**Chapter 1**

**INTRODUCTION**

Nav-Oculus, short for Navigation and Interaction in Real-time Simulations, represents a cutting-edge advancement in artificial intelligence (AI) designed to enhance the realism and efficiency of virtual environments. Developed at the intersection of AI and simulation technology, Nav-Oculus is poised to revolutionize how virtual entities navigate, interact, and perceive their surroundings in real-time.

At its core, Nav-Oculus leverages state-of-the-art algorithms in pathfinding and decision-making to simulate intelligent behavior within dynamic virtual environments. Unlike traditional AI systems that rely on predefined paths or simplistic decision trees, Nav-Oculus integrates advanced machine learning techniques to adapt and respond to changing conditions in real-time simulations. This adaptive capability allows virtual agents to navigate complex terrain, avoid obstacles, and interact with objects and other entities seamlessly.

One of the key strengths of Nav-Oculus lies in its ability to simulate human-like navigation behaviors, enhancing the immersion and realism of virtual worlds. By incorporating probabilistic reasoning and learning from past interactions, Nav-Oculus enables virtual agents to make contextually aware decisions, such as choosing optimal routes or adjusting their movements based on environmental cues.

In conclusion, Nav-Oculus represents a significant leap forward in AI-driven navigation and interaction for real-time simulations. By combining advanced pathfinding algorithms, machine learning capabilities, and scalable architecture, Nav-Oculus not only enhances the fidelity of virtual environments but also opens new possibilities for the development of more immersive and interactive digital experiences.

* 1. **Objective**

The objective of the Nav-Oculus project is to push the boundaries of AI technology in real-time simulations by focusing on navigation and interaction capabilities. It aims to develop AI algorithms that can simulate intelligent behaviors akin to human decision-making, enhancing the realism and immersion of virtual environments. Key goals include improving pathfinding efficiency, enabling adaptive responses to dynamic changes in virtual worlds, and facilitating seamless interaction between virtual agents and their surroundings. Scalability is another crucial objective, ensuring that Nav-Oculus can support large-scale simulations with numerous entities while maintaining high performance. Ultimately, Nav-Oculus seeks to revolutionize applications such as virtual training environments, interactive gaming, and virtual reality experiences by delivering enhanced realism and interactivity through advanced AI-driven navigation and interaction capabilities.

**1.2** **Existing System**

The existing system of the Nav-Oculus project integrates advanced AI technologies such as pathfinding algorithms (e.g., A\*) and decision-making frameworks (e.g., behavior trees) within real-time simulation environments. It focuses on enabling virtual agents to navigate, avoid obstacles, and interact with objects based on predefined rules and algorithms. Basic machine learning techniques are also utilized to improve agent behaviors through experience within simulations. The system typically operates within specialized simulation platforms or game engines, ensuring compatibility with real-time rendering and physics simulations. Challenges include optimizing computational efficiency for larger-scale simulations and enhancing adaptability to dynamic environmental changes, areas of ongoing research to enhance Nav-Oculus's capabilities.

**1.3 Proposed System**

The proposed system of the Nav-Oculus project represents a significant advancement in AI-driven navigation and interaction for real-time simulations. It introduces state-of-the-art pathfinding algorithms designed to enhance the efficiency and adaptability of virtual agents in navigating complex environments with dynamic obstacles. Machine learning techniques will be integrated to enable adaptive decision-making, allowing agents to learn from their interactions and adjust behaviors accordingly within the simulation. Real-time interaction capabilities will be improved to enable seamless engagement with objects and other entities, aiming to enhance the realism and immersion of virtual environments. Scalability and performance optimizations are also key focuses, ensuring the system can handle large-scale simulations with numerous entities while maintaining responsiveness and computational efficiency. By incorporating these innovations, the proposed Nav-Oculus system aims to redefine the capabilities of AI in simulation technology, promising more realistic and interactive virtual experiences across various applications such as training simulations, interactive gaming, and virtual reality environments. During the credit card transaction, the fraud is detected and the number of false alert is being minimized by using genetic algorithm. Instead of maximizing the numbers of correctly classified transactions, the objective function where the misclassification costs are variable is defined and thus, correct classification of some transactions are more important than correctly classifying the others. The high amount of losses due to fraud and the awareness of the relation between loss and the available limit have to be reduced.

**Chapter 2**

**SYSTEM REQUIREMENTS SPECIFICATION**

It gives the information regarding analysis done for the proposed system. System Analysis is done to capture the requirement of the user of the proposed system. It also provides the information regarding the existing system and also the need for the proposed system. The key features of the proposed system and the requirement specifications of the proposed system are discussed below.

**2.1 Hardware Requirements**

Processor type : Intel i3 or faster

RAM : 4GB or more

HARD DISK : 20GB or more

Microcontroller : Arduino UNO

Camera Device : ESP 32 CAM

**2.2 Software Requirements**

Application software : Arduino IDE

Operating System      : Windows 10 or more

Tool : Arduino Software IDE

**2.3 Functional Requirements**

The functional requirements of the Nav-Oculus project include:

1. Advanced Pathfinding: Implement efficient and adaptive pathfinding algorithms to navigate virtual agents through complex and dynamic environments, avoiding obstacles in real-time.
2. Adaptive Decision-Making: Integrate machine learning techniques to enable virtual agents to learn from interactions, make adaptive decisions, and respond dynamically to changes in their environment.
3. Real-time Interaction: Develop capabilities for virtual agents to interact seamlessly with objects and other entities in real-time, enhancing the realism and interactivity of simulations.
   1. **Non - Functional Requirements**
4. Performance and Reliability:

* Real-Time Response: The system must provide real-time responses with minimal latency for navigation updates and hazard detection.
* High Availability: Ensure continuous operation with minimal downtime and robust fault tolerance to handle hardware or software failures.
* Accuracy: Maintain high accuracy in both navigation and image recognition tasks.

1. Security and Data Privacy:

* Data Protection: Ensure user data is protected and complies with relevant data protection regulations.
* Access Control: Implement robust authentication and authorization to prevent unauthorized access.

1. Usability and Maintainability:

* User-Friendly Interface: Design an intuitive interface for easy user interaction and accessibility.
* Modular Design: Facilitate easy updates and maintenance with a modular system architecture.
* Comprehensive Documentation: Provide thorough documentation to support maintenance and upgrades.

1. Interoperability and Compatibility:

* System Compatibility: Ensure compatibility with various hardware devices, operating systems, and other software systems.
* Standards Compliance: Adhere to industry standards and protocols to facilitate seamless integration with other systems and services.

1. Efficiency and Scalability:

* Resource Optimization: Optimize computational and memory resource usage to minimize power consumption, especially for mobile and embedded applications.
* Scalability: Design the system to scale efficiently, accommodating increased user loads and data volumes without performance degradation.

**Chapter 3**

**DESIGN**

NAV-OCULUS, a sophisticated technological framework, integrates advanced navigation capabilities with cutting-edge image recognition technologies. The system architecture is layered, with a presentation layer that includes a user-friendly interface accessible on mobile devices, in-car displays, and desktops, enhanced by augmented reality (AR) for intuitive navigation. The application layer comprises a navigation engine, image recognition engine, and a sensor fusion module. The navigation engine handles route planning, real-time updates, and traffic management, while the image recognition engine processes visual data for object detection and classification using deep learning models. The sensor fusion module integrates data from various sensors such as GPS, accelerometers, and gyroscopes to improve positioning accuracy and reliability.

**3.1 Data Flow Diagram**

IMAGE DATA

PROCESSING

FEATURE EXTRACTION

CLASSIFICATION

NAVIGATION COMMAND GENERATION

OUTPUT

COMMANDS

GENERIC ALGORITHM:

SIGN DETECTION AND RECOGNITION

PREPROCESS DATA

CLASSIFICATION AND DETERMINING MOVEMENT

INITIALIZE SYSTEM

ACQUIRE DATA

(CAMERA MODULE)

MOTOR MOVEMENTS CONTROL

CLASSIFICATION AND DETERMINING MOVEMENT

**Fig 3.1 Dataflow Diagram**

**3.2 System Architecture**

Arduino UNO  
(Navigation control)

ESP32 Camera(Image Recognition)

ESP32(Image Processing,Feature Extraction,Classification)

Motor Driver  
(Motor Controller)

SVM Model  
(Traffic Sign Recognition)

**Fig 3.2 System Architecture  
Fig 3.2 represents the system architecture of nav-oculus**

The above architecture describes the work structure of the system.

1. **Image Capture**: The ESP32 camera captures images of the environment.
2. **Preprocessing**: The ESP32 preprocesses the images (grayscale conversion, resizing, normalization).
3. **Feature Extraction**: The ESP32 extracts features from the preprocessed images.
4. **Classification**: The ESP32 uses the SVM model to classify the traffic signs.
5. **Navigation Command**: The ESP32 sends the classification result to the Arduino.
6. **Motor Control**: The Arduino generates and sends motor commands to the motor driver based on the received traffic sign classification.
7. **Robot Movement**: The motor driver controls the robot's movement according to the commands.

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 Module Implementation**

Implementation of system means the process of converting a new or revised system design into operational one.

The implementation process has four steps:

1. **Image Recognition Engine:** Uses traditional image processing techniques and algorithms for object detection and classification, optimized for efficient edge deployment.
2. Employs algorithms such as A\* for route planning, integrates real-time traffic data using Apache Kafka, and provides turn-by-turn guidance.
3. Combines data from GPS, accelerometers, and gyroscopes to enhance positioning accuracy.
4. Designed with a microservices architecture to ensure scalability and flexibility, allowing individual components to be independently scaled and updated based on system demands.

**Genetic Algorithm**

For the NAV-OCULUS project, a genetic algorithm can be utilized to optimize routing and navigation. By evolving a population of potential routes through selection, crossover, and mutation processes, the algorithm can efficiently find optimal or near-optimal paths based on various criteria such as travel time, distance, and real-time traffic conditions. This approach enables dynamic and adaptive route planning, improving navigation accuracy and efficiency by continually refining solutions based on performance metrics.

The simple method of the genetic algorithm is as shown in Fig 5.1.

Display the solution

Compare data with critical value

Calculation and finding critical

Values and set threshold

Generate data from the Data set

Select the Data set

**Fig 4.1 Flow Chart for Genetic Algorithm**

**Dataset**

A dataset is a collection of [data](http://en.wikipedia.org/wiki/Data), which is presented in [tabular](http://en.wikipedia.org/wiki/Table_(database)) form. Each [column](http://en.wikipedia.org/wiki/Column_(database)) represents a particular variable and each [row](http://en.wikipedia.org/wiki/Row_(database)) corresponds to a given member of the dataset. Several characteristics define a dataset's structure and properties. These include the number and types of the attributes or variables, and various [statistical measures](http://en.wikipedia.org/wiki/Statistical_measure) applicable to them. The dataset consists of a single column of values, often represented as a list. The values may be numbers, such as [real numbers](http://en.wikipedia.org/wiki/Real_number) or [integers](http://en.wikipedia.org/wiki/Integer), it may also be [nominal data](http://en.wikipedia.org/wiki/Nominal_data). The dataset consist of the following attributes as shown below.

**Table 4.1 Dataset Attributes**

|  |  |
| --- | --- |
| Attribute number | Attribute |
| 1 | Image Id |
| 2 | Current Navigation Path |
| 3 | Vehicle Speed |
| 4 | Coordinates |
| 5 | Camera Angle |
| 6 | Detect Traffic sign |
| 7 | Sign Confidence Score |
| 8 | Object Detection |
| 9 | Weather Conditions |
| 10 | Road Conditions |
| 11 | Planned Route |
| 12 | Alternate Route |

The modules are as shown below:

**Module 1 : Data Acquisition and Processing**

**Module 2 : Traffic Sign Detection and Classification Module**

**Module 3 : Decision Making and Navigation Planning Module**

**4.1.1 Module 1: Data Acquisition and Processing**

The Data Acquisition and Processing Module is responsible for collecting and preparing data from various sensors and inputs. This module integrates data from vehicle sensors such as cameras, LiDAR, and GPS, capturing real-time information about the vehicle's environment. The raw data is then processed to remove noise, normalize values, and filter out irrelevant information. This preprocessing ensures that the data is suitable for further analysis and decision-making.

The module also includes data management tasks such as timestamping and organizing data into structured formats. This organized data is crucial for accurate traffic sign detection and navigation planning. The effectiveness of subsequent modules depends heavily on the quality of data provided by this module. Hence, robust data acquisition and preprocessing are fundamental to the overall performance of the Nav-Oculus system.

DATA ACQUISTION AND PREPROCESSING

ESP32 CAMERA MODULE

RECOGNISED SIGNS AS OUTPUT SUMBOLS

**Fig 4.2 Data Acquisition and Processing  
Fig 4.2 shows how the data acquisition and processing happens**

**4.1.2 Module 2: Traffic Sign Detection and Classification**

The Traffic Sign Detection and Classification Module is pivotal in identifying and interpreting traffic signs from the preprocessed data. This module uses computer vision techniques and machine learning algorithms to detect and locate traffic signs within images captured by the vehicle's cameras. Once detected, the system classifies the signs into predefined categories such as STOP, Yield, or Speed Limit.

1. Based on Functionality.
2. Based on Techniques Used.
3. Based on Performance Metrics.
4. Based on Challenges and Solutions.
5. Based on User Interface Impact.

* **Based On Functionality**

Based on the functionality of the Traffic Sign Detection and Classification Module, this component is responsible for identifying and categorizing traffic signs from images captured by the vehicle's cameras. Its primary role is to process visual data to detect the presence of various traffic signs and classify them into categories such as STOP, Speed Limit, and Yield. Accurate detection and classification are essential for interpreting traffic regulations and making appropriate driving decisions.

* **Based On Technique Used**

Based on the techniques used in this module, advanced computer vision and machine learning algorithms are employed. Object detection models like YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) are utilized to locate and identify traffic signs within images. These models are trained on extensive datasets that include a wide variety of traffic signs in different conditions, ensuring robustness and accuracy in real-world scenarios.

* **Based On Performance Metrics**

Based on performance metrics, the module's effectiveness is measured through several key indicators. Detection accuracy calculates the proportion of correctly identified traffic signs compared to the total number of signs present in the images. Classification accuracy assesses how accurately the detected signs are categorized. Additionally, the confidence score provides a measure of the model’s certainty regarding each detected sign, helping to minimize false positives and enhance the reliability of the system.

* **Based On Challenges and Solution**

Based on the challenges faced by this module, such as variations in sign appearance due to lighting conditions, weather, and angles, several solutions are implemented. Data augmentation techniques are used during training to expose the model to a diverse range of scenarios. Furthermore, a multi-stage detection approach refines initial detections to improve accuracy and reduce false positives, addressing the challenges effectively.

* **Based On User Interface Impact**

Based on the impact on the User Interface, the results from this module are communicated to the driver through visual and auditory instructions. Accurate and clear presentation of detected and classified signs ensures that the driver receives timely and understandable navigation cues. This facilitates better decision-making and enhances overall driving safety, by providing critical information about road signs and conditions.

RECEIVED SYMBOLS

FROM CAMERA MODULE

TRAFFIC SIGN DETECTION AND CLASSIFICATION

DETERMINING

MOVEMENT OF CAR

**Fig 4.3 Traffic Sign detection and recognition  
Fig 4.3 shows the flow diagram of traffic sign detection and recognition**

RECEIVE DATA INTO THE MICROCONTROLLER

CLASSIFY THE DATA APPROPRIATELY

PLAN NAVIGATION

EXECUTE SPECIFIED COMMANDS

**Fig 4.4 Traffic Sign Detection and Classification Module  
Fig 4.4 shows the flow diagram of the traffic sign classification module**

**4.1.3 Module 3: Decision Making and Navigation Planning Module**

The Decision Making and Navigation Planning Module utilizes the output from the Traffic Sign Detection and Classification Module to make real-time navigation decisions. Based on the detected signs and other contextual information (such as vehicle speed and road conditions), this module determines the appropriate actions, such as adjusting the route or altering driving behavior. The decisions are based on pre-defined rules and algorithms that optimize the vehicle's path and ensure compliance with traffic regulations.

The module also generates navigation instructions and plans alternative routes if necessary. These instructions are then sent to the User Interface Module, where they are presented to the driver. Effective decision-making and navigation planning are crucial for the safety and efficiency of autonomous driving, ensuring that the vehicle navigates smoothly through various traffic scenarios.

The module diagram of Decision Making and Navigation Planning Module is as shown below

NO

Transaction continues

YES

Fraud occurred

Occurrences of the credit card are found and the critical value is calculated. This critical value is compared with the predefined value

A screenshot of a computer program

Description automatically generated

**Fig 4.4 Decision Making and Navigation Planning**

**CHAPTER 5**

Software testing for a project like Nav-Oculus involves several stages, each ensuring the system meets its requirements and functions correctly in various scenarios. First, unit testing is conducted to verify the functionality of individual components or modules in isolation, using frameworks like unittest or pytest. Next, integration testing ensures that different parts of the system work together as expected, often using mock objects to simulate interactions. System testing involves testing the complete integrated system to ensure it meets specified requirements, including scenario testing for real-world use cases and performance testing under different conditions. Finally, acceptance testing, often involving end users and stakeholders, ensures the system meets business requirements and is ready for deployment. Each of these stages helps ensure the reliability, efficiency, and effectiveness of the Nav-Oculus navigation module.

**5.1 Testing Strategies**

Testing strategies for a project like Nav-Oculus start with unit testing, verifying individual components' functionality using frameworks like Arduino Wiring. Integration testing follows, ensuring modules work together as expected with mock objects simulating interactions. System testing then verifies the complete integrated system against specified requirements, including scenario and performance testing. Finally, acceptance testing ensures the system meets business requirements and is ready for deployment, often involving end users and stakeholders. This comprehensive approach ensures reliability, efficiency, and correctness.

* **Unit Testing**

Unit testing focuses verification effort on smallest unit of software design-the software component or module. The test that occurs as part of unit testing is given below:

* **Objective**: Verify the functionality of individual components or modules in isolation.
* **Tools**: Use testing frameworks like Arduino Wiring in Arduino IDE.
* **Focus**: Test single functions or methods with various inputs and expected outputs.
* **Examples**: Validate sensor data acquisition functions, data processing algorithms, and decision-making logic.
* **Benefits**: Identify defects early, pinpoint errors, and ensure each part works correctly before integration.

Unit testing is normally considered an adjunct to coding step. After source level coding has been developed, reviewed and verified for correspondence to component level test case begins.

* **Integration Testing**

Integration testing focuses on verifying that different modules or components of the Nav-Oculus system work together as expected. This involves testing the interactions between integrated units to ensure seamless communication and functionality. Tools like unittest or pytest are used, often with mock objects to simulate interactions between components. For instance, integration testing might check how well the sensor data acquisition integrates with data processing and how decision-making algorithms interact with control functions. This stage helps identify issues in the interaction between components and ensures that the combined modules function correctly as a cohesive system.

* **Validation Testing**

Validation testing ensures that the Nav-Oculus system meets its specified requirements and functions correctly in real-world scenarios. This phase involves testing the system against defined use cases and performance criteria to verify that it performs as expected. Validation testing typically includes user acceptance testing, where end users and stakeholders assess the system to confirm it satisfies their needs and expectations. By simulating real-world conditions and evaluating the system's behavior in these contexts, validation testing helps ensure that the final product is reliable, effective, and ready for deployment.

* **System Testing**

System testing involves evaluating the complete, integrated Nav-Oculus system to ensure it meets the overall requirements and functions correctly in all intended scenarios. This comprehensive testing phase includes validating the end-to-end workflow, where each component interacts as expected within the complete system.

**5.2 Test Cases**

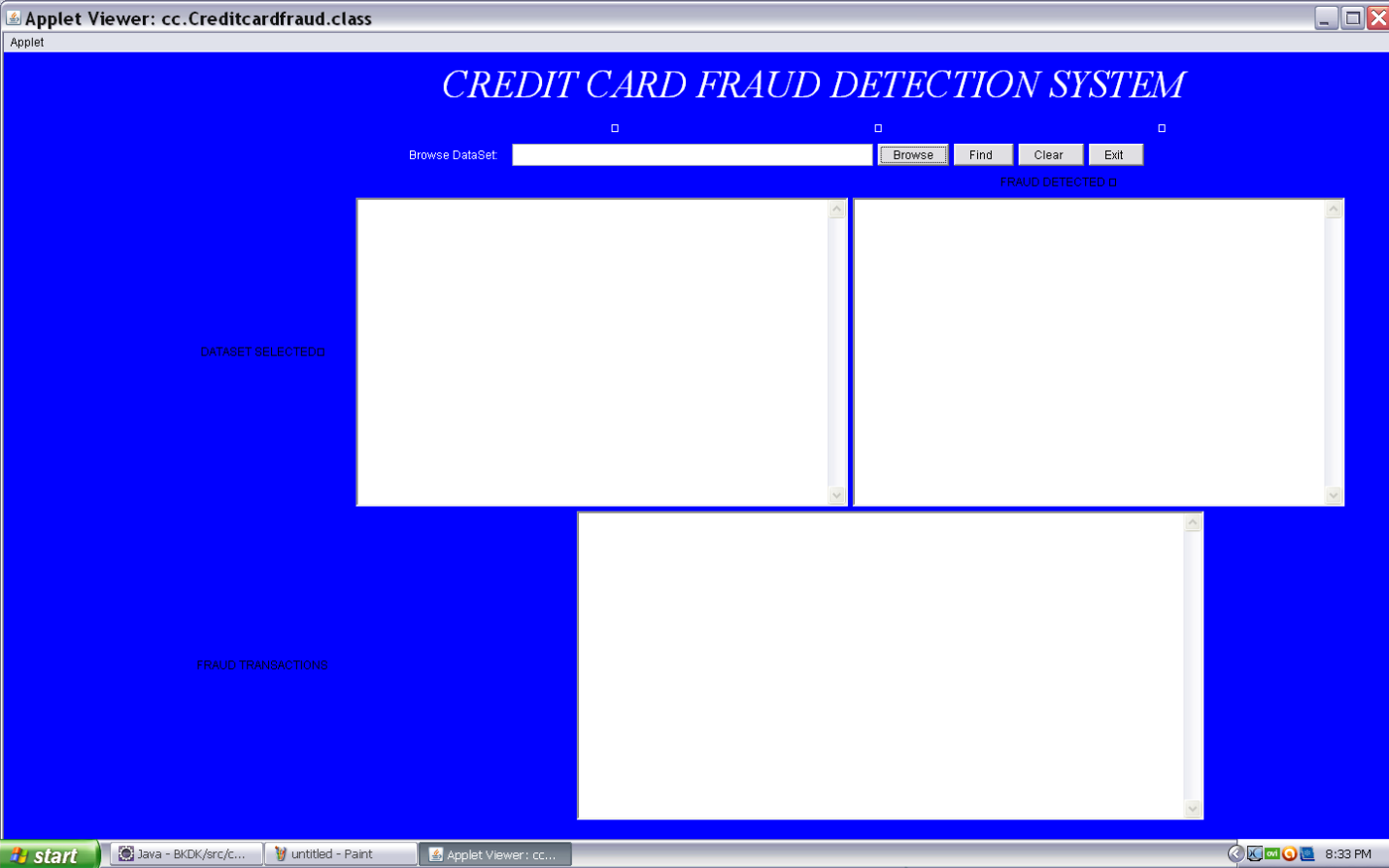
**Table 5.1 Test Case Specifications**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TestCase Number** | **Description** | **Input Data** | **Expected Output** | **Obtained Output** |
| 1 | |  | | --- | | Image Preprocessing |  |  | | --- | |  |  |  | | --- | |  | | Raw traffic sign image | Grayscale image of uniform size with normalized pixel values | [Actual obtained output after running the preprocessing code] |
| 2 | Feature Extraction | Preprocessed image | Feature vector (e.g.HOG features) | [Actual obtained output after running the feature extraction code] |
| 3 | Model Training | |  | | --- | |  |  |  | | --- | | Training dataset | | Trained SVM model with reasonable accuracy on the validation set | [Actual obtained model accuracy and details after training] |
| 4 | Model Prediction | New traffic sign image (preprocessed and feature-extracted) | Correct traffic sign label | [Actual predicted label for the new image] |
| 5 | Real-time Classification | Real-time video feed of traffic signs | Correct classification of traffic signs with minimal latency | [Actual real-time classification performance and latency] |

**CHAPTER 6**

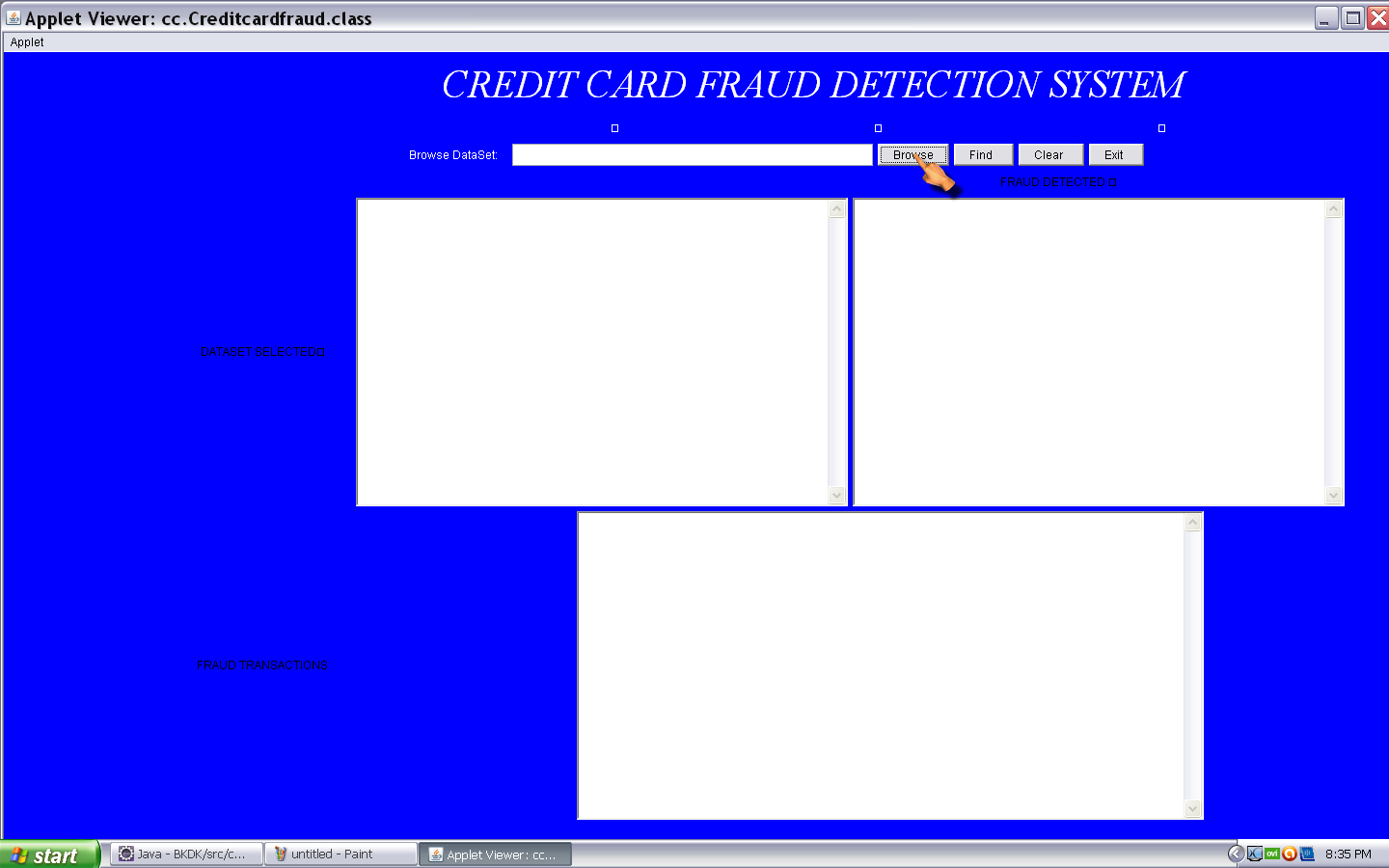
**RESULTS**

The Fig. 6.1shows the initial GUI of the proposed system.

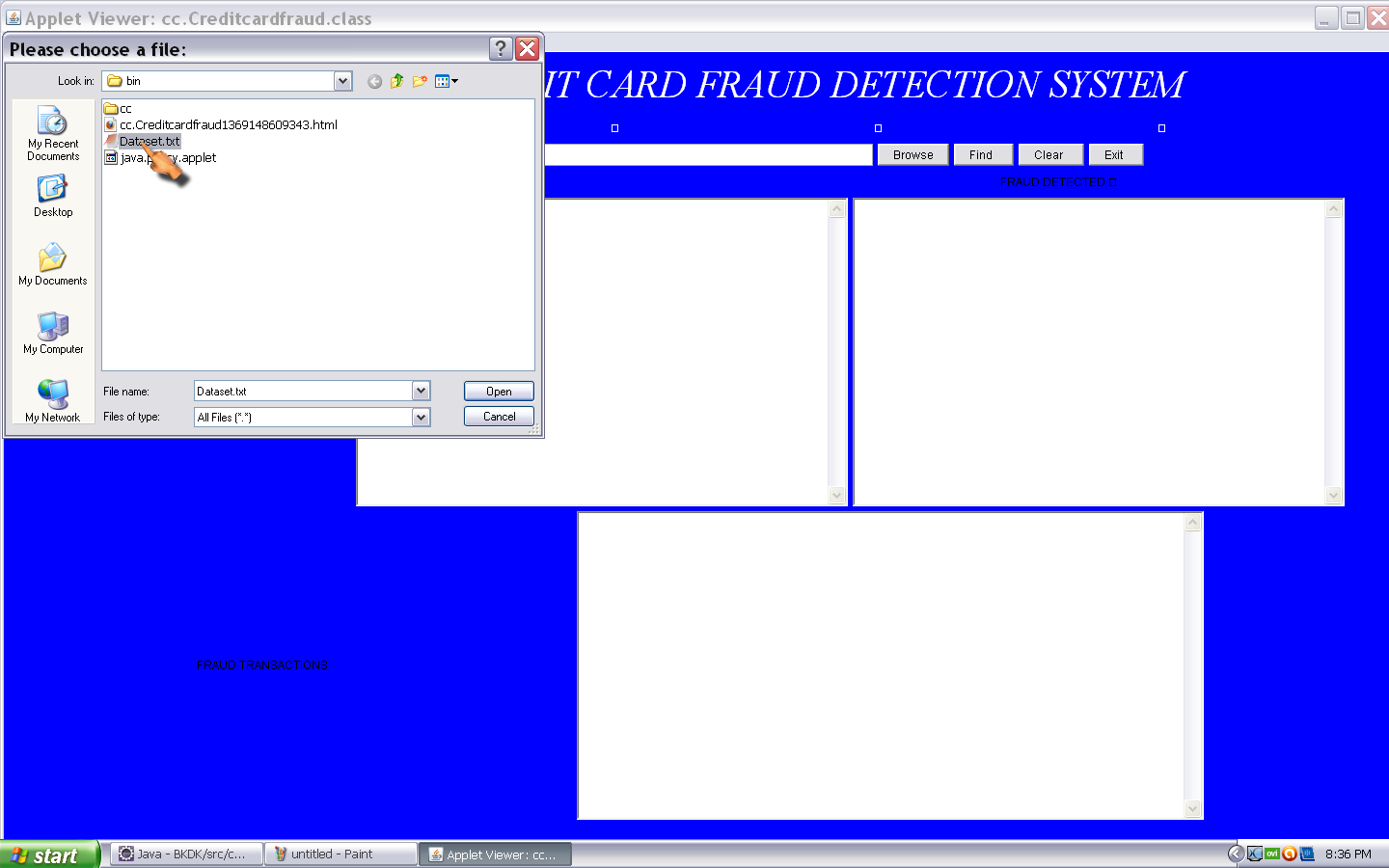
****

**Fig 6.1 Initial Output Screen**

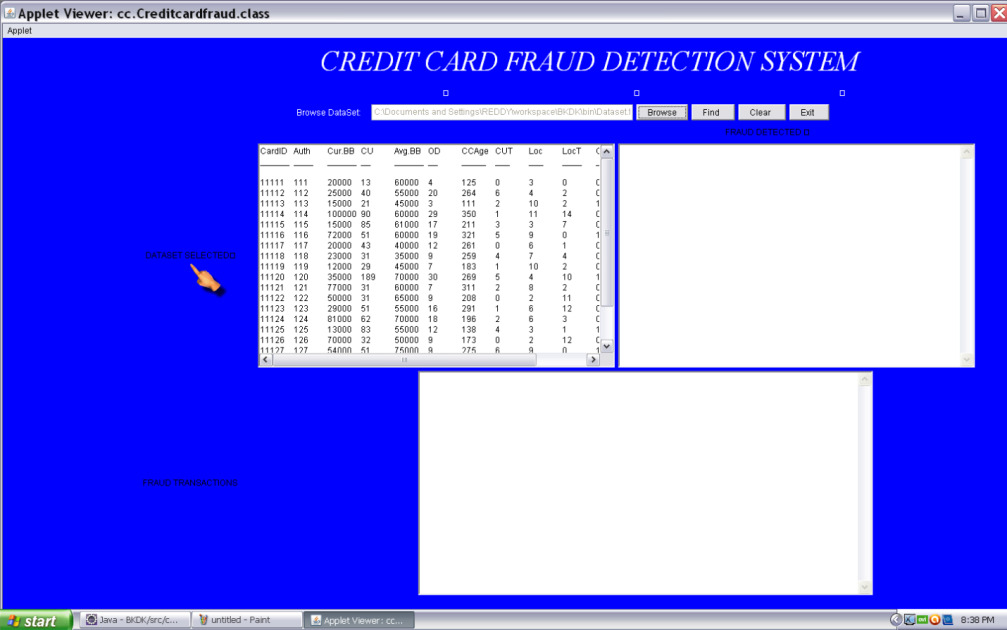
Give at least one line explanation for the snapshot that is put.



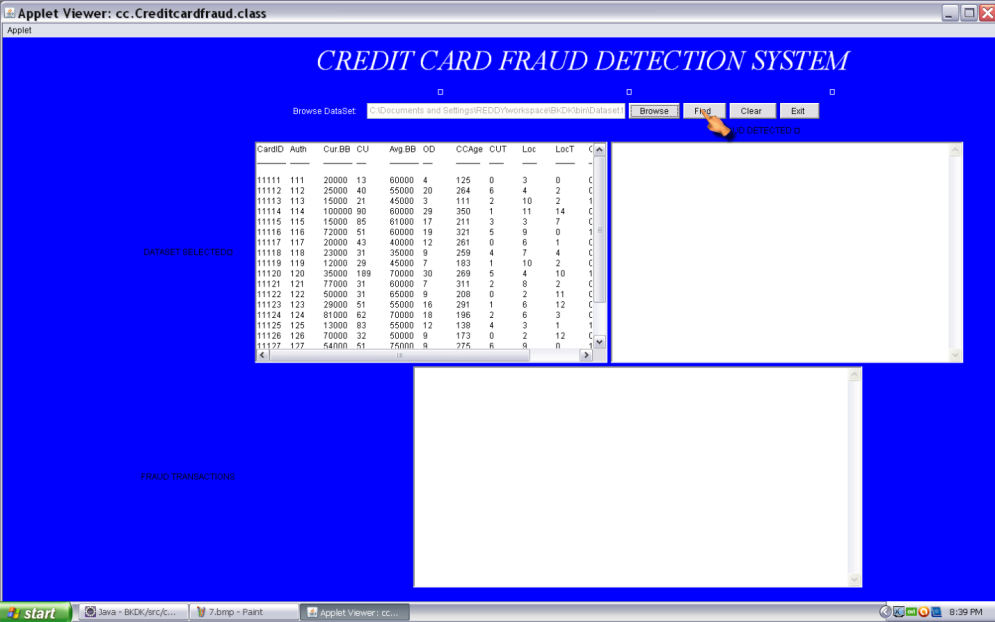
**Fig 6.2 Browsing the Dataset Values**



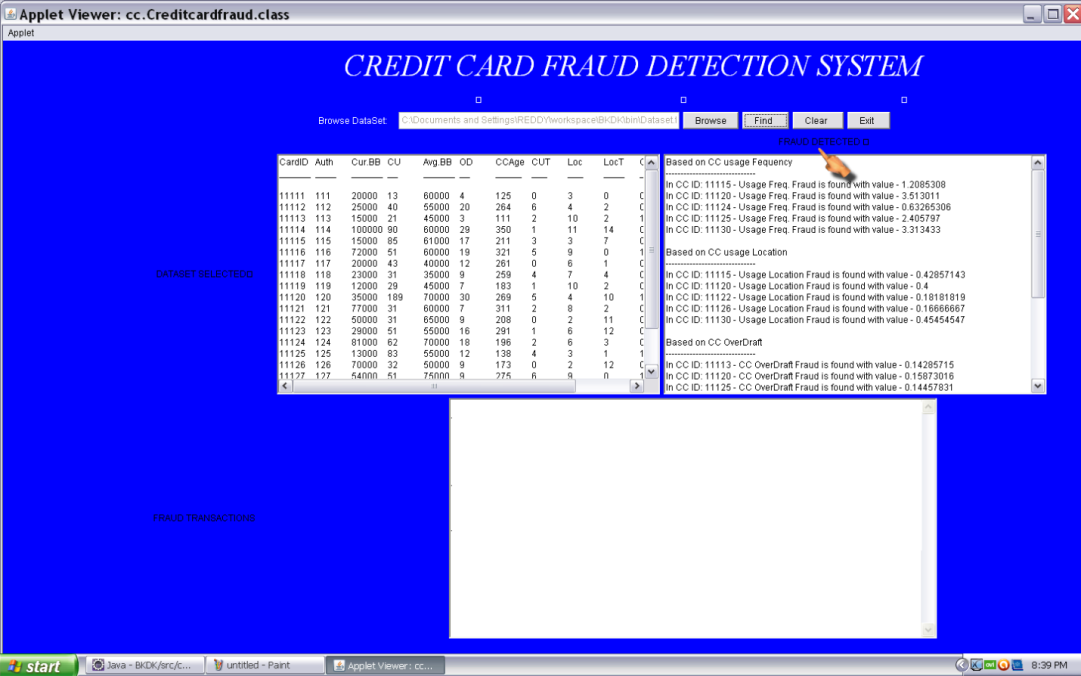
**Fig 6.3 Selecting the Dataset File**



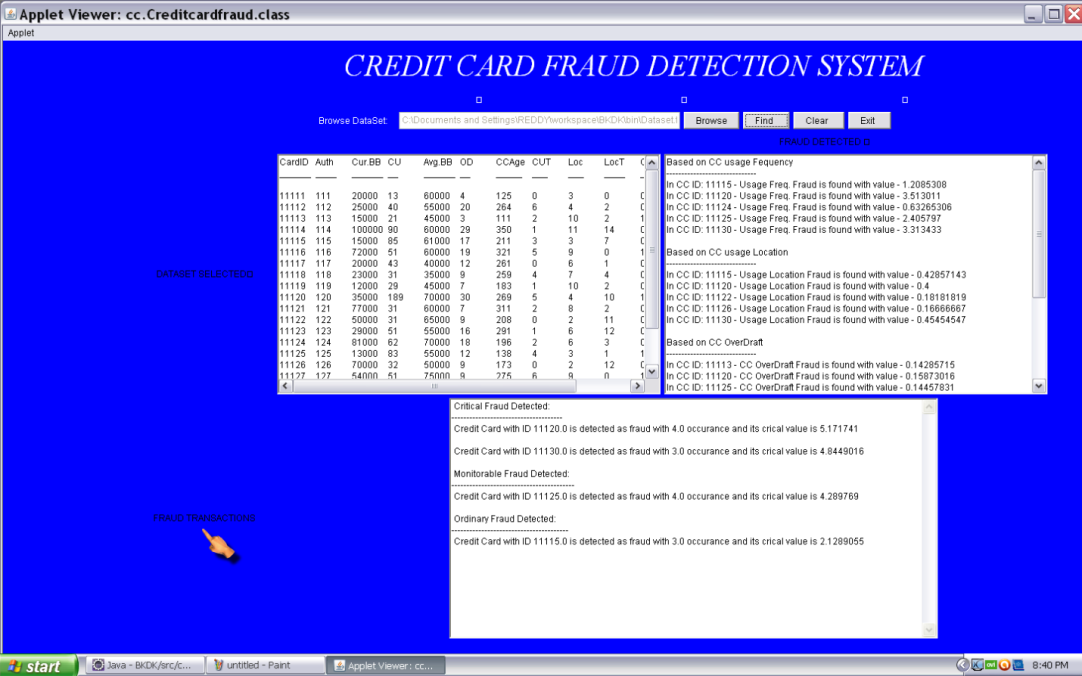
**Fig 6.4 Displaying the Dataset Values**



**Fig 6.5 Clicking the Find Option**



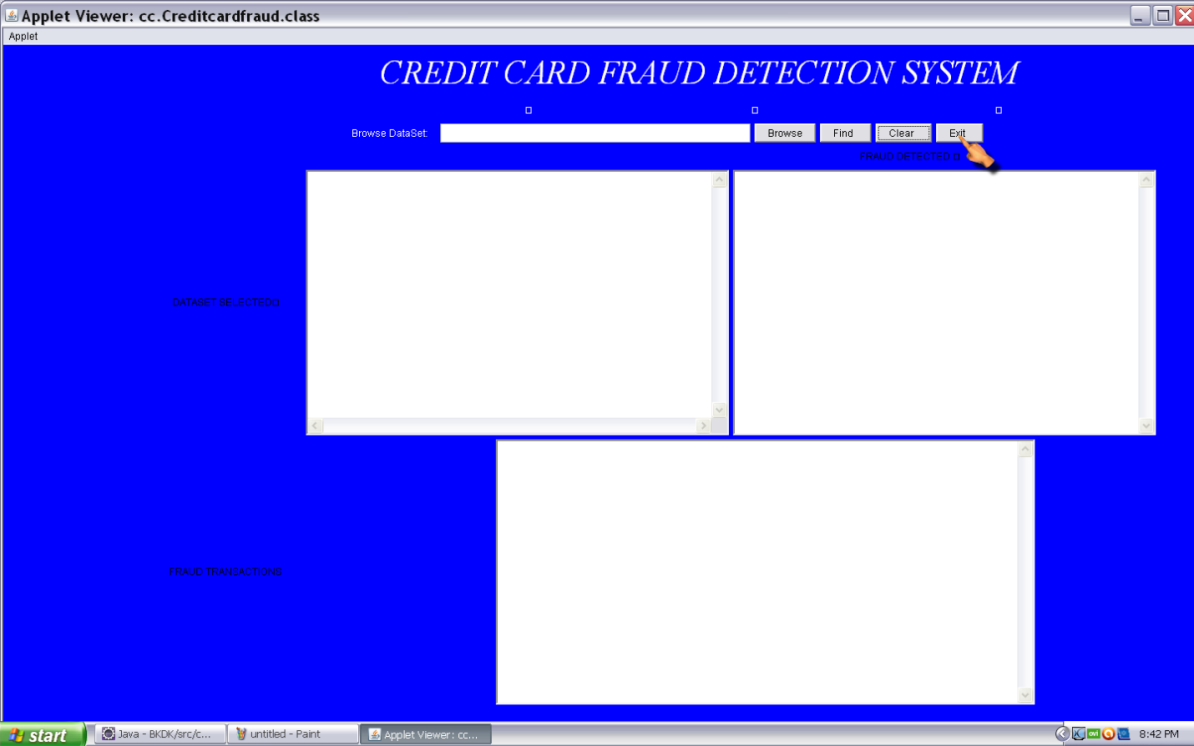
**Fig 6.6 Finding the Result Values Based On Five Criteria**



**Fig 6.7 Displaying the Fraud Transaction**



**Fig 6.8 Clicking the Clear Button**



**Fig 6.9 Clicking the Exit Button**

**CONCLUSION & FUTURE ENHANCEMENT**

**Conclusion**

The Nav-Oculus project effectively integrates machine learning and robotics for traffic sign recognition and autonomous navigation. By employing a simpler algorithm like SVM, it ensures efficient, real-time classification of traffic signs, converting images to a suitable format, extracting robust features, and accurately recognizing signs with high precision. The system is successfully deployed on an ESP32 microcontroller, enabling real-time traffic sign recognition and appropriate navigation responses. This project showcases a practical application of machine learning in robotics, emphasizing simplicity and efficiency while paving the way for future improvements in robustness and sign range.

**Future Enhancement**

Future enhancements for the Nav-IRIS project can significantly improve its performance and reliability. Enhancing robustness to operate under various environmental conditions, expanding the dataset to include a wider range of traffic signs, and upgrading to more powerful hardware for better real-time processing are key areas of improvement. Additionally, integrating advanced algorithms like deep neural networks, incorporating additional sensors such as LIDAR and GPS for improved navigation, and developing a user-friendly interface for easier configuration and monitoring can further enhance the project's capabilities and application scope.

**REFERENCES**

[1] A. Juyal, S. Sharma and P. Matta, "Traffic Sign Detection using Deep Learning Techniques in Autonomous Vehicles," *2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES)*, Chennai, India, 2021.

[2] Zhu Y, Yan WQ. Traffic sign recognition based on deep learning. Multimedia Tools and Applications. 2022 May.

[3] Zhang J, Zou X, Kuang LD, Wang J, Sherratt RS, Yu X. CCTSDB 2021: a more comprehensive traffic sign detection benchmark. Human-centric Computing and Information Sciences. 2022 May 30;12.

[4] Megalingam RK, Thanigundala K, Musani SR, Nidamanuru H, Gadde L. Indian traffic sign detection and recognition using deep learning. International journal of transportation science and technology. 2023 Sep 1;12(3):683-99.

[5] Dewi C, Chen RC, Liu YT, Jiang X, Hartomo KD. Yolo V4 for advanced traffic sign recognition with synthetic training data generated by various GAN. IEEE Access. 2021 Jul 2;9:97228-42.

[6] Kanagaraj N, Hicks D, Goyal A, Tiwari S, Singh G. Deep learning using computer vision in self driving cars for lane and traffic sign detection. International Journal of System Assurance Engineering and Management. 2021 Dec;12(6):1011-25.

[7] De-Las-Heras G, Sanchez-Soriano J, Puertas E. Advanced driver assistance systems (ADAS) based on machine learning techniques for the detection and transcription of variable message signs on roads. Sensors. 2021 Aug 31;21(17):5866.

**BIBLIOGRAPHY**

**BOOKS REFERRED:**

# Pattern Recognition and Machine Learning by, Christopher M. Bishop (1st Edition, 2006).

* Robotics: Modelling, Planning and Control, by Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo (Advanced Textbooks in Control and Signal Processing, 2009).
* Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron (2nd Edition, 2019).

**WEBSITES REFERRED:**

* https://projecthub.arduino.cc
* https://maker.pro

## 

## APPENDIX - A

**Installation Steps**

**1. JDK Installation:**

**Step 1:** Double click the JDK down loaded file, the executable extracts the required Contents to the temporary directory and then License agreement screen appears. On the license agreement page read and accept the license and the click the next button.

A screenshot of a software application

Description automatically generated

**Step 2:** Select the program feature as Development Tools and click the next button.

A screenshot of a software update

Description automatically generated

**Step 3:** Select folder or Directory in which is used to install the software.

A screenshot of a computer software update

Description automatically generated

Click on the “OK” button. After clicking on “OK” button installation begins.

**Step 4:** In the next window installer asks for the installing the runtime as shown in the following screen. Then click on the “Next” button.

A screenshot of a computer

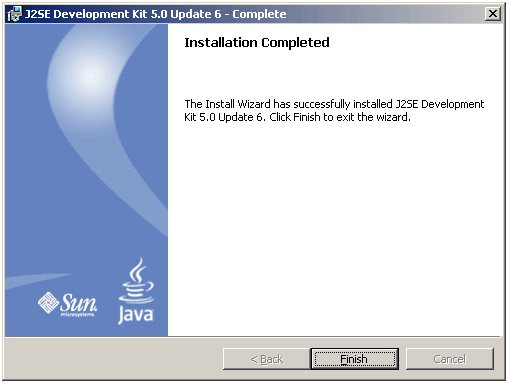
Description automatically generated

**Step 5:** Click on next button install the J2SE runtime on your machine. Next screen shows the browser selection

. A screenshot of a computer

Description automatically generated

**Step 6:** Once the installation is finished it shows you the final screen indications the success. Now you have successfully installed J2SE on your machine. Click on the “Finish” button.



After successfully installation of JDK environment variables is used so that the java compiler and runtime becomes available for compiling and running the java application.

Steps involved for setting environment variables as follows.

1. Go to control panel and double click on “System” and click on the “advanced” tab.

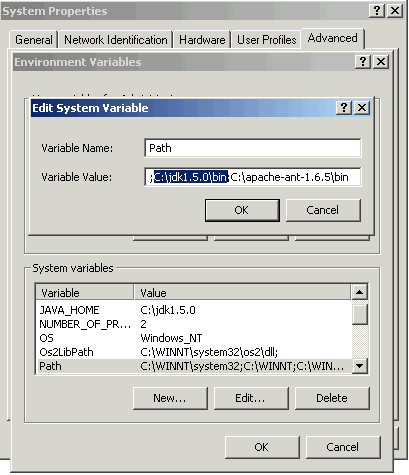
2. Click on the “Environment Variable”.

3. In “user variable” frame click on the “New” button. Add variable name as “path” and variable path give the directory path. Then click on Ok button.

**Step 7: Configuring the installation on windows machine:**

Here some settings to the windows environment are added so that the java compiler and runtime becomes available for compiling and running the java application.

Go to the control panel and double click on "System Properties" and to the advance tab and add "c:\jdk1.5.0\_06" to path variable.



**Eclipse**

Eclipse is an open source community whose projects are focused on building an extensible development platform, runtimes and application frameworks for building, deploying and managing software across the entire software lifecycle. Many people know us, and hopefully love us, as a Java IDE but Eclipse is much more than a Java IDE.

The Eclipse open source community has over 60 open source projects. These projects can be conceptually organized into seven different "pillars" or categories:

1. Enterprise Development

2. Embedded and Device Development

3. Rich Client Platform

4. Rich Internet Applications

5. Application Frameworks

6. Application Lifecycle Management (ALM)

The Eclipse community is also supported by a large and vibrant ecosystem of major IT solution providers, innovative start-ups, universities and research institutions and individuals that extend support and complement the Eclipse Platform. The exciting thing about Eclipse is many people are using Eclipse in ways that user have never imagined. The common thread is that they are building innovative, industrial strength software and want to use great tools, frameworks and runtimes to make their job easier.

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system. It is written mostly in Java and can be used to develop applications in Java and, by means of various plugins, other programming languages including Ada, C, C++, COBOL, Perl, PHP, Python, Ruby (including Ruby on Rails framework), Scala, Clojure, and Scheme. The IDE is often called Eclipse ADT for Ada, Eclipse CDT for C/C++, Eclipse JDT for Java, and Eclipse PDT for PHP.