Requirements Document

for

Hug the Rail IoT

Version 6.0

Prepared by Tristan Kensigner, Grace Mattern, Zachary Schuh, & Christian Szablewski-Paz

Stevens Institute of Technology

April 11, 2021

Table of Contents

Revision History

Name(s)	Date	Reason For Changes	Version
Christian	2/15	Create document/begin section 1: Intro	1.0
Grace	2/16	Revise section 1: Intro	1.0
Zach, Christian, Tristan	2/16	Organize section 1: Into paragraphs, add notes	1.0
Grace, Christian, Zach, Tristan	2/22	Add section 2: Overview	1.0
Christian	2/23	Rewrite bullet points into paragraphs	1.0
Tristan	2/23	Add section 9: Cost	1.0
Grace	3/1	Revise/organize entire document, add graphics	1.0
Tristan	3/1	Elaborate section 2.4, add graphics	1.0
Grace and Christian	3/2	Add section 3: Requirements	1.1
Grace	3/4	Revise section 3: Requirements	1.1
Grace, Tristan, Christian	3/9	Update section 3: Requirements with CTO of HTR recommended features	1.1
Grace	3/11	Update timeline, slippage system diagram, section 3: Requirements	2.0
Grace	3/16	Incorporate TA 1.0 notes, add section 4: Requirements Modeling, fix formatting	2.1
Christian, Grace, Tristan	3/17	Add use cases in section 4: Requirements Modeling	2.1
Christian	3/20	Work on model index cards	2.2
Christian, Tristan, Grace, Zach	3/21	Finish section 4: Requirements Modeling	3.0

Grace, Christian, Tristan, Zach	3/28	Make changes as directed to section 4: Requirements Modeling	4.0
Tristan, Christian, Grace	4/5	Add section 5: Software Architecture	5.0
Grace, Christian, Tristan	4/10	Revise section 5: Software Architecture	6.0

1. Introduction

1.1 Problem

Trains are a very common and popular form of transportation that require analysis of data derived from the weather conditions, fuel consumption, and time prediction, among others, in order to ensure safe and successful travel. Current trains retrieve data from servers to receive updates that are needed to make calculations for the Locomotive Control System (LCS). One major setback to this system is that trains need to be connected to servers to keep receiving updates and consequently provide safe travel. This leaves the trains vulnerable when a cellular wifi or internet connection is lost. No new data can be retrieved without this connection and thus the safety of the passengers and operator are jeopardized.

1.2 Purpose

The Internet of things (IoT) Hug the Rail (HTR) project is a system that will allow the LCS to make decisions locally in absence of cellular and wifi connectivity to back offices. For example, it will provide the operator with information regarding if the train is slipping or if there is an obstruction. Furthermore, the operator will be able to make informed decisions regarding the problems and cautions specified by the IoT HTR thereby maintaining safe operation and avoiding potentially dangerous situations. The overall purpose of this product would be to make HTR safer, less costly, and more efficient.

1.3 Team

Our dedicated team of developers are skilled in areas such as problem-solving, programming, and project management and are excited to participate in the project. Despite having little experience with embedded systems, GitLab, and IoT systems, we are driven to design a system that allows trains to efficiently operate without a dependence on a connection. Christian, Zach, and Tristan's main roles will be coders and problem solvers. Coders' responsibilities include coding, debugging and testing. Grace's role will be as a project manager due to her being highly organized and detail-oriented. The project manager's responsibilities include creating a schedule and timeline to execute each phase of the project. We hope to innovate and reimage the future of the locomotive industry. Our team will develop and implement IoT Hug the Rail from February 9th to April 20th of 2021.

1.4 Evolving Current Operations

Current operations on trains receive data from the departure station and cloud. During the travel, updated data is sent from servers in range and the operator utilizes the information to control the LCS. Once at the destination, the train, station, and cloud exchange data. The cloud communicates this information over the larger network. While the LCS has evolved over the years as it now can supply lots of information such as heat of the engine, fuel consumption, and generally offers a failsafe-like "software" to prevent the engine from operating outside of predefined safe limits.

1.5 Approach

The edge devices will capture data from other HTR locomotives and the environment instead of relying on data from servers. Many sensors will be implemented to provide crucial information that has now become unavailable (due to the lack of connection). Data from sensors will be sent to a time sensitive networking router (TSNR) which will act as a bridge between the raw sensor data and IoT HTR. IoT HTR will make any necessary calculations and display data from the sensors. IoT HTR's main goal is to make suggestions via the Display to the operator. The product will provide the capability for the operators to receive statuses and download the latest rules for operation from the Fog/Cloud onto the IoT software. Consequently, a finite state system will be created, and such features will help make the appropriate suggestions to the operator.

1.6 Timeline*

The timeline of this project will closely follow an agile methodology; throughout the development process changes will likely be made to both requirements and capabilities. Within each iteration of development, the team will work to recognize problems early and adapt to any changes while continuing to work efficiently. These include but are not limited to software design, technical requirements, deliverables, and the timeline itself. We find that this model helps provide a high-quality project that is attuned to the needs and desires of the consumers and stakeholders while also reducing the failure to analyze risks.

Week 1: 2/2 - **Communication phase** (constantly revisited, understand new desires of stakeholders)
Form teams
Decide roles and responsibilities
Week 2: 2/9 - **Planning phase** (revisited to address issues)
Understand the problem
Create documentation template
Start Section 1: Intro
Research existing systems
Week 3: 2/16 - Review/revise Section 1: Intro
Start Section 2: Overview
Research sensors that can be used to provide information to IoT HTR
Begin Section 10: Cost
Week 4: 2/23 - Familiarize team with GitLab/Git
Upload version 1.0 on GitLab
Research wheel sensors
Research front sensors
Research GPS interaction with wheel sensor
Research thermometer interaction with wheel sensor
Understand how sensors capture data from other HTR trains and surrounding environment
Week 5: 3/2 - **Requirement Analysis** (revisited if old model fails to fit new requirements)
Start Section 3: Requirements
Implement sensors that enable the IoT HTR to make suggestions
Add CTO of HTR features to Section 3
Research gate status sensors
Research hardware and OS for IoT HTR
Week 6: 3/9 - Continue Section 3
Use analytics engine to process info from the sensors to the LCS
Week 7: 3/16 - **Requirements Modeling**
Start Section 4: Use Cases
Enable operator to download the latest data from Fog/Cloud onto the IoT HTR when connected
Week 8: 3/23 - Continue Section 4
Week 9: 3/30 - Continue Section 4
Week 10: 4/1 - **Software Architecture**

Start Section 5
Week 11: 4/6 - **Coding**
Week 11: 4/13 - **Testing**
Continue coding
Begin testing
Week 11: 4/20 - **Release first iteration**
Copy code to Documentation (Section 6: Code)
Start Section 7: Testing

*Subject to change

2. Overview

2.1 Problem Statement

Current IoT technologies on trains are focused on live data received from the Central Operation Center Servers via WiFi/Cellular networks. This heavy dependence on WiFi/Cellular connectivity introduces a massive weakness regarding safety when a stable connection is lost. Therefore it is imperative that a reliable IoT system based on local data be developed to allow for the continuation of a safe trip. For a train to operate properly external data must be provided. In the case that wifi is unavailable to provide the necessary data, a backup solution must be put in place.

2.2 Data Required to Operate

IoT HTR will need to collect data regarding objects, their position relative to the train (front or back), and their movement (stationary or moving) . IoT HTR will alert the operator to slow down or stop. The train will also need information about the gate crossing's status (open or closed). If the gates are open, the train will need to slow down or halt because vehicles may be crossing the train crossing. We assume that the train will still have access to the GPS/satellite. The GPS data can coordinate with the wheel sensor data which will provide data on the wheels' spin rate (rpm). In a sense, these two data will act to verify each other since when they contradict each other it should alert the operator that the train is probably slipping.

2.3 Expected Output

The data will be accessible to IoT HTR via the TSNR, from which the operator can read the suggestions from IoT HTR Display and make the appropriate decisions. Specifically, IoT HTR will interface with sensors to collect data, analyze it, and use rule-based logic to issue recommendations or actions to the operator via the IoT HTR Display by using TSNR as a proxy. For example, if the front sensor detects an object is too close, it then will detect if it is moving. This information will be passed to TSNR which will then pass it to IoT HTR. IoT HTR will provide a hazard warning to the operator, who can then slow down or stop the train. Another example is the wheel sensor which will verify its data with the still working GPS. The GPS provides the speed the train is going while the wheel sensor will provide how fast the wheels are rotating (rpm) and thus if one of the data fails to corroborate the other, then a warning will appear on the IoT HTR Display. With the knowledge that the train is slipping from the warning message, the operator can slow down or brake.

2.4 Available Technologies

There are a multitude of different sensors available. A tachometer is the main sensor we intend to use to check for wheel slippage (along with the GPS data). Another important sensor is a radar/sonar/ultrasonic-like device that can map out the environment surrounding the train for the purpose of warning/alerting the operator of obstructions. Popular technologies include LED time of flight sensors and LiDAR sensors, the latter of which are not uncommon on cars and airplanes. A LiDAR sensor continually fires off beams of laser light and then measures how long it takes for the light to return to the sensor. In effect, a LiDAR sensor returns the distance of objects hit by the laser beam and thereby creating a 3D visualization of the environment. With continuous firing, it can also present the movement of objects surrounding the train just as it would notify a driver that a pedestrian, curb, or other vehicle is near. The GPS is a versatile technology that provides information about the position of the train. Not only is it highly reliable, but it also integrates well with the previous technologies.

The sensors for the IoT Hug the Rail Project would be mainly located in three spots. The tachometers will be located near the wheels of the train as these sensors require information from the wheel's rotation and friction. This data will provide the necessary information to IoT HTR about the train's speed, change of speed, and friction force. The front and back of the train will house sensors concerning 3D mapping for obstacle detection. Our IoT system will still have access to the GPS as it does not require a WiFi/cellular connection. The status (open/closed) of cross gates will be handled by object recognition cameras that continuously compare photos taken with a library of reference photos.

2.5 Systems

The IoT HTR system will collect data retrieved from the various sensors and interpret them accordingly. Results (i.e. weather, speed, obstructions, etc.) will be available via the IoT HTR Display and suggest operational changes to the operator based on its detections. Below are some conceptual architectures for our IoT. **Figure 1** and **Figure 2** communicate the general problem and solutions. **Figure 3** shows how IoT HTR would be used specifically regarding weather based slippage.

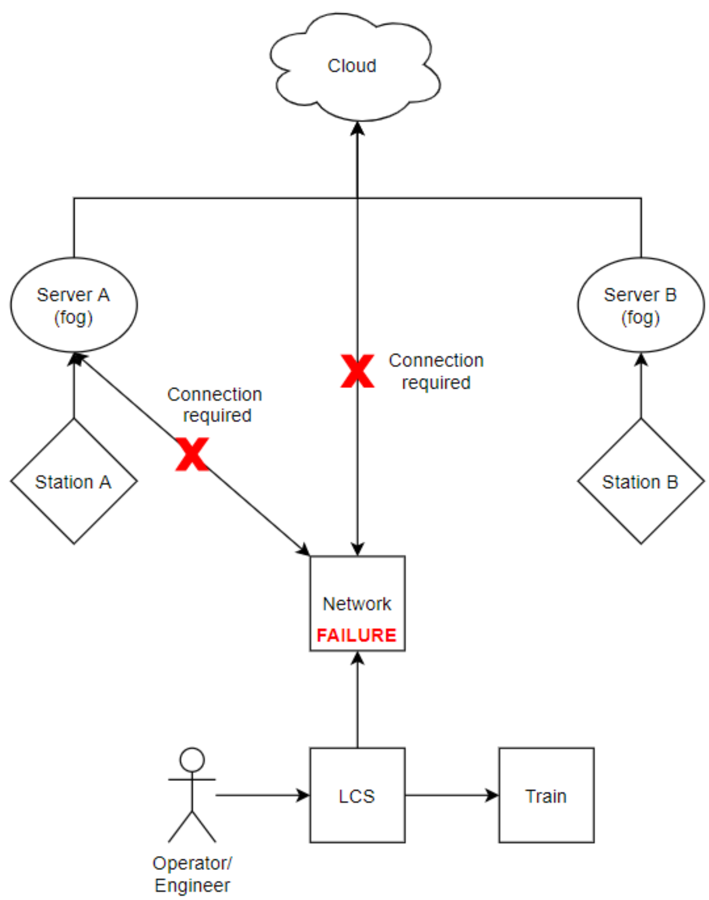


Figure 1: *shows the problems that occur if a connection is lost (represented by the red x's). Supposing the train starts in the range of station A, the LCS is connected to the network which receives data from server A and also the cloud. However when the connection is lost no new data is retrieved from server A or the cloud and the network fails.*

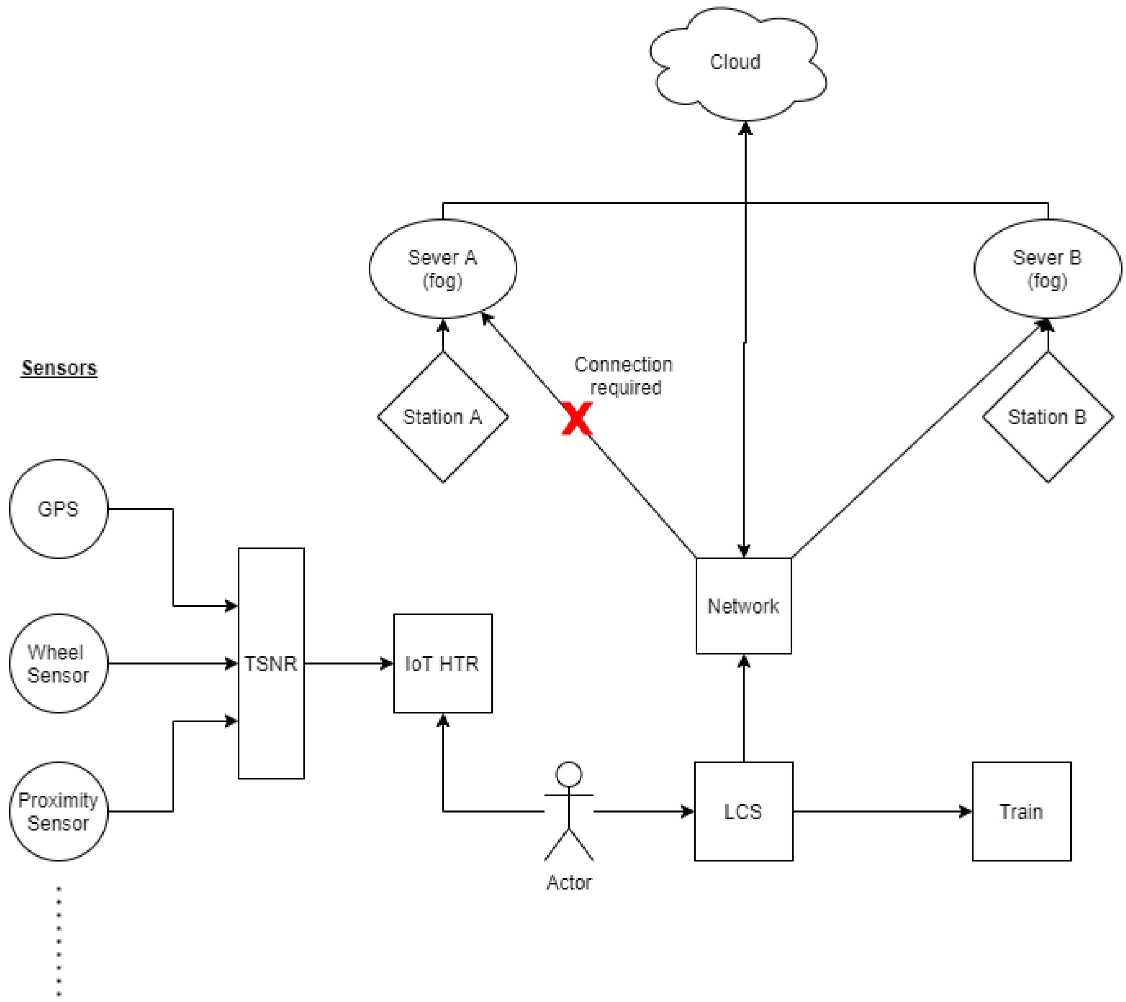


Figure 2: *shows the general solution IoT Hug the Rail Project will provide. Supposing the train is headed to station B without a connection, information from the GPS, proximity sensor, wheel sensor, and other sensors are passed through the TSNR and processed by the IoT HTR. IoT HTR displays the data and any warnings which are read by the operator who can then make the appropriate decisions. Once at station B, a connection is reestablished. The train, server B, and the cloud share data.*

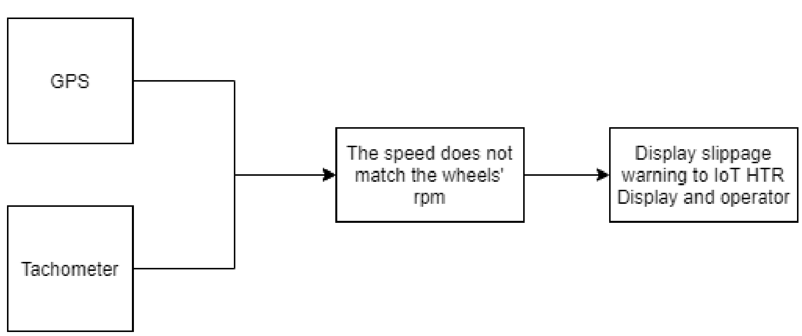


Figure 3: *The GPS could report the speed of the train but the tachometer would contradict the data by showing that the wheels are rotating faster than the given speed. The following data will be available to the operator on the IoT HTR Display.*

3. Requirements

3.1 Non-Functional Requirements

3.1.1 Security

R-1: IoT HTR shall only be accessed via user ID and password.
R-2: IoT HTR Admin shall have secured (admin ID/Password) access to all sensors and equipment.
R-3: Operators shall only see the operations of the train on the IoT HTR Display .
R-4: Only Admins shall have access to raw data and logs.
R-5: Only Admins shall have access to system configurations.
R-6: Only Admins shall be able to download/update the software.
R-7: Only Admins shall be able to resolve the IoT HTR if it fails.
R-8: IoT HTR network shall be secured by a LoRaWan Protocol.
R-9: IoT HTR shall endode all user-supplied output.
R-10: IoT HTR shall lock the system after 5 failed attempts to login.
R-11: IoT HTR shall unlock after 5 minutes.
R-12: Only Operators and Admins shall have access to capabilities listed in 3.2.

3.1.2 Performance

R-13: IoT HTR sensors shall process an event within 0.5 seconds.
R-14: TSNR shall send sensor data to IoT HTR within 0.5 seconds.
R-15: IoT HTR shall process an event within 0.5 seconds.
R-16: IoT HTR shall process 1,000 events per second without delay.
R-17: IoT HTR shall display processed events within 0.1 seconds.
R-18. The Log will log events every 0.01 seconds.

3.1.3 Reliability

R-19: IoT HTR shall operate with no failures 99.99% of the time.
R-20: IoT HTR shall alert the Operator how fast obstructions are moving with 95% accuracy.
R-21: The precision of distance-based calculations shall be at least 0.000001.
R-22: IoT HTR defect rate shall be less than 1 failure per 1000 hours of standard operation.
R-23: IoT HTR shall activate within 100 ms after the power button is pressed.
R-24: IoT HTR shall deactivate within 100 ms after the power button is pressed.
R-25: IoT HTR shall restart within 100 ms after the power button has been pressed for over 5 seconds.
R-26: IoT HTR shall meet or exceed 99.9% uptime.
R-27: IoT HTR sensor failures shall be identified in 0.01 seconds.
R-28: IoT HTR system failures shall be identified in 0.01 seconds.
R-29: The network of sensors shall be operational at least 99.99% of the time.

3.1.4 Other Non-Functional Requirements

R-30: IoT HTR shall be equipped with a touch screen display.
R-31: IoT HTR shall be equipped with a power button.
R-32: IoT HTR shall be equipped with an auxiliary storage device input.
R-33: IoT HTR shall be able to turn on/off regardless of the status (on/off) of the HTR train.
R-34: Operators and admins shall be able to start and stop the IoT HTR via a power button.

R-35: IoT HTR shall be able to restart when the power button is held down for over 5 seconds.
R-36: The Operator shall be prompted with a login screen after pressing on the power button.
R-37: The Display shall show recommendations from the IoT HTR.
R-38: The Display shall show functionalities as stated by the IoT HTR requirements.
R-39: The Display will display a continue normal operations message if no issues are presented by all the sensors.
R-40: IoT HTR shall send a warning message in the event of hardware or software failure.
R-41: The Log continuously logs data and sends it to a single log file to keep track of all events.
R-42: IoT HTR shall have a clock to attach timestamps to the log file entries.
R-43: IoT HTR shall be able to download data while at a station.
R-44: IoT HTR shall be able to upload operational data when at a station.

3.2 Functional Requirements

3.2.1 Standing Objects

R-45: There shall be four cameras, two in front and two in the back of the train.
R-46: There shall be two LiDAR sensors, one in the front and one in the back of the train.
R-47: The Cameras shall work with LiDAR sensors to take pictures of the environment.
R-48: The LiDAR sensors shall continuously take pictures to map the surroundings.
R-49: The Cameras shall provide obstruction pictures to the TSNR every 0.05 seconds.
R-50: The LiDAR sensors shall provide the distance of obstructions to the TSNR every 0.05 seconds.
R-51: IoT HTR shall receive obstruction data from the TSNR.
R-52: IoT HTR shall determine the position (front or back) of the obstruction to the HTR train.
R-53: IoT HTR shall display a warning message if an obstruction is detected via the Display.
R-54: IoT HTR shall display the distance of standing objects via the Display.
R-55: IoT HTR shall display that the object is moving at 0mph (as it is stationary) via the Display.
R-56: IoT HTR shall display if the standing object is in front or behind the train via the Display.
R-57: IoT HTR shall suggest to the Operator to slow down if there is an object within a 2.0 mile radius of the train via the Display.
R-58: IoT HTR shall suggest to the Operator to brake if there is an object within a 1.0 mile radius of the train via the Display.
R-59: IoT HTR shall suggest to the Operator to speed up when there is no object within a 2.0 mile radius of the train via the Display.

3.2.2 Moving Objects

R-60: There shall be four cameras, two in front and two in the back of the train.
R-61: There shall be two LiDAR sensors, one in the front and one in the back of the train.
R-62: There shall be four motion detection sensors distributed equally along the train's length.
R-63: The Cameras shall work with LiDAR sensors to take pictures of the environment.
R-64: The LiDAR sensors shall continuously take pictures and map the surroundings.
R-65: The Motion detection sensors shall continuously detect infrared energy (heat).
R-66: The Cameras shall provide obstruction pictures to the TSNR every 0.05 seconds
R-67: The LiDAR sensors shall provide the distance of obstructions to the TSNR every 0.05 seconds
R-68: The Motion detection sensors shall provide the speed of obstructions to TSNR every 0.05 seconds.

R-69: IoT HTR shall receive obstruction data from the TSNR.
R-70: IoT HTR shall determine the position (front or back) of the obstruction to the HTR train.
R-71: IoT HTR shall display a warning message if an obstruction is detected via the Display.
R-72: IoT HTR shall display the distance of moving objects via the Display.
R-73: IoT HTR shall display if the moving object is behind or in front of the train via the Display.
R-74: IoT HTR shall display the speed of the moving object via the Display.
R-75: IoT HTR shall suggest to the Operator to slow down if there is an object within a 2.0 mile radius of the train via the Display.
R-76: IoT HTR shall suggest to the Operator to brake if there is an object within a 1.0 mile radius of the train via the Display.
R-77: IoT HTR shall suggest to the Operator to speed up when there is no object within a 2.0 mile radius of the train via the Display.

3.2.3 Gate Crossings

A closed gate status means cars should stop at the railroad crossing as a train is approaching.
An opened gate means that cars can drive over the railroad crossing as no train is approaching.

R-78: There shall be one radar sensor located in the front of the train.
R-79: There shall be four cameras, two in the front and two in the back of the train.
R-80: The Radar sensors shall continuously emit radio waves and checks the change in frequency after omission.
R-81: The Cameras shall continuously take pictures to compare with a set of photos of gate crossing (opened and closed).
R-82: The Radar sensors shall provide the distance of a gate crossing to the TSNR every 0.05 seconds.
R-83: The Cameras shall provide the gate's status to the TSNR every 0.05 seconds.
R-84: IoT HTR shall receive gate crossing data from the TSNR.
R-85: IoT HTR shall display a warning message if the gate is open via the Display.
R-86: IoT HTR shall display the distance of the open gate crossing via the Display.
R-87: IoT HTR shall suggest to the Operator to slow down if the gate is open within a 3.0-mile Radius via the Display.
R-88: IoT HTR shall suggest to the Operator to brake if the gate is open within a 2.0-mile radius via the Display.
R-89: IoT HTR shall suggest to the Operator to speed up/maintain current speed when the gate is closed via the Display.

3.2.4 Wheel Slippage

R-90: IoT HTR is equipped with four tachometers on the four outermost train wheels.
R-91: IoT HTR is equipped with one GPS.
R-92: The Tachometers shall continuously measure the rotation speed of the train wheel in rpm.
R-93: The GPS shall continuously send and receive signals to calculate the distance and position of the train.
R-94: The Tachometers shall provide the wheels' rpm to the TSNR every 0.05 seconds.
R-95: The GPS shall provide the speed of the HTR train to the TSNR every 0.01 seconds.
R-96: IoT HTR shall receive wheel slippage data from the TSNR.
R-97: IoT HTR shall calculate the wheel rpm of the HTR train from the GPS data.
R-98: IoT HTR shall calculate the difference in the wheels' rpm given by the GPS and tachometer.
R-99: IoT HTR shall display a warning message if the train's wheels are slipping via the Display.
R-100: IoT HTR shall suggest to the Operator to slow down if the train is minorly slipping (5mph <

Δspeed < 10mph) via the Display.
R-101: IoT HTR shall suggest to the Operator to brake if the train is slipping severely (Δspeed > 10mph) via the Display.
R-102: IoT HTR shall suggest to the Operator to speed up with caution if slipping stops via the Display.

3.3 Hardware & Operating System

R-103: IoT HTR hardware shall be able to support at least 1,000 sensors.
R-104: IoT HTR shall support 5TB of data every day.
R-105: IoT HTR shall be able to import and export information obtained from operating.
R-106: IoT HTR shall be updated yearly with patches in between.

3.3.1 Operating System

R-107: IoT HTR shall be equipped with Microsoft Windows 10 IoT Enterprise
R-108: IoT HTR shall be equipped with Microsoft Windows 10, 64-bit

3.3.2 Processor

R-109: IoT HTR shall be equipped with ARM processor

3.3.3 RAM

R-110: IoT HTR shall be equipped with 2GB of RAM

3.3.4 Graphics Card

R-111: IoT HTR shall be equipped with a GPU that supports DirectX 9

3.3.5 Storage

R-112: IoT HTR shall be equipped with 32GB of storage

3.3.6 Sensors

R-113: IoT HTR shall be equipped with a Tachometer - measures the rpm of a wheel
R-114: IoT HTR shall be equipped with LiDAR sensors - maps surrounding environment
R-115: IoT HTR shall be equipped with Radar sensors - detect is the status of gate crossing
R-116: IoT HTR shall be equipped with Cameras - for gate status and LiDAR
R-117: IoT HTR shall be equipped with a GPS - give global positioning
R-118: IoT HTR shall be equipped with Motion detection sensors - for LiDAR and moving objects

4. Requirements Modeling

4.1 Use Cases

Yellow warning messages mean slow down.
Red warning messages mean stop and apply brakes.
If no warnings are detected/displayed, normal operations will be displayed.

Use case 1: Initialize IoT HTR
Primary actor: Operator
Secondary actors: IoT HTR
Goal in context: Turn on IoT HTR
Preconditions: IoT HTR is off
Trigger: Operator decides to initialize IoT HTR, that is, to press the power button
Scenario:

1. Operator presses the power button
2. Log begins to record events starting at the time of initialization
3. Display turns on
4. IoT HTR displays the login screen
5. Operator types in ID and password
6. IoT HTR displays operations of the HTR train

Exceptions:

1. Incorrect login: Warning message “Incorrect user ID or password” is displayed, Operator re-enters correct ID/password
2. Failed to log in five times: Warning message “Incorrect user ID or password. Wait 5 minutes” is displayed
3. Fails to initialize: Display does not turn on, abort use case. Admin is required to fix the issue(s)

Use case 2: Initialized IoT HTR with admin privileges
Primary actor: Admin
Secondary actors: IoT HTR
Goal in context: Turn on IoT HTR
Preconditions: IoT HTR is off
Trigger: Admin decides to initialize IoT HTR, that is, to press the power button
Scenario:

1. Admin presses the power button
2. Log begins to record events starting at the time of initialization
3. IoT HTR touch screen display turns on
4. IoT HTR displays the login screen
5. Admin types in admin ID and password
6. IoT HTR displays system configurations (software os)

Exceptions:

1. Incorrect login: Warning message “Incorrect user ID or password” is displayed, Admin re-enters correct ID/password
2. Failed to log in five times: Warning message “Incorrect user ID or password. Wait 5 minutes” is displayed
3. Fails to initialize: Display does not turn on, abort use case. Admin is required to fix the issue(s)

Use case 3: Obstruction detection
Primary actor: Proximity sensors
Secondary actors: IoT HTR
Goal in context: Notify Operator for any encounters of obstructions
Preconditions: IoT HTR is initialized
Trigger: Proximity sensors detect that there is an obstruction
Scenario:

1. LiDAR sensors detect obstructions that are within 2 miles and reports it to TSNR
2. Motion detection sensors detect the speed of the obstruction and reports it to TSNR
3. Cameras report pictures of the detected obstructions to TSNR
4. TSNR sends collected data to IoT HTR.
5. IoT HTR determines if the obstruction pictures were taken by front or back Cameras.
6. The Display displays if there is an obstruction.
7. The Display displays the speed of the obstruction.
8. The Display displays the position of the obstruction.
9. The Display displays the distance of the obstruction.
10. The Display displays a yellow warning message when the obstruction's distance is between 1 mile and 2 miles (1 mile < distance < 2 miles).
11. The Display displays a red warning message when the obstruction's distance is less than 1 mile (distance < 1 mile).
12. Log records obstructions.

Use case 4: Gate crossing status detection
Primary actor: Gate crossing sensors
Secondary actors: IoT HTR
Goal in context: Notify Operator of the gate status
Preconditions: IoT HTR is initialized
Trigger: Gate crossing gate based sensors detect the status of gate crossing
Scenario:

1. Radar sensors detect closed gates that are within 2 miles and reports it to the TSNR
2. Cameras send pictures of detected closed gates to the TSNR
3. TSNR sends collected data to IoT HTR.
4. The Display displays the status (opened/closed) of the gate crossing
5. The Display displays the distance of the open gate.
6. The Display displays a yellow warning message when the open gate's distance is between 2 miles and 3 miles (2 miles < distance < 3 miles).
7. The Display displays a red warning message when the open gate's distance is less than 2 mile (distance < 2 mile).
8. Log records gate status.

Use case 5: Wheel slippage detection
Primary actor: Wheel sensors
Secondary actors: IoT HTR
Goal in context: Notify Operator for any encounters of wheel slippage

Preconditions: IoT HTR is initialized.
Trigger: Wheel based sensor detects wheel slippage
Scenario:

1. The tachometer detects the rpm of the HTR train wheels and reports it to TSNR.
2. GPS detects the speed of the HTR train and reports it to TSNR
3. TSNR sends collected data to IoT HTR.
4. IoT HTR calculates wheel rpm of the HTR train from GPS data.
5. IoT HTR calculates the difference in wheel rpm given by the tachometer and GPS.
6. The Display will display the difference in the speed of the wheels (wheel slippage)
7. The Display displays a yellow warning message when the difference is between 5mph and 10mph (5mph < Δspeed < 10mph).
8. The Display displays a red warning message when the difference is between greater than 10mph (Δspeed > 10mph).
9. The Display will suggest to the conductor to accelerate with caution once the wheel slippage has stopped.
10. Log records slippage

Use case 6: Uninitialize IoT HTR
Primary actor: Operator
Secondary actors: IoT HTR
Goal in context: Turn off IoT HTR
Preconditions: IoT HTR is on
Trigger: Operator decides to uninitialize IoT HTR, that is, to press the power button
Scenario:

1. Operator presses the power button
2. IoT HTR displays the "Do you want to turn the power off" message
3. Operator presses "yes"
4. IoT HTR timestamps when IoT HTR was uninitiated into the logs
5. IoT HTR Display turns off

Exceptions:

1. "No" is selected: IoT HTR closes the message and continues to display the operations of the HTR train

---

Use case 7: Download logs
Primary actor: Admin
Secondary actors: IoT HTR
Goal in context: To check and download operational logs
Preconditions: There is data in the logs, that is, IoT HTR is been initialized at some point; IoT HTR is initialized
Trigger: Admin decides to download the operational logs to check data, that is, to connect an auxiliary storage device
Scenario:

1. Admin connects an auxiliary storage device to IoT HTR
2. IoT HTR detects an auxiliary storage device
3. IoT HTR request for admin ID/password
4. IoT HTR requests log(s) from log
5. Log copies the log(s) onto the storage device

6. IoT HTR displays “Log file(s) was successfully copied” message

Exceptions:

1. Incorrect admin login: Warning message “Incorrect user ID or password” is displayed, Operator re-enters
2. Failed admin login five times: Warning message “Incorrect admin ID or password. Wait 5 minutes” is displayed
3. Error copying log file(s) onto the auxiliary storage device: Warning message “Error copying log file(s) onto storage device” is displayed, abort use case

---

Use case 8:	Change configurations
Primary actor:	Admin
Secondary actors:	IoT HTR
Goal in context:	Change the configuration of the HTR train
Preconditions:	HTR train has IoT HTR system installed, IoT HTR is initialized
Trigger:	Admin decides to change the configurations of the HTR train

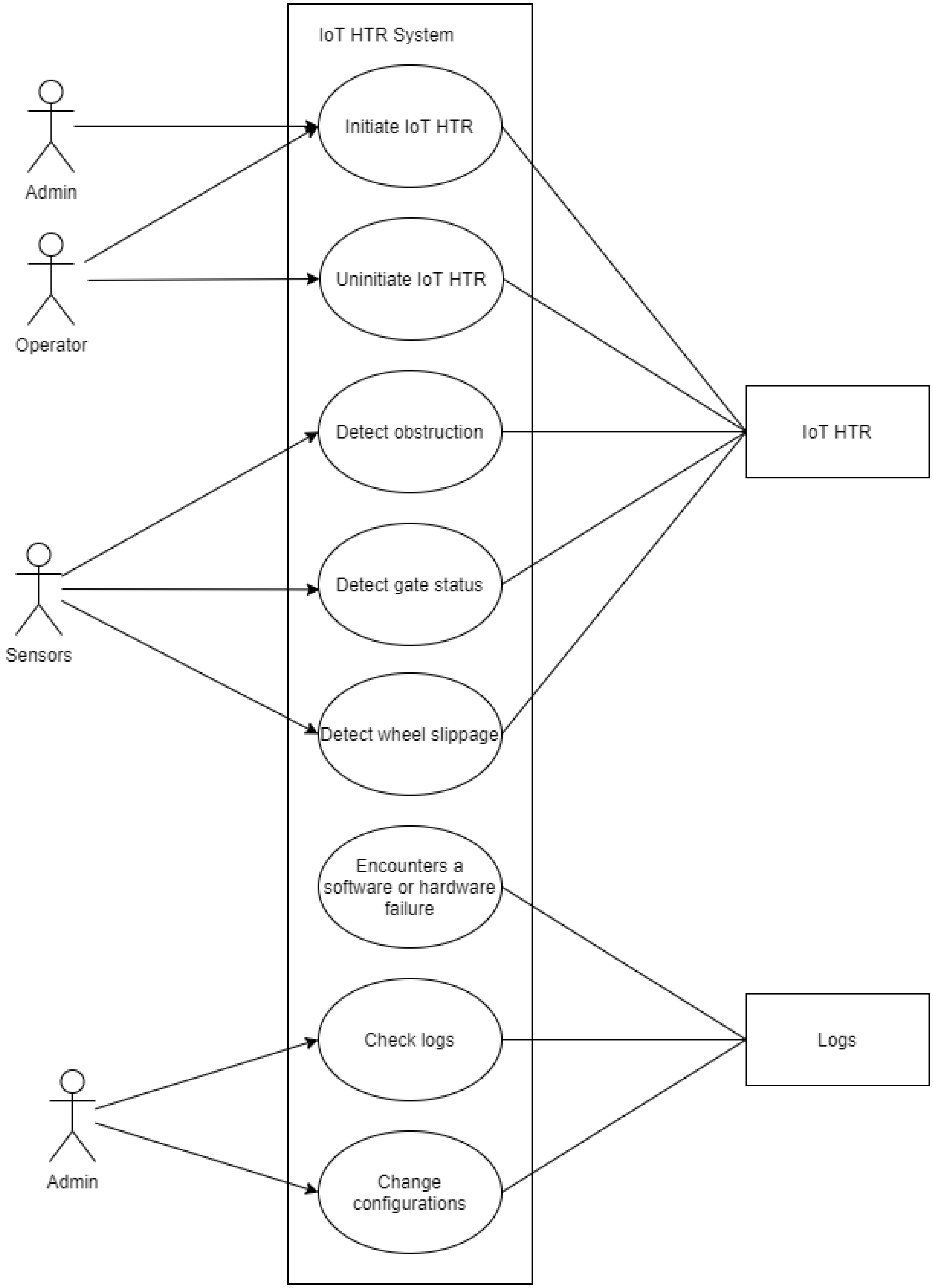
Scenario:

1. Admin supplies correct admin ID/password to IoT HTR
2. IoT HTR prompts Admin if they would like to make changes to configurations
3. Admin selects “yes”
4. Admin makes appropriate changes to IoT HTR
5. Log records changes.
6. Admin confirms changes and updates the system with new changes
7. Admin closes the configuration interface

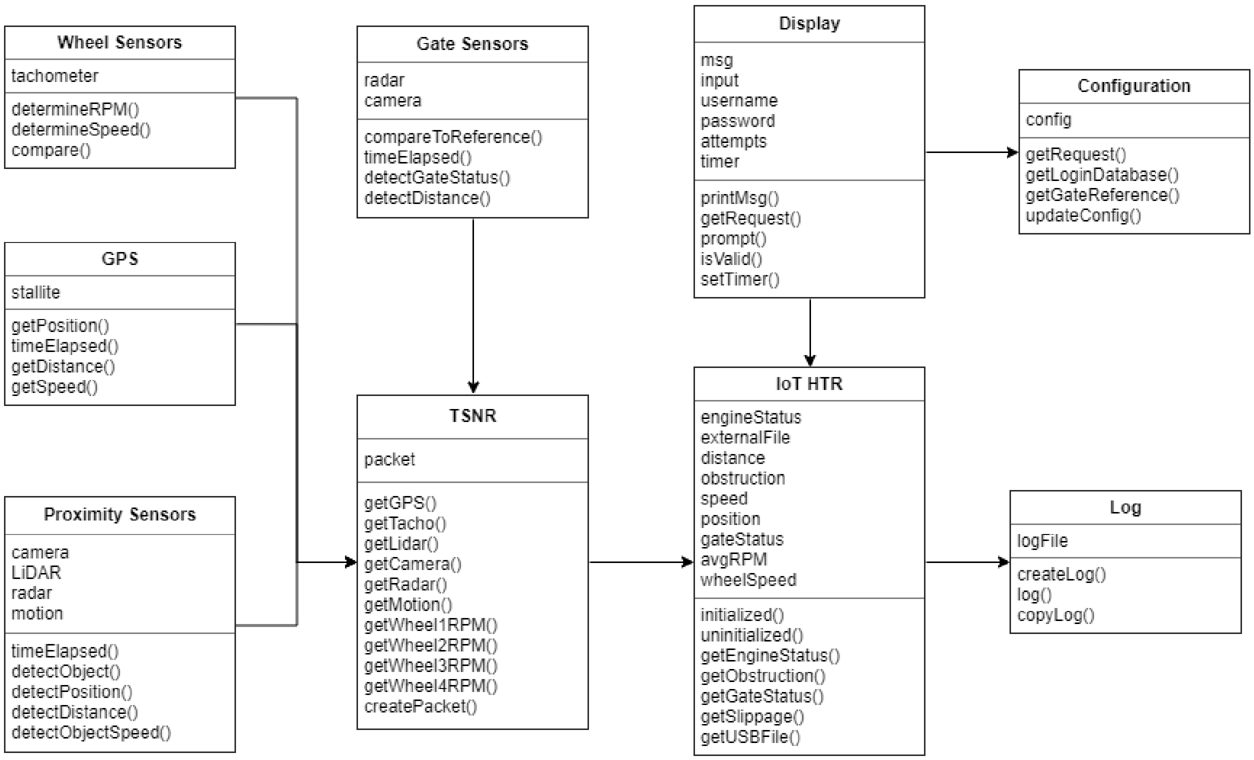
Exceptions:

1. Incorrect login: Warning message “Incorrect user ID or password” is displayed, Operator re-enters correct ID/password.
2. Failed to log in five times: Warning message “Incorrect user ID or password. Wait 5 minutes” is displayed.
3. Fail to make configurations: Warning message “Failed to configure” is displayed, abort use case.
4. “No” is selected: IoT HTR closes the configuration interface and continues to display the operations of the HTR train.

4.2 Use Case Diagram



4.3 Class Diagram



4.4 CRC Model Index Cards

Operator	
User who has limited access to IoT HTR	
Responsibility	Collaborator
Initiate IoT HTR	IoT HTR
Uninitiate IoT HTR	

Admin	
User who has complete access to IoT HTR	
Responsibility	Collaborator
Check logs	Log
Change configurations	Configurations

IoT HTR	
Takes in all the data from the TSNR, displays it, and determines if the operator needs to be alerted about hazards	
Responsibility	Collaborator
Initiate IoT HTR	
Uninitate IoT HTR	
Continuously log	Log
Display obstructions if present	Proximity sensors via TSNR
Display speed	GPS via TSNR
Display weather	Weather sensors via TSNR
Display slippage if present	Wheel sensors via TSNR
Display gate status when applicable	Gate sensors via TSNR

TSNR	
Takes in all the data from the sensors, processes it, and sends it to IoT HTR	
Responsibility	Collaborator
Send proximity (obstruction) data	Proximity sensors
Send global positioning data	GPS
Send weather data	Weather sensors
Send wheel slippage data	Wheel sensors
Send gate crossing data	Gate sensors

Proximity sensors	
Sensors that can detect objects (stationary and moving) within a 2-mile radius of the train	
Responsibility	Collaborator

Check if there are stationary obstructions	IoT HTR
Check if the stationary object is in the front or back of the HTR train	
Display the distance of stationary objects	
Check if there are moving obstructions	
Check if the moving object is in front or back of the HTR train	
Display the distance of moving objects	
Display the speed of obstructions	
Display warning message for obstructions and their position relative to the HTR train	
Display suggestion	

Wheel sensors	
Sensors that can detect if the train's wheels are slipping	
Responsibility	Collaborator
Determine the rpm of the train's wheels	IoT HTR
Determine the speed based on the wheel's rpm	
Compare wheel sensor's speed with GPS's speed to check for slippage	
Display warning message for slippage	
Display suggestion	

Gate sensors	
Sensors that can detect a gate crossing' status within a 3-mile radius of the train	
Responsibility	Collaborator
Determines if gate crossings are opened/closed	IoT HTR
Determine the distance of a gate crossing	

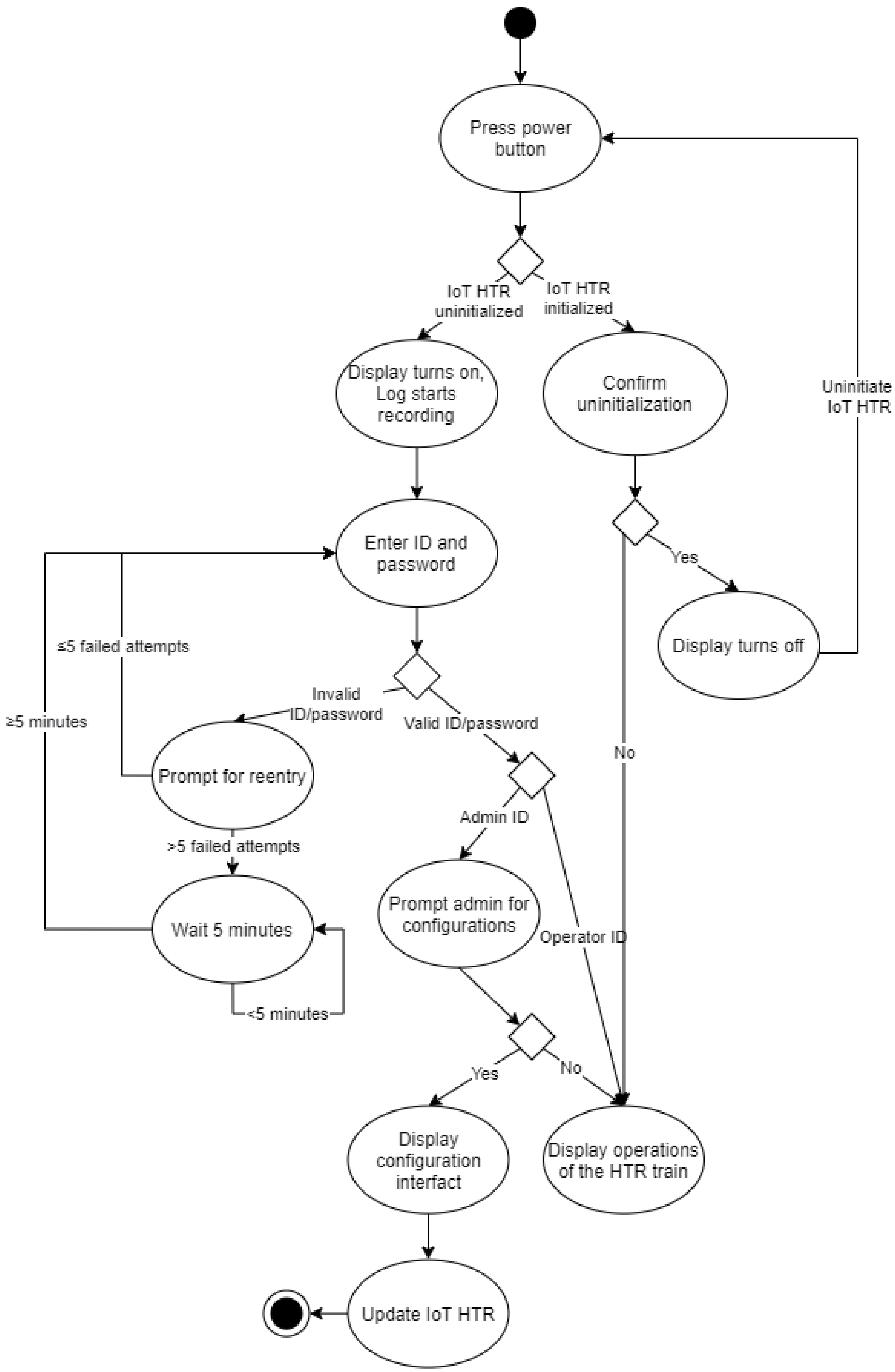
Display warning message for open gate crossing	
Display suggestion	

GPS	
Gets the global positioning of the HTR train	
Responsibility	Collaborator
Get the speed of the HTR train	IoT HTR

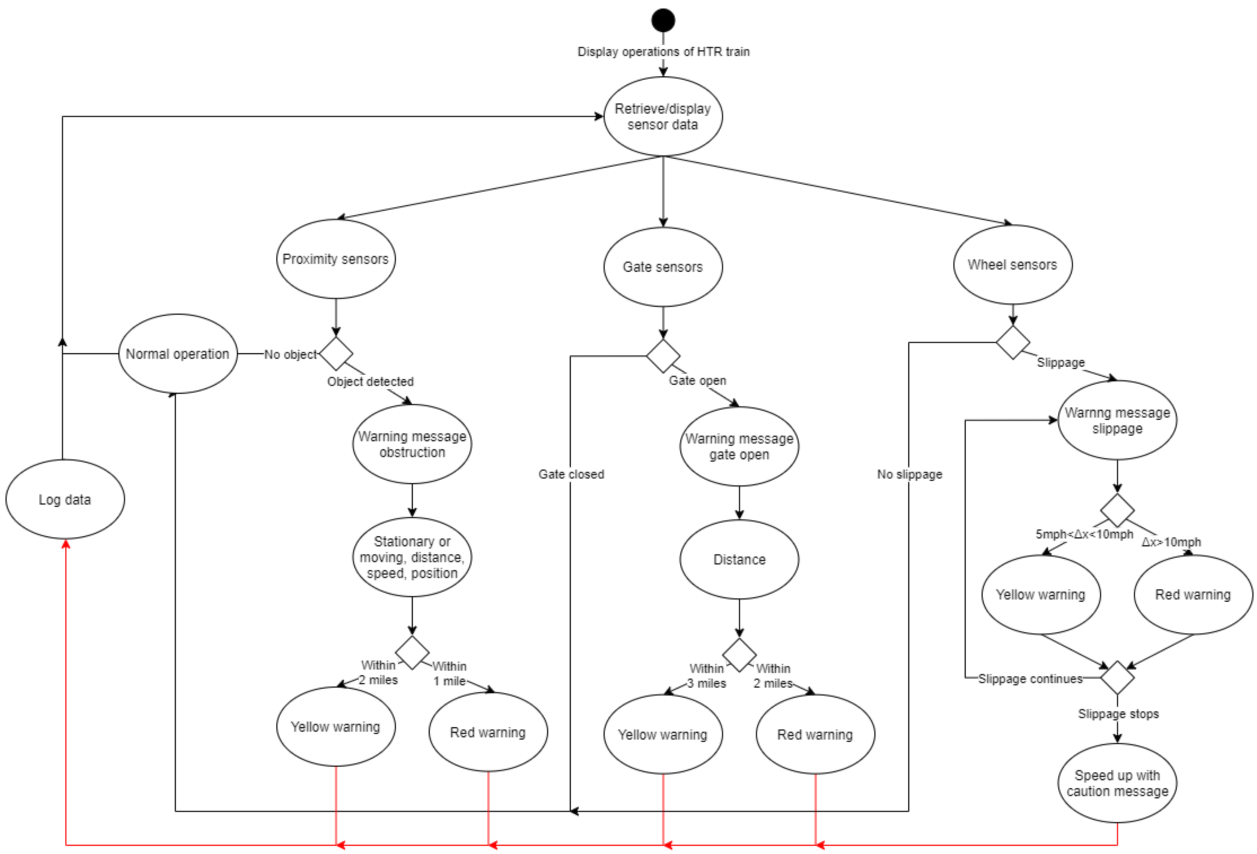
Log	
Logs all events while IoT HTR is on	
Responsibility	Collaborator
Get current time Log events	IoT HTR

Configurations	
User configurations for IoT HTR	
Responsibility	Collaborator
Read admin input to change/update system settings	IoT HTR

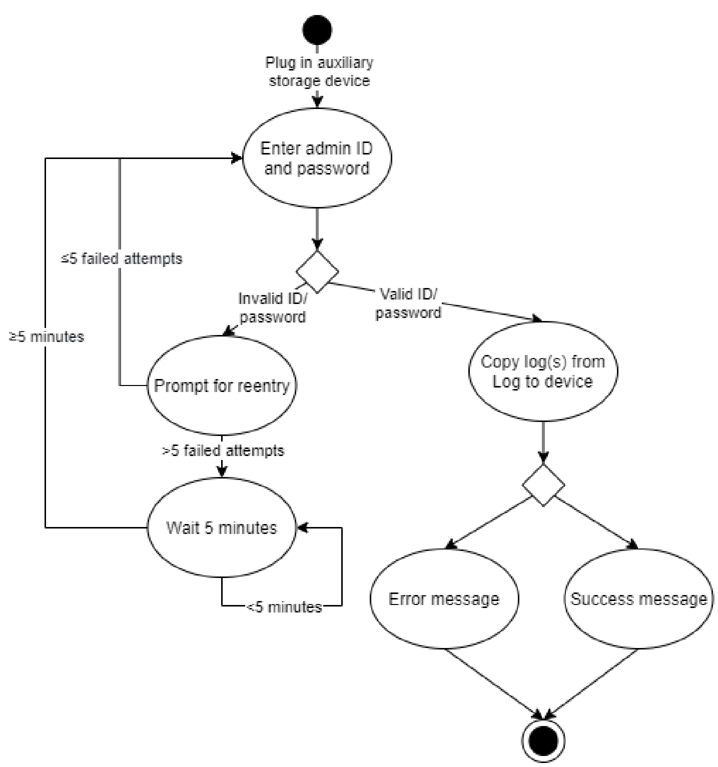
4.5 Activity Diagram



Precondition: IoT HTR has already been initialized

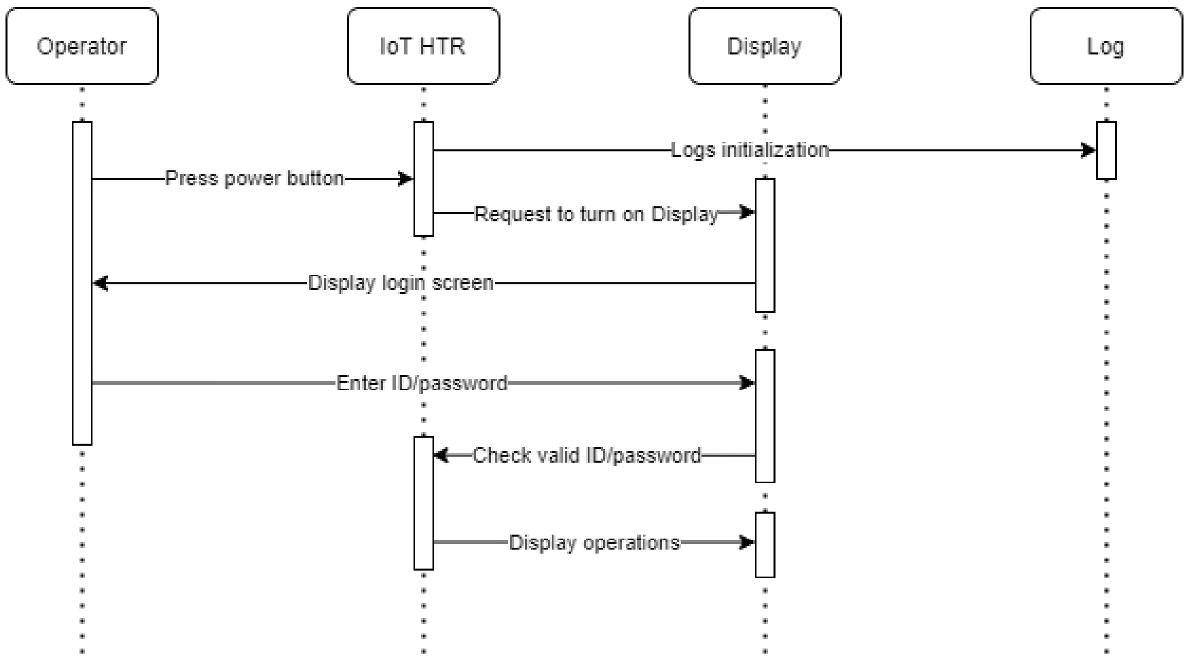


Precondition: IoT HTR has already been initialized

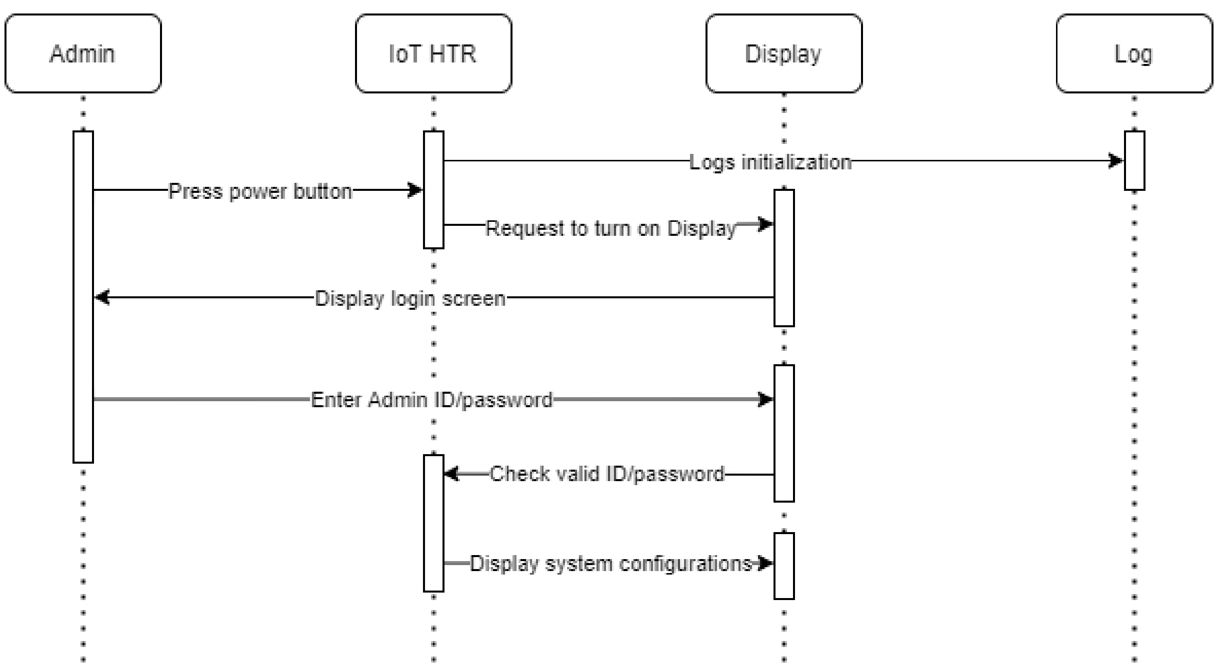


4.6 Sequence Diagrams

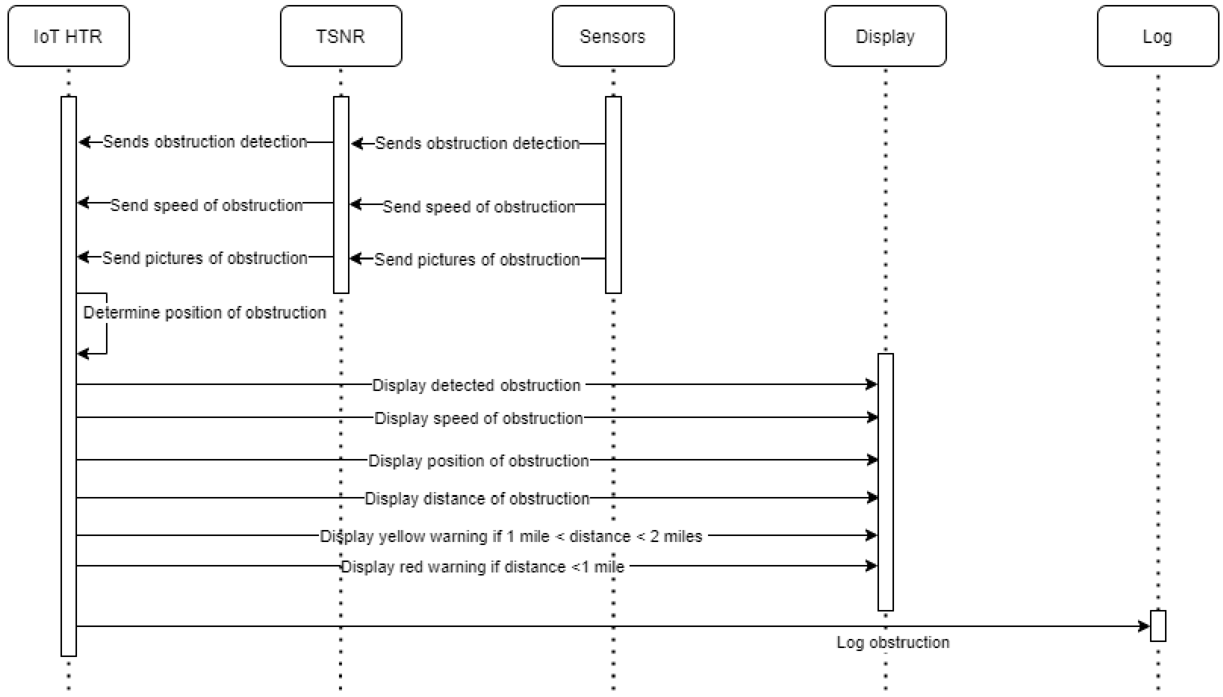
Use case 1:



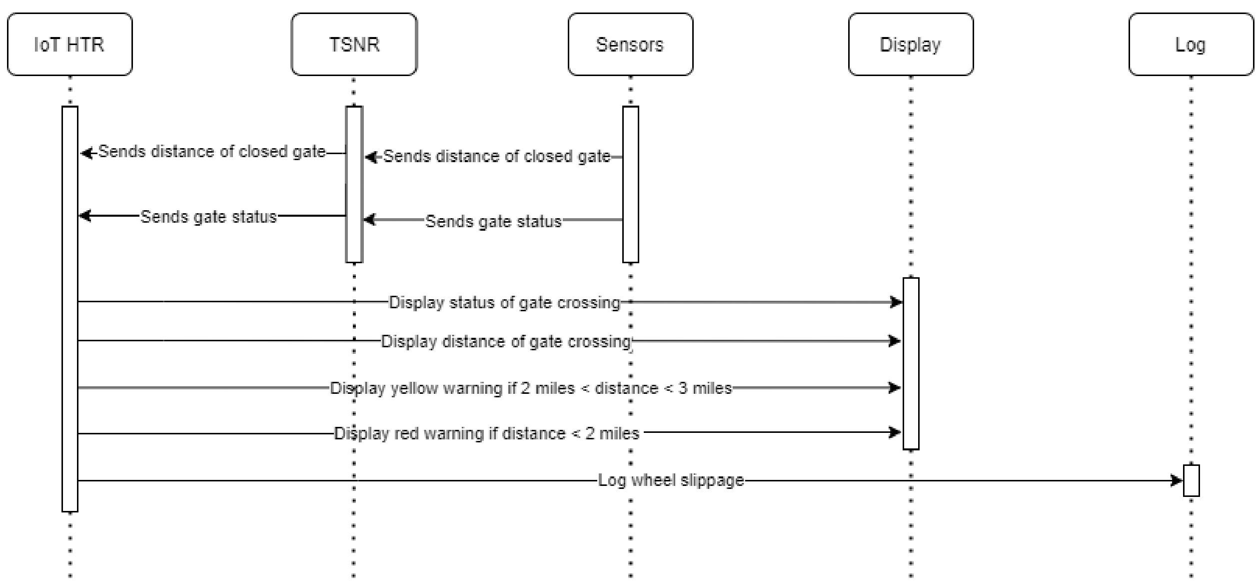
Use case 2:



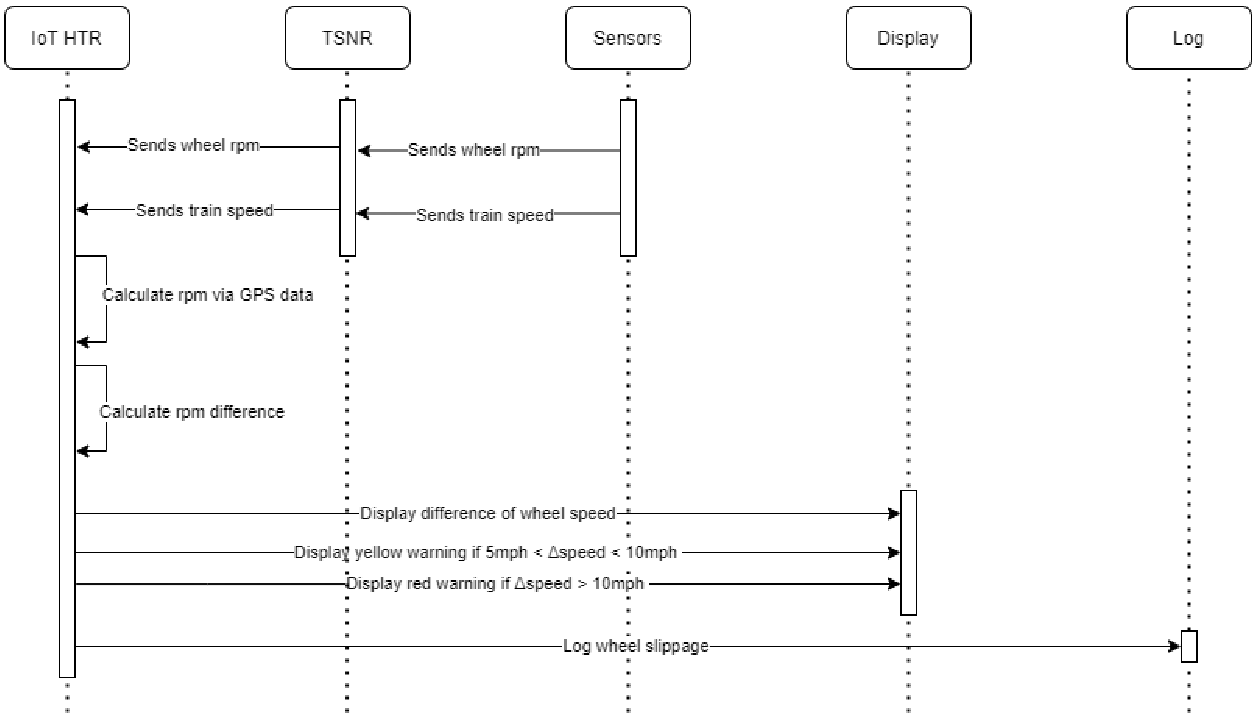
Use case 3:



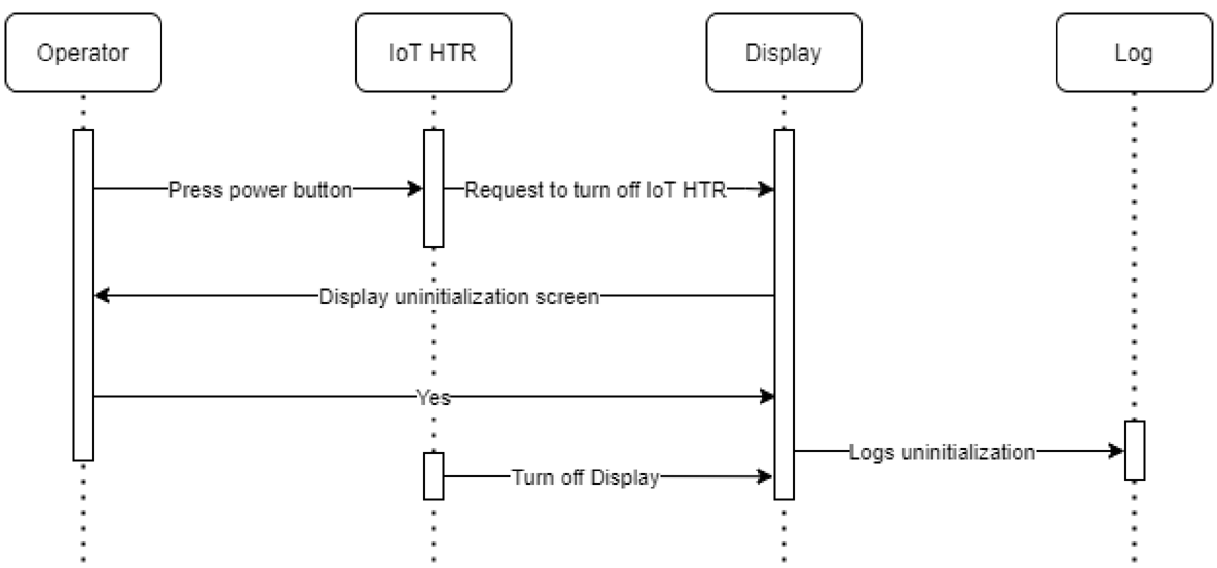
Use case 4:



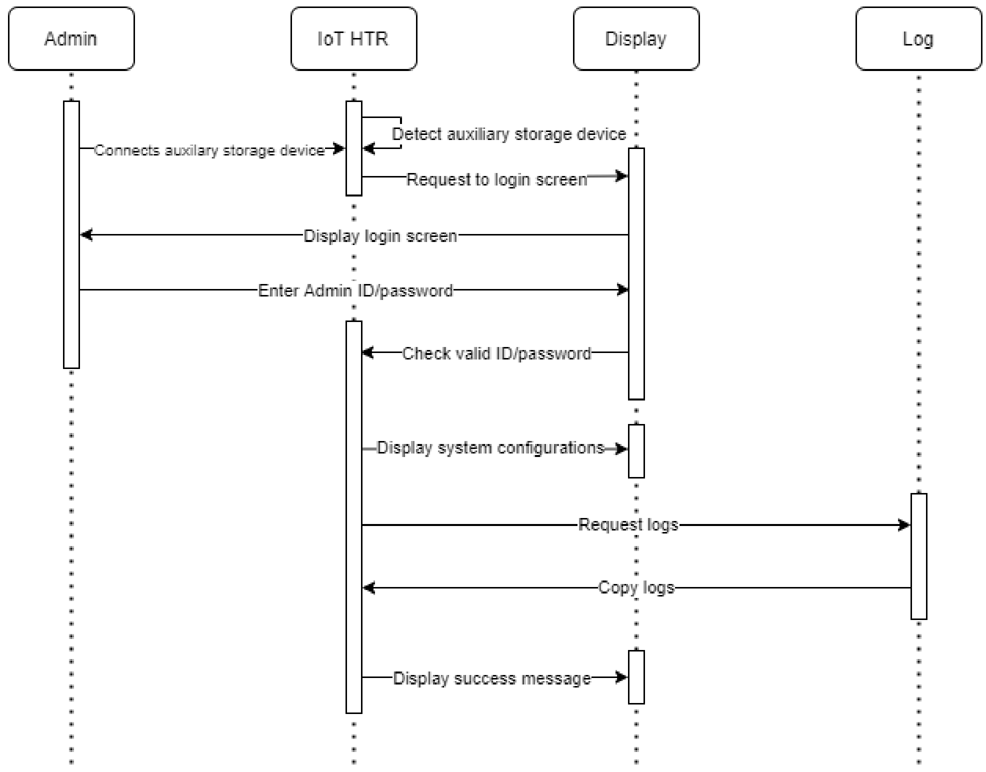
Use case 5:



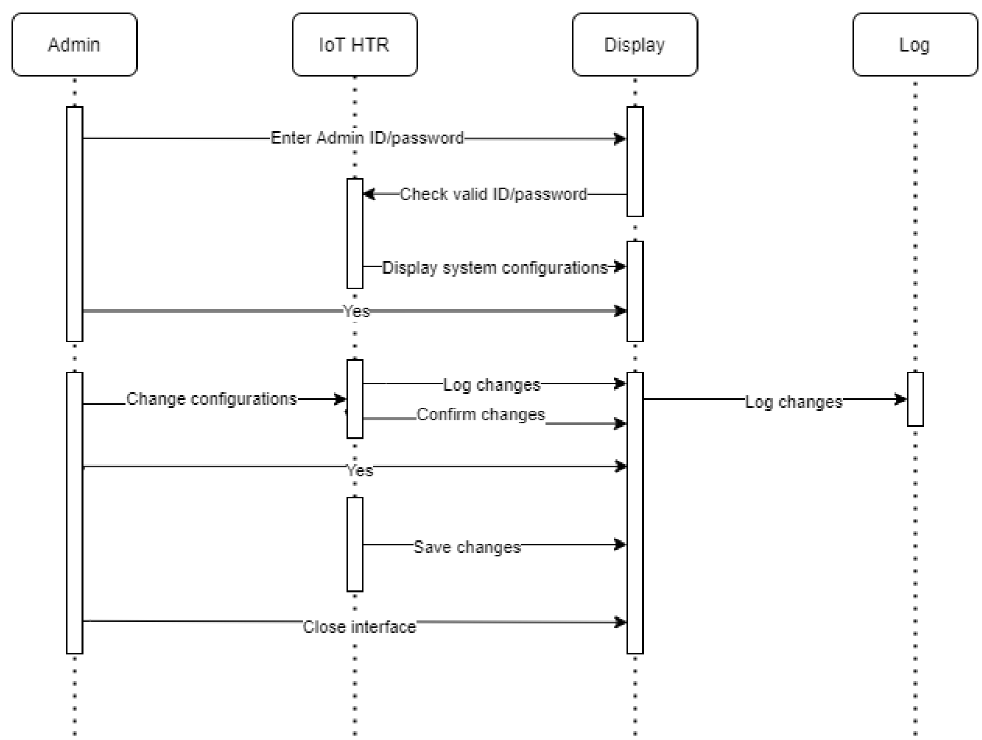
Use case 6:



Use case 7:

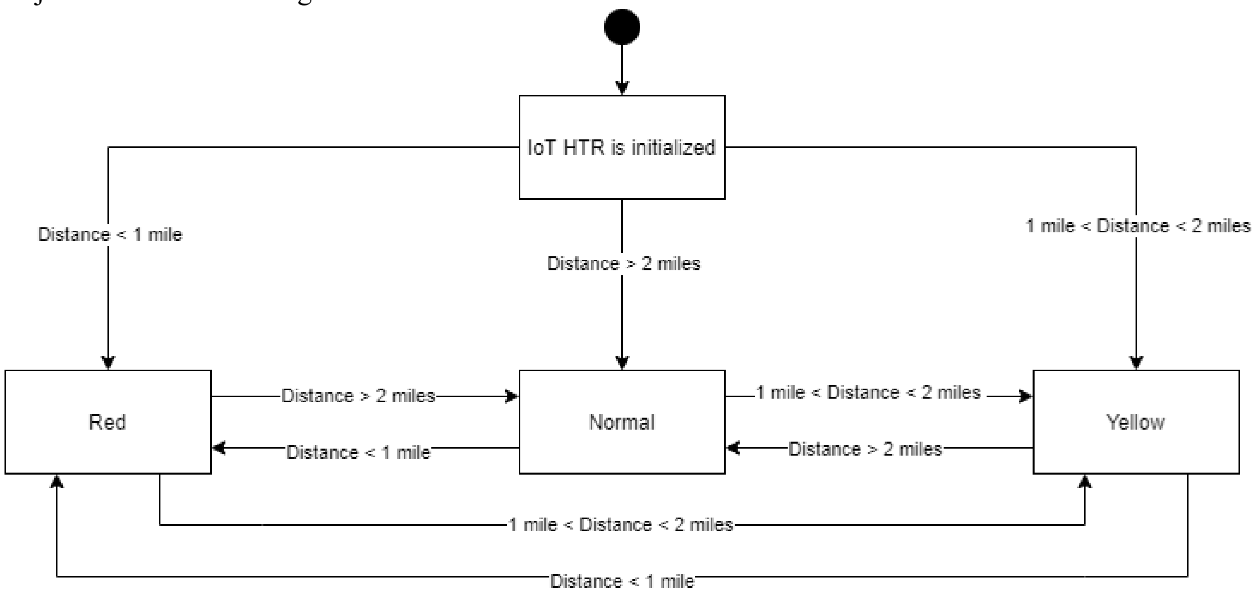


Use case 8:

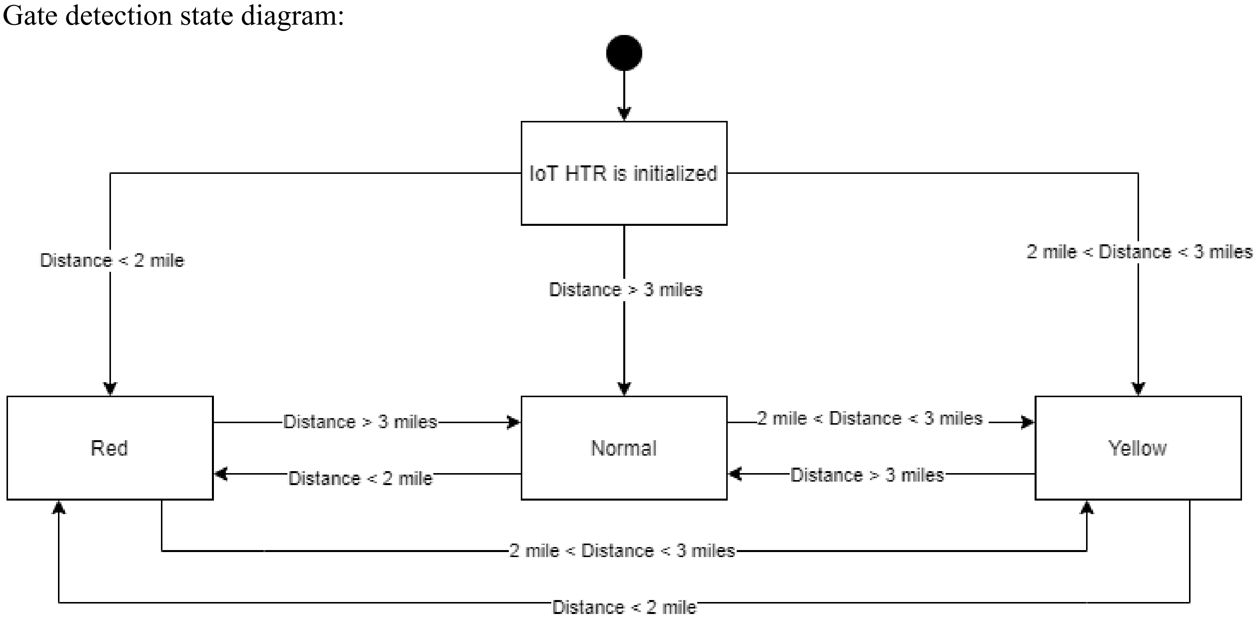


4.7 State Diagram

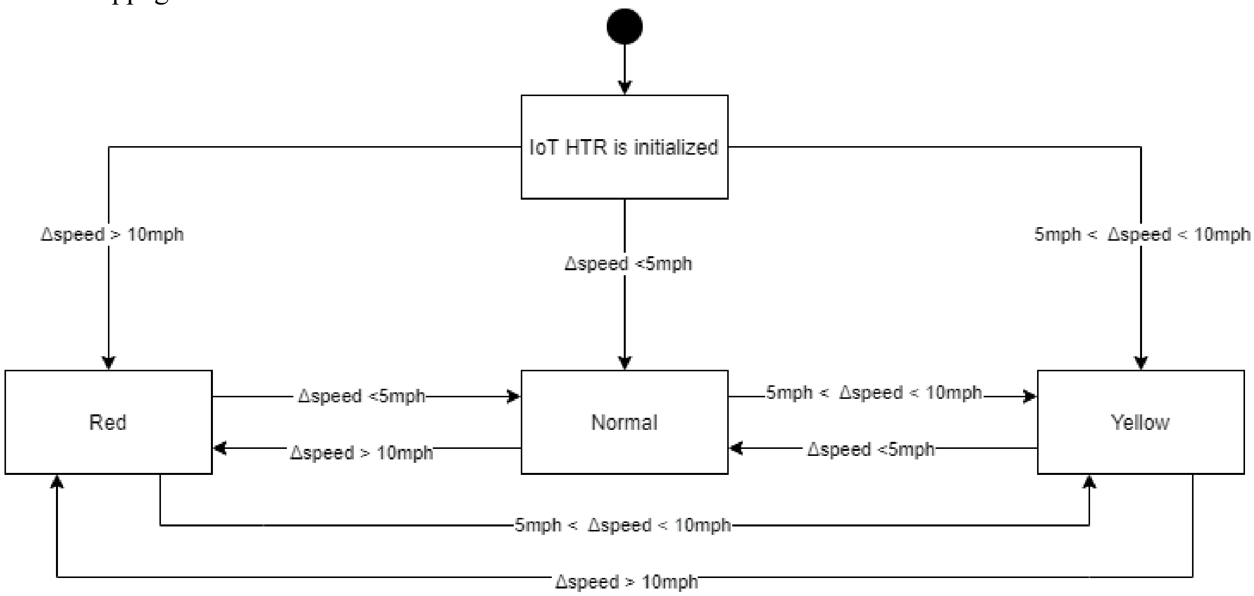
Object detection state diagram:



Gate detection state diagram:



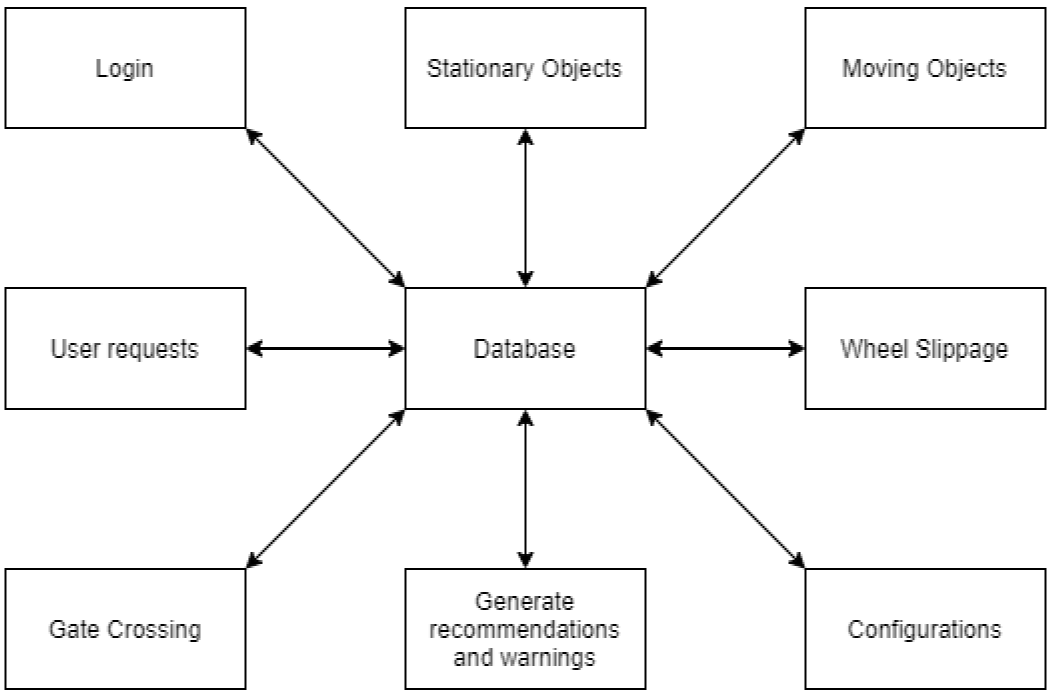
Wheel slippage detection:



5. Software Architecture

5.1 Data Centered Architecture

Arrows from the client software into the database illustrate the movement of sensor data going into the database via TSNR while arrows from the database into the client software represent sending data inputs into the respective process (e.g. stationary object detection) and displaying it on the Display.



Pros:
- All IoT data is logged into a central system
- The log is easily accessed from a single location
- Very modular and scalable in adding more software modules

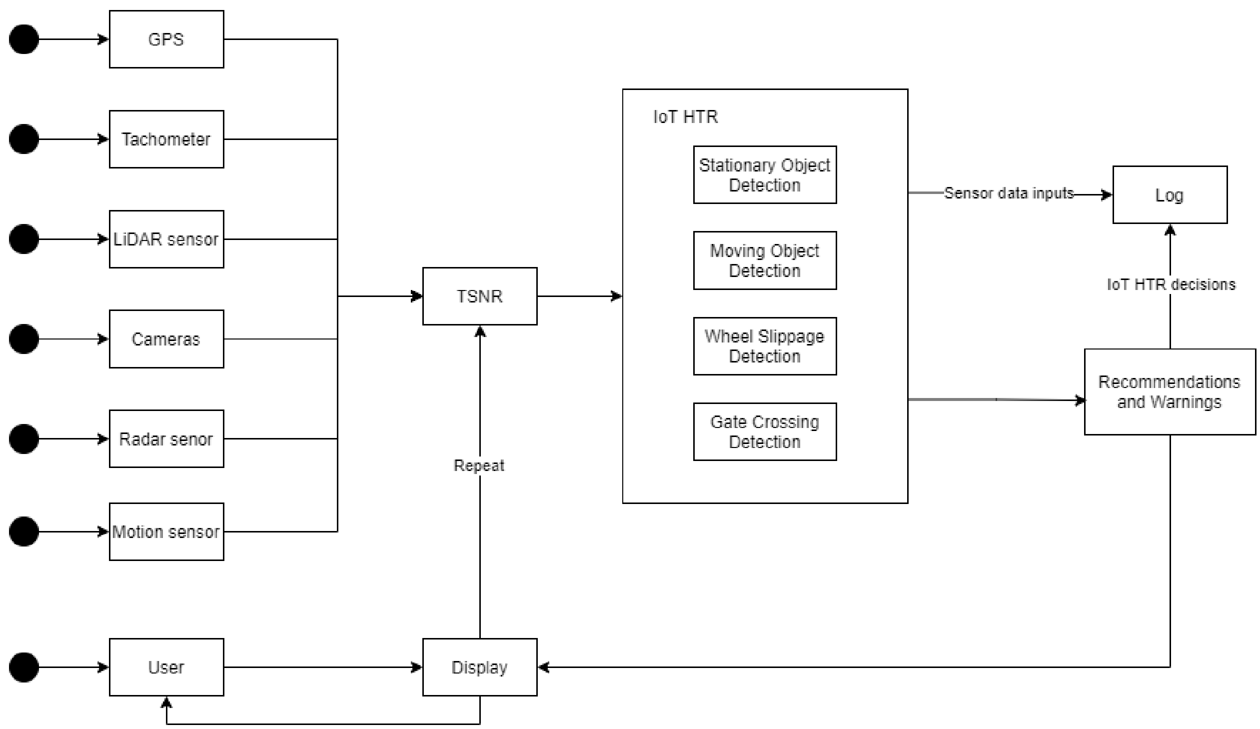
Cons:
- High dependency between the central system and individual software
- Failure of central system propagates into a failure of the whole log
- Data structure of central system must be consistent for all software
- The log must manage high traffic requests
- It is vulnerable to failure and data replication/duplication
- Partitioning of data/links slows down how quickly the data can be accessed

The Data Centered Architecture does not suit IoT's dynamic nature. The main issue of a Data Centered Architecture implementation into IoT is that the data center is too slow. Our system is a real time system therefore there is a time constraint for all processes. In a Data Centered Architecture implementation of IoT it would need to get the data from the sensors and store it into the database. Then IoT would have to get that data to analyze/process it. The decisions from IoT would also need to be stored into the database. At this point nothing has even been displayed on the Display. As illustrated, this is much too slow for a mission critical system.

Being primarily a storage-based architecture, the only IoT module that somewhat fits this role is the log. Even then, the log is not a suitable choice for utilizing the Data Centered Architecture. Each log file stores IoT's data from inilizationg to uninitialization, which does not require massive storage space typically found in a large database. Additionally, the log constantly updates and changes its stored values, which is not a feature of long-term database storage. Our IoT operates on current (most recent) data which is dynamic, therefore a file with older values serves no purpose to the calculations made by the software.

5.2 Data Flow Architecture

The following evaluation refers to a Pipes and Filters Approach to Data Flow Architecture.



Pros:

- Allows IoT to have high throughput of data processing
- Simplifies IoT's software execution into block chains
- Each block chain can run sequentially or independently in parallel (flexible architecture)
- Individual software filters are easy to maintain and modify

Cons:

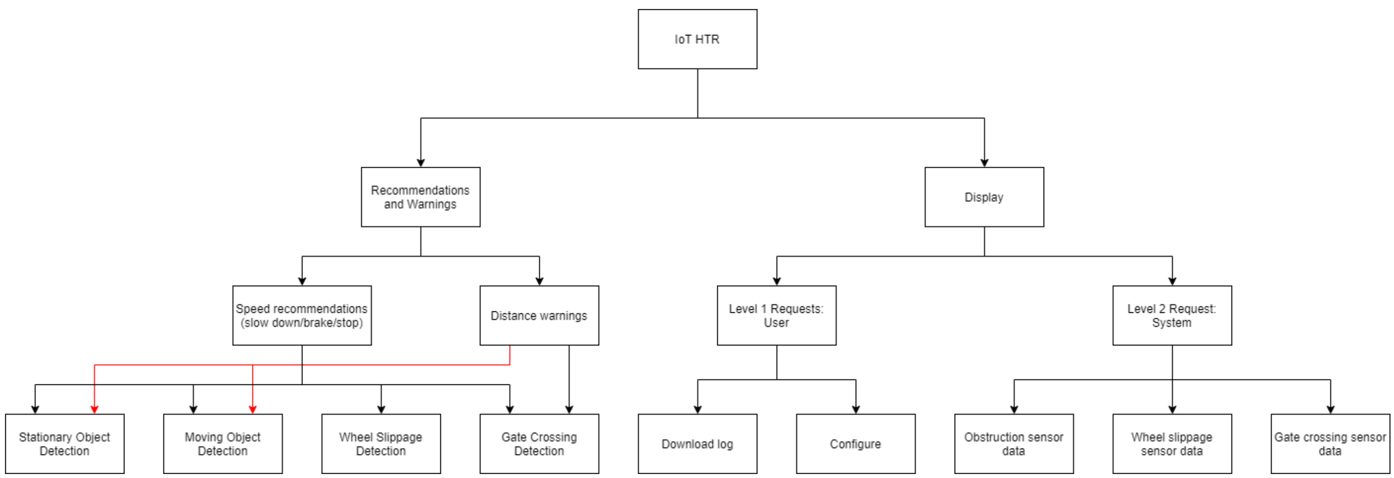
- More sequential software increases IoT's processing time
- Does not allow IoT to modify/redirect existing filters dynamically
- Failure in a software filter will create bottlenecks in subsequent filters
- Software filters have a hard time working on a large problem
- Software filters cannot effectively store calculations
- Not suitable for dynamic interactions and dynamic change

The Data Flow Architecture is a decent option for IoT. One of the highlights of this architecture is the ability for IoT to run data processes in parallel. If all six types of sensors were to haphazardly enter IoT's four main detection processes, IoT would likely hang. However, by running these processes separately in *parallel and with TSNR, IoT would have a higher throughput of data. Additionally, chains of parallel filters create distinct blocks for the data to flow, simplifying the readability and development of IoT's software.*

However, the biggest drawback of having predefined filter blocks is that IoT cannot make decisions outside those predefined filters. In this sense there is no dynamic decision making. For example if the rails were icy, IoT HTR would not be able to alert the operator to proceed with caution thereby putting the passengers and operator at risk. By having a fixed number of filters, there can only be a finite number of possibilities that the data can flow. This inhibits IoT from making decisions to larger problems outside the scope of what was naively intended. Furthermore, having a finite number of filters means that IoT cannot effectively store calculations and values, a key component in generating recommendations. Adding more filters to account for these variations just increases disorganization. Thus, although the Data Flow Architecture is promising in its parallel filters, its fixed nature prevents IoT from running effectively.

5.3 Call Return Architecture

The data enters the model in the following manner: Sensor Data → TSNR → IoT HTR



Pros:

- Reduces complexity of IoT into smaller and simpler subprograms (separation of concerns)
- Subprograms can be hidden from one another
- Groups of subprograms create easy to understand hierarchy of IoT
- Modular in adding new subprograms to IoT

Cons:

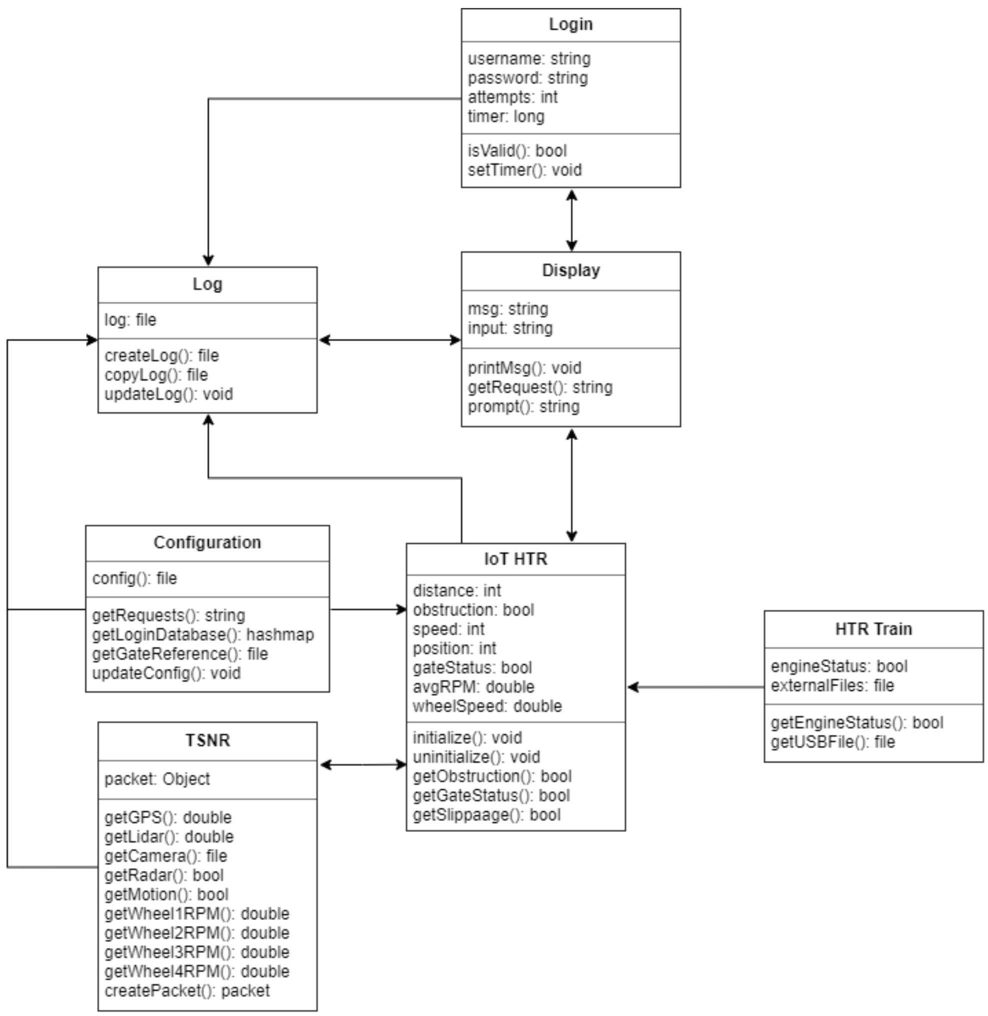
- High dependency between output of one subprogram and the input of another subprogram
- Failure of subprogram will propagate up into IoT's main process
- Does not scale well as IoT's complexity increases
- More subprograms increase process time of IoT

The Call Return Architecture is a strong candidate for IoT because of its hierarchical system. Through visualizing IoT from a hierarchical perspective, complex processes such as recommendation and warning generation can be further broken down into much simpler and easier to manage subproblems. For example, all terminal requests in IoT can be grouped and divided into Level 1 and Level 2 requests, and distance warnings can be derived from object detection and Gate Crossing detection.

Additionally, the hierarchical approach in the Call Return Architecture satisfies IoT's computation requirement. At every second, IoT must process TSNR's packet which includes data from six different types of sensors. Then, IoT HTR must complete calculations for each individual sensor data. This requires IoT to be *quick in computing, storing, and retrieving values to anticipate the next TSNR packet a moment later. Because this architecture is built on sharing and returning data between subprograms, this scenario is good for supporting IoT's calculation process.*

The one fault in this architecture is complications arising from scaling up IoT. When IoT becomes bigger, there must be exponentially more subprograms in the hierarchy, which will slow down development and computation time. Fortunately, IoT at this moment is not big enough to pose scalability issues while using the Call Return Architecture.

5.4 Object Oriented Architecture



Pros:

- Uses abstraction to reduce complexity of IoT
- Objects and methods are reusable, modular, and dynamic
- Implementation of IoT objects are hidden from one another
- IoT objects can be constructed and executed independently
- Classes can be reused without having to change them fundamentally for IoT (e.g. sensors)
- Easy to implement the IoT login process

Cons:

- Inefficient for high performance calculations in IoT

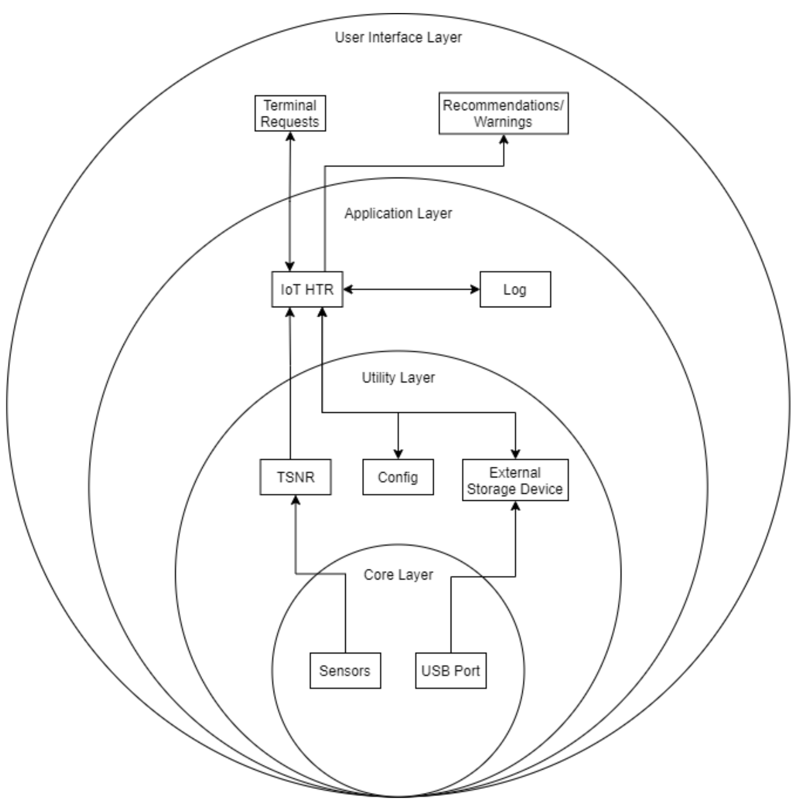
- Prone to excess and large amounts of code (cost of size and performance)
- Difficult implementation without a high level plan of IoT

The Object Oriented Architecture is an excellent option for IoT. The main advantage of utilizing this architecture is using abstraction to minimize and simplify IoT's software, By only specifying required classes and methods in IoT's objects, abstraction enables only the bare minimum and necessary amount of code to sufficiently run IoT. Thus, the run time of IoT can be reduced while simultaneously increasing overall efficiency, readability, and performance of IoT. Furthermore, by creating distinct classes such as the Config or Log, IoT's objects can run independently without the risk of one object's dependencies clashing with another object.

Another major benefit of using an Object Oriented Architecture is that classes can be reused without having to write them again. For example, a sensor class can be reused with little modification as most sensors are similar. Furthermore the logic for making recommendations/warnings are generally the same. They all have the same output essentially and require similar inputs. The only main difference in each logic is how to process the data. For instance, in the stationary object, IoT has to receive from the sensor that the object is moving at 0mph while for the moving object it needs the speed to be greater than 0mph. Another reason why an Object Oriented Architecture is easy to implement is because sensors can be represented as objects and its attributes will be the sensor data.

However, the Object Oriented Architecture assumes that IoT's abstracted code is well optimized for performance and thoroughly planned out. Objects require an abundance of space to store methods and stored data. When the code is unoptimized, duplicate and unneeded objects may still linger after its initial use, taking up valuable space in IoT. Additionally, it is difficult to optimize IoT's code without a thorough understanding and high level software plan of IoT. There needs to be a balance between having the bare minimum amount of code and a sufficient amount to successfully execute IoT.

5.5 Layered Architecture



Pros:

- Distinct layers simplifies complexity and understanding of IoT
- Divides IoT into four unique hardware, software, and interface layers
- Changes in one layer do no significantly impact development of other layers
- Robust for grouping and customizing IoT's functionality, specs, and interface

Cons:

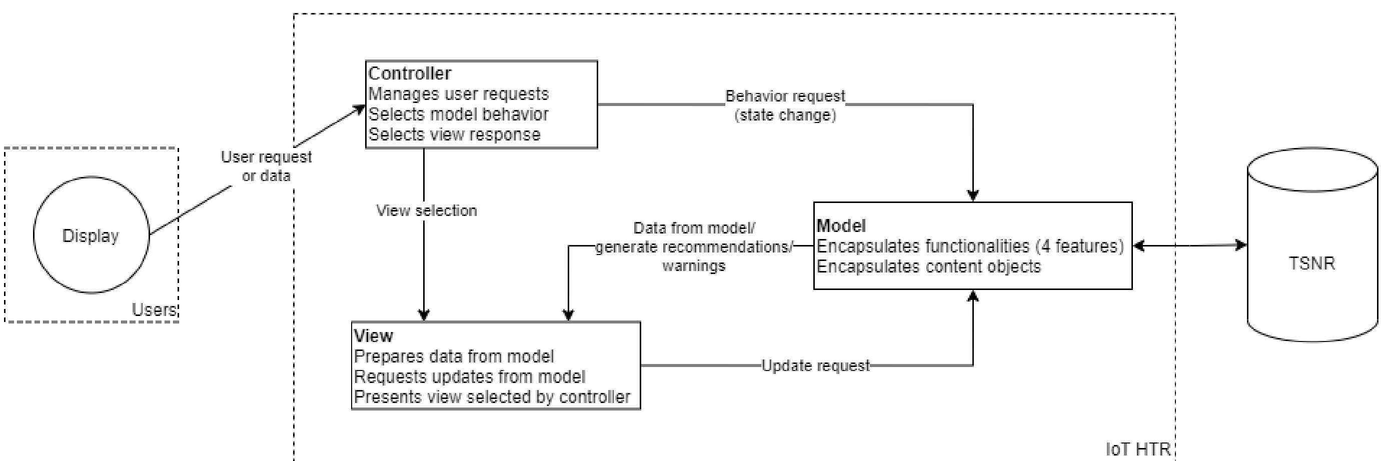
- More independent layers make it harder to maintain and modify IoT
- Not effective for higher modularity and scalability of IoT
- Moving through multiple layers lowers performance and increases process time for IoT
- Layers that are far apart have a hard time working together
- Not necessarily the fastest as we are pushing sensor data through multiple layers

The Layered Architecture is a decent fit for IoT due to its powerful organization of IoT's design layout. By separating IoT's hardware, software, and interface into four unique layers, each layer will have a designated purpose in IoT that makes customization and maintenance of the software easier. Furthermore, having unique layers enables independence during the software development process. For example, modifying existing or adding new Terminal requests in the User Interface Layer will not have a significant impact on the Utility and Core Layers, as these layers are far away from the user interface Layer. This ensures faster development of IoT without severely altering development of other layers.

Similar to the Call Return Architecture, the only major flaw of the Layered Architecture is when scaling up IoT. Because there are only four unique layers, adding new modules to IoT will overflow their respective layers until it will become hard to organize and maintain each individual layer. However, as stated in the Call Return Architecture, IoT at this moment is not large enough to encounter this scenario. Thus, the Layered Architecture is a powerful organizer of IoT's layout

However, one of the biggest challenges the Layered Architecture faces when implemented in our IoT system is that sensor data has to be pushed through multiple layers. This slows down processing time and presents performance and real time issues. As stated before, IoT runs in real time and is a mission critical system therefore a Layered Architecture would pose a threat to the safety of passengers due to its slowness.

5.6 Model View Controller Architecture



Pros:

- Oriented for IoT's user interface
- Splits IoT into Controller, Model, and View units enables for easier maintenance and readability

- Changes in one unit do not significantly impact the other two units
- Three separate units ensures independence

Cons:

- Not expandable outside of user interface
- Difficult implementation without a high level plan of IoT's interface
- IoT data follows a strict flow in this architecture
- Model unit becomes more complex as functionalities are added to IoT
- View is dependent on the Controller and Model
- Model oftentime does much of the work

The Model View Controller (MVC) Architecture is a good option for IoT. For IoT, the MVC Architecture fits quite nicely when implemented properly. Similar to how the Layered Architecture improves organization of IoT by separating the hardware, software, and interface into four different layers, the MVC architecture separates IoT's terminal interface into a Controller, View, and Model unit. Each unit is independent and serves its designated role in IoT. Many of the features and logic of IoT can easily be translated into a MVC Architecture. For example, the Controller unit is responsible for maintaining user requests, while the View unit is responsible for displaying recommendations and warnings onto the Display for the user. In addition to representing the structure of IoT, it also can represent a simple and straightforward login wherein the user requests from the controller to login, the controller requests the model to log into IoT, and the view displays the UI and success of logging in.

The biggest problem with this architecture is that it is hard to implement the three units without a thorough understanding and high level plan for IoT's interface. Although only having three units may seem simple to implement, each unit is unique and has its own data flow. Interactions between each unit requires knowledge of how the two units function separately and the triggers that cause the sharing of information in the first place.

5.7 Conclusion

The best architecture for our implementation of IoT is a mixture between the Model View Controller and Object Oriented Architecture. The MVC Architecture provides a powerful way to organized IoT's design layout into three distinct units: Controller, View, and Model. By organizing IoT into these three units, the implementation of IoT will be easier to maintain and customize due to having all related and shared components grouped together. The three unit architecture also helps in simplifying what would otherwise be a complex system. Additionally, each unit is independent therefore it has the benefit of ensuring that the impact of one unit does not severely alter the others during IoT's implementation.

The Object Oriented Architecture will provide an excellent avenue to use abstraction to minimize and simplify IoT's software. An Object Oriented Architecture substantially allows for a streamlined program with only the most essential code. Thus, the run time of IoT can be reduced while simultaneously increasing overall efficiency, readability, and performance of IoT. These qualities are extremely important to a real time and mission critical system. Many of IoT's functionalities behave in comparable ways and therefore classes and objects can be utilized to reduce rewriting unnecessary code. Sensors can easily be represented as objects and its attributes will be the sensor data. Additionally, because we are all familiar with this architecture the team will be able more easily implement it effectively and therefore have a faster software development process.

5.8 Implementation

5.8.1 Model

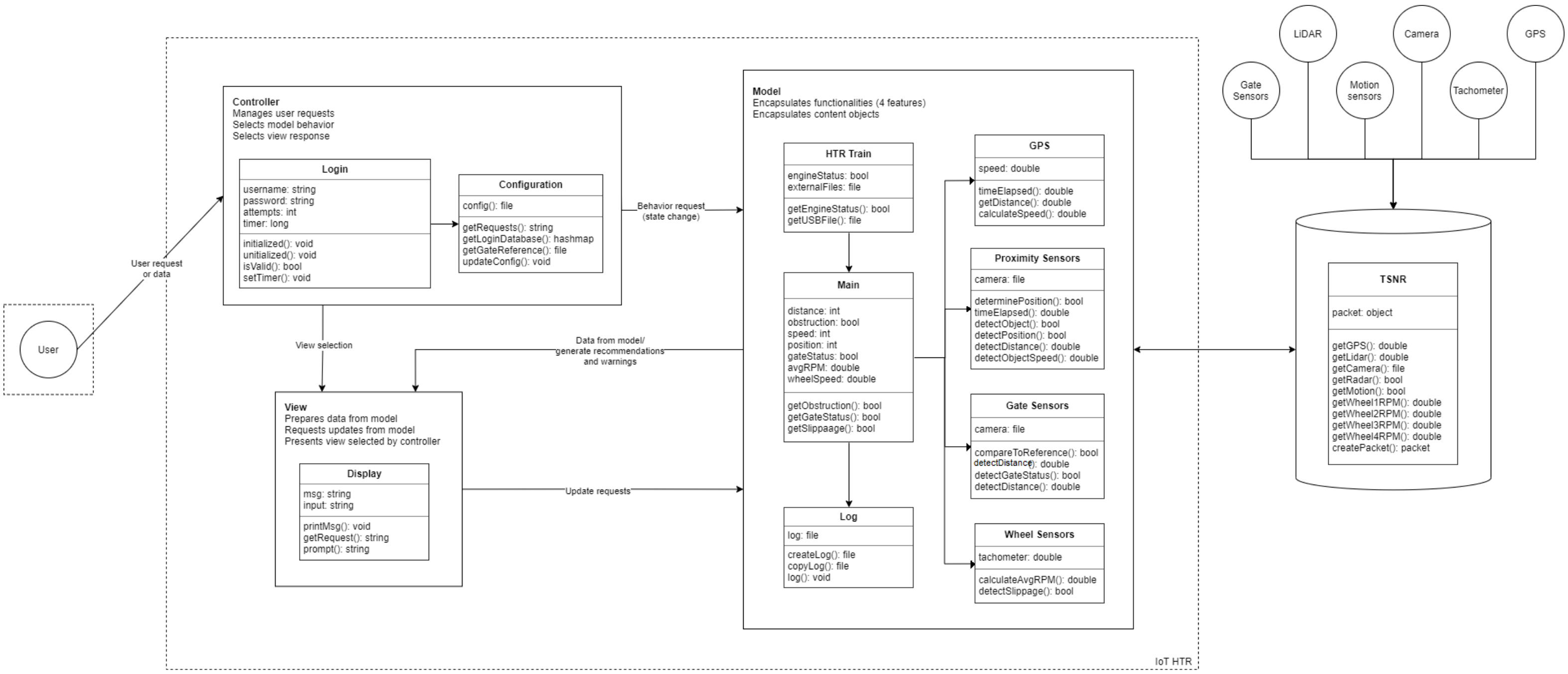


Figure 4: *Data will be fed into the software by means of the TSNR. The physical sensors (represented by the circles and the respective names) send data into the TSNR which is then processed and sent to IoT HTR's main function that is housed within the Model unit. The main calls on sensor classes which handles further calculations that need to be made. For example, calculating the average RPM of the wheels from the four tachometer sensor data. The main logs the data and sends recommendations and warnings to the Display which is housed in the View unit. User input is handled by the Controller. For instance, the user requests to login or change configurations. Visual feedback is produced by the View unit.*

5.8.2 Classes

Login:

1. username: a string passed through by the user to be used in authentication to the software
2. password: a secret string passed through by the user to be used in authentication to the software
3. attempts: an integer counting the number of attempts made to sign in that were unsuccessful
4. timer: a long that holds how much time has passed since last lock system due to five or more failure attempts to login
5. initialized(): a void method that turns on the IoT HTR by logging the initialization and turning on the Display
6. unitialized(): a void method that turns off the IoT HTR by logging the uninitialization and turning off the Display
7. isValid(): a method that returns boolean based on the username and password being a valid pair. Works by checking the supplied username/password with the login database housed in the configurations class.
8. setTimer(): A void method that starts the cpu timer. The timer begins when five or more failed

attempts to login have occurred

Log:

1. log: a file consisting of the entire log of events that were encountered during the initialization of IoT HTR
2. createLog(): A function that creates a blank log file
3. copyLog(): A function that returns a copy of the current log by retrieving it and copying it onto an external storage device
4. log(): A void function that updates the log by taking data passed into it (e.g. sensor data, object detection, recommendations, etc) from the main and writing it into the log file

Display:

1. msg: a string that is displayed to the user on the front end
2. input: a string that is taken in by the user and used to request a certain action by the system (e.g. log in by entering a username/password)
3. printMsg(): void function that prints message to screen
4. getRequest(): function that calls for an action as directed by the user (e.g. requesting to display HTR operations by supplying a valid username/password)
5. prompt(): function that displays what is requested

Configuration:

1. config(): returns a file consisting of the configurations for the IoT
2. getRequest(): function that returns a string that access to the configurations interface has been granted
3. getLoginDatabase(): returns a hashmap of all of the users on the system with each of their corresponding usernames/passwords and privileges
4. getGateReference(): a file that contains the library of references of different gate statuses. Is used in the comparison of camera photos of gate crossings. This method allows the admin to view/change/update the library
5. updateConfig(): updates the configuration file

HTR Train:

1. engineStatus: a boolean variable that is true if the engine is turned on and working correctly and is set to false otherwise
2. externalFiles: a file variable consisting of any external files concerning the IoT
3. getEngineStatus(): returns the value of the engineStatus boolean
4. getUSBFile(): returns a copy of the logs onto the USB (external storage device)

Main:

1. distance: an int value representing the distance of objects (obstructions, gates)
2. obstruction: a boolean value that is true when there is an object detected and false otherwise
3. speed: an int value representing the speed of objects
4. position: a boolean value that is true when the position of objects are in front of the HTR train and false when the object is in the back
5. gateStatus: a boolean that true when the gate crossing is open and false otherwise
6. avgRPM: a double that represents the average of the four tachometer RPM wheel data
7. wheelSpeed: a double that represents the speed of the HTR train based on the GPS data
8. getObstruction(): a boolean method that communicates that a warning and recommendation need to be displayed. True means there is an object otherwise false is returned. To see how an

obstruction is detected go to the Proximity Sensor Class

9. getGateStatus(): a boolean method that communicates that a warning and recommendation need to be displayed. True means there is an open gate crossing otherwise false is returned. To see how a gate crossing is detected go to the Gate Sensor Class
10. getSlippage(): a boolean method that communicates that a warning and recommendation need to be displayed. True means there is wheel slippage otherwise false is returned. To see how wheel slippage is detected go to the Wheel Sensor Class

GPS:

1. speed: a double to represent the speed of the HTR train based on the GPS data
2. timeElapsed(): is a method that uses the timer to determine the time elapsed since the last packet of GPS data was received from TSNR
3. getDistance(): is a method that calculates the distance traveled based on the GPS data
4. calculateSpeed(): returns a double that represents the speed of the train by dividing the getDistance() by the timeElapsed()

Proximity Sensors*:

1. camera: a file that contains pictures of the environment around the HTR train
2. determinePosition(): a boolean method that determines if the camera photos were from front cameras or back cameras by checking the ID number of the camera that took the photo
3. timeElapsed(): is a method that uses the timer to determine the time elapsed since the last packet of proximity data was received from TSNR
4. detectObject(): a boolean method that returns true if there is an object and false otherwise
5. detectPosition(): a boolean method that gets the position of the object from determinePosition()
6. detectDistance(): a double method that gets the distance of the object
7. detectObjectSpeed(): a double method that returns the speed of the object
8. Detects if there is an obstruction by mapping the surround with the LiDAR and camera sensor data
9. Calculates the speed of the object by dividing the distance over the time elapsed
10. Outputs a warning flag if there is an obstruction as defined in the above two statements
11. Outputs if it is a stationary object by checking if the speed is 0mph otherwise the object is moving
12. Outputs a recommendation flag based on the speed, distance, and location of the object

Gate Sensors*:

1. camera: a file that contains pictures of the environment around the HTR train
2. compareToReference(): a boolean method that determines if it found a match of the gate crossing to the library of references. Works by comparing the similarities of the photos from the camera to the photos in the library of references installed on IoT HTR
3. detectDistance(): returns a double that represents the distance of the gate crossing
4. detectGateStatus(): returns a boolean true if it found a match in the compareToReference() function otherwise false. If the gate is open then true is returned otherwise false is returned.
5. detectDistance(): returns a double with the distance of the gate crossing
6. Outputs a warning flag to the main if the gate is open as determined by the above statement
7. Outputs a recommendation flag to the main based on the distance of the gate crossing (3 mile radius)

Wheel Sensors:

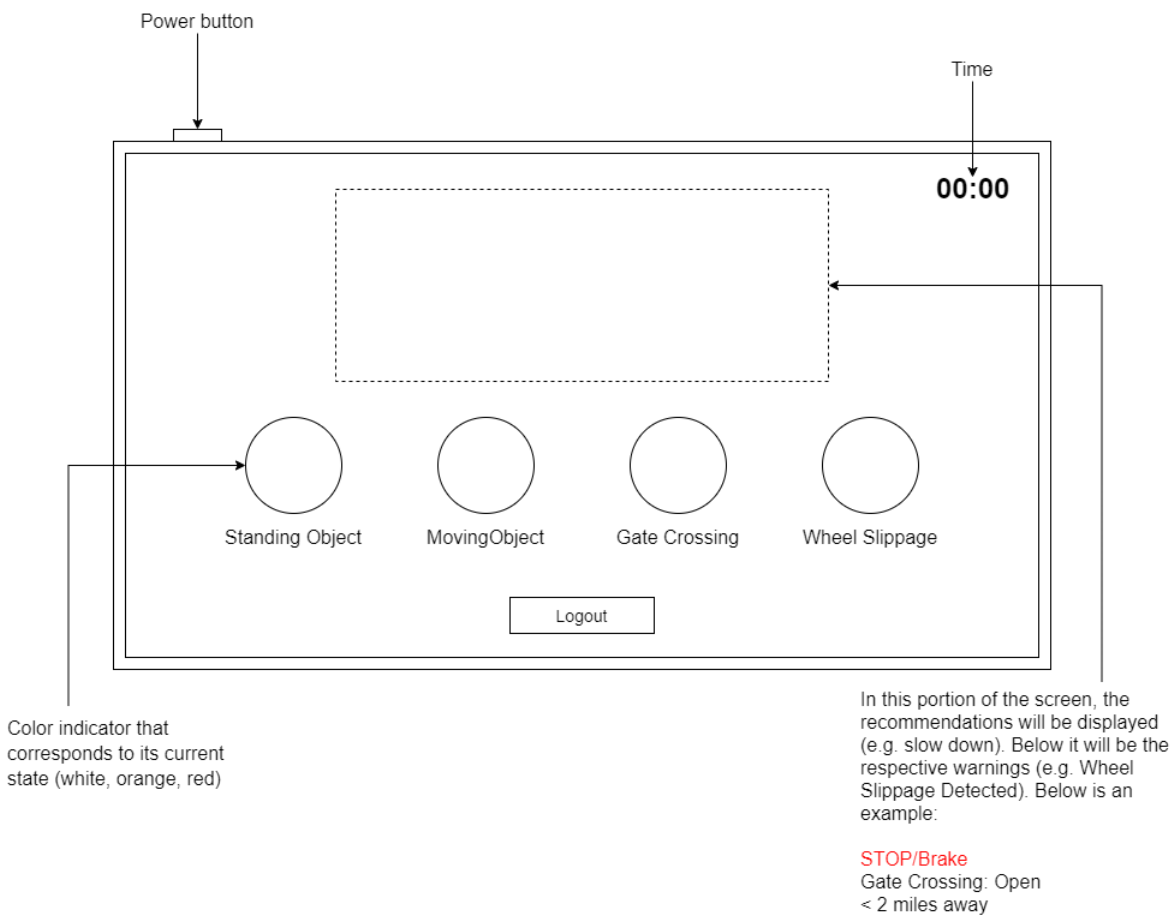
1. tachometer: a double that represents the RPM of the wheel it is attached to

2. calculateAvgRPM(): a method that returns a double by adding all four wheel RPM data and dividing it by four
3. detectSlippage(): a method that returns true if the difference between the speed from the GPS and the speed from the tachometer data is greater than 5mph
4. Outputs a warning and recommendation flag to the main if slippage is present (Δspeed >5mph) by comparing the difference in the speed from the GPS and the speed from the tachometer data

TSNR:

1. packet: an object that contains sensor data
2. Each getX() method receives data from the respective sensor
3. createPacket(): receives all sensor data by using the getX() methods and creates packets of data to be sent to IoT HTR.

5.8.3 Display UI



6. Code

7. Test Cases

8. Issues

9. Cost

Costs for any project can be daunting and often difficult to calculate because the project expands and evolves over time. We can give a rough cost estimation of this project based on a sample of devices one would buy. These costs are subject to change and there are multiple companies with varying prices that sell these sensors. It also should be noted that a large portion of the project will be software which does not have any external costs.

Sensor	**Purpose**	**Unit Price**
Anemometer	Wind speed	$50-$800
Hygrometer	Humidity	$200
Rain Gauge	Rain	$300
Thermometer	Temperature	$200
Tachometer	Wheel's RPM	$50-$900
LiDAR	3D mapping, detect obstructions	$500-$1,000
Friction/Shear Force	Friction	$6,000-$8,000
System of solid state circuits and pressure sensors	Gate	$1000+
IoT Display and software	UI/UX	
GPS	Position/speed	$100-$1000
Assemble		
Radar	Status of crossing gate	
Cameras	Photos, object recognition	
Motion detection sensors	Detect moving objects	