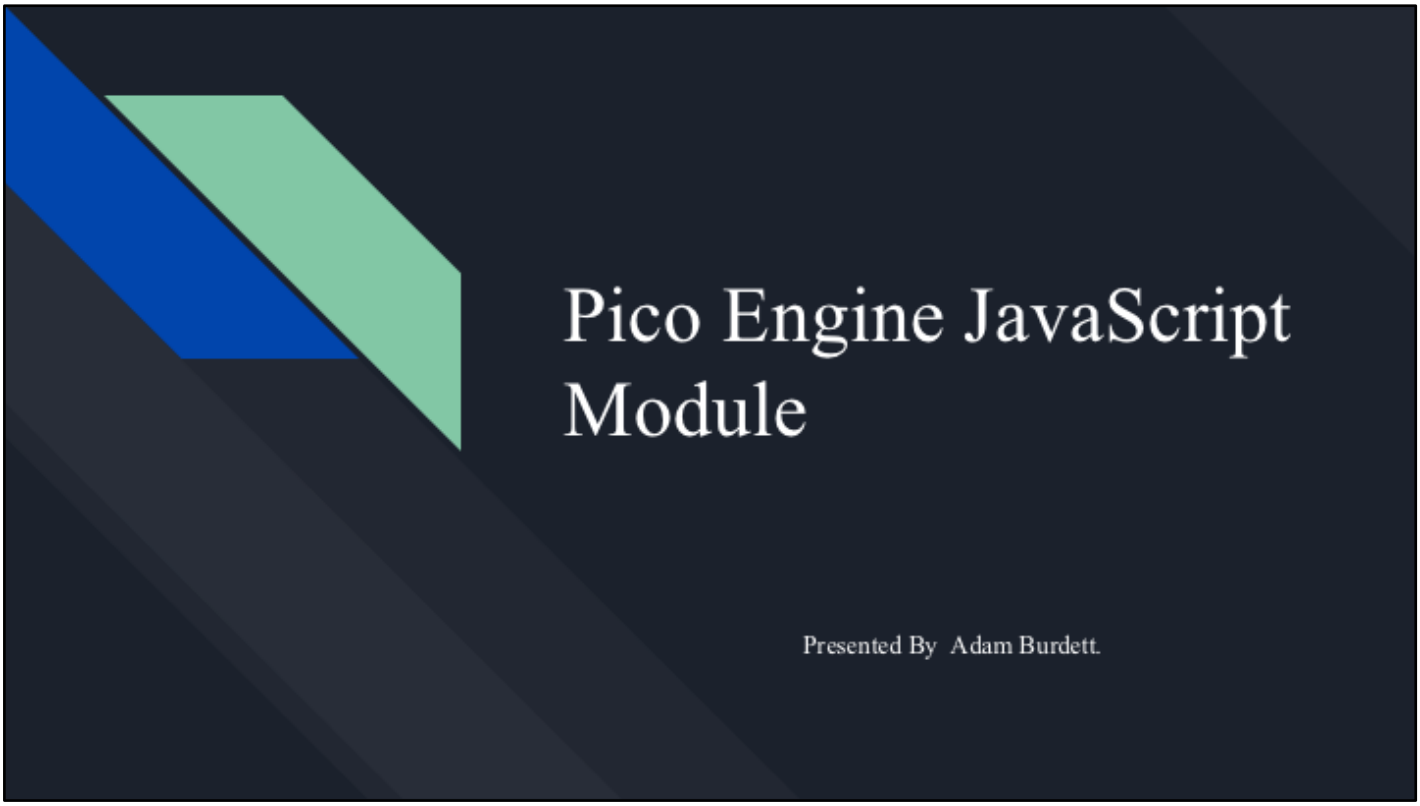


Hi, My name is Adam Burdett and Im presenting my research into Pico Engine javascript modules.

Pico Engine

The Pico Engine is an emerging IoT platform that evaluates KRL rules against events associated with Pico device shadows.

KRL Example

```
rule hello_world {
  select when echo hello
  send_directive("say", {"something": "Hello World"})
}
```

This is the Hello world KRL rule. This rule selects on a echo hello event and sends a directive called say with an object containing the string hello world.

JavaScript Modules

```
rule hello_world {
  select when echo hello
  send_directive("say", {"something": random:word()})
}
```

The Pico Engine provides a standard library in the form of KRL modules. For example we have just changed the hell world rule to return a directive with a random word instead of a hello world string. This is done by calling an engine module called random with a function called word. But what if you have an amazing idea for a killer app which requires a module the engine doesn't provide?
For example what if your killer app needs cowsay,

JavaScript Modules

```
rule hello_world {
  select when echo hello
  send_directive("say", {"something": cowsay:say("Pico")})
}
```

You would like a module called cowsay with a function called say. This function would take a string and return…

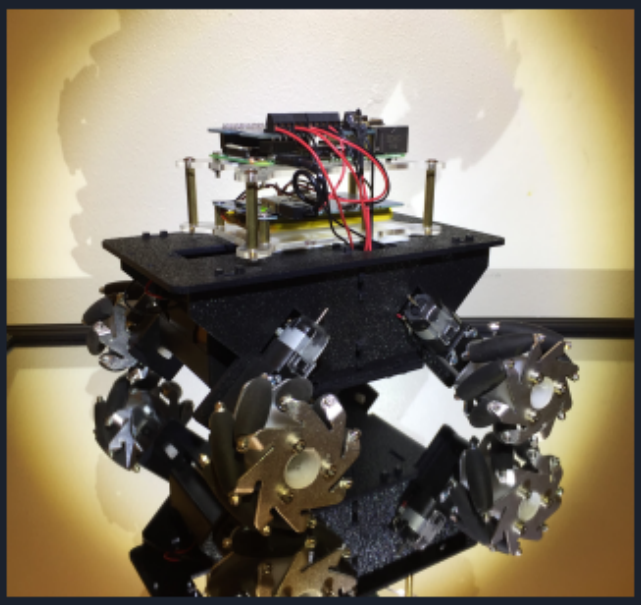
JavaScript Modules

```
rule hello_world {
  select when echo hello
  send_directive("say", {"something": cowsay:say("Pico")})
}
```

```
 ______
< PICO >
 ------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

The Cowsay of that string. My research was done by investigating the engines potential to provide users the ability to create custom javaScript Modules.

It was believed the Pico engine had the potential to extend KRL with custom JavaScript modules, but was never developed until now. This research developed user-defined JavaScript modules for the Pico engine and proposed and tested a single Pico to a single resource model.



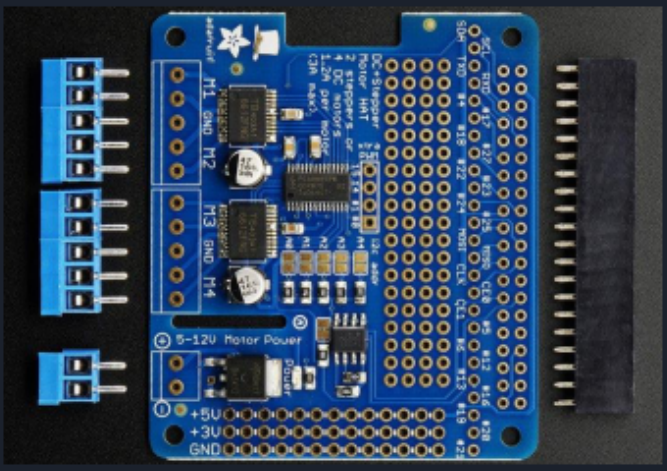
Pico Rover

To drive this research I proposed developing a pico rover. The idea is to extend KRL language to natively manage hardware on the raspberry pi. I am going to cover the main hardware of the pico rover.

Raspberry Pi

The pico rover is controlled by a raspberry pi, which is a small computer that runs linux. This is ideal because the pico engine is built on top of node.js, which the raspberry pi supports.

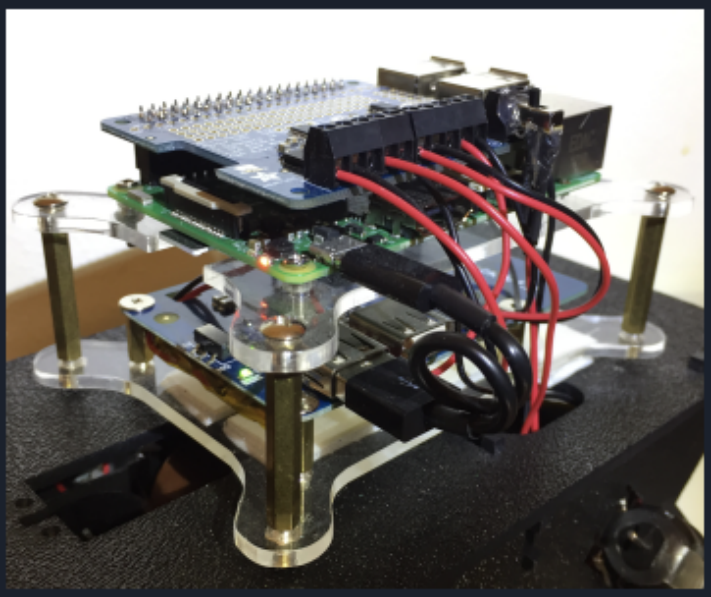
Motor Controller



This is an adafruit motor controller. I specifically picked this controller because I was able to find a node.js library that supports it. I will explain later in the slides why that was important.

Mecanum Wheels

Go Big or Go Home! Right? This is a mecanum wheel. It has a free floating roller that sits on a 45 degree angle, which allows the pico rover to move in any direction.
I chose this so it would help strain test my model.



Pico Rover

I will now go into details about the software.

Extending KRL

```
// random.js
var randomWords = require("random-words");
module.exports = function(core){
    return {
        def: { //...
            word: mkKRLfn([
            ], function(ctx, args, callback){
                callback(null, randomWords());
            })
            , //...
        }
    };
};
```

Remember back to the random word example. I was able to find in the engine where that module is declared.
Random module lives in random.js. In there I found require "random words" being called.
This is important because that is a node js library.

Extending KRL

```
// random.js
var randomWords = require("random-words");
module.exports = function(core){
    return {
        def: { //...
            word: mkKRLfn([
            ], function(ctx, args, callback){
                callback(null, randomWords());
            })
            , //...
        }
    };
};
```

I also recognize module.exports. This is how node.js exports module objects.

Extending KRL

```
// random.js
var randomWords = require("random-words");
module.exports = function(core){
    return {
        def: { //...
            word: mkKRLfn([
            ], function(ctx, args, callback){
                callback(null, randomWords());
            })
            , //...
        }
    };
};
```

If you look at where random words library is being called, you will realize that it is simply being wrapped inside the exported object.
This is when I realized all that needs to be done is adding a configuration for a developer to add a js file and describe exactly what the engine is expecting to see in that exported object.
I made these feature suggestions to the main engine developer and a new module pattern was developed.

Extending KRL

```
// motorHat.js
var motorHat = require('motor-hat')(spec);
module.exports = {...
    dc_run: {
        type: "action",
        args: ["index","direction"],
        fn: function(args, callback){
            motorHat.dcs[args.index].run(args.direction);
            callback();
        },
    },...
};
```

This is code taken from the motorHat.js library this research produced. Remember I picked a motor driver I found a node.js module for.

Extending KRL

```
// motorHat.js
var motorHat = require('motor-hat')(spec)
module.exports = {...
    dc_run: {
        type: "action",
        args: ["index","direction"],
        fn: function(args, callback){
            motorHat.dcs[args.index].run(args.direction);
            callback();
        },
    },...
};
```

This is where I called it. Being able to use that library prevented me from rewriting all the low level code.
NPM is a package manager used for node.js. The pico engine is build on node js, which makes all the libraries of npm easily added as modules in the engine.

Extending KRL

```
// motorHat.js
var motorHat = require('motor-hat')(spec)
module.exports = {...
    dc_run: {
        type: "action",
        args: ["index","direction"],
        fn: function(args, callback){
            motorHat.dcs[args.index].run(args.direction);
            callback();
        },
    },...
};
```

Notice motorHat.js exports a node module object just as before. But this object is declared a little differently.
I declare the name of the modules function, dc_run and provide configuration that the engine is expecting.
The important part is to note that we call the motor-hat library and wrap it in the object that's exported.

MotorHat Modules

```
rule hello_world {
  select when echo hello
  motorHat:dc_run(1,"fwd")
}
```

This is the same helo world rule from earlier. but the directive has been replaced with my new dc_run function provided by the motorHat module. This would start moving the second wheel forward when a echo hello event is raised.

Documentation

developer

+ Pages

- Pico Engine Quickstart
- Pico Programming Lessons
- KRL Manual
  - Overview
  - Rulesets
  - Event Expressions
  - KRL Expression Langua
  - Libraries
  - Persistent Variables
  - User Defined Modules
    - JavaScript Module
  - Grammar
- Wrangler Operating System
- KRL Cookbook
- How-to articles
- Event Exchange Protocol

developer / ... / User Defined Modules

JavaScript Modules

Adam Burdett
Last modified Dec 21, 2017

JavaScript Modules allow you to extend KRL language within a pico engine. They are often used to take advantage of platform specific features. For example, extending KRL to natively manage GPIO pins of a raspberry pi hosting a pico engine.

JavaScript Modules can also be used to create pico engine singletons. While every pico should be treated as its own threaded object, restricted resources of an engine should be shared across picos as a singleton. For example, a raspberry pi only has one set of gpio pins. The access to the gpio pins can be encapsulated inside a JavaScript module as a singleton to be used by any pico. Another use could be websocket singleton which any pico could use to send events on.

The following sections describe how JavaScript Modules work:

- Defining a JavaScript Module
- Package.json Example
- JavaScript Module Example
- Module Instantiation Example
- Using Npm Inside A JavaScript Module
- MotorHat Module Instantiation Example
- MotorHat Library Configuration
- MotorHat Module Methods
- Conclusion

Defining a JavaScript Module

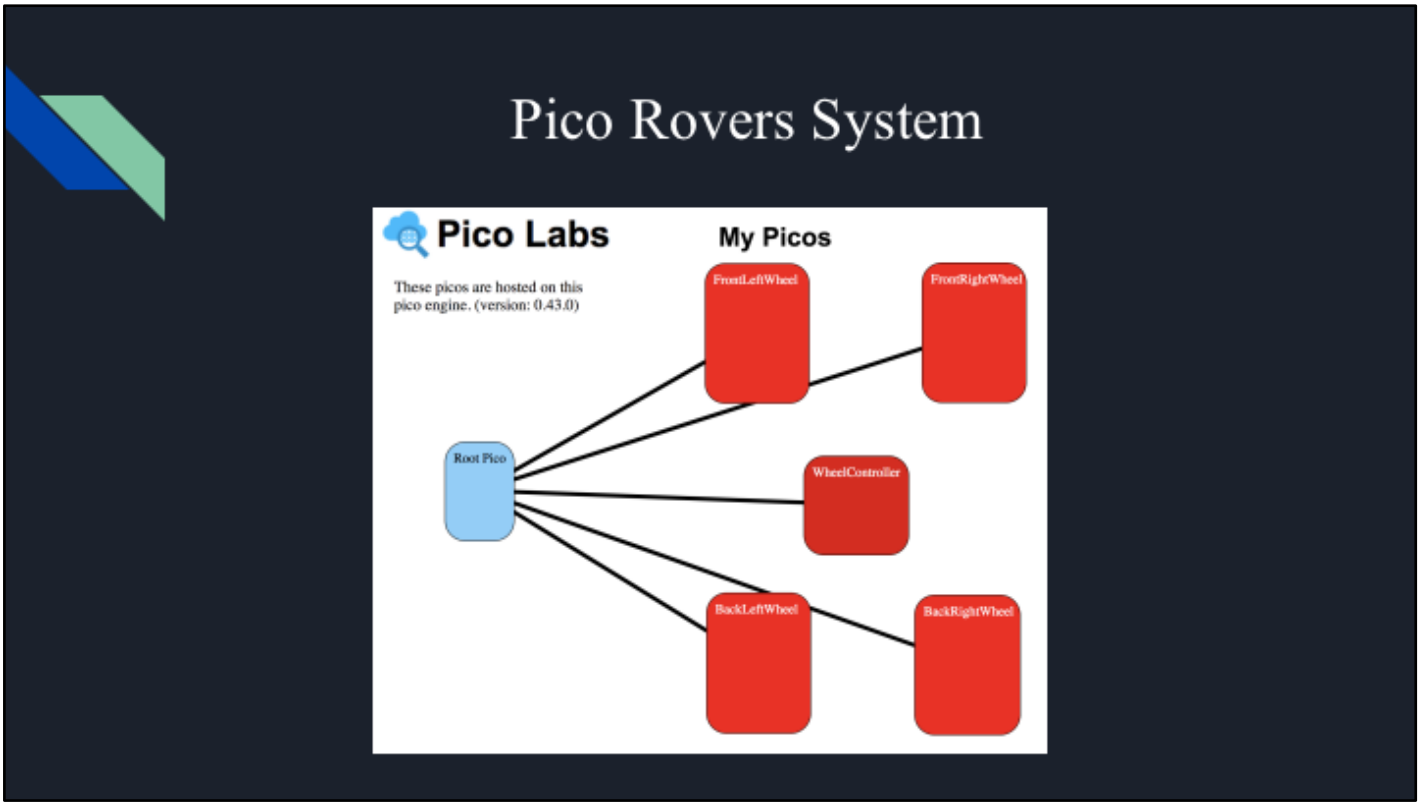
To create a custom JavaScript Module you must start with a new node project (example).

Currently the pico engine only supports defining these modules in the package.json file of a node project with the pico engine as a dependency. In the future, the engine may use environment variables to define modules, but at the time of writing this, does not.

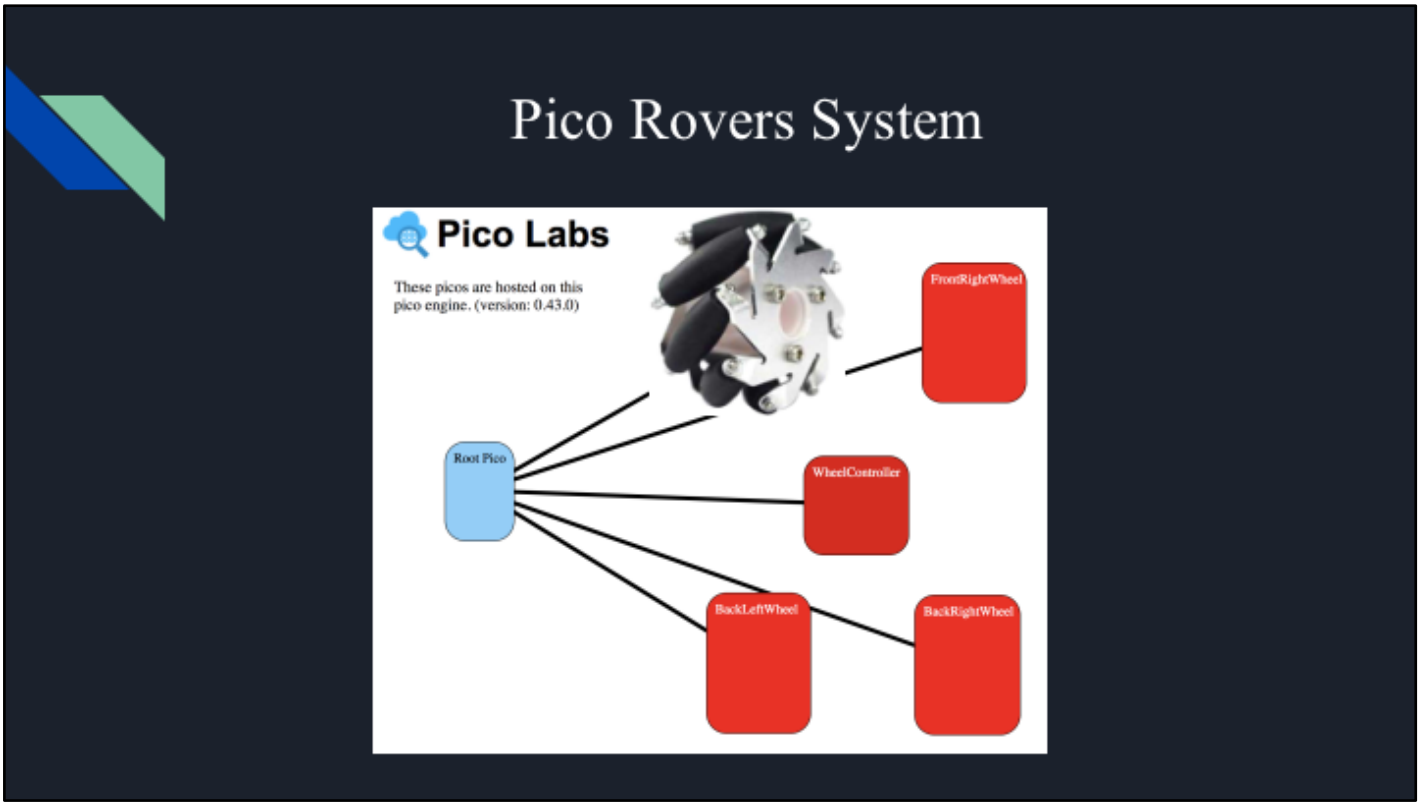
The package.json needs to lists the pico engine as a dependency (this will cause pico engine to be installed with the `npm install` command) as well as provide configurations for the pico engine.

https://picolabs.atlassian.net/wiki/spaces/docs/pages/96370693/JavaScript+Modules

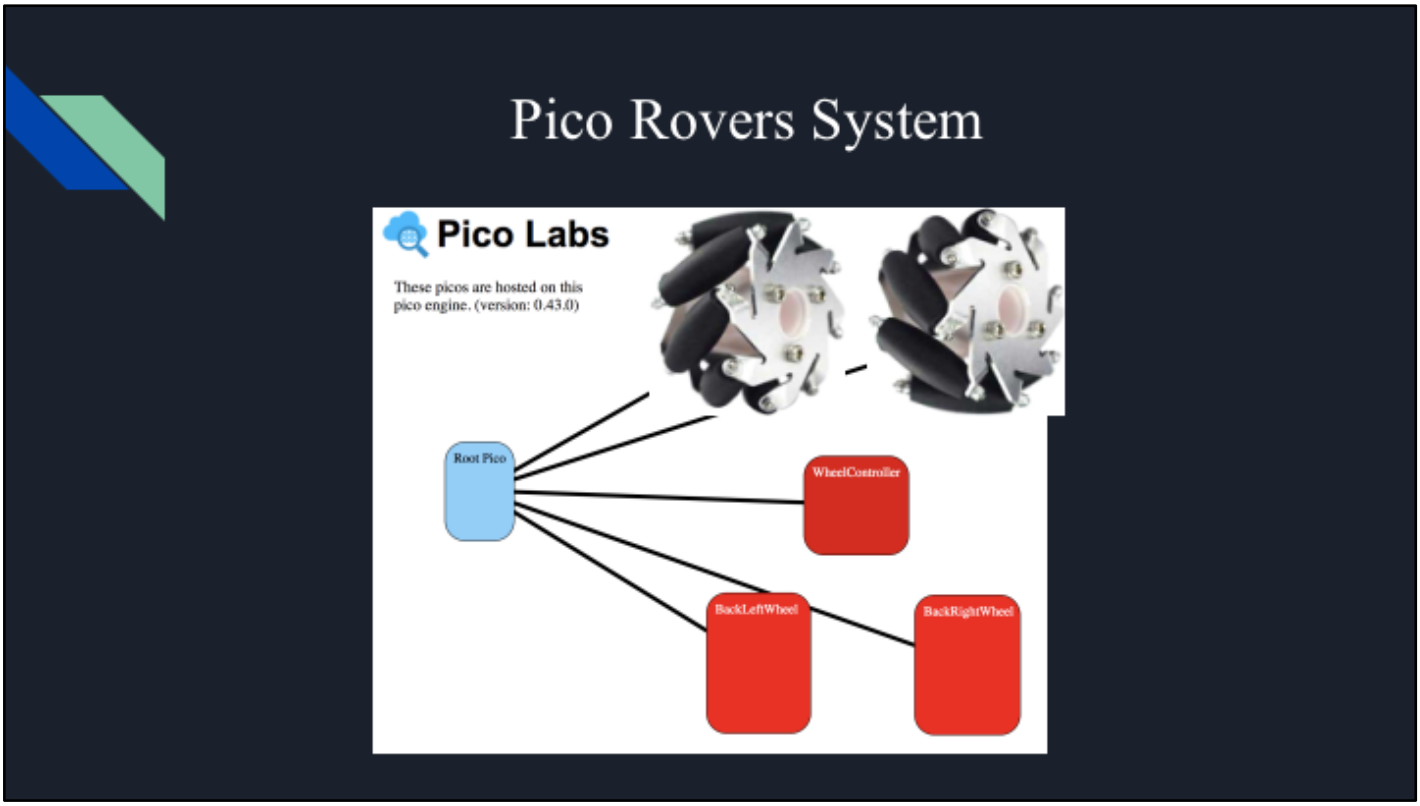
For greater detail you can go to my documentation on the picolabs website.



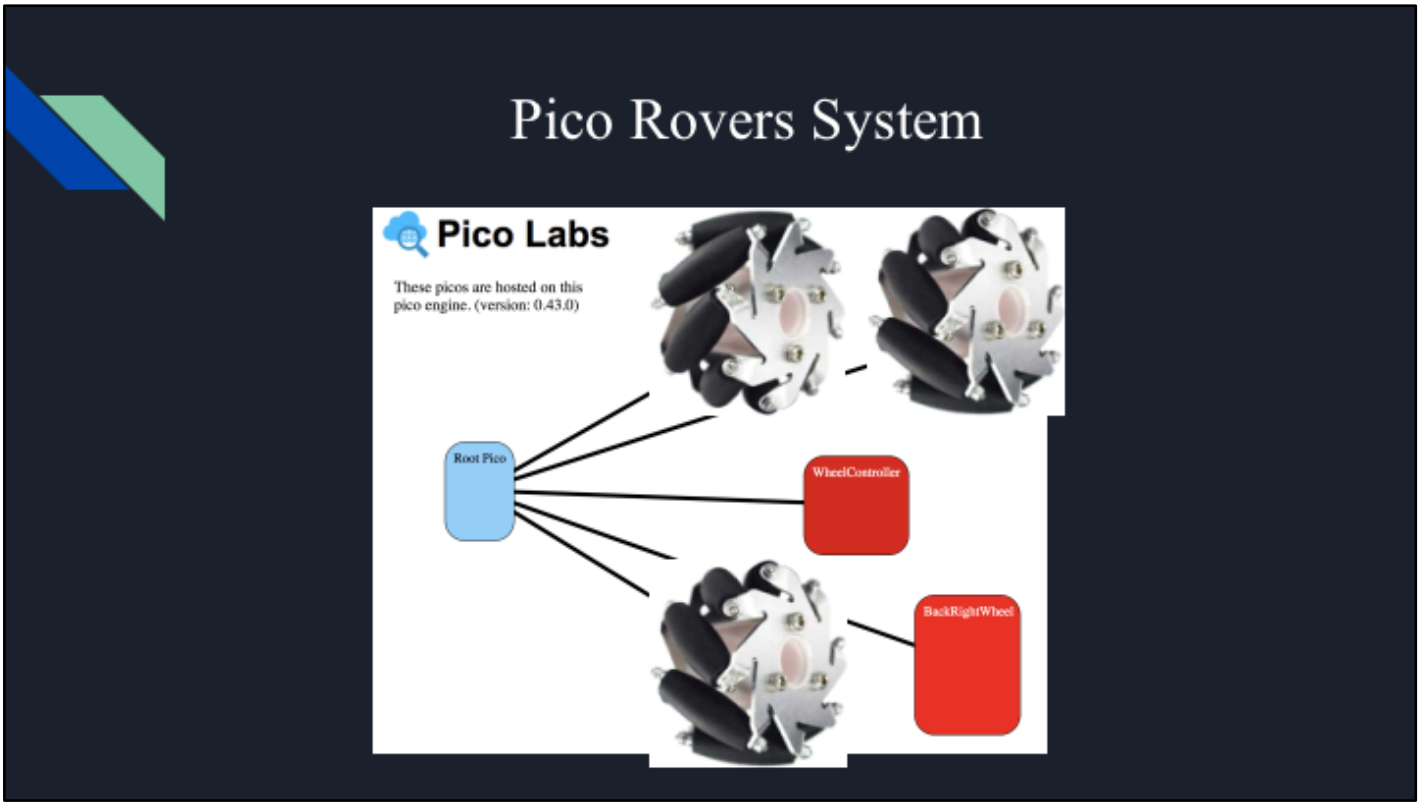
The single Pico to single resource model was implemented where each wheel motor connected to the motorHat is accessed by a single Pico which controls its direction and speed.



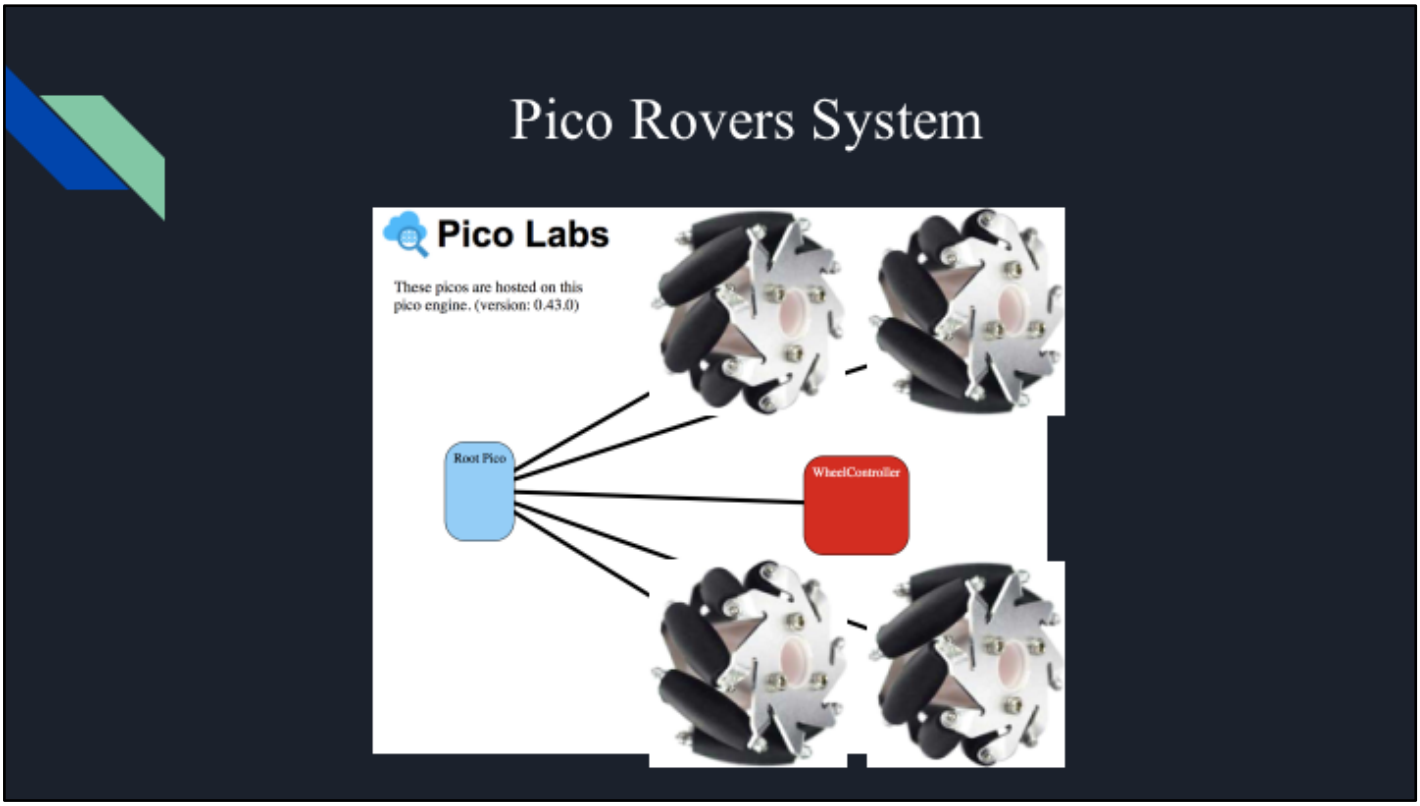
The single Pico to single resource model was implemented where each wheel motor connected to the motorHat is accessed by a single Pico which controls its direction and speed.



The single Pico to single resource model was implemented where each wheel motor connected to the motorHat is accessed by a single Pico which controls its direction and speed.



The single Pico to single resource model was implemented where each wheel motor connected to the motorHat is accessed by a single Pico which controls its direction and speed.



The single Pico to single resource model was implemented where each wheel motor connected to the motorHat is accessed by a single Pico which controls its direction and speed.

Pico Rover Demonstration

Questions

References

https://www.xtendiot.com/wp-content/uploads/2017/08/iot-platform.jpg

https://images-na.ssl-images-amazon.com/images/I/91zSu44%2B34L._SX355_.jpg

https://cdn-shop.adafruit.com/970x728/2348-05.jpg

https://www.robotshop.com/media/files/images2/60mm-mecanum-wheel-set-2x-left-2x-right-2.jpg

https://github.com/Picolab/pico-rover/blob/master/motor-hat.js

https://eg2.gallerycdn.vsassets.io/extensions/eg2/vscode-npm-script/0.3.3/1509888056659/Microsoft.VisualStudio.Services.Icons.Default

References

https://i2.wp.com/garywoodfine.com/wp-content/uploads/2016/02/nodejs.png?fit=1200%2C335&ssl=1&resize=350%2C200