Evaluation Module

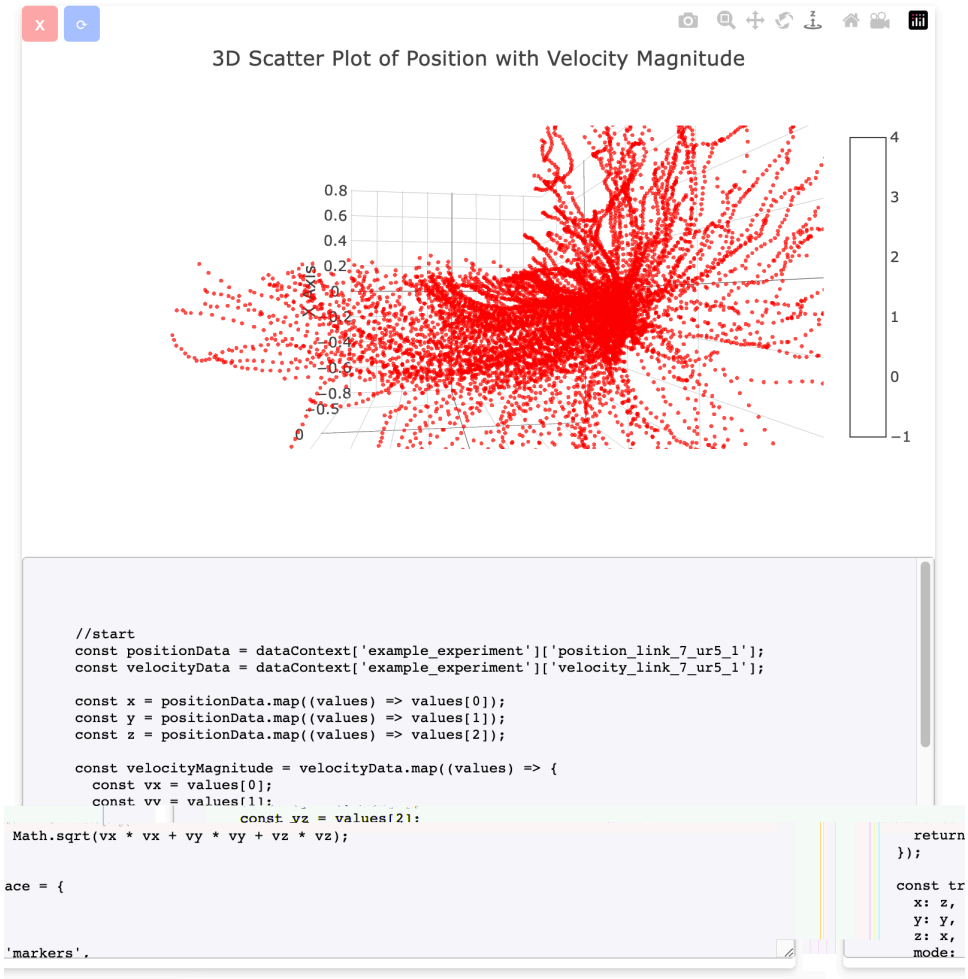
Link to repo and branch:

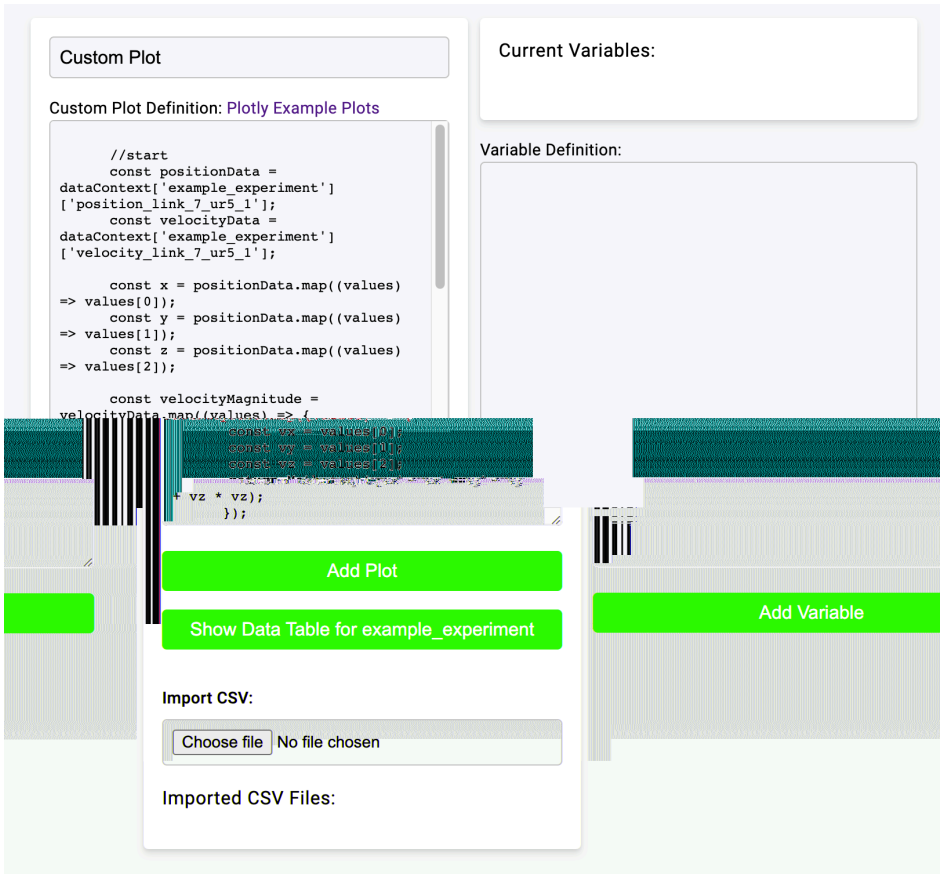
https://github.com/ignc-research/IRAF-AI/tree/Evaluation

The Evaluation Module is an Angular application that provides interactive visualization of experiment data from CSV files. It includes multiple types of plots, such as trajectory, bar, and line plots, as well as a custom plot component. Users can import their own CSV files, and the module will generate plots based on the data provided.

Features

- Import CSV files containing experiment data
- Display custom plots
- Data table for viewing raw data
- Customizable plots through user-provided code
- Add custom Functions and Variables





Installation

1) Clone the repository and navigate to the project directory:

git clone --single-branch —branch Evaluation https://github.com/ignc-research/IRAF-AI.git
cd fe

2) Install the required dependencies:

npm install —legacy-peer-deps

3) Inside /fe start the Angular Development Server

ng serve

4) Open http://localhost:4200/evaluation (port might be different for you)

How it works

Main Components

The Evaluation Module consists of several components and services:

- **EvaluationComponent**: The main component of the module, which manages the overall layout and user interactions.
- **CustomPlotComponent**: A component that allows users to create their own custom plots by providing JavaScript code.
- **DataTableComponent**: A component for displaying the raw data from the experiments in a tabular format.

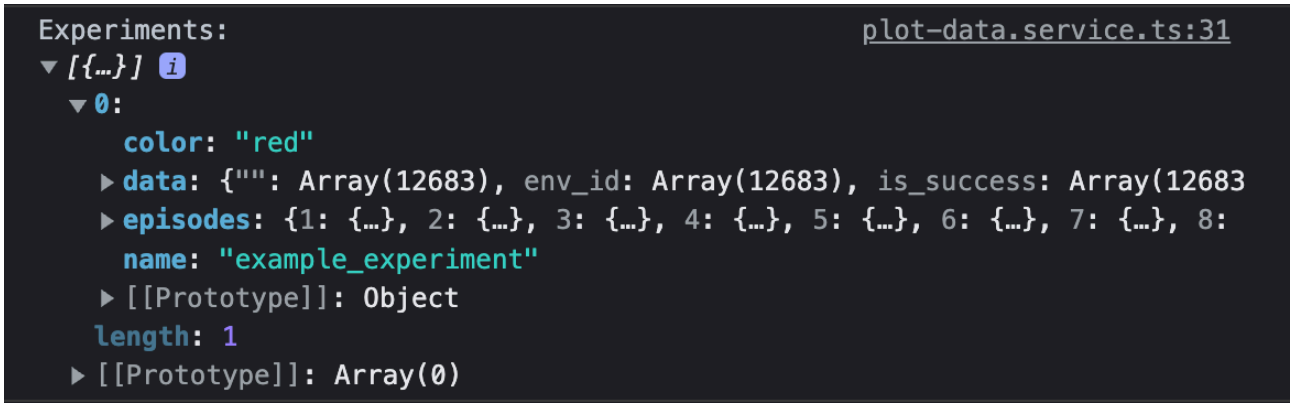
Services

- **PlotDataService**: A service responsible for loading and processing the CSV data, as well as managing the experiments.
- **ColorService**: A service for generating and managing colors used in the plots.

Process

- The user imports a CSV file containing experiment data.

- The **PlotDataService** processes the CSV data and groups it by experiment and episode.



- The **EvaluationComponent** displays the available plots based on the processed data.

- The user can interact with the plots and use all Plotly UI controls.

- Users can define custom variables and functions which are executed on adding. These custom definitions have access to the imported data through the `dataContext` variable, which stores all experiments and their specific data fields. (*Check Example Variables*)
    - dataContext data is accessible through: dataContext[<experiment name>][<column name>]
    - **Demo of functions and variables:** https://github.com/ignc-research/IRAF-AI/blob/Evaluation/fe/evaluation_demo.mp4

- Custom definitions can utilize external libraries like NumJS or any other pre-added variables to process and manipulate the data for custom plots or calculations.

- To create a custom plot, the user can provide JavaScript code that generates the plot using the **CustomPlotComponent**. Additionally, users can save and share their custom plots.
    - Custom added variables and functions are fully usable as variables in the custom plot definition.
    - Example Plotly definitions: https://plotly.com/javascript/
- Upon adding a new custom plot, all existing custom variables or functions are included in the plot definition code as constants holding the result values. In that way the values are available for use in the plots.

- The user can also view the raw data in a table format using the **DataTableComponent**.

- Custom definitions can utilize external libraries like NumJS or any other pre-added variables to process and manipulate the data for custom plots or calculations.

Example Variable Definitions

1) Custom Function using NumJS

```
const ajs = function() {
  const data = nj.array([1, 2, 3, 4, 5]);
  const meanValue = nj.mean(data);
  return meanValue;
};
```

2) Custom Function using NumJS and CSV data

```
const abs = function() {
  const data = nj.array(dataContext['example_experiment']['position_link_7_ur5_1'][0]);
  const meanValue = nj.mean(data);
```

```
  return meanValue;
};
```

3) Custom Variable using CSV data array and number

```
const arrayval = dataContext['example_experiment']['position_link_7_ur5_1'][1];

const numberval = dataContext['example_experiment']['position_link_7_ur5_1'][1][0];
```

4) Custom Variable using NumJS and number variable defined in 3)

```
const acs = function() {
  const data = nj.array([1, 2, 3, 4, numberval]);
  const meanValue = nj.mean(data);
  return meanValue;
};
```

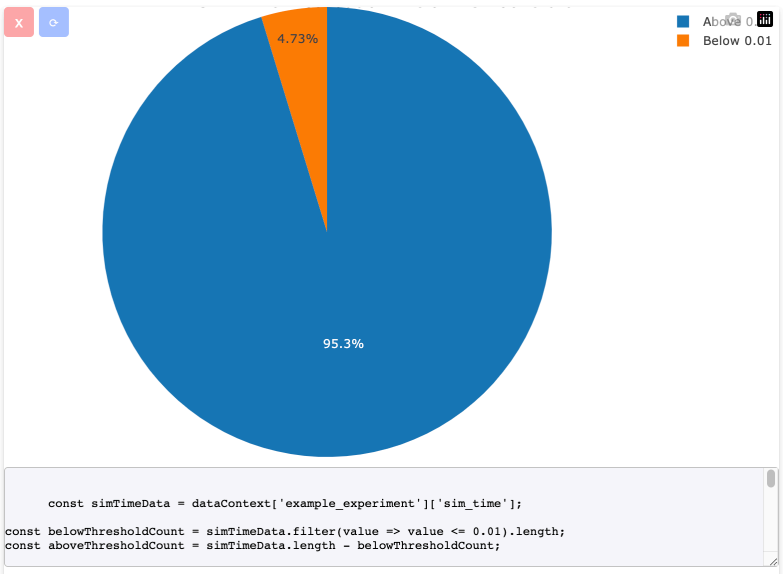
Example Plot Definitions

1) Averages Bar Plot

```
const successRateData = dataContext['example_experiment']['success_rate'];
  const outOfBoundsRateData = dataContext['example_experiment']['out_of_bounds_rate'];
  const timeoutRateData = dataContext['example_experiment']['timeout_rate'];
  const collisionRateData = dataContext['example_experiment']['collision_rate'];

  const trace1 = {
    x: ['Success Rate', 'Out of Bounds Rate', 'Timeout Rate', 'Collision Rate'],
    y: [
      successRateData.reduce((a, b) => a + b, 0) / successRateData.length,
      outOfBoundsRateData.reduce((a, b) => a + b, 0) / outOfBoundsRateData.length,
      timeoutRateData.reduce((a, b) => a + b, 0) / timeoutRateData.length,
      collisionRateData.reduce((a, b) => a + b, 0) / collisionRateData.length,
    ],
    type: 'bar',
  };

  const layout = {
    title: 'Experiment Averages',
    xaxis: { title: 'Metrics' },
    yaxis: { title: 'Rate' },
  };

  const data = [trace1];

  return { data: data, layout: layout };
```

2) Pie Plot

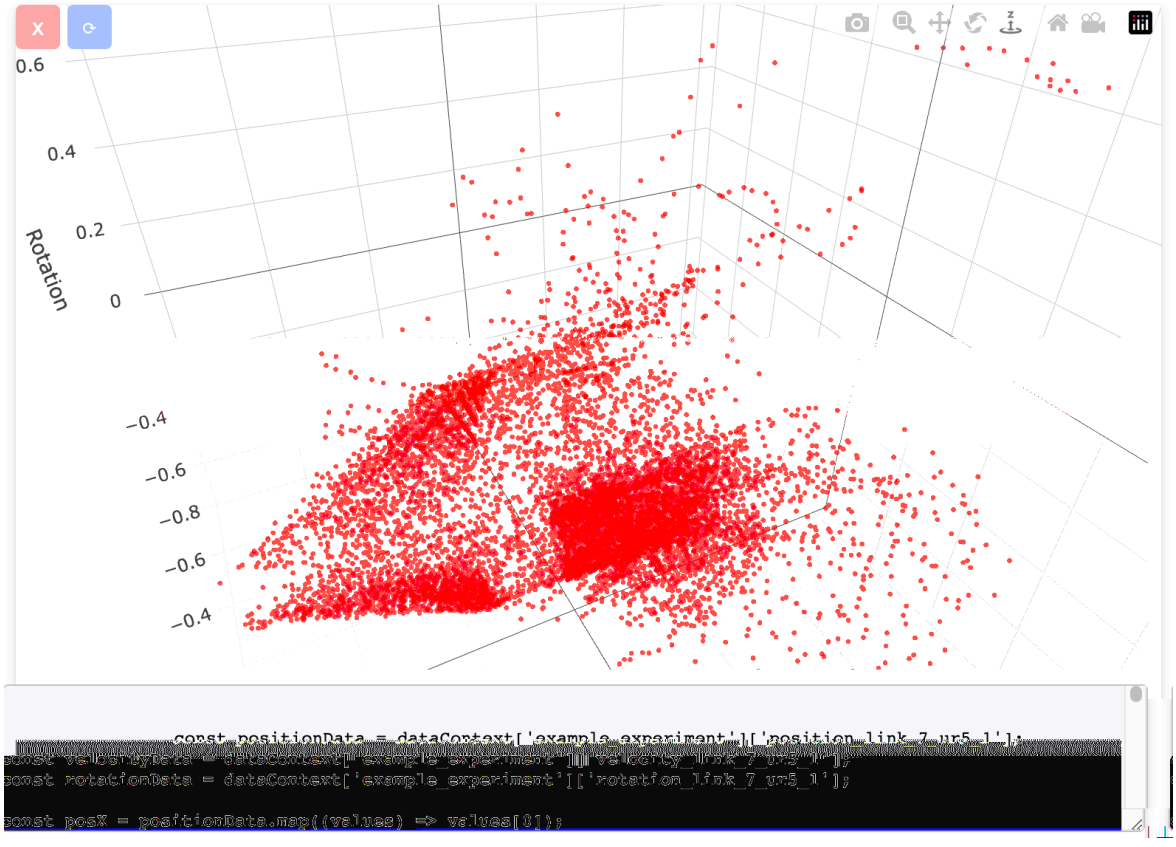
```
const simTimeData = dataContext['example_experiment']['sim_time'];

const belowThresholdCount = simTimeData.filter(value => value <= 0.01).length;
const aboveThresholdCount = simTimeData.length - belowThresholdCount;

const trace = {
  labels: ['Below 0.01', 'Above 0.01'],
  values: [belowThresholdCount, aboveThresholdCount],
  type: 'pie',
};

const layout = {
  title: 'Sim Time Distribution: Below vs Above 0.01',
  margin: {
    l: 0,
    r: 0,
    b: 0,
    t: 0,
  },
};

const data = [trace];

return { data: data, layout: layout };
```



3) Scatter Plot

```
        const positionData = dataContext['example_experiment']['position_link_7_ur5_1'];
const velocityData = dataContext['example_experiment']['velocity_link_7_ur5_1'];
const rotationData = dataContext['example_experiment']['rotation_link_7_ur5_1'];

const posX = positionData.map((values) => values[0]);
const posY = positionData.map((values) => values[1]);
const posZ = positionData.map((values) => values[2]);

const velX = velocityData.map((values) => values[0]);
const velY = velocityData.map((values) => values[1]);
const velZ = velocityData.map((values) => values[2]);

const rotX = rotationData.map((values) => values[0]);
const rotY = rotationData.map((values) => values[1]);
const rotZ = rotationData.map((values) => values[2]);

const trace = {
  x: posX,
  y: velY,
  z: rotZ,
  mode: 'markers',
  marker: {
    size: 2,
    color: posZ,
    colorscale: 'Blacks',
    opacity: 0.8,
    symbol: 'circle',
  },
  type: 'scatter3d',
};

const layout = {
  scene: {
    xaxis: { title: 'Position' },
    yaxis: { title: 'Velocity' },
```

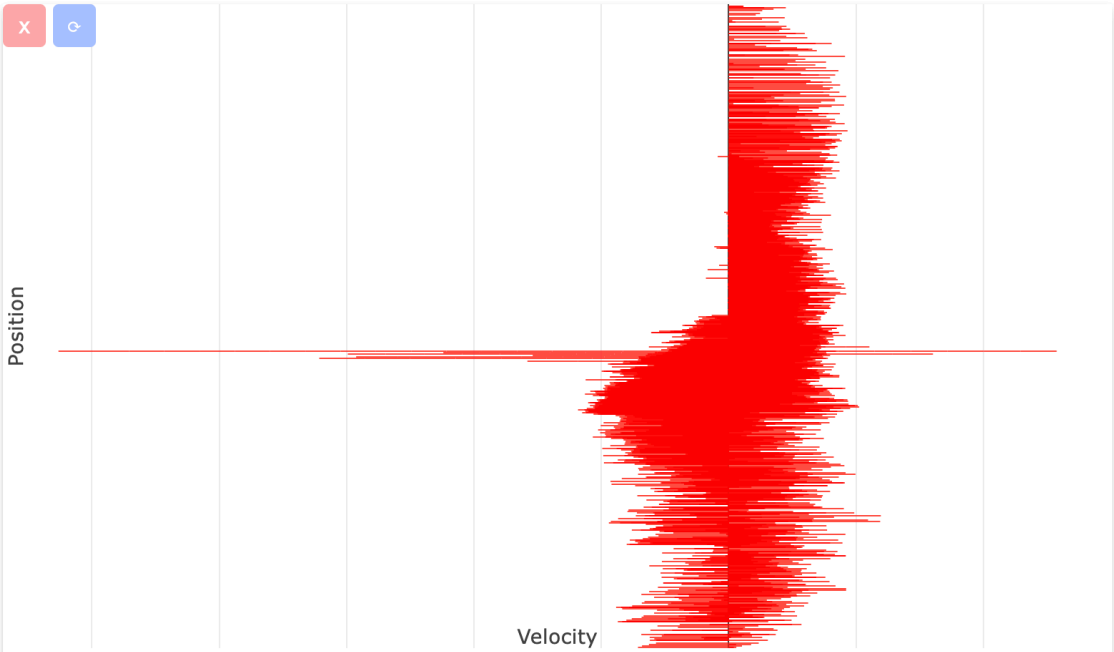
```
    zaxis: { title: 'Rotation' },
  },
  margin: {
    l: 0,
    r: 0,
    b: 0,
    t: 0,
  },
};

const data = [trace];

return { data: data, layout: layout };
        return { data: data, layout: layout };
```



4) Horizontal Bar Plot

```
  const positionData = dataContext['example_experiment']['position_link_7_ur5_1'];
const velocityData = dataContext['example_experiment']['velocity_link_7_ur5_1'];
const rotationData = dataContext['example_experiment']['rotation_link_7_ur5_1'];

const posX = positionData.map((values) => values[0]);
const posY = positionData.map((values) => values[1]);
const posZ = positionData.map((values) => values[2]);

const velX = velocityData.map((values) => values[0]);
const velY = velocityData.map((values) => values[1]);
const velZ = velocityData.map((values) => values[2]);

const rotX = rotationData.map((values) => values[0]);
const rotY = rotationData.map((values) => values[1]);
const rotZ = rotationData.map((values) => values[2]);



const trace = {
  x: velY,
  y: posX,
  marker: {
    color: posZ,
    colorscale: 'Viridis',
    opacity: 0.8,
  },
  type: 'bar',
  orientation: 'h', // horizontal orientation
};

const layout = {
  xaxis: { title: 'Velocity' },
  yaxis: { title: 'Position' },
  margin: {
```

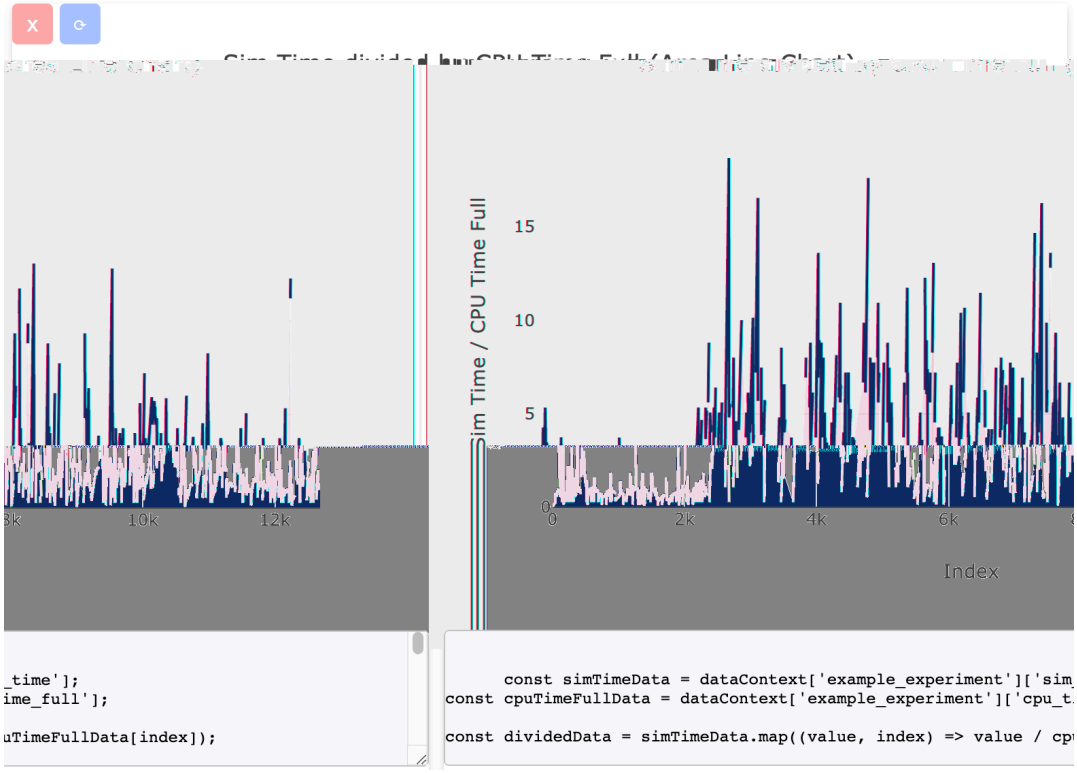
```
    l: 0,
    r: 0,
    b: 0,
    t: 0,
  },
};

const data = [trace];

return { data: data, layout: layout };
```




```
  const positionData = dataContext['example_experiment']['position_link_7_ur5_1'];
const velocityData = dataContext['example_experiment']['velocity_link_7_ur5_1'];
```

5) Line Area Plot

```
const simTimeData = dataContext['example_experiment']['sim_time'];
const cpuTimeFullData = dataContext['example_experiment']['cpu_time_full'];

const dividedData = simTimeData.map((value, index) => value / cpuTimeFullData[index]);

const trace = {
  x: Array.from({ length: dividedData.length }, (v, k) => k + 1), // Assuming a sequential index
  y: dividedData,
  mode: 'lines',
  line: {
    color: 'darkgreen', // A nice blue color
    width: 2,
  },
  fill: 'tozeroy',
  fillcolor: 'rgba(31, 119, 180, 0.2)',
  name: 'Sim Time / CPU Time Full',
};

const layout = {
  title: 'Sim Time divided by CPU Time Full (Area Line Chart)',
  xaxis: {
    title: 'Index',
  },
  yaxis: {
    title: 'Sim Time / CPU Time Full',
  },
};

const data = [trace];

return { data: data, layout: layout };
```


Sim Time / CPU Time Full
15
10
5
0
0
2k
4k
6k
Index
10k
12k
_time'];
ime_full'];
uTimeFullData[index]);
const simTimeData = dataContext['example_experiment']['sim
const cpuTimeFullData = dataContext['example_experiment']['cpu_t
const dividedData = simTimeData.map((value, index) => value / cp