**Signals Heatmap– How To Guide**

**Gracen Alleman, Intern 7/29/2024**

With the **Signals Heatmap Graph** and functions, you can make traffic counts for the Kinetic signals device event data. With the data you can make a graph showing how much each phase is being served or you can make a heatmap of the whole intersection over time and day. Please note that that although the functions are taking in turning movements, turning movements counts are being underreported. Also note that it is unclear how accurate the traffic counts are at the moment.

Get your data from Kinetic signals device event data

* Go to Kinetic signals and export as a csv device event data for the date range that you want. Note that some people cannot get more than 20 kilobytes of data from Kinetic at one time, so be conservative, think 5 days. If you need to, you can join your data after getting it from Kinetic.

Step 2: Load in functions and import libraries

* In Python, import the libraries io, timedelta, numpy as np, datetime as date, pandas, as pd, matplotlib.pyplot as plt, and plotly.graph\_objects as go. Load in the three functions myround, sortbyphaz, and traffic count.

Text

Description automatically generated

* Read in your Kinetic signals data as

> name\_for\_data\_frame = pd.read\_csv(“Name of file.csv”)

Step 3: Call the function *sortbyphaz*

* First format the *Timestamp* column to be in pd date time.

> df['Timestamp'] = pd.to\_datetime(df['Timestamp'], format="%m/%d/%Y %H:%M:%S.%f")

* Call the *sortbyphaz* function with the data frame as the input and the kwargs for every detector. Label the name of each detector with the phase number followed by an a, b, c, or r. You should be able to find the names of the detector on the electrical plan. You might get an error that says "Warning: Event Value 3 did not match any phase.” If you get that error, check to see if you are missing that value or if the detector doesn’t match up with anything in the electrical plan.

>df\_new=sortbyphaz(df,\*\*{'1a':[1],'1b':[29],'2a':[2],'2b':[3],'6a':[16],'6b':[17],'8a':[22],'8b':[23])

* The *sortbyphaz* function works by iterating over the rows in the Dataframe and seeing if the event ID matches the detector group, which is 81 through 100. It appends to the key matching the phases in the dictionary. It also looks for event IDs between 0 and 20, which are phase events to help with identifying what phase is being served.

> phases = {'1a': [], '1b': [],'1c':[], '2a': [] …

>for \_, row in df.iterrows():

>if 81 <= row['Event ID'] <= 100:

>for phase, values in kwargs.items():

>if row['Event Value'] in values

>phases[phase].append(row.copy())

>elif 0 <= event\_id <= 20:

>timeline.append(row)

* The function then iterates through the different open values in the dictionary and converts the values to a data frame, then it sorts by Timestamp and calculates the time between the previous row in seconds, which create the column Time Between Rows.

>for key, rows in phases.items():

>phases[key] = pd.DataFrame(rows)

>phases[key] = phases[key].sort\_values(by='Timestamp')

>phases[key]['Time Between Rows'] = phases[key]['Timestamp'].diff().dt.total\_seconds()

* The function then converts the timeline and sorts by time. It adds a new column that rounds the time to the nearest 15 minutes of date time minute using the function *myround*.

>timeline\_df = pd.DataFrame(timeline)

>timeline\_df = timeline\_df.sort\_values(by='Timestamp')

>new\_df = pd.concat(list(phases.values()) + [timeline\_df])

>new\_df = new\_df.sort\_values(by='Timestamp')

>new\_df['Minuterowd'] = new\_df['Timestamp'].apply( lambda dt: dt + pd.Timedelta(minutes=myround(dt.minute,15) - dt.minute,seconds=-dt.second, microseconds=-dt.microsecond))

Step 4: Call the function *traffic\_count*.

* Take the output from *sortbyphaz* and put that dataframe as an input for the function *traffic\_count*, with the kwargs for the different phases and for right turning movements indicated with the name of the phase followed by a r.

>df\_traffic\_count=traffic\_count(df\_new,\*\*{'1': [1,29], '2': [2,3], '6': [16,17], '8': [22,23]})

* Get rid of the initial counting by filtering the date 1970-01-01 00:00:00

>df\_traffic\_count = df\_traffic\_count.drop(df\_traffic\_count [df\_traffic\_count ['Start Minute'] == '1970-01-01 00:00:00'].index)

* Separate the column *start minute* into two columns, *date* and *time*.

> df\_traffic\_count ['date'] = df\_traffic\_count ['Start Minute'].apply(lambda x: x.date())

> df\_traffic\_count ['time'] = df\_traffic\_count ['Start Minute'].apply(lambda x: x.time())

* This data frame is the output of the functions. You could now export it as a csv or excel file, if wanted.
* The function works by initializing the keys in the dictionary which are the kwargs. It iterates over the rows, checking for detector off and checking if that time is equal to the previous 15 minutes it continues counting as normal the moment it sees a new 15 minute section it prints that count to a new row with the columns *Fifteen Minute Average, Start Minute,* and *Phase*. The *fifteen minutes average* converted to volume per hour, the *starting minute* column is the first of the fifteen minutes, and the label of the *phase*, i.e. ‘2,’ are all printed to the new row. The counting is also happening for the for the whole intersection, which is the same method but just for every phase: these two things are happening in parallel with each other.

>counts = { 'intersection': 0, '1': 0, '2': 0, …

>old = rawdata.iloc[0]['Minuterowd'].minute # Initialize the time variable for the intersection as a whole with the first minute.

>phase\_old = {key: rawdata.iloc[0]['Minuterowd'].minute for key in counts.keys() if key != '15'}

>for c, row in rawdata.iterrows():

>if row['Event Name'] == "Detector Off":

>if row['Minuterowd'] == old:

>counts['intersection'] += 1

>old= row['Minuterowd']

>else: new\_row = { "Fifteen Minute Average": (counts['intersection'] \*4), "Start Minute": pd.to\_datetime(old), "Phase": "intersection"}

>newdf.append(new\_row)

>old= row['Minuterowd']

>counts['intersection'] = 1

>for key, value in kwargs.items():

>if row['Event Value'] in value:

>if row['Minuterowd'] == phase\_old[key]:

>counts[key] += 1

>phase\_old[key] = row['Minuterowd']

>else: new\_row = {"Fifteen Minute Average": counts[key] \* 4, "Start Minute": pd.to\_datetime(phase\_old[key]), "Phase": key}

>newdf.append(new\_row)

>counts[key] = 1

>phase\_old[key] = row['Minuterowd']

Step 4.5: If you are joining multiple time periods together:

* One has two options: you could join them together when you read in both data sets, or after both as gone through the functions. It would make sense to do the latter when one has already gone through functions.

>df = pd.concat([join1, join2], ignore\_index=True)

Step 5: Graph

* I use plotly.graph\_objects to graph in python, but you can use whatever you desire.

>dfnew = df\_traffic\_count.sort\_values(by='Start Minute', ascending=False)

> df\_new\_new = dfnew[dfnew['Phase'] == 'intersection']

>contour\_plot = go.Heatmap(x= df\_new\_new ['date'], y= df\_new\_new ['time'], z= df\_new\_new ["Fifteen Minute Average"],colorscale='Sunset', reversescale=False, colorbar=dict( title="Traffic Count", titleside="top"))

>fig = go.Figure(data=[contour\_plot])

>fig.update\_layout(title='heat map of the intersshon', height=650)

>fig.show()

* One could also graph using *plotly.express* as px to graph the different phases.

> fig = px.line(df\_traffic\_count, x='Start Minute', y='Fifteen Minute Average', color='Phase', color\_discrete\_map={'1': 'red', '2': 'blue', '3':'Green'})

>fig.show()

**Next steps in heat map project.**

Using the amount of time the detector is on for more complex calculations and estimations of traffic counts. Beter calculations could be used for better traffic counts of turning movements, which are being undercounted. This could also be paired with any timeline data to figure out what phase is being served or if there are any events happening in the intersection.