Development Engineering Project CP-301
Fire Extinguisher Modelling

Mid-Semester Report
Department of Mechanical Engineering, IIT Ropar

Supervised by:
Dr. Chandrakant K. Nirala

Submitted by:
Aditi - 2022MEB1288
Aditi Garg - 2022MEB1289
Arnav Maitreya - 2022MEB1299
Vaibhav Mehta - 2022MEB1360

Submitted on: 8th May, 2025

Content

Chapter 1: Introduction

1.1 Background and Motivation

Fire hazards pose a significant threat to life, infrastructure, and the environment, necessitating rapid and effective suppression strategies. Conventional fire extinguishing systems, such as sprinklers, drones, and manually operated extinguishers, often face limitations in terms of reach, efficiency, and deployment speed. For instance, sprinkler systems are fixed in place and may not effectively suppress fires in outdoor or high-ceiling environments, while drone-based systems require complex navigation and may be hindered by adverse conditions such as smoke and high temperatures. Manual fire extinguishing methods, though effective in small-scale fires, are slow and require human presence, posing safety risks in extreme situations. In industrial settings, warehouses, or hazardous environments where fires spread quickly, there is a growing need for an automated, precise, and rapid-response suppression system that can effectively reach and neutralize fire sources with minimal delay.

To address these challenges, we propose a novel fire extinguishing system that leverages optimized projectile motion to deliver a nitrogen-filled ball to the fire location. This integrated system offers a potential solution for rapid and targeted fire suppression in critical environments where conventional methods may fall short.

1.2 Problem Description

Fire suppression in hazardous environments requires rapid response and precise targeting to minimize damage and prevent escalation. Conventional fire suppression systems often rely on fixed sprinklers or manually operated extinguishers, which may be inefficient in dynamic or remote fire incidents. This research proposes an automated fire suppression mechanism utilizing a spring-actuated projectile system to deliver nitrogen-based fire suppressant balls to detected fire locations.
The system consists of an integrated fire detection module, a computational control unit, and a spring-mass launch mechanism. Upon fire detection, infrared/thermal sensors pinpoint the exact coordinates of the fire and relay the data to the central control system. The control unit then employs kinematic and dynamic modeling of projectile motion to determine the optimal launch parameters, including initial velocity, launch angle, and required spring compression. The objective is to ensure that the nitrogen ball follows a precise trajectory to reach the fire location efficiently.

This study focuses on the optimization of launch parameters. Additionally, the research explores control strategies for dynamic trajectory adjustments to enhance accuracy. The findings from this study aim to contribute to the development of autonomous, targeted fire suppression technologies suitable for industrial, military, and remote firefighting applications.

1.3 Objectives

1. Develop a mathematical model for projectile motion under ideal conditions to determine optimal launch velocity, angle, azimuth, and time of flight, using optimization algorithms for precise targeting.
2. Design a spring-based launching mechanism and calculate the initial static displacement required to achieve the optimized velocity while ensuring the mechanical feasibility of the system.
3. Implement a control system to automate fire detection, targeting, and launching, ensuring precision, consistency, and rapid response.
4. Select suitable materials for nitrogen-filled balls to ensure safe deployment and effective fire suppression while considering factors like durability and impact absorption.
5. Validate the optimized launch parameters by comparing them with standard projectile motion equations, conduct numerical simulations, and explore potential integration with automated fire detection systems for enhanced efficiency.

1.4 Mid-Sem Summary & Transition to Software Development

By mid-semester, our project successfully modeled the optimized parabolic motion of a nitrogen ball propelled by a spring towards a specified target point. We developed a simulation that generated 3D plots of the projectile trajectory and calculated the necessary spring deflection to achieve the desired launch velocity. Additionally, we created a website interface where users could input target coordinates (x, y) and visualize the resulting projectile path.

Post mid-semester, we shifted focus towards enhancing functionality and user experience. The code was optimized to handle negative x and y inputs, expanding the system's operational range. We introduced a chessboard-like interactive floor interface representing the room's grip pattern; clicking any square automatically generated projectile plots along with detailed calculations for initial velocity, time of flight, and launch angle. A 2D plot was also integrated for a clearer side-view of the projectile. Parallelly, we extended the project scope by developing a Convolutional Neural Network (CNN) model capable of real-time fire detection, adding an intelligent safety layer to the system. This marked our transition from basic physical modeling to a more comprehensive software development phase, integrating simulation, UI/UX, and AI-driven detection.

Chapter 2 : Existing Studies and Literature Review

1.1 Optimizing the Launch of a Projectile to Hit a Target[1]

1.1.1 Summary

Projectile motion has been widely studied in physics, particularly in optimizing launch conditions to achieve precise targeting. A significant contribution to this field is Carl E. Mungan's study, *Optimizing the Launch of a Projectile to Hit a Target* (2017), which explores the theoretical and experimental aspects of determining the minimum required launch speed and optimal angle to hit a target at given coordinates. This work is particularly relevant for applications in sports, military ballistics, and physics education.

1.1.2 Mathematical Formulation

Mungan's study considers a projectile launched with speed and angle to hit a target located at rectangular coordinates from the launch point **(refer to Fig 1 for the projectile motion trajectory)**. The governing kinematic equations for projectile motion, assuming negligible air resistance, are:

$$t = \frac{x}{v\cos\theta} \qquad y = (v\sin\theta)t - \frac{1}{2}gt^2$$

where g = 9.8 m/s^2 is Earth's surface free-fall acceleration. Using these both equations and rearranging them leads to

$$v = \sec\theta\sqrt{\frac{gx/2}{\tan\theta - \tan\phi'}}$$

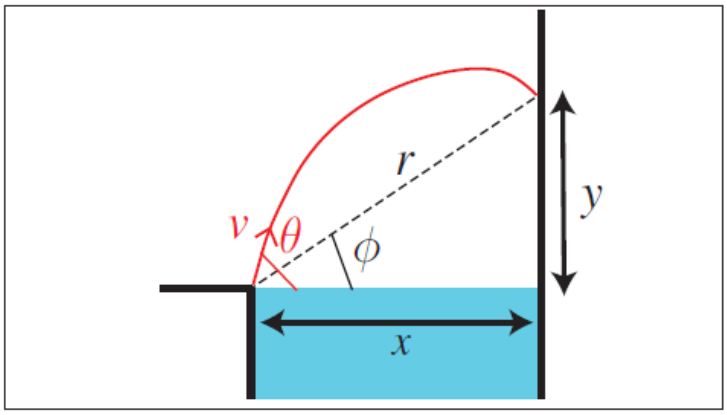

Fig. 1. Trajectory of a stone launched with optimal speed v and angle such that it passes through a hole located at rectangular coordinates (x,y) relative to the launch point. In polar coordinates the hole is located at (r, ϕ) [1]

where tan = y/x **(from Fig. 1)**. Taking the derivative of the above equation with respect to the launch angle and setting it equal to zero to minimize the launch speed. The result then simplifies to the quadratic equation

$$tan^2\theta - 2tan\phi tan\theta - 1 = 0$$

whose positive root is

$$tan\theta = tan\phi + sec\phi$$

This equation has the simple solution

$$\theta = 45^\circ + \frac{\phi}{2}$$

Substituting the positive root of the quadratic equation (i.e derivative of velocity equation) again into the velocity equation, we get the minimum launch speed

$$v = \sqrt{g(y + r)}$$

where $r = (x^2 + y^2)^{1/2}$ is the distance from the launch point to the final destination, **as shown in Fig. 1**.

These equations have the expected limiting forms for y=0 (namely $\Theta = 45^o$ and $\nu = \sqrt{gx}$, which are familiar formulas for maximum range) and for $x = 0$ (namely $\Theta = 90^o$ and $\nu = \sqrt{2gy}$), corresponding to a vertical throw).

Intuitively, minimizing ν with respect to Θ for fixed x and y should be equivalent to maximizing x with respect to Θ for fixed y and v. To prove it, solve $\nu = \sqrt{g(y + r)}$ for r, square the result and equate it to $x^2 + y^2$, and then isolate x to get

$$x = \frac{v^2}{g}\sqrt{1 - \frac{2gy}{v^2}}$$

This formula indeed gives the maximum range of a projectile launched from height h = –y relative to the final level. It can alternatively be written in the compact form $x = v^2_{ave}/g$, where v_{avg} is the geometric average of the initial speed v and the final speed $(v^2 - 2gy)^{1/2}$ from energy conservation.

1.2 Literature Study on Portable Water Mist Fire Extinguishers[2]

1.2.1 Introduction

Fire extinguishers play a critical role in safeguarding life and property by controlling fire outbreaks. Conventional extinguishers—such as water, powder, foam, and carbon dioxide types—have been widely used but come with limitations in terms of environmental impact, safety, and their ability to handle different fire classes. This has led to increased interest in advanced fire suppression methods, notably water mist technology.

1.2.2 Working Principle of Water Mist Extinguishers and Experimental Setup

Water mist extinguishers function by dispersing ultra-fine water droplets that cool the flame and surrounding surfaces, thereby interrupting the combustion process. The mist provides a high surface area for heat absorption, improving efficiency. Depending on the fire type, the suppression mechanisms differ:

- **Flammable Liquid Fires**: Primarily cooled at the flame zone, lowering combustion temperature.
- **Cooking Oil Fires**: Rapid vaporization occurs, potentially intensifying the fire momentarily before achieving suppression.
- **Wood Crib Fires**: Cooling of the fuel surface reduces pyrolysis and flame propagation.
- **Electrical Fires (Class C)**: Proper discharge distances ensure leakage currents remain below hazardous levels, confirming safe operation.

The referenced study tested two portable water mist extinguisher prototypes, each featuring distinct nozzle designs and spray characteristics. These were evaluated across multiple fire types to assess performance, efficiency, and safety.

1.2.3 Key Findings and Conclusion

- **Flammable Liquid Fires**: One prototype extinguished fires within 2.5 seconds using just 0.5–0.6 L of water, while the second failed, highlighting the importance of optimized nozzle design.
- **Cooking Oil Fires**: Extinguishment times reached up to 19 seconds due to the high thermal mass of the oil.
- **Wood Crib Fires**: Uniform, targeted spraying was essential; random application was ineffective.
- **Electrical Safety**: Low leakage currents were recorded, validating the safe use of water mist near electrical equipment when proper precautions were observed.

Water mist fire extinguishers present a versatile, environmentally friendly alternative to conventional systems. Their effectiveness depends on several design parameters, including droplet size, spray distribution, and discharge pressure. Continued development in nozzle engineering and performance optimization is necessary to enhance their practicality for widespread use across various fire scenarios.

1.3 Use of Fire-Extinguishing Balls for a Conceptual System[3]

1.3.1 Introduction

Advancements in unmanned aerial systems (UAS) and novel fire suppression technologies have created new opportunities in wildfire management. A recent transdisciplinary research initiative explores the use of drone networks integrated with fire-extinguishing balls as a rapid-response method for localized fire control. This approach seeks to overcome the limitations of traditional suppression methods, particularly in hard-to-access wildland-urban interfaces.

1.3.2 System Overview and Deployment Strategy

The proposed system comprises three drone classes: scouting drones for fire detection, communication drones for data relay, and fire-fighting drones equipped to deploy extinguishing balls. These balls, ranging from 0.5 to 1.3 kg, have shown effectiveness in extinguishing short grass fires, with suppression radii between 1 to 2.6 meters per ball **(Fig2)**. However, their effectiveness in building fires was limited, especially in closed-structure environments. Early prototypes using 15 kg-payload drones demonstrated potential, though challenges in autonomous coordination and payload-range optimization remain.

Fig. 2. Nitrogen gas based fire extinguishing balls [3]

1.3.3 Advantages and Limitations

The drone-deployed fire-extinguishing balls release nitrogen gas upon impact, displacing oxygen and lowering concentrations below the combustion threshold (~15%), effectively suppressing fires. Key advantages include:

- **Non-toxic and residue-free**: Ideal for sensitive environments like data centers.
- **Rapid activation**: Suppression occurs within 3–5 seconds.
- **Minimal collateral damage**: Unlike water or foam, it doesn't harm assets or infrastructure.

However, several challenges limit the technology's application:

- **Payload vs. Range Tradeoffs**: Larger payloads reduce drone range, requiring hybrid propulsion for larger payloads.
- **Environmental Interference**: High winds and smoke reduce sensor accuracy; LIDAR integration is planned.
- **Regulatory Constraints**: Autonomous drone operations require FAA waivers, and legal frameworks for emergency firefighting are under development.

Despite these limitations, the technology holds significant potential for rapid, localized fire suppression in wildland-urban interfaces.

1.3.6 Conclusion

The integration of drone networks with fire-extinguishing balls represents a transformative step in wildfire suppression. While not yet viable for enclosed structure fires, the approach shows strong potential in open vegetation fire scenarios. Continued work on drone autonomy, payload scalability, and legal frameworks is essential for real-world deployment.

1.4 An Experiment on Projectile Motion[4]

This paper presents a practical, hands-on approach to teaching projectile motion physics to high school students through a low-cost experimental setup. Developed by researchers from the Homi Bhabha Centre for Science Education and Mainadevi Bajaj International School in Mumbai, the experiment formalizes students' everyday experiences with projectile motion using simple tools and rigorous scientific methods.

1.4.1 Summary

The research paper describes an experimental setup designed to understand projectile motion concepts using affordable materials. The apparatus consists of a launcher assembled using plastic vials, a spring with known spring constant, a syringe piston that serves as a release mechanism, a protractor for angle measurement, and a steel ball bearing (weighing approximately 1.7g) as the projectile. The setup includes a measuring tape to determine the range and a sand-filled tray to capture the ball's landing position without bouncing.

The launcher is designed so that the spring can be compressed to a consistent extent, locked with a pin, and then released to propel the steel ball at various angles. This carefully constructed apparatus allows students to observe how the angle of projection affects the range of a projectile while controlling other variables in the system.

1.4.2 Theory

The paper establishes the theoretical framework of projectile motion, explaining that when an object is projected under gravitational force with initial velocity "v" (muzzle velocity) at various angles, its trajectory forms a parabola when air resistance is ignore. The paper elaborates on the spring mechanics, noting that a restoring force proportional to compression/elongation (x) is set up within the spring. When released, this force decreases linearly until the compression/elongation reaches zero, with the average force being $\frac{1}{2}kx^2$

The relationship between the velocity of projection, spring properties, and angle is given by:

$$v^2 = \frac{2k(\frac{x}{2}^2)}{m} - 2\frac{x}{2}gsin\theta$$

This equation accounts for both the spring force and earth's gravity acting on the ball during launch.

1.4.3 Conclusion

The primary contribution of this paper is a well-designed educational experiment that illustrates important physics concepts using affordable materials. The experiment allows students to empirically verify theoretical projectile motion formulas while developing laboratory skills. The apparatus design demonstrates how sophisticated physics concepts can be taught with simple tools, making advanced scientific principles accessible

The educational experiment provides a strong foundation for our project, where we propose a spring-based mechanism to impart controlled projectile motion to a ball from a specific height. By calibrating spring force and launch angle, we can study and verify projectile trajectories while replicating key aspects of data collection, graphical analysis, and real-world validation, alongside introducing mechanical design and energy transfer concepts.

1.5 A Fire Detection Algorithm Using Convolutional Neural Network[5]

1.5.1 Summary

This research paper presents a novel approach for fire detection using deep learning and image processing techniques. Developed by Ahmed A. Alsheikhy from Northern Border University in Saudi Arabia, the algorithm aims to detect fires at an early stage to minimize loss of life and property damage.

The proposed method combines image processing with a Convolutional Neural Network (CNN) tool called AlexNet to create a fast and accurate fire detection system. The algorithm filters images based on color thresholds, flame texture, and reflection characteristics to identify both fire and smoke. Using MATLAB as the simulation environment, the author conducted extensive experiments with a dataset from Kaggle to demonstrate the algorithm's effectiveness. The results show an impressive accuracy rate exceeding 97%, outperforming previous methods in the literature.

1.5.2 Theory

The theoretical foundation of this research rests on several interconnected technologies:

Convolutional Neural Networks (CNN)

The paper explains that CNN is a deep learning approach that manages information with features represented in 2D or 3D models. These networks take input through images (color or grayscale) and consist of multiple layers that form kernels to extract required features. As shown in **(Figure 3)**, a typical CNN structure includes various layers that progressively process the image data.

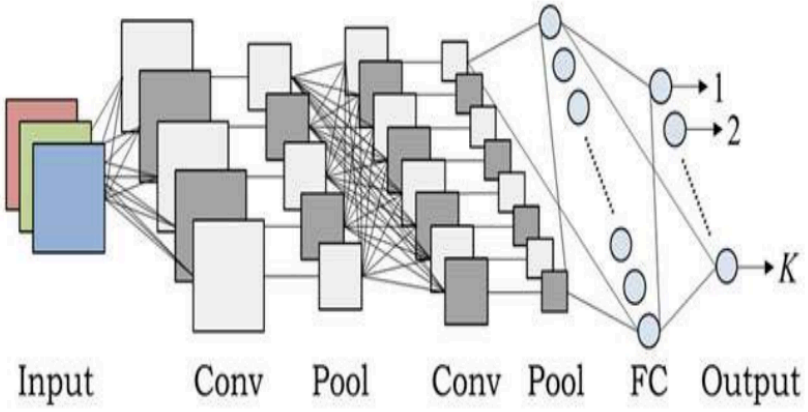

Fig 3 A typical CNN structure includes various layers .

AlexNet Architecture

The algorithm specifically employs AlexNet, which was developed in 2012. AlexNet contains eight layers-five convolutional layers (designated as conv1-5) and three fully connected layers (FC6-8). This architecture provides the deep learning capabilities essential for accurate fire detection.

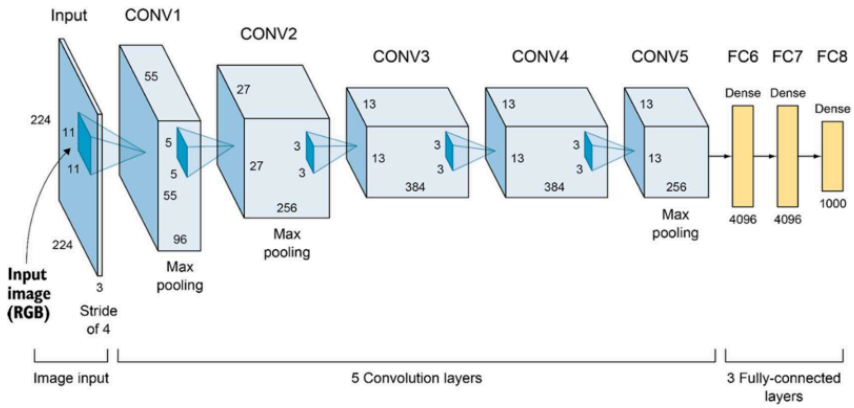

Fig 4 AlexNet Architecture

Image Processing Techniques

The paper details several image processing methods used in the algorithm:

1. Color Space Transformations: The algorithm starts by detecting Red, Green, and Blue colors, then converts images to grayscale for further processing.
2. HSV Conversion: Images are transformed into HSV (Hue, Saturation, Value) to better identify illumination and intensity attributes.
3. Edge Detection: Image segmentation uses the Gradient technique with three operators: Sobel, Prewitt, and Robert to find discontinuities in regions and observe significant changes in gray color.
4. YCrCb Analysis: The algorithm computes YCrCb values, where Y represents brightness (which human eyes are particularly sensitive to), while Cr and Cb represent red and blue components relative to green.

1.5.3 Readings and Finding

The experimental verification of the algorithm was conducted using MATLAB's Image Processing Toolbox. The dataset, downloaded from Kaggle (originally uploaded for NASA Space Applications Challenge in 2018), contained 999 images, with 244 being fire-free.

Dataset and Testing

For thorough evaluation, the author expanded testing to 1,400 images divided into three parts:

- 70% (980 images) for training
- 15% (220 images) for testing
- The remaining percentage for validation

Performance Metrics

The algorithm's performance was assessed using several metrics (as shown in confusion matrix, **fig 5**)

- True Positive (TP): 186 (correctly identified fire images)
- False Positive (FP): 3 (incorrectly identified as fire)
- True Negative (TN): 29 (correctly identified non-fire images)
- False Negative (FN): 2 (fires missed by the algorithm)
- Precision: 98.41%
- Recall: 98.94%
- Accuracy: 97.73%

Fig 5 Confusion Matrix

1.5.4 Conclusion

The author concludes that the proposed fire detection algorithm provides excellent support for early fire detection with an accuracy exceeding 97%. The research successfully demonstrates that through a combination of image processing and deep learning techniques, fires can be detected at various scales-from small initial flames to large conflagrations-and even smoke can be identified as an early warning sign.

The algorithm's high precision (98.41%) and recall (98.94%) rates indicate its reliability in real-world applications. These metrics significantly surpass those of previous methods in the literature, positioning this approach as a potentially valuable tool for fire safety systems.

The author acknowledges that computation time remains a challenge and suggests that future work should focus on minimizing this aspect. Additionally, the paper recommends testing the algorithm under severe and extreme environmental conditions such as dusty weather, fog, and high winds to further validate its robustness in diverse scenarios.

By enabling early detection of both fire and smoke, this algorithm has the potential to save lives, reduce injuries, and minimize property damage, addressing a critical need in fire safety technology.

Chapter 3: Methodology

3.1 Coordinate/Grid Definition

To ensure precise fire suppression, we first establish a coordinate system by mapping the floor of the target room into a grid structure, with the center as the origin (0,0). This approach allows for an organized and systematic method to identify the fire's location accurately. The room will be divided into equal grid sections, and each section will have unique (x, y) coordinates. This setup provides a reference frame for fire detection and projectile calculations.

Once the grid is established, we will assign coordinates to known objects and boundaries within the room to prevent collisions and interference during projectile launch. The mapping will also consider potential obstructions, ensuring that the fire suppression system can determine an optimal launch path. The grid-based coordinate system acts as the foundation for fire detection and projectile motion optimization, enabling accurate targeting and launch parameter calculations.

3.2 Sensing Temperature and Camera-Based Fire Detection

Fire detection is a crucial component of the system. We employ a temperature sensor and a camera-based detection system to identify the exact location of the fire within the predefined grid. The temperature sensor continuously monitors thermal changes in the environment and helps confirm the presence of fire. The camera system, powered by a Convolutional Neural Network (CNN) model, is used to recognize fire patterns and shapes, distinguishing actual fire from false alarms such as reflections or bright lights.

The CNN model is trained using a dataset of fire images to detect and classify flames effectively. Once the fire is detected, the system extracts its exact coordinates using image processing techniques and maps them onto the established grid. These coordinates are then fed into the optimization algorithm to determine the precise projectile parameters required for fire suppression. This dual detection mechanism enhances accuracy and reliability, ensuring that the fire suppression system correctly identifies and responds to fire incidents.

3.3 Velocity and Angle Optimization + Deflection of Spring Required

Once the fire's coordinates are identified, the next step is to determine the optimal launch parameters for the nitrogen ball to reach the fire. Our optimization algorithm calculates the velocity, launch angle, and azimuth required for accurate targeting. Initially, we considered time-based optimization, but it led to imaginary roots due to the complex nature of the equations, making the computation inefficient. Instead, we focus on optimizing velocity and angle, which simplifies calculations while ensuring accurate projectile motion.

The projectile motion equations are solved to find the ideal initial velocity and launch angle required for the nitrogen ball to hit the fire location. These optimized parameters are then used to determine the deflection required in the spring to achieve the computed launch velocity.

Since the system employs a spring-based launch mechanism, Hooke's Law is used to calculate the initial displacement of the spring:

$$F=kx$$

where F is the required force, k is the stiffness of the spring, and x is the displacement. By correlating the optimized velocity with spring deflection, we ensure that the nitrogen ball is launched with the precise force needed to reach the fire.

3.4 Control and Feedback of Spring-Mass System

A control system is integrated into the spring-mass launching mechanism to ensure the system achieves the required spring deflection accurately. The control system continuously monitors the spring's displacement and compares it to the optimized value. Sensors are used to measure the actual compression of the spring, and feedback is provided to adjust the system if deviations are detected.

The control system operates as a closed-loop feedback mechanism, ensuring that the spring is compressed to the exact displacement required before launch. Once the correct launch velocity and angles are confirmed, the system initiates the firing mechanism, launching the nitrogen ball toward the fire. This feedback-based control enhances the accuracy and reliability of the fire suppression system, ensuring that each launch is precisely executed according to the computed parameters.

(A flow chart of the proposed methodology has been attached for reference in Fig 6)

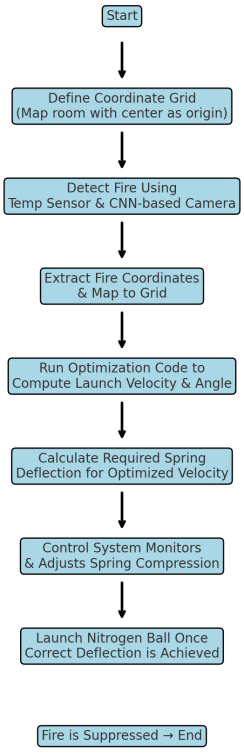

Fig. 6. Methodology

Chapter 4: Work Progress

4.1 Optimization Code

4.1.1 Initialization and Input Handling

The code begins by importing essential scientific computing libraries: NumPy for array operations, SciPy's optimization module for trajectory calculation, Plotly for 3D visualization, and the math module for fundamental mathematical functions. Three physical constants are defined:

- Gravitational acceleration: g = 9.8 m/s²
- Initial launch height: h_0 = 10 m
- Target elevation: z□$_{aaox}$ = 0

User input is collected through input() calls to specify the target coordinates in the horizontal plane , thereby establishing a 3D targeting system. This configuration facilitates parametric analysis of projectile trajectories under varying target specifications.

4.1.2 Objective Function for Trajectory Optimization

The core optimization routine employs a custom objective function that minimizes the squared error between calculated and target positions. It takes three parameters:

- Initial velocity: v_0
- Elevation angle: θ
- Azimuth angle: ϕ

Computational Steps:

1. Angle Conversion: Converts angles from degrees to radians for trigonometric calculations.
2. Vertical Motion Calculation: The equation of motion in the vertical direction is:

$$z(t) = h_0 + v_0 sin\theta \cdot t - \frac{1}{2} g t^2$$

   This is solved using the quadratic formula.
3. Horizontal Displacements:

$$v_0 cos\theta cos\phi \cdot ty(t) = v_0 cos\theta sin\phi \cdot t$$

4. Error Calculation: The function returns the position error:

$$(x - x_{target})^2 + (y - y_{target})^2$$

5. Discriminant Check: Prevents imaginary solutions for time by imposing a large penalty for non-physical trajectories.

4.1.3 Spring-Mass Displacement Calculation

A secondary physics model is implemented in the calculate_displacement function to compute spring compression using energy conservation principles. Given:

- Mass: m = 1.5 kg
- Spring constant: k = 2000 N/m

The displacement is derived from the equivalence of kinetic and elastic potential energy:

$$\frac{1}{2}mv^2 = \frac{1}{2}kx^2$$

Solving for , we obtain:

$$x = \sqrt{\frac{mv^2}{k}}$$

This formulation enables an integrated analysis of projectile launch dynamics and mechanical system performance.

4.1.4 Optimization Setup and Execution

The system is constrained by practical operational limits:

- Velocity Bound: $0 \leq v_0 \leq 15$ (Preventing unrealistic speeds)
- Elevation Angle Constraint: $-90^\circ \leq \theta \leq 0^\circ$ (Excluding downward launches)
- Azimuth Angle Range: (Full rotational freedom)

Using scipy.optimize.minimize, an initial guess of: is supplied. The algorithm employs gradient-based optimization to determine the optimal launch parameters, circumventing the complexities of analytical solutions for multivariable constrained problems.

4.1.5 Result Extraction and Validation

Following optimization, the solution undergoes physical validation:

- Time-of-flight is recomputed using the optimized parameters.
- Negative time roots are filtered using: $t_{final} = max(t_1, t_2)$
- Results are formatted and displayed, including the spring displacement.

This validation process ensures that the obtained solutions are physically meaningful and adhere to the principles of projectile motion.

4.1.6 Trajectory Visualization

The visualization module operates as follows:

1. Generates 500 time samples up to t_{final}
2. Computes 3D coordinates using the motion equations.
3. Filters data points maintaining $z \geq 0$ (ensuring all points are above ground level).
4. Uses Plotly's 3D Scatter to plot the trajectory along with the target marker.

The interactive plot **(Fig 7)** facilitates spatial analysis of the trajectory profile, launch angles, and impact geometry through rotatable 3D axes.

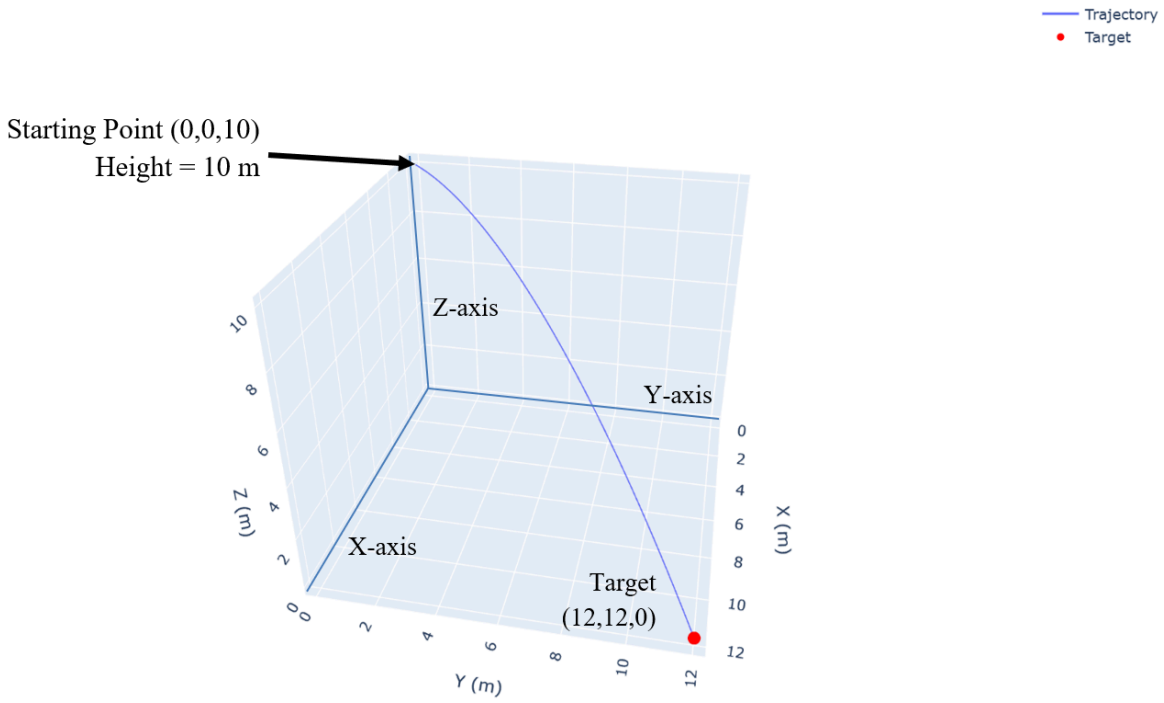

Fig 7 Spatial analysis of the trajectory profile

4.1.7 Interactive Plotting

A significant portion of the project effort was dedicated to developing clear, intuitive, and dynamic visual representations of the projectile system. To ensure compatibility across platforms and enable real-time interactive plotting within a desktop environment, we utilized the matplotlib library with the 'TkAgg' backend (import matplotlib; matplotlib.use('TkAgg')). This backend was chosen specifically to facilitate smooth integration of the plot windows with our user interface.

The system generates both 3D and 2D plots of the nitrogen ball's trajectory. The 3D plot offers a spatial view of the projectile motion, while the 2D plot provides a side projection to better understand height and distance over time. These visualizations are dynamically rendered based on user inputs or selected grid points, allowing immediate feedback and intuitive understanding of the motion.

Additionally, an interactive grid-based floor layout—similar to a chessboard—was created. Each cell in the grid represents a potential fire location. When a user clicks a specific square, the corresponding projectile path is auto-generated with updated plots and key motion parameters **(See Fig 8)**. These visual tools greatly enhanced the usability and presentation value of the system.

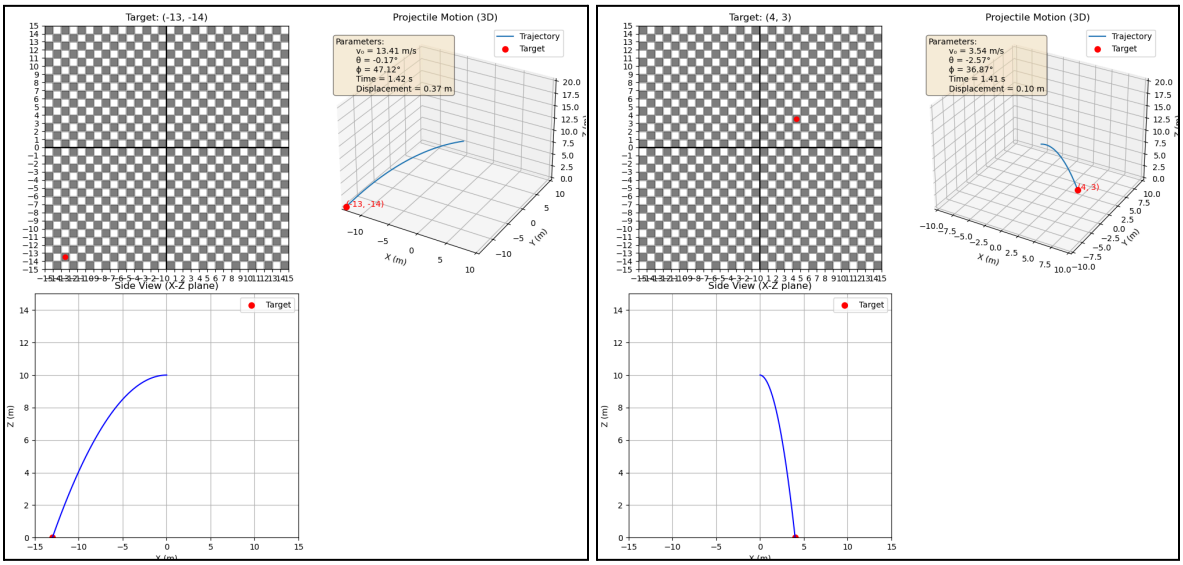

Fig 8 Visualization with a facility of grid-based floor layout to take inputs .

4.2 Fire Detection CNN Model

The fire detection system uses a Convolutional Neural Network (CNN) built in a sequential architecture, designed to classify images into two categories: **fire** and **no fire**. This model is lightweight yet powerful enough to extract key features from images and make real-time decisions for triggering a fire-extinguishing mechanism.

4.2.1 Layer-Wise Breakdown and Functionality

Layer 1: Conv2D (32 filters)

The first convolutional layer has 32 filters with a kernel size (default is 3x3), and it outputs a feature map of size 254x254x32. This layer learns to detect low-level features such as edges, color gradients, and corners. It is the most basic level of visual understanding and forms the foundation for further abstraction. The number of trainable parameters here is **896**.

Layer 2: MaxPooling2D

Following the first convolutional layer, a max-pooling layer with a 2x2 window is used. It reduces the spatial dimensions by half, resulting in an output of 127x127x32. This helps in reducing computational cost and retaining dominant features while discarding noise.

Layer 3: Conv2D (64 filters)

This second convolutional layer doubles the number of filters to 64 and produces an output of 125x125x64. It captures more complex patterns such as flame shapes, smoke outlines, and hot zones in the image. With **18,496** parameters, this layer contributes significantly to feature extraction.

Layer 4: MaxPooling2D

Another max-pooling layer further down samples the output to 62x62x64. This consistent downsampling helps in controlling overfitting and improves generalization.

Layer 5: Conv2D (128 filters)

The third convolutional layer has 128 filters, producing an output of 60x60x128. This layer captures high-level abstract features specifically relevant to fire patterns, such as the chaotic structure of flames, brightness, and texture. The number of trainable parameters here is **73,856**, indicating this layer has a dense learning capacity.

Layer 6: MaxPooling2D

Again, a pooling layer is applied to reduce dimensions, bringing the output size to 30x30x128. This ensures that only the most significant fire-specific features are retained for the classification layers.

Layer 7: GlobalAveragePooling2D

This layer reduces the 3D tensor of shape 30x30x128 into a 1D vector of size 128 by taking the average of each feature map. It helps in flattening the output and reducing the number of parameters, minimizing the risk of overfitting and providing better generalization than traditional flattening.

Layer 8: Dense Layer (128 units)

A fully connected dense layer with 128 neurons is added next. It processes the compressed vector from the previous layer and helps learn nonlinear combinations of features. With **16,512** trainable parameters, this layer plays a key role in final decision making.

Layer 9: Dropout Layer

A dropout layer is introduced to randomly deactivate a fraction of the neurons during training, typically 0.5 or 50%. This regularization technique prevents overfitting and enhances the model's robustness, especially for a limited dataset.

Layer 10: Output Dense Layer (1 unit)

The final layer has a single neuron with a sigmoid activation function, which outputs a probability between 0 and 1. If the value is below 0.5, the image is classified as "fire"; otherwise, "no fire". The layer has only **129** trainable parameters and acts as the final decision-maker.

4.2.2 Total Trainable Parameters

The entire model has **109,889 trainable parameters**, all of which contribute to learning patterns from the input images to make accurate classifications. Despite its relatively small size, the model is capable of high performance due to efficient layer design and regularization techniques.

4.2.3 Role of the Model in the Fire Extinguishing System

This CNN model acts as the **central detection unit** in an automated fire-extinguishing system. Here's how it works:

- A **camera feed** continuously captures images of the environment (e.g., an industrial setup).
- These images are passed through the CNN model for real-time classification.
- If **fire is detected**, the model sends a signal to a **controller unit**.
- This controller computes the **coordinates** of the fire and determines the necessary **angle, velocity, and spring compression** needed to launch a **nitrogen ball** using a spring-mass system.
- The **ball is projected** accurately to the fire's location, extinguishing it quickly and efficiently.

This integration ensures minimal human intervention, faster response times, and higher accuracy in fire suppression, especially in hazardous or remote areas.

The CNN-based model is not only effective at fire classification but is also optimized for real-world deployment. Its architecture is balanced, combining depth and generalization. By integrating this model with mechanical control systems, it provides a **smart, reliable, and autonomous solution** to fire emergencies. The project exemplifies a strong intersection of **deep learning**, **real-time automation**, and **mechanical actuation**.

4.3 Web development

4.3.1 Introduction

The 3D Projectile Motion Simulator is an interactive web application designed to visualize projectile motion in three dimensions **(Fig 9 - 12)**. The application allows users to input target coordinates (X & Y) and computes the optimal launch parameters (initial velocity, launch angle, azimuth angle, time to target, and displacement). The simulation results are displayed along with a 3D trajectory visualization.

4.3.2 Tech Stack

4.3.2.1 Frontend

- HTML, CSS, JavaScript: Used for structuring, styling, and client-side scripting.
- Plotly.js: For rendering the 3D trajectory visualization.
- Flask (Jinja2 Templating Engine): Handles dynamic rendering of the HTML pages.

4.3.2.2 Backend

- Flask (Python): Manages HTTP requests and server-side logic.
- NumPy & SciPy: Used for numerical calculations and optimization.
- Plotly (Python): Generates the 3D plot of the projectile motion.
- Mathematical Libraries: Includes functions for trigonometric calculations and physics-based computations.

4.3.3 Frontend Development

The frontend is designed to be interactive and visually appealing with the following components:

- User Input Form: Allows users to enter target X and Y coordinates.
- Result Display Section: Displays computed values such as velocity, angles, time, and displacement.
- 3D Visualization Panel: Uses Plotly.js to render the projectile's trajectory dynamically.
- Loading Animation: Enhances user experience by indicating processing time.

Frontend Workflow:

1. User enters X and Y coordinates in the input fields.
2. Upon form submission, JavaScript fetches the computed results from the backend.
3. The UI updates with the calculated values.
4. The trajectory graph is updated dynamically using Plotly.js.

4.3.4 Backend Development

The backend is responsible for processing user inputs, computing projectile motion, and generating a visualization. It follows a structured workflow:

Backend Workflow:

- Receive Input:
    - The Flask server receives a POST request containing target X and Y coordinates.
- Compute Optimal Launch Parameters:
    - Uses SciPy's optimization functions to determine the best velocity, launch angle, and azimuth angle that ensure the projectile reaches the target.
    - Solves quadratic equations for time calculations.
    - Computes displacement using energy equations.
- Generate 3D Trajectory Data:
    - Calculates projectile positions over time.
    - Filters invalid points where the projectile goes below ground level.
- Create a 3D Plot using Plotly (Python):
    - Plots the calculated trajectory along with the target marker.
    - Converts the plot into JSON format for frontend rendering.
- Send Response to Frontend:
    - Returns computed values and the trajectory plot data in JSON format.

4.3.5 Graphs and Visualization

- Plotly.js (Frontend): Renders the received trajectory data as a 3D scatter plot.
- Plotly (Python - Backend): Generates a 3D trajectory plot and converts it into JSON for frontend visualization.
- Graph Components:
    - Projectile Path: Displayed as a 3D curve.
    - Target Location: Marked as a red point.
    - Axes Labels: X (horizontal distance), Y (side distance), Z (height).

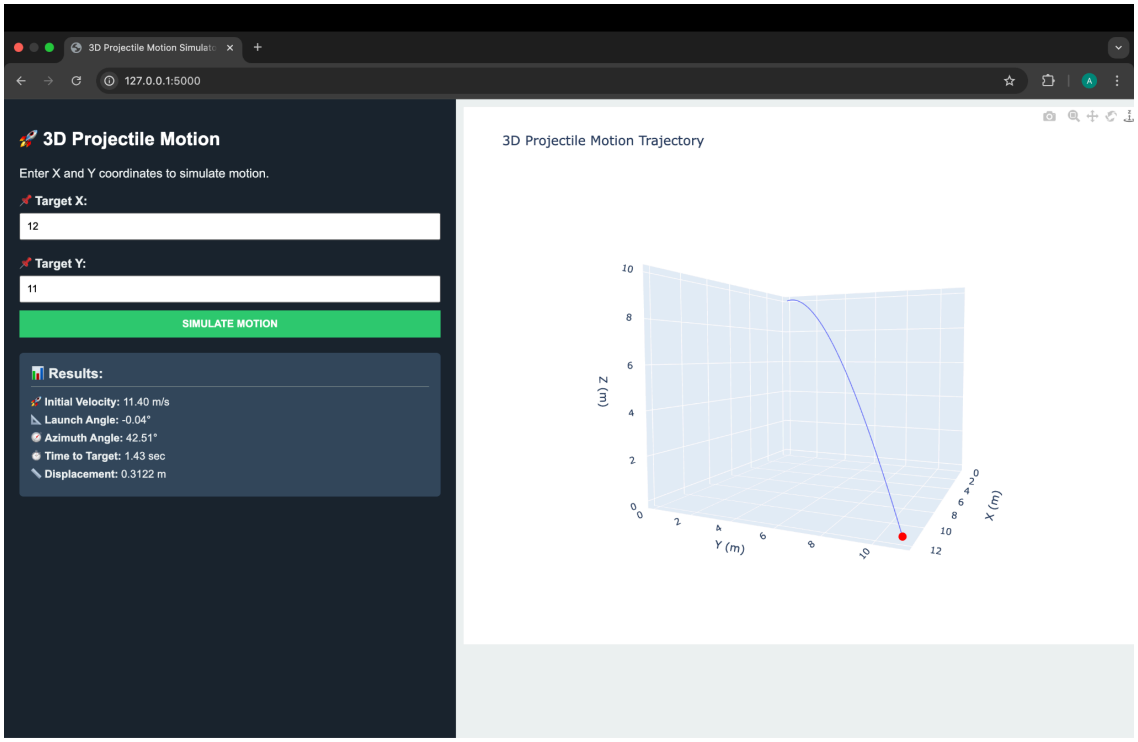

Fig. 9. User Interface of Web Page , 3D View Plot 1

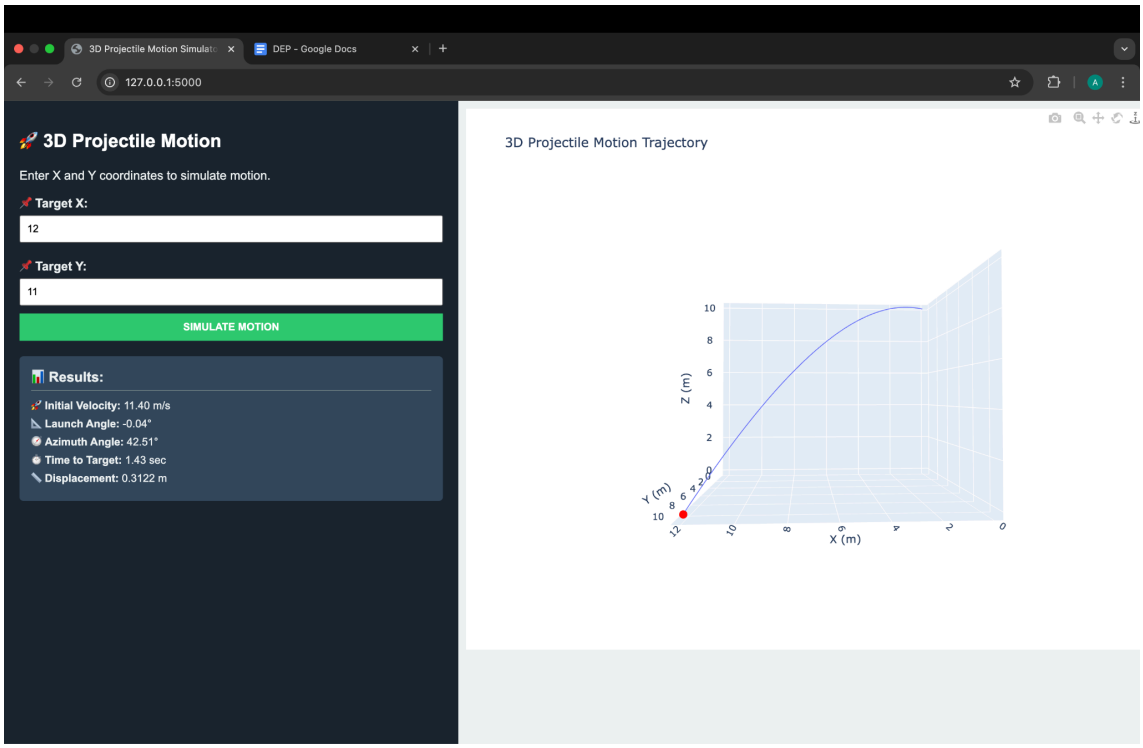

Fig 10 User Interface of Web Page , 3D View Plot 2

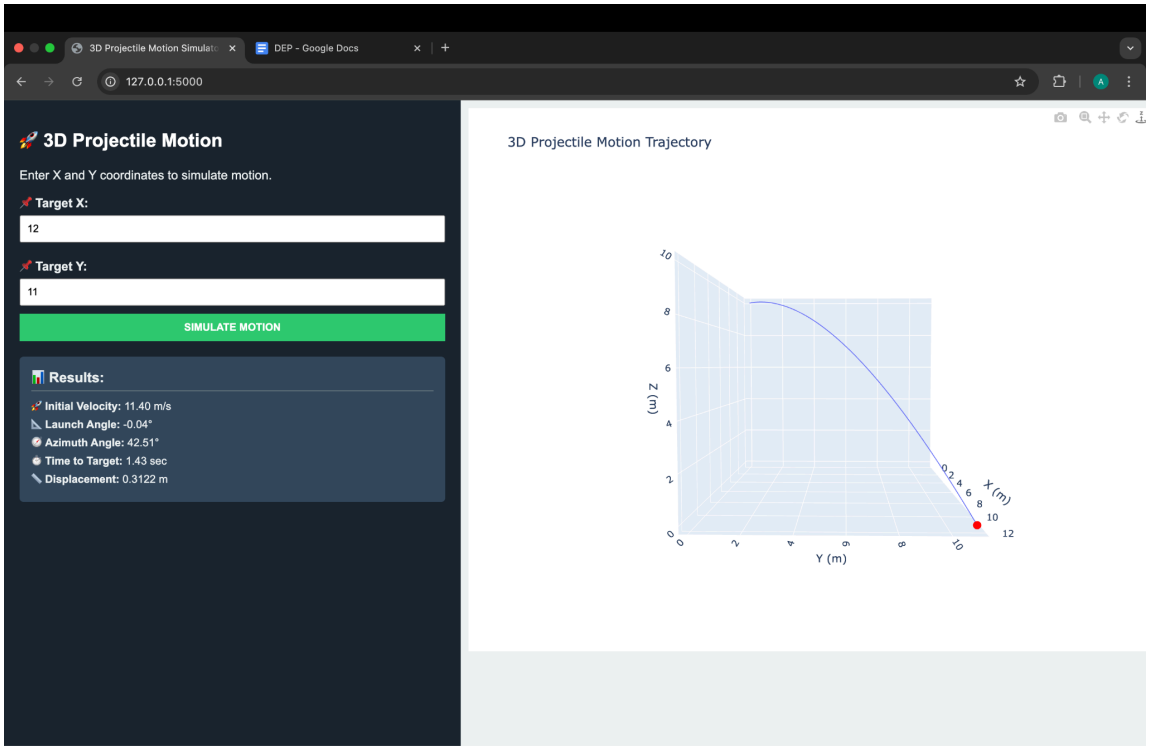

Fig 11 User Interface of Web Page , 3D View Plot 3

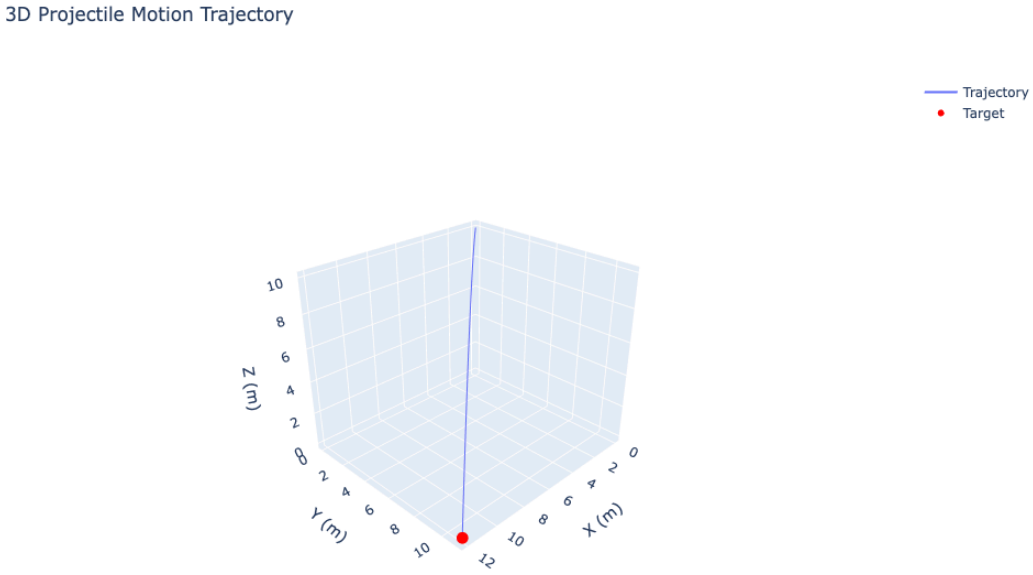

Fig 12 Snapshot of Plot

Chapter 6: Challenges Faced

The development of the interactive 3D projectile motion simulator involved several engineering, mathematical, and software integration challenges. While the goal was to create a seamless and educational interface that visualizes how projectiles behave in real time, implementing this required balancing computation, physics, interactivity, and optimization. This chapter outlines the major difficulties faced and how they were mitigated, while also discussing potential future considerations.

6.1 Code Complexity and Real-Time Constraints

A critical challenge was managing the trade-off between computational complexity and the responsiveness expected from an interactive tool. Every user click on the 2D plane triggers a numerical optimization problem to determine the velocity magnitude and launch angles necessary to hit the target from a fixed height.

- **Real-Time Expectations**: Users expect instant feedback, especially when interacting with visual elements. However, solving a nonlinear optimization problem is computationally intensive and can easily introduce latency.
- **Optimization Overhead**: The function simulate_projectile() relies on SciPy minimize() method. This method, while robust, iteratively tests candidate solutions, and each iteration calls the objective function which performs trajectory calculations and checks for validity.
- **Figure Redrawing Bottlenecks**: Plotting trajectories, impact points, and annotations in 3D and 2D views using Matplotlib requires significant rendering effort. Without careful management, the GUI becomes sluggish and unresponsive.
- **Mitigation Strategy**: Tightened bounds on the optimizer, avoided redundant figure updates, and used efficient NumPy operations. Code was modularized to separate plotting from optimization, and only valid, changed elements were redrawn to preserve speed.

6.2 Avoiding Non-Physical Trajectories (Imaginary Roots)

Another frequent issue arose from the fundamental equations of motion. The simulator computes the time of flight by solving a quadratic equation in the vertical direction:

$$z(t) = h_0 + v_0 sin(\theta) t - \frac{1}{2} g t^2 = 0$$

Solving for time t, we get:

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Here, a **negative discriminant** leads to imaginary solutions, which are physically meaningless for projectile motion. These cases occur when the combination of velocity and angle is insufficient to reach the ground.

- **Symptoms**: The optimizer fails or returns NaN values, and the plot either disappears or crashes.
- **Typical Scenarios**: Low velocities with high negative elevation angles, or large horizontal distances requiring flatter trajectories than possible.
- **Fix Implemented**: Used a try-except block and checked if the discriminant was negative. When detected, the objective function returned a large penalty (e.g., 1e6) to signal the optimizer to reject that path.

This approach maintained the stability of the optimizer and guided it away from non-physical solutions without crashing the program.

6.3 Model Accuracy vs Computation Speed Tradeoffs

Choosing the right model for projectile motion was another challenge. While real-life scenarios often involve air drag, wind, or even spin, incorporating these adds numerical complexity and slows down computation.

- **High Accuracy Models**: Would require solving differential equations numerically, for instance with Runge-Kutta or finite difference solvers. These are precise but computationally expensive.
- **Speed-First Approach**: Since this was an educational and interactive tool, the decision was made to prioritize fast feedback by using a simplified model:
    - Constant gravity g=9.8 m/s^2
    - No air resistance
    - Rigid body with no spin or deformation
- **Outcome**: The model can generate near-instantaneous responses suitable for educational demonstrations and optimization-based control logic, albeit with slightly reduced realism.

6.4 Choosing Bounds and Initial Guesses for Optimization

The success of the numerical optimizer is highly sensitive to both the **initial guess** and the **boundaries** defined for each parameter:

- **Initial Guess Used**: $[v_0, \theta, \phi] = [10, -45^{\circ}, 45^{\circ}]$
- **Parameter Ranges**:
  - Velocity: 0 to 15 m/s
  - Elevation angle: -90° to 0° (since projectile is launched downward from a height)
  - Azimuth: 0° to 360°
- **Challenge**: Poor initial guesses or too restrictive bounds lead to:
  - Increased computation time
  - Local minima trapping
  - Missed feasible solutions
- **Solution**: Calibrated bounds based on empirical testing, and used domain-specific knowledge (e.g., realistic maximum velocity) to narrow the space and speed up convergence.

6.5 Visualization Synchronization and Axes Management

The simulator uses three synchronized plots:

- A 3D plot showing the projectile in space
- A 2D side view
- Annotations of computed parameters

Maintaining consistency between these was harder than anticipated:

- **Dynamic Axis Ranges**: As the user clicks farther from the origin, the axis limits must be recalculated and updated to ensure visibility.
- **Cleanup Between Frames**: Overlapping lines, dots, and texts can clutter the display if not removed or refreshed properly.
- **Consistency**: The coordinate systems of all subplots had to remain in sync so the user could intuitively map what they clicked to what was displayed.

Mitigated by explicitly clearing axes with ax.clear() and modularizing the plotting code for clean reinitialization.

6.6 Handling Edge Cases and Grid Boundary Clicks

The simulation grid was defined from -15 to 15 meters in X and Y. However, user behavior is unpredictable:

- **Boundary Cases**: Clicking near the edges results in extremely shallow angles or very long distances, which are physically impossible within the capped velocity.
- **Unreachable Points**: Some clicked points simply require a launch speed higher than the max allowed (15 m/s).
- **Symmetry Management**: To simplify calculations, the optimizer assumes the projectile is in the first quadrant, then maps the result to the clicked quadrant.

Implemented checks for:

- Reflection handling
- Out-of-bounds warnings
- Avoidance of optimizer calls if the distance was clearly too far

6.7 Optimization Robustness and Convergence Reliability

While scipy.optimize.minimize() is suitable for general-purpose optimization, it's not always ideal for the sharp, penalty-based objective surface we used:

- **Non-Smooth Surface**: The heavy penalty for invalid trajectories introduces discontinuities that break gradient-based solvers.
- **Local Minima**: The solver can get stuck in suboptimal regions, particularly if the initial guess is too far off.
- **No Convergence Guarantee**: There's always a chance that the optimizer fails to find any valid trajectory, especially for edge cases.

Planned future improvements include:

- Trying global optimization methods (like differential_evolution)
- Multiple random seeds to retry convergence
- Adding visualization of optimizer progress to debug convergence failures

Chapter 7: Results

7.1 Optimisation Code

7.1.1 Result

The simulation successfully computes and visualizes the 3D trajectory of a projectile launched from a fixed height toward a user-selected point on a 2D grid interface. Upon each click on the grid:

- The system calculates the optimal initial velocity (v_0), launch angle (θ), and azimuth angle (ϕ) required to hit the target.
- A 3D trajectory plot displays the full motion path, while a 2D side view shows the motion in the X-Z plane.
- Key parameters such as time of flight and the corresponding spring displacement (based on energy conservation) are displayed alongside the plots.
- The simulation provides immediate visual and numerical feedback for each attempted target location.

7.1.2 Discussion

This simulation effectively demonstrates the principles of projectile motion, combined with energy transformation into a spring-mass system. By allowing user interaction through grid selection, it offers a hands-on approach to visualizing how launch conditions affect trajectory.

The optimization algorithm ensures accurate targeting within defined bounds, while the graphical output enhances understanding of spatial motion. The inclusion of spring displacement offers additional insight into kinetic-to-potential energy conversion.

However, the model assumes ideal conditions (no air resistance, constant gravity), which may not reflect real-world complexity. Future improvements could include drag effects, obstacle modeling, or animated motion to enhance realism. Overall, the simulation serves as a strong educational tool for physics and engineering concepts.

7.2 CNN Model

The performance of the Convolutional Neural Network (CNN) model for fire detection was evaluated using training and validation curves, classification metrics, confusion matrix, and final test set evaluation.

7.2.1 Training and Validation Accuracy & Loss

As observed in the training history plots (Figure), the model demonstrated an improving trend in training accuracy and a decreasing trend in training loss over 15 epochs. The training accuracy approached nearly 100%, while the training loss dropped below 0.2.

However, the validation accuracy(*Fig 13)* and loss(*Fig 14)* showed noticeable fluctuations. Validation accuracy oscillated significantly, ranging from near zero to nearly perfect accuracy, and validation loss peaked sporadically. These patterns may be attributed to either a small or unbalanced validation set or high intra-class variability in the validation images. Despite this, the final validation accuracy converged to a value similar to the training accuracy, indicating reasonable generalization.

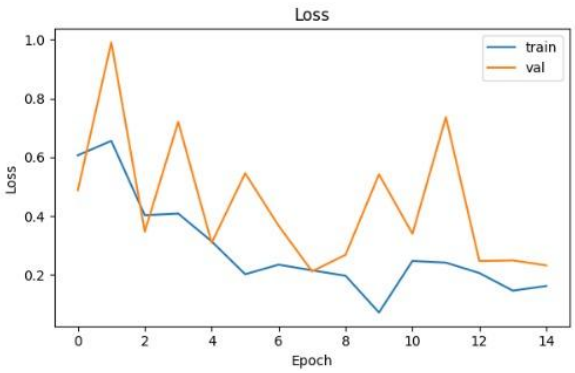

Fig 13 Validation loss Plot

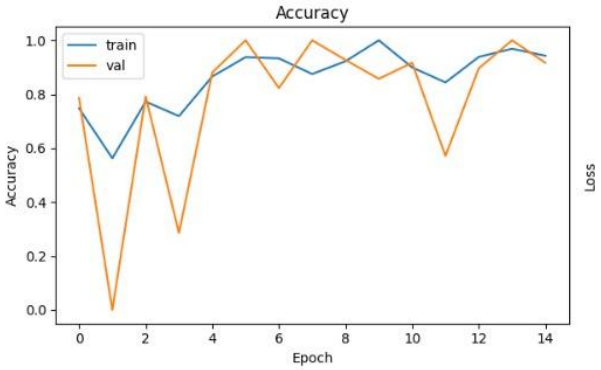

Fig 14 Validation Accuracy Plot

7.2.2 Confusion Matrix and Classification Metrics

The confusion matrix(*Fig 15)* and classification report provide a more detailed assessment of the model's performance:

- **Fire Detection**: 144 out of 151 fire images correctly classified (Recall: 0.95, Precision: 0.99)
- **No-Fire Detection**: 49 out of 50 no-fire images correctly classified (Recall: 0.98, Precision: 0.88)
- **Overall Accuracy**: 96%

The macro-averaged F1-score of **0.95** and weighted average F1-score of **0.96** reflect the model's balanced ability to classify both classes. Slight class imbalance was handled effectively.

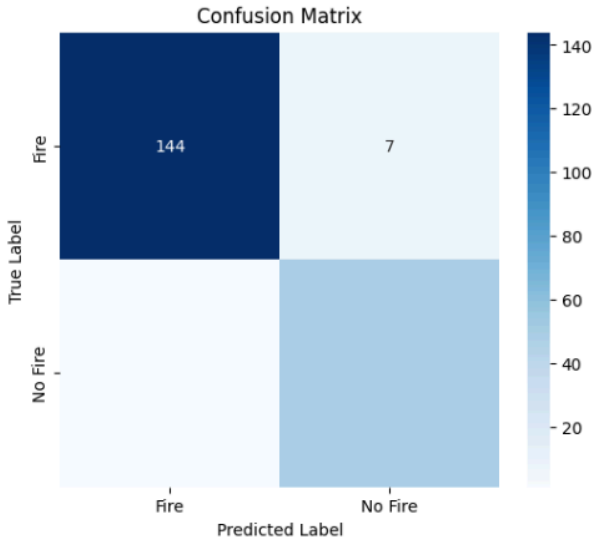

Fig 15 Confusion Matrix

Fig 16 Fire Detection Results

7.2.3 Test Set Evaluation

To assess the model's generalization capability, it was evaluated on an unseen test dataset, resulting in:

- **Test Loss**: 0.1039
- **Test Accuracy**: 95.83%

These values confirm that the model generalizes well beyond the training and validation sets. The low test loss implies confident predictions, while the high test accuracy reaffirms the model's reliability in real-world scenarios.

7.2.4 R^2 Score Interpretation

The model achieved an **R^2 score of 0.8372**, which, though more common in regression tasks, can provide insights into how well the model predictions approximate the ground truth. An R^2 value close to 1 indicates that the model captures a significant portion of the variance in the data. A score of 0.8372 suggests that approximately 84% of the variance in the class predictions can be explained by the model—highlighting its predictive strength.

Discussion

The high classification accuracy, strong precision-recall balance, and low loss indicate that the CNN model is effective for fire detection. The slightly unstable validation performance suggests the need for improvements like:

- Increased validation data
- Enhanced data augmentation
- Regularization techniques such as dropout
- Stratified cross-validation

Nonetheless, the model maintained robust performance on the test set, with minimal performance degradation. The high recall for both classes is especially critical for fire detection applications, where false negatives (missed fires) must be minimized. Thus, the model demonstrates strong potential for deployment in safety-critical systems, such as early fire detection in surveillance footage.

Chapter 8: Future Aspects

As we transition from a simulation-based fire extinguishing model to a deployable system, multiple hardware, software, and intelligence-driven extensions are envisioned. These improvements will enable the system to detect, analyze, and neutralize fire threats in real-time with scalability, reliability, and efficiency.

8.1 Hardware Integration for Fire Detection and Launch

The current simulation assumes that the fire coordinates are known and fixed. However, in a real-world scenario, detecting fire, localizing its coordinates, and launching an extinguishing projectile autonomously is critical.

Future Goals

- Sensor Integration: Use infrared (IR) sensors, thermal cameras, or smoke detectors to locate fire sources in real-time.
- Microcontroller Interface: Implement control logic on a microcontroller (e.g., Arduino or Raspberry Pi) to interface between the sensor input and launcher hardware.
- Projectile Launcher Mechanism: Design a physical spring or compressed-air-based launching mechanism capable of adjusting angle and velocity based on optimizer output.
- Feedback Loop: Use encoders and IMUs to monitor the actual angles of the launcher and provide feedback to ensure correct targeting.

Outcome

A semi-autonomous system that can detect fire, compute projectile parameters, and execute physical launch in real-world settings.

8.2 Real-Time Targeting and Motion Compensation

In dynamic scenarios (e.g., a forest fire with moving wind fronts or a moving drone), both the target and the launcher may be in motion. The current model does not account for this.

Future Enhancements

- Relative Motion Modeling: Implement kinematic equations to account for movement of the fire (target) and/or the launcher.
- Sensor Fusion: Use GPS + IMU + computer vision (CV) for real-time localization and tracking.
- Predictive Targeting: Implement a Kalman filter or similar motion estimation algorithms to predict the future position of fire and adjust launch parameters accordingly.
- Time-Delay Compensation: Add compensation logic for the time delay between calculation and projectile impact.

Outcome

An adaptive system capable of launching accurate projectiles even when the target or launcher is mobile.

8.3 Integration with IoT, Cloud Control, and Notifications

To make the system scalable and remotely operable, a cloud-connected architecture is essential.

Proposed Architecture

- Sensor Data Logging: All sensor data (temperature, humidity, fire location) logged to a cloud dashboard using platforms like AWS IoT or Google Firebase.
- Remote Launch Capability: Launchers can be remotely armed or triggered via secure mobile/web interface.
- Fire Event Notifications: Automated SMS/email/WhatsApp alerts can be sent when fire is detected.
- Data-Driven Improvements: Historical fire data can be used to train models or refine targeting.

Outcome

An IoT-enabled, cloud-monitored system for distributed deployment, useful in remote or high-risk zones.

8.4 Machine Learning for Fire Spread Prediction

In large-scale deployments, especially in forests or industrial sites, predicting how a fire will spread is crucial to plan effective responses.

Future Plan

- Data Collection: Collect data on temperature, humidity, wind speed, and terrain during fire simulations or from public datasets.
- ML Models: Use supervised learning models like Random Forests or neural networks to predict fire spread direction and speed.
- Simulation Feedback Loop: Combine ML predictions with real-time simulations to identify high-risk areas and prioritize targeting.
- Fire Containment Strategy: Integrate ML output to recommend optimal launcher placement or firebreak creation.

Outcome

A predictive system that not only reacts to fire but anticipates its movement and deploys countermeasures proactively.

8.5 Redesign for Multiple Launchers and Fire Zones

The existing implementation assumes a single launcher and a single target. However, most real-world fires involve multiple hotspots that need coordinated action.

Next Steps

- **Multi-Launcher Synchronization**: Develop communication protocols (e.g., via MQTT or ROS) for multiple launchers to operate in coordination.
- **Zone Allocation Algorithm**: Use clustering algorithms (e.g., K-Means) to divide the affected region into zones, assigning specific launchers to each.
- **Central Control System**: Implement a cloud or server-based controller that monitors all zones and dispatches launch commands.
- **Collision Avoidance**: When operating in close proximity, ensure that projectiles or launch fields of different units do not interfere.

Outcome
A scalable architecture for deploying multiple launchers in a coordinated way to extinguish fire across a large area.

8.6 Battery and Power Optimization

Goals

- Integrate **solar energy systems** to support off-grid, long-duration operation.
- Design **low-power electronic components** with power-saving modes (e.g., wake-on-fire-detection).
- Use **battery health monitoring** to ensure timely recharge or replacement.

Outcomes
Greater system longevity and sustainability in remote or rural fire-prone regions.

Chapter 9: Conclusion

This project presents a novel and integrated approach to automated fire suppression by leveraging projectile motion, spring-mass mechanics, and real-time computer vision. Through the simulation and development of a spring-actuated launcher, we demonstrated the feasibility of using nitrogen-filled fire-extinguishing balls to target and neutralize fire outbreaks in predefined spaces. The system employs optimized launch parameters — including velocity, launch angle, and azimuth — computed using a numerical solver to ensure precise targeting of the fire source from a fixed height. By modeling the projectile's motion in 3D space, the simulator provides both visual and numerical feedback, enhancing understanding and usability.

A key contribution of the project is the integration of a Convolutional Neural Network (CNN) trained to detect fire and non-fire images in real time. This AI component transforms the system into an intelligent detection platform, minimizing the risk of false positives while enabling swift identification of fire locations. The CNN achieved a high classification accuracy (~96%) and demonstrated strong generalization on test data, confirming its suitability for real-world application in fire detection.

Moreover, the development of a user-friendly web interface allows for interactive input, where users can specify fire coordinates or select grid points. Upon input, the system dynamically computes optimal parameters and visualizes the projectile trajectory using 3D and 2D plots. This functionality bridges theoretical physics with practical application, making the system both informative and demonstrative.

The project also identified and addressed various engineering challenges such as avoiding non-physical trajectories, balancing model accuracy with computational speed, and ensuring synchronization across visual outputs. Strategic optimization techniques and error-handling protocols were incorporated to enhance system robustness.

Looking ahead, the simulation offers a scalable framework for further developments. These include hardware implementation of the launcher system, real-time sensor integration, cloud-based control, and predictive modeling of fire spread. The possibility of deploying multiple launchers in coordinated zones was also explored, suggesting applications in industrial, forested, and hazardous environments.

In conclusion, the project not only demonstrates a functional prototype but also contributes to the growing field of smart fire safety systems. By integrating mechanical, computational, and AI components, it paves the way for future systems that can autonomously detect, evaluate, and respond to fire emergencies with speed and precision.

Chapter 10: References

[1] C. E. Mungan, “Optimizing the launch of a projectile to hit a target,” *U.S. Naval Academy, Annapolis, MD*.

[2] Z. Liu, A. Kim, and D. Carpenter, “A study of portable water mist fire extinguishers used for extinguishment of multiple fire types,” *Fire Safety Journal*, vol. 42, no. 1, pp. 25–42, Feb. 2007, doi: 10.1016/j.firesaf.2006.06.008.

[3] B. Aydin, E. Selvi, J. Tao, and M. J. Starek, “Use of fire-extinguishing balls for a conceptual system of drone-assisted wildfire fighting,” *Department of Engineering & Technology, Texas A&M University-Commerce*, Feb. 2019.

[4] A. A. Alsheikhy, "A Fire Detection Algorithm Using Convolutional Neural Network," *JKAU: Engineering Sciences*, vol. 32, no. 2, pp. 39–55, 2022, doi: 10.4197/Eng.32-2.3.

[5] P. K. Joshi, Z. Attaree, and P. K. Nawale, "Projectile motion," *Homi Bhabha Centre for Science Education, TIFR*, Mumbai, [Online].
Available: https://www.tifr.res.in/~pkjoshi/articles/projectile-motion.pdf

Chapter 11: Appendix

11.1 Codes

11.1.1 Optimization + Plotting

```
import matplotlib.pyplot as plt
import matplotlib
matplotlib.use('TkAgg')
import numpy as np
import math
from scipy.optimize import minimize
from mpl_toolkits.mplot3d import Axes3D

# Constants
g = 9.8  # gravity (m/s²)
h0 = 10  # launch height (m)
grid_size = 30

# Setup figure and subplots
fig = plt.figure(figsize=(10, 10))  # wider figure
ax_board = fig.add_subplot(2, 2, 1)  # chessboard
ax_traj = fig.add_subplot(2, 2, 2, projection='3d')  # 3D trajectory
ax_sideview = fig.add_subplot(2, 2, 3)  # 2D side view

current_dot = None

# --------- Draw Chessboard ----------
def draw_chessboard():
    ax_board.clear()
    ax_board.set_aspect('equal')
    ax_board.set_xlim(-grid_size//2, grid_size//2)
    ax_board.set_ylim(-grid_size//2, grid_size//2)
    ax_board.set_title(f"Click to Launch")
    for x in range(-grid_size//2, grid_size//2):
        for y in range(-grid_size//2, grid_size//2):
            color = 'white' if (x + y) % 2 == 0 else 'gray'
            rect = plt.Rectangle((x, y), 1, 1, facecolor=color, edgecolor='black')
            ax_board.add_patch(rect)
    ax_board.axhline(0, color='black')
    ax_board.axvline(0, color='black')
    ax_board.set_xticks(range(-grid_size//2, grid_size//2 + 1))
    ax_board.set_yticks(range(-grid_size//2, grid_size//2 + 1))
    ax_board.grid(True)

# --------- Trajectory Calculation ----------
def simulate_projectile(x_input, y_input):
    z_target = 0
    x_target = abs(x_input)
```

```
    y_target = abs(y_input)

    def objective(params):
        v0, theta, phi = params
        theta_rad = np.radians(theta)
        phi_rad = np.radians(phi)
        a = -0.5 * g
        b = v0 * np.sin(theta_rad)
        c = h0 - z_target
        discriminant = b**2 - 4 * a * c
        if discriminant < 0:
            return 1e6
        t1 = (-b + np.sqrt(discriminant)) / (2 * a)
        t2 = (-b - np.sqrt(discriminant)) / (2 * a)
        t = max(t1, t2) if max(t1, t2) > 0 else 1e6
        x = v0 * np.cos(theta_rad) * np.cos(phi_rad) * t
        y = abs(v0 * np.cos(theta_rad) * np.sin(phi_rad) * t)
        return (x - x_target)**2 + (y - y_target)**2

    result = minimize(objective, [10, 45, 45], bounds=[(0, 15), (-90, 0), (0, 360)])
    v0, theta, phi = result.x
    theta_rad = np.radians(theta)
    phi_rad = np.radians(phi)
```

```
    a = -0.5 * g
    b = v0 * np.sin(theta_rad)
    c = h0 - z_target
    discriminant = b**2 - 4 * a * c
    if discriminant < 0:
        return None

    t1 = (-b + np.sqrt(discriminant)) / (2 * a)
    t2 = (-b - np.sqrt(discriminant)) / (2 * a)
    t_final = max(t1, t2)

    t_vals = np.linspace(0, t_final, 500)
    x_vals = v0 * np.cos(theta_rad) * np.cos(phi_rad) * t_vals
    y_vals = v0 * np.cos(theta_rad) * np.sin(phi_rad) * t_vals
    z_vals = h0 + v0 * np.sin(theta_rad) * t_vals - 0.5 * g * t_vals ** 2

    if x_input < 0: x_vals = -x_vals
    if y_input < 0: y_vals = -y_vals

    #Spring_mass
    m = 1.5  # kg
    k = 2000  # N/m
    # Compute displacement
    disp = math.sqrt((m * v0 ** 2) / k)

    # Return extra parameters too
    return x_vals, y_vals, z_vals, x_input, y_input, v0, theta, phi, t_final, disp
```

```
# --------- On Click Event ----------
def on_click(event):
    global current_dot
    if event.inaxes != ax_board:
        return

    x, y = int(np.floor(event.xdata)), int(np.floor(event.ydata))
    print(f"\nClicked: ({x}, {y})")

    if current_dot:
        current_dot.remove()

    current_dot = ax_board.plot(x + 0.5, y + 0.5, 'ro')[0]
    ax_board.set_title(f"Target: ({x}, {y})")

    traj = simulate_projectile(x, y)
    ax_traj.clear()
    ax_sideview.clear()

    if traj:
        x_vals, y_vals, z_vals, xt, yt, v0, theta, phi, t_final, disp = traj

        # --- 3D Trajectory ---
        ax_traj.plot3D(x_vals, y_vals, z_vals, label='Trajectory')
        ax_traj.scatter([xt], [yt], [0], color='red', s=50, label='Target')
        ax_traj.text(xt, yt, 0, f"({xt}, {yt})", color='red', fontsize=10)
        ax_traj.set_xlim(min(-10, min(x_vals)), max(10, max(x_vals)))
        ax_traj.set_ylim(min(-10, min(y_vals)), max(10, max(y_vals)))
        ax_traj.set_zlim(0, max(20, max(z_vals)))
        ax_traj.set_xlabel('X (m)')
        ax_traj.set_ylabel('Y (m)')
        ax_traj.set_zlabel('Z (m)')
        ax_traj.set_title("Projectile Motion (3D)")
        ax_traj.legend()

        params_text = f"""Parameters:
        v₀ = {v0:.2f} m/s
        θ = {theta:.2f}°
        φ = {phi:.2f}°
        Time = {t_final:.2f} s
        Displacement = {disp:.2f} m"""
        ax_traj.text2D(0.05, 0.95, params_text, transform=ax_traj.transAxes,
                       fontsize=10, verticalalignment='top', bbox=dict(boxstyle="round", facecolor='wheat', alpha=0.5))
```

```
        # --- 2D Side View (X vs Z) ---
        ax_sideview.plot(x_vals, z_vals, color='blue')
        ax_sideview.scatter([xt], [0], color='red', s=50, label='Target')
        ax_sideview.set_xlim(min(-15, min(x_vals)), max(15, max(x_vals)))
        ax_sideview.set_ylim(0, max(15, max(z_vals)))
        ax_sideview.set_xlabel('X (m)')
        ax_sideview.set_ylabel('Z (m)')
        ax_sideview.set_title("Side View (X-Z plane)")
        ax_sideview.grid(True)
        ax_sideview.legend()

    else:
        ax_traj.set_title("Invalid solution")
        ax_sideview.set_title("Invalid solution")

    plt.draw()


# --------- Initialize ---------
draw_chessboard()
fig.canvas.mpl_connect('button_press_event', on_click)
plt.tight_layout()
plt.show()
```

11.1.2 CNN

```
import splitfolders

splitfolders.ratio(
    r'C:\Users\HP\Downloads\Mehta\fire_detection\dataset',          # fire/  no_fire/  (all images)
    output=r'C:\Users\HP\Downloads\Mehta\fire_detection\split_dataset',  # → train/ val/ test/ sub-dirs
    seed=42,
    ratio=(.6, .2, .2),              # 60% train, 20% val, 20% test
    group_prefix=None,              # keep paired images together if desired
    move=False                      # copy (default) or move files
)
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (Conv2D, MaxPooling2D,
                                     GlobalAveragePooling2D, Dense, Dropout)
import matplotlib.pyplot as plt
import numpy as np
import os

# -------------------- parameters --------------------
IMG_SIZE   = (256, 256)        # beware of GPU RAM needs
BATCH_SIZE = 32
EPOCHS     = 15
base_path  = r'C:\Users\HP\Downloads\Mehta\fire_detection\split_dataset'

# ----------------------------------------------------

# 1) data augmentation only for *training*
train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

# 2) bare rescale for val / test
testval_datagen = ImageDataGenerator(rescale=1./255)

train_gen = train_datagen.flow_from_directory(
        os.path.join(base_path, 'train'),
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='binary',
        shuffle=True)

val_gen = testval_datagen.flow_from_directory(
        os.path.join(base_path, 'val'),
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='binary',
        shuffle=False)          # keep order for nice reports
```

```
test_gen = testval_datagen.flow_from_directory(
        os.path.join(base_path, 'test'),
        target_size=IMG_SIZE,
        batch_size=BATCH_SIZE,
        class_mode='binary',
        shuffle=False)

model = Sequential([
    Conv2D(32, 3, activation='relu', input_shape=IMG_SIZE + (3,)),
    MaxPooling2D(2),

    Conv2D(64, 3, activation='relu'),
    MaxPooling2D(2),

    Conv2D(128, 3, activation='relu'),
    MaxPooling2D(2),

    # Replaces Flatten() so it works with any IMG_SIZE
    GlobalAveragePooling2D(),

    Dense(128, activation='relu'),
    Dropout(0.5),

    Dense(1, activation='sigmoid')
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

model.summary()


# ── 4. Training ──────────────────────────────────────────────────────
history = model.fit(
    train_gen,
    epochs=EPOCHS,
    validation_data=val_gen,
    steps_per_epoch=train_gen.samples // BATCH_SIZE,
    validation_steps=val_gen.samples // BATCH_SIZE
)
```

```
# ── 5. Plot Learning Curves ──────────────────────────────────────
plt.figure(figsize=(12,4))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epoch'); plt.ylabel('Accuracy')
plt.legend(); plt.title('Accuracy')

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('Epoch'); plt.ylabel('Loss')
plt.legend(); plt.title('Loss')

plt.tight_layout()
plt.show()


# ── 6. Final Test Evaluation ─────────────────────────────────────
test_loss, test_acc = model.evaluate(
    test_gen,
    steps=test_gen.samples // BATCH_SIZE
)
print(f"\nTest  Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_acc:.4f}")


# ── 7. Save the Model ────────────────────────────────────────────
model.save('fire_detection_model.h5')


# ── 8. Single-Image Prediction ───────────────────────────────────
from tensorflow.keras.utils import load_img, img_to_array

def predict_fire(image_path):
    img = load_img(image_path, target_size=IMG_SIZE)
    x   = img_to_array(img)[None, ...] / 255.0
    p   = model.predict(x)[0,0]
    label = "🔥 Fire Detected!" if p<0.5 else "✅ No Fire"
    print(f"{label} (conf={p*100:.1f}%)")

    plt.imshow(img)
    plt.title(label)
    plt.axis('off')
    plt.show()

img= r'C:\Users\HP\Downloads\Mehta\fire.jpg'
predict_fire(img)
```

```
def predict_fire(image_path):
    img = load_img(image_path, target_size=IMG_SIZE)
    x   = img_to_array(img)[None, ...] / 255.0
    p   = model.predict(x)[0,0]
    label = "🔥 Fire Detected!" if p<0.5 else "✅ No Fire"
    print(f"{label} (conf={p*100:.1f}%)")

    plt.imshow(img)
    plt.title(label)
    plt.axis('off')
    plt.show()

img= r'C:\Users\HP\Downloads\Mehta\non_fire.jpg'
predict_fire(img)
```