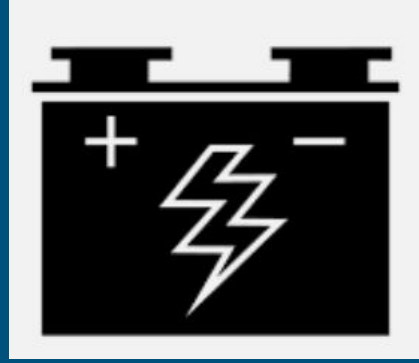
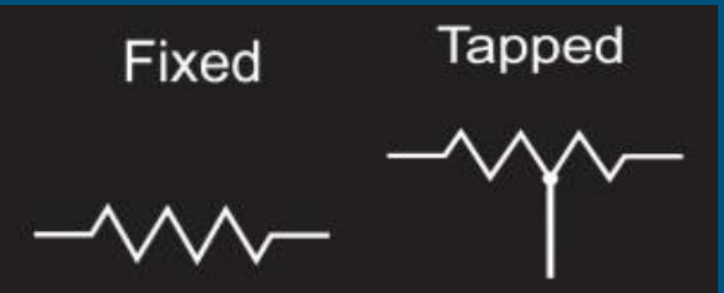
AI-VVO sdmay22-36 Weekly Update #8

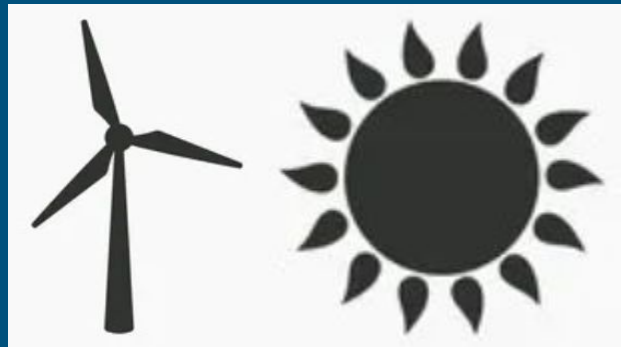
11/16/2021 - 12/2/2021

Front-end (This Week)

- Added power resistor icons to google doc
- Sun represents solar generation
- Windmill represents wind generation
- Fixed icon for line segments between buses
- Tapped for voltage regulators







Front-end (This Week)

- Looked into sending data to backend through marker click
- Use axios to send data to the backend

```
35    axios.post('LineData.csv', {
36        nodes: nodes,
37        links:links
38    })
39    .then(function (response) {
40        console.log(response);
41    })
42    .catch(function (error) {
43        console.log(error);
44    });
45
46    const markerPress = () => {
47        document.getElementById("marker").style.display = 'none'
48        var newData = {
49            nodes:nodes,
50            links:links
51        }
52        setData(newData);
53    }
```

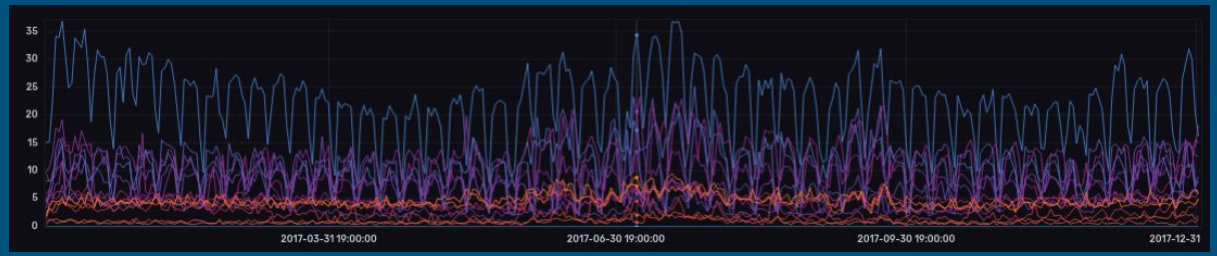
```
63            {locations.map((location) => (
64                <Marker
65                position={location.position}
66                icon={GetIcon(location.size, location.iType)}
67                id={location.id}
68                onClick={markerPress}
69                >
70                    <Popup>
71                        {location.name} - {location.iStatus}
72                    </Popup>
73                </Marker>
74            ))};
```

Front-end (Next Week)

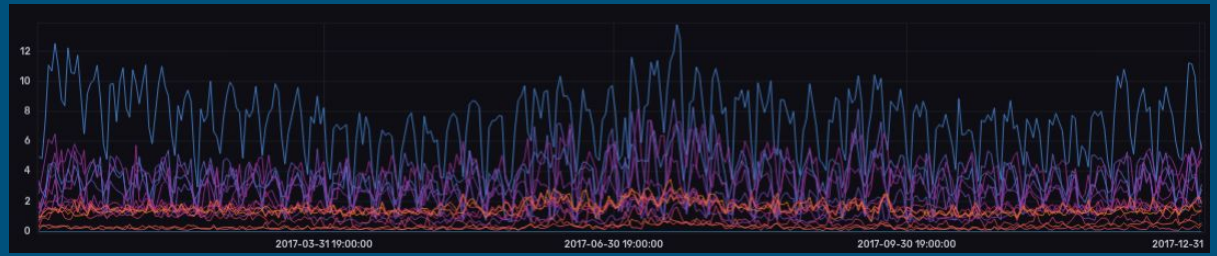
- Present to senior design panel

Back-end (This Week)

- Added Feeder data to InfluxDB
- Refactored csvParser.py file to upload to InfluxDB



^ FeederAP data



^ FeederAQ data

Back-end (This Week)

- Using the Pandas dataframe and InfluxDBClient we can easily read or write data to Influx

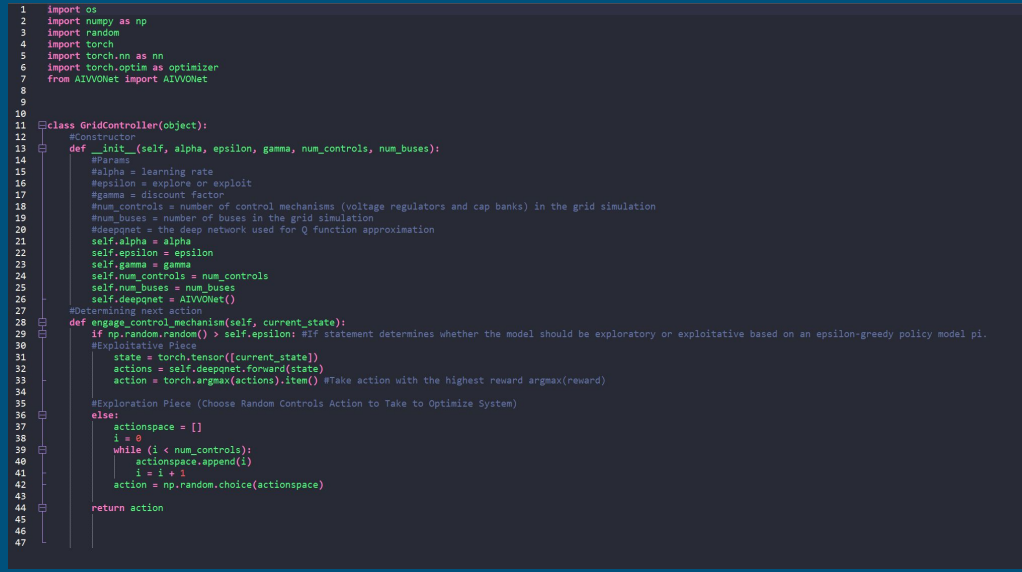
```
client = InfluxDBClient(url="http://localhost:8086", token=token, org=org)


#the P and Q feeder data have different time formats
timeFormatP = "%m/%d/%Y %H:%M"
timeFormatQ = "%m/%d/%y %I:%M %p"
filePath = "/home/ubuntu/Documents/development/sdmay22-36/research/"

#initialize all the csv objects
feeders = [
            FeederDataCsv("FeederAP", "FeederAP.csv", timeFormatP),
            FeederDataCsv("FeederAQ", "FeederAQ.csv", timeFormatQ),
            FeederDataCsv("FeederBP", "FeederB_P.csv", timeFormatP),
            FeederDataCsv("FeederBQ", "FeederB_Q.csv", timeFormatQ),
            FeederDataCsv("FeederCP", "FeederC_P.csv", timeFormatQ),
            ]

for feeder in feeders:
    #create dataframe from csv
    dataframe = pd.read_csv(filePath + feeder.fileName)

    #add the measurement column that is required by influx
    dataframe['datapoint'] = 'datapoint'

    #reformat the date format is different is P vs Q files
    dataframe['time'] = pd.to_datetime(dataframe['time'], format=feeder.timeFormat)
    dataframe.set_index(['time'], inplace=True)

    #create influxdb write client
    writeClient = client.write_api()

    writeClient.write(feeder.bucket, org, dataframe, data_frame_measurement_name="datapoint", protocol='line')

    writeClient.close()
    print('upload of ' + feeder.fileName + ' completed')

print('finished!')
```

Back-end (Next Week)

- Finish presentation for Senior Design Panel

Machine Learning (This Week)

- Wrote GridController agent class that implements an epsilon greedy policy
- Epsilon greedy policy weighs the benefits of exploiting currently known rewards and exploring for greater rewards with the risk of losing out on the currently known rewards
- Pushed class files to Gitlab



```python
import os
import numpy as np
import random
import torch
import torch.nn as nn
import torch.optim as optimizer
from AIVVONet import AIVVONet



class GridController(object):
    #Constructor
    def __init__(self, alpha, epsilon, gamma, num_controls, num_buses):
        #Params
        #alpha = learning rate
        #epsilon = explore or exploit
        #gamma = discount factor
        #num_controls = number of control mechanisms (voltage regulators and cap banks) in the grid simulation
        #num_buses = number of buses in the grid simulation
        #deepqnet = the deep network used for Q function approximation
        self.alpha = alpha
        self.epsilon = epsilon
        self.gamma = gamma
        self.num_controls = num_controls
        self.num_buses = num_buses
        self.deepqnet = AIVVONet()
    #Determining next action
    def engage_control_mechanism(self, current_state):
        if np.random.random() > self.epsilon: #If statement determines whether the model should be exploratory or exploitative based on an epsilon-greedy policy model pi.
        #Exploitative Piece
            state = torch.tensor([current_state])
            actions = self.deepqnet.forward(state)
            action = torch.argmax(actions).item() #Take action with the highest reward argmax(reward)

        #Exploration Piece (Choose Random Controls Action to Take to Optimize System)
        else:
            actionspace = []
            i = 0
            while (i < num_controls):
                actionspace.append(i)
                i = i + 1
            action = np.random.choice(actionspace)

        return action
```

Machine Learning (This Week)

- Explanation of parameters:
  - alpha is the learning rate of the system
  - epsilon is the factor between 0 and 1 that determines if the GridController will exploit or explore
  - gamma is the discount factor that determines the time horizon for what rewards should be considered in the system (implementation of discount factor will come later after further research into what our time horizon should be)
  - num _buses is the number of buses in the simulated grid
  - num_controls is the number of control mechanisms in the simulated grid
  - deepqnet is an instantiation of the AIVVONet() class and is used for determining the set of actions available for an agent in a given state

Machine Learning (This Week)

- Wrote main class that will be the runtime of the GridController and AIVVONet instances

```python
import os
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optimizer
from AIVVONet import AIVVONet
from GridController import GridController


#Params
#alpha = learning rate
#epsilon = explore or exploit
#gamma = discount factor
#num_controls = number of control mechanisms (voltage regulators and cap banks) in the grid simulation
#num_buses = number of buses in the grid simulation

 Agent = GridController(alpha=0.05, epsilon=1, gamma=0.5,num_controls=2662,num_buses=239)

 num_epochs = #Will be determined by experimentation during testing

for i in range(num_epochs)

    #Data will be pulled from backend database and imported into Agent GridController class

    #This data,in conjunction with forward function, will be used to predict what future reward will be

    #Using this reward we will predict an action

    #We will then output our concluded data to be stored in the database
```

Machine Learning (Next Week)

- Present to the Senior Design Panel