# An Analysis of Jolts Experienced on Amtrak Railways

# Trips we took

A.k.a public transport where we scared unsuspecting commuters with our devices



# Specific Railways Studied

1. Illinois Service Line between Chicago & Champaign
    - Multiple trips with three or more DAQ devices present
    - Primary route; most detailed analysis

2. Italian Intercity Trains

    - For comparison with American railways

3. Hiawatha Line between Chicago & Milwaukee, Wi.

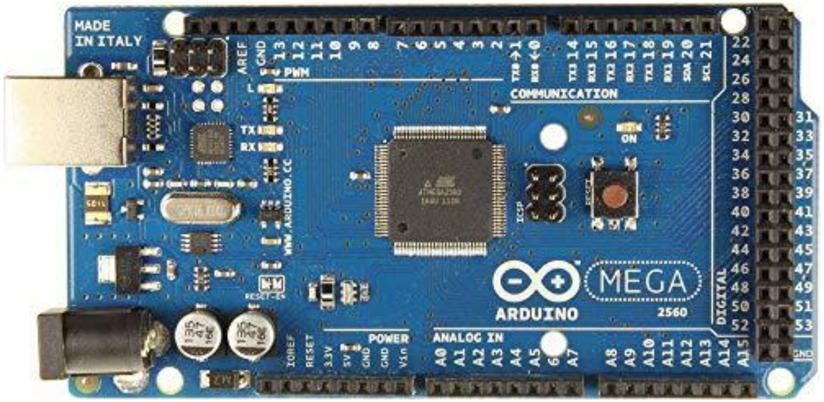
# The State of Railways in the United States

- **<u>Amtrak:</u>** 2,142 railway cars and 425 locomotives
- 21,400 miles within the contiguous United States & Southern Canada.
- Recent years: reputation for being rougher, less efficient and slower than international counterparts (i.e. European, East Asian railways).
- Conditions of rails: below what American Society of Civil Engineers considers optimal.
- Current annual budget just enough to keep rails *safe*

Logo Courtesy of National Railroad Passenger Corporation

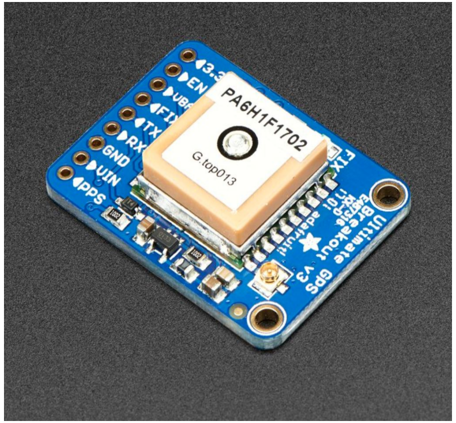
# THE ARDUINO

- Arduino MEGA 2560
  - Features 256 kilobytes of volatile flash memory
  - 8 kilobytes of RAM
- Built in 16 channel 10 bit Analog-to-Digital Converter (ADC)
- Hardware Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit communication (I2C) protocol



# Ultimate GPS

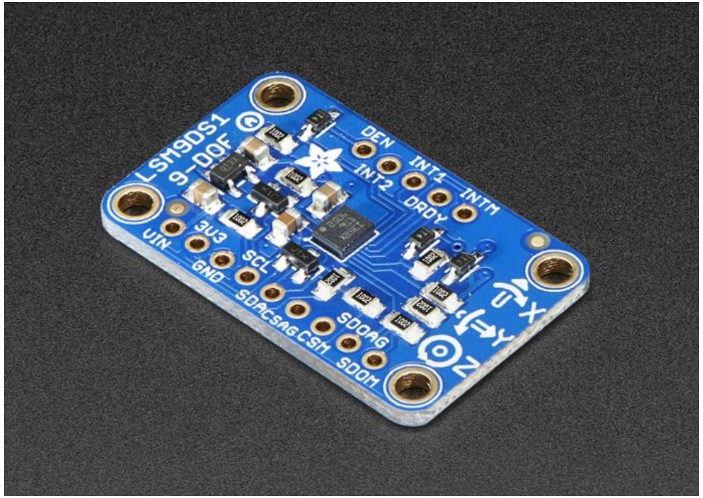
- Can 22 satellites
- -165 dBm sensitivity, 10 Hz updates, 66 channels
- 5V friendly design and only 20mA current draw
- Breadboard friendly + two mounting holes
- Built-in datalogging
- PPS output on fix
- Internal patch antenna + u.FL connector for external active antenna




# LSM9DS1

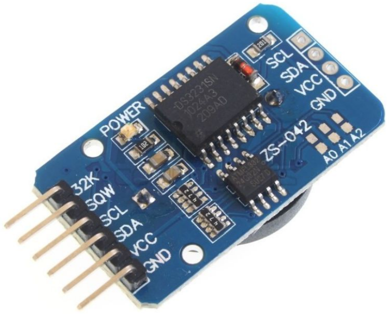
9 DOF sensor:

- 3-axis accelerometer
  - Measures gravity
  - Informs the user how fast the board is accelerating in 3D space
- 3-axis magnetometer
  - Detects which direction the magnetic north lies
    - Done by sensing where the strongest magnetic force is coming from
- 3-axis gyroscope
  - Uses Earth's gravity to measure spin and twist
  - Ultimately help determine orientation.



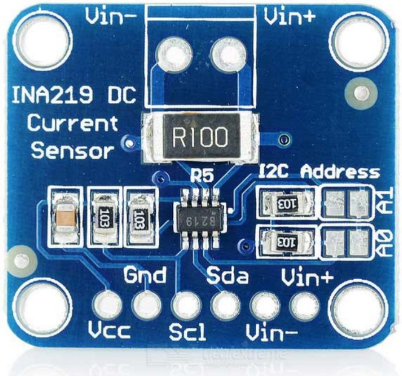
# DS3231 RTC

- Precision clock
- Coin cell on the back of the sensor
  - The user to take years of data even if the main power is lost.
- Synchronizes time read outs
  - Matches GPS data with position data
- Contains an extremely accurate internal crystal
  - Accounts for drifting caused by external crystals

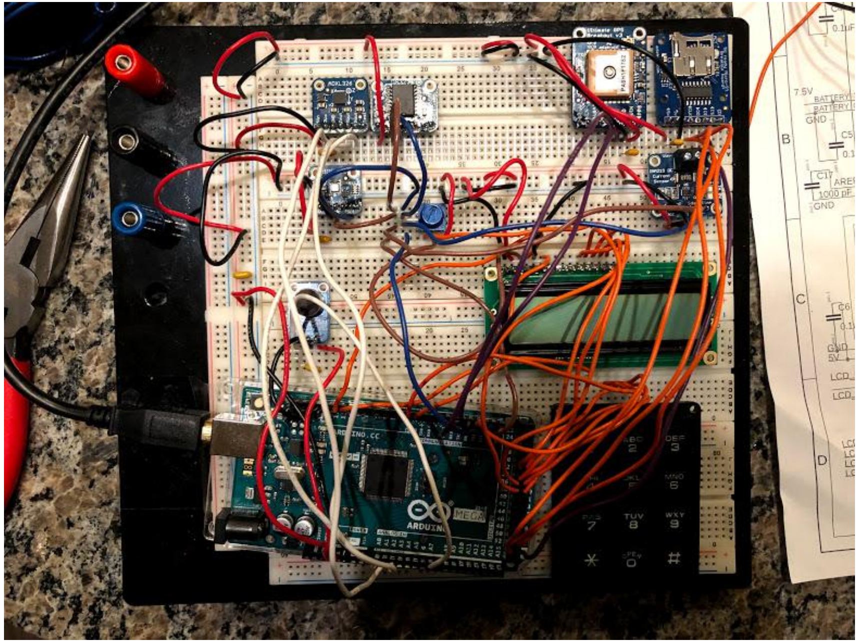


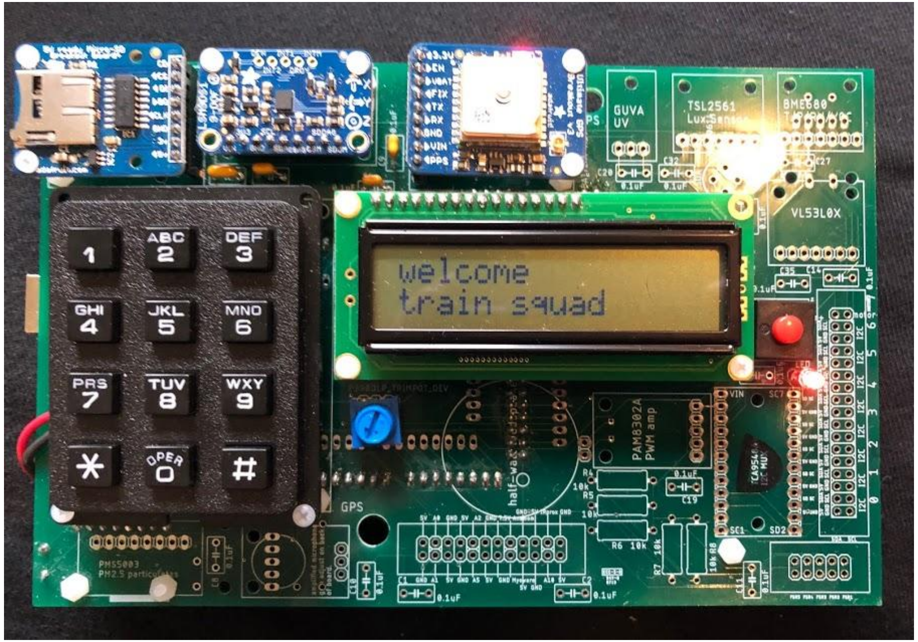

# INA219

- A high side DC current sensor
- A precision amplifier that measures up to ±3.2 Amps
- Used for power-monitoring related problems
- Uses I2C to measure both the high side voltage and DC current draw
  - 1% precision



# Breadboarded & PCB Versions of DAQ





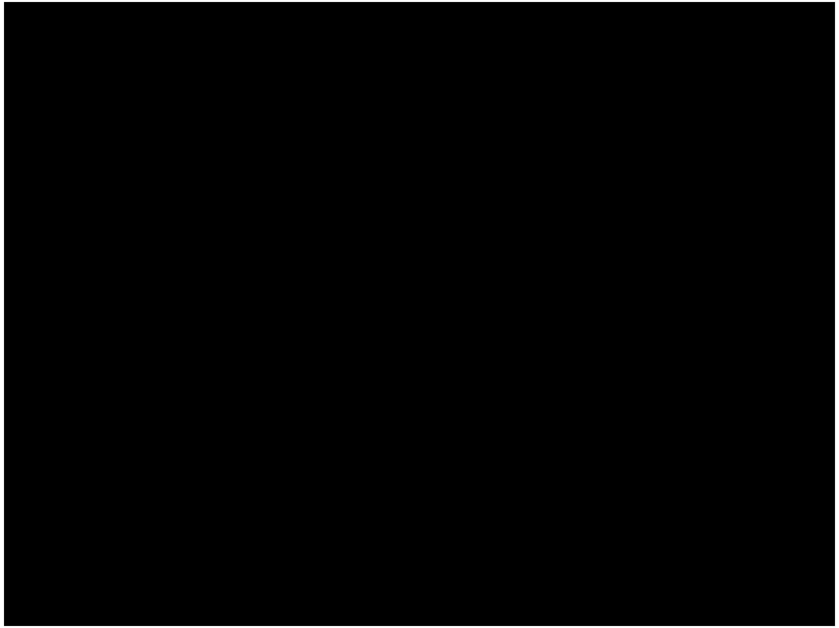

writing to
data001.txt

# Software


- Arduino Data Acquisition Code
- Python Processing
  - Repairing Data
  - Calculating Data
  - Averaging Data

# Arduino Data Acquisition

- Automatically updating filenames
- Using Keypad for multiple functions
  - Electrical information
  - File Closing
  - Current Filename
  - The Magic Conch --->
- GPS function
  - And subsequent issues
- Writing to file
  - Inserting NaNs when appropriate



# The ~Python Sequence~

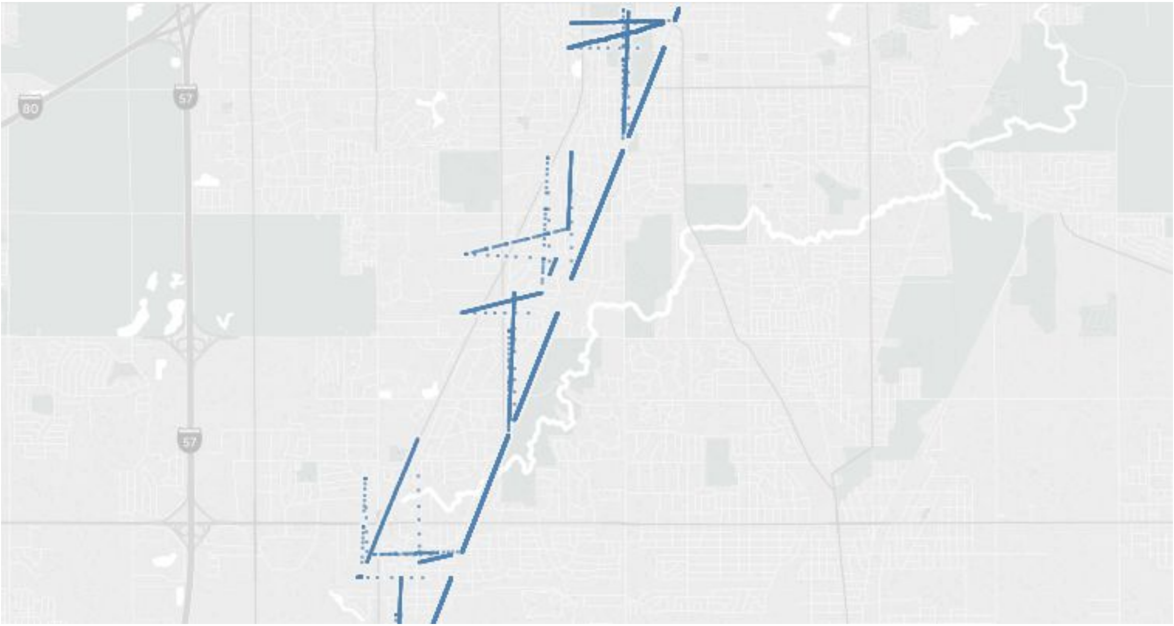
| Processing Phase | Operation(s) | Filename |
|---|---|---|
| I | Repair GPS data | `GPS_repair.py` |
| II | Organize data into arrays, perform minor calculations, & prepare for mapping in Tableau | `TrainPy3.py` |
| III | Calculations of time-averaged acceleration & jerk for use in jolt analysis | `GPS_calculations.py` |

Table 4: Brief overview of the processing pipeline through which all data collected during both test-runs and Amtrak rides passes. Note: the filename of step II includes an integer corresponding to the latest version of the program.

# GPS -troubles-

Error in GPS Parsing Conversions caused this pattern to appear in our maps.

This is why we needed a repair script.



# Repairing

- Our GPS gives out values in DDMM.MMMM, so the minutes are in decimals.
- Issue was stripping zeros from minute string when converting from string to integer. (ex: 4000.0063 would show up in data as 4000.63)
- Split string at period, and count digits that remain in the second one
- If less than 4, add appropriate number of zeros to string
- Write to output file


**Us, when we saw the first map that didn't look like a glitch in the simulation**

# Arrays & Calculations

- Intially read in data with pandas, also used numpy
- Had to convert DDMM.MMMM values to only degrees (1)
- Located indices where GPS had values.
    - Used this to apply a mask to other arrays to visualize raw data and interpolate (2)
- Interpolate GPS coordinates using original and masked millis arrays (2)
- Multiplied longitude by -1
- Subtract 1 from z acceleration array
- Output to CSV, ready to plot and analyze

1

```
def degrees(coordinate):

    coordinate_mod = coordinate%100
    coordinate_floor = coordinate//100

    coordinate_degrees = (coordinate_mod/60) + coordinate_floor

    return(coordinate_degrees)

longitude = degrees(longitude_raw)
latitude = degrees(latitude_raw)
```

2

```
lat_index = np.array([index for index,value \
                      in enumerate(latitude_raw) if np.isnan(value) == False])
millis_lat = millis[lat_index]
lat_int = np.interp(millis, millis_lat, latitude_stripped)
```

# Final Averaging and Output

- Large amount of small-timestep acceleration points didn't show us the bigger picture
- Decided to average over a quarter second to get better idea of rough patches in track (1)
- Inserted "jerk" calculation
    - Difference of subsequent acceleration averages (2)
- Output all data to final CSV, ready for plotting and analyzation of bigger picture

```
if(len(row) > lat_index + 2 and line_index > 1):

    # pick up starting time (in ms) from first non-heading line
    #if(line_index == 2):
        #tstart = int(row[9])

    t_now = float(row[18])
    ax_now = float(row[4])
    ay_now = float(row[5])
    az_now = float(row[6])

    ax_sum = ax_sum + ax_now
    ay_sum = ay_sum + ay_now
    az_sum = az_sum + az_now
    ax_number = ax_number + 1

    lat_now = float(row[lat_index])
    long_now = float(row[long_index])

    lat_sum += lat_now
    long_sum += long_now
    coord_number += 1


    # time to calculae an average?
    if(t_now - t_last_average >= t_bin_width):

        ax_average = float(ax_sum) / ax_number
        ay_average = float(ay_sum) / ax_number
        az_average = float(az_sum) / ax_number
        axn = float(ax_number)
        h_ax.hfill(ax_average)
        h_ay.hfill(ay_average)
        h_az.hfill(az_average)
        h_num_ax.hfill(axn)


        ax_arr[i] = ax_average
        ay_arr[i] = ay_average
        az_arr[i] = az_average


        lat_average = float(lat_sum) / coord_number
        long_average = float(long_sum) / coord_number



        lat_arr[i] = lat_average
        long_arr[i] = long_average
```

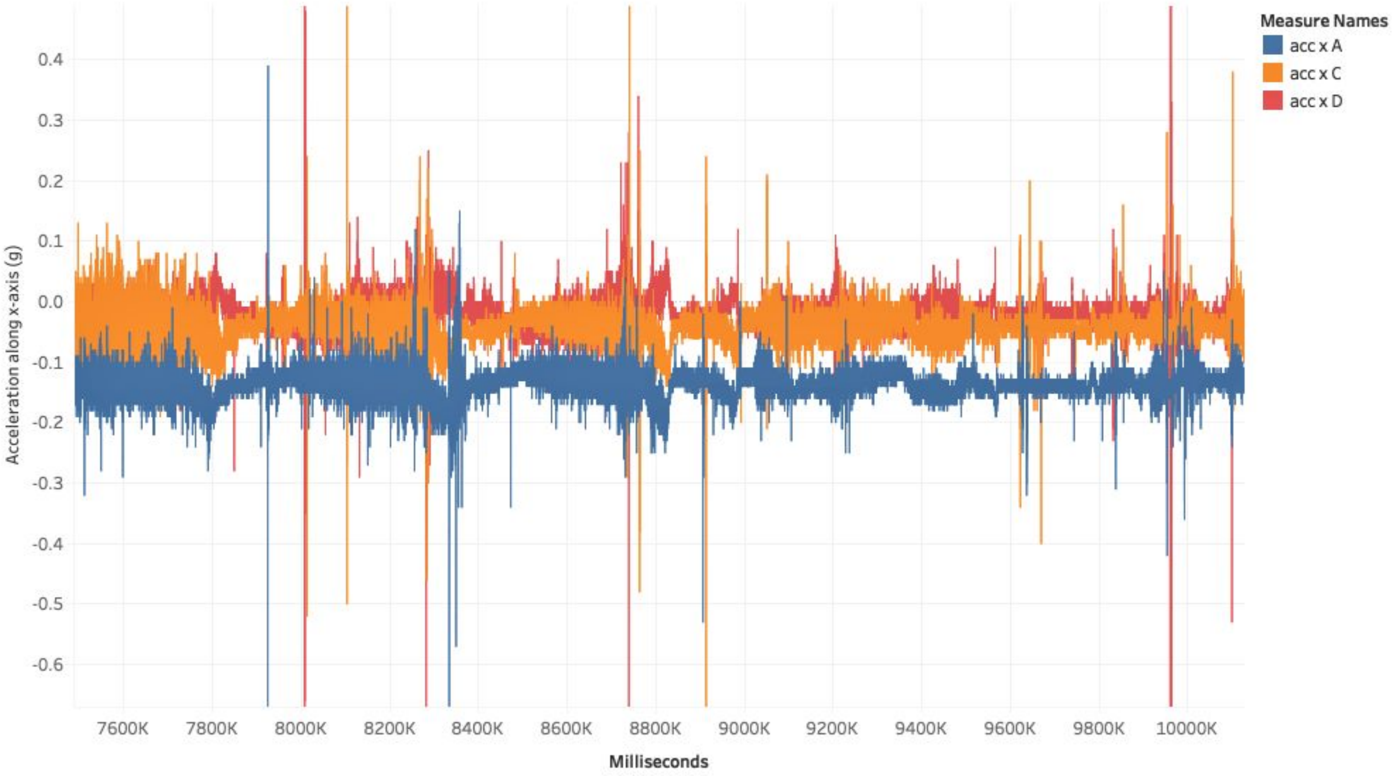
1

```
        # can we calculate "jerk" now?
        if(ax_number >= 2):
            xjerk = (ax_average - last_ax_average) / t_bin_width
            yjerk = (ay_average - last_ay_average) / t_bin_width
            zjerk = (az_average - last_az_average) / t_bin_width

            last_ax_average = ax_average
            last_ay_average = ay_average
            last_az_average = az_average

            h_xjerk.hfill(xjerk)
            h_yjerk.hfill(yjerk)
            h_zjerk.hfill(zjerk)


            jx_arr[i] = xjerk
            jy_arr[i] = yjerk
            jz_arr[i] = zjerk

        else:
            jx_arr[i] = 'NaN'
            jy_arr[i] = 'NaN'
            jz_arr[i] = 'NaN'

        t_arr[i] = t_now

        t_last_average = t_now
        ax_number = 0
        ax_sum = 0
        ay_sum = 0
        az_sum = 0

        coord_number = 0
        lat_sum = 0
        long_sum = 0

        i += 1
```

2

# DAQ Synchronization

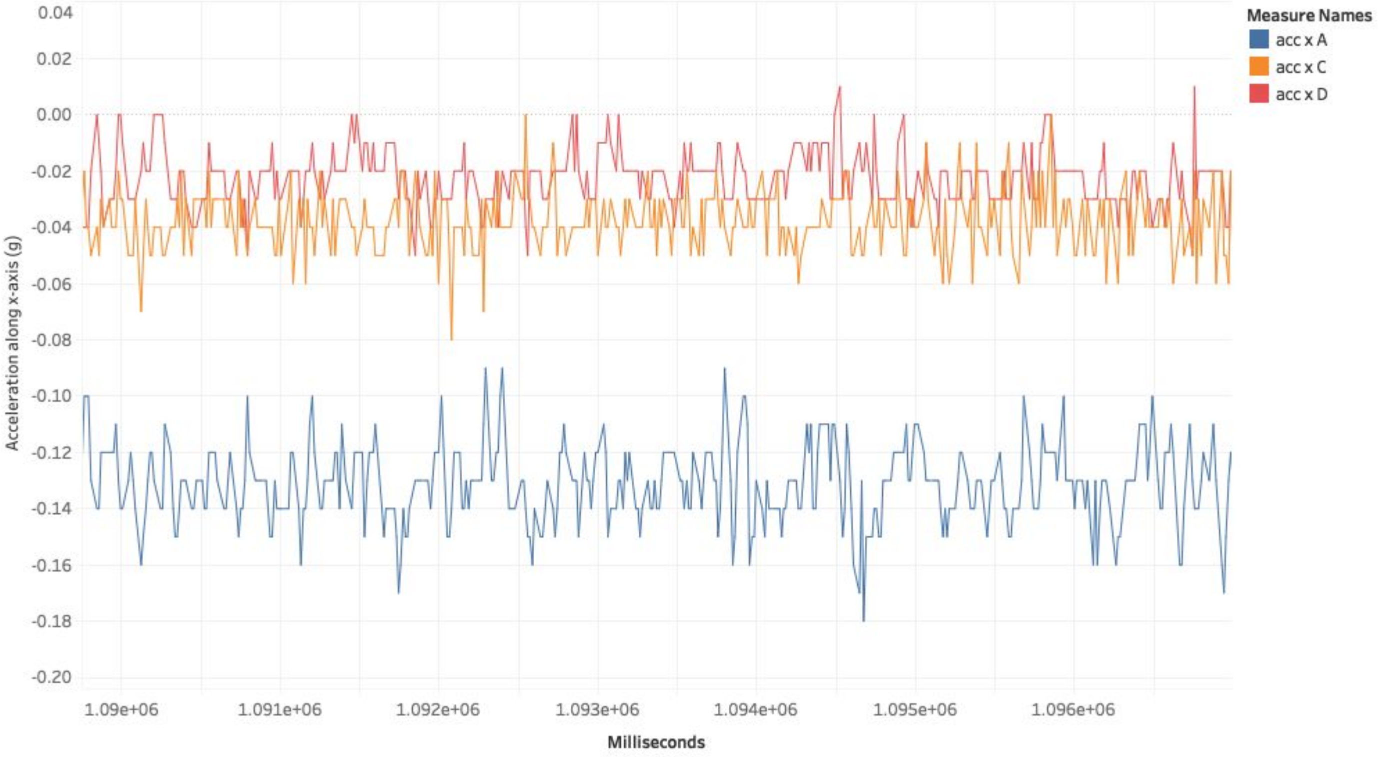
Acceleration data along x-axis measured on three different boards.




Millis vs. acc x A, acc x C and acc x D. Color shows details about acc x A, acc x C and acc x D.

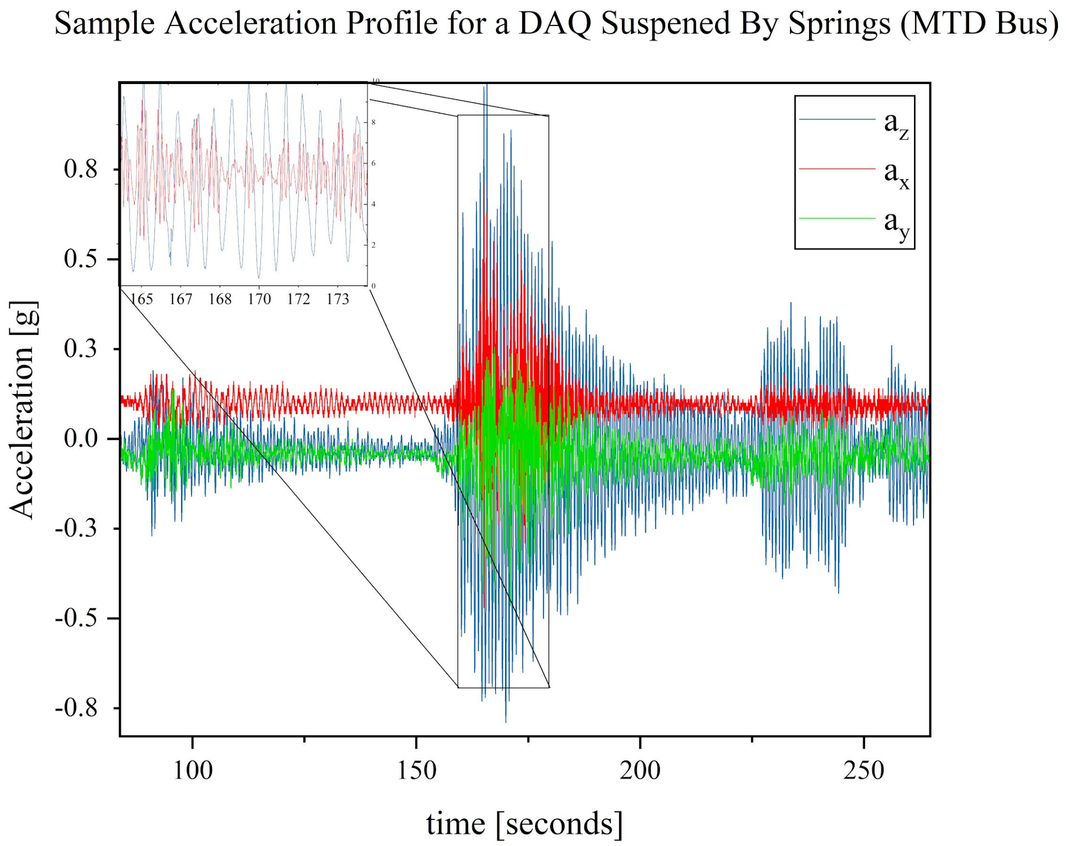
# DAQ Synchronization

Acceleration data along x-axis measured on three different boards.

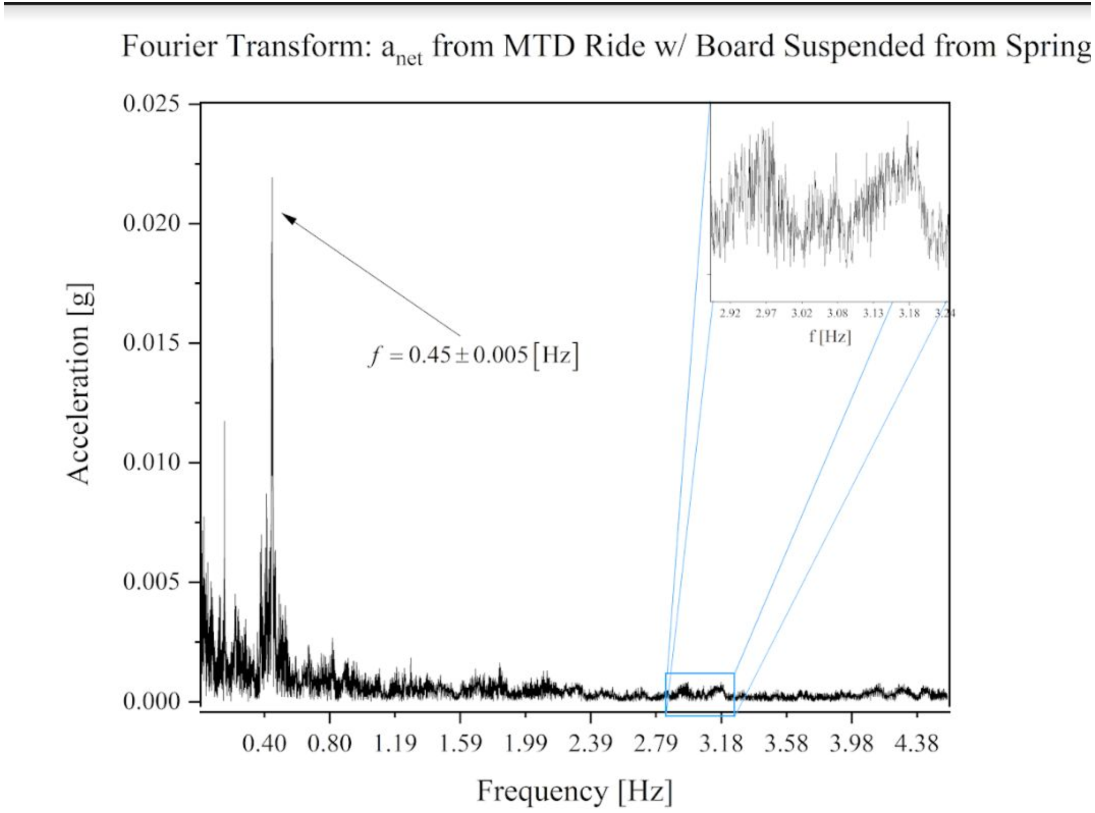



Millis vs. Avg. acc x A, Avg. acc x C and Avg. acc x D. Color shows details about Avg. acc x A, Avg. acc x C and Avg. acc x D.

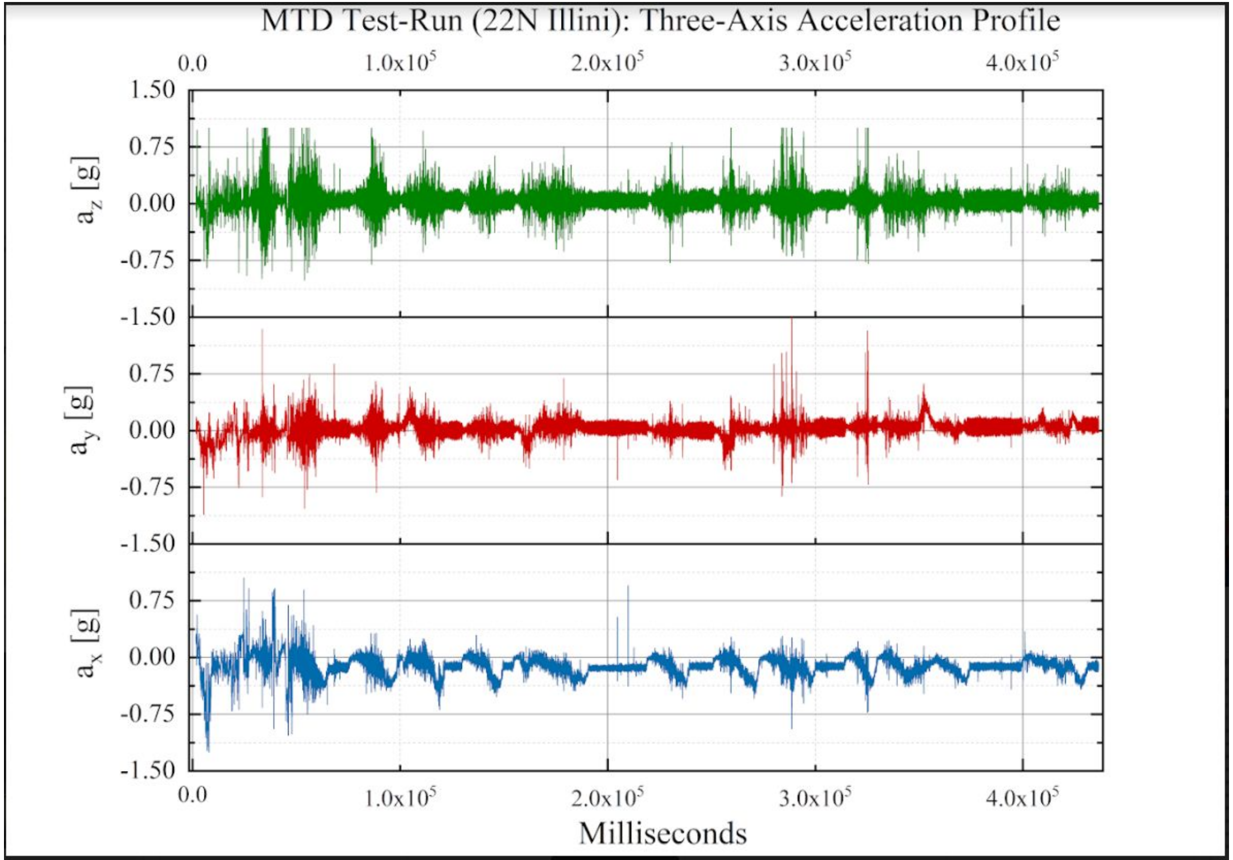
# Test-Runs - Champaign-Urbana MTD



# Test-Runs - Champaign-Urbana MTD



# Test-Runs - Champaign-Urbana MTD



# Test-Runs - Chicago Metra trains

BNSF Outbound from Chicago to Clarendon Hills, Il.
2:40pm - 3:25pm
**No antenna.**


Map based on Longitude and Latitude. Color shows Millis.

# Test-Runs - Chicago Metra trains

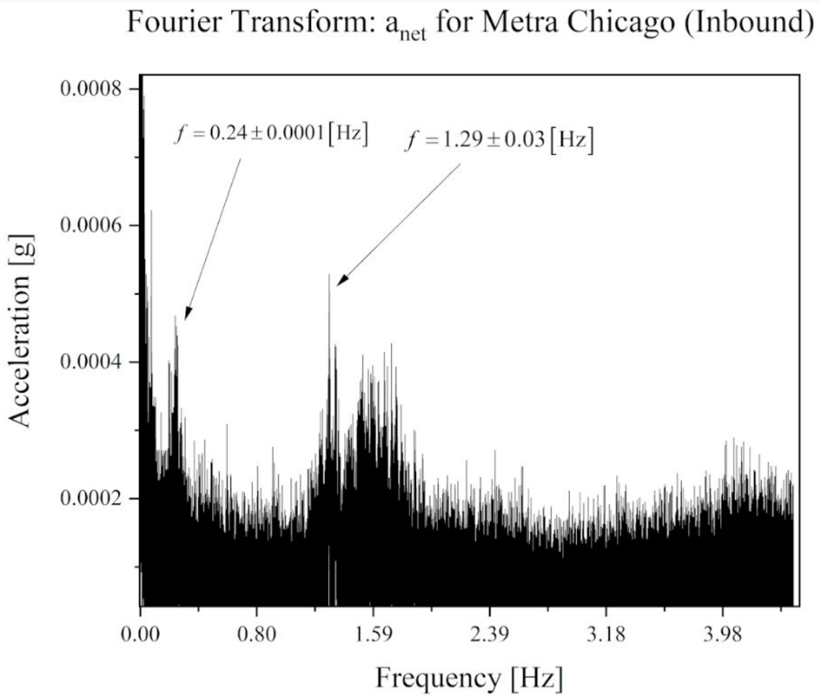
Metra BNSF Inbound (3/16/2019)
Repaired GPS Data
**Board D**

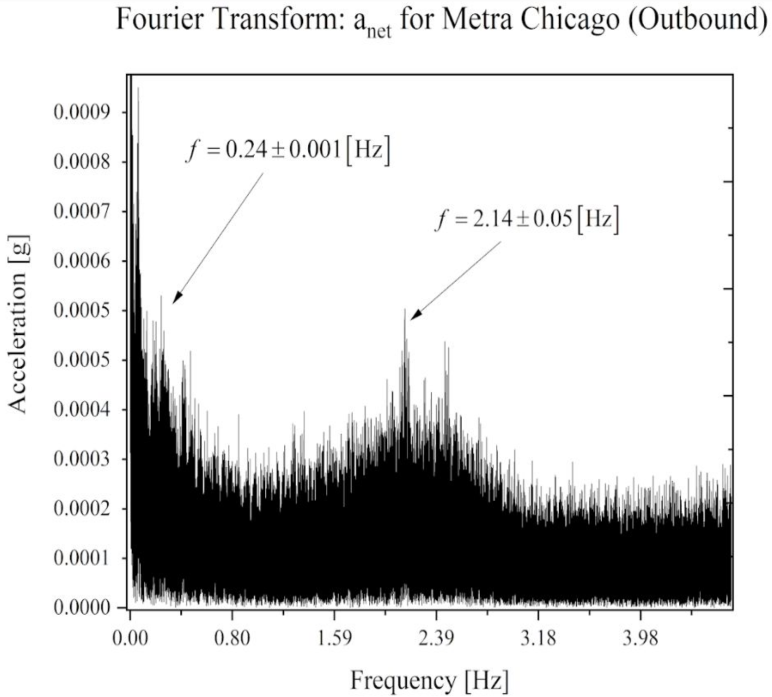

Map based on Longitude and Latitude. Color shows Millis. Size shows Acc.

**Net-Acceleration [g]**
- 0.0436
- 0.2000
- 0.4000
- 0.6000
- 0.8734

**Millis**
1,626 – 2,428,578

# Test-Runs - Chicago Metra trains



# Data Analysis - Characterizing Jolts/Lurches

- Analyze both acceleration & *jerk* to characterize severity of  jolts
- Overall ‘roughness’ of ride characterized w/ RMS of net-acceleration

$$\vec{\mathcal{J}} = \frac{d\vec{a}}{dt} = \frac{d^2\vec{v}}{dt^2} = \frac{d^3\vec{r}}{dt^3}$$

$$a_{\mathrm{RMS}} = \sqrt{\langle a_{\mathrm{net}} \rangle^2} = \sqrt{\frac{1}{N}\sum_{i}^{N}\left(a_{\hat{x},i}^2 + a_{\hat{y},i}^2 + a_{\hat{z},i}^2\right)}$$

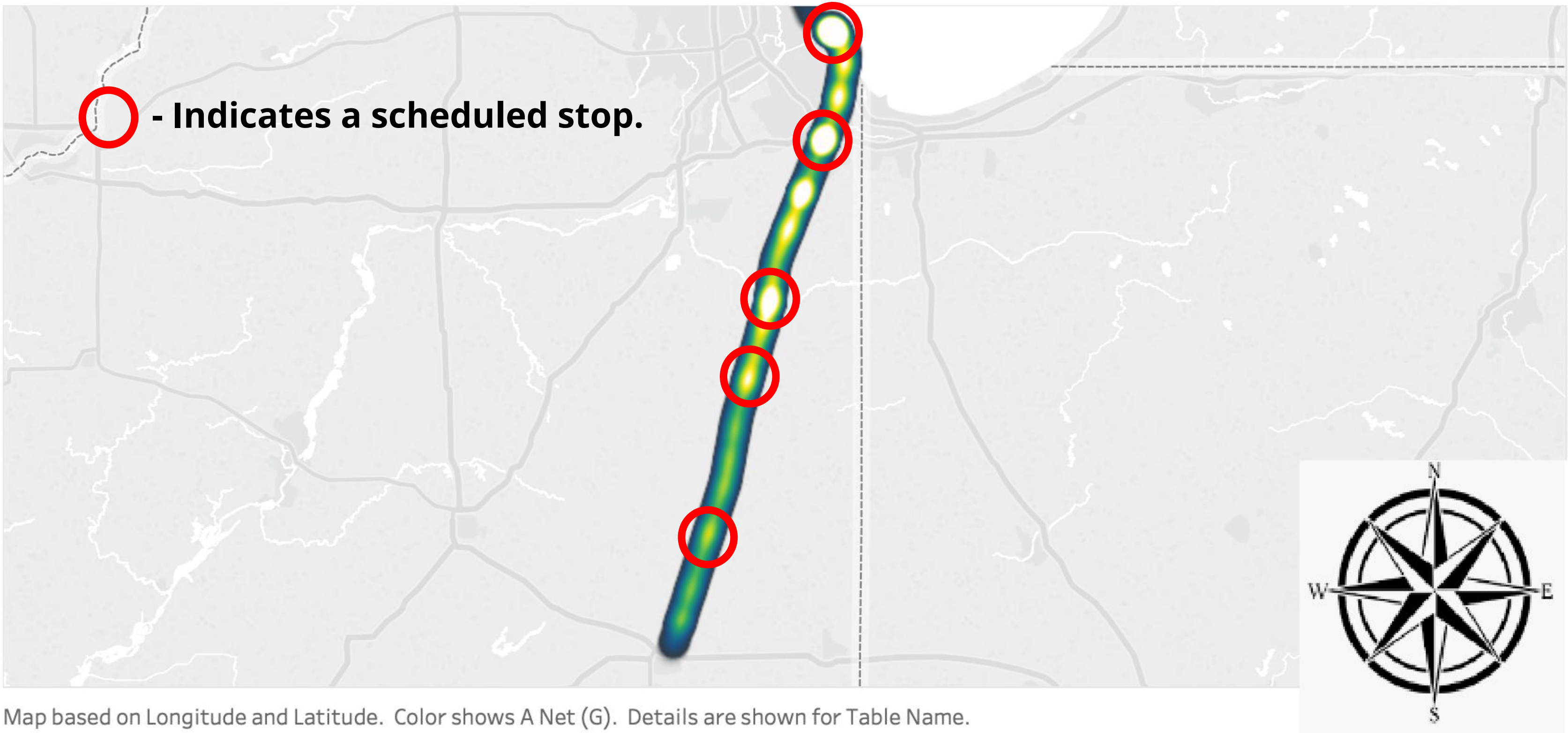
# Data Analysis - Characterizing Jolts/Lurches

- Net-acceleration calculated by adding respective x-y-z components in quadrature
- Extremely important to subtract-off the constant 1 [g] of acceleration measured in the z-direction due to Earth's gravity
- Dilution; washes-out the jolts we are actually interested in

$$a_{\text{net}} = \sqrt{a_{\hat{x}}^2 + a_{\hat{y}}^2 + a_{\hat{z}}^2}$$

# Amtrak: Chicago to Champaign

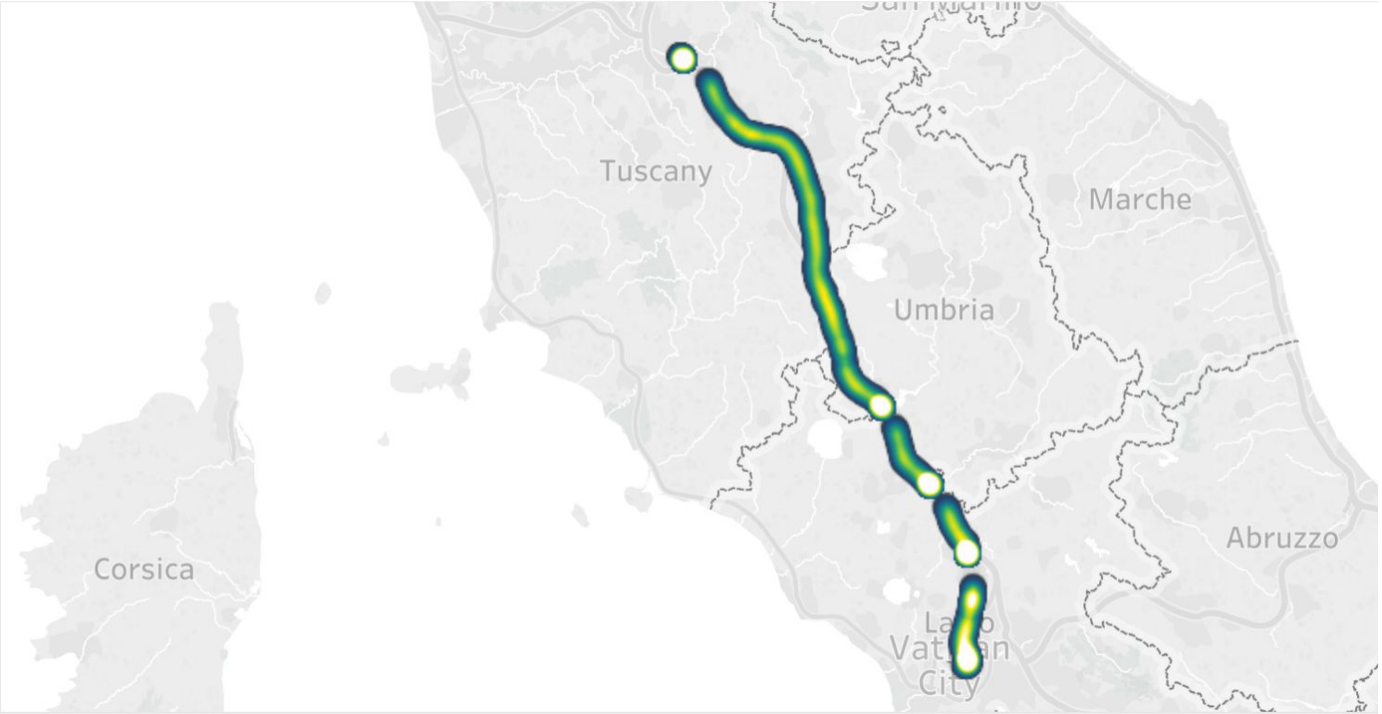
Net Acceleration (All Data, Full Track)




Map based on Longitude and Latitude. Color shows A Net (G). Details are shown for Table Name.

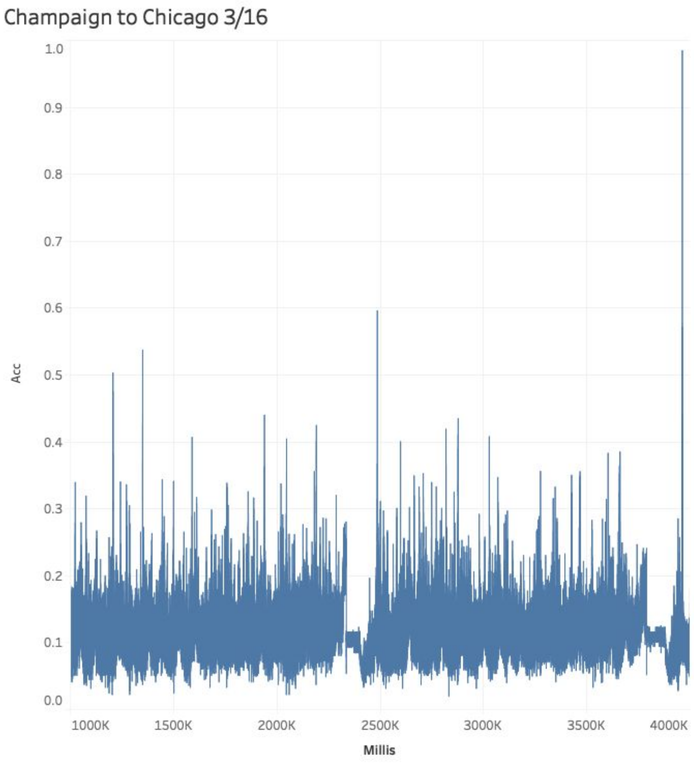
# Italian Railways

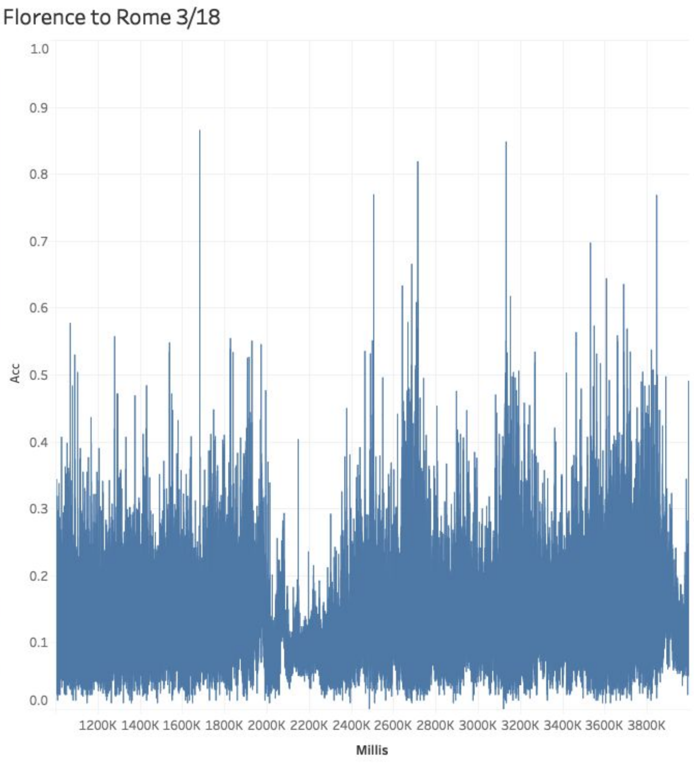
Florence to Rome




Map based on Longitude and Latitude.  Color shows Jerk.

# Italian Railways




<u>Scaled graphs</u>

Acceleration: 0-1g
Time: 1000k-4000k millis

# References

1. J. P. Powell, R. Palacin. *Passenger Stability Within Moving Railway Vehicles: Limits on Maximum Longitudinal Acceleration.* Urban Rail Transit (2015) 1(2):95–103. DOI: 10.1007/s40864-015-0012-y.

2. D. Martin, D. Litwhiler. *An Investigation of Acceleration and Jerk Profiles of Public Transportation Vehicles*. Pennsylvania State University-Berks. American Society for Engineering Education, 2008.

3. Adafruit Industries. Breakout-board datasheets.

4. American Society of Civil Engineers. *2017 Infrastructure Report Card: Rail.* p71 - p75.