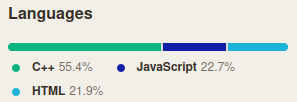
This section describes the software that drives the project. The software stack is mainly composed of the following:

* HTML/CSS/JS for the Web Remote Control interface.
* NodeJS, WebSockets and Johnny Five for the Controller Server.
* C++ Firmata Kernel for the Arduino serial interface.



Johnny-Five is a Robotics Platform written in JavaScript. It’s main goal is to simplify prototyping and communication between software and hardware by standardizing. It uses a growing set of supported instructions and interfaces to make development simpler. Firmata is one of the generic protocols that Johnny-Five uses for communicating with the Arduino hardware. The Firmata protocol communicates over a serial connection with the Arduino, allowing the software on a host computer to completely control the Arduino.

In summary, the Arduino is burnt with the Configurable Firmata sketch, which is a basic sketch that gives the Arduino an ability to communicate to the computer using Firmata. The host computer, connected to the Arduino via a USB cable, has a customized web server integrated with the Johnny-Five API to interface commands sent from a web browser (over WebSockets) to the Arduino.

Synchronization occurs at three levels:

1. The webpage prevents keybound command congestion by using mutual exclusion.
2. The Web Server prevents multiple WebSocket sessions by using a semaphore. This helps prevent unauthorized users from gaining control and executing the critical section.
3. Finally, the Custom Stepper class has a mutual exclusion lock added to prevent Firmata command congestion, since Johnny-Five does not handle this by itself.

**The HTML page**

The web server shows a “private” webpage to the user. Upon loading the webpage, a custom JavaScript initiates a WebSocket to the Web Server. This webpage also has keybindings (as shown in the diagram below), with each keybinding sending a specific instruction to the Web Server.

**The Web Server and WebSocket API Server**

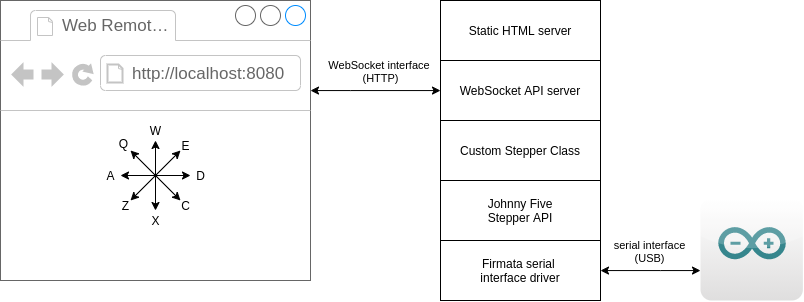
The web server serves a static copy of the page, and the WebSocket API helps transfer the commands over from the webpage to the backend server.

**Custom Stepper Class**

The custom Stepper class serves as an abstraction wrapper between Johnny-Five and WebSocket API. Each Stepper motor is controlled by an instance of the Stepper class. This class implements a mutual exclusion lock for a Stepper motor, while sending the Step command with the count and direction.

**Firmata**

Finally, the Firmata on the Arduino operates by receiving commands from the Johnny-Five wrapper class incorporated into the Web Server. The Arduino then executes the commands sent over by turning the required pins high and low.



**Firmata (kernel code burnt into the Arduino):**

/\*

Firmata is a generic protocol for communicating with microcontrollers

from software on a host computer. It is intended to work with

any host computer software package.

To download a host software package, please clink on the following link

to open the download page in your default browser.

https://github.com/firmata/ConfigurableFirmata#firmata-client-libraries

Copyright (C) 2006-2008 Hans-Christoph Steiner. All rights reserved.

Copyright (C) 2010-2011 Paul Stoffregen. All rights reserved.

Copyright (C) 2009 Shigeru Kobayashi. All rights reserved.

Copyright (C) 2013 Norbert Truchsess. All rights reserved.

Copyright (C) 2014 Nicolas Panel. All rights reserved.

Copyright (C) 2009-2017 Jeff Hoefs. All rights reserved.

This library is free software; you can redistribute it and/or

modify it under the terms of the GNU Lesser General Public

License as published by the Free Software Foundation; either

version 2.1 of the License, or (at your option) any later version.

See file LICENSE.txt for further informations on licensing terms.

Last updated: September 16th, 2017

\*/

/\*

README

This is an example use of ConfigurableFirmata. The easiest way to create a configuration is to

use http://firmatabuilder.com and select the communication transport and the firmata features

to include and an Arduino sketch (.ino) file will be generated and downloaded automatically.

To manually configure a sketch, copy this file and follow the instructions in the

ETHERNET CONFIGURATION OPTION (if you want to use Ethernet instead of Serial/USB) and

FIRMATA FEATURE CONFIGURATION sections in this file.

\*/

#include "ConfigurableFirmata.h"

/\*==============================================================================

\* ETHERNET CONFIGURATION OPTION

\*

\* By default Firmata uses the Serial-port (over USB) of the Arduino. ConfigurableFirmata may also

\* comunicate over ethernet using tcp/ip. To configure this sketch to use Ethernet instead of

\* Serial, uncomment the approprate includes for your particular hardware. See STEPS 1 - 5 below.

\* If you want to use Serial (over USB) then skip ahead to the FIRMATA FEATURE CONFIGURATION

\* section further down in this file.

\*

\* If you enable Ethernet, you will need a Firmata client library with a network transport that can

\* act as a server in order to establish a connection between ConfigurableFirmataEthernet and the

\* Firmata host application (your application).

\*

\* To use ConfigurableFirmata with Ethernet you will need to have one of the following

\* boards or shields:

\*

\* - Arduino Ethernet shield (or clone)

\* - Arduino Ethernet board (or clone)

\* - Arduino Yun

\*

\* If you are using an Arduino Ethernet shield you cannot use the following pins on

\* the following boards. Firmata will ignore any requests to use these pins:

\*

\* - Arduino Uno or other ATMega328 boards: (D4, D10, D11, D12, D13)

\* - Arduino Mega: (D4, D10, D50, D51, D52, D53)

\* - Arduino Leonardo: (D4, D10)

\* - Arduino Due: (D4, D10)

\* - Arduino Zero: (D4, D10)

\*

\* If you are using an ArduinoEthernet board, the following pins cannot be used (same as Uno):

\* - D4, D10, D11, D12, D13

\*============================================================================\*/

// STEP 1 [REQUIRED]

// Uncomment / comment the appropriate set of includes for your hardware (OPTION A, B or C)

/\*

\* OPTION A: Configure for Arduino Ethernet board or Arduino Ethernet shield (or clone)

\*

\* To configure ConfigurableFirmata to use the an Arduino Ethernet Shield or Arduino Ethernet

\* Board (both use the same WIZ5100-based Ethernet controller), uncomment the SPI and Ethernet

\* includes below.

\*/

//#include <SPI.h>

//#include <Ethernet.h>

/\*

\* OPTION B: Configure for a board or shield using an ENC28J60-based Ethernet controller,

\* uncomment out the UIPEthernet include below.

\*

\* The UIPEthernet-library can be downloaded

\* from: https://github.com/ntruchsess/arduino\_uip

\*/

//#include <UIPEthernet.h>

/\*

\* OPTION C: Configure for Arduino Yun

\*

\* The Ethernet port on the Arduino Yun board can be used with Firmata in this configuration.

\* To execute StandardFirmataEthernet on Yun uncomment the Bridge and YunClient includes below.

\*

\* NOTE: in order to compile for the Yun you will also need to comment out some of the includes

\* and declarations in the FIRMATA FEATURE CONFIGURATION section later in this file. Including all

\* features exceeds the RAM and Flash memory of the Yun. Comment out anything you don't need.

\*

\* On Yun there's no need to configure local\_ip and mac address as this is automatically

\* configured on the linux-side of Yun.

\*

\* Establishing a connection with the Yun may take several seconds.

\*/

//#include <Bridge.h>

//#include <YunClient.h>

#if defined ethernet\_h || defined UIPETHERNET\_H || defined \_YUN\_CLIENT\_H\_

#define NETWORK\_FIRMATA

// STEP 2 [REQUIRED for all boards and shields]

// replace with IP of the server you want to connect to, comment out if using 'remote\_host'

#define remote\_ip IPAddress(192, 168, 0, 1)

// OR replace with hostname of server you want to connect to, comment out if using 'remote\_ip'

// #define remote\_host "server.local"

// STEP 3 [REQUIRED unless using Arduino Yun]

// Replace with the port that your server is listening on

#define remote\_port 3030

// STEP 4 [REQUIRED unless using Arduino Yun OR if not using DHCP]

// Replace with your board or Ethernet shield's IP address

// Comment out if you want to use DHCP

#define local\_ip IPAddress(192, 168, 0, 6)

// STEP 5 [REQUIRED unless using Arduino Yun]

// replace with Ethernet shield mac. Must be unique for your network

const byte mac[] = {0x90, 0xA2, 0xDA, 0x0D, 0x07, 0x02};

#endif

/\*==============================================================================

\* FIRMATA FEATURE CONFIGURATION

\*

\* Comment out the include and declaration for any features that you do not need

\* below.

\*

\* WARNING: Including all of the following features (especially if also using

\* Ethernet) may exceed the Flash and/or RAM of lower memory boards such as the

\* Arduino Uno or Leonardo.

\*============================================================================\*/

#include <DigitalInputFirmata.h>

DigitalInputFirmata digitalInput;

#include <DigitalOutputFirmata.h>

DigitalOutputFirmata digitalOutput;

#include <AnalogInputFirmata.h>

AnalogInputFirmata analogInput;

#include <AnalogOutputFirmata.h>

AnalogOutputFirmata analogOutput;

#include <Servo.h>

#include <ServoFirmata.h>

ServoFirmata servo;

// ServoFirmata depends on AnalogOutputFirmata

#if defined ServoFirmata\_h && ! defined AnalogOutputFirmata\_h

#error AnalogOutputFirmata must be included to use ServoFirmata

#endif

#include <Wire.h>

#include <I2CFirmata.h>

I2CFirmata i2c;

#include <OneWireFirmata.h>

OneWireFirmata oneWire;

// StepperFirmata is deprecated as of ConfigurableFirmata v2.10.0. Please update your

// client implementation to use the new, more full featured and scalable AccelStepperFirmata.

#include <StepperFirmata.h>

StepperFirmata stepper;

#include <AccelStepperFirmata.h>

AccelStepperFirmata accelStepper;

#include <SerialFirmata.h>

SerialFirmata serial;

#include <FirmataExt.h>

FirmataExt firmataExt;

#include <FirmataScheduler.h>

FirmataScheduler scheduler;

// To add Encoder support you must first install the FirmataEncoder and Encoder libraries:

// https://github.com/firmata/FirmataEncoder

// https://www.pjrc.com/teensy/td\_libs\_Encoder.html

// #include <Encoder.h>

// #include <FirmataEncoder.h>

// FirmataEncoder encoder;

/\*===================================================================================

\* END FEATURE CONFIGURATION - you should not need to change anything below this line

\*==================================================================================\*/

// dependencies. Do not comment out the following lines

#if defined AnalogOutputFirmata\_h || defined ServoFirmata\_h

#include <AnalogWrite.h>

#endif

#if defined AnalogInputFirmata\_h || defined I2CFirmata\_h || defined FirmataEncoder\_h

#include <FirmataReporting.h>

FirmataReporting reporting;

#endif

// dependencies for Network Firmata. Do not comment out.

#ifdef NETWORK\_FIRMATA

#if defined remote\_ip && defined remote\_host

#error "cannot define both remote\_ip and remote\_host at the same time!"

#endif

#include <EthernetClientStream.h>

#ifdef \_YUN\_CLIENT\_H\_

YunClient client;

#else

EthernetClient client;

#endif

#if defined remote\_ip && !defined remote\_host

#ifdef local\_ip

EthernetClientStream stream(client, local\_ip, remote\_ip, NULL, remote\_port);

#else

EthernetClientStream stream(client, IPAddress(0, 0, 0, 0), remote\_ip, NULL, remote\_port);

#endif

#endif

#if !defined remote\_ip && defined remote\_host

#ifdef local\_ip

EthernetClientStream stream(client, local\_ip, IPAddress(0, 0, 0, 0), remote\_host, remote\_port);

#else

EthernetClientStream stream(client, IPAddress(0, 0, 0, 0), IPAddress(0, 0, 0, 0), remote\_host, remote\_port);

#endif

#endif

#endif

/\*==============================================================================

\* FUNCTIONS

\*============================================================================\*/

void systemResetCallback()

{

// initialize a default state

// pins with analog capability default to analog input

// otherwise, pins default to digital output

for (byte i = 0; i < TOTAL\_PINS; i++) {

if (IS\_PIN\_ANALOG(i)) {

#ifdef AnalogInputFirmata\_h

// turns off pull-up, configures everything

Firmata.setPinMode(i, PIN\_MODE\_ANALOG);

#endif

} else if (IS\_PIN\_DIGITAL(i)) {

#ifdef DigitalOutputFirmata\_h

// sets the output to 0, configures portConfigInputs

Firmata.setPinMode(i, OUTPUT);

#endif

}

}

#ifdef FirmataExt\_h

firmataExt.reset();

#endif

}

/\*==============================================================================

\* SETUP()

\*============================================================================\*/

void setup()

{

/\*

\* ETHERNET SETUP

\*/

#ifdef NETWORK\_FIRMATA

#ifdef \_YUN\_CLIENT\_H\_

Bridge.begin();

#else

#ifdef local\_ip

Ethernet.begin((uint8\_t \*)mac, local\_ip); //start Ethernet

#else

Ethernet.begin((uint8\_t \*)mac); //start Ethernet using dhcp

#endif

#endif

delay(1000);

#endif

/\*

\* FIRMATA SETUP

\*/

Firmata.setFirmwareVersion(FIRMATA\_FIRMWARE\_MAJOR\_VERSION, FIRMATA\_FIRMWARE\_MINOR\_VERSION);

#ifdef FirmataExt\_h

#ifdef DigitalInputFirmata\_h

firmataExt.addFeature(digitalInput);

#endif

#ifdef DigitalOutputFirmata\_h

firmataExt.addFeature(digitalOutput);

#endif

#ifdef AnalogInputFirmata\_h

firmataExt.addFeature(analogInput);

#endif

#ifdef AnalogOutputFirmata\_h

firmataExt.addFeature(analogOutput);

#endif

#ifdef ServoFirmata\_h

firmataExt.addFeature(servo);

#endif

#ifdef I2CFirmata\_h

firmataExt.addFeature(i2c);

#endif

#ifdef OneWireFirmata\_h

firmataExt.addFeature(oneWire);

#endif

#ifdef StepperFirmata\_h

firmataExt.addFeature(stepper);

#endif

#ifdef AccelStepperFirmata\_h

firmataExt.addFeature(accelStepper);

#endif

#ifdef SerialFirmata\_h

firmataExt.addFeature(serial);

#endif

#ifdef FirmataReporting\_h

firmataExt.addFeature(reporting);

#endif

#ifdef FirmataScheduler\_h

firmataExt.addFeature(scheduler);

#endif

#ifdef FirmataEncoder\_h

firmataExt.addFeature(encoder);

#endif

#endif

/\* systemResetCallback is declared here (in ConfigurableFirmata.ino) \*/

Firmata.attach(SYSTEM\_RESET, systemResetCallback);

// Network Firmata communicates with Ethernet-shields over SPI. Therefor all

// SPI-pins must be set to PIN\_MODE\_IGNORE. Otherwise Firmata would break SPI-communication.

// add Pin 10 and configure pin 53 as output if using a MEGA with Ethernetshield.

// No need to ignore pin 10 on MEGA with ENC28J60, as here pin 53 should be connected to SS:

#ifdef NETWORK\_FIRMATA

#ifndef \_YUN\_CLIENT\_H\_

// ignore SPI and pin 4 that is SS for SD-Card on Ethernet-shield

for (byte i = 0; i < TOTAL\_PINS; i++) {

if (IS\_PIN\_SPI(i)

|| 4 == i // SD Card on Ethernet shield uses pin 4 for SS

|| 10 == i // Ethernet-shield uses pin 10 for SS

) {

Firmata.setPinMode(i, PIN\_MODE\_IGNORE);

}

}

// pinMode(PIN\_TO\_DIGITAL(53), OUTPUT); configure hardware-SS as output on MEGA

pinMode(PIN\_TO\_DIGITAL(4), OUTPUT); // switch off SD-card bypassing Firmata

digitalWrite(PIN\_TO\_DIGITAL(4), HIGH); // SS is active low;

#endif

#if defined(\_\_AVR\_ATmega1280\_\_) || defined(\_\_AVR\_ATmega2560\_\_)

pinMode(PIN\_TO\_DIGITAL(53), OUTPUT); // configure hardware SS as output on MEGA

#endif

// start up Network Firmata:

Firmata.begin(stream);

#else

// Uncomment to save a couple of seconds by disabling the startup blink sequence.

// Firmata.disableBlinkVersion();

// start up the default Firmata using Serial interface:

Firmata.begin(57600);

#endif

Firmata.parse(SYSTEM\_RESET); // reset to default config

}

/\*==============================================================================

\* LOOP()

\*============================================================================\*/

void loop()

{

#ifdef DigitalInputFirmata\_h

/\* DIGITALREAD - as fast as possible, check for changes and output them to the

\* stream buffer using Firmata.write() \*/

digitalInput.report();

#endif

/\* STREAMREAD - processing incoming message as soon as possible, while still

\* checking digital inputs. \*/

while (Firmata.available()) {

Firmata.processInput();

#ifdef FirmataScheduler\_h

if (!Firmata.isParsingMessage()) {

goto runtasks;

}

}

if (!Firmata.isParsingMessage()) {

runtasks: scheduler.runTasks();

#endif

}

/\* SEND STREAM WRITE BUFFER - TO DO: make sure that the stream buffer doesn't go over

\* 60 bytes. use a timer to sending an event character every 4 ms to

\* trigger the buffer to dump. \*/

#ifdef FirmataReporting\_h

if (reporting.elapsed()) {

#ifdef AnalogInputFirmata\_h

/\* ANALOGREAD - do all analogReads() at the configured sampling interval \*/

analogInput.report();

#endif

#ifdef I2CFirmata\_h

// report i2c data for all device with read continuous mode enabled

i2c.report();

#endif

#ifdef FirmataEncoder\_h

// report encoders positions if reporting enabled.

encoder.report();

#endif

}

#endif

#ifdef StepperFirmata\_h

stepper.update();

#endif

#ifdef AccelStepperFirmata\_h

accelStepper.update();

#endif

#ifdef SerialFirmata\_h

serial.update();

#endif

#if defined NETWORK\_FIRMATA && !defined local\_ip &&!defined \_YUN\_CLIENT\_H\_

// only necessary when using DHCP, ensures local IP is updated appropriately if it changes

if (Ethernet.maintain()) {

stream.maintain(Ethernet.localIP());

}

#endif

}

**Stepper.js - Stepper Motor class (the template code to operate a single stepper)**

/\*\*

\* In order to use the Stepper class, your board must be flashed with

\* either of the following:

\*

\* - AdvancedFirmata https://github.com/soundanalogous/AdvancedFirmata

\* - ConfigurableFirmata https://github.com/firmata/arduino/releases/tag/v2.6.2

\*

\*/

const { Stepper } = require("johnny-five");

const { stepsPerRev, rpm, acceleration, deceleration, limits } = require('./config');

// Reference: http://johnny-five.io/examples/stepper-driver/

// Reference: https://www.instructables.com/Stepper-Motor-With-Arduino-UNO/

const stepperStep = (stepper, steps) => {

return new Promise((resolve) => {

let direction = steps > 0 ? // if \_steps is positive

Stepper.DIRECTION.CW // spin the motor clockwise

: Stepper.DIRECTION.CCW; // else spin counter-clockwise

stepper.step({

steps: Math.abs(steps), // spin the motor for \_steps

direction // in the direction determined above

},

() => resolve(steps));

});

};

class Motor {

stepper; // the stepper motor instance referred to here

spinning = false; // a lock to establish critical section(i.e. already spinning) for motor.

/\*\*

\*

\* @param {number} step Step pin connected to stepper driver.

\* @param {number} dir Dir pin connected to stepper driver.

\* @param {object} config (optional) configuration variables.

\*/

constructor(step, dir, config={}) {

if(typeof step !== 'number' && typeof dir !== 'number')

throw 'step and dir values expect pin numbers.';

this.stepper = new Stepper({

type: Stepper.TYPE.DRIVER,

stepsPerRev: config.stepsPerRev || stepsPerRev,

pins: { step, dir, }

});

this.stepper

.cw() // Set the default spin to clockwise.

.rpm(config.rpm || rpm) // Set the Motor RPM,

// .accel(config.accel || acceleration) // the acceleration,

// .decel(config.decel || deceleration); // and the deceleration.

}

/\*\*

\* Spins the motor a fixed number of steps.

\* The sign of the number entered determines the motor spin direction.

\* Enter positive number for clockwise, negative number for anti-clockwise.

\* @param {number} steps The number of steps to spin.

\* @returns {Promise} Resolves on success, rejects if already spinning.

\*/

async step(steps) {

if(!this.Spinning()) { // if not already spinning

this.setSpinning(true); // enter critical section (motor now spinning)

return await stepperStep(this.stepper, steps) // actually start the motor spinning

.then(() => this.setSpinning(false)); // leave critical section after spinning

} else throw 'Already spinning'; // throw if already spinning

}

setSpinning(value) {

return (this.spinning = value);

}

Spinning() {

return this.spinning;

}

}

module.exports = { Motor };

**Config.js – Configuration variables for easier customization**

const stepsPerRev = 200; // 1 revolution = 360 deg

const limits = 20; // the number of steps to limit the stepper

const rpm = 180;

const acceleration = 1600;

const deceleration = 1600;

module.exports = {

stepsPerRev, limits, rpm, acceleration, deceleration

};

**Server.js – The Server side code which interfaces to the Arduino over Firmata interface**

/\*\*

\* This is the server side code, NOT client side.

\*/

const express = require('express')

const path = require('path')

const app = express()

const { Motor } = require('../index')

var server = require('http').createServer(app)

var io = require('socket.io')(server, {

cors: true,

origins: ["http://localhost:8080"],

});

const { Board } = require('johnny-five');

const board = new Board({

repl: true,

timeout: 10000,

});

board.on("ready", () => {

console.log(new Date(), 'Board ready');

const motor = [

{ motor: new Motor(9, 8), event: 'motorIML:step' },

{ motor: new Motor(7, 6), event: 'motorA:step' },

{ motor: new Motor(5, 4), event: 'motorB:step' },

{ motor: new Motor(3, 2), event: 'motorC:step' },

];

let idConnected = ""; // holds the ID of the active room...

io.on('connection', async socket => {

if (!idConnected) idConnected = socket.id; // if first connection, save the room ID

else socket.disconnect(); // disconnect every subsequent request which does not match

console.log(new Date(), socket.id, 'connected:', socket.connected, 'at', socket.conn.remoteAddress);

try {

// create socket events for each motor in the array above.

for(let m of motor)

socket.on(m.event, async (data) => {

if (data === 'cw') await m.motor.step(1);

if (data === 'ccw') await m.motor.step(-1);

});

} catch(e) {

console.error(e);

}

socket.on('disconnect', () => {

// if the connected room is closing, forget the connection

if(socket.id === idConnected) idConnected = "";

console.log(new Date(), socket.id, 'attempted to close connection at', socket.conn.remoteAddress);

});

});

app.use('/', express.static(\_\_dirname));

app.get('/rtt/:time', (req, res) => {

res.send(req.params.time); // used to obtain round trip time at client to prevent congestion of commands...

});

server.listen(8080, () => {

console.log(new Date(), 'Now listening for commands');

});

});

**index.html – The HTML page which will show on the web browser for control**

<head>

<title>Web Remote Control</title>

<style>

fieldset {

padding: 20px;

}

</style>

<script src="/socket.io.min.js" integrity="sha384-cPwlPLvBTa3sKAgddT6krw0cJat7egBga3DJepJyrLl4Q9/5WLra3rrnMcyTyOnh"

crossorigin="anonymous"></script>

<script>

var socket = io.connect('http://localhost:8080');

window.onload = () => {

// start by hiding the error message

document.querySelector('#error').style.display = 'none';

socket.on('connect', (data) => {

// return page elements back to normal on successful connection.

document.querySelector("#key").textContent = 'yet to press';

async function step(motor, c) {

await fetch('/rtt/' + Date.now()); // wait for rtt to return

socket.emit(`${motor}:step`, c); // then emit signal

return;

}

const keys = [

'KeyS', 'KeyW', 'KeyE', 'KeyD', 'KeyC', 'KeyX', 'KeyZ', 'KeyA', 'KeyQ', 'KeyU', 'KeyJ',

];

function keyToDirection(code) {

return keys.indexOf(code);

}

let locked = false;

document.onkeydown = async event => {

if (!locked) {

locked = true;

let motors = [], clockwise = false;

switch (keyToDirection(event.code)) {

case 1: clockwise = true;

case 5: document.querySelector("#key").textContent = event.code;

motors = ['A', 'B', 'C'];

break;

case 2: clockwise = true;

case 6: document.querySelector("#key").textContent = event.code;

motors = ['B', 'C', 'A'];

break;

case 3: clockwise = true;

case 7: document.querySelector("#key").textContent = event.code;

motors = ['-', 'B', 'C'];

break;

case 4: clockwise = true;

case 8: document.querySelector("#key").textContent = event.code;

motors = ['C', 'A', 'B'];

break;

case 9: clockwise = true;

case 10: // only intermediate link motor will operate on U and J

document.querySelector("#key").textContent = event.code;

await step(`motorIML`, clockwise ? 'cw' : 'ccw');

locked = false; // unlock after operation

return;

default: return;

}

if (motors[0] !== '-') {

await step(`motor${motors[0]}`, clockwise ? 'cw' : 'ccw');

step(`motor${motors[0]}`, clockwise ? 'cw' : 'ccw');

}

step(`motor${motors[1]}`, clockwise ? 'cw' : 'ccw');

step(`motor${motors[2]}`, clockwise ? 'cw' : 'ccw')

.then(o => locked = false);

}

}

socket.on('disconnect', () => {

// show the error message on server-side socket disconnect.

document.querySelector('#error').style.display = 'block';

document.querySelector('#error').style.color = 'red';

});

});

};

</script>

</head>

<body>

<p id="error">

<em>NOTE: Do ensure only one instance of this page is open!</em>

</p>

Using qwedcxza keys with keyS keybindings.

<img src="/mdewsp.png" width="240" alt="Diagram showing key to direction mapping." />

<p>Current direction: <span id="key"></span><br></p>

</body>