INTERFACE CONTROL DOCUMENT

Revision A

AIR FORCE RESEARCH LAB SCHOLARS - SUMMER 2020

AUTONOMOUS ROVER INTERFACE WITH OPENMCT FRAMEWORK

Revisions

REVISION	DATE	DESCRIPTION	APPROVAL
A	07/29/2020	Initial ICD creation.	

APPROVED BY: ____ DATE: ________________

APPROVED BY: ____ DATE: ________________

APPROVED BY: ____ DATE: ________________

Table of Contents

Table of Figures

1.0 SCOPE

1.1 Purpose

The autonomous rover interface visualizes a multi-rover exploration domain that utilizes a neuro-evolutionary learning algorithm facilitated through CCEA and basic reward functions. OpenMCT is used as the primary framework for visualizing data, and data transfer is done via HTTP servers.

2.0 INTERFACE REQUIREMENTS

In order to use the autonomous rover interface, the user must first download the code from the corresponding repository: https://github.com/Noveriia/RoverInterface

Any code editor or IDE may be used to open the files. Visual Studio Code was used during production. OpenMCT is primarily programmed in JavaScript with additional functionalities added by Node.js. The rover domain is programmed in python. You will need Node.js installed to properly use this interface, and though not mandatory, it is also strongly suggested to use git as well.

2.1 Pre-requisite Downloads

Node.js

a) Mac OS: Homebrew (link to Homebrew main page) is recommended to install node.

```
$ brew install node
```

b) Windows: https://nodejs.org/en/download/ (Link to Node.js official site)
c) Linux: https://nodejs.org/en/download/ (Link to Node.js official site)

Git

a) Mac OS: If you have XCode installed then git is already available from your command line. If not, git can be installed using Homebrew (link to Homebrew main page).

```
$ brew install git
```

b) Windows: https://git-scm.com/downloads (Link to git's official site)
c) Linux: https://git-scm.com/downloads (Link to git's official site)

Installing the Interface

```
git clone https://github.com/Noveriia/RoverInterface.git
```

2.2 Starting up the Interface

In order to run the system correctly, you should start the python server, then start the main rover code, then start OpenMCT. Both OpenMCT and the main rover code are dependent on the python server supplementing data, and they will crash otherwise. The OpenMCT component, main python file and python server should all be run on separate terminals concurrently.

To run the backend component:

```
cd roverdomain
python server.py
python main.py
```

To run the frontend component:

```
cd openmct
npm install
npm start
```

All the servers used are run locally, and the program runs both on and offline. You can then find the OpenMCT code running on: http://localhost:8080/

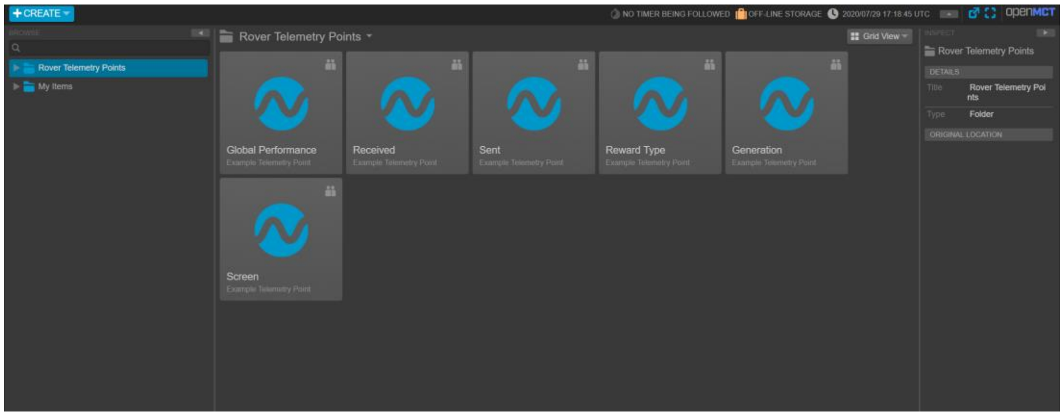


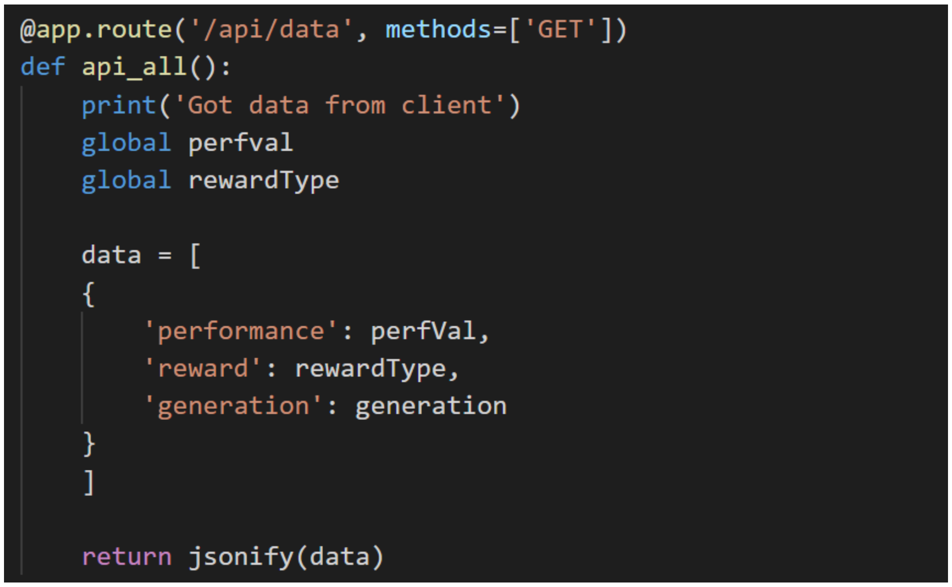
Figure 1. Telemetry points on a running OpenMCT webpage.

3.0 PYTHON SERVER

The server and application programming interface (API) for the backend rover domain are contained in the “server.py” file under the “roverdomain” directory. The purpose of the server is to facilitate data transfer between the front end and the backend with RESTful API. The python server depends on a microframework called Flask (link to Flask documentation) to run.

The JSON-Sending Method. There are global variables at the top of "server.py" file that correspond to variables from the "main.py" file in the "roverdomain" directory. They are global in order to prevent scope errors. All of the global server variables are compiled into a json called "data" that is sent in the "api_all()" method.

The "api_all()"method is what the client will call periodically in order to update the status of the rovers. This python server is responsible for sending the client a JSON file containing the variable values for a single instance of the rover domain. The client side is responsible for keeping a history log of past values and keeping track of the time the data was received.



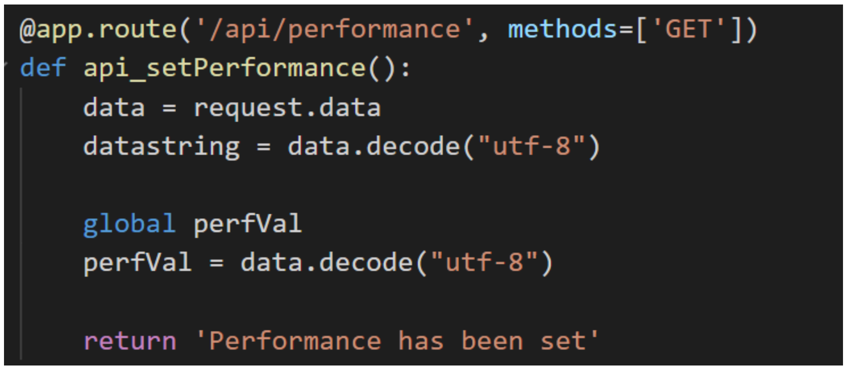
```python
@app.route('/api/data', methods=['GET'])
def api_all():
    print('Got data from client')
    global perfval
    global rewardType

    data = [
    {
        'performance': perfVal,
        'reward': rewardType,
        'generation': generation
    }
    ]

    return jsonify(data)
```

Figure 2. A snippet of code from the "server.py" file that compiles variables into a JSON to send to the client.

The Server-Updating Methods. The variables sent in the json are individually updated in their own functions with specific URL routes for each one. The python "main.py" file has server calls that use these variable setting methods to update the server. Additional variables may be added using this format as well.



```python
@app.route('/api/performance', methods=['GET'])
def api_setPerformance():
    data = request.data
    datastring = data.decode("utf-8")

    global perfVal
    perfVal = data.decode("utf-8")

    return 'Performance has been set'
```

Figure 3. A snippet of code from the "server.py" file that sets a server variable.

4.0 THE ROVER DOMAIN

The rover domain is a multi-rover exploration domain that uses neuro-evolutionary learning with a cooperative coevolutionary algorithm (CCEA) and reward functions. It features rover objects that are given points of interest (POIs) on a 2D plane. Each of these POIs are given a utility score, or reward value. When a rover enters the range of POI observation, denoted by a small circle, it is given a reward. A visual example of the rover domain is shown in the figure on the left.

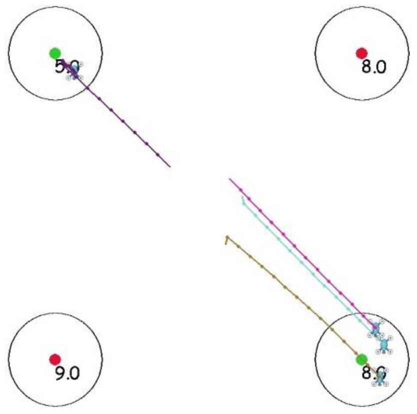
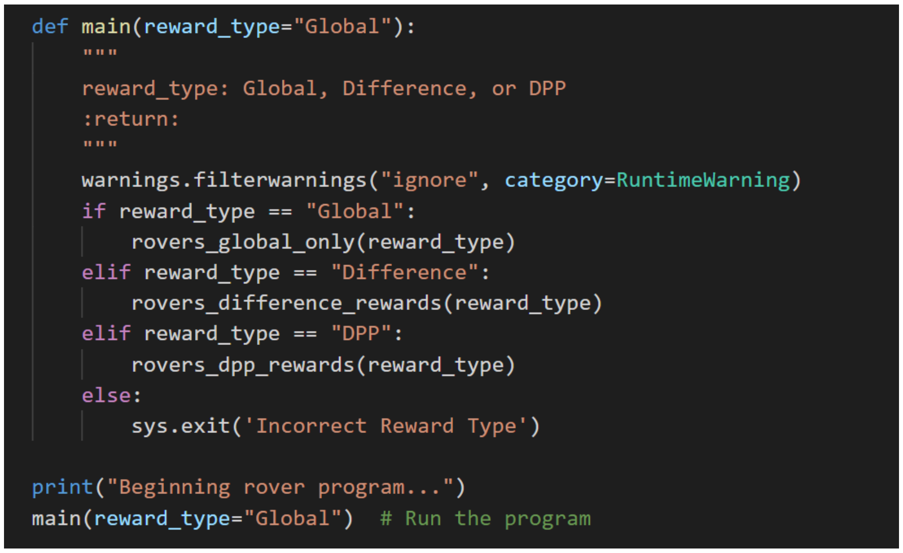


Figure 4. An image of the rover domain created by the rover domain's visualizer.

Reward Types. Alterations to the rover reward signal can be chosen in the "main.py" file, which serves as the rover domain's driver. The reward type can be manually changed in the main function shown below. There are three reward settings, global, difference, and DPP.

Global rewards based on the highest POI observed score, difference begins to consider rover's positions and performance relative to other rovers, and DPP takes counterfactuals into account. The differences in these reward types are further outlined in the "reward_functions.py" file.

Additional program parameters are located at the top of the "main.py" file and can be modified as well. These include general parameters, like number of rovers and POIs, as well as more complex parameters for the CCEA and neural network. Though the rover domain has many complexities, a deep understanding of its algorithms and neural networks is not required in order to utilize the rover domain interface.



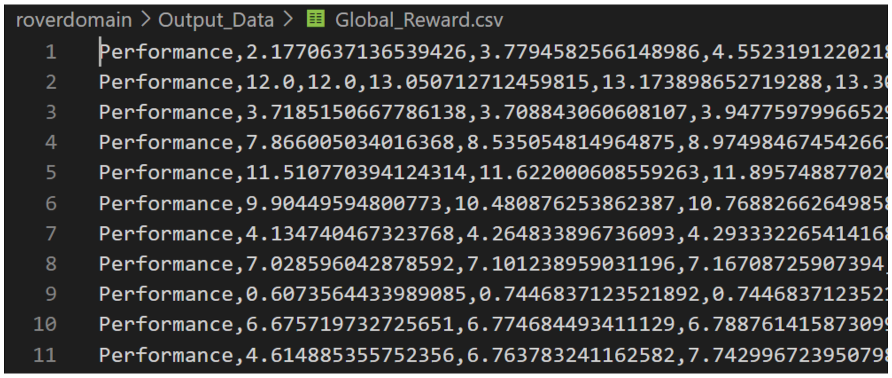
```python
def main(reward_type="Global"):
    """
    reward_type: Global, Difference, or DPP
    :return:
    """
    warnings.filterwarnings("ignore", category=RuntimeWarning)
    if reward_type == "Global":
        rovers_global_only(reward_type)
    elif reward_type == "Difference":
        rovers_difference_rewards(reward_type)
    elif reward_type == "DPP":
        rovers_dpp_rewards(reward_type)
    else:
        sys.exit('Incorrect Reward Type')

print("Beginning rover program...")
main(reward_type="Global")  # Run the program
```

Figure 5. A snippet of code from "main.py" in the rover domain that is responsible for setting the domain's reward type.

4.1 Output Data

The rover domain stores all its outputs in the “Output_Data” and “Screenshots” folders. The “Output_Data” folder contains files with performance history logs (based on achieved utility scores), rover paths, and POI choice. The “Screenshots” folder contains images created by the rover domain’s pygame-based visualizer. These images are created after every run of the program. An example of what a reward history log looks like after multiple runs is shown in the image below.

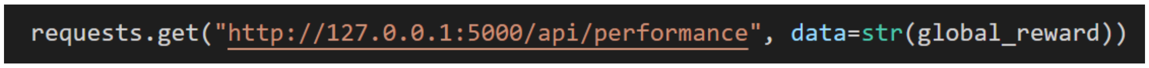


```
Performance,2.1770637136539426,3.7794582566148986,4.552319122021
Performance,12.0,12.0,13.050712712459815,13.173898652719288,13.3
Performance,3.7185150667786138,3.708843060608107,3.9477597996652
Performance,7.866005034016368,8.535054814964875,8.97498467454266
Performance,11.510770394124314,11.622000608559263,11.89574887702
Performance,9.90449594800773,10.480876253862387,10.7688266264985
Performance,4.134740467323768,4.264833896736093,4.2933322654141
Performance,7.028596042878592,7.101238959031196,7.16708725907394
Performance,0.6073564433989085,0.7446837123521892,0.744683712352
Performance,6.675719732725651,6.774684493411129,6.78876141587309
Performance,4.614885355752356,6.763783241162582,7.74299672395079
```

Figure 6. Performance data generated by the rover domain.

4.2 Server Calls

Server calls are used throughout the “main.py” file to update the server when variables change. The “requests” library is used to simplify the server call process and reduce code. An example of a server call to update performance/reward values is shown below. The data input for the server call should be type cast to a string or else errors may occur.



```
requests.get("http://127.0.0.1:5000/api/performance", data=str(global_reward))
```

Figure 7. A requests call from "main.py" that updates the server's global reward value.

5.0 OPENMCT

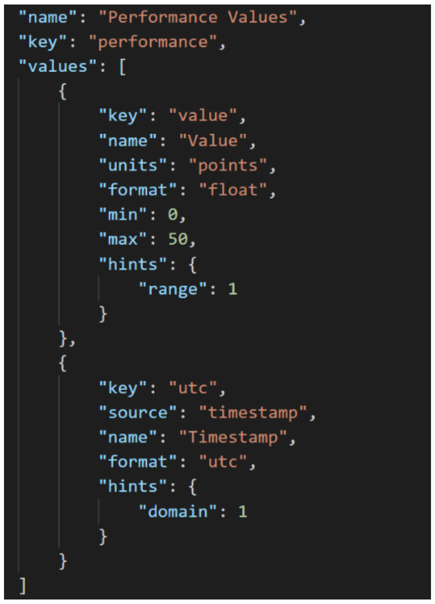
OpenMCT (Open Mission Control Technologies) is a web-based data visualization framework. It was developed by NASA’s Ames Research Center as a modern, next-generation alternative to traditional mission control software. The base code for OpenMCT (Link to base OpenMCT code) can be found on their GitHub. The rover domain interface uses the OpenMCT plugin tutorial (Link to OpenMCT tutorial code) as a base because it provides history and real time plugins to use in our build. Plugins are bundles of code that implement functionalities into an OpenMCT based interface. OpenMCT’s plugin tutorial explains the basics of creating a new telemetry source and is a useful resource to reference for future plugin additions.

5.1 The Rover Plugin

There are three files that are responsible for our rover plugin: “rover-data.json”, “rover-plugin.js”, and under the example-server folder, the “rovers.js” file.

The Rover Data File. The “rover-data.json” file contains a list of the attributes or telemetry points our rover has. It does not include the actual data points (i.e. the number values to be presented on a graph, or variable values), rather, it defines the details of how OpenMCT should interpret those data points.

This includes things like what the data point’s name shows up as on the user interface, what units the data points are in, and the range of values that are displayed when graphed. More information on how telemetry metadata is used can be found in OpenMCT’s API Documentation (Link to OpenMCT’s documentation page) under “Telemetry API”. A snippet of code from “rover-data.json” can be seen on the right.



```
"name": "Performance Values",
"key": "performance",
"values": [
    {
        "key": "value",
        "name": "Value",
        "units": "points",
        "format": "float",
        "min": 0,
        "max": 50,
        "hints": {
            "range": 1
        }
    },
    {
        "key": "utc",
        "source": "timestamp",
        "name": "Timestamp",
        "format": "utc",
        "hints": {
            "domain": 1
        }
    }
]
```

Figure 8. A snippet of code from "rover-data.json".

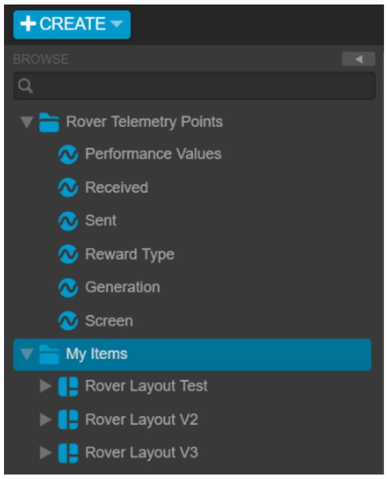


Figure 9. OpenMCT's navigation bar.

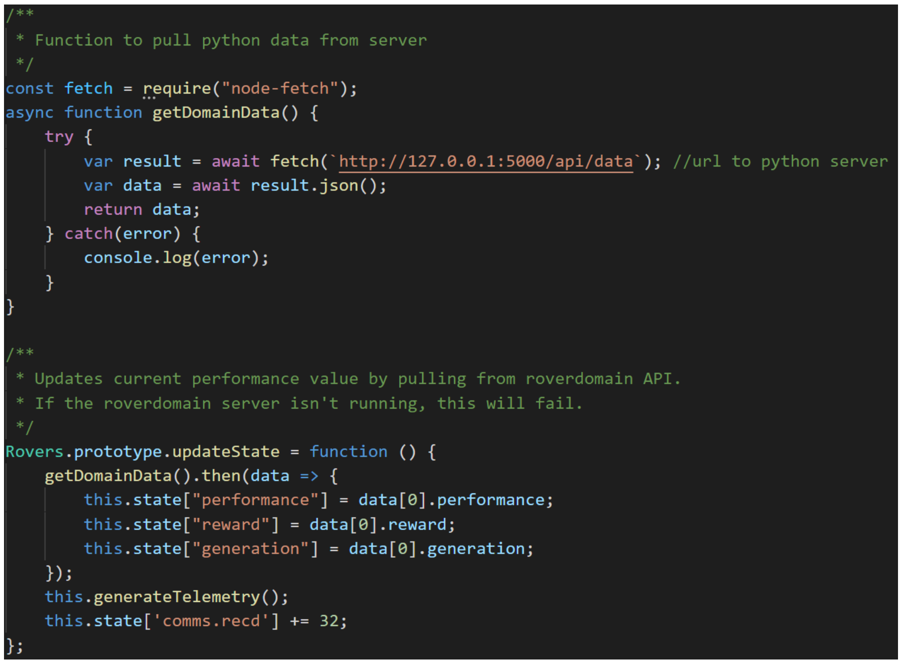
The Rover Plugin File. The “rover-plugin.js” file defines the rover object, deals with domain object creation, and determines the folder hierarchy (i.e. root objects and subfolders). More information on this can be found in OpenMCT’s API Documentation under “Domain Objects and Identifiers”.

When the interface is running, the folder hierarchy determines what the navigation bar looks like. An example of the interface’s folder navigation bar can be seen on the left.

The Client-Side Rover File. The “rovers.js” file, which is found in the “example-server” folder, is the client-side representation of our rovers. This file defines the rover object state, and constantly fetches data from the python server.

Inside the “rovers.js” file, the “updateState()” and “getDomainData()” functions are where the code interacts with the python server. If additional variables were added, the “updateState()” function would have to be modified to reflect that.

The “updateState()” function updates the rover system, the “getDomainData()” function returns a JSON with rover data from the python server, and the “generateTelemetry()” function is what lets OpenMCT know that changes have been made, finalizes those changes, and sends them to be displayed on the interface.



```
/**
 * Function to pull python data from server
 */
const fetch = require("node-fetch");
async function getDomainData() {
    try {
        var result = await fetch(`http://127.0.0.1:5000/api/data`); //url to python server
        var data = await result.json();
        return data;
    } catch(error) {
        console.log(error);
    }
}

/**
 * Updates current performance value by pulling from roverdomain API.
 * If the roverdomain server isn't running, this will fail.
 */
Rovers.prototype.updateState = function () {
    getDomainData().then(data => {
        this.state["performance"] = data[0].performance;
        this.state["reward"] = data[0].reward;
        this.state["generation"] = data[0].generation;
    });
    this.generateTelemetry();
    this.state['comms.recd'] += 32;
};
```

Figure 10. A snippet of code that shows some asynchronous functions in "rovers.js".

5.2 Creating an Interface Layout

Once you have established telemetry points, you can create new interface layouts by using the “create” button once you have started running the OpenMCT application on your browser. These layouts are initially blank but can be easily filled with telemetry by applying data points to interface objects.

An example of what a blank layout looks like can be seen below.

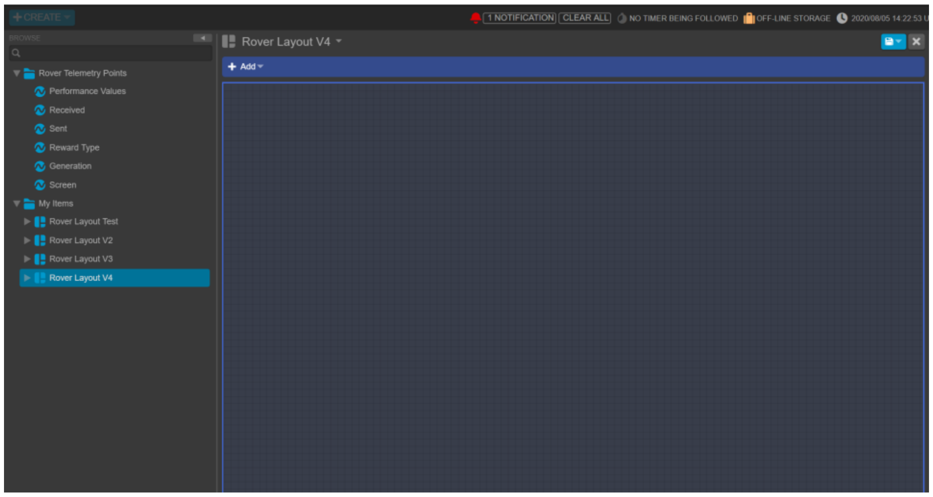
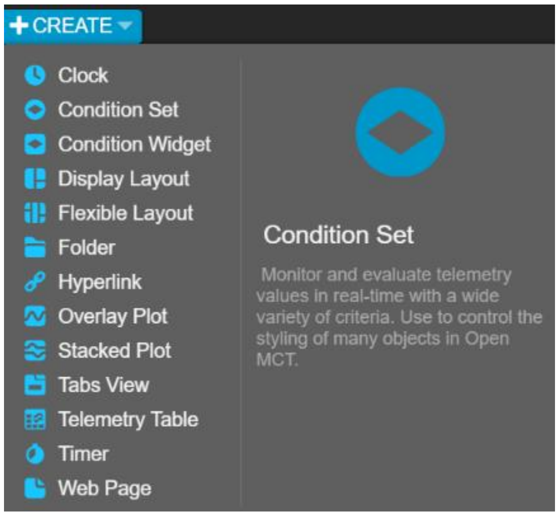


Figure 11. A blank OpenMCT layout.



Monitor and evaluate telemetry values in real-time with a wide variety of criteria. Use to control the styling of many objects in Open MCT.

Figure 12. The "create" drop down menu.

New Objects. New objects can be created by clicking the create button on the top left; all new objects are automatically saved into the "my items" folder. The "my items" folder is a domain object that allows you to persist, or save, any object you create. However, these saved objects are saved locally and do not carry over when another person uses your interface code.

Once you create an object, you can select what telemetry you want to showcase in it with the options screen. Alternatively, you may also drag telemetry points from the folder bar on the left straight onto your object.

Many actions in OpenMCT, like editing object positions, adding telemetry data to blank overlay/stacked plots, and moving objects into different folders, can be accomplished through dragging and dropping with your mouse.

Saving Layouts. In order to permanently save a created layout or object as a part of your interface, you will have to export that object as a JSON. This can be done by opening the pull-down menu next to the object title.

Once the JSON is downloaded, you will need to create a plugin to install it. An example of this can be seen in the OpenMCT demo files (link to OpenMCT demo GitHub).

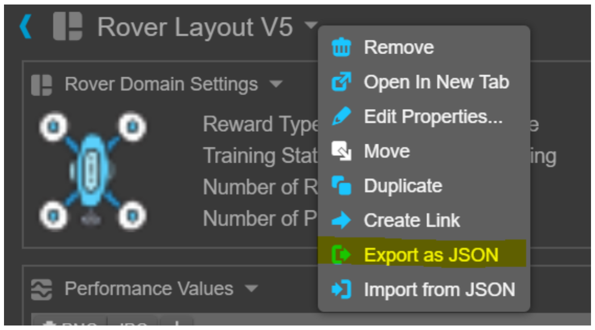
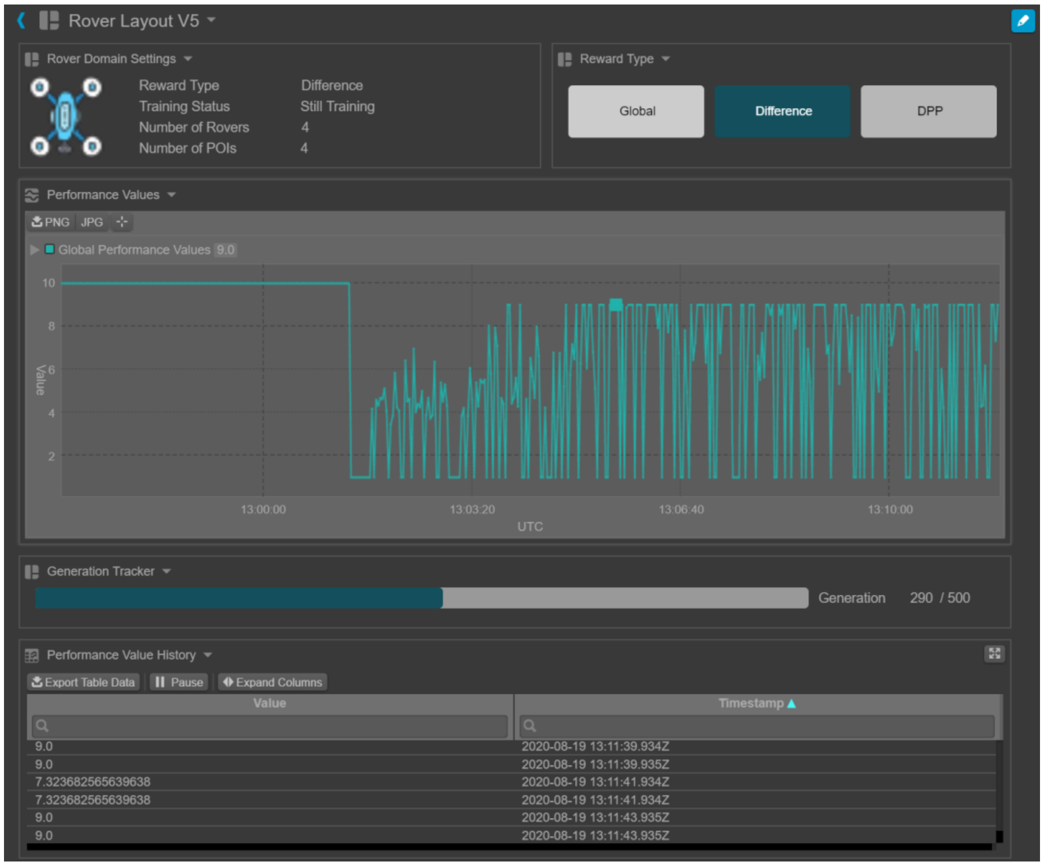


Figure 13. The "Export as JSON" option.

An example layout for the rover domain can be found further down in the interface control document. The JSON for this layout can be found in the "rover-layout-example.json" file.

Layout Tips. It is recommended to group components of your interface into small layouts that you then display in one larger layout. This makes it easier to move and manage. Some examples of this can be seen in the example rover domain interface: the reward type buttons, the generation tracker, and the rover domain settings are all contained in their own layouts.

Another useful feature that can be seen in the rover layout is the use of conditional widgets, conditional styling, and condition sets, which can be seen in the example interface's reward type buttons and generation tracker progress bar.



Reward Type	Difference
Training Status	Still Training
Number of Rovers	4
Number of POIs	4

Generation 290 / 500

Value	Timestamp
9.0	2020-08-19 13:11:39.934Z
9.0	2020-08-19 13:11:39.935Z
7.323682565639638	2020-08-19 13:11:41.934Z
7.323682565639638	2020-08-19 13:11:41.934Z
9.0	2020-08-19 13:11:43.935Z
9.0	2020-08-19 13:11:43.935Z

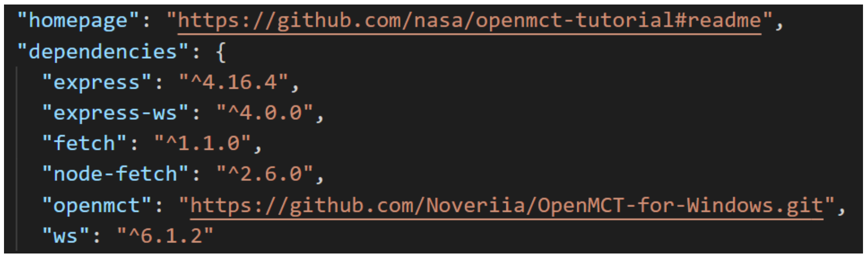
Figure 14. An example OpenMCT interface.

5.3 Known Errors and Dependencies

If you are running OpenMCT on Windows, you may encounter a problem with installing dependencies for the OpenMCT tutorial due to some windows incompatibilities with OpenMCT's Sass file loader. An example of what the error message will read can be seen below.

```
ERROR in ./src/plugins/themes/espresso-theme.scss
Module build failed (from ./node_modules/mini-css-extract-plugin/dist/loader.js):
ModuleBuildError: Module build failed (from ./node_modules/fast-sass-loader/lib/index.js):
Error: import file cannot be resolved: "@import "~styles/vendor/normalize-min";" @C:\code\
    at Object.importReplacer (C:\code\openmct\node_modules\fast-sass-loader\lib\index.js:2
    at importReplacer.throw (<anonymous>)
    at onRejected (C:\code\openmct\node_modules\co\index.js:81:24)
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:94:5)
    at C:\code\openmct\node_modules\webpack\lib\NormalModule.js:316:20
    at C:\code\openmct\node_modules\loader-runner\lib\LoaderRunner.js:367:11
    at C:\code\openmct\node_modules\loader-runner\lib\LoaderRunner.js:233:18
    at context.callback (C:\code\openmct\node_modules\loader-runner\lib\LoaderRunner.js:11
    at C:\code\openmct\node_modules\fast-sass-loader\lib\index.js:311:5
    at runMicrotasks (<anonymous>)
    at processTicksAndRejections (internal/process/task_queues.js:94:5)
```

In the rover interface code, under the “package.json” file, I used my own modified version of OpenMCT as a dependency instead of the actual base OpenMCT code to remedy this issue. A screenshot of the “package.json” file can be seen in the following image.

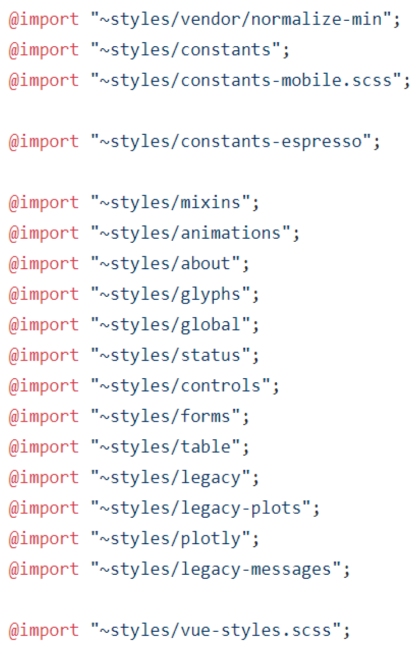


```
"homepage": "https://github.com/nasa/openmct-tutorial#readme",
"dependencies": {
  "express": "^4.16.4",
  "express-ws": "^4.0.0",
  "fetch": "^1.1.0",
  "node-fetch": "^2.6.0",
  "openmct": "https://github.com/Noveriia/OpenMCT-for-Windows.git",
  "ws": "^6.1.2"
```

Figure 15. OpenMCT dependency location in "package.json".

However, this means that my version of OpenMCT is bound to become outdated. In the case that you are using a non-Windows operating system, then the openmct dependency URL can be swapped for the base OpenMCT URL (Link to base OpenMCT code). In the case that you are using a Windows operating system, this issue has not been fixed, and you may have to upload a personal, modified version of the base OpenMCT repository to use a dependency.

The modification you will have to make is to all of the “.scss” files in the “themes” folder. The path to the “themes” folder is as follows: openmct > src > plugins > themes. The original files will look like the image on the left, the modified files will look like the image on the right.

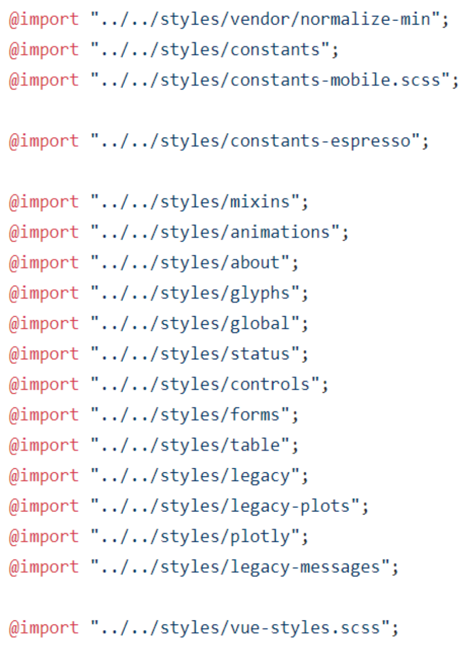


```
@import "~styles/vendor/normalize-min";
@import "~styles/constants";
@import "~styles/constants-mobile.scss";

@import "~styles/constants-espresso";

@import "~styles/mixins";
@import "~styles/animations";
@import "~styles/about";
@import "~styles/glyphs";
@import "~styles/global";
@import "~styles/status";
@import "~styles/controls";
@import "~styles/forms";
@import "~styles/table";
@import "~styles/legacy";
@import "~styles/legacy-plots";
@import "~styles/plotly";
@import "~styles/legacy-messages";

@import "~styles/vue-styles.scss";
```



```
@import "../../styles/vendor/normalize-min";
@import "../../styles/constants";
@import "../../styles/constants-mobile.scss";

@import "../../styles/constants-espresso";

@import "../../styles/mixins";
@import "../../styles/animations";
@import "../../styles/about";
@import "../../styles/glyphs";
@import "../../styles/global";
@import "../../styles/status";
@import "../../styles/controls";
@import "../../styles/forms";
@import "../../styles/table";
@import "../../styles/legacy";
@import "../../styles/legacy-plots";
@import "../../styles/plotly";
@import "../../styles/legacy-messages";

@import "../../styles/vue-styles.scss";
```

Figure 16. Code changes for OpenMCT .scss error.

6.0 DEFINITIONS, ABBREVIATIONS AND ACRONYMS

2D: Two Dimensional

AFRL: Air Force Research Laboratory

API: Application Programming Interface

CCEA: Cooperative Coevolutionary Algorithm

ICD: Interface Control Document

OpenMCT: Open Mission Control Technologies

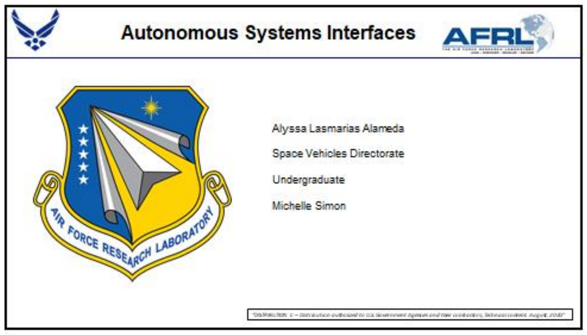
POI: Point of Interest

REST: Representational State Transfer

USRA: University Space Research Association

VS Code: Visual Studio Code

7.0 APPENDIX A - PRESENTATIONS



Autonomous Systems Interfaces

Alyssa Lasmarias Alameda

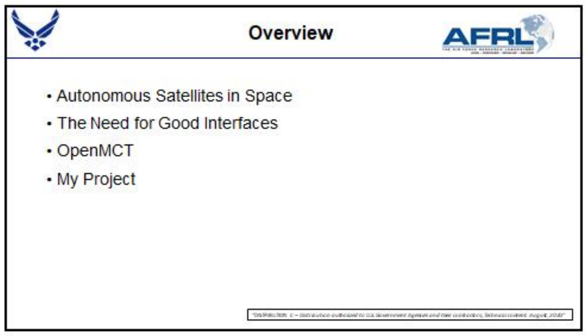
Space Vehicles Directorate

Undergraduate

Michelle Simon

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August 2018"

1

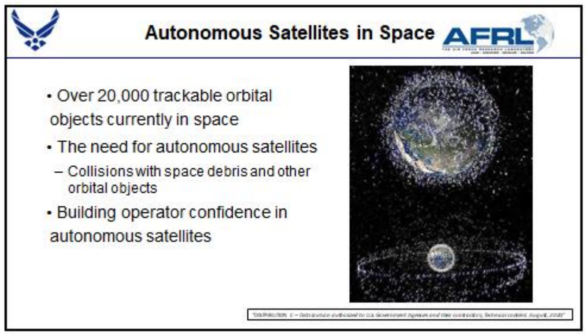


Overview

- Autonomous Satellites in Space
- The Need for Good Interfaces
- OpenMCT
- My Project

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August 2018"

2



Autonomous Satellites in Space

- Over 20,000 trackable orbital objects currently in space
- The need for autonomous satellites
  - Collisions with space debris and other orbital objects
- Building operator confidence in autonomous satellites

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August 2018"

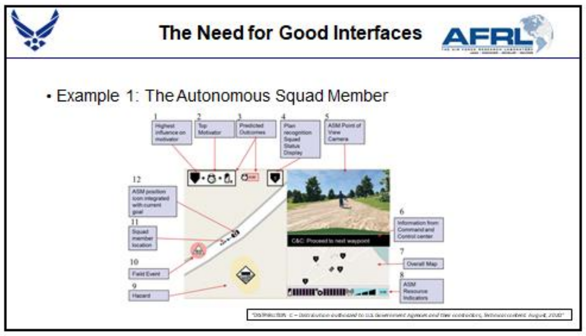
3

The Need for Good Interfaces

- Teaming with autonomous agents
- Agent transparency and motivation
- Interfaces as a vehicle for human-agent communication

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August 2018"

4



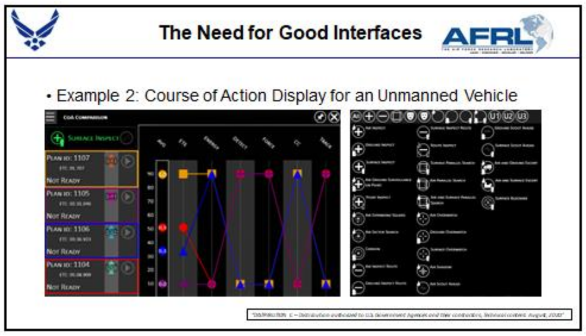
The Need for Good Interfaces

- Example 1: The Autonomous Squad Member

1 Highest influence on motivator
2 Top Motivator
3 Predicted Outcomes
4 Plan recognition Squad Status Display
5 ASM Point of View Camera
6 Information from Command and Control center
7 Overall Map
8 ASM Resource Indicators
9 Hazard
10 Field Event
11 Squad member location
12 ASM position icon integrated with current goal
C&C: Proceed to next waypoint

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August 2018"

5



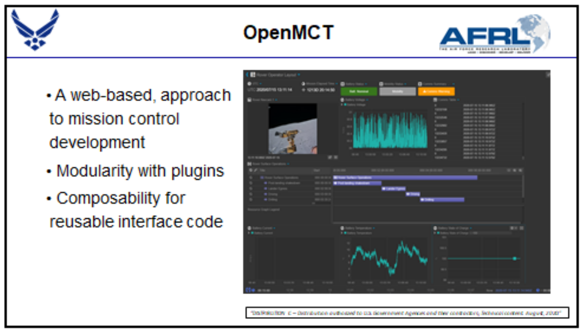
The Need for Good Interfaces

- Example 2: Course of Action Display for an Unmanned Vehicle

COA COMPARISON
SURFACE INSPECT
PLAN ID: 1107
NOT READY
PLAN ID: 1105
NOT READY
PLAN ID: 1106
NOT READY
PLAN ID: 1104
NOT READY

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August 2018"

6

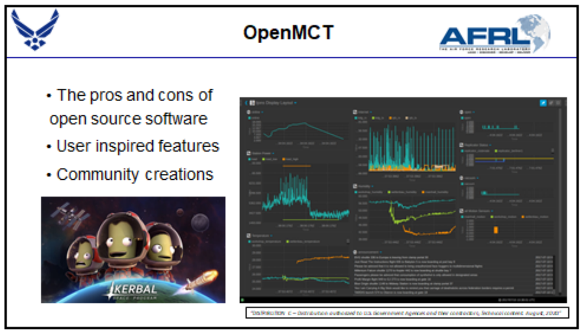


OpenMCT

- A web-based, approach to mission control development
- Modularity with plugins
- Composability for reusable interface code

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

7

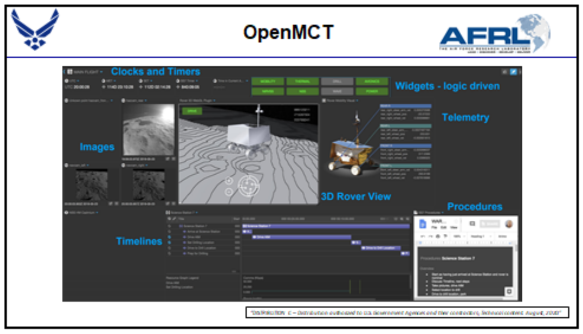


OpenMCT

- The pros and cons of open source software
- User inspired features
- Community creations

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

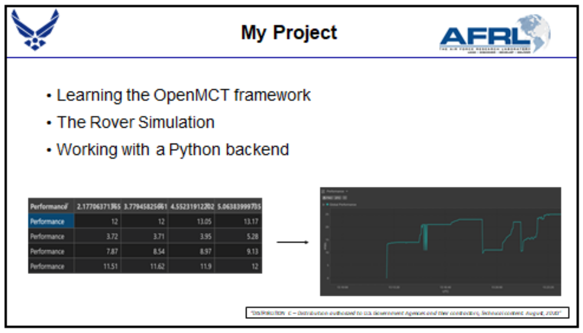
8



OpenMCT

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

9



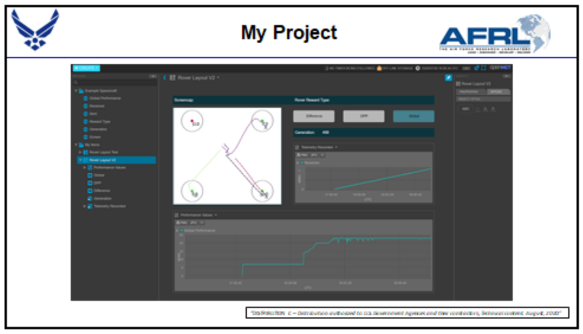
My Project

- Learning the OpenMCT framework
- The Rover Simulation
- Working with a Python backend

Performance	2.17706371365	3.77945825461	4.55231912202	5.06383999705
Performance	12	12	13.05	13.17
Performance	3.72	3.71	3.95	5.28
Performance	7.87	8.54	8.97	9.13
Performance	11.51	11.62	11.9	12

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

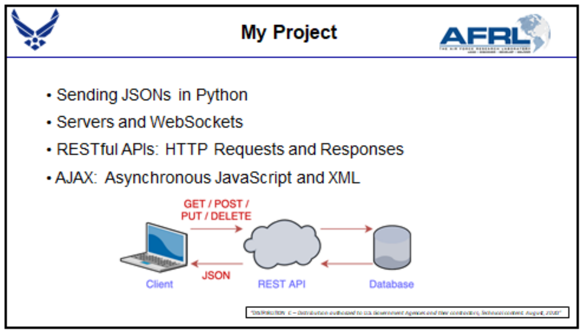
10



My Project

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

11

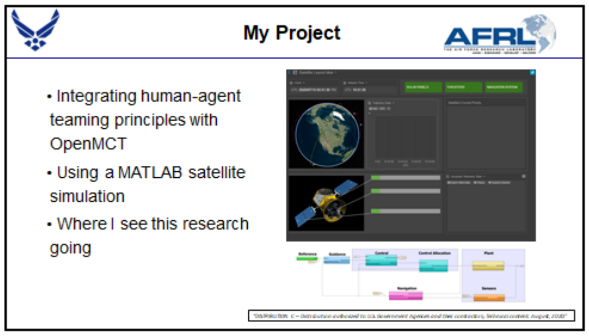


My Project

- Sending JSONs in Python
- Servers and WebSockets
- RESTful APIs: HTTP Requests and Responses
- AJAX: Asynchronous JavaScript and XML

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

12

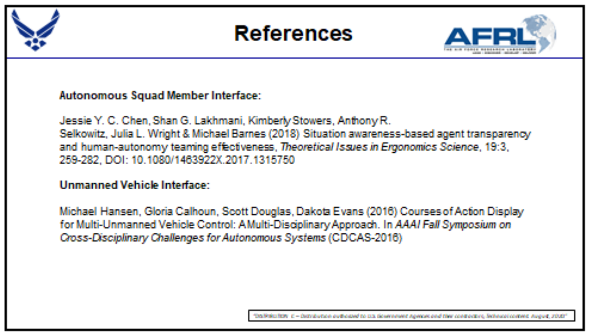


My Project

- Integrating human-agent teaming principles with OpenMCT
- Using a MATLAB satellite simulation
- Where I see this research going

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

13



References

Autonomous Squad Member Interface:

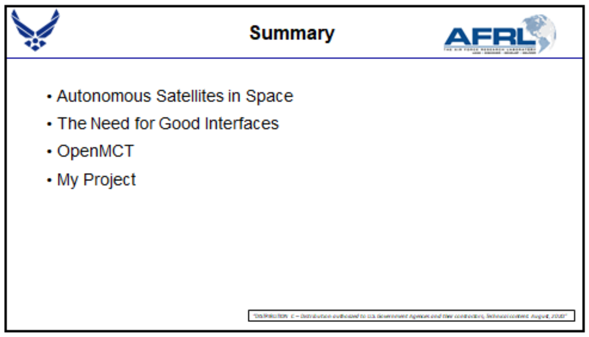
Jessie Y. C. Chen, Shan G. Lakhmani, Kimberly Stowers, Anthony R. Selkowitz, Julia L. Wright & Michael Barnes (2018) Situation awareness-based agent transparency and human-autonomy teaming effectiveness, *Theoretical Issues in Ergonomics Science*, 19:3, 259-282, DOI: 10.1080/1463922X.2017.1315750

Unmanned Vehicle Interface:

Michael Hansen, Gloria Calhoun, Scott Douglas, Dakota Evans (2016) Courses of Action Display for Multi-Unmanned Vehicle Control: A Multi-Disciplinary Approach. In *AAAI Fall Symposium on Cross-Disciplinary Challenges for Autonomous Systems* (CDCAS-2016)

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

16

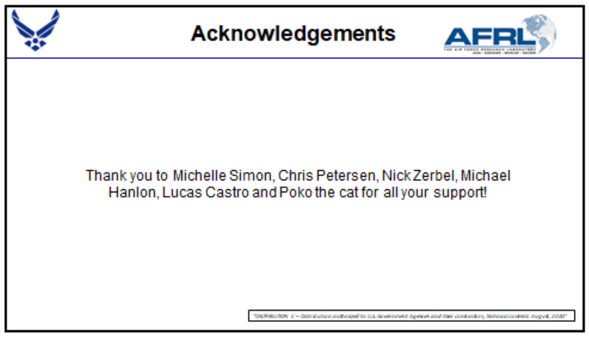


Summary

- Autonomous Satellites in Space
- The Need for Good Interfaces
- OpenMCT
- My Project

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

14

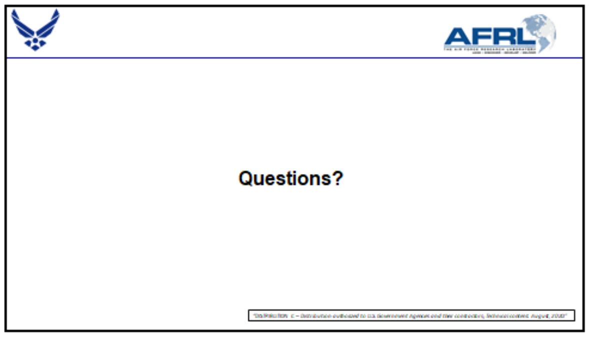


Acknowledgements

Thank you to Michelle Simon, Chris Petersen, Nick Zerbel, Michael Hanlon, Lucas Castro and Poko the cat for all your support!

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

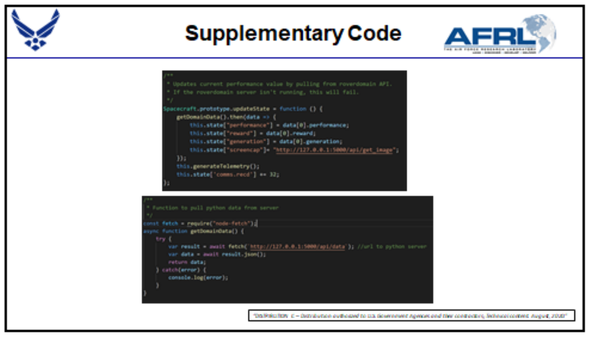
17



Questions?

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

15



Supplementary Code

```
/**
 * Updates current performance value by pulling from roverdomain API.
 * If the roverdomain server isn't running, this will fail.
 */
Spacecraft.prototype.updateState = function () {
    getDomainData().then(data => {
        this.state["performance"] = data[0].performance;
        this.state["reward"] = data[0].reward;
        this.state["generation"] = data[0].generation;
        this.state["screencap"]= "http://127.0.0.1:5000/api/get_image";
    });
    this.generateTelemetry();
    this.state['comms.recd'] += 32;
};
```

```
/**
 * Function to pull python data from server
 */
const fetch = require("node-fetch");
async function getDomainData() {
    try {
        var result = await fetch('http://127.0.0.1:5000/api/data'); //url to python server
        var data = await result.json();
        return data;
    } catch(error) {
        console.log(error);
    }
}
```

"DISTRIBUTION C – Distribution authorized to U.S. Government Agencies and their contractors, Technical content, August, 2020"

18