Hug The Rail IoT Project



Team Pseudocode

Members: GaYoung Park,

Kyle Henderson,

Shinya Abe,

Joseph Thompson IV

Table of Contents

1. Introduction––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––– 3

1.1 Problem Statement

1.2 Purpose of the Product

1.2.1 Importance and Value to the operation

1.3 Team Information

1.4 Evolving current operation

1.5 Approach to the solution

1.6 Reviews

1. Overview–––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––– 4

2.1 Timeline

2.2 Roles and responsibilities

2.3 Problems

2.4 Possible Hazards

2.5 Solutions to the Problems

2.6 Sensor types

1. Requirements––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––– 6

3.1 Non-Functional Requirements

3.2 Functional Requirements

1. Requirements Modeling––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––– 9

4.1 Use Cases

4.2 Use Case Diagram

4.3 Class-Based Modeling

4.4 CRC Modeling/Cards

4.5 Activity Diagrams

4.6 Sequence Diagrams

4.7 State Diagrams

1. Software Architecture –––––––––––––––––––––––––––––––––––––––––––––––––––––––––––– 29

5.1 Software Architecture Models

5.2 Pros and Cons

5.3 Optimal Software Architecture IoT HTR

1. Project Code –––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––– 32
2. Tests –––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––– 49

**1. Introduction**

1.1 Problem Statement

There is a need to provide backup to the train provided by Hug the Rail, so that it is able to make decisions locally if connection to wifi and cell service is lost. This is in order to have an extra safety measure that would allow productivity to continue in this likely event.

1.2 Purpose of the Product ( IoT Hugs The Rail):

The purpose of the product is to make local decisions without need of connection to an offsite service, and use data based off of IoT HTR devices, to allow a local operator to input different commands into the system and have those commands override the automated decisions, and to be able to update rules of operation of the train easily from a set station.

1.2.1 Importance and Value to the operation:

The project will help Hugs the Rail reduce danger, improve efficiency for the conductor, save costs, and improve customer service.

1.3 Team Information (Pseudocode):

* Shinya Abe
* GaYoung Park
* Joseph Thompson IV
* Kyle Henderson (Team Leader)

1.4 Evolving Current Operation:

This project is adaptable and as a result features are added and improved while the product is in service

1.5 Approach to the Solution:

Our approach to this problem will be done through the Software Development Process of the Iterative Model and the use of the Java programming language. We will implement sensors, so that in the case of network failure, the train will still have access to vital information to keep itself safe. This will act as a form of dashboard to the conductor, while giving advice on how to best operate the train. In the case of an emergency (once it hits past a certain threshold that life might be at stake), the onboard system will be able to dynamically adjust the conditions of the train such as speed, but also have the flexibility for the on-board conductor to override the system if the system is in error.

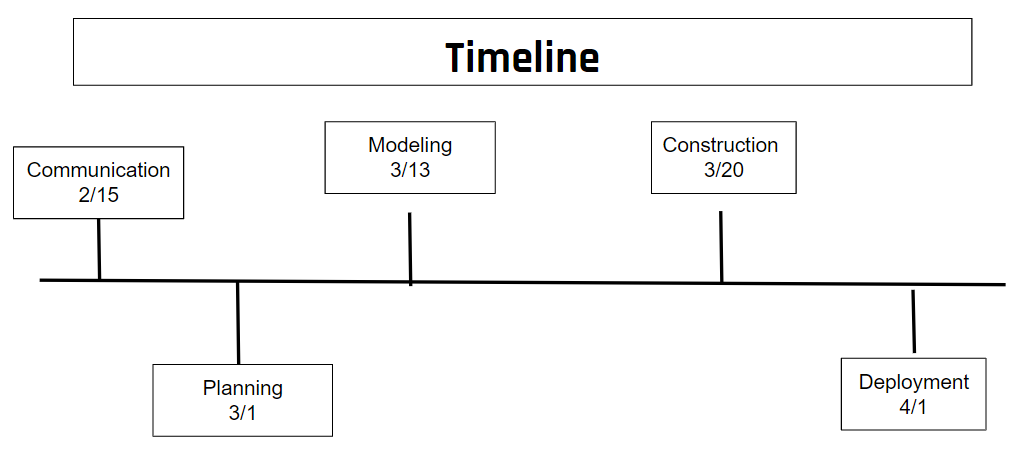
1.7 Reviews

1.7.1 Review with Team: By reviewing with the team members regularly, our team will ensure that the team is addressing the problems and meeting the deadlines. For each review, there will be a test run in order to make sure the progress that has been made is working properly.

1.7.2 Review with Stakeholders and Users: Talking to the stakeholders and users and understanding their needs is crucial to providing the best service possible. Our team will be regularly checking with the stakeholders to ensure that their concerns and needs are being addressed and known. They will be given a report of our progress so that the team can make changes when necessary. In order to test the functionality of the completed project, some users will be asked to test the working train and address any difficulties or concerns that may rise during the process.

**2. Overview**

2.1 Timeline



2.2 Roles and Responsibilities

Kyle Henderson: As the team leader, he will be responsible for assigning roles to each member of the team, making sure that the team is on track, and scheduling meetings with stakeholders.

Shinya Abe, Joseph Thompson IV, GaYoung Park: As software engineers, these team members will be responsible for creating, updating, testing, and maintaining the software based on the demands and problems that rise and address the stakeholder’s needs.

2.3 Problems

Trains depend on WiFi/Cellular networks to receive live data about their environments and surrounding traffic. When the train loses their Wi-Fi/cellular network, they need local information to operate safely. In the case of wifi-/network failure, the train would be able to detect hazardous conditions and as a result will be able to improve safety. Local information such as sensors are useful when encountering hazardous conditions such as ice,rain, wind, and objects.

2.4 Possible Hazards

For this project we need to be aware of a few hazards. Ice, snow, and rain would hinder the ability for the train to brake during sharp turns, at stations, or emergencies. Wind can unintentionally increase or decrease the speed of the train which could cause accidents. Finally, there can be objects such as a car that are hazardous.

2.5 Solutions to the Problems

2.5.1 Alert that there is something on the track using the distance sensors.

2.5.2 Sense other stopped or moving trains and provide rule for action using distance sensors and

motion sensors.

2.5.3 Follow instructions for each different weather conditions and detect outside temperature using

heat sensors.

2.5.4 Recommend action based on situation to operator.

2.6 Sensor Types

2.6.1 Motion sensors: Motion sensors can be installed on tracks so that when there are moving objects on the track, the sensors can warn the operator about the presence of objects (that could be animals, boxes, etc) as well as if the train is slightly displaced on the rail (does by measuring vibrations).

2.6.2 Heat sensors: Heat sensors can be used to detect weather conditions based on the outside temperature. Based on the temperature, it can predict the presence of snow, ice, or rain and warn the user when the temperature is too hot and could lead to the train overheating.

2.6.3 Distance sensors: Distance sensors can be used to detect the presence of objects on the rail and warn the user to slow down or prepare to stop in order to prevent possible injuries. Distance sensors can also be used to detect the presence of different trains approaching in order to avoid trains crashing.

2.6.4 GPS sensor: GPS sensors can be used to receive the location of other trains so that their expected location will be known based on their direction and speed. This will also prevent trains crashing and allow the facilitation of train operation.

**3. Requirements**

3.1 Non-Functional Requirements

3.1.1 Sensor Requirements

NR-1: Two heat sensors, two cameras, two distance sensors shall be installed to the front, middle, and the back of the train.

NR-2: One GPS sensor shall be installed in the front and the back of the train.

NR-3: One motion, vibration, and heat sensor shall be installed for each 2 meter of the track.

NR-4: Sensors shall be water-proof such that they last for 1000 hours under water.

NR-5: Sensors shall be able to withstand 100 N.

NR-6: One warning light and sound should be each installed to the front and back of the train.

3.1.2 IoT HTR shall be supported by a hardwired LoRaWan network.

NR-6: Sensors operating in the LoRaWan network would be energy efficient while simultaneously covering a wide area.

NR-7: The locomotive shall maintain its full functionality even when the network fails.

NR-8: When network connection is unstable or interrupted, the IoT HTR and the operator shall be known of the problem within 0.1 second.

3.1.3 Performance

NR-8: When the conductor logs into the IoT HTR, it shall be operational within 2 seconds.

NR-10: IoT HTR shall process an event within 0.1 second of its occurrence and decide whether to perform an immediate action in emergencies or wait for conductor response.

NR-11: Sensors shall send the updated information every second and the IoT HTR data will be displayed on the IoT screen.

NR-12: The sensors should be able to detect its malfunctions or damages and the functionalities shall be displayed to the IoT screen with 99.99% accuracy within 0.1 second of detection.

NR-13: The sensors shall be able to withstand various weather conditions such as heavy rain, storm, and snow.

NR-14: All sensors shall operate with 99.99% accuracy and fully work during its operation.

3.1.4 Reliability

NR-15: IoT HTR as well as the IoT HTR sensors reliabilities shall be no less than 99.99%.

NR-16: In order to ensure that the reliability remains accurate, there will be regular test runs before and after IoT HTR operation and weekly checks to test its accuracy.

NR-17: In case of a IoT screen malfunction, two back up IoT screens should be installed in the train that the conductor can log into.

3.1.5 Security

NR-18: Access to IoT HTR and the sensors should remain secured and limited to authorized individuals by maintaining secure login information and access codes/cards.

NR-19: Conductor room shall require an employee card as well as an access code that updates weekly.

NR-20: IoT HTR access shall be only granted when the conductor’s employee ID and password have been entered.   
NR-21: Software updates shall be done daily when IoT HTR is not in use by using the conductor’s ID and password as well as the system administrator’s.

NR-22: System administrators should have access to the log data, system update, and system operation updates by using their ID and password.

3.2 Functional Requirements

Functionalities will be discussed and tested with other companies to make sure that their products meet our requirements. Upon meeting the CTO of HTR, it was concluded that at minimum the following features are required:

3.2.1 Detect standing objects on the path of the train with distance and suggest action to the operator for

braking or increasing/decreasing/stopping the speed of the train.

R-1: The motion sensors on the tracks should be able to detect the objects on the path every 0.1 second with 99% accuracy and send the information to IoT HTR in 0.1 second.

R-2: The sensor has three states: normal, warning, and emergency.

R-3: In the case of an emergency, the train should proceed to stop within 0.01 seconds after detecting the emergency and come to a complete stop within 3 seconds.

R-4: In case of a warning, the IoT screen displays a warning on the screen and slows down or speeds up.

R-5: Conductor can access the image of the path for the object when given the location by accessing the nearby camera from the IoT HTR.

R-6: Whenever the conductor is given an option to decide from and a response is not given within 3 seconds, the IoT HTR performs based on its emergency manual.

R-7: If the object on the train’s path is bigger than 10x10x4 (in), the warning should be displayed on the IoT screen with its estimated size location and update the info on the IoT screen every 0.5 second until it is removed.

R-8: If the object on the path that had been detected and was requested for a removal has not been removed and the train is 1 km away from the object, an emergency should be displayed on the IoT screen and the train must begin to decelerate.

R-9: When living creatures are detected on the path of the train by the motion and heat sensors, the rail conductor around the location shall be notified within 1 second and should display a warning on the IoT screen.

3.2.2 Detect moving objects and distance ahead and behind and their speed and suggest to the conductor braking or changing speed.

R-10: Moving objects that are within the 10 km of the train should be reported to the conductor within 0.01 seconds with 99% accuracy.

R-11: Calculate the suggested speed based on the distance from the object to the train and train’s speed so that the train will come to a complete stop within 1km from the object if the object is in front of the train.

R-12: If an object is approaching the train from behind, display the image of the object from the camera located at the back of the train on the IoT screen.

R-13: If the object approaching from behind is another train and the distance between is less than 20km, warn the other train to reduce and keep the distance of 30km minimum.

R-14: Once the object is removed, the train should operate at its normal speed.

3.2.3 Detects gate crossing open/closed, distance and suggests to change speed or to brake or to brake to stop the operator.

R-15: Scheduled gate opening and closing should have already been updated in the system and known to the operator.

R-16: In the case of unexpected changes or failure in gate opening and closing, the operator should be notified before the train is within the 50 km mark from the gate.

R-17: In the case of a sudden emergency of gate not opening, the sensors should detect the gate failure before entering the distance of 5 km from the gate. Detecting the gate failure, the train should notify the operator within 0.01 second and come to a complete stop within 3 seconds.

3.2.4 Detects wheel slippage using GPS speed data and compares it with wheel RPM and suggests to the operator change speed or brake.

R-18: The data should be sent to the operator within 0.01 second of detecting differences in speed with 99.9% accuracy.

R-19: Sensors pass numbers of RPM to IoT HTR. IoT HTRshould calculate the slippage based on the wheel size, rotation per minute, and speed from GPS.

R-20: Compare the speed differences and when the difference is larger than 5%, slippage is detected and speed change is initiated.

R-21: The optimal speed should be calculated and sent to the operator with 99.9% accuracy and should adjust to the optimal speed.

**4. Requirements Modeling**

4.1 Use Cases

**1) Use Case:** Sensors detect standing objects on the path of the train.

Use Case No: 4.1.1

Primary Actor: Conductor

Secondary Actor: Sensor

Goal: Stop the train.

Preconditions: Train is in operation; train is in motion; object detected is non-living;

Trigger: Object is detected in the path of the train

Scenario:

1. Sensor detects an object on the path that is larger than 10x10x4 (in).
2. Sensor shows the location, size, time until collision on the IoT screen for the conductor.
3. The conductor and IoT HTR can access the cameras located on the path and IoT HTR makes a decision within 1 second.
4. IoT HTR can decide to slow down or stop, slow down to requested speed or stop and request the removal of the object on the path.

Exceptions:

1. The train is moving less than 25mph and the object is not alive.
2. The object is smaller than 10x10x4 (in).

**2) Use Case**: Sensors detect moving objects and distance ahead and behind.

Use Case No: 4.1.2

Primary Actor: IoT HTR

Secondary Actor: Sensor

Goal: Stops or slows the train depending on the movement of the object.

Preconditions: Train is in operation

Trigger: Either living or non-living object is detected moving towards or away from the train.

Scenario:

1. Sensor detects a moving object on the path that is larger than 10x10x4 (in).
2. Sensor sends the speed, size, temperature, and location of the object to IoT HTR.
3. Send a sound and light warning to the direction the object is detected.
4. IoT HTR calculates how much the train needs to slow down so that the train will come to a complete stop within 1km from the object when the object is in front of the train.
5. IoT HTR calculates how much the train needs to speed up so that the object/train behind will not collide with the train.
6. When the object is removed, the train will operate at its normal speed.

Exceptions:

1. The object is removed.
2. Smaller than 10x10x4 (in).

**3) Use Case:** IoT HTR decides to slow down or stop.

Use Case No: 4.1.3

Primary Actor: IoT HTR

Secondary Actor: Sensor

Goal: Slow or stop the train given the size of the detected non-living object

Preconditions: Train is in operation; train is in motion; object detected is non-living

Trigger: Object is detected in the path of the train

Scenario:

1. IoT HTR gives a warning about an object on the track on the IoT screen.
2. IoT HTR calculates the optimal speed and slows down or speeds up so that collisions with the object will be avoided.

Exceptions:

1. N/A

**4) Use Case:** Sensors detect that the gate is closed.

Use Case No: 4.1.4

Primary Actor: IoT HTR

Secondary Actor: Sensor

Goal: Notify the conductor and stop the train.

Preconditions: Not connected to wifi; Gate is failing

Trigger: Given gate fails to open.

Scenario:

1. Gate attempts to open and fails.
2. Sensor identifies failure and sends a warning to Conductor.
3. IoT HTR stops the train leaving 1 km between the train and the gates.

Exceptions:

1. N/A

**5) Use Case:** Sensors detect wheel slippage using GPS speed data.

Use Case No: 4.1.5

Primary Actor: IoT HTR

Secondary Actors: Wheel Sensors, GPS Sensor

Goal: Adjust speed to stop slippage

Preconditions: Train is in motion; Slippage is detected

Trigger: Wheel rotation does not match with train speed from GPS

Scenario:

1. Train accelerates in one direction (turning, speeds up, slows down).
2. Wheel Sensors give rotation data to IoT HTR.
3. GPS Sensor speed is given to IoT HTR
4. IoT HTRdetects that the rotation and speed do not match.
5. IoT HTRcalculates necessary speed to stop slippage.
6. IoT HTRadjusts speed to stop slippage.

Exceptions:

1. N/A

**6) Use Case:** Conductor or system administrator maintains and updates the IoT HTR.

Use Case No: 4.1.6

Primary Actors: Conductor; System administrator

Secondary Actor: Control Panel

Goal: Replace old Iot HTR build with new updated build

Preconditions: Primary Actor is logged in; Software input is compatible

Trigger: Software update is initiated by Primary Actor

Scenario:

1. Primary Actor inputs new Software into the system.
2. New Software is run.
3. Old Software is deleted.

Exceptions:

1. Failure in new Build occurs; Restart old Build

4.2 Use Case Diagram



4.3 Class-Based Modeling



4.4 CRC Modeling/Cards

| **IoT HTR** | |
| --- | --- |
| Description: Receives information received from sensors and sends it to the Control Panel accordingly. It warns and suggests the conductor about possible dangers or adjusts to the optimal speed automatically depending on the situation. In case of an emergency, immediately slow down and stop the train. | |
| **Responsibility:** | **Collaborator:** |
| start(), deactivate()  Activates and deactivates upon the conductor log in. Deactivates when wifi is connected or the conductor logs out. | Train, Control panel |
| keepCurrentSpeed()  Maintain the current speed of the train. | Sensor, Train |
| setSpeed(), increaseSpeed(), decreaseSpeed()  Adjust to the suggested speed. Increase or decrease speed to avoid collisions using throttle() and brake. | Sensor, Train |
| enterEmergency()  Slows the train immediately and makes it come to a complete stop. | Sensor, Train |
| detectDamage()  Detects any malfunctioning sensors or train components based on the data it receives or the train operation. | Sensor, Train |

| **Sensors** | |
| --- | --- |
| Description: Receive information from the sensors and send it to the IoT HTR and display it on the IoT screen for the conductor. | |
| **Responsibility:** | **Collaborator:** |
| returnInfo()  Returns its collected data to the control panel. | Control panel, IoT HTR |

| **Control Panel** | |
| --- | --- |
| Description: The Control Panel receives information from the sensors, changes the data to be more accessible for the conductor. | |
| **Responsibility:** | **Collaborator:** |
| displaySpeed(), getSpeed()  Receives the speed from the sensors and displays it for the conductor. | Sensors, IoT HTR |
| displayObjectLocation()  Receives the location of the object from the sensors and displays it. | Sensors, IoT HTR |
| displayLiving(), displayMoving()  Displays whether the object detected is alive or moving. | Sensors, IoT HTR |
| displayDistance()  Displays the distance between the train and the object detected. | Sensors, IoT HTR |
| displaySlippage()  Displays the condition of the path and the slippage amount. | Sensors, IoT HTR |
| displayGPSInfo()  Display locations of different trains. | Sensors, IoT HTR |

| **Train** | |
| --- | --- |
| Description: Adjusts the actual speed of the train based on the information given from the IoT HTR. | |
| **Responsibility:** | **Collaborator:** |
| throttle()  When the IoT HTR asks to adjust the speed, alter the speed of the train to meet the suggested speed. | IoT HTR, Sensor, Control Panel |
| sendWarning()  When living objects are detected around the train, send a sound and a red light warning. | Sensor, IoT HTR |

4.5 Activity Diagrams







4.6 Sequence Diagrams

**Non-moving Object is Detected on Track**



**Moving Object is Detected on Track**



**Sensors Detect Gate Failure**

**Sensors Detect Wheel Slippage**



**IoT Decides to Slow Down or Stop**

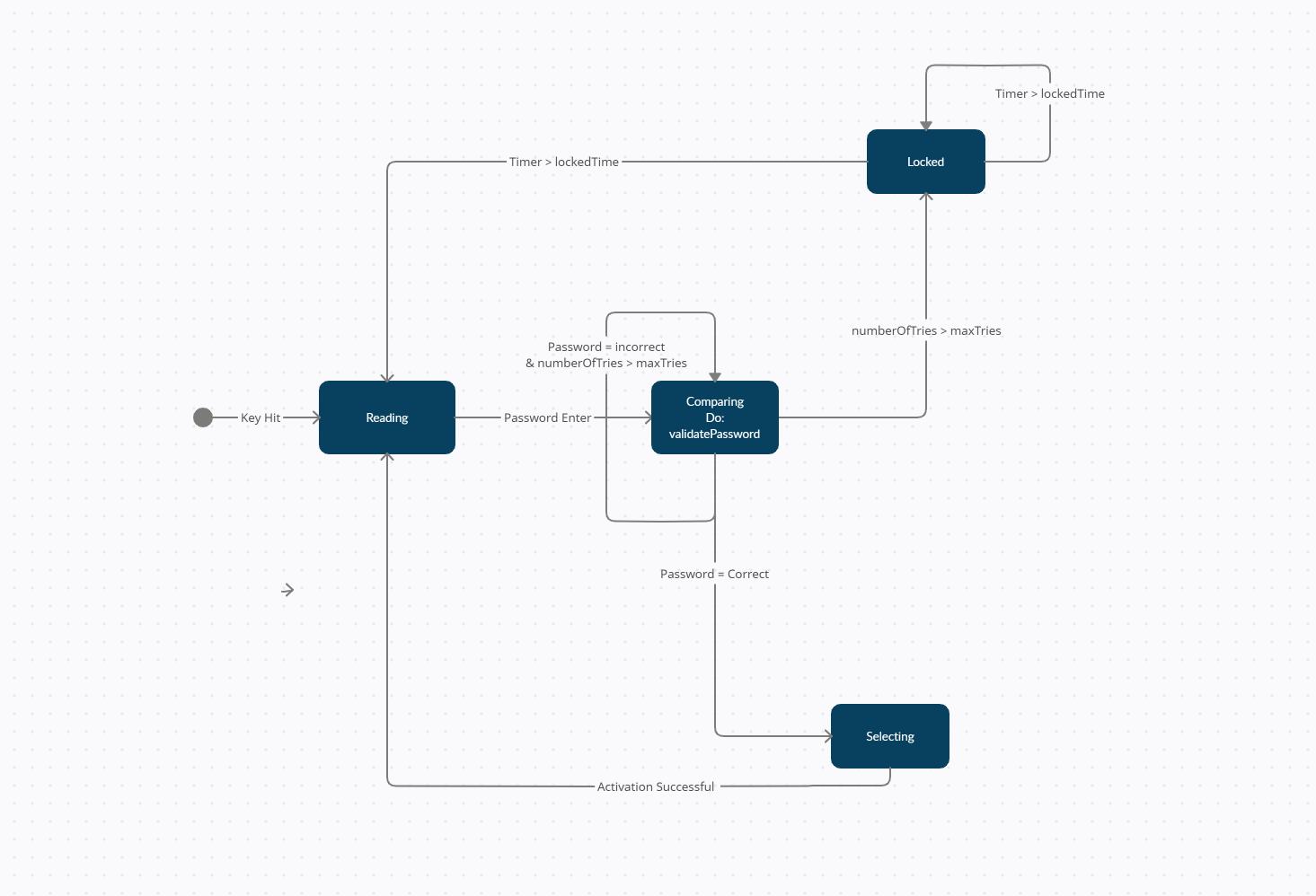


**Conductor or System Administrator Maintains and Updates the IoT HTR**

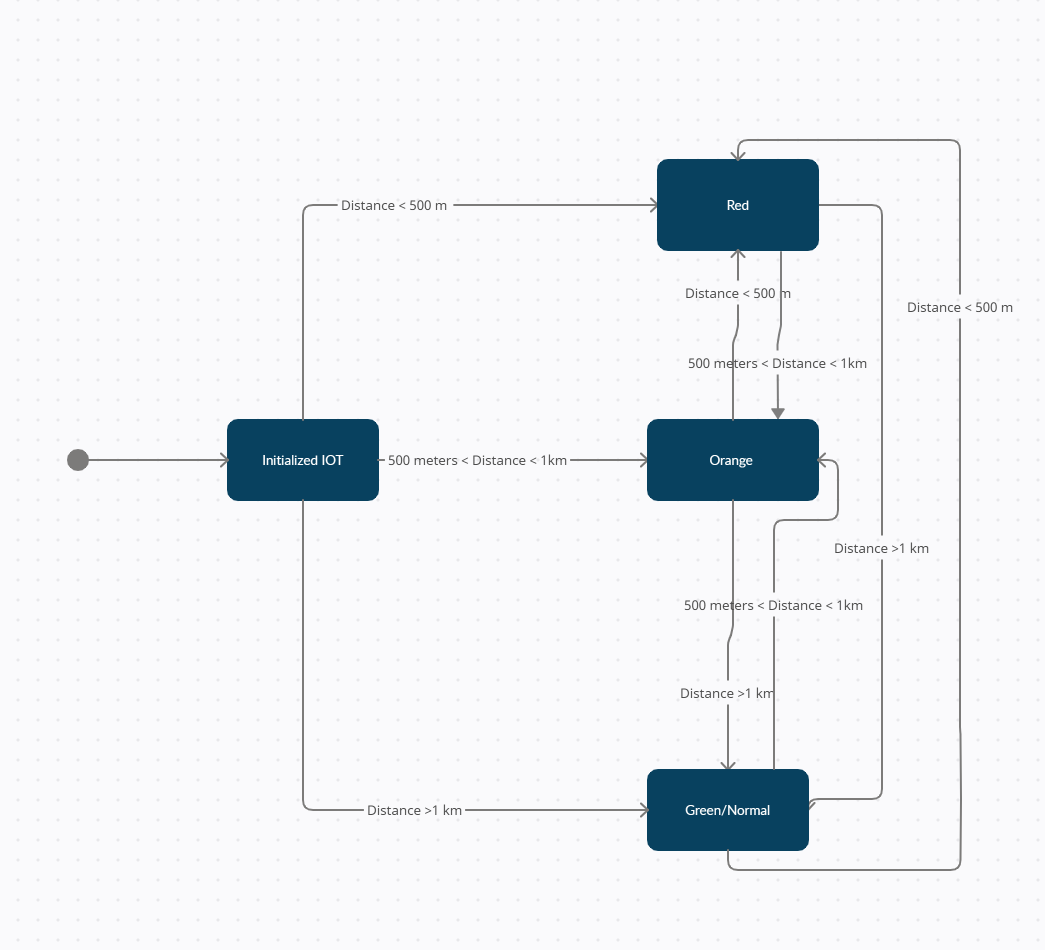


4.7 State Diagrams

System Protection:



Object Detection:



**5. Software Architecture**

5.1 Software Architecture Models

5.1.1 Data Centered Architecture - Not that feasible because we want the train to be independent of outside servers and data centers via on board sensors. Data that would increase the safety of the train are tied to static data in an external data center. Although we can utilize logs or local databases, the data centered architecture would make for slower decision making.

5.1.2 Data Flow Architecture - This is feasible because input data from sensors are to be transformed through a series of computational or manipulative components into output data.

5.1.3 Call Return Architecture - This is feasible because the train would need to have multiple subprograms running at the same time for each sensor.

5.1.4 Object-Oriented Architecture - This is feasible because the objects would be senors, TSNR, IoT HTR, and the IoT screen.

5.1.5 Layered Architecture - This is not feasible because laying and abstraction is not the best for real-time data and information. Computations inside the layer go through several abstractions in order to make real-time notifications, and therefore, it is not a good real-time implementation.

5.1.6 Model View Controller Architecture - This is feasible because it takes in an input from the sensors and produces an output that is readable.

5.2 Pros and Cons

5.2.1 Data Centered Architecture

| Pros | Cons |
| --- | --- |
| Stores the Username and Password of conductor and preferences | Slow signal from immediate onsite data from sensors on the train |

5.2.2 Data Flow Architecture

| Pros | Cons |
| --- | --- |
| Can process a large amount of data quickly | Can have high latency because each batch must be completed before the next |
| System Errors are easily fixed | Does not support dynamic interaction between filters |
| Easy to change details of program | Disturbed responses cannot be handled |

5.2.3 Call Return Architecture

| Pros | Cons |
| --- | --- |
| Modularity | As the number of sensors increase, the IoT HTR responsibility increases |
| Scalarable to an extent | Difficult to do two tasks at the same time |

5.2.4 Object-Oriented Architecture

| Pros | Cons |
| --- | --- |
| Improved maintainability | Larger program size |
| Faster development time | Slower Programs |
| Lower cost because of reusability | Not suitable for all types of problems |

5.2.5 Layered Architecture

| Pros | Cons |
| --- | --- |
| Separates problem into smaller scopes | Slow development time and more costly with too many layers |
| Provides modularity and clear interfaces | Slower performance as more layers are added |
| More testable |  |

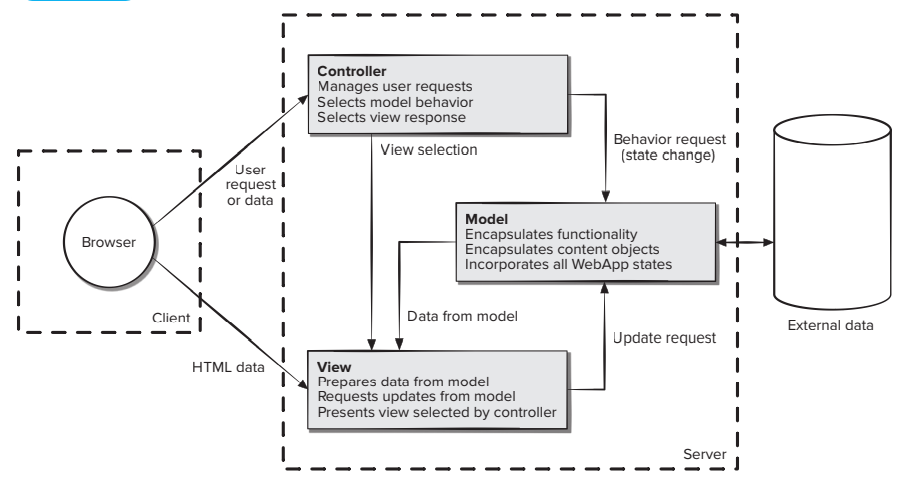
5.2.6 Model View Controller Architecture

| Pros | Cons |
| --- | --- |
| Faster development | Must have strict rules on methods |
| Easier for multiple developers to collaborate |  |
| Easier to update |  |
| Easier Debug |  |

5.3 Optimal Software Architecture for IoT HTR

5.3.1 Model View Controller Architecture (MVC architecture)

MVC architecture is the optimal software architecture for IoT HTR. The view prepares data from the model, requests updates, and presents the information on the IoT screen. The model object has access to data stored and it is organized by the appropriate view object, which would be displayed on the IoT screen. Browser is the IoT Screen. From the display, the user only signs into the controller. From there on, the controller takes over and selects the view which would be either a log view that provides the data for the user or will allow another view that allows the conductor or the administrator to see the previous log datas and the operating system for updates. In short, the view will consist of the conductor view that consists of information from the sensors, sign in view, and the update view. Controller selects which view is provided to the user. Model will be the IoT HTR, where all the functionalities/calculations happen. This calculates whether there is a gate opening, object detection, or slippage when the data is given by the sensors. The external data is the TSNR object that where the sensors sent the data to. Every snapshot of time of TNSR has the data associated with the object detection that may say 1 or 0 to show true or false for factors such as object present, slippage occurring, or gate closure problem, or etc. Other information related to these such as distance to object, RPM, or suggested speed. External data is the repeated snapshots of the information above. The data from the external data is sent to the model and finally to the view and then to the browser for the conductor to see. Every selected time, the updated data is presented to the conductor. The information is given synchronously and updated every 1 second. The example of the roles of each part in MVC architecture is shown in the figure below.



**6. Project Code**

package IoTHTR;

import java.util.\*;

import java.util.concurrent.TimeUnit;

public class ControlPanel {

public static String GPSInfo="";

public static void displaySpeed() {

System.out.println("Current speed: "+ String.format("%.2f",Wheels.currentSpeed) + " km/h");

}

public static void displayObjectLocation() {

if(!Distance.DistanceDamage) {

System.out.println("Location: [" + Distance.location + "] Distance: " + String.format("%.2f", Distance.distance) + " km");

}

else

System.out.println("Distance sensor malfunction.");

}

public static void displayMoving() {

if(!Motion.MotionDamage) {

if(Motion.object && !Heat.living && Wheels.currentSpeed > 5){

System.out.println("Warning, object detected.");

displayObjectLocation();

}

else if(Heat.living && Wheels.currentSpeed > 0){

System.out.println("Warning, living object detected.");

displayObjectLocation();

}

else{

System.out.println("No object detected.");

}

}

else

System.out.println("Motion sensor malfunction.");

}

public static void displaySlipping() {

if(!Wheels.WheelsDamage) {

if(Wheels.slippage && Wheels.currentSpeed > 10)

System.out.println("Warning: slippage detected.");

else{

System.out.println("No slippage detected.");

}

}

else

System.out.println("Wheel sensor malfunction.");

}

public static void displayGate(){

if(!Distance.DistanceDamage) {

if(Distance.gate && Wheels.currentSpeed > 0){

System.out.println("Gate malfunction detected.");

}

else{

System.out.println("No gate malfunction detected.");

}

}

else{

System.out.println("Distance sensor malfunction.");

}

}

public static void displayGPSInfo() {

if(!GPS.GPSDamage) {

if(Wheels.currentSpeed > 0){

System.out.println("Actual Speed is: " + String.format("%.2f", GPS.actualSpeed) + " km/hr");

}

}

else{

System.out.println("GPS malfunction.");

}

}

public static void main(String[] args) throws InterruptedException {

// enter password

Scanner input = new Scanner(System.in);

System.out.print("Enter Username: ");

String username = input.nextLine();

System.out.print("Enter Password: ");

String password = input.nextLine();

if(username.compareTo(IoTHTR.user) == 0){

if(password.compareTo(IoTHTR.password) == 0){

System.out.println("Loading.....");

TimeUnit.SECONDS.sleep(1);

IoTHTR.start();

}

}

else{

System.out.println("Incorrect Password or Username.");

System.exit(1);

}

while(IoTHTR.log\_in){

String log\_out = input.nextLine();

if(log\_out.compareTo("exit") == 0){

input.close();

IoTHTR.deactivate();

}

else{

//if not testing, input random values

if(!test.testing){

GPS.returnInfo();

Wheels.returnInfo();

Heat.returnInfo();

Motion.returnInfo();

Distance.returnInfo();

}

else{

test.testValues();

}

displaySpeed();

displayMoving();

displayGate();

displaySlipping();

if(Distance.DistanceDamage || GPS.GPSDamage || Heat.heatDamage || Motion.MotionDamage || Wheels.WheelsDamage){

IoTHTR.enterEmergency();

}

if(!Distance.gate && Distance.gateDistance <= 10 && !Distance.DistanceDamage){

System.out.println("-----------------------------------------------");

System.out.println("Distance to gate: " + String.format("%.2f", Distance.gateDistance) + " km");

System.out.println("Conductor, honk the horn.");

System.out.println("-----------------------------------------------");

}

if(Distance.gate && (Motion.object || Heat.living)) {

if(Distance.gateDistance > Distance.distance) {

Distance.gate = false;

}

else {

Motion.object = false;

Heat.living = false;

}

}

if(!Motion.MotionDamage && !Heat.living && Motion.object && Wheels.currentSpeed > 0 && Distance.location.equals("Front")){

System.out.println("\nReducing speed and coming to a complete stop.");

IoTHTR.setSpeed(0, Distance.distance, "object");

}

//living object detected at the front

else if(Heat.living && Wheels.currentSpeed > 0 && Distance.location.equals("Front") && !Heat.heatDamage && Heat.living\_speed == 0){

System.out.println("\nReducing speed and coming to a complete stop.");

IoTHTR.setSpeed(0, Distance.distance, "living object");

}

//living object detected at the back

else if(Heat.living && Wheels.currentSpeed > 0 && Distance.location.equals("Back") && Heat.living\_speed > 0 && !Heat.heatDamage){

System.out.println("\nIncreasing speed to avoid collision. \nLiving Object Speed: " + String.format("%.2f", Heat.living\_speed) + " km/h" );

double newSpeed = Wheels.currentSpeed + Heat.living\_speed;

if (newSpeed > 220){

newSpeed = 220;

}

IoTHTR.setSpeed(newSpeed, Distance.distance, "living object");

}

//living object detected right

else if(Heat.living && Wheels.currentSpeed > 0 && Distance.location.equals("Right")

&& Heat.living\_speed > 0 && Distance.distance > 5 && !Heat.heatDamage){

System.out.println("\nIncreasing speed to avoid collision. \nLiving Object Speed: " + String.format("%.2f", Heat.living\_speed) + " km/h" );

double newSpeed = Wheels.currentSpeed + Heat.living\_speed;

if (newSpeed > 220){

newSpeed = 220;

}

IoTHTR.setSpeed(newSpeed, Distance.distance, "living object");

}

//living object detected right & close

else if(Heat.living && Wheels.currentSpeed > 0 && Distance.location.equals("Right")

&& Heat.living\_speed > 0 && Distance.distance <= 5 && !Heat.heatDamage){

System.out.println("\nDecreasing speed to avoid collision or reduce injury. \nLiving Object Speed: " + String.format("%.2f", Heat.living\_speed) + " km/h" );

IoTHTR.setSpeed(0, Distance.distance, "living object");

}

//living object detected left

else if(Heat.living && Wheels.currentSpeed > 0 && Distance.location.equals("Left")

&& Heat.living\_speed > 0 && Distance.distance > 5 && !Heat.heatDamage){

System.out.println("\nIncreasing speed to avoid collision. \nLiving Object Speed: " + String.format("%.2f", Heat.living\_speed) + " km/h" );

double newSpeed = Wheels.currentSpeed + Heat.living\_speed;

if (newSpeed > 220){

newSpeed = 220;

}

IoTHTR.setSpeed(newSpeed, Distance.distance, "living object");

}

//living object detected left & close

else if(Heat.living && Wheels.currentSpeed > 0 && Distance.location.equals("Left")

&& Heat.living\_speed > 0 && Distance.distance <= 5 && !Heat.heatDamage){

System.out.println("\nDecreasing speed to avoid collision or reduce injury. \nLiving Object Speed: " + String.format("%.2f", Heat.living\_speed) + " km/h" );

IoTHTR.setSpeed(0, Distance.distance, "living object");

}

//gate malfunction detected

else if(Distance.gate && !Distance.DistanceDamage && !GPS.GPSDamage){

System.out.println("\nReducing speed and coming to a complete stop.");

IoTHTR.setSpeed(0, Distance.gateDistance, "gate");

}

else if(Wheels.slippage && !Wheels.WheelsDamage){

System.out.println("Slippage occurring. Reducing Speed.");

Wheels.calRPM(Wheels.currentSpeed);

IoTHTR.slippageSpeed(Wheels.RPM);

}

}

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

}

}

package IoTHTR;

public class Distance {

public static String location;

public static double distance;

public static boolean gate;

public static double gateDistance;

public static boolean DistanceDamage;

public static void reportDamage() {

int x=(int) (1000\*Math.random());

if(x==0)

DistanceDamage = true;

else

DistanceDamage = false;

}

public static void returnInfo() {

//location setup

String[] locations = {"Front", "Back", "Right", "Left"};

int index=(int) (4\*Math.random());

int ran = (int)(2\*Math.random());

int gateRan = (int) (100\*Math.random());

location = locations[index];

//distance setup

if(ran == 0)

distance= 1.5 + (10\*Math.random());//10 kilometers max

else

distance = -(10\*Math.random());

gateDistance = (30\*Math.random());

//gate setup

if(gateRan == 0){

gate = true;

}

else{

gate = false;

}

}

}

package IoTHTR;

public class GPS {

public static double actualSpeed;

public static boolean GPSDamage;

public static void reportDamage() {

int x=(int) (1000\*Math.random());

if(x==0)

GPSDamage = true;

else

GPSDamage = false;

}

public static void returnInfo() {

actualSpeed = (220\*Math.random());

}

}

package IoTHTR;

public class Heat {

public static boolean living;

public static double living\_speed; // negative number indicates moving away,

public static boolean heatDamage; // positive number indicates moving forward

public static void reportDamage() {

int x = (int) (1000\*Math.random());

if(x == 0)

heatDamage = true;

else

heatDamage = false;

}

public static void returnInfo() {

int x = (int) (10\*Math.random());

living\_speed = (100\*Math.random());

if(x==0){

living = true;

living\_speed = living\_speed \* -1;

}

else{

living = false;

}

}

}

package IoTHTR;

import java.util.concurrent.TimeUnit;

public class IoTHTR {

static public boolean log\_in = false;;

static public String user = "user";

static public String password = "password";

//LOG-IN

public static void start(){

log\_in = true;

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_IoT HTR \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Log-in Successful\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

public static void deactivate(){

log\_in = false;

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_IoT HTR \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Log-out Successful\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

//SPEED

public static void setSpeed(double speed, double distance, String s) throws InterruptedException{

if(speed > 220 || speed < 0 || distance < 0){

System.out.println("Invalid parameters");

}

else{

if(Wheels.currentSpeed == speed){

keepCurrentSpeed(speed);

}

else if(Wheels.currentSpeed < speed){

increaseSpeed(speed, distance, s);

}

else{

decreaseSpeed(speed, distance, s);

}

}

}

public static void keepCurrentSpeed(double speed){

Wheels.currentSpeed = speed;

}

public static void increaseSpeed(double speed, double distance, String s) throws InterruptedException{

double val = (speed - Wheels.currentSpeed)/(distance -0.5);

while(Wheels.currentSpeed < speed && distance > 0){

Wheels.currentSpeed+= val;

distance++;

Distance.distance = distance;

System.out.println("-----------------------------------------------");

ControlPanel.displaySpeed();

System.out.println("Distance to " + s +": " + String.format("%.2f", Distance.distance) + " km");

TimeUnit.SECONDS.sleep(1);

}

}

public static void decreaseSpeed(double speed, double distance, String s) throws InterruptedException{

double val = (Wheels.currentSpeed - speed)/(distance -0.5);

while(Wheels.currentSpeed > speed && distance > 1.5){

Wheels.currentSpeed -= val;

distance--;

Distance.distance = distance;

System.out.println("-----------------------------------------------");

if(Wheels.currentSpeed < val){

System.out.println("Current speed: 0.00 km/h");

}

else{

ControlPanel.displaySpeed();

}

System.out.println("Distance to " + s +": " + String.format("%.2f", Distance.distance) + " km");

TimeUnit.SECONDS.sleep(1);

}

}

public static void slippageSpeed(int rpm) throws InterruptedException{

//change RPM and change currentSpeed to equal to actual Speed

System.out.println("Actual Speed: " + String.format("%.2f", GPS.actualSpeed) + " km/h");

System.out.println("Current speed: " + String.format("%.2f", Wheels.currentSpeed) + " km/h");

System.out.println("Current RPM: " + rpm);

System.out.println("------------------------------------------------");

System.out.println("Adjusting RPM and current speed.");

System.out.println("------------------------------------------------");

TimeUnit.SECONDS.sleep(1);

Wheels.calRPM(GPS.actualSpeed);

Wheels.currentSpeed = GPS.actualSpeed;

System.out.println("Actual speed and current speed match");

System.out.println("Current speed: " + String.format("%.2f", Wheels.currentSpeed) + " km/h");

System.out.println("Current RPM: " + Wheels.RPM);

}

public static void enterEmergency() {

System.out.println("------------------------------------------------");

System.out.println("Entering Emergency Mode.");

System.out.println("Coming to a complete stop.");

Wheels.currentSpeed = 0;

}

}

package IoTHTR;

public class Motion {

public static boolean object;

public static boolean MotionDamage;

public static void reportDamage() {

int x=(int) (1000\*Math.random());

if(x==0)

MotionDamage = true;

else

MotionDamage = false;

}

public static void returnInfo() {

int x=(int) (10\*Math.random());

if(x==0)

object = true;

else

object = false;

}

}

package IoTHTR;

public class Wheels {

public static double currentSpeed;

public static boolean slippage;

public static int RPM;

public static boolean WheelsDamage;

public static void reportDamage() {

int x=(int) (1000\*Math.random());

if(x==0)

WheelsDamage = true;

else

WheelsDamage = false;

}

//return speed

public static void returnInfo() {

double diff = (10\*Math.random());

currentSpeed = GPS.actualSpeed + diff;

if (java.lang.Math.abs(Wheels.currentSpeed - GPS.actualSpeed) > 5){

slippage = true;

}

else{

slippage = false;

}

}

public static void calRPM(double speed){

RPM = (int)((speed/60 \* 1000)/(2\*3.14));

}

}

**7. Tests**

IoT HTR

T-1: When incorrect username or password is given, print “Incorrect username or password” and exit the program.

T-1.1 Input: username, password

Output: Incorrect username or password

T-1.2 Input: user, password

Output: Log-in Successful

T-2: When the update was not successful, print “Update failed” and exit the program.

T-2.1 Input: Old update information

Output: Update Failed

T-2.2 Input: Correct update information

Output: Update Successful

Sensors

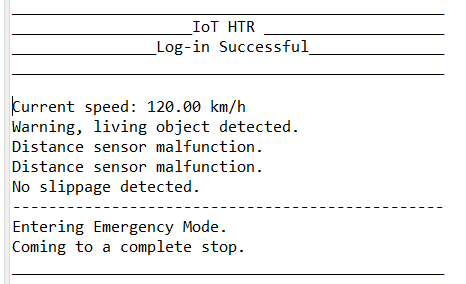
T-3: When sensors damaged, print “[Name of Sensor] sensor damaged.”

T-3.1 Input: Distance Sensor Malfunction

Output: Distance sensor damaged.

T-3.2 Input: No damaged sensors

Output: No output



T-4: When a non-living object that is bigger than 10 x 10 x 4 (in) is located in front of the train, within 10 km from the train and the train is moving faster than 25mph, send a warning message: “Warning, object detected. Location: [Front/Back] Distance: [distance] Reducing speed and coming to a complete stop.” and gradually reduce the speed so that the train will stop 500 meters away from the object.

T-4.1 Input: A object of size 14 x 20 x 8 (in) on the path of the train, 10 km away when train is going at

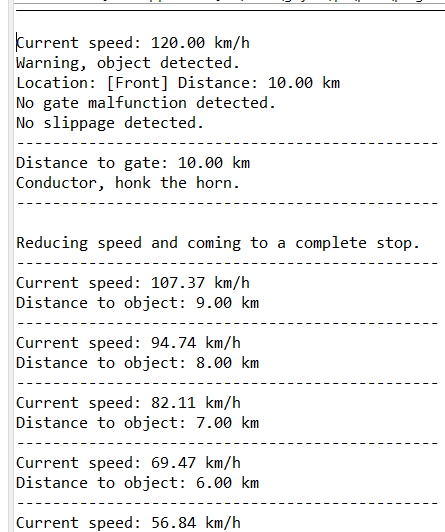
100 km/hr.

Output: Warning, object detected. Location: Front Distance: 10 km Reducing speed and coming to a complete stop.

Train: Calculate deceleration. Current speed / Distance - 500 meters

100 kilometers/hr / (10 km - .5 km) =

Speed Reduction of 10.5 km per km, 2.6 m/s



T-4.2 Input: A piece of paper on the path of the train, 3 km away when the train is going

at 20 km/hr.

Output: No output

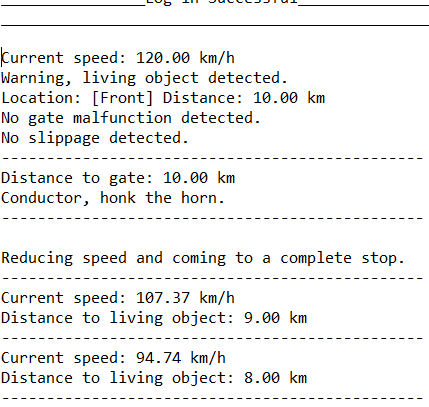
T-5: When a living object is detected within 10 km of the train and train speed is more than 0 km/hr, send a warning message: “Warning, living object detected. Location: [Front/Back/Right/Left] Distance: [distance] Object Speed: [speed] Reducing speed and coming to a complete stop.” Send a sound and light warning to the direction the object is detected. Gradually come to a stop or speed up.

T-5.1 Input: A living object not moving on the path of the train, 10 km away when train

is going at 100 km/hr

Output: Emergency, living object detected. Location: Front Distance: 10 km Coming to a complete stop in 5 seconds.

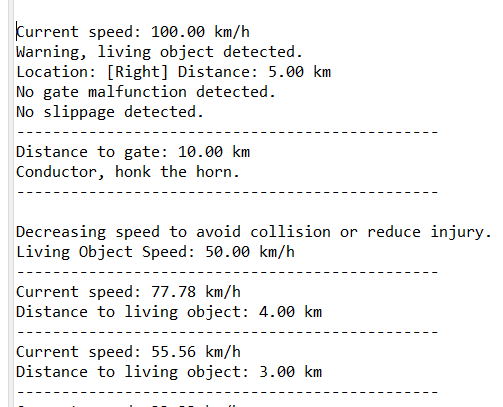
Train: Send light and sound warnings at the front.



T-5.2 Input: A living object to the right of the train, 5 km away when the train is going

at 100 km/hr.

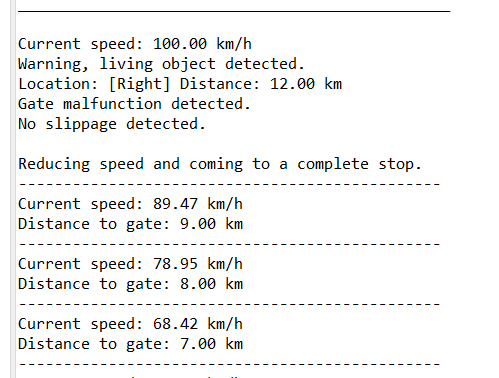
Output: Warning, living object detected. Location: Right Distance: 5km Object Speed: 5 km/hr (Positive numbers indicate moving towards the train, negative numbers indicate moving away from the train)



T-6: When gate malfunction is detected within 10 km on the path of the train and the train is under operation, send a warning message: “Warning, gate malfunction detected. Gate: [gate number] Distance: [distance] Reducing speed and coming to a complete stop.” Reduce the speed so that the train stops 1km away from the gate.

T-6.1 Input: Approaching gate reports malfunction, train is under operation and is moving at 100km/hr, gate is closed and the train is 10 km away from the gate.

Output: Warning, gate malfunction detected. Gate: 10 Distance: 10 km Reducing speed and coming to a complete stop.



T-6.2 Input: A piece of paper on the path of the train, 3 km away when the train is going

at 20 km/hr.

Output: No output

T-7: When sensors detect wheel slippage using GPS speed data and the train is under operation, send a warning message: “Warning, slippage detected.”

T-7.1 Input: abs( regular RPM - current RPM)/regular RPM > 5 %. The difference in

RPM is greater than 5 percent.

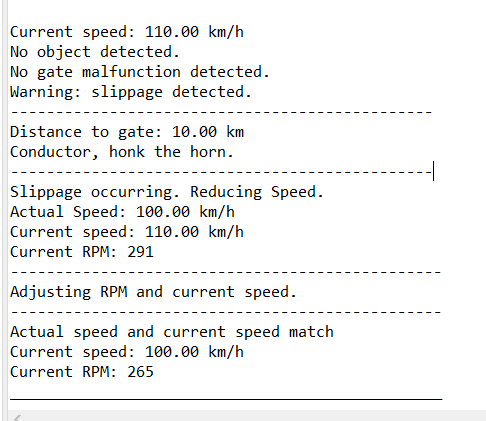
Regular RPM = 400

Current RPM = 350

Output: Warning, slippage detected.

T-7.2 Input: Regular RPM = 400 Current RPM = 401

Output: No output



TestFile

package IoTHTR;

public class test {

public static boolean testing = true;

//values can be altered for testing purposes

public static void testValues(){

//Sensor Damages

Distance.DistanceDamage = false;

GPS.GPSDamage = false;

Heat.heatDamage = false;

Motion.MotionDamage = false;

Wheels.WheelsDamage = false;

//Where the object is found; Values:Front, Back, Left, Right

Distance.location = "Right";

//Distance of object from train; Value from 1 - 10 in km

Distance.distance = 8;

//gate malfunctioning is occurring or not; values: true & false

Distance.gate = false;

//distance of the train from the gate, value from 1 - 30 (km)

Distance.gateDistance = 15;

//GPS detected speed in km/h; values between 0 - 220 (km/hr),

//similar to the current speed

GPS.actualSpeed = 100;

//whether the object detected is alive; values: true or false

Heat.living = true;

//the speed of the live object, values between 0 - 90 km/hr

Heat.living\_speed = 10;

//detects object around train; values: true and false

Motion.object = true;

//gives current speed of the train; 0 - 220 km/hr

Wheels.currentSpeed = 110;

//whether slippage occurs

Wheels.slippage = false;

}

}