Train Trax: Train Monitor for Positive Train Control Test Beds

Software Design Document

Version 1.5

12/08/2015

Stephen Jalbert

Rashad Madyun

Corey Sanders

|  |  |  |  |
| --- | --- | --- | --- |
| Revision History | | | |
| Version | Date | Description | Author |
| 1.0 | 10/25/2015 | Initial Version. Created temporary template for software design specification. | Stephen Jalbert  Rashad Madyun  Corey Sanders |
| 1.1 | 11/9/2015 | Added definitions to document. Updated document with overview information about the system. Updated the Train Navigation Database design section with details of track geometry collection. | Corey Sanders |
| 1.2 | 11/16/2015 | Created an outline for how each component of Train Trax is to detail its design.  Added information about all of the interfaces planned to be used with the system.  Updated the scope to describe the flow of the system.  Added details of the design of the Train Navigation GUI, Train Navigation Service, and Train Navigation Database.  Improved details used in the SDD Design overview.  Renamed document from the “Software Design Specification” to the “Software Design Document”.  Improved the labeling of sections. Added Captions to Figures.  Reorganized the structure of the design document. | Rashad Madyun  Corey Sanders |
| 1.3 | 11/24/15 | Updated document with details of the Train Navigation Database Design, including track geometry data collection.  Added details for the Motion Detection Unit Design.  Added Position Estimation Algorithm Details  Updated details of GUI Design.  Improved formatting | Corey Sanders  Rashad Madyun |
| 1.4 | 12/01/15 | Added Traceability Matrices.  Made Page Numbers Visible.  Fixed Page Number Formatting  Replaced references for Train Controller with Train Command Station and Track Switch Controller.  Updated figure references.  Updated figures where flow was unclear.  Added figures to explain how Train Trax communicates | Corey Sanders |
| 1.5 | 12/08/15 | Updated several figures based on instructor feedback.  Add diagrams to show interactions of the GUI for important use cases.  Improved the Scope of project description.  Clarified how communication between objects in the system works.  Added details of the function of the Train Control Terminal.  Updated View Figures | Corey Sanders |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Revision History 2](#_Toc437039535)

[1 Introduction 7](#_Toc437039536)

[1.1 Purpose of this document 7](#_Toc437039537)

[1.2 Scope of the development project 7](#_Toc437039538)

[1.3 Definitions, acronyms, and abbreviations 8](#_Toc437039539)

[1.4 Overview of document 11](#_Toc437039540)

[**2** **System architecture description** 11](#_Toc437039541)

[2.1 Overview of modules / components 11](#_Toc437039542)

[2.2 Structure and relationships 15](#_Toc437039543)

[2.3 User Interface 15](#_Toc437039544)

[2.4 Hardware Interfaces 17](#_Toc437039545)

[2.5 Communication Interfaces 17](#_Toc437039546)

[2.6 Software Interfaces 20](#_Toc437039547)

[3 Detailed Description of Components 20](#_Toc437039548)

[3.1 Component Template Description 20](#_Toc437039549)

[3.2 Hardware Device: Motion Detection Unit 21](#_Toc437039550)

[3.2.1 Hardware 21](#_Toc437039551)

[3.2.2 Firmware 25](#_Toc437039552)

[3.3 Software Program: Train Navigation Service 28](#_Toc437039553)

[3.3.1 Structural 29](#_Toc437039554)

[3.3.2 Flow 31](#_Toc437039555)

[3.3.3 Behavioral 31](#_Toc437039556)

[3.3.4 Calculating Orientation from Gyroscope Data 32](#_Toc437039557)

[3.3.5 Calculating Acceleration from Accelerometer Measurements 33](#_Toc437039558)

[3.3.6 Calculation Position 34](#_Toc437039559)

[3.3.7 Requirements Traceability 36](#_Toc437039560)

[3.4 Software Program: Train Navigation Database 37](#_Toc437039561)

[3.4.1 Structural 38](#_Toc437039562)

[3.4.2 Flow 40](#_Toc437039563)

[3.4.3 Behavioral 40](#_Toc437039564)

[3.4.4 Track Geometry Measurements 41](#_Toc437039565)

[3.4.5 Requirements Traceability 51](#_Toc437039566)

[3.5 Software Program: Train Monitor Terminal GUI 53](#_Toc437039567)

[3.5.1 GUI Overview 53](#_Toc437039568)

[3.5.2 Structural 54](#_Toc437039569)

[3.5.3 Flow 60](#_Toc437039570)

[3.5.4 Behavioral 61](#_Toc437039571)

[3.5.5 Requirements Traceability 62](#_Toc437039572)

[**4** **Reuse and relationships to other products** 64](#_Toc437039573)

[**5** **Design decisions and tradeoffs** 64](#_Toc437039574)

[5.1 Database Engine Selection for Train Navigation Database 64](#_Toc437039575)

[**6** **Pseudo code for components** 65](#_Toc437039576)

[6.1 Object Position Estimation Algorithm 65](#_Toc437039577)

[7 References 68](#_Toc437039578)

# Introduction

## Purpose of this document

The computer engineering department owns a Positive Train Control Test Bed that is intended to mirror a typical train environment. The purpose of the train track is to be a teaching tool for instructing students on creating safety critical software. It is desired for the department Positive Train Control Test Bed to be able to track the location in for each train for this reason. Like subway trains, the department Positive Train Control Test Bed is completely indoors, so a Global Position System (GPS) is not possible.

The purpose of this document is to describe the design for the Train Trax Train Monitor to assist the department with tracking trains as they move along the Positive Train Control Test Bed. It will cover the design for both the desktop application and the embedded system software.

## Scope of the development project

Train Trax's primary purpose is to estimate the position of each train operating along the Positive Train Control Test Bed accurately enough to allow Train Operators schedule trains to run close enough to operation on the same section of track with minimal risk of collision. Additionally, Train Trax provides a means for Train Operators to easily control switches on the train track without the need to using any additional train control software. Train Trax is only a monitor for trains, not train control software. Furthermore, the development team is to assist the department with any modifications necessary to the Positive Train Control Test Bed to support proper operation of Train Trax, including the placement of markers on the track at pre-designated locations.

Train Trax consists of hardware that is equipped on either the train engine or rail cars to measure train movement. It also consists of software that will run on existing equipment within the department to graphically display train positions and to control movement.

A unit is attached to a rail car that is equipped with an Inertial Motion Unit (IMU) that measures the acceleration and angular velocity (rotational vectors) of the rail car as it is tugged by the train along the track. This unit, called a Motion Detection Unit, will send its collected measurements over WIFI to a train monitor terminal (i.e. computer) that will estimate the train’s position using numerical integration to solve for displacement kinematic equations. The resulting position is then displayed on the terminal as well as the layout of the track itself. RFID tags, whose position is already recorded in a database, will be used as the track markers and placed strategically throughout the track so that they can correct the position calculated from IMU measurements. Lastly, the monitor terminal displays representations of all of the switches on the track and allows the user to control them through a GUI that sends LOCONET messages to the track's switch controllers, which then control relays to change a switch’s state. Train control software, such as JMRI, is expected to be used to control/throttle the movement of the train via LOCONET messages to the Train Command Station.

## Definitions, acronyms, and abbreviations

**Digital Command Control (DCC)**

Digital Command Control protocol which is a electric signaling protocol used to control train engines on a train track through the rails.

**Java Model Railroad Interface (JMRI)**

Popular open-source software suite for controlling model trains.

**Inertial Motion Unit (IMU)**

A hardware device often composed of an accelerometer and a gyroscope used to perform dead-reckoning of the position of objects based on measurements of effects of forces acting on an object in space.

**LocoNet**

An Ethernet-link proprietary communication protocol created by DigiTrax for full train and track layout control of model train sets.

**Position**

A description of where a given object is located on the Position Train Control Test Bed. It uses a relative coordinate system based on the distance from a fixed point on the table.

**Positive Train Control Test Bed**

A model train system designed to scale to represent actual railway systems. Its purpose is to facilitate the testing, design, and training of train control systems without the risk of associated performing these activities on live trains, such as bodily injury and costs for scheduling and operating full scale trains.

.

**Rail Car**  
Simple wheeled container that is attached to the train to carry cargo.

**Railway System Owner**

The entity that owns Positive Test Control Test Bed.

**Radio Frequency Identification (RFID)**

Data exchange method that relies on the properties of induction to read information imprinted on a device when in close proximity.

**Track**  
The track is a pair of metal rails that the train runs on top of to move. It provides both power and control signals to the train. It is divided into different physical pieces called sections to simplify its assembly.

**Track Block**

A segment of the entire track of the test bed, which has been divided and identified into segments by the Train Technician and Train Operator, which is used to highlight areas of interest by these individuals and to divide the track into regions from which trains can go in different directions on the track. In practice, a block is Track Circuit Block. It is a single element where the Positive Train Control Test Bed Can Detect whether one or more trains is on it or not.

**Track Marker**  
Special hardware placed at different spots on the track to highlight places of interest on the track. Examples of train markers include RFID tags that are read by the train as it moves along the track, and track sections that signal when one or more trains are present.

**Track Switch**Devices on the track to control the direction of train engine movement by changing the sections of track that are connected together.

**Track Switch Controller**

A hardware device that is the bridge between hardware that physically controls switches of the test bed and software being used to remotely control the test bed. It is attached to the track that translates requests from operators to control track switches on the test bed into signals to switch relays that move the switches into different positions.

**Train**

A to-scale model of a commercial train engine. It is the primary vehicle used to move along the test bed

**Train Command Station**

A hardware device that is the bridge between hardware that physically controls trains of the test bed and software being used to remotely control the test bed. It is attached to the track that translates requests from operators to control the train into control signals that the train understands.

**Train Control Terminal**

The equipment, such as a laptop, used by the system to allow operators to control trains that belong to the test bed.

**Train Monitor Development Team**

A group of people who have been commissioned by the Railway System Owner to create a system for tracking the movement of trains along the railways system real time.

**Train Monitor Terminal**

The display equipment, such as a laptop, used by the system visually display to operators information about the test bed.

**Train Occupancy Detector**

A hardware device that is the bridge between hardware that physically detects when one or more trains are on a section of track and software being used to report train locations. It is attached to the track and uses changes in current draw that occur when one or more trains are being powered by a track block in order to detect train occupancy in that block. Lastly, it can issue messages for when a train is entering or exiting a track block.

**Train Operator**

A person or machine that controls one or more of the trains on the Positive Train Control Test Bed.

**Train Technician**

A train technician is a person who maintains the Positive Train Control Test Bed.

## Overview of document

The remainder of the SDD will provide an overview of the system architecture and then describe the detailed design of each of the system components.

# **System architecture description**

## Overview of modules / components



Figure 1 Positive Train Control Test Bed Without train trax

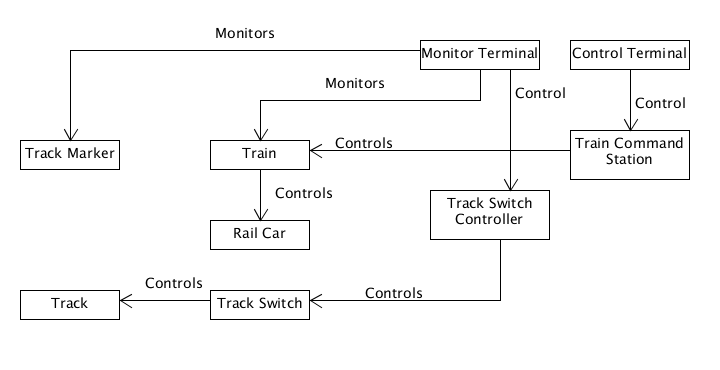


Figure 2 Control Flow of Positive Train Control Test Bed without Train Trax

Figure 1 and Figure 2 show how the Positive Train Control Test Bed currently operates without Train Trax. In the existing Positive Train Control Test Bed, track markers are actually the track blocks themselves. There is hardware on the track, called a Train Occupancy Detector, to detect when one or more trains enter a block and when there are not any trains on a block based on the current draw on the track block. The Train Occupancy Detector sends messages for transitions for when the block is occupied (one or more trains on the block) and unoccupied (no trains on the block). When the track crosses into a track marker (i.e. track block), information about the marker that was crossed is relayed to the Monitor Terminal so that it can update the train’s last known position based on the known position of the track marker. Since the rail cars are attached to the train, the train controls where the rail cars move. The Train Command Station controls the speed of the train the direction that it moves along the track: either backward or forward. The Track Switch Controller controls track switches which in turn change the configuration of the track so that the path that the train moves along the track can be controlled.

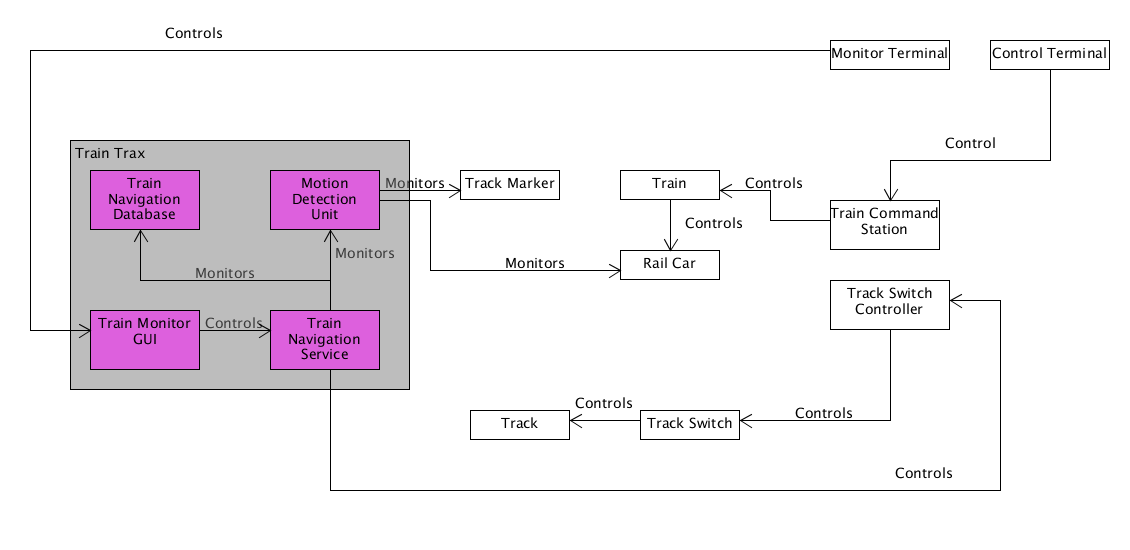


Figure 3 Control Flow of Train Trax Components with Positive Train Control Test Bed

Figure 3 shows how the test bed works with Train Trax incorporated. The Train Trax project consists of four top level components: the Motion Detection Unit, Train Navigation Service, Train Navigation Database, Train Monitor Terminal GUI. The Motion Detection Unit is the hardware that is used to measure train movement. It uses sensors to measure acceleration, and orientation of the train as well as crossing of track markers. The Train Navigation Service is a background service that is used to determine the position of each train and to control switches on the track. The Train Navigation Database stores all of the navigation information collected for the track and trains, including details on the geometry of the track (location of switches, sections of track, etc.). The Train Monitor Terminal GUI is the primary display that Train Operators use to interact with the system. It displays the train position of trains on the track and processes requests from users to change track switches.

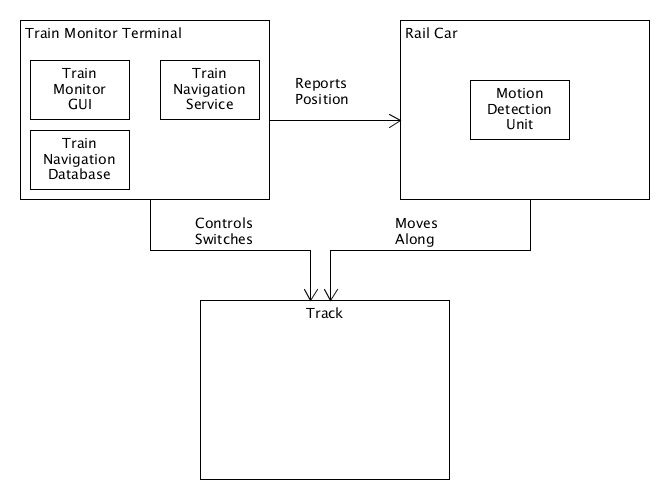


Figure 4 Placement of System Components Within the Positive Train Control Test Bed

Figure 4 shows how Train Trax integrates into the existing Positive Train Control Test Bed. The Train Navigation Service, Train Navigation Database, and Train Monitor GUI operate as software packages that run from the Train Monitor Terminal. They work together to allow the Train Monitor Terminal fulfill its responsibility to track trains and control switches on the track. Since there is very little space on the train engine itself, the movement of the train must be observed through an attached rail car instead. The Motion Detection Unit is equipped onto the Rail Car so that it can measurement movement of the rail car as it moves along the track.

## Structure and relationships

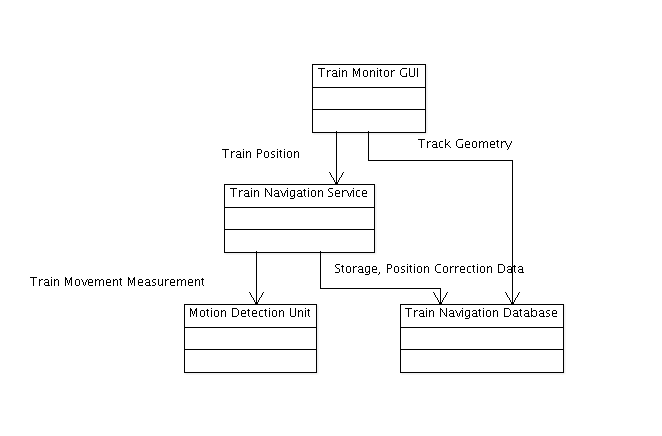


Figure 5 Train Trax system Component relationships

Figure 5 shows how the parts of Train Trax depend upon each other. The Train Navigation Service depends on the Motion Detection Unit to provide the raw measurements of train movement for the navigation calculations for each train. It also depends on the Train Position Database to save its calculated train position estimates and to lookup the positions of Track Markers. The Train Navigation GUI depends on the Train Position Database to render the history of each train's movement, and for track geometry information, such as the location of switches, track sections, and track markers. The GUI also depends on the Train Navigation Service to control track switches and to determine the current position of each train.

## User Interface

The following section describes an overview of the graphical user interface for the Trax Train Monitor project. The graphical user interface will consist of all windows that are required for the user to enter, store, and view information associated with the Trax system. A detailed design description of functionality and an outline of the user interface will be provided in Section 3.4.5 and for all of the views that will make up the graphical user interface including how most of the UI views will look, the functionality that will be executed after the interaction with the UI (i.e. a button push), and the communication between modules.

The UI will include the following types of views:

* Main Menu View
  + Initial view at startup. Presents the operations available to the system. This will show the layout of the track without any trains on it. It should resemble the general shape of the track and display switches and the division of train blocks on the track.
* Train Monitor View
  + This is the view for the Train Operator to observe trains as they move along the track. This is primary window that will be used by train tracks. Train Operators can also change the state of switches by tapping on switches from this display.
* Train History View
  + This view allows the Train Operator to learn about where trains have travelled during a given span of time. It should be able to list a table of all of the reported movements of the train. As a bonus, it should graphically display the path travelled by each train.

## Hardware Interfaces

|  |  |  |
| --- | --- | --- |
| Type | Description | Purpose |
| RS-232 | Popular protocol for serial communication across a DB-9 serial cable. | Used to Program Motion Detection Unit. Used to connect RFID Reader to Motion Detection Unit |
| Serial Peripheral Interface (SPI) | Three-wire serial protocol used to connect a CPU to peripherals. | Used to connect Wireless Ethernet Module to Motion Detection Unit |
| Inter-Integrated Circuit (I2C) | Two-wire serial protocol used to connect a CPU to peripherals | Used to connect IMU and Optical Sensor to Motion Detection Unit. |
| Universal Serial Bus (USB) | Popular four-wire serial protocol used for connection devices to PCs. | Used to connect RFID reader for testing.  Used by the Train Monitor Terminal to send LocoNet messages to test bed hardware, such as trains and switch controllers. |

Table 1 Hardware Interfaces

## Communication Interfaces



Figure 6 Train Trax Communication diagram

|  |  |  |
| --- | --- | --- |
| Type | Description | Purpose |
| IEEE 802.11 abgn Wireless Ethernet | Popular 2.4 GHz Radio Physical and Datalink Protocols for Exchanging Information Between Machines | Establishes communication with the Train Command Station and Track Switch Controller.  Delivers real-time measurements of train movement from the Motion Detection Unit |
| Transmission Control Protocol (TCP) / Internet Protocol (IP) | Standard Transport / Network Protocols for transferring data across the Internet and within Local Networks. | Transport for measurements from the Motion Detection Unit.  Transport for communication with the Train Navigation Database. |
| LOCONET | Ethernet-like messaging protocol used to monitor and control model train systems. | Used for communication with the Train Command Station and Track Switch Controller. |

Table 2 Communication Interfaces

As shown in Figure 6 and Table 2, most of Train Trax’s communications interfaces are for communication with different parts of itself. LOCONET messages are the primary vehicle for Train Trax to exchange information with parts of the test bed itself. These messages are transported via Wireless Ethernet. For this to happen, the Motion Detection Unit will authenticate onto the University Wi-Fi network or impersonate an authenticated device. Communication between the Motion Detection Unit attached to a rail car and the Navigation Service running within the Train Monitor Terminal is to be custom message set created specifically for communicating with the Motion Detection Unit called MDU messages. MDU messages will be transported using a base station to relay messages from the Motion Detection Unit to the Train Monitor Terminal and vice versa. If possible, the base station will be eliminated in the future so that the Train Monitor Terminal directly exchanges information with the Motion Detection Unit.

## Software Interfaces

|  |  |  |
| --- | --- | --- |
| Type | Description | Purpose |
| Java Runtime API | Standard libraries provided by the Java Runtime Environment | Used to assist in Train Position Calculations by the Train Navigation Service  Used to render graphical displays for the Train Navigation GUI.  Provides interfaces for retrieving train measurements and other network data from Motion Detection Unit |
| Structured Query Language (SQL) | Standard language used to query and interact with databases. | Used for communication with the Train Navigation Database to access and store information. |
| Java Database Connectivity (JDBC) | Standard interface for connecting Java to a database. | Used to provide SQL access to the Train Navigation Database to the Train Navigation Service and the Train Navigation GUI. |

Table 3 Software Interfaces

# Detailed Description of Components

## Component Template Description

Each system component of Train Trax provides the following information to deliver different representations of how each component is architected:

* Structural
* Identifies the primary entities that used to create the component, and the responsibilities for each.
* Should include Class Diagrams
* Behavioral
* Describes how subcomponents interact with each other for each of the relevant use case scenarios.
* Flow
* Describe control flow: which subcomponents are responsible for controlling other subcomponents; which subcomponents are responsible for monitoring other subcomponents.
* Describe data flow: what type of data is exchanged between subcomponents.

## Hardware Device: Motion Detection Unit

The Motion Detection Unit is a device that is attached to a rail car attached to the train to measure the motion of the train, including the acceleration and angular velocity of the attached rail car. The Train Trax project includes both implementing the hardware and software for the device.

### Hardware

The Motion Detection Unit is essentially a composition of commercial off-the-shelf (COTS) hardware modules. This approach was taken to minimize the amount of time necessary for testing the hardware and to minimize the risk of hardware unreliability when testing the system as a whole. The motherboard is the brains of the device and is where all of the main processing of the device is coordinated. An Inertial Measurement Unit (IMU) is used to collect raw acceleration and angular velocity measurements. A RFID Tag Reader is used to detect when the object that the device is attached to crosses an RFID Tag. Lastly, a wireless communication module is used to send collected measurements from the device to an external machine where they can be processed.

#### Components

##### Motherboard

**Arduino Pro Mini 328 - 3.3V/8MHz -** [Link](https://www.sparkfun.com/products/11114)

Dimensions: 0.7x1.3" (18x33mm)

Notes: 3.3V operating voltage. Has voltage regulator that can receive up 12V.

I2C and SPI interfaces

##### Sensors

**SparkFun 6 Degrees of Freedom IMU Digital Combo Board - ITG3200/ADXL345 -** [Link](https://www.sparkfun.com/products/10121)

Range: Accel:±2, 4, 8, 16g, Up to 12-bits precision per g (+/- 0.01 m/s^2) Gyro:±2000°/s (+/- 0.01 radians per second)

Sensors:

ADXL345 accelerometer    [Datasheet](https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf)

ITG-3200 MEMS gyro.    [Datasheet](https://www.sparkfun.com/datasheets/Sensors/Gyro/PS-ITG-3200-00-01.4.pdf)

Dimensions: None given. Smaller than a quarter

Notes: I2C interface with no onboard processing. 3.3V. Data update rate 4-8000 Hz

**SparkFun Triple Axis Accelerometer and Gyro Breakout - MPU-6050 -** [Link](https://www.sparkfun.com/products/11028)

Range: Accel:±2, 4, 8, 16g, Up to 12-bits precision per g (+/- 0.01 m/s^2) Gyro:±250, 500, 1000, 2000°/s (+/- 0.01 radians per second)

Sensors:

    MPU-6050 MEMS 3-axis gyroscope and 3-axis accelerometer     [Datasheet](https://cdn.sparkfun.com/datasheets/Components/General%20IC/PS-MPU-6000A.pdf)

Dimensions:  1 x 0.6 x 0.09" (25.5 x 15.2 x 2.48mm)

Notes: I2C output. 3.3V. Register update rate 4-8000 Hz

**SparkFun USB RFID Reader** [Link](https://www.sparkfun.com/products/9963)

Supports ID-3LA, ID-12LA, and ID-20LA readers

Access to Serial Pins so that RS232 can be used to connect to hardware.

##### Wireless Communication Module

XBee Wifi Module [Link](https://www.sparkfun.com/products/12568)

IEEE 802.11 wireless communication

RS-232 and SPI interfaces

##### Rationale for Hardware Selection

The selected sensors provide the most common sensors available in current IMUs. The selected Arduino has both I2c used by the selected sensors and SPI allowing for potential integration of additional sensors at a later time. The Arduino operates on the 3.3V used by the sensors and contains a voltage regulator that can have an input of up to 12V removing the need for any additional voltage regulation or steeping hardware.

#### Structural

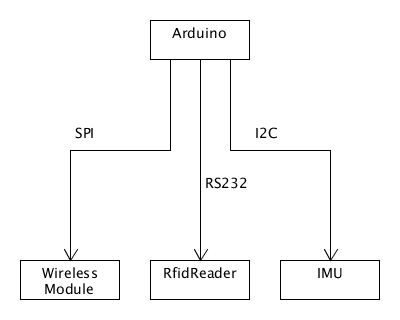


Figure 7 Motion Detection Hardware Module Assembly

#### Flow

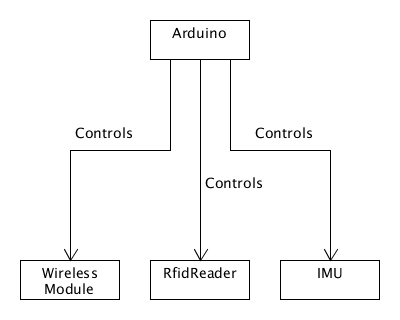


Figure 8 Motion detection unit hardware control flow

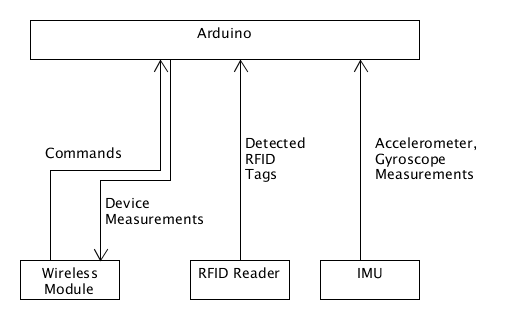


Figure 9 Motion Detection unit hardware data flow

#### Behavioral

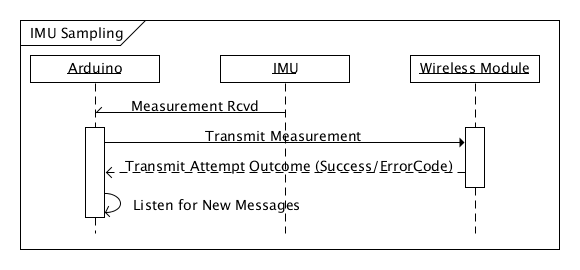


Figure 10 Motion Detection Unit Hardware IMU Read Sequence Diagram

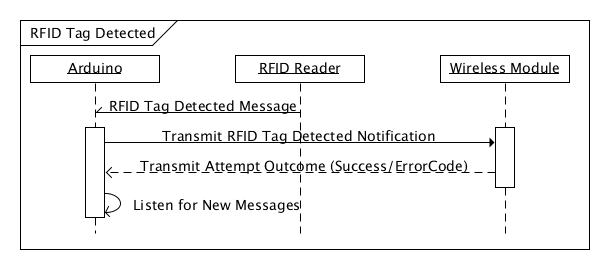


Figure 11 Motion Detection Unit Hardware RFID Tag Read Sequence Diagram

#### Requirements Traceability

|  |  |
| --- | --- |
| **Hardware Component** | **Associated Requirements** |
| IMU | MDU-1010, MDU-1020, MDU-2000, MDU-2010, MDU-2020 |
| RFID Reader | MDU-1040 |
| Motherboard | MDU-1010, MDU-1020, MDU-1040, MDU-3000 |
| Wireless | MDU-3000 |

### Firmware

The Motion Detection Unit is uses a hierarchical design pattern where most modules run concurrently and two modules control the system. The first controlling module is a relay module that orders messages that it receives from the drivers in a queue to form a stream of messages that need to be forwarded to target subscribers on remote machines. The second controlling module brokers commands from remote clients. It listens for commands and forwards them to the intended target modules. The Drivers run independently to communicate with their respective hardware devices.

#### Structural

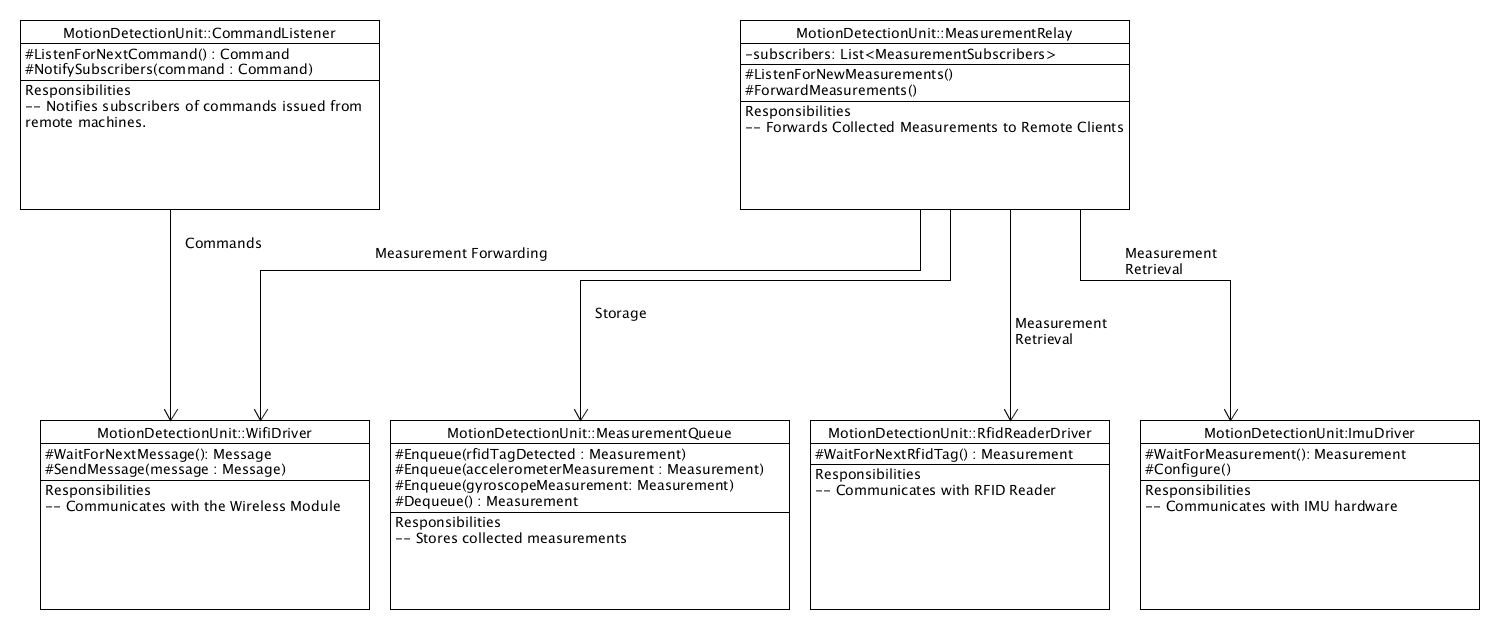


Figure 12 Motion Detection Unit Software Entity State Diagram

Figure 12 shows the critical software components involved with operating the Motion Detection Unit. The MeasurementRelay module is the primary module of the Motion Detection Unit. It takes measurements that have been collected by the device and forwards them to clients listening for them. It acts as the relay module referenced earlier. This class is responsible for handling any authentication necessary for joining the wireless network. The MeasurementQueue stores all of the measurements collected by the device. Its purpose is to temporarily house measurements until a connection can be made to deliver them to clients. In practice, items should be typically stored in the queue for fractions of a second. The IMUDriver module handles all communications with the Inertial Measurement Unit (IMU). The RfidReaderDriver module handles all communications with the RFID Tag Reader equipped on the device. The WifiDriver module handles all communications with the Wireless module used to communicate with other machines on the network, especially clients that are listening for broadcasts of the latest measurements. The CommandListener module is responsible for passing any instructions received over Wi-Fi to the appropriate modules. It is the command broker referenced earlier.

#### Flow

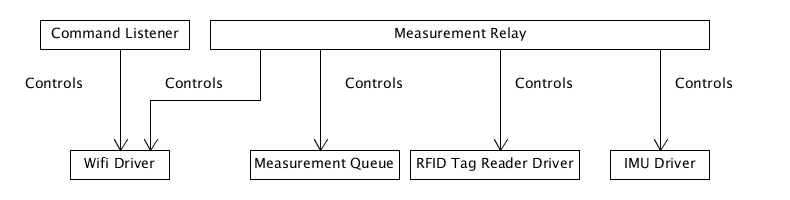


Figure 13 Motion Detection Unit Software Control Flow

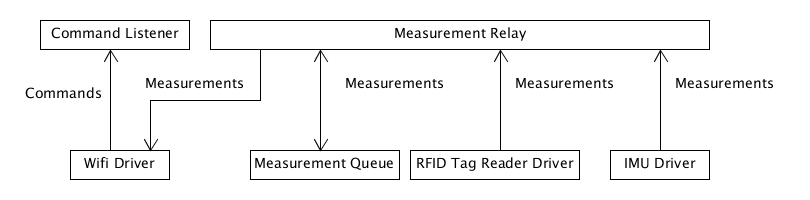


Figure 14 Motion Detection Unit Software Data Flow

#### Behavioral

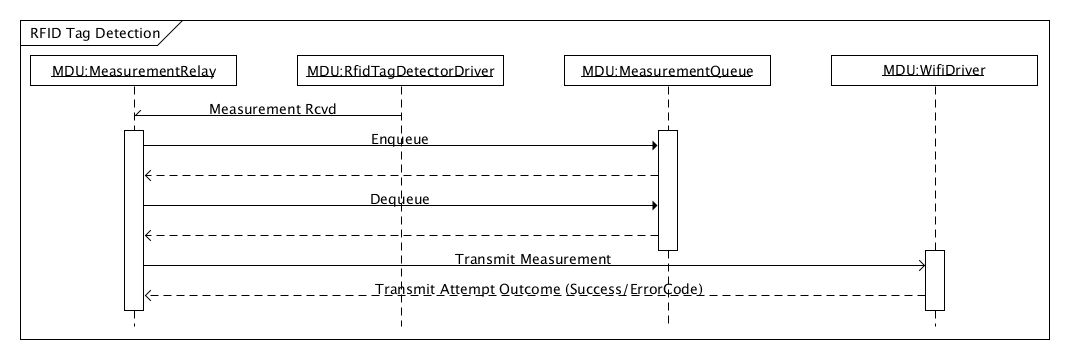


Figure 15 Motion Detection Unit Software IMU Read Sequence Diagram

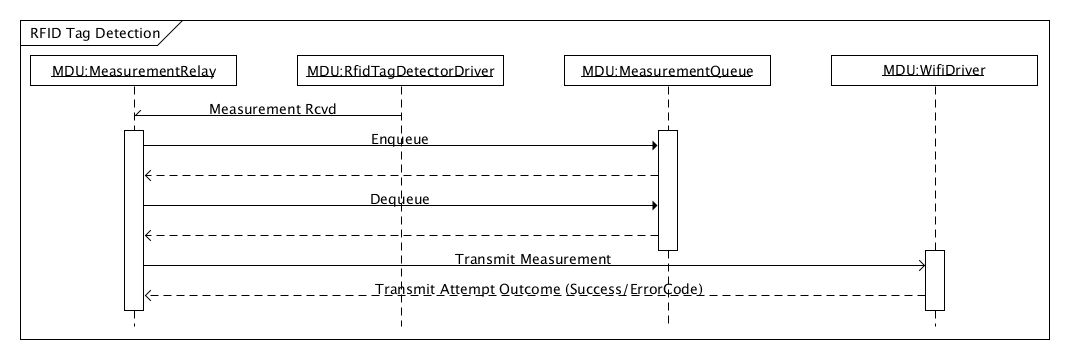


Figure 16 Motion Detection Unit Software RFID Tag Detection Sequence Diagram

#### Requirements Traceability

|  |  |
| --- | --- |
| **Software Component** | **Associated Requirements** |
| IMU Driver | MDU-1030, MDU-1050 |
| RFID Reader Driver | MDU-1060 |
| Measurement Relay | MDU-1030, MDU-1050, MDU-1060 |

## Software Program: Train Navigation Service

The Train Navigation Service is the heart of the Train Trax. It is a stand-alone program for reporting train positions on the Positive Train Control Test Bed and for controlling switches on the test bed track. It is responsible for estimating the location of each train on the test bed. Also it is responsible for controlling the direction that trains move along the track by controlling the switches on the track. The Train Navigation Service is organized as a hierarchy of classes where there is one primary object acting as the root of the hierarchy that coordinates all of the actions of the support classes to achieve all of the duties of the service.

### Structural

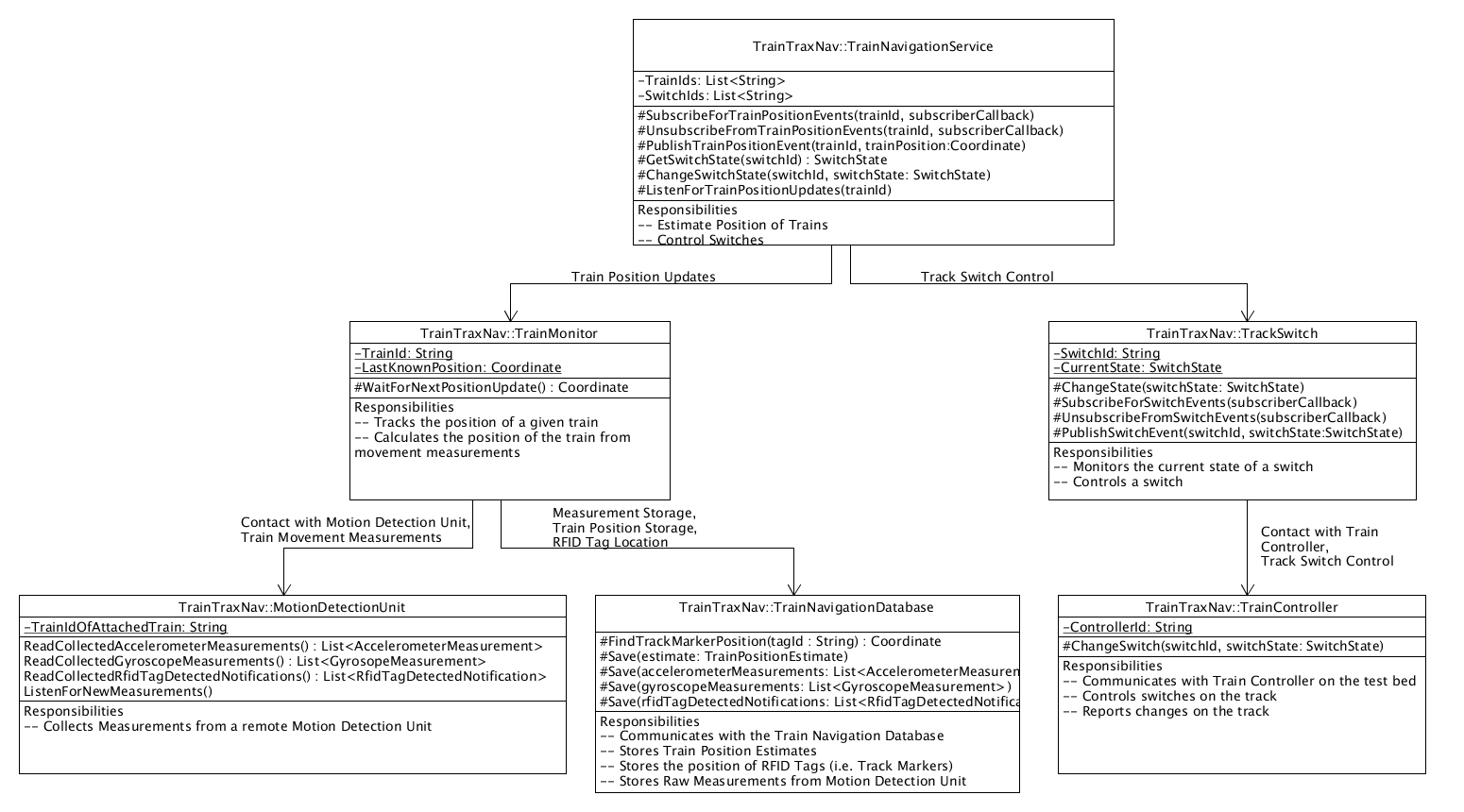


Figure 17 Train Navigation Service Entity State Diagram

As shown in Figure 17, the TrainNavigationService is the primary object responsible for the responsibilities of the service and all subsequent classes support this class. The TrainNavigationService class acts as the front end of the service. It is the primary object that developers using the service interact with. It is responsible allowing uses to control track switches and reporting train position estimates to external locations. The TrainMonitor class is responsible for tracking / estimating the position of a single Train on the Positive Train Control Test Bed. The MotionDetectionUnit class is a class that represents a single Motion Detection Unit hardware device that is equipped onto a rail car that is attached to a given train. It is responsible for coordinating all information exchanged between the target Motion Detection Unit and the service. It is the class that collects the asynchronous messages that are issued from the Motion Detection Unit hardware so that the rest of the program may use them for later processing, such as calculating a train’s position.The TrackSwitch class is a class that represents a single switch on the Positive Train Control Test Bed. It is responsible to keeping track of the current state of the switch and changing the state of the switch upon request. The TrainController class is a class that represents a Track Switch Controller devices equipped on the Positive Train Control Test Bed to control switches based on commands encoded as LOCONET messages. It is responsible for coordinating all information exchanged between Track Switch Controller devices and the service. The TrainNavigationRepository is a datastore for all of the information that pertains to tracking trains around the Positive Train Control Test Bed, and about the track of the test bed itself. This is the primary object that interacts with any database used by Train Trax.

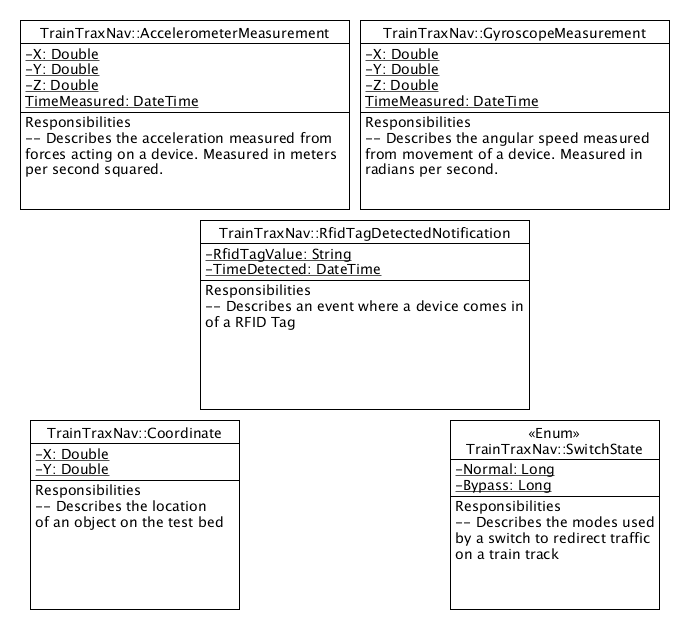


Figure 18 Train Navigation Service Data Classes

The classes shown in Figure 18 represent all of the data that gets exchanged between classes. The AccelerometerMeasurement class represents a single sample measured from an accelerometer on the Motion Detection Unit. The GyroscopeMeasurement class represents a single sample measured from a gyroscope on the Motion Detection Unit. The RfidTagDetectedNotification class represents a single instance in time where the Motion Detection Unit came in contact with a RFID tag. The Coordinate class represents the position of an object on the test bed. The SwitchState class represents how a given switch on the track is configured to direct train traffic along the track.

### Flow

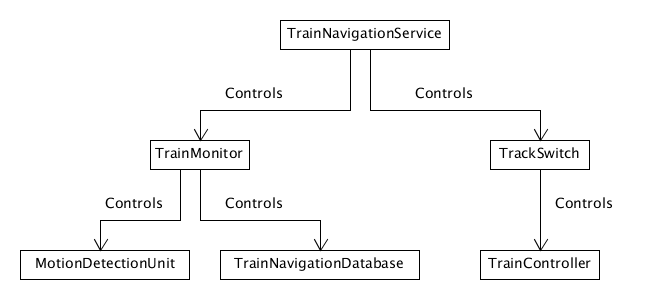


Figure 19 Train Navigation Service Control Flow Diagram

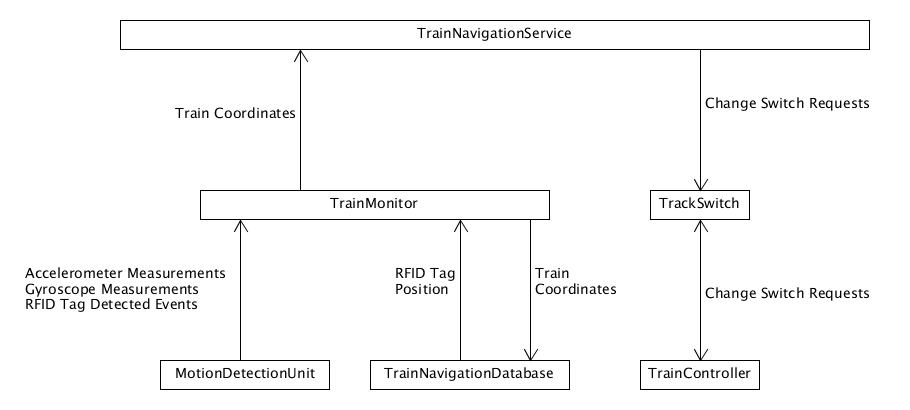


Figure 20 Train Navigation Service Data Flow Diagram

### Behavioral

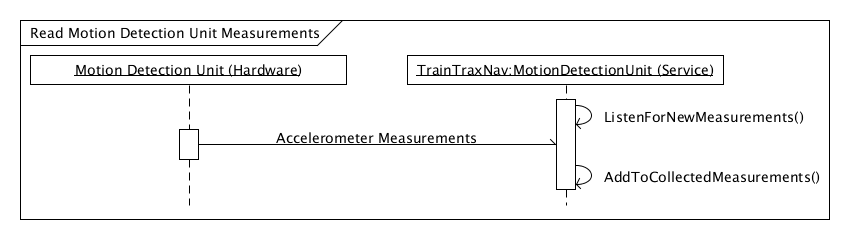


Figure 21 Sequence Diagram for Reading Measurements from the Motion Detection Unit

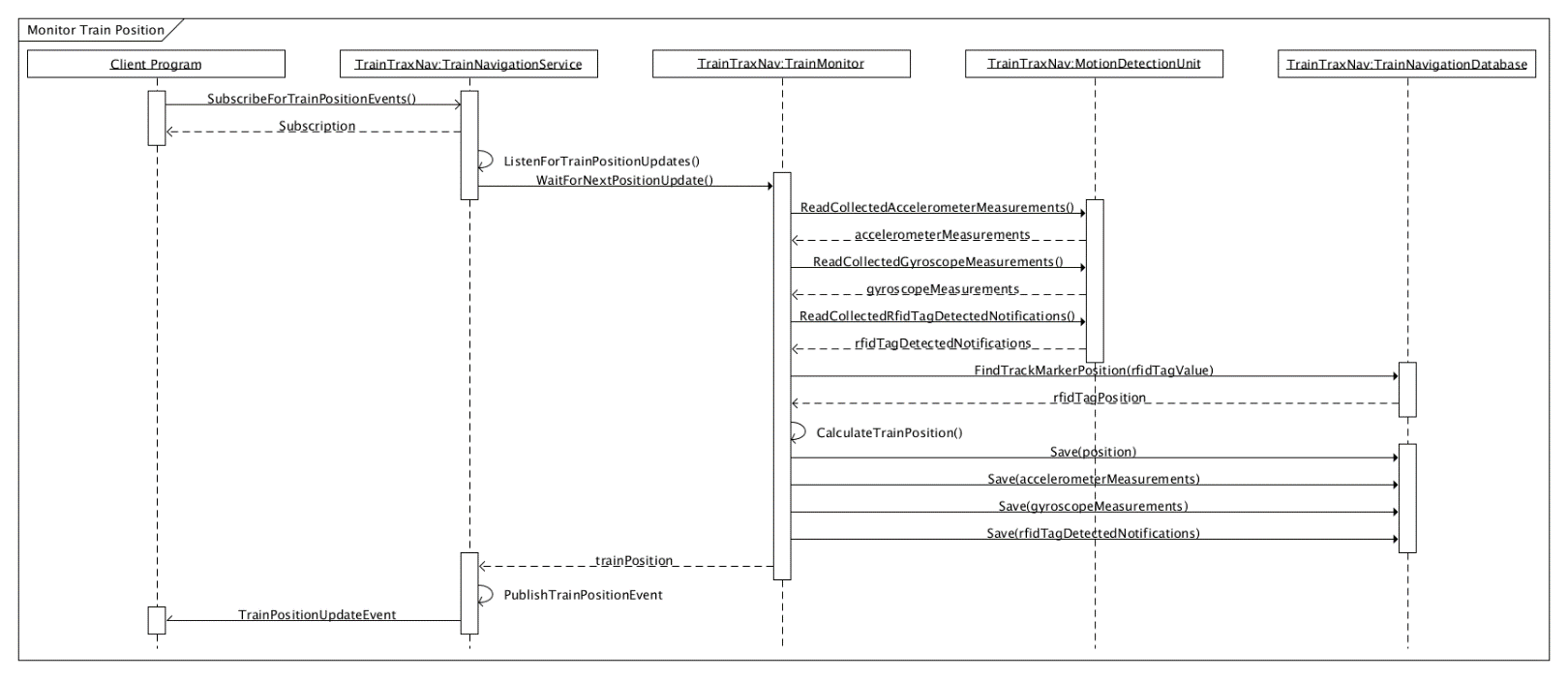


Figure 22 Sequence Diagram for Monitoring a Train’s Position on the Test Bed

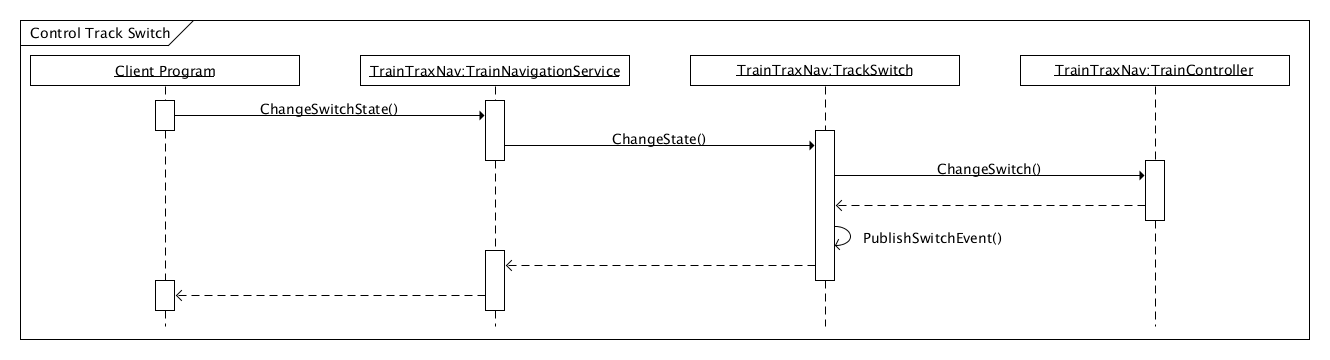


Figure 23 Sequence diagram for Controlling a Switch on the Test Bed

### Calculating Orientation from Gyroscope Data

Gryoscopes measure the rate of rotation along 3 independent axes on the device. The problem is that we are interested in the rate of rotation along 3 different independent axes: where X, Y, and Z are relative to the Earth, not the device. Each of these groups of axes are referred to as an inertial reference frame because they can be used to express the movement of a body in space relative to something (hence the reference) else. For the purposes of calculating the position of trains, we are using an Earth-fixed inertial reference frame. In particular, the North-East-Down (NED) inertial reference frame where North is Y, East is X, and pointing toward the center of the Earth is Z. The 3 independent axes on the device are referred to as the Body reference frame. So in order to do anything useful with the gyroscope measurements, we need to transform the measurements from the body reference frame to the NED inertial reference frame. Figure 24, courtesy of CH Robotics, shows how this conversion is made.

p, q, r are the total angle changes observed along the body frame x, y, and z axes respectively

You get p, q, and r. By multiplying the angular rates from the gyroscope for x, y, and z times the change in time respectively.

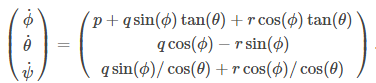


Figure 24 Body Frame Angular Velocity to NED Inertial frame Angular Velocity

### Calculating Acceleration from Accelerometer Measurements

Just like the gyroscope, accelerometer measurements are made using the 3 independent axes on the device (i.e. the Body reference frame). As a result, in order to use the measurements to determine the acceleration observed relative to the Earth, another transformation must be made from the body frame to the NED inertial reference frame. Figure 25, courtesy of CH Robotics, describes the transformation. A rotation matrix is used express the conversion from the body frame to another reference frame (in this case the NED inertial frame). We need the NED-based rotation vectors that we calculate from the gyroscope measurements to account for the fact that the orientation of the device is changing as it is moves. In other words, the conversion from the body frame to the inertial reference frame is a moving target over time.

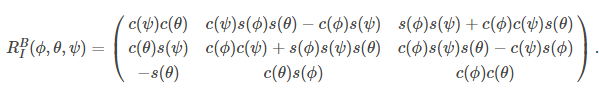


Figure 25 Body Frame Vector to NED Inertial frame vector Transform

Fee, Theta, and Psi in the figure represent the angle of rotation along the X, Y, and Z axes of the NED inertial frame respectively. The acceleration vector measured by the accelerometer is multiplied by the matrix to produce the acceleration vector related to the NED inertial reference frame.

### Calculation Position

The rectangular sums method of numerical integration approximation is used to calculate the position of an object from the following kinematic equations:

Let A(t) = Acceleration Vector of an Object as a function of time

Let V(t) = Velocity Vector of an Object as a function of time

Let R(t) = Relative Position Vector of an Object as a function of time.

dt= Change (or delta) in time.

t0=initial point in time calculations start.

v0 = initial velocity vector

r0 = initial position vector

V(t) = Integral Of A(t)dt

R(t) = Integral Of V(t) dt

V(t) = Let T = from t to t0, SUM( A(T)\* ( T-t0)) + v0

R(t) = Let T = from t to t0, SUM( V(T)\* ( T-t0)) + r0

#### Error Estimation

Acceleration Error from Accelerometer

Acceleration Error from Gyroscope

##### Accelerometer

Assuming that accelerometer measurements are consistently off by 0.01 meters per second squared (the best possible accuracy from our selected hardware) for each sample and there are 1000 samples a second, the following describes the amount of error we can expect to accumulate after running for 3 seconds.

Assuming a maximum acceleration of 2 g,

Total Error = 0.5 \* a \* (Avg Error) \* T^2 = 0.5 \*(2)\*0.01\*(3)^2 = 0.01 m/s^2 \* (9 s^2) = 0.09 m = 9 cm \* (1 inch / 2.54 cm) = **3.54 inches**

##### Gyroscope

Let w = 1 : the maximum angular velocity experience by the train

Let ang = the change in orientation.

Let dt = change in time

Let Eang = the error in the change of orientation

Let Werror = Average % error in angular velocity measured

ang = w\*dt

Eang = w\*Werror\*dt

Let a = the maximum acceleration experienced.

Let Ea = the error in the change of acceleration.

Ea = a\*sin(Ea);

This is because biggest amount of change due to a change in angle is closest near zero. The formula above assumes a unit circle and that all force is acting on the x-axis. This calculates worst-case how much force would bleed into the y-axis if orientation corrections from the gyroscope are off by Ea.

Let’s calculate the total error the same scenario used for the accelerometer error. The scenario of interest is for a 3 second interval and with Werror equal to 1% (Best possible from gyroscope).

Eang = (1 radians per second)\*(0.01)\*(3 seconds) = 0.03 radians

Let a = 2 g (g=acceleration from Earth’s gravity) = 2\*(9.81 m/s^2);

Ea = a\*sin(Ea) = 2\*(9.81 m/s^2) \* sin(0.03 radians) = 0.6 m / s^2

This dwarfs the 0.01 m/s^2 error from the accelerometer.

Total Error = 0.5 \* a \* t^2 = 0.5 \* (0.6 m/s^2) \* (3 s)^ 2 = **2.65 meters**

In other words, the project is feasible to estimate position as long as we have very accurate orientation data. This means that it is important for us to make sure that our orientation measurements are as accurate as possible. A technique such as complementary filters or Kalman Filters may need to be used against gyroscope measurements to reduce the error further if this level of error is experienced in practice.

A secondary set of sensors, such as a secondary IMU, may be equipped onto the device.  
Kalman and complementary filtering are techniques that will allow errors from multiple noisy sensors to cancel out so that you have a lower effective amount of error. (i.e. Multiple measurements in the same point and time are used to correct for the error reported by each individual measurement). Furthermore, high pass and low pass filtering, can help eliminate error from biases in the sensors themselves. Worst case, the distance between RFID tags is setup to be small enough that the truth data from measurements alone can be used for orientation approximation if needed by using curve fitting.

### Requirements Traceability

| Requirement Number | Feature | Software Class |  |
| --- | --- | --- | --- |
| Requirement Text | | | |
| TNE-1000 | Report Train Position (Rollup) | TrainNavigationService,  TrainMonitor |  |
| The Train Navigation Service shall calculate the position of a given train along the test bed. | | | |
| TNE-1010 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service shall calculate the position of a given train relative to a fixed point on the test bed. | | | |
| TNE-1020 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service shall calculate the position of a given train within a radius that allows for two trains to run on a 14-inch track block. | | | |
| TNE-1021 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service shall calculate the position of a given train within a 6-inch radius of its actual position. | | | |
| TNE-1022 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service should calculate the position of a given train with a 2-inch radius of its actual position. | | | |
| TNE-1030 | Report Train Position (Rollup) | TrainMonitor, MotionDetectionUnit |  |
| The Train Navigation Service shall use measurements from the Motion Detection Unit to calculate the position of a given train. | | | |
| TNE-1031 | Report Train Position | TrainMonitor, MotionDetectionUnit |  |
| The Train Navigation Service shall calculate the acceleration of the train from the Motion Detection Unit’s acceleration measurements. | | | |
| TNE-1032 | Report Train Position | TrainMonitor, MotionDetectionUnit |  |
| The Train Navigation Service shall calculate the velocity of the train from the Motion Detection Unit’s angular velocity measurements and acceleration measurements. | | | |
| TNE-1033 | Report Train Position | TrainMonitor, MotionDetectionUnit,  TrainNavigationDatabase |  |
| The Train Navigation Service shall estimate the orientation of the train from the Motion Detection Unit’s angular velocity measurements. | | | |
| TNE-1040 | Report Train Position | TrainMonitor, MotionDetectionUnit,  TrainNavigationDatabase |  |
| When the Motion Detection Unit of a train crosses a track marker, the Train Navigation Service shall update the position of that train. | | | |
| TNE-1050 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service should estimate of the accuracy of its train position calculations. | | | |
| TNE-2000 | Report Train Position (Rollup) | TrainMonitor |  |
| The Train Navigation Service shall report the location of a given train for all modes of operation of the train. | | | |
| TNE-2010 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service shall report the location of a given train when it is moving forward. | | | |
| TNE-2020 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service shall report the location of a given train when it is at rest. | | | |
| TNE-2030 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service shall report the location of a given train when it is in reverse. | | | |
| TNE-4000 | Control Track | TrainNavigationService,  TrackSwitch,  TrainController |  |
| The Train Navigation Service shall issue commands to the rail system track switch controller to change the state of switches on the rail system. | | | |
| TNE-5000 | Report Train Position | TrainMonitor, TrainNavigationDatabase |  |
| The Train Navigation Service should detect any discrepancy between its estimates of train position and know track locations (i.e. significant differences between calculated position and RFID tag listed position or incorrect RFID tag order). | | | |
| TNE-6000 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service should detect when the train is at rest. | | | |
| TNE-7000 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service should detect when the train reverses direction. | | | |
| TNE-8000 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service should report the time of the last position correction. | | | |
| TNE-9000 | Report Train Position | TrainMonitor |  |
| The Train Navigation Service shall estimate the speed of a given train on the test bed. | | | |

## Software Program: Train Navigation Database

The Train Navigation Database is used to store all track layout and positioning information for calculating a train’s position. It is a program responsible for persisting data and providing classes to reading/writing data. It also stores all of the train positions reported by Train Trax to create a history of movement of each train over the period of time observed. It also provides the location of each RFID tag (i.e. Track Marker) placed on the track so that train position corrections can be made each time a train crosses a RFID tag.

### Structural

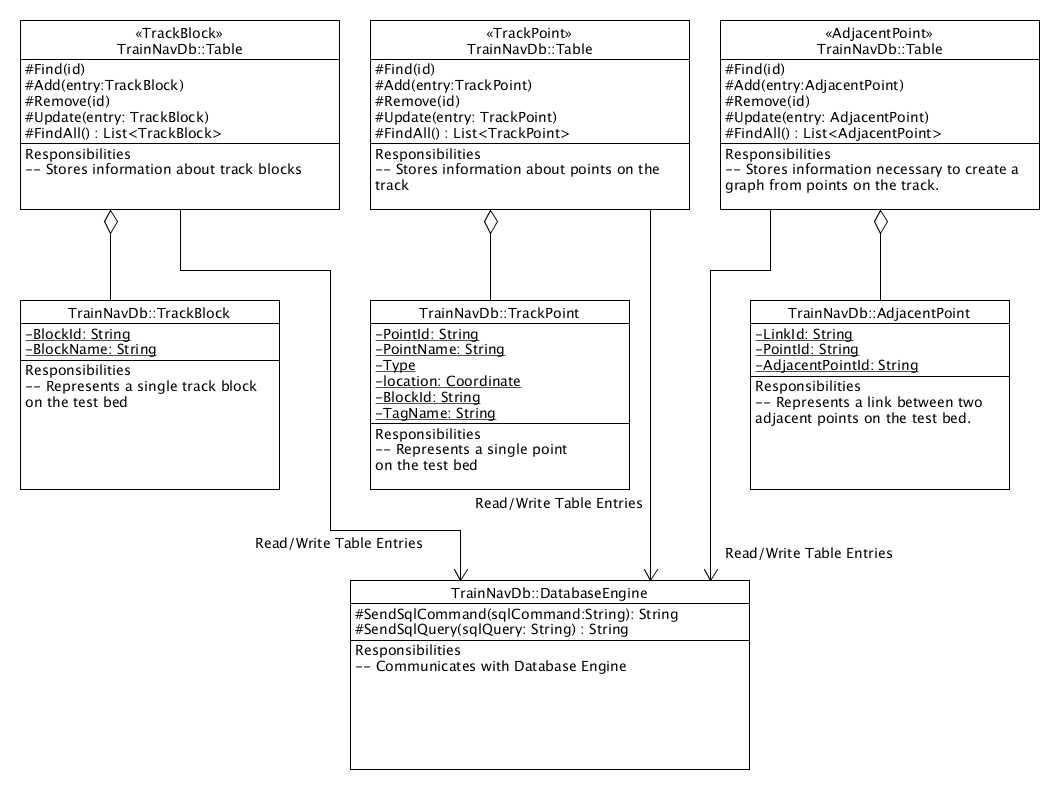


Figure 26 Class Diagram for Track Geometry entities

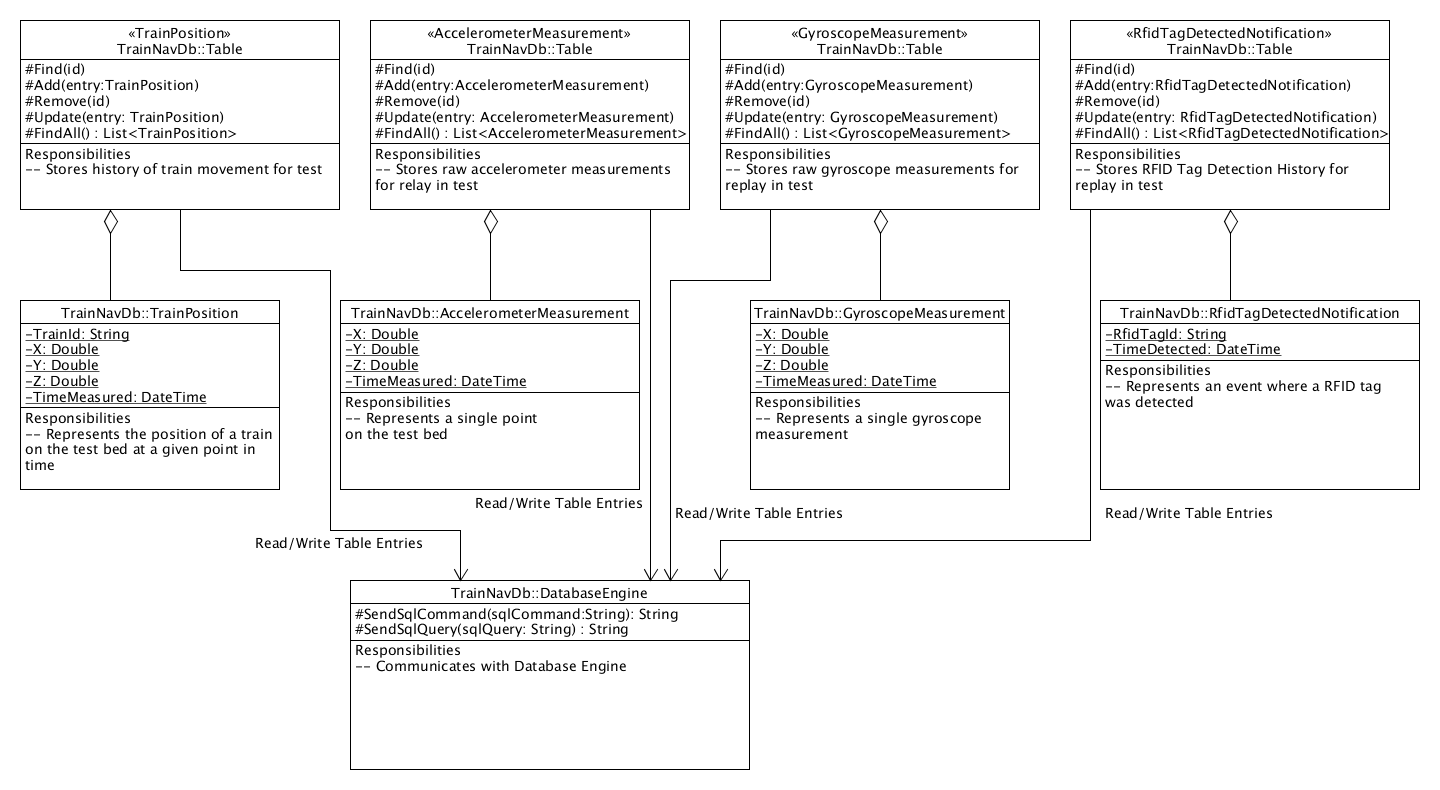


Figure 27 Class Diagram for Train Motion Entities

Figure 26 and Figure 27 show the essential tables that are included in the Train Navigation Database to organize the information that needs to be used by Train Trax. Table classes represent each table hosted inside the Database Engine that is being used by Train Trax. Table Entry classes represent the data in a single row of a given table. The DatabaseEngineAdapter brokers all communication between the Train Navigation Database and the backing database engine.

### Flow

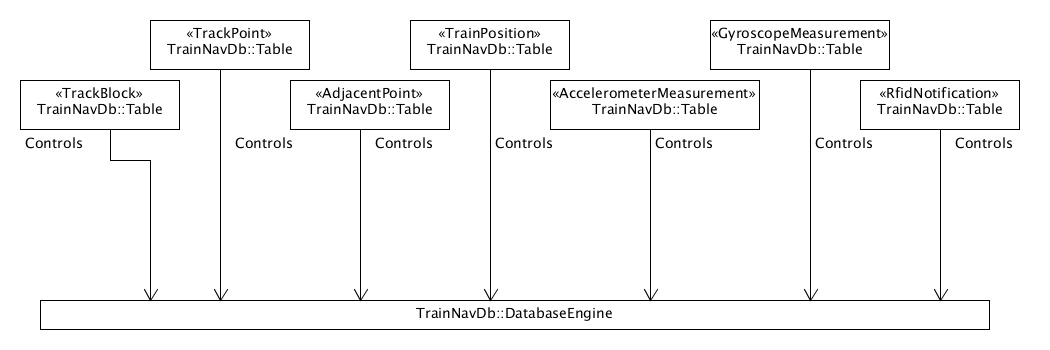


Figure 28 Train Navigation database Control flow

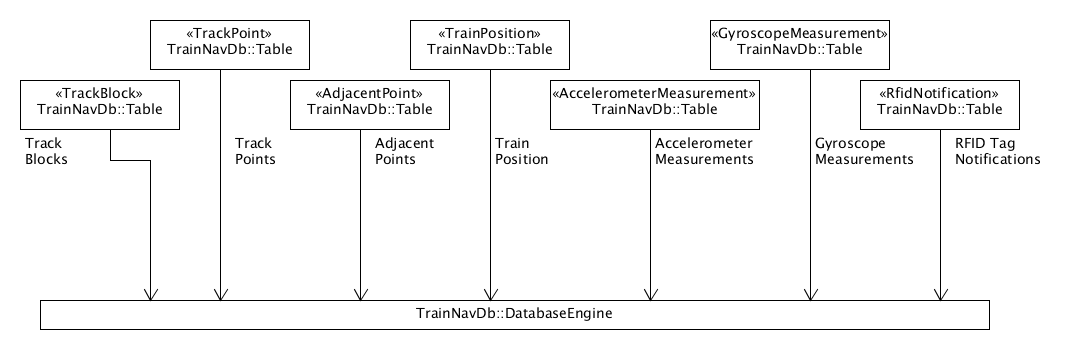


Figure 29 Train Navigation database data flow

### Behavioral

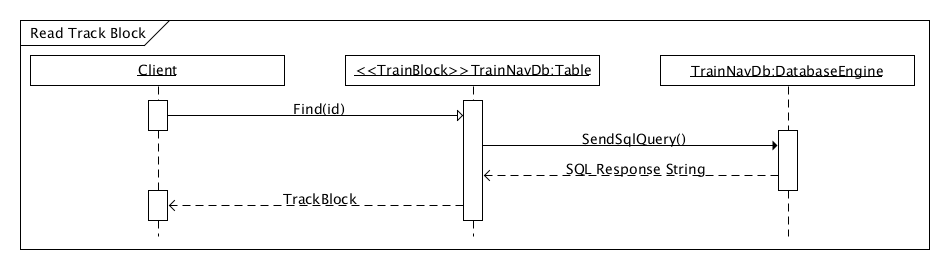


Figure 30 Track Block Search Sequence Diagram

### Track Geometry Measurements

#### Overview

The primary data that is stored in the Train Navigation Database is track geometry information. Track Geometry information is all of the information necessary to describe the shape and size of the track. It also includes a description of all of the relevant points of the track that are essential to making it function, such as identification of track blocks and the location of RFID tags.

All of this information is necessary for the database to have enough information for the Train Navigation Service to correct its estimates of train positions as they cross track markers. Furthermore, it is necessary so that there is enough information for Train Operators to have context of where each train is located relative to both the track and each other.

Since this data is not already available, part of the design is to collect the track geometry data required for the database. The sections below describe details on the type of track geometry information collected as well as the process of collecting it.

Units = inches  
Desired Accuracy of Measurements: +/- 0.1 inches.

#### Coordinate System

Train Trax uses a relative coordinate system based on the distance from a fixed point on the table. In particular, it is the bottom left corner of the table as the origin (0,0). This was selected because if we have this quality of granularity in measurement. We can still choose to relay the location of the train according to a grid if needed to simplify the problem, but we cannot get further accuracy if all that we collected was grid information.

In other words, the location of an object, such as a switch, is the same as the measurement of the width and depth that the center of that object is away from the bottom left corner of the table.

All measurements used to express the coordinate system are in inches. Figure 31 is a clear example of the coordinate system at work. The location of the RFID tag in the figure is at (4, 10). This translates to the RFID tag being 4 inches away from the bottom, left corner width-wise and 10 inches away from the bottom, left corner of the table depth-wise.



Figure 31 Train Trax Coordinate System

#### Calculating Coordinates



Figure 32 Trilateration of Points On Positive train control test bed

As shown in Figure 32, the coordinates of a given point on the table is determined by measuring the distance of the point of interest from three different measurement points. This provides enough information to determine exactly where the point is located using trilateration. The measurement of the distance from the point of interest to a reference point creates a circle of potential points of where the target point could be located. Using a second point, you can reduce the candidates for where the point can be located down to two points: where the two circles intersect. With a third reference point, you can eliminate another point and find the exact location of the point of interest. Circles can be expressed algebraically using the following equation:

x^2 + y^2 = r^2.

The formulas below express using circles to determine the coordinates for the point of interest as described above:

Let (x,y) = Position of Point Desired to Be Triangulated (Point T)

(a1,b1) = Position of Point 1

r1 = Distance of Point 1 from Point T

(a2,b2) = Position of Point 2

r2 = Distance of Point 2 from Point T

(a3,b3) = Position of Point 3

r3 = Distance of Point 3 from Point T

System of Equations:

A: (x – a1)^2 + (y – b1)^2 = r1^2

B: (x – a2)^2 + (y – b2)^2 = r2^2

C: (x – a3)^2 + (y – b3)^2 = r3^2

A: x^2 - 2a1x - 2b1y + y^2 + a1^2 + b1^2 = r1^2

B: x^2 - 2a2x - 2b2y + y^2 + a2^2 + b2^2 = r2^2

C: x^2 - 2a3x - 2b3y + y^2 + a3^2 + b3^2 = r3^2

You can eliminate the x^2 and y^2 components by subtracting the equations from each other:

(A - B) : -2a1x + 2a2x - 2b1y + 2b2y + a1^2 - a2^2 + b1^2 - b2^2 = r1^2 - r2^2

(B – C): -2a2x + 2a3x - 2b2y + 2b3y + a2^2 - a3^2 + b2^2 - b3^2 = r2^2 – r3^2

From here, you have two variables and a pair of linear equations so you solve normally to get x and y.

#### Calculating Orientation



Figure 33 Orientation Calculation From Object Displacement

The orientation of the train at a specific point of interest is calculated by determining the displacement vector from an adjacent train track rung to the point of interest as shown in Figure 33. The reason these points are measured so closely together is to give the closest representation possible of the alignment of the track relative to the left and bottom edges of the table (the y and x axes respectively)   
The rail car must follow along the track therefore the alignment of the track (displacement vector) narrows down the heading to two vectors: the same direction as the displacement vector or the opposite. The vector that is closest to the displacement vector calculated from the two last crossed tag can be used determines which of those two represents the train's current orientation.

Let (x1,y1) be the coordinates of the point on the adjacent rung to the point of interest.

Let (x2, y2) be the coordinates of the point of interest.

The distance between these points is represented by:

d = sqrt((x2 – x1)^2 + (y2 – y1)^2)

(change in x) = dx = (x2 – x1)

(change in y) = dy = (y2 – y1)

angle relative to y axis = arcsin(dx/d)

For example, let (x1, y1) = (1,1) and (x2, y2) = (2,3). Also let the y-axis points North and the x-axis points East.

d = sqrt( (2-1)^2 + (3 – 1)^2) = sqrt (1 + 4) = sqrt(5)

dx = 1

dy = 2

angle relative to y axis = arcsin(1/sqrt(5)) = +/- 26.56 degrees North

Assuming that the train moved from (x1, y1) to (x2, y2), then the orientation of the track is 26.56 degrees North.

#### Calculating Track Layout

In order to render the graph, a Depth First Search(DFS) is performed on the Vertex list of the graph and all of the edges that have not already been drawn, are drawn as the search is performed. Figure 34 gives an example of a graph that can be generated to represent a train track.



Figure 34 Example of a Test Bed Track Graph

Let V be a list of all of the RFID Tag and Orientation Points collected. Each point acts as a vertex of the graph.

Let V1 = V(1), which is the first vertex in the vertices list.

To Draw the edges, DrawEdges(V1) is invoked.

DrawEdges(v) {

Adj = GetAdjacentVertices(v);

for each a in Adj {

if(!a.HasBeenDrawn){

DrawLine(v, a);

DrawEdges(a);

}

}

}

#### RFID Tag Placement

RFID Tags will be placed 6 inches apart from each other so that the distance between position measurements of the track is 6 inches. See the list below for the reasons that 6 inches was chosen to be the spacing between position measurements:

1. We want to make sure that there is at least one point on each Track  
   Block. (Smallest block should be 7 inches)
2. Estimated maximum amount of distance that IMU measurements will be  
   able to be used to determine position within desired accuracy (+/- 3  
   inches = 3 seconds. Assuming traveling at 2 inches per second).
3. We need to ensure that at least two trains can occupy a 14-inch block.

#### Estimated Number of Points to Measure

**# of Points to Measure = # of RFID Tags + # of Switches**

Size of the Table = 8ft x 18 ft = 96ft^2 = 1152 in^2  
Perimeter of Table = 8ft + 8 ft + 18 ft + 18 ft = 52 ft = 624 in  
  
Estimated # of Tags = [Perimeter of Table] / [Distance Between  
Samples] X 2 = ((624 in) / 6) \* 2 = 104 \*2 = **208**  
(We want to make sure that there are enough tags to Cover the Entire  
Track, Plus Additional If Needed For A Higher Sample Rate Where Needed  
And for Troubleshooting)

**# of RFID Tags = Approximately 200.**

**# of Switches = Estimated to be approximately 50 (as reported by the customer).**

#### Types of Data Collected

* Measured
  + Distance from Laser Range Finders
  + Track Block Name that Point Belongs To
  + Adjacent Points
* Calculated
  + Coordinate of the Point
  + Track Orientation at the Point.
  + Graph that represents the tracking
    - Derived from adjacency list formed from Adjacent Points Data

#### Objects to Measure

* RFID Tags
* Switches

#### Track Geometry Data Collection Procedure

Before you start geometry collection, partition the table into 4 sections along the width of the table where the Test Bed is located as shown in Figure 35. Each section will be a phase of where data is collected. This intended to reduce the amount of work necessary for data collection at any one time, and to give us the opportunity to refine the geometry collection procedure before all of the table has been measured. The table is 18 ft wide and 8 ft deep. Therefore, each partition of the table should have an area of 4.5 ft x 8 ft.



Figure 35 Partitioning of Positive Train Control Test Bed Track Geometry Measurement Phases

##### Tools

* Spreadsheet
  + Used to record the measurement data
* Apple iPhone or Alternative Phone that is compatible with Ryobi Phone Works.
* Needed to run the Ryobi Phone Works App
* Ryobi Phone Works App
* Use to read measurements from Laser Range Finder
* Ryobi Phone Works Laser Range Finder
  + Position Measurement
* 3 Camera Tripod Ballheads
  + Placement of Laser Range Finders
  + Ensures that Range Finders are high enough to avoid any interference from objects on the track.
* Fabric Tape Measure
  + Verifies the accuracy of the laser range finder measurements
* 500 Stickers
* Stickers are small colored removable labels.
* Used to mark positions to measure on the track.
* Used as placeholder for RFID tags
* 12 Inch, ½ inch diameter wooden dowel.
* Acts a poll of Rail Car
* Is column that marks position of interest.
* Used by aim laser range finders to measure point of interest.
* 1, ¾ inch Binder Clip
* Secures dowel in upright position in Rail Car
* 1 Mounting Tape
* Secures Binder Clip to Rail Car
* Measurement Rail Car
  + Rail car that has been fitted with a poll.
  + Poll is what laser range finders will target to align and perform measurements.

##### Setup

* Reference Points
  + Prepare a hole to mount each of the tripod ball heads into 3 of the corners of the table.
  + Use the tape measure to measure the coordinates of the center of each of the holes. This should be the position of the center of the tripod ball heads.
  + Measure the width and depth in inches from the bottom, left corner of the table.
  + Mount a tripod ball head into each of the holes.
* Partitions
  + Divide table into 4 Regions Along the Length of the Table
* Stickers
  + Label stickers to unique identify each position a sticker is placed
  + Label each sticker for RFID with a unique number in hex
  + Label each sticker for switch with 'X' plus a hex number
  + Place stickers onto the track to mark switches and where each RFID tag will be placed.
  + Place stickers so that they are centered between both rails at the point of interest.
  + Place a sticker where block U-52;4-16 and block U-55;1-10 meet.
  + Place a sticker spaced every 6 inches starting from that point. Use the fabric table measurer to measure the distance between each sticker.
  + Place a sticker an unlabeled stick on the leftmost adjacent rung to each of the RFID labels.
* Laser Range Finders
  + Use the phone clip of each Ryobi Laser Range Finder to clip to the range finder instead of the phone. This should allow you to mount each Laser Range Finder to a Tripod Ball Head.
  + Mount each Laser Range Finder to a Tripod Ball Head on the table.
* Rail Car
  + Cut a 1 inch / 1 inch square of mounting tape and place it on the top of the test rail car near the center of the car.
  + Place the Binder clip onto the mounting tape inside the are so that mouth of the clip is upright.
  + Insert the down into the binder clip such that it is snug and secure. The dowel shall now serve as an upright poll that will be the focal point for all measurements.

##### Measurement

**Procedure for a Given Partition of the Test Bed**

When searching for an object of interest that has not been measured yet, search from top-to-down and from left-to-right. The x-axis is the edge of the table that is along the width of the table and touches the origin. The y-axis is the edge of the table that is along the depth of the table and touches the origin.

**Measuring the Position of an Object on the Track**

* Locate the sticker that you want to measure.
* Move the measurement rail car into position so that the poll of the car is aligned with the sticker on the track.
* Adjust the laser range finders on each ball head so that the finders can measure the distance from the ball head to the poll of the rail car.
* On the spreadsheet, record all measurements under the name that is on the label. If a sticker does not have a label, then there should be another sticker on an adjacent rung of the track. This is a sticker that represents a secondary point to calculate the orientation of the track. Record it as the name of the adjacent stick plus “V”. For example, if the blank stick is next to label “ 1B”, then Record measurements under the name “1BV”.
* On the spreadsheet, record the measurements of the distance from each range finder for each rail car measurement marker.
* On the spreadsheet, record the name of the track block that the sticker belongs to.
* On the spreadsheet, record the name of all of the stickers that are adjacent to the one measured.

##### RFID Tag Placement

Place a RFID tag for each RFID tag sticker. Make sure that it is centered, length-wise between the two rails so that the middle of the tag is aligned with the middle of the sticker.

### Requirements Traceability

| Requirement Number | Feature | Software Class |  |
| --- | --- | --- | --- |
| Requirement Text | | | |
| TND-1000 | Report Train Position History | TrainPosition Table |  |
| The Train Navigation Database shall save the history of train positions reported by the Navigation Service. | | | |
| TND-1010 | Report Train Position History | TrainPosition Table |  |
| The Train Navigation Database shall save estimates of the position of a given train on a track reported by the Navigation Service. | | | |
| TND-1020 | Report Train Position History | TrainPosition Table |  |
| The Train Navigation Database shall save estimates on the speed of a given train along the track reported by the Navigation Service. | | | |
| TND-2000 | Collect Raw Measurements (Rollup) | AccelerometerMeasurements Table,  GyroscopeMeasurements Table,  RfidTagDetectedNotification Table |  |
| The Train Navigation Database shall save the measurements collected by the Motion Detection Unit. | | | |
| TND-2010 | Collect Raw Measurements | AccelerometerMeasurements Table, |  |
| The Train Navigation Database shall save Motion Detection Unit acceleration measurements. | | | |
| TND-2020 | Collect Raw Measurements | GyroscopeMeasurements Table, |  |
| The Train Navigation Database shall save Motion Detection Unit angular velocity measurements. | | | |
| TND-2030 | Collect Raw Measurements (Rollup) | RfidTagDetectedNotification Table |  |
| The Train Navigation Database shall save Motion Detection Unit notifications about when it crosses a track marker (RFID Tag). | | | |
| TND-2031 | Collect Raw Measurements | RfidTagDetectedNotification Table |  |
| The Train Navigation Database shall save the time that the Motion Detection Unit crosses the train marker. | | | |
| TND-2032 | Collect Raw Measurements | RfidTagDetectedNotification Table |  |
| The Train Navigation Database shall save the unique identifier for the train marker (i.e. RFID Tag ID). | | | |
| TND-3000 | Collect Track Geometry (Rollup) | TrackBlock Table,  TrackPoint Table,  AdjacentPoint Table |  |
| The Train Navigation Database shall save the railway system track geometry (track size, track shape, and connectivity between track blocks) | | | |
| TND-3010 | Collect Track Geometry | TrackPoint Table |  |
| The Train Navigation Database shall save the position of track markers (i.e. RFID Tags) on the test bed. | | | |
| TND-3020 | Collect Track Geometry | TrackBlock Table,  TrackPoint Table |  |
| The Train Navigation Database shall save the track block that a given track marker (i.e. RFID Tag) belongs to. | | | |
| TND-3030 | Collect Track Geometry | TrackPoint Table |  |
| The Train Navigation Database shall save the position of track switches on the test bed. | | | |
| TND-3060 | Collect Track Geometry | TrackPoint Table |  |
| The Train Navigation Database should save the orientation of the track at a given track marker’s (i.e. RFID Tag’s) position. | | | |
| TND-4000 | Collect Track Geometry | AdjacentPoint Table |  |
| The Train Navigation Database shall save which track markers (i.e. RFID tags) that are adjacent to another track marker or switch. | | | |
| TND-4011 | Collect Track Geometry | TrackPoint Table |  |
| The Train Navigation Database shall save the position of objects on the test bed as the distance from a fixed point on the test bed. | | | |
| TND-5000 | Report Train Position | TrainPosition Table |  |
| The Train Navigation Database shall save the unique identifier associated with each train that belongs to the Positive Train Control Test Bed. | | | |

## Software Program: Train Monitor Terminal GUI

### GUI Overview

The Train Trax system GUI is a user-friendly program for controlling the track and monitoring trains. It will consist of a main menu that includes submenus to enter information. On the main menu the user will be able to select an option to Monitor the train. The Monitor Train window will display a diagram of the track with an icon of the train’s position and speed, and an icon of all switches on the track. From this window the user also has the option to control track switch changes, and an option to display a train’s position and speed history in database format. If the Train History option is selected from the Train Monitor window, the train’s position and speed information over a specified amount time will displayed in a excel like table format. If the Control Track Switch changes are selected, the state of the switch will be changed and the Monitor Train window updated to reflect the latest state.

The Model-View-ViewModel design pattern is being used to design the GUI where the model contains the business logic, the view deal with presenting and retrieving information from the user, and the viewmodel acts as a bridge for interactions between the view and the models.

### Structural

#### Views

##### Main Menu View

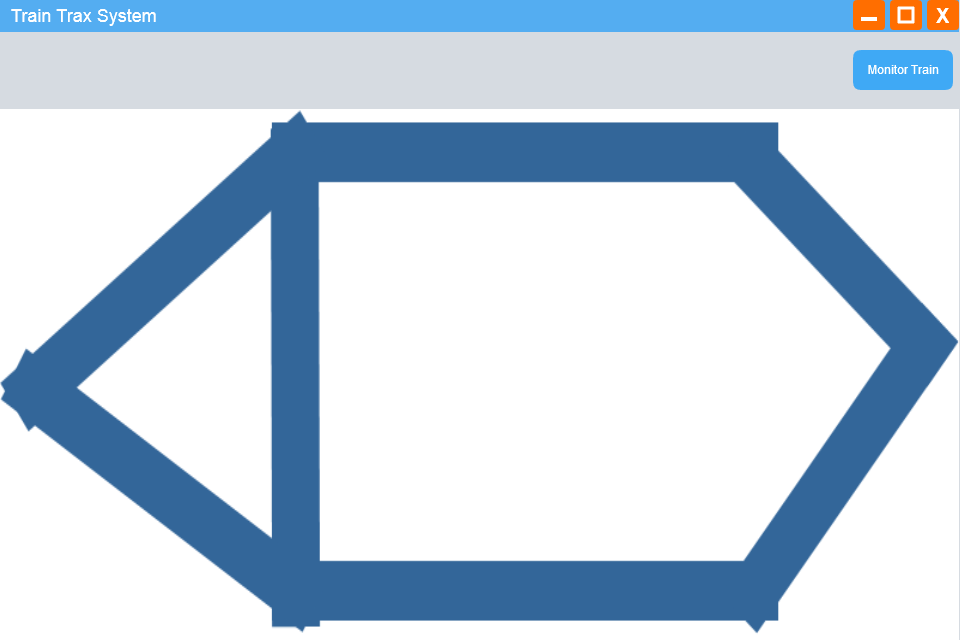


Figure 36 TRAX UI Main Menu View

|  |
| --- |
| **MainMenuView** |
| - trainMonitorButton : boolean = false  - trackLayout : boolean = false |
| + displaytrainMonitorView()  + discovertrackLayoutView()  + startTrax() |

Table 4 Class Diagram for Main Menu View

|  |  |
| --- | --- |
| **Detailed Design for MainMenuView** | |
| **Attributes** | trainMonitorButton = false |
| **Operations** | displaytrainMonitorView()  begin;  if(trainMonitorButton)  {  Create New Window;  GetTrackLayout from database;  Draw Diagram of Track;  Add state and location of switches to track;  Compute location of train;  Add Train location icon to display its current position;  Display speed of train;  }  End; |
|  | startTrax()  begin;  if (tracklayout)  {  Load position data of track diagram from database  Draw coordinates of track diagram on screen  }  else(!tracklayout)  {  //no track geometry loaded need to start in discover track view  Display error message saying no geometry data has been loaded  Render blank screen  }  End; |

Table 5 Detailed Design for Main Menu View

##### Train Monitor Window View

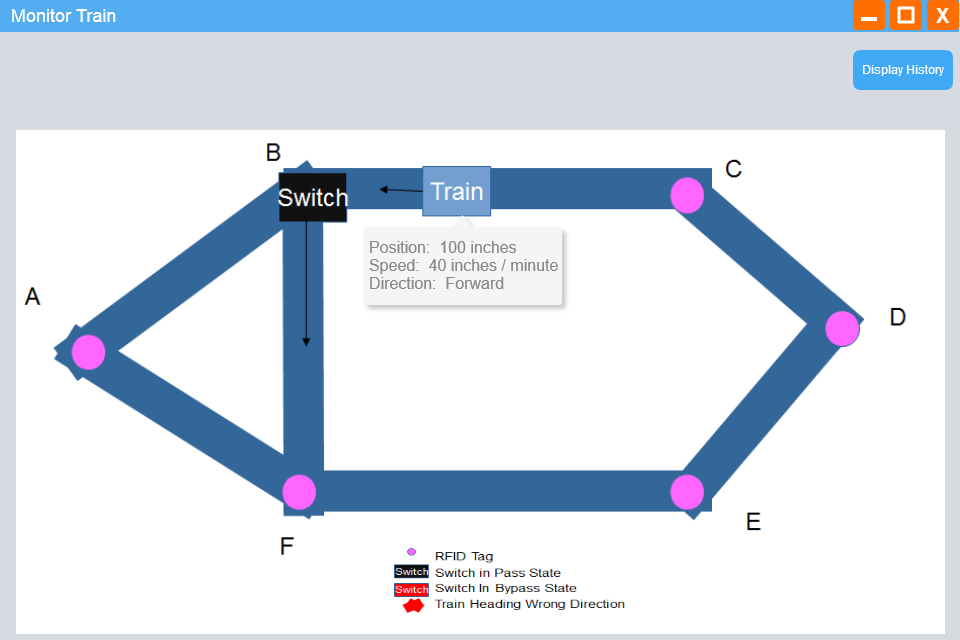


Figure 37 TRAX UI Main Train Monitor View

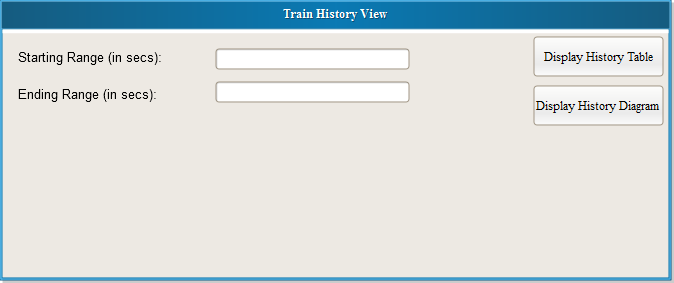
|  |
| --- |
| **TrainMonitorView** |
| - trainHistoryButton : boolean = false  - numswitches : int= 0  - switchstate ; boolean = false |
| + displaytrainHistoryView()  + changeSwitchState() |

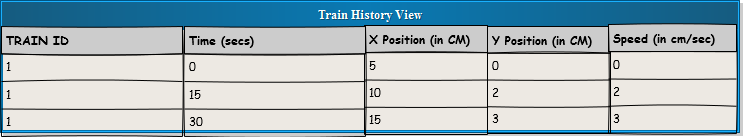
Table 6 Class Diagram for Train Monitor View

|  |  |
| --- | --- |
| **Detailed Design for TrainMonitorView** | |
| **Attributes** | trainHistoryButton = false, switchPressed = false, numswitches = 0, switchstate=false |
|  | displaytrainHistoryView ()  begin;  if (trainHistoryButton)  {  Get Train ID from Train Navigation System;  Create New TrainHistoryView Window with options for displaying the train position history in a table and in a diagram form;  }  End; |
|  | changeSwitchState()  if(switchPressed)  {  begin;  Prompt user with dialog box to verify they want a switch change  If (no) exit;  For numswitches  Get Current Value of Switch (switchstate)  If True  Set Switchstate to false;  Else  Set Switchstate to true;  If Switchstate true  Switchicon is highlighted;  }  End; |

Table 7 Detailed Design for Train Monitor View

##### Display Train History View





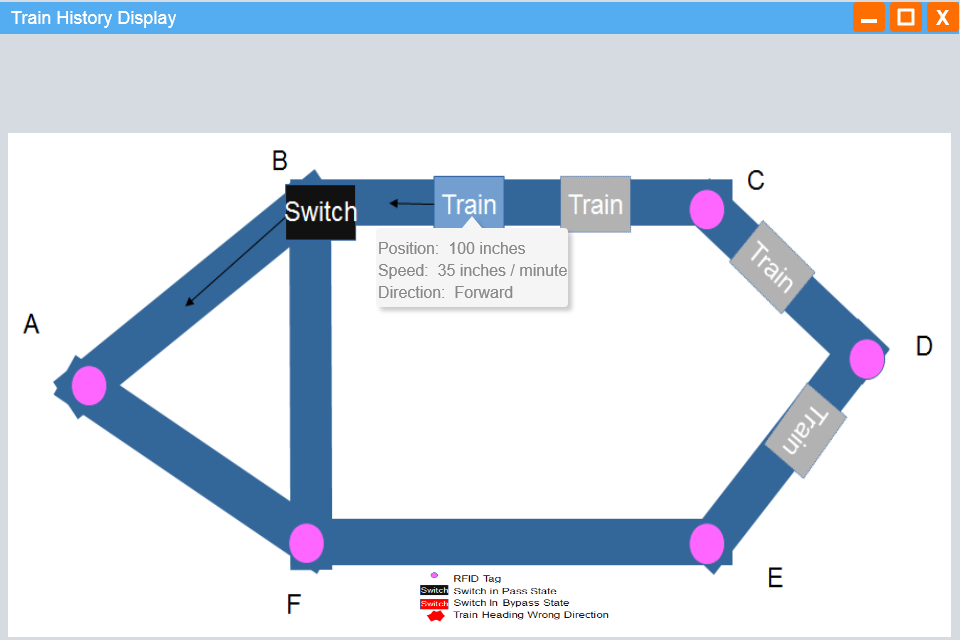


Figure 38 TRAX UI Display Train History Views

|  |
| --- |
| **TrainHistoryView** |
| - trainIdList  - trainPositionTable  - timeRangeControl  - trackControl |
| + showTrainPositionTable(trainId, dateRange)  +showPathTrainTraveled(trainId,dateRange)  + displayHistory(trainId, dateRange) |

Table 8 Class Diagram for Train History View

|  |  |
| --- | --- |
| **Detailed Design for TrainHistoryView** | |
| **Attributes** | trainIdList, trainPositionTable, timeRangeControl, trackControl,  trainPositionButton= false, trainTableButton = false; |
|  | showTrainPositionTable (trainId, dateRange)  begin;  if (trainPositionButton)  {  Lookup trainID with record in the database;  Time = starttime;  While (time < endtime)  {  Get train position coordinates at time  Display Train ID, Time, Coordinate and Speed in Position Table;  Time += interval;  }  End; |
|  | showPathTrainTraveled ()  begin;  if (trainPositionButton)  {  Lookup trainID with record in the database;  Time = starttime;  While (time < endtime)  {  Get train position coordinates at time  Time += interval;  }  Draw Coordinates on display;  }  End; |
|  | displaytrainHistory(trainID, dateRange)  begin  call showTrainPositionTable  call showPathTrainTraveled |

Table 9 Detailed Design for Train History View

#### ViewModels

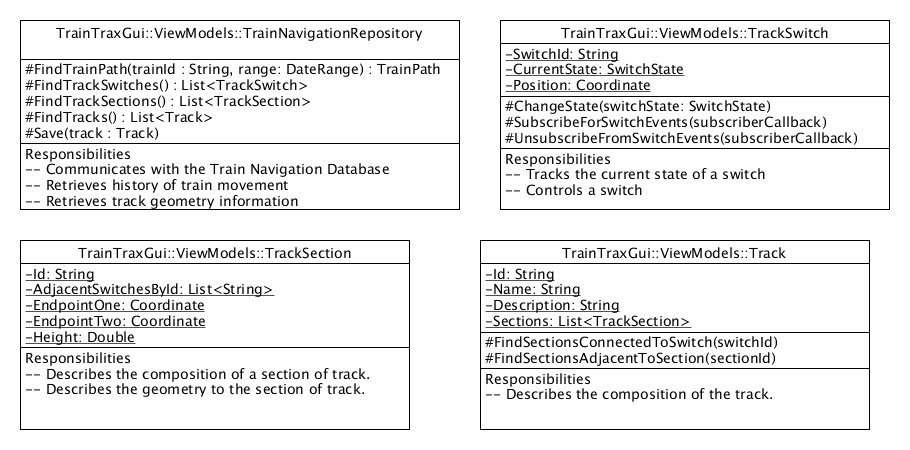


Figure 39 Train Monitor Terminal GUI Entity State Diagram for ViewModels

Figure 39 shows the critical viewmodels used by the GUI to bridge models to the views in the UI that use them. The TrackSwitch class represents a single switch on the Positive Train Control Test Bed. It is responsible for reporting to users the current state of the switch. It also processes requests from users to change the state of the switch. The Train class represents all of the information known about a given train on the Positive Train Control Test Bed. It is responsible for reporting information, such as its current position and speed. The TrackBlock class represents a single section of track used to construct the Positive Train Control Test Bed. It includes information such as an identifier for the block, other blocks that it is connected to, and switches and track markers that are on it. The Track class represents all of the information known about the complete track of the Positive Train Control Test Bed. The TrainNavigationRepository is a datastore for all of the information that pertains to tracking trains around the Positive Train Control Test Bed, and about the track of the test bed itself. This is the primary object that interacts with any database used by Train Trax.

#### Models

The models used for the GUI are the public classes made available from the Train Navigation Database and the Train Navigation Service.

### Flow

The primary flow is where data is retrieved from the model to the viewmodel and the view model reports the information to the view which then updates its controls to display the information to the user.

If the user makes any kind of command, such as to control a switch, then the change to a control on a view invokes a command on the viewmodel which intern results in invoking a method on the target model(s) responsible for enacting the requested change. Feedback from the command is handled like any other output from the model.

### Behavioral

Behavior is described from the view-level and is shown in the pseudocode in section 3.5.2.1. In the sections below, the interactions between the views, viewmodels, and models are demonstrated.

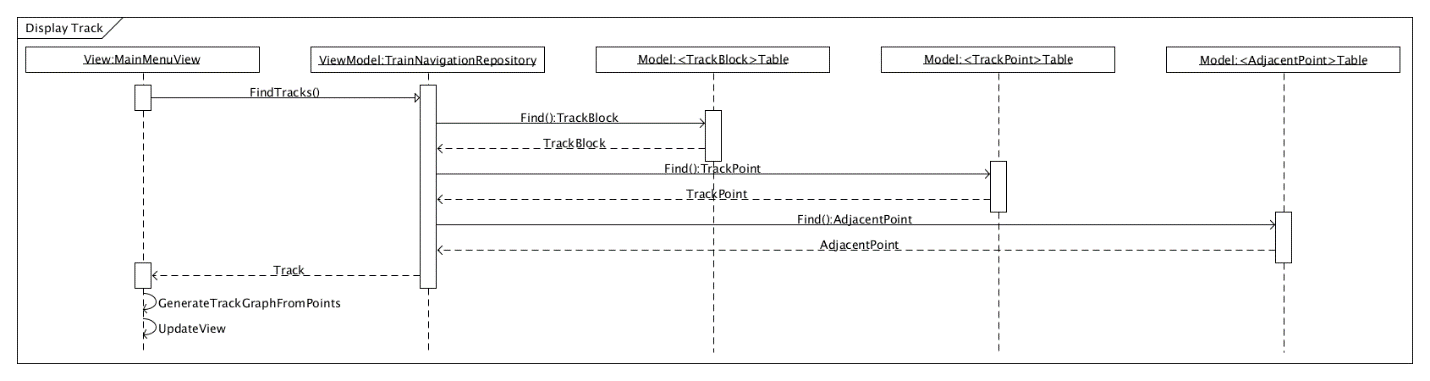


Figure 40 GUI Display of Test Bed Track Sequence Diagram

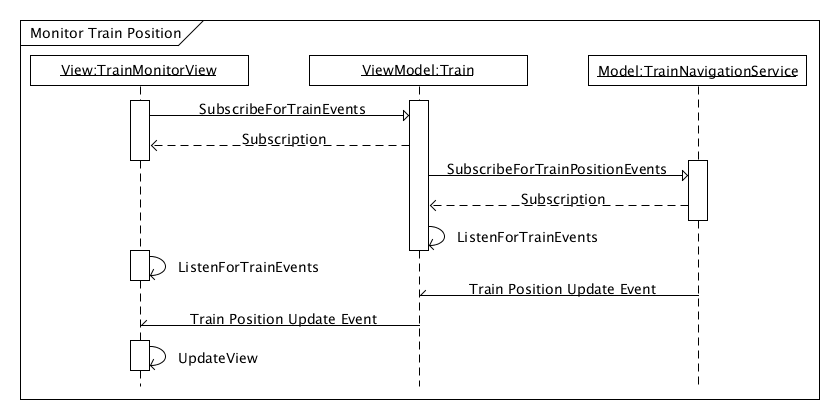


Figure 41 GUI Update Train Position Sequence Diagram

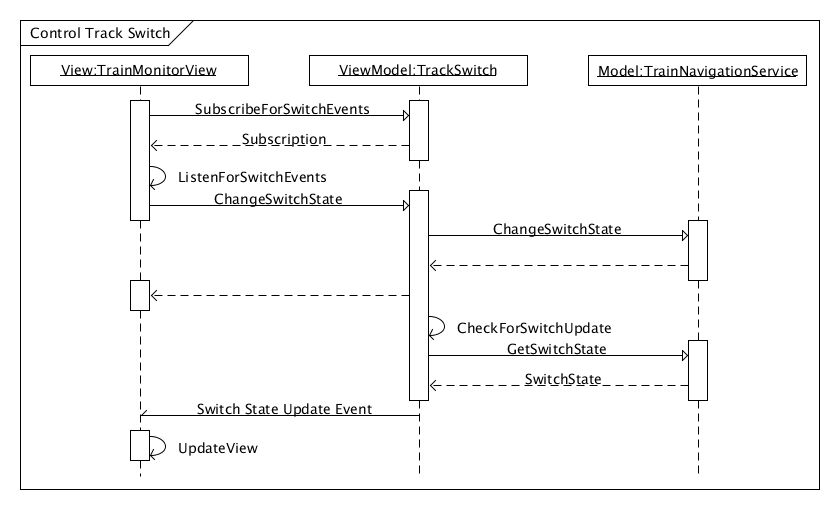


Figure 42 GUI Control of a Track Switch Sequence Diagram

### Requirements Traceability

| Requirement Number | Feature | Software Class |  |
| --- | --- | --- | --- |
| Requirement Text | | | |
| GUI-1000 | Report Train Position | TrainMonitorView |  |
| The Train System GUI shall display to users the last reported position of a given train on the train track by the Train Navigation Service. | | | |
| GUI-2000 | Report Train Position, Control Track | MainMenuView,  TrainMonitorView |  |
| The Train System GUI shall display the Positive Train Control Test Bed track. | | | |
| GUI-2020 | Report Train Position, Control Track | TrainMonitorView |  |
| The Train System GUI should display the position of track markers from the Train Navigation Database. | | | |
| GUI-2030 | Report Train Position, Control Track | MainMenuView,  TrainMonitorView |  |
| The Train System GUI shall display track switches on the test bed from the Train Navigation Database. | | | |
| GUI-3000 | Report Train Position | TrainMonitorView |  |
| The Train System GUI shall display the speed of trains on track as last reported by the Train Navigation Service. | | | |
| GUI-4000 | Control Track | TrainMonitorView |  |
| The Train System GUI shall allow users to control track switches. | | | |
| GUI-5000 | Report Train Position History (Rollup) | TrainHistoryView |  |
| The Train System GUI shall display the history of movement collected for a given train. | | | |
| GUI-5010 | Report Train Position History | TrainHistoryView |  |
| The Train System GUI shall display estimates of train positions from the Train Navigation Database. | | | |
| GUI-6000 | Control Track (Rollup) | TrainMonitorView |  |
| The Train System GUI shall display to the user the current state of switches on the rail system. | | | |
| GUI-6010 | Control Track | TrainMonitorView |  |
| The Train System GUI shall send requests to the Train Navigation Service to change the state of switches on the rail system. | | | |
| GUI-7000 | Alert When Train Stopped | TrainMonitorView |  |
| The Train System GUI should alert when the train stops. | | | |
| GUI-8000 | Alert When Train Reverses Direction | TrainMonitorView |  |
| The Train System GUI should alert when the train reverses direction. | | | |
| GUI-9000 | Train Position Prediction | TrainMonitorView |  |
| The Train System GUI should indicate the direction that a given train will go when it crosses the next switch in its path. | | | |

# **Reuse and relationships to other products**

Motion Detection Unit Firmware

* Arduino 1.0 Library
* FreeIMU Software library

Train Navigation GUI

* Java Platform Standard Edition 7
  + Provides UI Primitives used to build GUI.
  + Communication with the Train Navigation Service

Train Navigation Service

* Java Platform Standard Edition 7
  + Communication with Motion Detection Unit
  + Support Train Position Calculations
* JMRI Library
  + Contract with Train Command Station

Train Navigation Database

* Java Platform Standard Edition 7
  + Support Classes used to access database.
* MySql
  + Database Engine used
* Standard MySql JDBC Driver
  + Intermediary used to contact Database Engine

# **Design decisions and tradeoffs**

## Database Engine Selection for Train Navigation Database

Decision: Database Selection

Description:

The Train Navigation Database requires a database service to handle saving and reading information.

Selection Criteria:

Longevity: Database must be believed to be able to be supported at least 2-3 years after the project.

Support: Reliable vehicles for the team to obtain information for questions that arise while implementing and supporting Train Trax

Ease of use: How easy it is for a new developer to understand how to setup and use the database and the software that interface with it.

Scale: (1(Poor) – 5 (Great))

Alternatives:

Alernative 1: MySql

Longevity: 5. Active community. Used in many commercial applications

Support: 5. Well documentation for using database and driver. Commercial options also available to get support.

Ease of use: 4. Setup is needed to configure the database before being able to use.

Alternative 2: SQLite

Longevity: 5. Active community. Used in many commercial applications

Support: 3. There is a lot of documentation available for setup and using the C interface and the is an active community for that. No activity communities were found for any of the Java Drivers to the database.

Ease of use: 3. In order to use it from java, multiple java drivers have to be evaluated because of the spectrum of supported version advertised. Otherwise, a JNI interface would need to be developed to ensure that we have the support we need.

Outcome:

MySql was selected because it has the best chance of being supported long term and the fewest steps necessary to implement a working solution.

# **Pseudo code for components**

## Object Position Estimation Algorithm

Assumptions:

This assumes that before this algorithm is used, the target object is at rest and is at a known position. In practice, this means that the front of the test rail car must be position on top of a RFID tag.

Let dtr = the distance from the front of the train to the front of the rail car.

Let p0 = the initial position of the rail car.

Let v0 = the initial velocity of the rail car = 0.

Let a0 = the initial acceleration of the rail car = 0.

Let or0 = the initial orientation of the rail car.

CalculateTrainPosition(dtr, p0, v0, a0, or0){

a\_list = GetAccelerometerMeasurements();

g\_list = GetGyroscopeMeasurements();

g\_list = InterpolateValues(g\_list, a\_list.times);

a\_list = InterpolateValues(a\_list, g\_list.times);

orientation\_list = CalculateOrientationsFromGyroscope(g\_list, or0);

a\_list = ConvertFromBodyFrameToNorthEastDownInertialFrame(a\_list, orientation\_list);

v\_list = CalculateVelocitiesFromAccelerometer(a\_list, v0, a0);

p\_list = CalculationPositionsFromVelocities(v\_list, p0);

af = a\_list[end];

vf = v\_list[end];

pf = p\_list[end];

train\_position = pf + dtr;

return train\_position, pf, vf, af

}

RfidTagReceived(lastPosition, RfIdtagId){

p0=lookupPosition(tagId);

pOr = lookupOrientationPointPosition(tagId);

if(distanceBetween(lastPosition, p0) > distanceBetween(lastPosition, pOr)){

or0 = calculateOrientation(pOr, p0);

}

Else{

or0 = calculateOrientation(p0, pOr);

}

//Resetting Calculations since we have a new known position.

ClearAccelerometerMeasurements();

ClearGyroscopeMeasurements();

Return p0, or0;

}

CalculateVelocitiesFromAccelerometer(A, t, v0) {

t\_prev = t(1); // First measurement in time.

for T in t{

dt = ( T – t\_prev);

V(T) = A(T)\*dt + v0;

t\_prev = T;

}

return V;

}

CalculatePositionFromVelocities (V, t, r0) {

t\_prev = t(1); // First measurement in time.

for T in t{

dt = ( T – t\_prev);

R(T) = V(T)\*dt + r0;

t\_prev = T;

}

return V;

}

CalculateOrientationFromGyroscope(G, or0) {

t=G.times;

t\_prev = t(1); // First measurement in time.

for T in t{

dt = ( T – t\_prev);

OR(T) = G(T)\*dt;

OR(T) = convertAnglesToNedInertialFrame(OR(T)) + or0

t\_prev = T;

}

return OR;

}

# References

*IMUs and INS*. (2015, 11 23). Retrieved from VectorNav: http://www.vectornav.com/support/library/imu-and-ins

Society, I. C. (2009, July 20). IEEE Standard 1016: Software Design Description. New York, New York, USA. Retrieved from IEEE Xplorer.

*Triangulate Your Position*. (2004, June 4). Retrieved from Everything 2: http://everything2.com/title/Triangulate

*Understanding Euler Angles*. (2015, November 23). Retrieved from CH Robotics: http://www.chrobotics.com/library/understanding-euler-angles