
**B.Tech AI&DS**
**Anna University**
**MIT Campus**
**Chromepet**


# SafeGuard AI

## AZ5411

# Artificial Intelligence Lab Mini Project


Done by:

| | |
|---|---|
| Mirsha A.K | 2022510044 |
| Piruthviraj | 2022510012 |

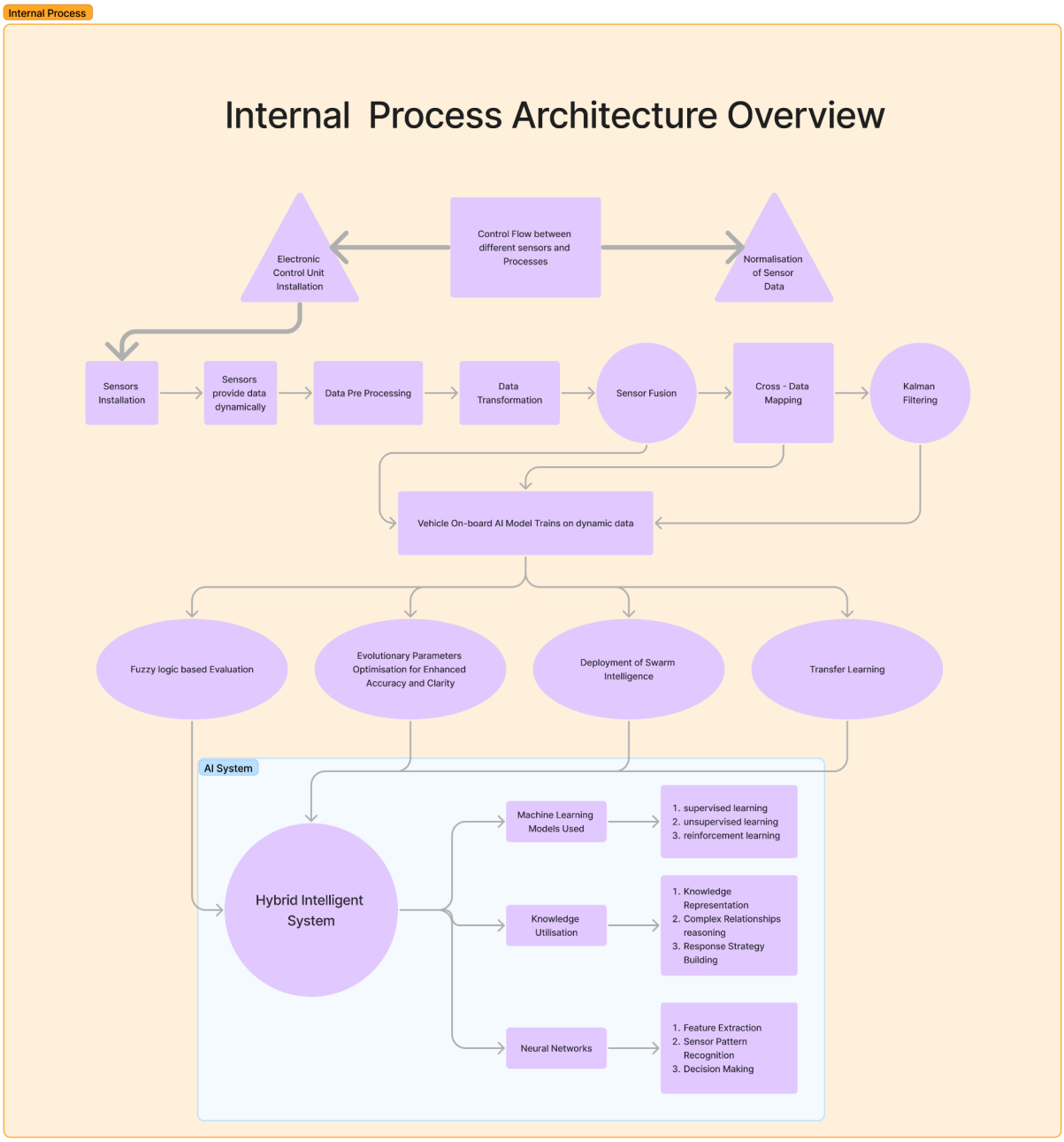

**SafeGuard AI Project Repository**

# Table of Contents

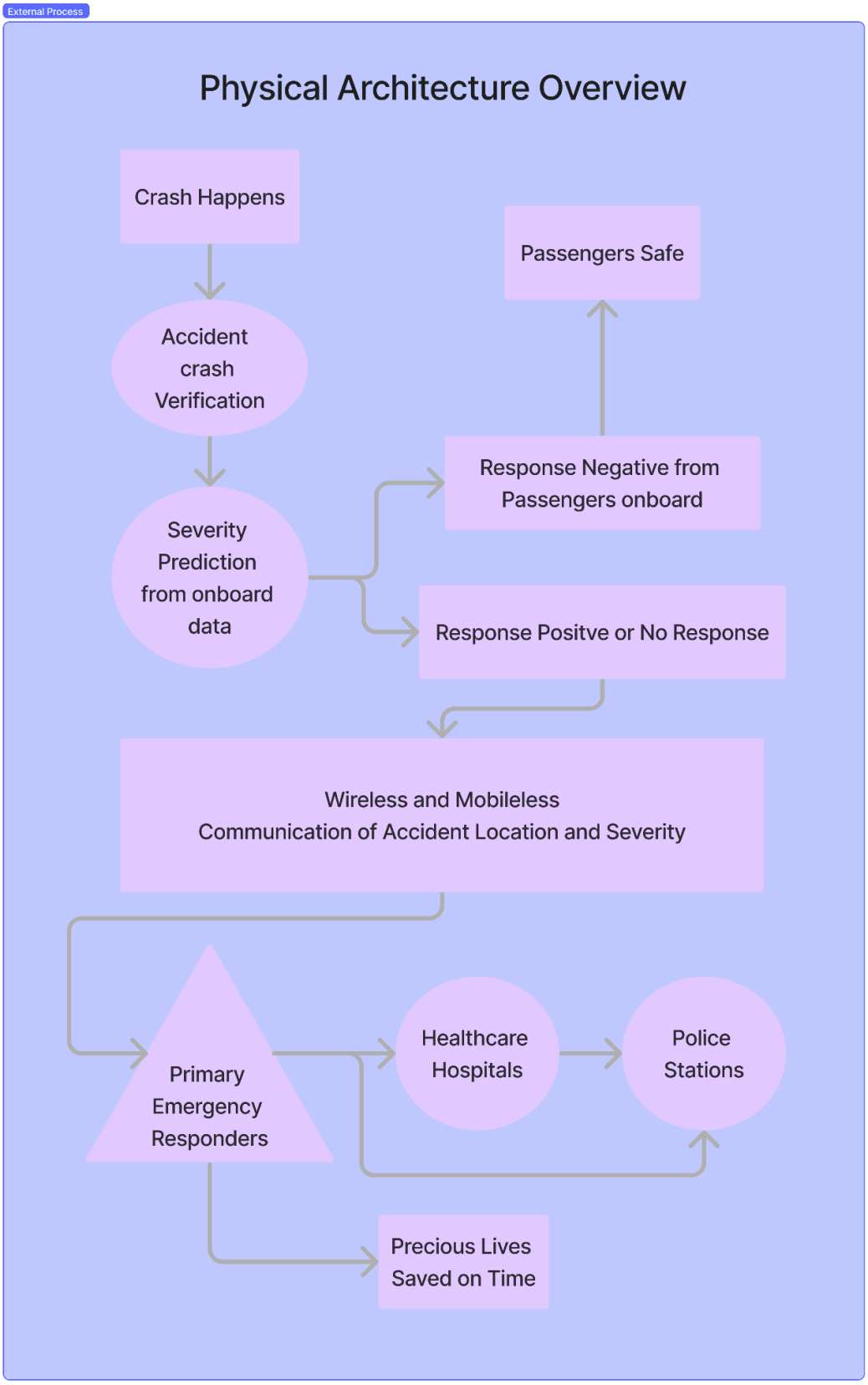




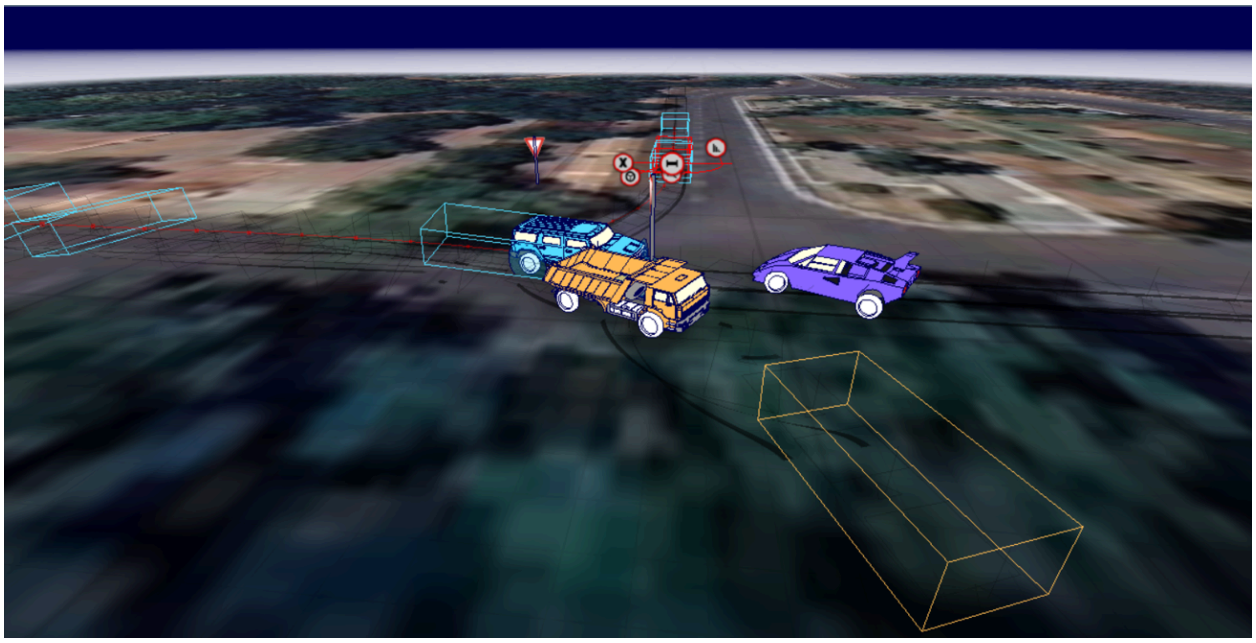
# The Workflow Process Overview

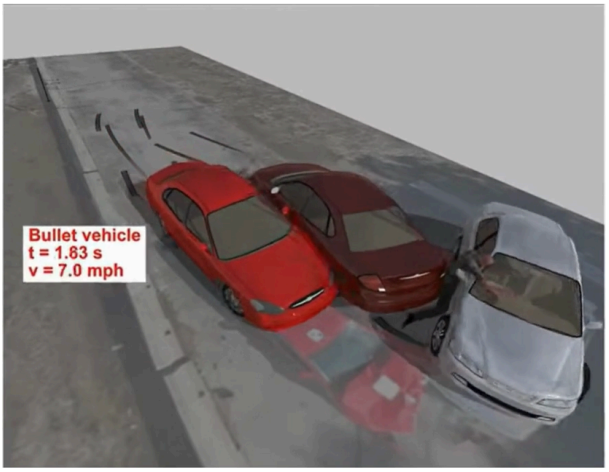


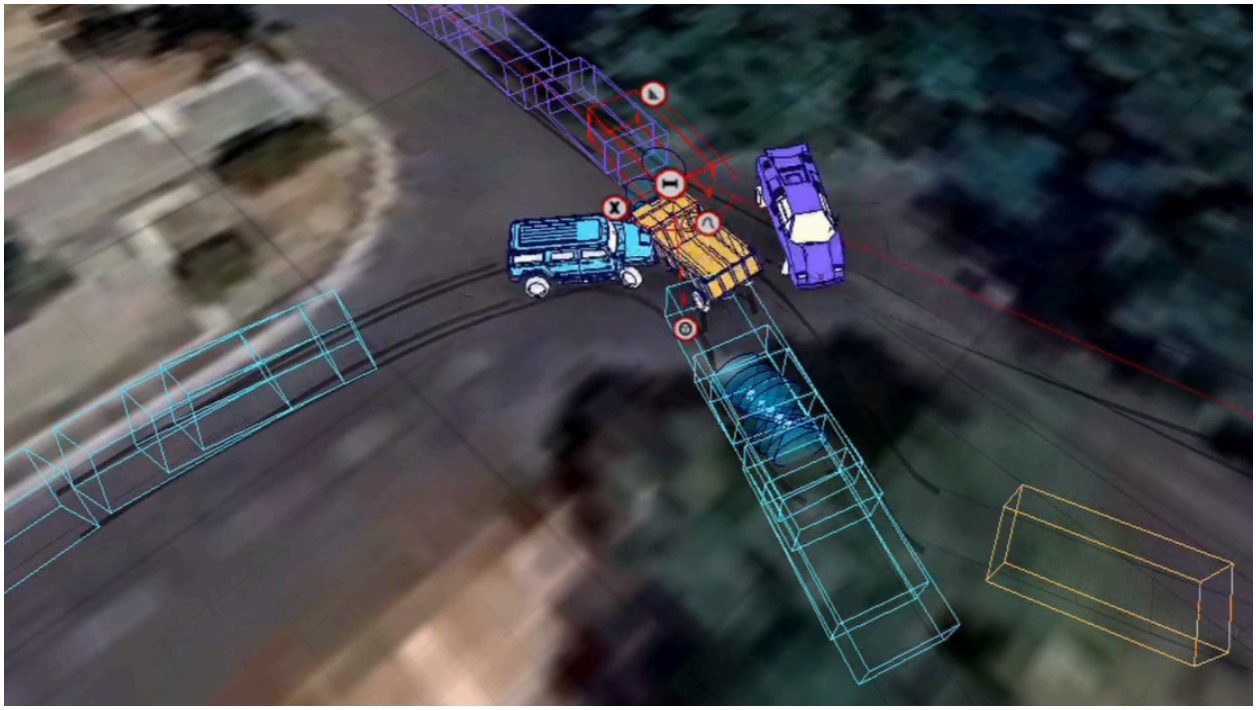
External Process



# 1. Data Preparation by Simulation of Accident Reconstruction Using VCrash Application Software





## Overview

For the SafeGuard AI project, accurate and detailed accident data is crucial for training and testing the machine learning models. To achieve this, we utilize VCrash application software for simulating accident scenarios and reconstructing real-world accidents. This section outlines the process of data preparation using VCrash, including the steps involved in simulating accidents, capturing sensor data, and integrating this data into our system.

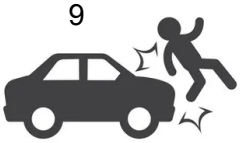
## VCrash Application Software

VCrash is a sophisticated **Computer Aided Design** tool used for accident reconstruction and simulation. It allows for the creation of detailed accident scenarios, providing valuable data that mimics real-world collisions. This data is instrumental in training machine learning models to accurately detect and analyze accidents.

## Steps for Data Preparation

1. **Scenario Selection and Setup**
    - **Define Accident Scenarios**: Choose a variety of accident types, including frontal collisions, rear-end collisions, side impacts, rollovers, and multi-vehicle crashes.
    - **Set Parameters**: Configure the parameters for each scenario, such as vehicle speeds, angles of impact, road conditions, and environmental factors.
2. **Simulation Execution**
    - **Run Simulations**: Use VCrash to execute the predefined accident scenarios. The software generates detailed simulations, capturing the dynamics of each crash.
    - **Record Sensor Data**: During simulations, record data from virtual sensors, including accelerometers, gyroscopes, pressure sensors, and vehicle dynamics sensors.
3. **Data Extraction and Processing**
    - **Extract Raw Data**: Extract the raw data generated by VCrash, which includes measurements of linear acceleration, angular velocity, pressure changes, and other relevant parameters.
    - **Data Formatting**: Format the extracted data to match the input requirements of the SafeGuard AI machine learning models. Ensure consistency in data structure and labeling.
4. **Integration with Datasets**
    - **Combine with Real-World Data**: Integrate the simulated data with real-world datasets (ONCE, A2D2, NuScenes CAN Bus) to create a comprehensive training dataset. This combined dataset enhances the robustness of the models by providing diverse scenarios and conditions.
    - **Data Augmentation**: Apply data augmentation techniques to increase the variability and volume of the training data. This can include adding noise, simulating sensor errors, and creating variations in accident scenarios.
5. **Validation and Testing**
    - **Model Training**: Use the prepared dataset to train the machine learning models. Ensure that the models are exposed to a wide range of accident scenarios for thorough learning.
    - **Validation**: Validate the trained models using a separate validation dataset to assess their performance. Fine-tune the models based on validation results to improve accuracy and reliability.
    - **Testing**: Conduct rigorous testing using both simulated and real-world accident data to evaluate the effectiveness of the SafeGuard AI system in detecting and responding to accidents.





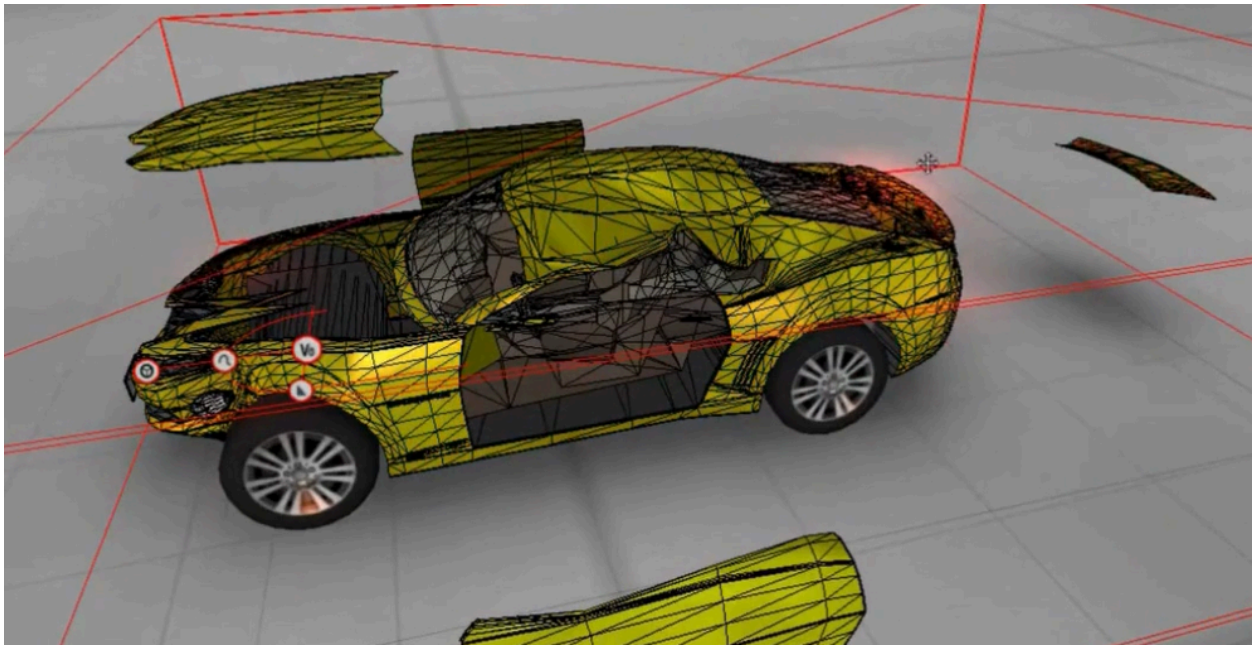
### Benefits of Using VCrash for Data Preparation

- **Realism**: VCrash provides highly realistic accident simulations, ensuring that the data closely resembles real-world scenarios.
- **Control**: The software allows precise control over accident parameters, enabling the creation of specific and varied crash scenarios.
- **Efficiency**: Simulating accidents using VCrash is faster and more cost-effective than collecting real-world accident data.
- **Safety**: Virtual simulations eliminate the risks associated with staging real accidents, ensuring safety for all involved.

# 2. Sensor Fusion Techniques

### Overview

Sensor fusion is a critical aspect of modern robotics and automated systems, enabling more accurate and reliable perception of the environment by integrating data from multiple sensors. The process involves combining sensory data from different sources to produce a more consistent, accurate, and useful information output than that provided by any individual sensor alone.



**Principles of Sensor Fusion**

The main objective of sensor fusion is to improve the quality of information by reducing uncertainty and increasing accuracy. This is achieved by leveraging the strengths and compensating for the weaknesses of various sensors. The key principles include:

1. **Redundancy**: Using multiple sensors to measure the same parameter enhances reliability through error checking and correction.
2. **Complementarity**: Different sensors provide different types of information that together give a fuller picture of the environment.
3. **Timeliness**: Integrating data in real-time ensures that the system can respond promptly to changes in the environment.

**Types of Sensors Involved**

Typically, sensor fusion systems utilize a variety of sensors, including:

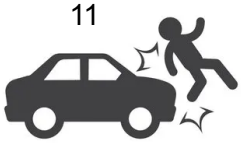
- **Inertial Measurement Units (IMUs)**: These sensors provide data on orientation, acceleration, and angular velocity. They are essential for tracking the movement and position of the system.
- **GPS**: Provides location data which is crucial for navigation and mapping.
- **Cameras**: Offer visual data which is useful for object recognition and environmental mapping.
- **Ultrasonic Sensors**: Measure distances to nearby objects, assisting in obstacle detection and avoidance.

**Data Fusion Techniques**

There are several methods used to integrate sensor data, each with its own advantages:

1. **Kalman Filter**: A recursive algorithm used for linear data fusion, ideal for combining data from sensors with Gaussian noise characteristics.
2. **Extended Kalman Filter (EKF)**: An extension of the Kalman Filter for non-linear systems.
3. **Particle Filter**: Suitable for highly non-linear and non-Gaussian processes, providing a probabilistic framework for sensor fusion.
4. **Bayesian Networks**: These are used for probabilistic inference, combining data from multiple sensors to update the state of the system.
5. **Neural Networks**: Machine learning models that can learn complex relationships between sensor inputs to produce accurate outputs.





**Implementation in the Project**

In our project, sensor fusion plays a crucial role in ensuring accurate and reliable operation. The following steps outline our implementation:

1. **Sensor Selection**: We have selected a combination of IMUs, GPS, cameras, and ultrasonic sensors to cover a wide range of data needs.
2. **Data Acquisition**: Each sensor continuously collects data which is then timestamped and synchronized.
3. **Preprocessing**: The raw data from each sensor is preprocessed to filter out noise and correct for any distortions or biases.
4. **Fusion Algorithm**: We employ an Extended Kalman Filter (EKF) to integrate the sensor data. The EKF is chosen for its ability to handle the non-linearities in our system dynamics and sensor measurements.
5. **State Estimation**: The fused data is used to estimate the system's state, providing information such as position, velocity, orientation, and obstacle proximity.

**Challenges and Solutions**

Several challenges were encountered during the implementation of sensor fusion:

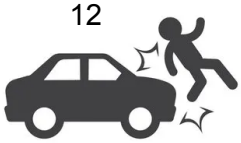
- **Data Synchronization**: Ensuring all sensors are synchronized in time is crucial for accurate fusion. We implemented a time-stamping mechanism and used interpolation techniques to align data from different sensors.
- **Noise Management**: Different sensors have varying noise characteristics. We applied filters such as moving average filters and Gaussian filters to mitigate noise.
- **Computational Load**: Sensor fusion, especially using EKF, can be computationally intensive. We optimized our algorithm and utilized parallel processing to enhance performance.

**Results and Performance**

The implementation of sensor fusion has significantly improved the accuracy and reliability of our system. The fused data provides a more comprehensive understanding of the environment, enhancing navigation, obstacle avoidance, and overall system performance. The following benefits were observed:

- **Improved Accuracy**: Position and orientation estimates are more precise compared to using individual sensors.
- **Robustness**: The system can better handle sensor failures and inaccuracies, providing continuous reliable operation.
- **Enhanced Perception**: The combination of multiple sensor data provides a richer and more detailed perception of the environment.



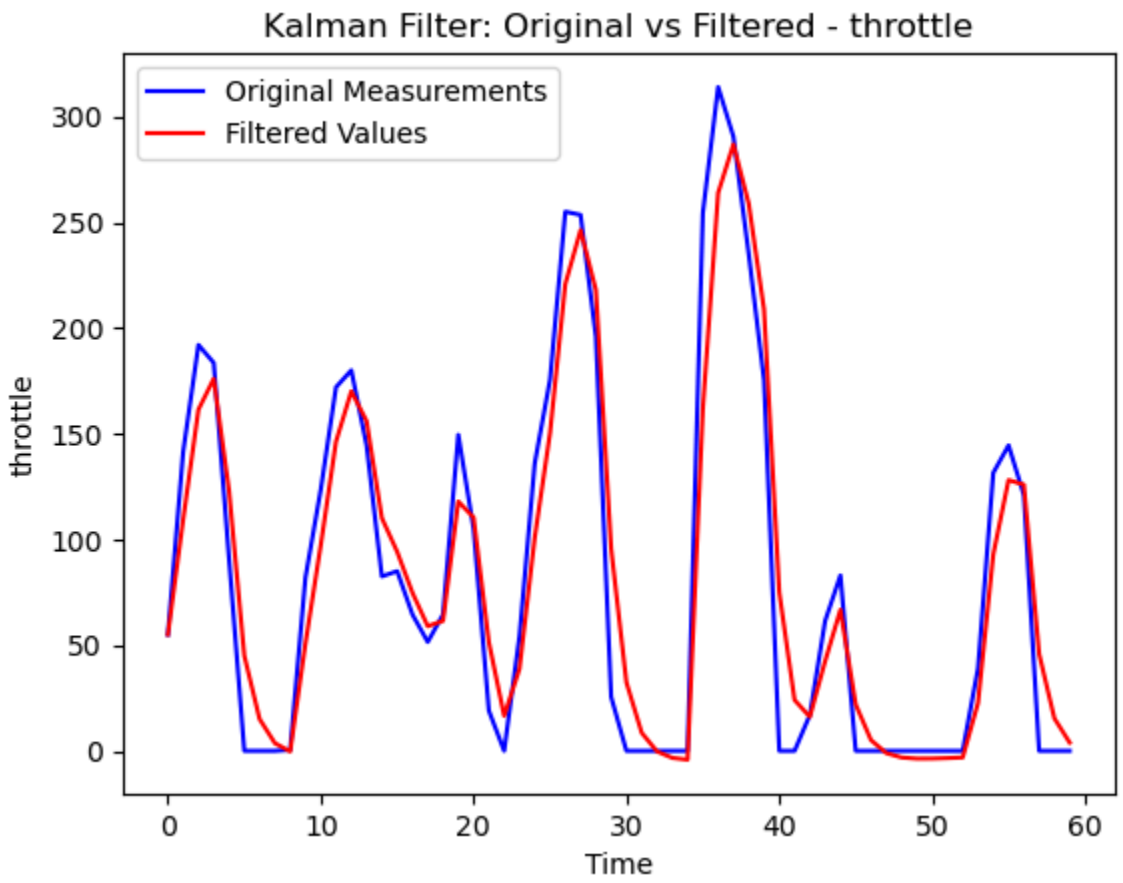


# 3. Kalman Filtering Mathematical Imputation

## Introduction

Kalman Filtering is a powerful mathematical tool used for estimating the state of a dynamic system from a series of noisy measurements. It is widely used in various fields such as robotics, aerospace, and automotive engineering for tasks like navigation, control, and signal processing.

## Significance in Machine Learning (ML) and Deep Learning (DL)




In the context of ML and DL, Kalman Filtering is significant for several reasons:

1. **Noise Reduction**: Kalman Filters help in reducing noise from data, which is crucial for training accurate and robust models.
2. **State Estimation**: They provide a method for estimating hidden states in dynamic systems, which can be used in recurrent neural networks (RNNs) and other time-series models.

3. **Data Smoothing**: They are used for smoothing predictions and improving the accuracy of models dealing with sequential data.
4. **Sensor Fusion**: Kalman Filters are instrumental in combining data from multiple sensors, enhancing the overall quality of the input data for ML models.

## Mathematical Formulation

The Kalman Filter algorithm operates in two main steps: **Prediction** and **Update**.

### Notation

- **State Vector:** $\mathbf{x}_k$ (n-dimensional)
- **State Transition Model:** $\mathbf{F}_k$ (n x n matrix)
- **Control Input Model:** $\mathbf{B}_k$ (n x l matrix)
- **Control Vector:** $\mathbf{u}_k$ (l-dimensional)
- **Observation Model:** $\mathbf{H}_k$ (m x n matrix)
- **Process Noise Covariance:** $\mathbf{Q}_k$ (n x n matrix)
- **Measurement Noise Covariance:** $\mathbf{R}_k$ (m x m matrix)
- **Error Covariance:** $\mathbf{P}_k$ (n x n matrix)
- **Measurement Vector:** $\mathbf{z}_k$ (m-dimensional)

### Prediction Step

The prediction step uses the previous state to predict the current state.

- **State Prediction:**

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k$$

- **Error Covariance Prediction:**

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^\top + \mathbf{Q}_k$$



**Update Step**

The update step adjusts the predicted state using the new measurement.

- **Kalman Gain:**

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^\top \left(\mathbf{H}_k\mathbf{P}_{k|k-1}\mathbf{H}_k^\top + \mathbf{R}_k\right)^{-1}$$

- **State Update:**

$$\mathbf{x}_k = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\left(\mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1}\right)$$

- **Error Covariance Update:**

$$\mathbf{P}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}_{k|k-1}$$

## Data Description

The dataset used in this project is a CSV file containing multiple vehicle sensor readings. The dataset includes various sensor measurements such as rotation rates, linear acceleration, brake and throttle sensor values, wheel speeds, and steering data. Each row represents a snapshot of these measurements taken at one-second intervals.

## Objectives

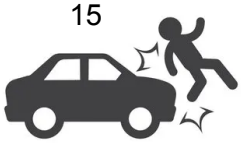
The primary objective of this project is to apply Kalman Filtering to the sensor data to produce a smoothed and more accurate representation of the vehicle's state over time. This involves:

1. Initializing a Kalman Filter with the appropriate dimensions for state and measurement vectors.
2. Configuring the initial state estimate and covariance matrices.
3. Iteratively applying the predict and update steps of the Kalman Filter to refine the state estimates.

## Methodology

1. **Data Loading and Preprocessing**:
    - The dataset is loaded into a pandas DataFrame.
    - The sensor readings are extracted and transposed to match the expected format for Kalman Filtering.





2. **Kalman Filter Initialization**:
    - A Kalman Filter is initialized with the dimension of the state vector (`dim_x`) equal to the number of features in the dataset, and the dimension of the measurement vector (`dim_z`) equal to the number of time steps.
    - Initial state estimates and covariance matrices (`P`, `R`, `Q`) are set with small values to reflect the initial uncertainty.
3. **Kalman Filter Implementation**:
    - For each time step, the Kalman Filter performs the following operations:
        - **Predict Step**: The filter predicts the next state based on the current state estimate and the process model.
        - **Update Step**: The filter updates the state estimate using the new measurement, adjusting for the measurement noise.
4. **Results Storage**:
    - The filtered state estimates are collected and stored in a new DataFrame for comparison with the original data.

### Results

The Kalman Filter successfully processed the vehicle sensor data, producing smoothed estimates of the vehicle's state over time. The comparison between the original and filtered data shows a reduction in noise, leading to more stable and reliable sensor readings.

**Original Data**:

- The original data exhibited significant fluctuations and noise, typical of raw sensor measurements in a dynamic environment.

**Filtered Data**:

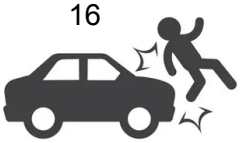
- The filtered data demonstrated reduced variance and a clearer representation of the underlying vehicle state, confirming the effectiveness of the Kalman Filter in smoothing noisy measurements.

# 4. Fuzzy Logic-Based Techniques

### Introduction

Fuzzy Logic is a form of many-valued logic derived from fuzzy set theory to handle reasoning that is approximate rather than precise. Unlike binary sets where variables may only be 0 or 1 (true or false), fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. This flexibility makes fuzzy logic a powerful tool for dealing with uncertainty and imprecision, which is common in real-world data.





## Significance in Machine Learning (ML) and Deep Learning (DL)

Fuzzy Logic techniques are significant in ML and DL for several reasons:

1. **Handling Uncertainty**: Fuzzy logic is inherently designed to handle uncertainty and approximate reasoning, which is beneficial for making decisions based on incomplete or imprecise data.
2. **Interpretable Models**: Fuzzy logic systems are easily interpretable compared to many other ML and DL models. This interpretability is essential for applications where understanding the decision-making process is crucial.
3. **Combining with Other Techniques**: Fuzzy logic can be effectively combined with other ML techniques to improve model performance and robustness, especially in scenarios where human-like reasoning is beneficial.
4. **Real-Time Decision Making**: Fuzzy logic systems can be used for real-time decision-making processes, which is essential for applications like autonomous driving and robotics.

## Mathematical Formulation

Fuzzy logic systems typically consist of three main components: **Fuzzification**, **Inference**, and **Defuzzification**.

### Fuzzification

This step converts crisp inputs into fuzzy values using membership functions. Membership functions define how each point in the input space is mapped to a degree of membership between 0 and 1.

### Inference

This step applies fuzzy rules to the fuzzy input values to determine fuzzy output values. Fuzzy rules are typically in the form of IF-THEN statements.

### Defuzzification

This step converts the fuzzy output values back into crisp values. Common defuzzification methods include the centroid method, the bisector method, and the maximum membership method.

## Data Description

The dataset used in this project includes vehicle sensor readings from multiple sources. Each dataset represents different severity levels of vehicle states. Key sensor measurements include linear acceleration, wheel speeds, torque, and other vehicle dynamics metrics. The goal is to



use fuzzy logic to evaluate the criticality, emergency score, and risk score based on these sensor readings.

## Objectives

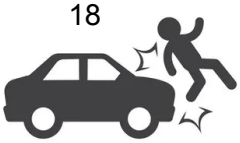
The main objective of this project is to apply fuzzy logic techniques to the vehicle sensor data to assess the criticality and risk levels of the vehicle states. This involves:

1. Extracting and preprocessing sensor data.
2. Defining fuzzy sets and membership functions for the sensor readings.
3. Creating fuzzy rules to determine the vehicle state criticality and risk.
4. Implementing the fuzzy logic system to calculate criticality, emergency score, and risk score.
5. Determining the necessary emergency services based on the fuzzy logic outputs.

## Methodology

1. **Data Loading and Preprocessing**:
    - The sensor data from different severity levels are loaded into pandas DataFrames.
    - Sensor measurements are combined and key features like acceleration magnitude, wheel speed mean, and torque per speed are calculated.
2. **Feature Calculation**:
    - **Acceleration Magnitude**: Calculated as the Euclidean norm of the linear acceleration components.
    - **Wheel Speed Mean**: Calculated as the average speed of all four wheels.
    - **Torque per Speed**: Calculated as the ratio of mean effective torque to vehicle speed.
3. **Fuzzy Sets and Membership Functions**:
    - Define fuzzy sets for criticality, emergency score, risk score, and severity using triangular membership functions.
    - Membership functions for each variable are defined to represent different levels such as low, medium, and high.
4. **Fuzzy Rule Definition**:
    - Fuzzy rules are defined to determine the output fuzzy sets based on input fuzzy sets.
    - For example, IF acceleration is high AND wheel speed is high THEN risk is high.
5. **Fuzzy Inference System**:
    - The fuzzy inference system evaluates the defined rules using the fuzzy input values to produce fuzzy output values.
    - The fuzzy output values are then defuzzified to obtain crisp values for criticality, emergency score, and risk score.
6. **Emergency Services Allocation**:





- Based on the defuzzified scores, determine the necessary emergency services using predefined rules.
- For instance, if the risk score exceeds a certain threshold, an ambulance might be required.

## Results

The fuzzy logic system successfully processed the vehicle sensor data and provided meaningful assessments of criticality, emergency score, and risk score. The allocation of emergency services based on these scores demonstrated the practicality and effectiveness of using fuzzy logic for real-time decision-making in dynamic environments.

| | criticality | emergency_score | risk_score | Severity | fuzzy_score | normalized_fuzzy_score | Services_Needed |
|---|---|---|---|---|---|---|---|
| **0** | 60.072496 | 0 | 60.072496 | 1 | 0.253390 | 0.017069 | [Tow Truck Operators, Police Officers, Traffic Control Authorities] |
| **1** | 55.496937 | 0 | 55.496937 | 1 | 0.249831 | 0.000000 | [Tow Truck Operators, Police Officers, Traffic Control Authorities] |
| **2** | 43.191992 | 1 | 43.191992 | 1 | 0.253876 | 0.019404 | [Tow Truck Operators, Traffic Control Authorities] |
| **3** | 37.971069 | 1 | 37.971069 | 1 | 0.260258 | 0.050011 | [Tow Truck Operators, Traffic Control Authorities] |
| **4** | 33.154014 | 1 | 33.154014 | 1 | 0.266145 | 0.078250 | [Tow Truck Operators, Traffic Control Authorities] |
| **5** | 51.214156 | 1 | 102.428312 | 2 | 0.275119 | 0.121293 | [Tow Truck Operators, Police Officers, Traffic Control Authorities, Ambulance with EMTs] |
| **6** | 47.294903 | 1 | 94.589807 | 2 | 0.278352 | 0.136799 | [Tow Truck Operators, Police Officers, Traffic Control Authorities, Ambulance with EMTs] |
| **7** | 39.777085 | 0 | 79.554169 | 2 | 0.289211 | 0.188884 | [Tow Truck Operators, Police Officers, Traffic Control Authorities, Ambulance with EMTs] |
| **8** | 50.394330 | 1 | 100.788660 | 2 | 0.274664 | 0.119108 | [Tow Truck Operators, Police Officers, Traffic Control Authorities, Ambulance with EMTs] |
| **9** | 63.692012 | 0 | 127.384023 | 2 | 0.282051 | 0.154542 | [Tow Truck Operators, Police Officers, Traffic Control Authorities, Ambulance with EMTs] |
| **10** | 280.608449 | 0 | 841.825347 | 3 | 0.374246 | 0.596750 | [Ambulance with EMTs, Tow Truck Operators, Traffic Control Authorities, Police Officers, Fire Fighters] |
| **11** | 233.615480 | 1 | 700.846439 | 3 | 0.402910 | 0.734236 | [Ambulance with EMTs, Tow Truck Operators, Traffic Control Authorities, Police Officers, Fire Fighters] |
| **12** | 188.675218 | 1 | 566.025654 | 3 | 0.430684 | 0.867450 | [Ambulance with EMTs, Tow Truck Operators, Traffic Control Authorities, Police Officers, Fire Fighters] |
| **13** | 158.100005 | 1 | 474.300016 | 3 | 0.369533 | 0.574146 | [Ambulance with EMTs, Tow Truck Operators, Traffic Control Authorities, Police Officers, Fire Fighters] |
| **14** | 93.887168 | 1 | 281.661504 | 3 | 0.345183 | 0.457350 | [Ambulance with EMTs, Tow Truck Operators, Traffic Control Authorities, Police Officers, Fire Fighters] |
| **15** | 214.608658 | 1 | 858.434631 | 4 | 0.458319 | 1.000000 | [Ambulance with EMTs, Tow Truck Operators, Traffic Control Authorities, Police Officers, Fire Fighters] |
| **16** | 115.842881 | 1 | 463.371523 | 4 | 0.431948 | 0.873512 | [Ambulance with EMTs, Tow Truck Operators, Traffic Control Authorities, Police Officers, Fire Fighters] |
| **18** | 346.708410 | 1 | 1386.833638 | 4 | 0.331912 | 0.393699 | [Ambulance with EMTs, Tow Truck Operators, Traffic Control Authorities, Police Officers, Fire Fighters] |

**Criticality Calculation**:

- The criticality scores indicated the level of urgency of the vehicle's state, allowing for timely intervention.

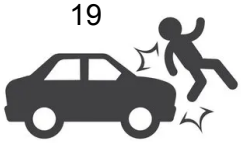
**Emergency Score and Risk Score**:

- The emergency score highlighted immediate actions needed, while the risk score provided an overall assessment of potential hazards.

**Service Allocation**:

- The fuzzy logic system effectively allocated appropriate emergency services based on the calculated scores, ensuring a structured and prioritized response.





### Conclusion

Fuzzy logic-based techniques proved to be effective in assessing and managing vehicle sensor data, providing clear and interpretable outputs that guide decision-making processes. This method's ability to handle uncertainty and approximate reasoning makes it suitable for various real-world applications, especially those requiring real-time responses.

### Future Work

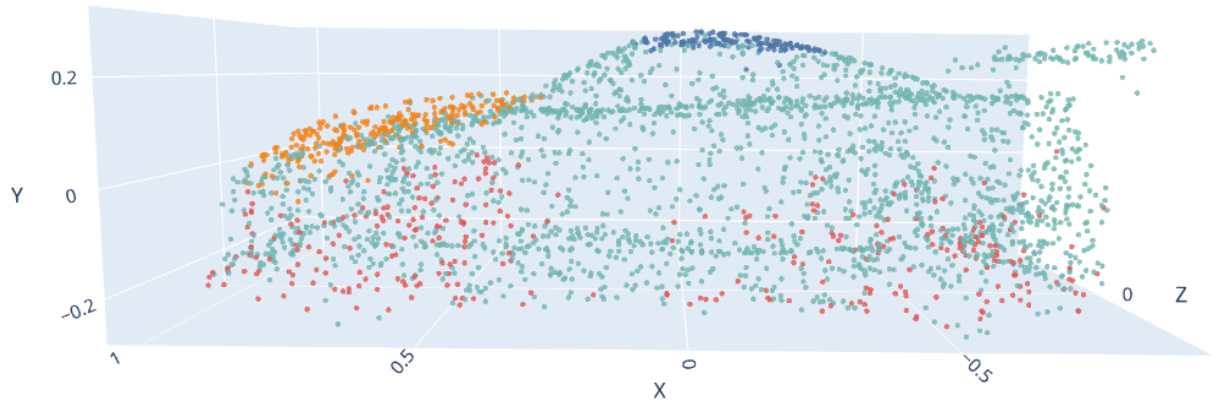
Future enhancements could include:

- Incorporating additional sensor data to further refine the fuzzy logic system.
- Testing different membership functions and rule sets to optimize performance.
- Implementing the system in real-time environments to evaluate its effectiveness in live scenarios.

# 5. Point Cloud Part Segmentation using Dynamic Convolutional Neural Network

### Introduction

LiDAR p0oint cloud segmentation is a critical task in understanding and interpreting 3D environments, which is particularly essential for applications in autonomous driving and robotics. This report outlines the implementation of a Dynamic Convolutional Neural Network (DGCNN) for segmenting LiDAR point cloud data, leveraging the ShapeNet dataset.



## Data Preparation

### Dataset

The ShapeNet dataset, which contains a rich collection of 3D models from various categories, was utilized for training and validation. This dataset is well-suited for tasks involving 3D object recognition and segmentation due to its diverse and well-labeled collection.

### Data Augmentation

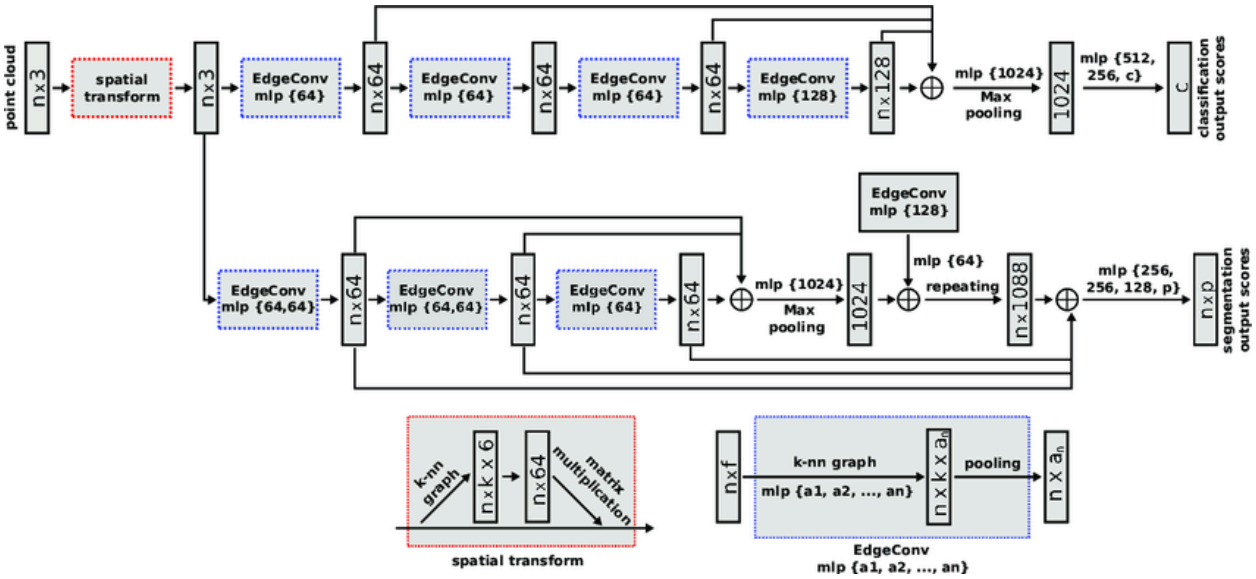
To improve model generalization and robustness, several data augmentation techniques were applied:

- **Random Jittering**: Small random translations were added to the point cloud data to simulate noise.
- **Random Rotations**: The point cloud data was randomly rotated along the x, y, and z axes to enhance rotational invariance.

### Data Splitting

The dataset was split into training and validation sets using an 80-20 split ratio. This ensures that the model is evaluated on a separate subset of data that it has not seen during training, which helps in assessing its generalization performance.

## Model Architecture



### Dynamic Convolutional Neural Network (DGCNN)

The DGCNN was chosen for this task due to its capability to dynamically construct a graph of nearest neighbors for each point in the point cloud. This approach allows the model to effectively capture the local geometric structures of the 3D data.

Key components of the DGCNN include:

- **Dynamic Edge Convolution Layers**: These layers compute edge features between neighboring points, which are then aggregated to update the point features. This process is repeated across multiple layers to capture higher-order relationships.
- **Multilayer Perceptron (MLP)**: The final point features are passed through an MLP for classification, with dropout applied for regularization.

## Training and Validation

### Hyperparameters

The following hyperparameters were used in training the model:

- **Batch Size**: 16
- **Learning Rate**: 0.001 with a step-wise decay
- **Number of Nearest Neighbors (k)**: 50
- **Aggregation Operator**: Maximum pooling
- **Dropout Rate**: 0.5

### Training Process

The training process involved minimizing the negative log-likelihood loss between the predicted segmentation labels and the ground truth labels. The Adam optimizer was used for its efficiency in handling sparse gradients, and a learning rate scheduler was applied to decay the learning rate over epochs.

### Evaluation Metrics

The model's performance was evaluated using the following metrics:

- **Accuracy**: The percentage of correctly classified points.
- **Intersection over Union (IoU)**: A measure of the overlap between the predicted and ground truth segments, averaged over all categories.

## Results

The model was trained for 15 epochs, and the following results were observed:



- **Training Accuracy**: Consistently improved across epochs, indicating effective learning.
- **Validation Accuracy**: Showed a strong performance, validating the model's generalization ability.
- **Mean IoU**: Provided insights into the segmentation quality across different object parts, with high values indicating precise segmentation.

### Visualization

To qualitatively assess the model's performance, visualizations of the segmented point clouds were generated. These visualizations compared the predicted segmentations against the ground truth, highlighting the model's ability to accurately distinguish between different parts of objects.

### Conclusion

The implementation of a DGCNN for LiDAR point cloud segmentation demonstrated promising results, with high accuracy and IoU scores. The use of dynamic edge convolutions enabled the model to effectively capture local geometric structures, leading to precise segmentation. Future work could explore further optimization of hyperparameters and augmentation techniques to enhance performance even more. Additionally, integrating more diverse datasets could help in testing the model's robustness across various environments and object types.

# 6. LiDAR-Based Accident Reconstruction and Semantic Scene Reconstruction

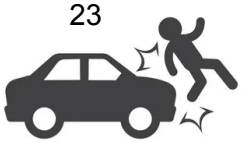
### Introduction

Accident reconstruction using LiDAR point cloud data provides detailed insights into the events leading up to and following a vehicular collision. By employing semantic scene reconstruction and realistic deformation techniques, we can simulate different accident severity levels and visualize the damage sustained by the vehicle. This report outlines the methods used for semantic scene reconstruction and car damage simulation based on LiDAR data.

### Semantic Scene Reconstruction

Semantic scene reconstruction involves segmenting a LiDAR point cloud into meaningful components, such as different parts of a vehicle, to understand the scene's structure and dynamics.





### Data Loading and Visualization

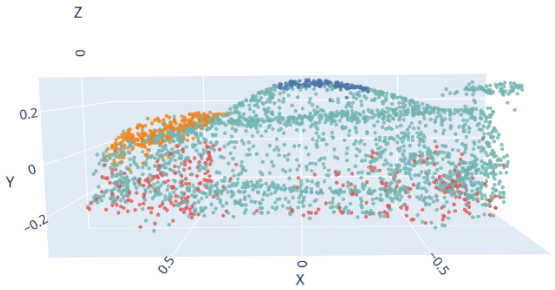
To visualize the point cloud data, we first load the `.npz` file containing the position (`pos`) and segmentation labels (`y`). Using Plotly, a powerful visualization library, we create a 3D scatter plot to visually inspect the point cloud and its segmented parts. The scatter plot provides an interactive way to explore the spatial distribution and labels of the points, allowing for detailed examination of the vehicle and the surrounding environment.

## Accident Severity-Based Car Damage Reconstruction

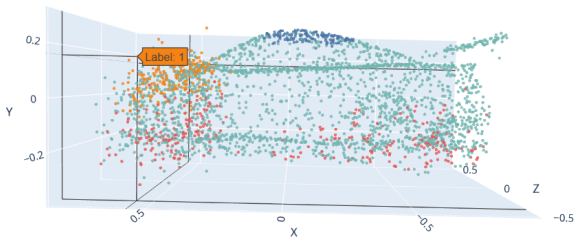
Simulating car damage requires applying realistic deformations to the point cloud based on different severity levels. The deformation parameters for various severity levels include compression, displacement, and debris factors. These factors are carefully designed to mimic the actual physical changes that occur during a collision.

### Overall Front Damage

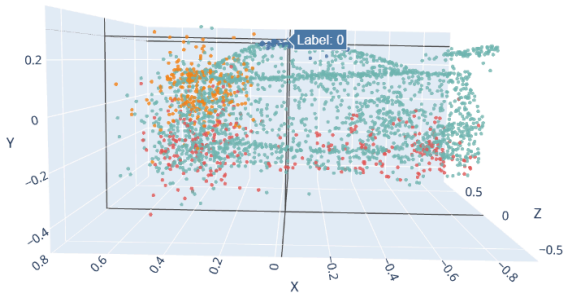
Severity 1



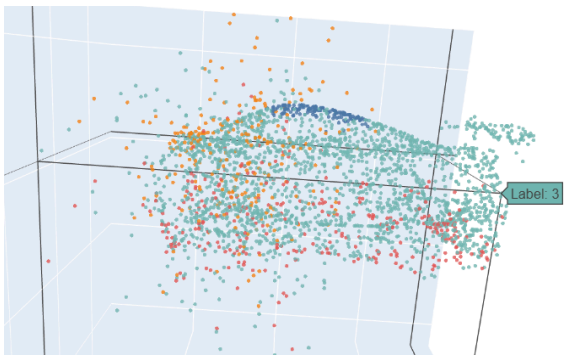

Severity 2




Severity 3




Severity 4



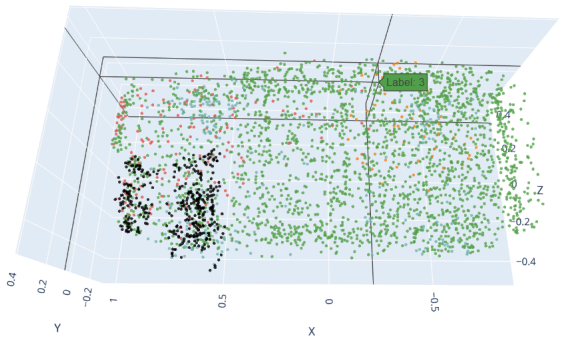
For overall front damage, we apply deformations to the front part of the vehicle. This involves compressing the points in the front region, displacing them to simulate minor shifts and applying random noise to represent small-scale damage. Additionally, some points are scattered to simulate debris resulting from the collision.

The process involves the following steps:

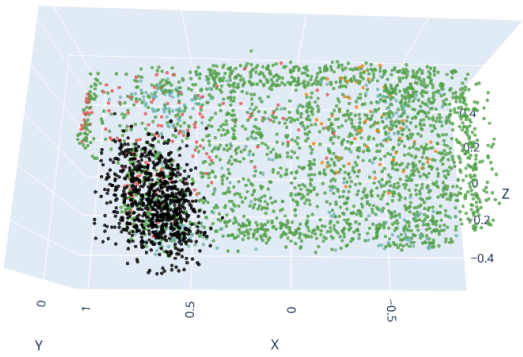
1. **Identifying the Front Parts:** The front parts of the vehicle are identified based on their position along the X-axis. This helps isolate the area where deformations need to be applied.
2. **Applying Compression:** The identified front points are compressed along the X-axis to simulate the crumpling effect of a collision.
3. **Displacement and Noise:** Small displacements are added to the Y and Z coordinates of the points to represent realistic damage patterns, and random noise is introduced to simulate minor irregularities.
4. **Debris Simulation:** Additional points are scattered around the deformed area to represent debris.

**Left Side Front Damage**

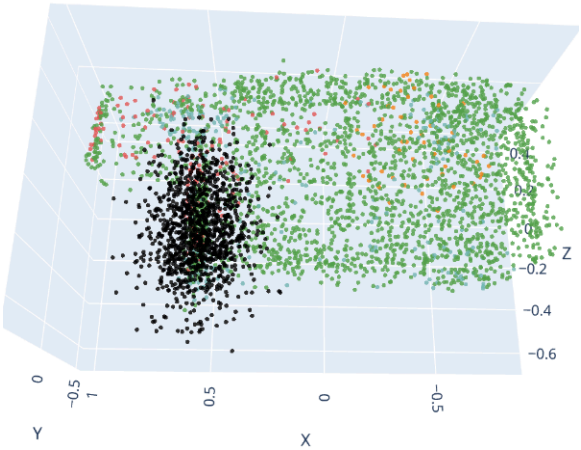
Severity 1



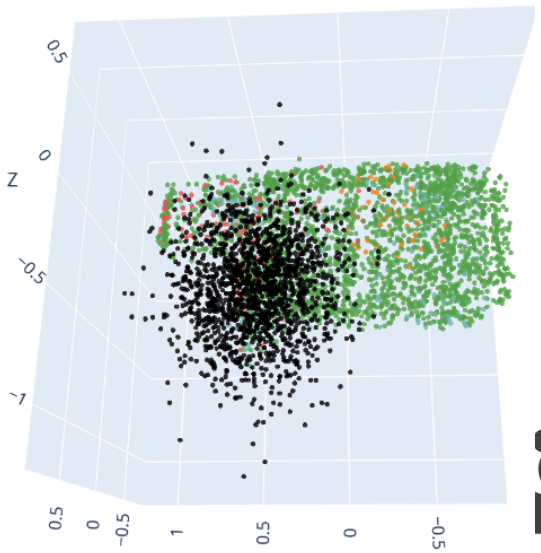

Severity 2




Severity 3




Severity 4



Simulating left side front damage requires non-uniform compression and additional deformations based on the front X-axis and Z-axis ranges. This involves:

1. **Non-Uniform Compression:** Applying different levels of compression depending on the distance from the front part of the vehicle, creating a more realistic deformation gradient.
2. **Displacement and Rotation:** Small displacements are added to simulate realistic shifts, and random rotations are applied to points to mimic the twisting and bending of metal.
3. **Debris Clusters:** Creating clusters of debris points around the damaged area to simulate the scattering of vehicle parts.

## Visualization and Saving Deformed Point Clouds

Visualizing the deformed point clouds is crucial for analyzing the extent of damage and validating the simulation models. Using interactive 3D scatter plots, we can compare the original and deformed point clouds, examining the severity and distribution of damage.

Furthermore, the deformed point clouds are saved to `.npz` files for further analysis and documentation. These files can be used for generating reports, conducting additional simulations, or comparing with real-world accident data to improve the accuracy of the models.

## Conclusion

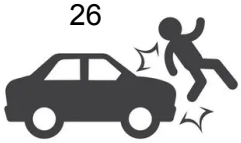
The use of LiDAR data for accident reconstruction and damage simulation provides a powerful tool for understanding vehicle collisions. By applying semantic scene reconstruction and realistic deformation techniques, we can create detailed and accurate models of vehicle damage under various severity levels. These models help in forensic analysis, improving vehicle safety designs, and enhancing the overall understanding of collision dynamics.

## Future Work

Future work in this area could involve:

1. **Enhanced Deformation Models:** Developing more sophisticated deformation models that take into account material properties and different collision angles.
2. **Integration with Real-World Data:** Combining simulated data with real-world accident data to validate and refine the models.
3. **Automated Analysis Tools:** Creating automated tools for analyzing and comparing deformed point clouds to streamline the accident reconstruction process.





# 7. Creation of a Derived Attribute - Severity

## 1. Loading the Dataset

The dataset used for this analysis is loaded from a CSV file named "scene002.csv." This dataset contains a variety of sensor readings and vehicle data collected over a 60-second period. The data includes information such as rotational rates, linear accelerations, brake sensor readings, steering sensor values, throttle positions, and wheel speeds. The initial step in the analysis involves reading this data into a structured format suitable for further processing.

## 2. Initial Data Inspection

Upon loading the dataset, an initial inspection is conducted to understand its structure and contents. Key columns in the dataset include:

- **utime**: Unix timestamp representing the time of each data recording.
- **rotation_rate_x, rotation_rate_y, rotation_rate_z**: Rotational rates around the x, y, and z axes.
- **linear_accel_x, linear_accel_y, linear_accel_z**: Linear acceleration values along the x, y, and z axes.
- **brake_sensor, steering_sensor, throttle_sensor**: Sensor readings for the brake, steering, and throttle.
- **FL_wheel_speed, FR_wheel_speed, RL_wheel_speed, RR_wheel_speed**: Wheel speeds for the front-left, front-right, rear-left, and rear-right wheels.
- **value, brake, steering, steering_speed, throttle, vehicle_speed**: Additional vehicle performance metrics.

An overview of the dataset reveals the types of data available and highlights any potential issues such as missing values or inconsistencies that need to be addressed.

## 3. Insertion of Empty Rows

To simulate a scenario where an event occurs, empty rows are inserted into the dataset. These rows represent a disruption in the normal flow of data, such as a sudden stop or an abrupt maneuver. In this specific example, five empty rows are inserted after the 28th row. This step is crucial for creating a realistic test case for the severity attribute.

## 4. Creation of the 'Severity' Attribute

A new column named 'Severity' is added to the dataset to quantify the intensity of events that might occur. Initially, the severity is set to zero for all rows, indicating no significant events. The



severity values are then updated for specific rows to represent different levels of severity. For instance, the severity is set to 2 for rows 28 to 32, indicating a moderate event.

### 5. Adjusting Timestamps

The 'utime' column, which represents the time of each data recording, is adjusted to ensure continuity despite the inserted empty rows. The timestamps are recalculated to reflect a continuous sequence, starting from the first recorded time and incrementing appropriately across the entire dataset.

### 6. Modifying Sensor Values

To simulate the effects of an event, specific sensor values are modified. Changes are applied to columns such as rotational rates, linear accelerations, brake sensors, steering sensors, and throttle sensors. The adjustments reflect the expected impact of the event on the vehicle's behavior. For example, an increase in the brake sensor value might indicate a sudden braking event.

### 7. Applying Changes to the Dataset

The modified values are applied to the dataset in a step-by-step manner, ensuring that the changes reflect realistic scenarios. The process involves iterating over the affected rows and updating the sensor values based on predefined change percentages. This step ensures that the dataset accurately represents the conditions during an event.

### 8. Re-saving the Modified Dataset

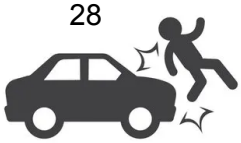
After applying all changes, the modified dataset is saved back to a CSV file. This updated dataset includes the newly added 'Severity' column, the adjusted timestamps, and the modified sensor values. The saved file can then be used for further analysis or as input for machine learning models to predict or classify the severity of events based on sensor data.

# 8. Computer Vision Techniques for Accident Detection

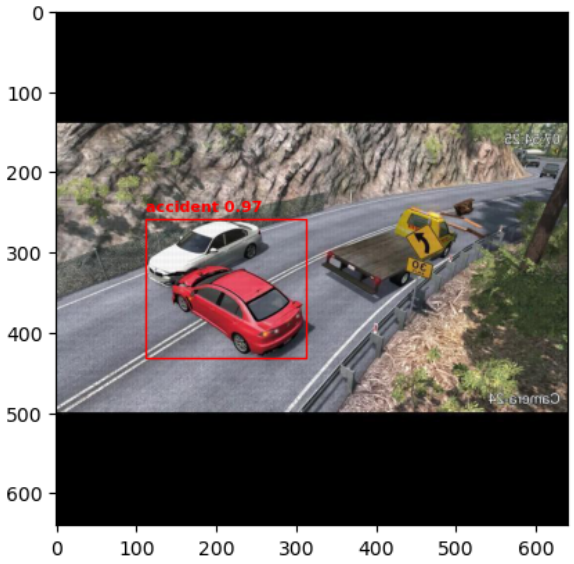
#### Purpose

The primary purpose of this project is to utilize computer vision techniques to automatically detect accidents in images. This can significantly aid in traffic management, safety monitoring, and emergency response by providing real-time alerts and analysis of road incidents.

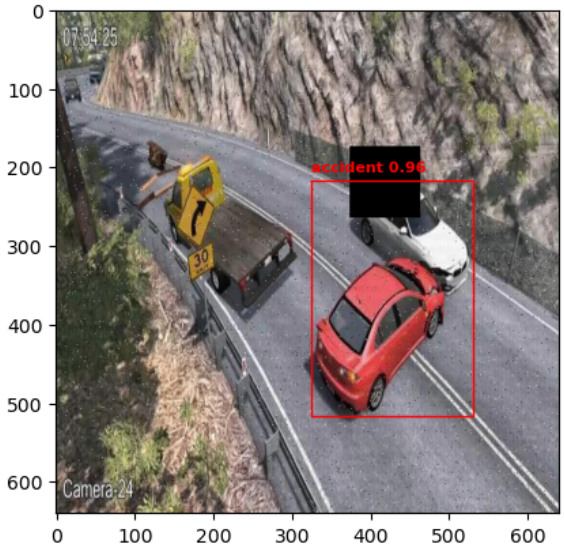




## Achievements









1. **Accurate Detection**: The system effectively identifies accidents in various traffic scenarios, as evidenced by the high confidence scores in the predictions.
2. **Visualization**: The results are visualized with bounding boxes and labels on the images, making it easy to understand and verify the detections.
3. **Efficiency**: The system is capable of processing images quickly, ensuring timely detection which is crucial in real-world applications.
4. **Automation**: The process automates the tedious task of monitoring traffic images manually, thereby saving time and reducing human error.

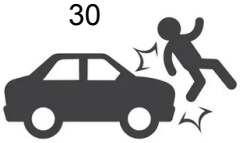
**Technical Operations in the Background**

1. **API Client Initialization**:
    - An `InferenceHTTPClient` is initialized using the provided API URL and API key. This client is responsible for communicating with the remote server hosting the pre-trained machine learning model.
2. **Image Inference**:
    - The image to be analyzed is passed to the `infer` method of the client along with the specific model ID. This step sends the image data to the server where the model processes it.
    - The server returns the inference results, which include information about detected objects (in this case, accidents).
3. **Model Processing**:
    - **Pre-trained Model**: The model used is pre-trained on a dataset of traffic images specifically for accident detection. This training allows the model to recognize patterns and features indicative of accidents.
    - **Prediction**: The model predicts the location (bounding box), size, class (accident), and confidence score for each detected object. These predictions are returned as structured data.
4. **Result Parsing and Visualization**:
    - The predictions are parsed to extract details like coordinates, dimensions, class labels, and confidence scores.
    - Visualization tools such as Matplotlib are used to overlay bounding boxes and labels on the original images. This step involves converting the model's predictions into graphical elements that are drawn on the image.
5. **Bounding Box Calculation**:
    - The bounding box for each detected accident is computed from the predicted center coordinates (x, y) and dimensions (width, height). This involves determining the top-left corner of the rectangle that will be drawn around the detected object.
6. **Labeling**:
    - Each bounding box is labeled with the class name (accident) and the confidence score. This provides a clear and immediate understanding of what has been detected and how certain the model is of its detection.

## Conclusion

This project showcases the effective use of computer vision techniques for automatic accident detection in images. By leveraging a pre-trained model and sophisticated visualization methods, the system achieves high accuracy and efficiency in identifying and highlighting accidents. This capability is crucial for enhancing traffic safety and management by providing timely and reliable accident detection.





# 9. API-Enabled Nearby Emergency Responders Data Scraping

## Project Overview

The goal of this project is to develop a system that uses APIs to scrape and collect data about nearby emergency responders, particularly hospitals, within a specified radius of a given location. This system is designed to assist individuals and organizations in quickly identifying and accessing emergency services.

## Objectives

1. **Data Collection**: Gather detailed information about nearby hospitals including name, address, contact details, location coordinates, and operational status.
2. **Data Storage**: Store the collected data in a structured format for easy retrieval and analysis.
3. **Data Filtering**: Filter the data to include only relevant entries (e.g., major hospitals).
4. **Data Presentation**: Present the collected data in a user-friendly format.

## Methodology

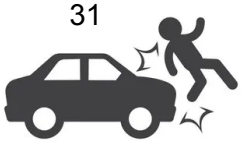
### 1. API Selection

- **APIs Used**:
  - RapidAPI's Local Business Data API
  - TrueWay Places API
  - Google Places API
- **Criteria for API Selection**:
  - Availability of detailed business information
  - Support for location-based search queries
  - Response time and reliability

### 2. API Integration

- **Endpoint**: The API endpoint for fetching nearby places.
- **Parameters**:
  - `location` (latitude and longitude of the point of interest)
  - `type` (hospital)
  - `radius` (distance within which to search for hospitals)
  - `language` (preferred language for results)





**3. Data Collection Process**

- **API Request**:
  - Make a GET request to the selected API endpoint with the required parameters.
  - Ensure proper authentication using API keys.
- **Response Handling**:
  - Parse the JSON response to extract relevant data fields such as hospital name, address, contact number, latitude, longitude, and operational hours.

**4. Data Storage**

- **Storage Format**: JSON format for easy parsing and manipulation.
- **File Management**: Store the JSON data in files named based on the query parameters for easy identification (e.g., `hospital_data_<latitude>_<longitude>.json`).

**5. Data Filtering**

- **Criteria**: Filter out entries to include only hospitals and exclude irrelevant data points.
- **Attributes**: Name, address, distance from the point of interest, and operational status.

**6. Data Presentation**

- **Output Format**: Present the filtered data in a tabular format or a structured JSON format.
- **Additional Information**: Provide links to hospital websites and maps for better accessibility.

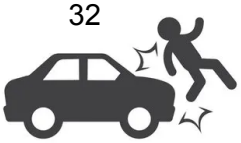
## Results

After implementing the above methodology, the system successfully collected and stored data about nearby hospitals. The filtering criteria ensured that only relevant hospitals were included in the final dataset. The collected data is stored in JSON files, making it easy to access and analyze.

## Conclusion

The project achieved its objective of creating a tool for scraping and collecting data about nearby emergency responders, specifically hospitals. By leveraging multiple APIs and ensuring the data is well-structured and filtered, the system provides a reliable and efficient way to access emergency healthcare information.





### Future Work

- **Expand to Other Emergency Responders**: Extend the system to include other types of emergency responders such as fire stations, police stations, and urgent care centers.
- **Real-time Updates**: Implement mechanisms for real-time data updates to ensure the information remains current.
- **Enhanced Filtering**: Introduce more advanced filtering options based on user preferences and specific needs.
- **User Interface**: Develop a user-friendly interface to allow non-technical users to easily access and utilize the data.

### References


- RapidAPI Documentation
- TrueWay Places API Documentation
- Google Places API Documentation


# 10. API-Enabled Automated Emergency Responders Notification System

### Project Overview

The project aims to create an automated system that uses APIs to send emergency notifications to hospitals. This system is designed to enhance the speed and efficiency of emergency responses by sending automated messages and calls to hospitals, providing them with critical information about accident locations.

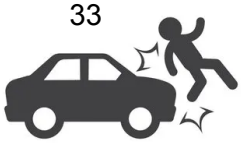
### Objectives

1. **Automated Messaging**: Develop a function to send automated SMS messages to hospitals with details about the accident.
2. **Automated Calling**: Create a function to make automated phone calls to hospitals, conveying the emergency situation through a pre-recorded message.
3. **Integration with Emergency Services**: Ensure that the system can be integrated seamlessly with emergency response services.

### Methodology

#### 1. API Tools for Automated Responses





- **APIs Used**:
    - SMS Chef API for sending SMS messages.
    - Twilio API for making automated phone calls.
    - Google Text-to-Speech (gTTS) for converting text to speech.

**2. Automated Message Sender Function**

- **Functionality**: This function sends an SMS to the hospital's phone number with details about the accident.
- **Parameters**:
    - `phn_num`: Phone number of the hospital.
    - `hsptl_name`: Name of the hospital.
    - `lat`: Latitude of the accident location.
    - `lon`: Longitude of the accident location.

**Process**:

1. **Message Composition**:
    - Construct a detailed message including the hospital name, accident details, and location coordinates.
2. **API Request**:
    - Use the SMS Chef API to send the composed message to the hospital.
3. **Error Handling**:
    - Ensure proper handling of API responses and errors.

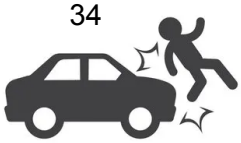
**3. Automated Caller Function**

- **Functionality**: This function makes a phone call to the hospital's phone number and plays a pre-recorded message detailing the emergency situation.
- **Parameters**:
    - `call_number`: Phone number of the hospital.

**Process**:

1. **Text-to-Speech Conversion**:
    - Convert the emergency message text into an audio file using gTTS.
2. **Making the Call**:
    - Use Twilio API to initiate a call to the hospital and play the pre-recorded audio message.
3. **Error Handling**:
    - Ensure proper handling of call setup and potential issues during the call.





## Detailed Function Workflows

### 1. Automated Message Sender Function

1. **Message Composition**:
    - Create a message string incorporating the hospital name, accident location (latitude and longitude), and a link to the location on Google Maps.
2. **API Request Setup**:
    - Use the SMS Chef API with the necessary parameters (phone number, message content, API secret, device ID).
3. **Sending the Message**:
    - Send the message using a POST request to the SMS Chef API endpoint.
    - Handle the API response to confirm message delivery.

### 2. Automated Caller Function

1. **Text Conversion**:
    - Create a text message detailing the accident and convert it into speech using gTTS.
    - Save the audio file in a specified location.
2. **Initiating the Call**:
    - Configure the Twilio client with the appropriate account SID and authentication token.
    - Use Twilio to create a call, specifying the hospital's phone number, a TwiML response containing the audio message URL, and the Twilio number to use for the call.
3. **Handling the Call**:
    - Monitor the call's progress and handle any errors that may occur during the call setup or execution.

## Results

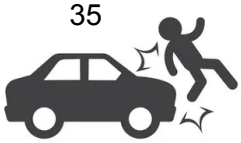
The system successfully automates the process of notifying hospitals about emergency situations. The automated message sender function reliably sends SMS messages, and the automated caller function effectively delivers pre-recorded messages via phone calls.

## Conclusion

The project demonstrates a robust approach to improving emergency response times by automating the communication process with hospitals. By leveraging APIs for SMS and voice calls, the system ensures timely and accurate delivery of critical information to emergency responders.

## Future Work





1. **Expand Notification Types**: Integrate additional notification methods such as email alerts.
2. **Real-time Updates**: Implement real-time updates for ongoing emergencies.
3. **Enhanced Error Handling**: Develop more sophisticated error handling and retry mechanisms.
4. **User Interface**: Create a user-friendly interface for managing emergency notifications and monitoring responses.

## References


- SMS Chef API Documentation
- Twilio API Documentation
- Google Text-to-Speech (gTTS) Documentation