Guardian Angel

Name: Kavya Chandrika Vempalli

ASURITE: kvempall

ASU ID: 1229837858

Our version of guardian angel is a personalized context-aware application aimed at improving the quality of life and safety of individuals with diabetes, enhancing rider safety and emergency response. The project's primary objective is to develop a comprehensive system that monitors and manages blood sugar levels in real-time and enhances rider safety through an autonomous vehicle advisory controller.

This project includes,

- ➔ Emergency Services Notification
- ➔ Enhanced Rider Safety Measures
- ➔ Health-Centric Food Recommendations
- ➔ Tailored Accommodation Suggestions
- ➔ Reliable Advisory Controls.

The project is divided into three parts.

Section 1: In this section, user information is taken through a mobile application. User Information typically includes Heart Rate, Respiratory Rate, Blood Sugar levels, User Schedules, User's Travel Bookings and Medical Records. All this data is stored in the database.

Section 2: In this section, user's vehicle and travel information is taken. Vehicle information comprises of distance between two cars, Speed of the car. Travel information comprises of source and destination information of the travel and Road Condition.

Section 3: In this section, Advisory Controller is used. It takes the heart rate, respiratory rate and vehicle information from task 1 & task 2 and calculates whether the user has to switch or not. If the output is no switch, then it suggests a prompt "Relax", if the ouput is switch, it throws an alert through the mobile app. If the output is crash, emergency services are contacted. The Integration is done in project 5.

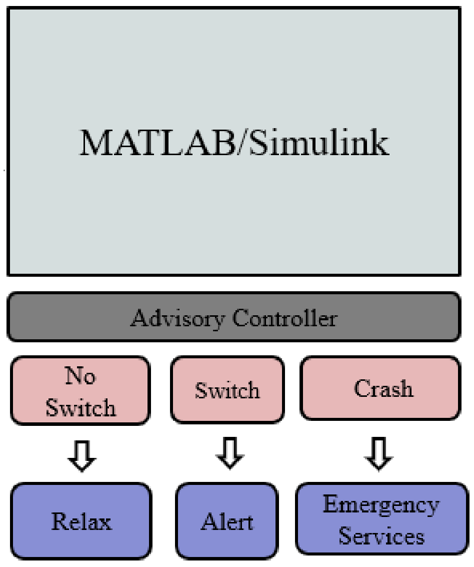
Alignment with Guardian Angel:

Created advisory controller using the user and vehicle data and determined if the user needs to be alert or if there is any emergency or if he can relax. Within project 4, I concentrated on the MATLAB segment of the implementation. To gather user data from a mobile app, a channel was established in thingsSpeak, encompassing six fields: HeartRate, RespRate, Speed, Distance, RoadCondition, and Collision_status. The initial five fields serve as inputs for the .m file, aiding in the computation of reaction time based on road conditions. LaneMaintainSystem.slx employs the initial Speed and Distance values to determine the speed limit and distance limit parameters.

During model simulation, distance and time series values are produced. These values are instrumental in calculating the time of collision. If all distance values generated exhibit negativity, no immediate action is necessary, and the user can remain relaxed, denoted by a collision status of 1.

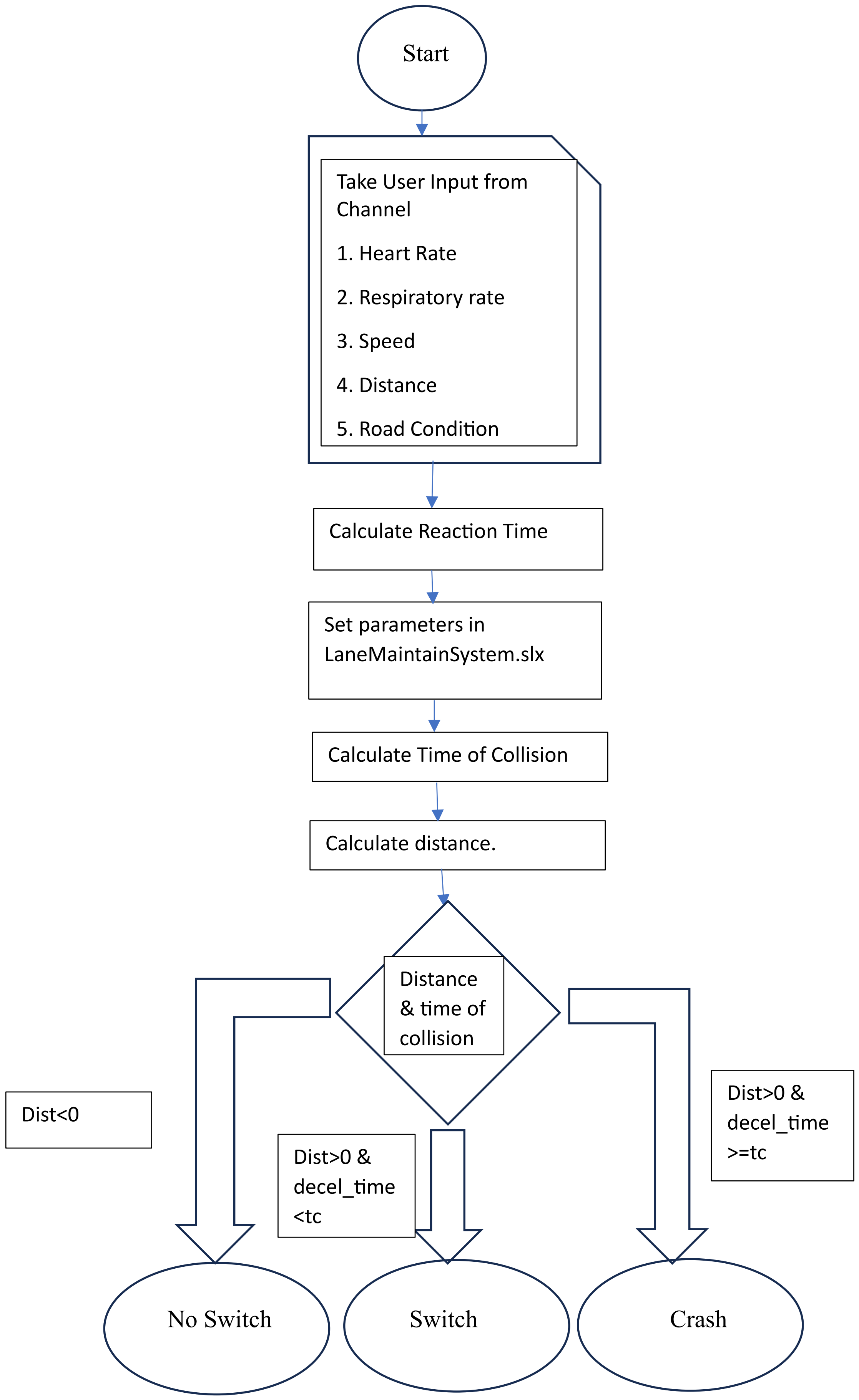
However, if some distances surpass zero, the HumanActionModel is executed to evaluate the feasibility of switching based on reaction time. If the duration required to decelerate the car is shorter than the time of collision, a switch is deemed possible (collision status 2). Conversely, if deceleration isn't completed before the impending collision time, a crash occurs, marked by a collision status of 3. The resultant output is channelled back for access by the mobile application.

Specifications



- For taking the user data from mobile app, a channel is created in thingsSpeak. The Channel consists of total six fields, HeartRate, RespRate, Speed, Distance, RoadCondition and Collision_status.
- The first five fields are taken as input by the .m file and depending on road condition, reaction time is calculated.
- Using the initial Speed and Distance values, the speed limit and distance limit parameters are LaneMaintainSystem.slx
- When the model gets simulated, it gives distance and time series values. Depending on the value of distance and time, time of collision is calculated.
- If all the distance values generated are negative, then there is no requirement to switch and the user can relax. The collision status is 1.
- If some of the distances are greater than 0, then the HumanActionModel is simulated to check if it is possible to switch or not, depending on reaction time.
- If the time taken to decelerate the car is less than the time of collision, then there is a possibility to switch. Here the collision status is 2. Else, the deceleration is not completed before the time of collision, then it results in crash. The collision status is 3 in this case. The output is given to the channel, to be accessed by the mobile application.

Design:


Start
Take User Input from Channel
1. Heart Rate
2. Respiratory rate
3. Speed
4. Distance
5. Road Condition
Calculate Reaction Time
Set parameters in LaneMaintainSystem.slx
Calculate Time of Collision
Calculate distance.
Distance & time of collision
Dist<0
Dist>0 & decel_time <tc
Dist>0 & decel_time >=tc
No Switch
Switch
Crash

Testing Strategies

1. One of the crucial testing strategies involves Simulink terminal output checking. This method entails meticulously examining the outputs produced at various stages of Simulink simulations. Engineers and developers assess these terminal outputs to verify the accuracy of computations, assess intermediate results, and validate the functioning of different components within the Simulink model. By scrutinizing the terminal outputs, potential errors or discrepancies can be detected and rectified, ensuring the model operates as intended and adheres to desired specifications.

2. Another testing approach involves the alteration of gain values within the Simulink model. Engineers employ this strategy to evaluate the sensitivity of the system to changes in gain parameters. By systematically adjusting these values and observing the resultant outputs, testers can gauge the impact on system behavior. This method helps in optimizing the performance of the model, identifying potential instabilities, and ensuring the system's robustness under varying gain configurations.

3. Utilizing a scope to examine and analyze crashes forms another integral part of the testing process. Engineers leverage the scope functionality within Simulink to visualize and monitor specific signals or variables of interest during simulations. By tracking critical parameters related to crash scenarios, such as vehicle speed, distance, or collision-related data, engineers can observe trends, anomalies, or potential issues. This facilitates a comprehensive assessment of the model's behavior and aids in identifying any unexpected behavior or irregularities that might lead to crashes.

4. Placement of the Human Action Model within the simulation framework serves as a critical step in assessing the feasibility of switch actions. Engineers strategically integrate the Human Action Model to simulate and evaluate human responses or actions concerning potential switches in the system. By simulating various scenarios and assessing the timing and appropriateness of human-driven actions, testers can determine the system's response in different situations. This step ensures that the model adequately accounts for human decision-making processes, validating the system's ability to react appropriately based on human actions.

These testing strategies collectively contribute to the comprehensive evaluation and validation of Simulink models, ensuring their accuracy, robustness, and adherence to the desired functionalities and safety standards.

Navigating Challenges

1. Initially, attempts were made to utilize JSON output as a means to relay information to the mobile application. However, this approach posed certain limitations, resembling the process of hardcoding information. Despite JSON's flexibility in structuring and transmitting data, the method proved to be insufficient for dynamically transferring real-time or dynamic information between the MATLAB system and the mobile application. The rigidity of the JSON output constrained the fluidity required for seamless and adaptable data exchange between the two platforms, prompting exploration of alternative methods.

2. Exploration continued with the utilization of MATLAB Web applications in conjunction with the mobile application to establish a connection. This strategy aimed to create an interface that would facilitate communication and interaction between the MATLAB system and the mobile application. The development of a MATLAB Web app enabled a more interactive and accessible means for users to interact with the system, offering potential advantages in data exchange and control. However, this approach also revealed certain limitations in terms of real-time data transmission and the complexity of maintaining a synchronized connection between the MATLAB Web app and the mobile platform.

3. Subsequently, efforts were directed towards employing Application Programming Interfaces (APIs) to establish a link between the mobile application and MATLAB. APIs provided a structured and standardized approach for enabling communication and data exchange between different software systems. By leveraging APIs, attempts were made to streamline the interaction between the mobile application and the MATLAB environment. This approach aimed to offer a more efficient and flexible means for transmitting data, commands, or requests between the two platforms. However, implementing this method necessitated meticulous attention to API design, integration, and security measures to ensure a robust and reliable connection.

These explorations into various methods—such as JSON output, MATLAB Web apps, and APIs—highlight the iterative process undertaken to establish an effective and efficient link between the mobile application and the MATLAB system. Each approach revealed unique strengths and limitations, prompting a continued search for the most suitable method that could seamlessly facilitate dynamic data exchange while ensuring reliability and usability for both platforms.