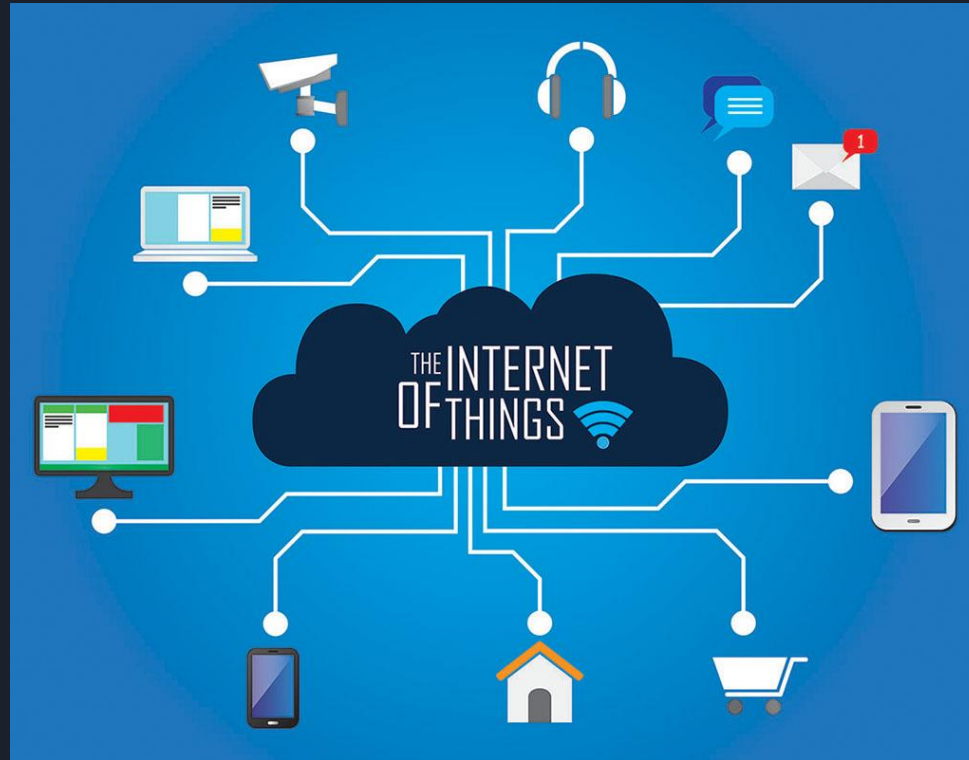


A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is a light greenish-blue. They are positioned diagonally, with the blue one partially covering the green one.

Pico Engine JavaScript Module

Presented By Adam Burdett.

Pico Engine





KRL Example

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



JavaScript Modules

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



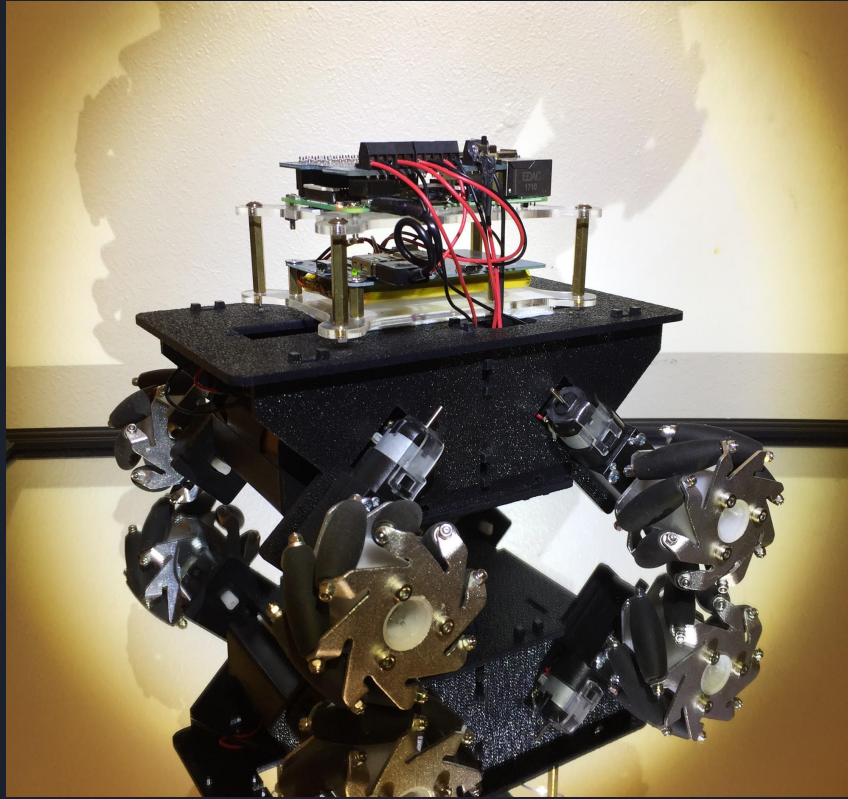
JavaScript Modules

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

JavaScript Modules

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



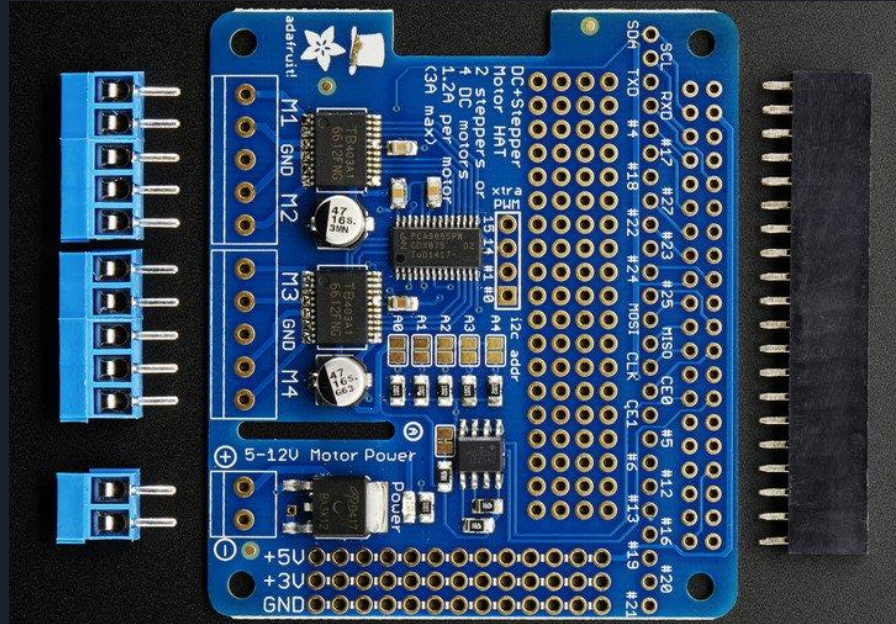


Pico Rover

Raspberry Pi

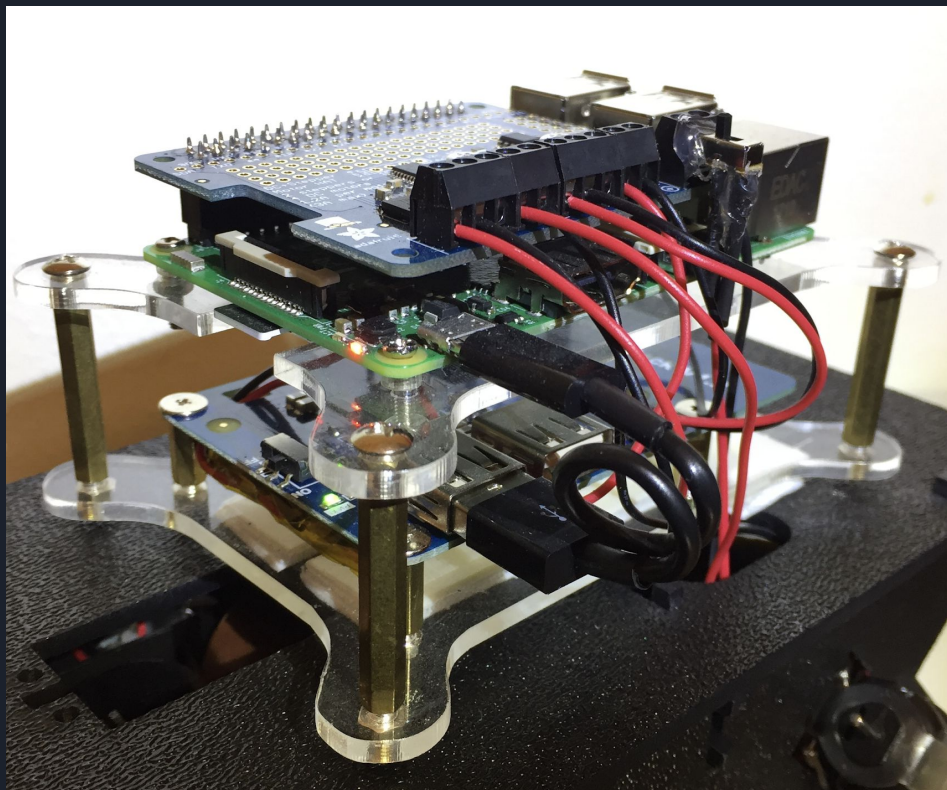


Motor Controller



Mecanum Wheels





Pico Rover

Extending KRL

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

Extending KRL

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



Extending KRL



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



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();
    },
  }, ...
};
```

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();
    },
  }, ...
};
```


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();
    },
  }, ...
};
```

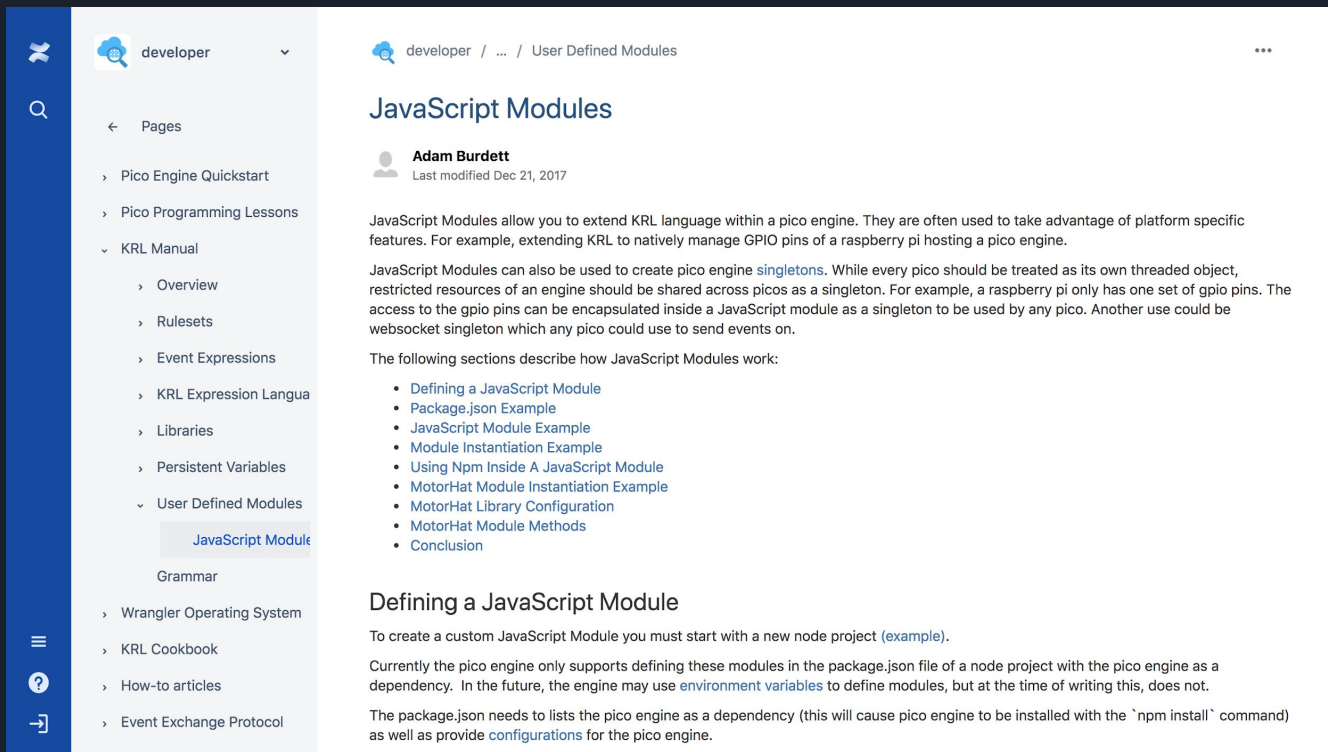




MotorHat Modules

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

Documentation



The screenshot shows the Pico Labs documentation website. On the left is a blue sidebar with navigation icons and a search bar. The main content area is white and displays the 'JavaScript Modules' page. The page header shows the user 'developer' and the path 'User Defined Modules'. The page title is 'JavaScript Modules' by Adam Burdett, last modified Dec 21, 2017. The page content explains that JavaScript Modules allow extending the KRL language within a Pico engine, used for platform-specific features like GPIO pins. It also mentions that these modules can create Pico engine singletons. A list of sections follows: 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, and Conclusion. The 'Defining a JavaScript Module' section is highlighted, stating that a new node project is needed and that the Pico engine currently only supports defining modules in the package.json file.

developer

Pages

- Pico Engine Quickstart
- Pico Programming Lessons
- KRL Manual
 - Overview
 - Rulesets
 - Event Expressions
 - KRL Expression Language
 - Libraries
 - Persistent Variables
- User Defined Modules
 - JavaScript Modules**
 - 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

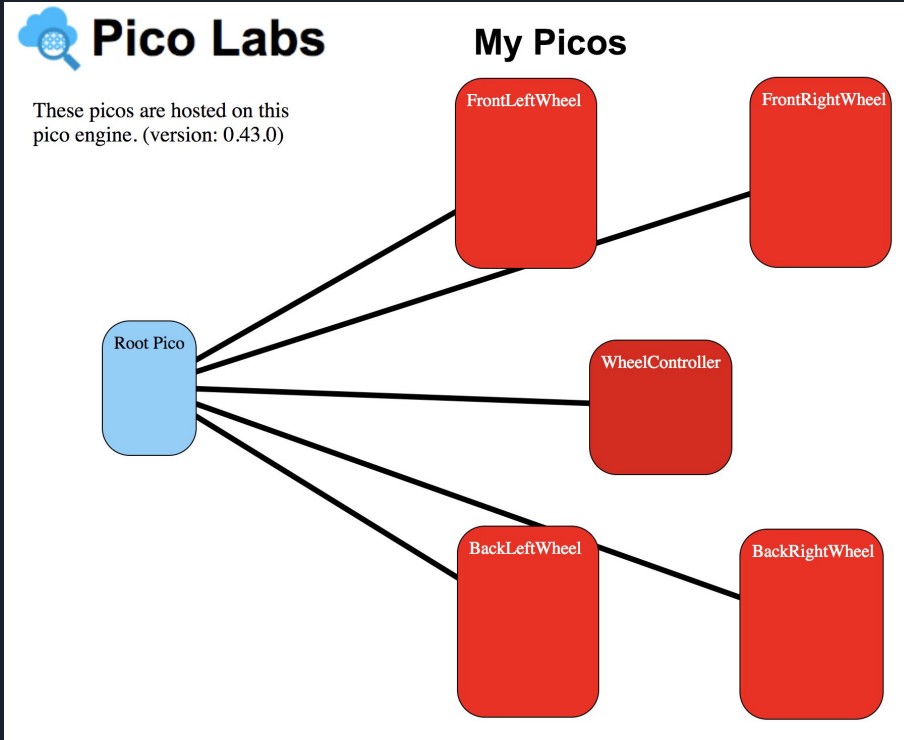
To create a custom JavaScript Module you must start with a new node project ([example](#)).

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.

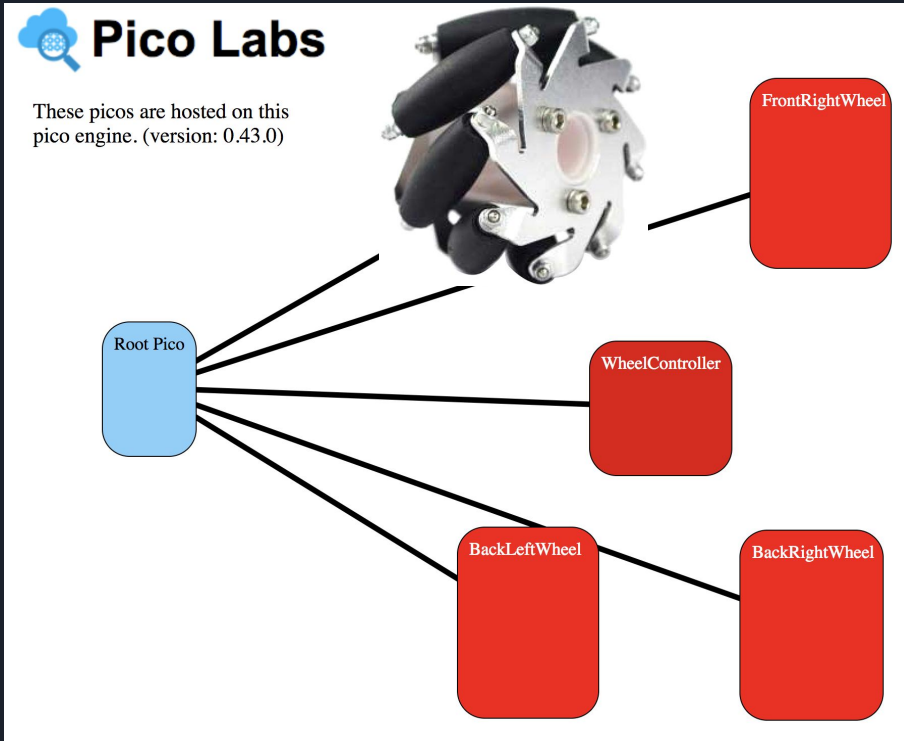
The package.json needs to list the pico engine as a dependency (this will cause pico engine to be installed with the `npm install` command) as well as provide [configurations](#) for the pico engine.

<https://picolabs.atlassian.net/wiki/spaces/docs/pages/96370693/JavaScript+Module>

Pico Rovers System



Pico Rovers System

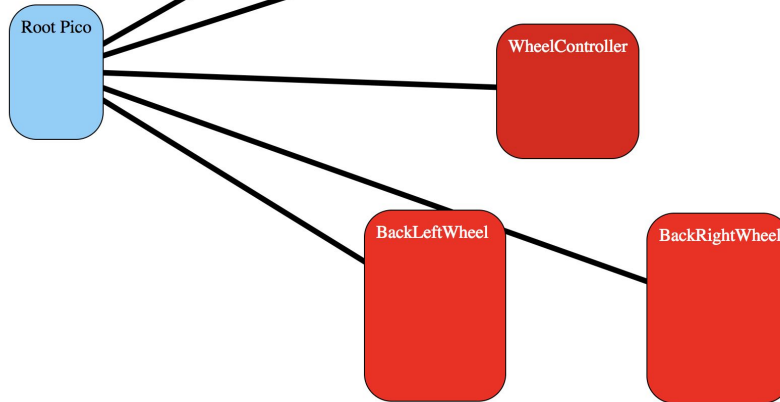


Pico Rovers System

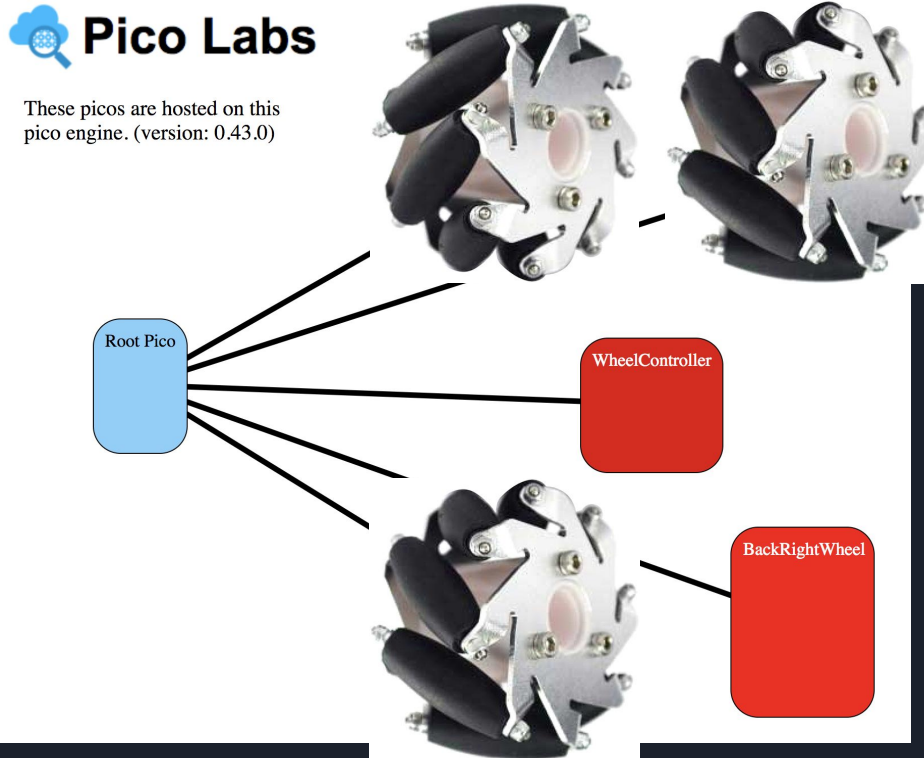


Pico Labs

These picos are hosted on this
pico engine. (version: 0.43.0)



Pico Rovers System



Pico Rovers System



These picos are hosted on this
pico engine. (version: 0.43.0)



WheelController

```
graph LR; RootPico[Root Pico] --- W1(( )); RootPico --- W2(( )); RootPico --- W3(( )); RootPico --- W4(( )); RootPico --- W5(( )); RootPico --- W6(( )); RootPico --- WC[WheelController];
```

A red rounded rectangle representing the WheelController node. It is connected by one line from the Root Pico node.



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.is>

<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>