysis of Willie Hoppe's Stroke'. In: University of Michigan, College of Engineering, 1947.

[16] Sean Ogle. *Breaking Eighty*. 2022. URL: https://breakingeighty.com/blast-motion-golf-review (visited on 13/10/2022).

[17] Stanley Plagenhoef, F. Gaynor Evans and Thomas Abdelnour. 'Anatomical Data for Analyzing Human Motion'. In: *RESEARCH QUARTERLY FOR EXERCISE AND SPORT* 54.2 (1983), pp. 169 –178.

[18] Neil Smyth. *Android Studio 3.3 Development Essentials - Android 9 Edition: Developing Android 9 Apps Using Android Studio 3.3, Java and Android Jetpack*. Payload Media, Inc., 2019. ISBN: 978-1795654760.

[19] Yunjie Tian, Qixiang Ye and David Doermann. *YOLO12: Attention-Centric Real-Time Object Detectors*. 2025. URL: https://github.com/sunsmarterjie/yolov12.

Appendices

Index of abbreviations and symbols

DNA	deoxyribonucleic acid
MVC	model–view–controller
N	cardinality of data set
μ	membership function of a fuzzy set
$\mathbb{E}$	set of edges of a graph
$\mathcal{L}$	Laplace transformation

Listings

(Put long listings here.)

```
1 if (_nClusters < 1)
2     throw std::string ("unknown number of clusters");
3 if (_nIterations < 1 and _epsilon < 0)
4     throw std::string ("You should set a maximal number of
          iteration or minimal difference — epsilon.");
5 if (_nIterations > 0 and _epsilon > 0)
6     throw std::string ("Both number of iterations and minimal
          epsilon set — you should set either number of iterations
          or minimal epsilon.");
```

List of additional files in electronic submission (if applicable)

Additional files uploaded to the system include:

- source code of the application,
- test data,
- a video file showing how software or hardware developed for thesis is used,
- etc.

List of Figures

List of Tables