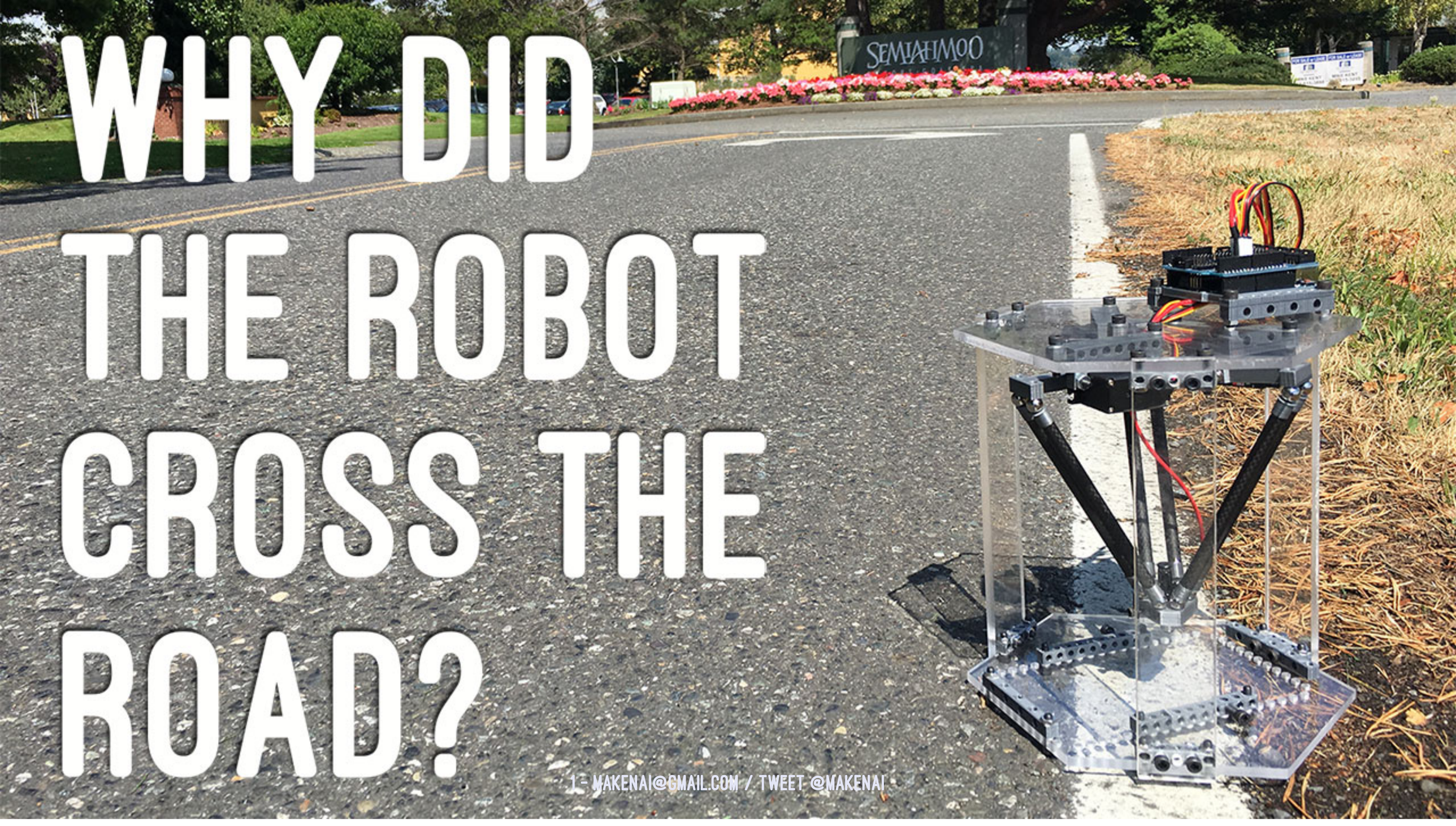
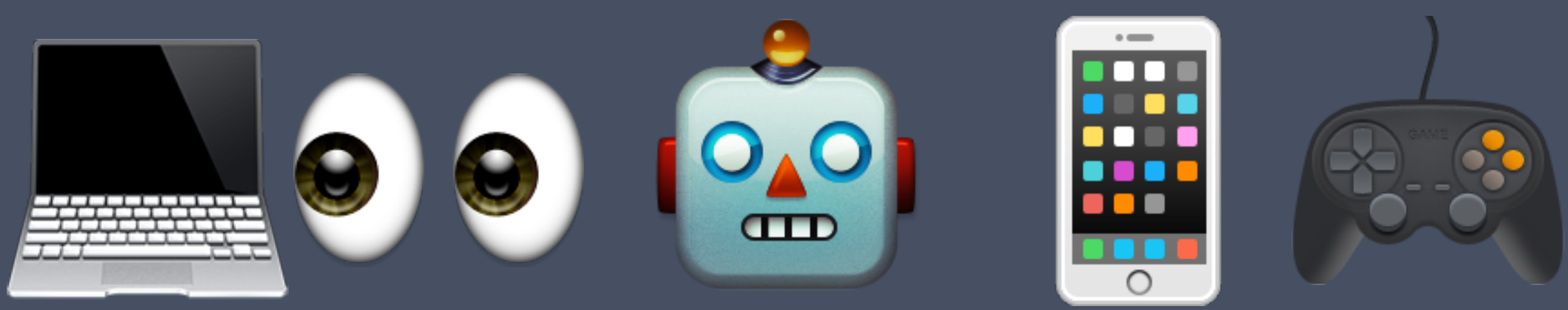
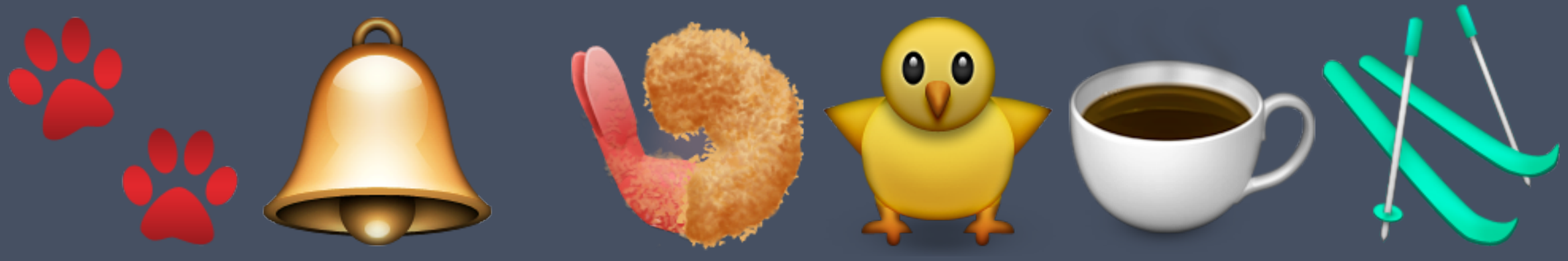
WHY DID THE ROBOT CROSS THE ROAD?



1- MAKENAI@GMAIL.COM / TWEET @MAKENAI



We're going to cover a bit about computer vision, robots and mobile games!



BY @MAKENAI (PAWEL SZYMCZYKOWSKI)

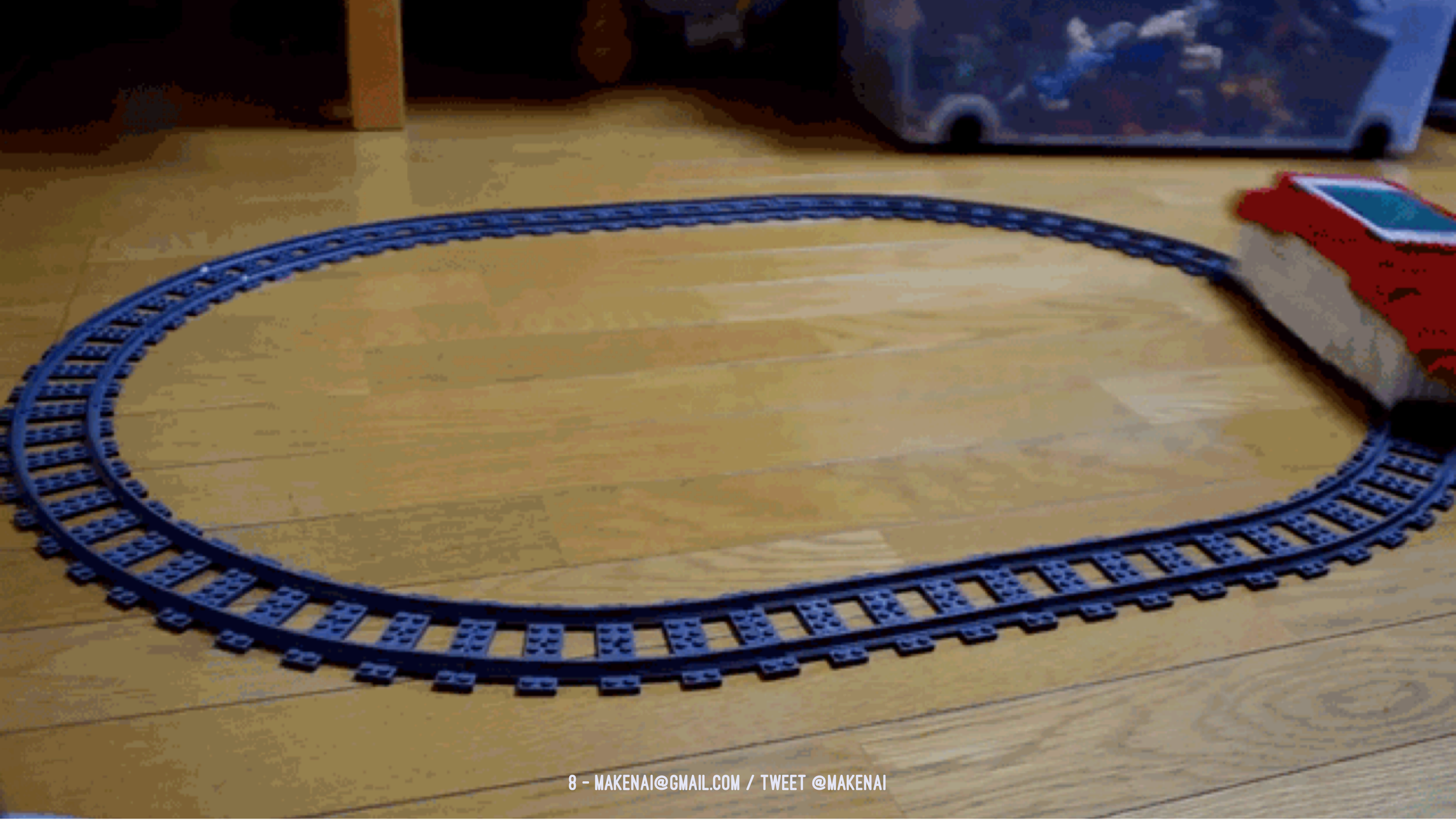
My name is Pawel Szymczykowski - pronounced pretty much like those emoji there. You can tweet stuff at me on twitter using @makenai.

I'm from Las Vegas, Nevada. I'm in to lot of maker-y things like electronics, robotics and 3D printing. I work as a developer at a small startup in Vegas called Wedgies.

Has anyone ever played this or know what it is? It's called Crossy Road and it's kind of an endless frogger. For a while, we played this a lot at the office.

One popular feature of this game is that it has a lot of unlockable characters. Much like Pokemon, we tried to collect them all. You can unlock some with skills, but others can only be unlocked with either in game or real coin. You collect coins in the game, or by watching commercials, which is completely tedious.

Through my cursory knowledge of robotics, I knew that robots are great at automating tasks that are too precise, dangerous or tedious for humans. I decided that a robot would be the answer to my problems. This is the story of getting there.



Side Note: though it wasn't out yet when I started working on this talk, Pokemon Go players sure have come up a lot of creative automations!

IMAGE: GRPH3B18 VIA WIKIPEDIA[1]

[1] HTTPS://EN.WIKIPEDIA.ORG/WIKI/POINTINGDEVICEGESTURE

The main thing with mobile games is that they are designed for interacting with human fingers and gestures like taps and swipes that are hard to replicate for robots that don't usually have sausage fingers.

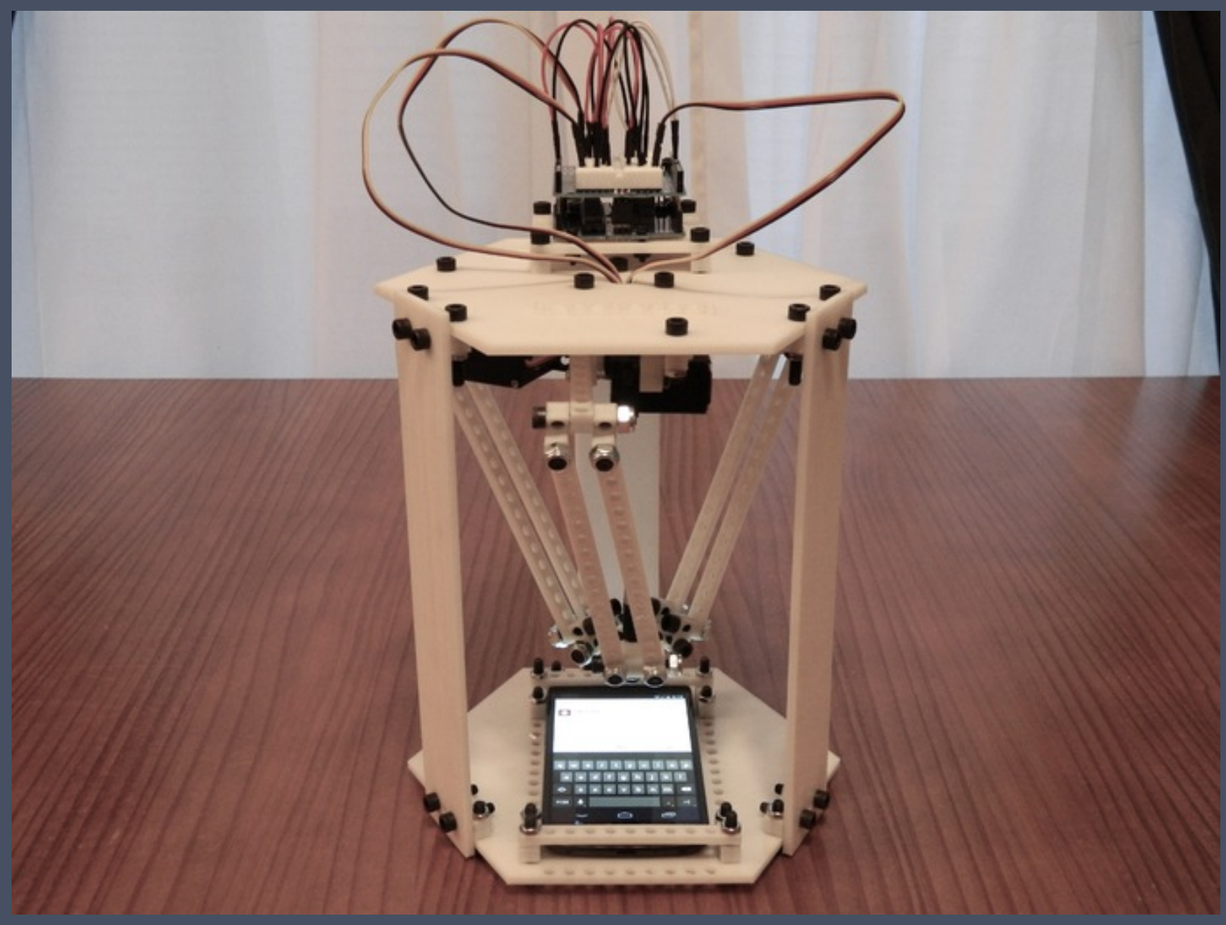


IMAGE VIA JASON HUGGINS

WWW.TAPSTER.IO / @TAPSTERBOT

Luckily, this is one problem that has already been solved for us, and is open source. I really love open source hardware. A lot. This project is Tapsterbot. A cool guy named Jason Huggins designed this and published it on GitHub. It was designed for mobile testing at his company Sauce Labs, driving a stylus and interacting with phones and tablets. If you have access to a 3D printer, you can print most of the parts for this robot. If you don't know anyone with a 3D printer, look up your local hackerspace!

Tapster is what's called a Delta Robot. It's in a family of robots known as cartesian coordinate robots that can move their business end (or end effector) anywhere on their X, Y or Z axis.

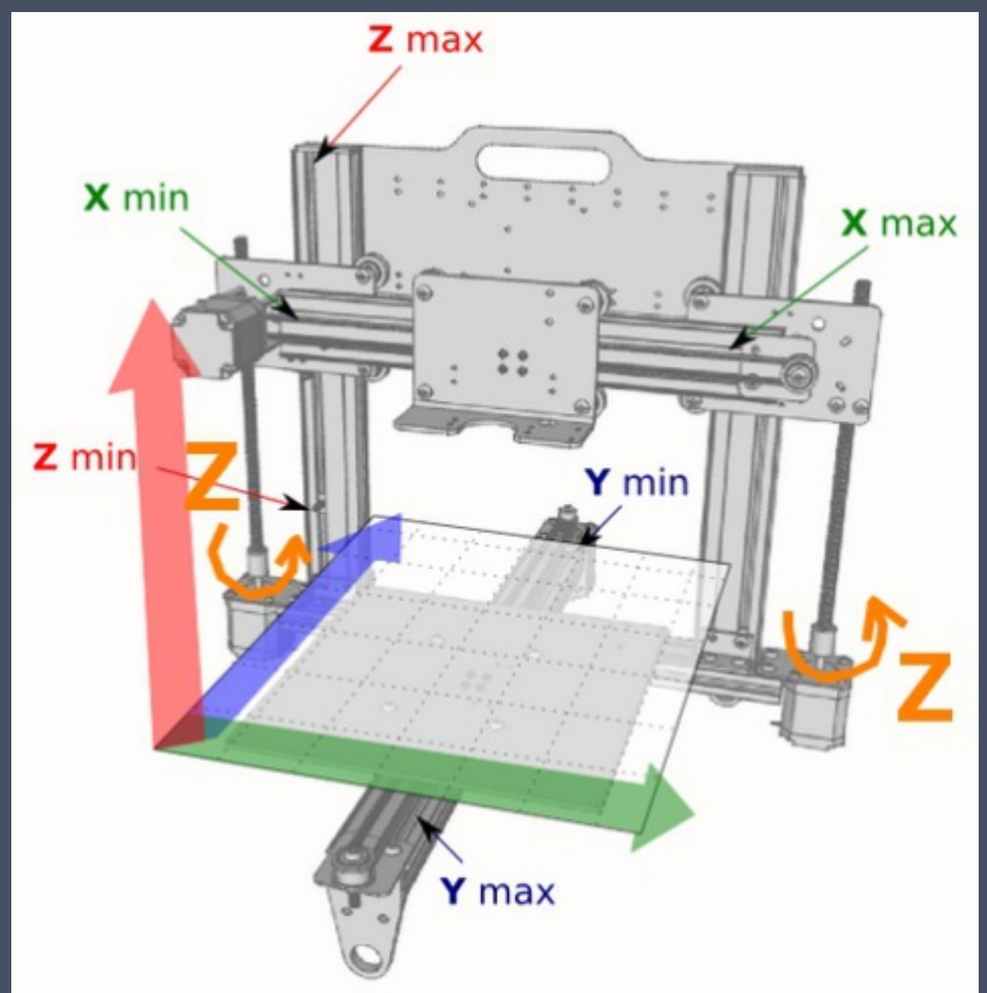


IMAGE: MAKERSLIDE-MACHINES.COM[3]

[3] HTTP://WWW.MAKERSLIDE-MACHINES.COM/CO/PAGE-Z-AXIS-CARTESIAN-3D-PRINTER.HTML

A more common example of a cartesian device is a 3D printer. Those tend to use pretty specialized parts like belts and screw drives and linear bearings, so they are pretty complicated to build. They are pretty simple to control though, since they're one motor per axis you can expect that if you move the motor on the X axis by 20 cm, end end effector will also move by a predictable amount on the X axis based on your pulley size or screw drive pitch.

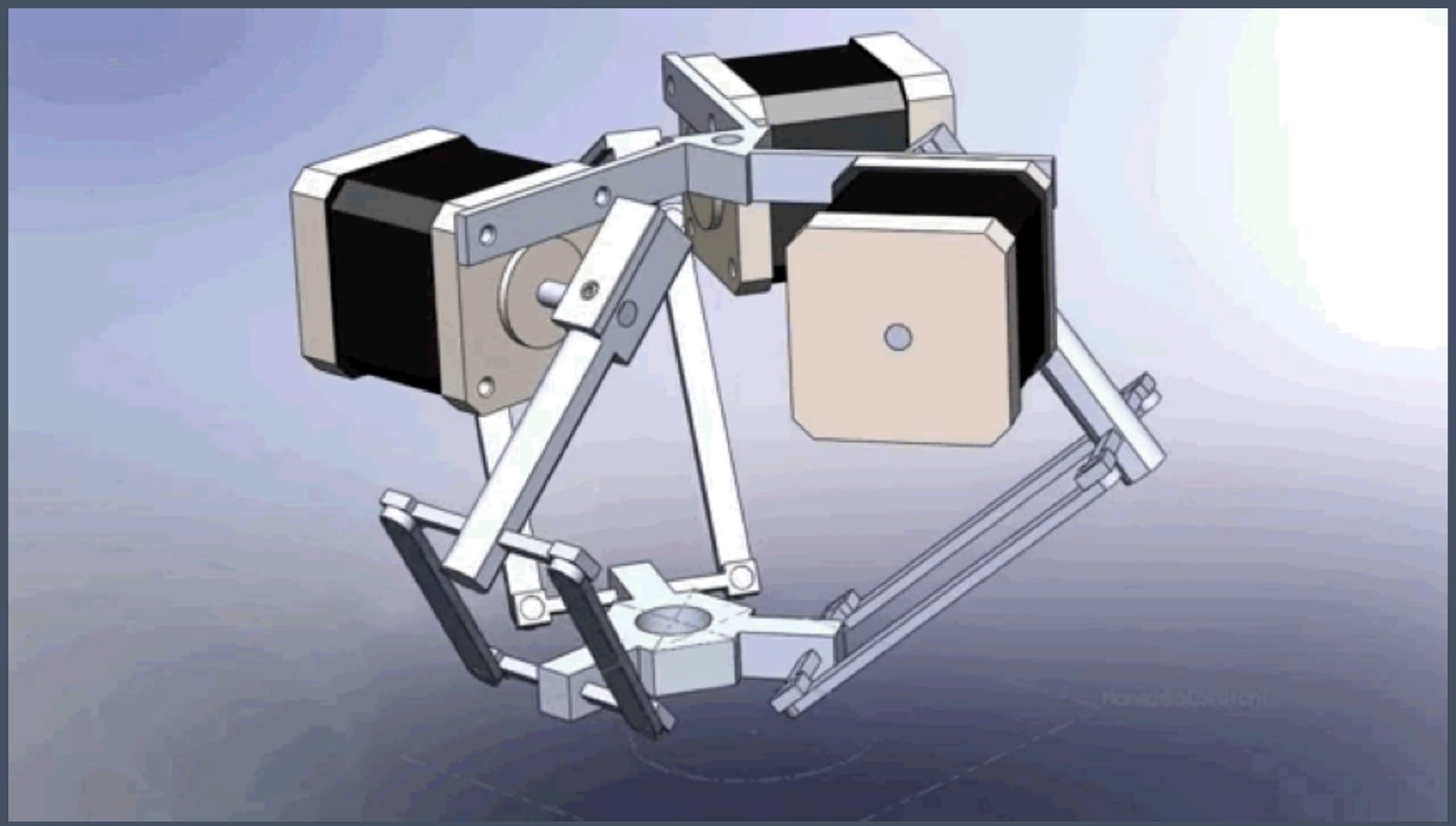
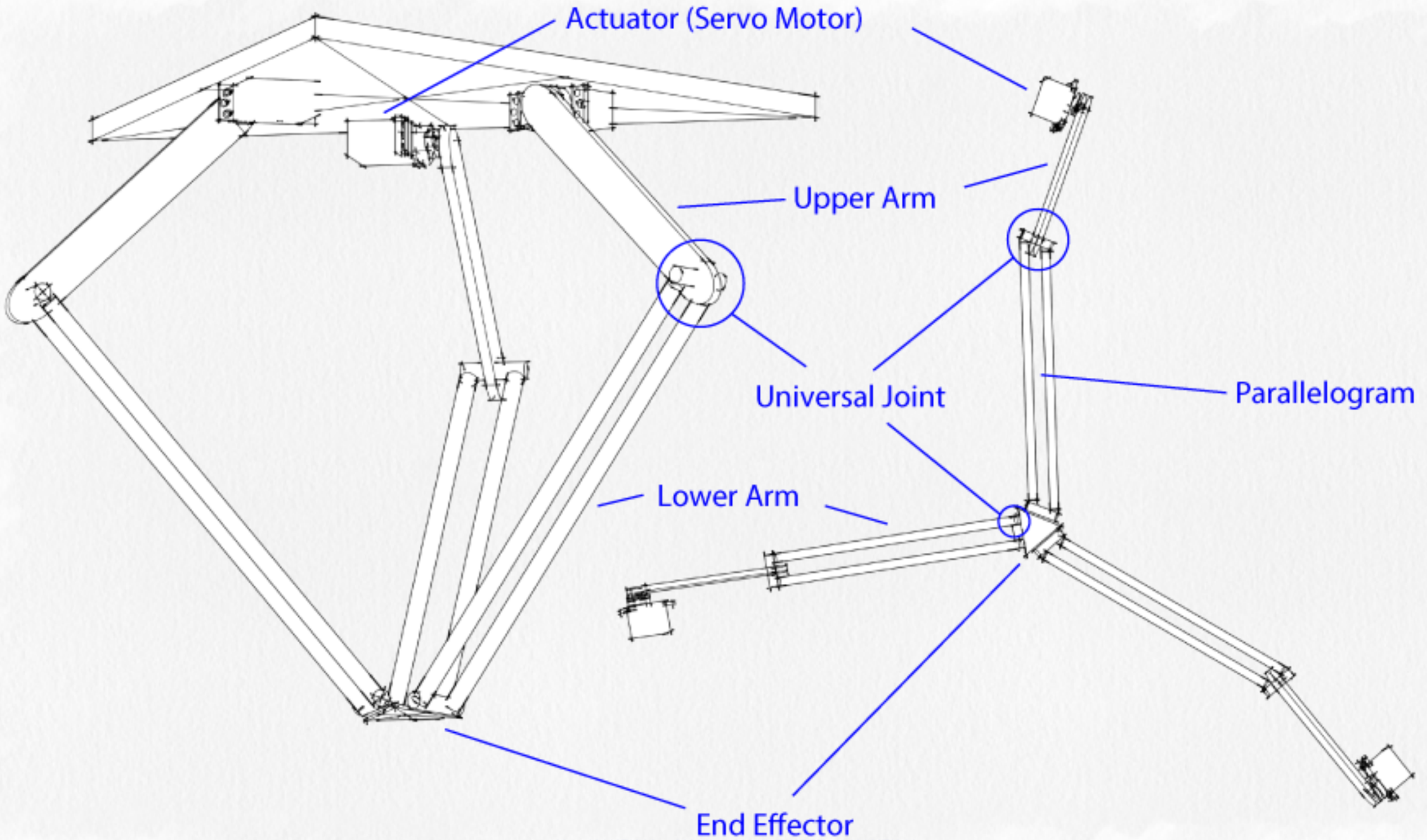


IMAGE: AGENTDEXTER47 VIA YOUTUBE[4]

[4] HTTPS://WWW.YOUTUBE.COM/WATCH?V=SP2FMPGPNSO

A delta on the other hand is really simple to build. It only has motors, arms and joints. They are much harder to control though. Rather than having one motor control one axis, each of the three motors work in collaboration to position the end effector and you need some kinematics calculations to figure out their proper position.



Here are the basic parts of a delta robot. The thing to take away from this slide is that the servo motors are up top and that the upper arms only move back and forth in a half circle pattern between 0 and 90 degrees. The universal joints kind of a ball and socket joint similar to your shoulders that can rotate around in 360 degrees, effectively making a sphere shape. Circles up top with a radius of the upper, and spheres at the bottom with the radius of the lower arm. Also, remember the end effector - that's going to come up again.

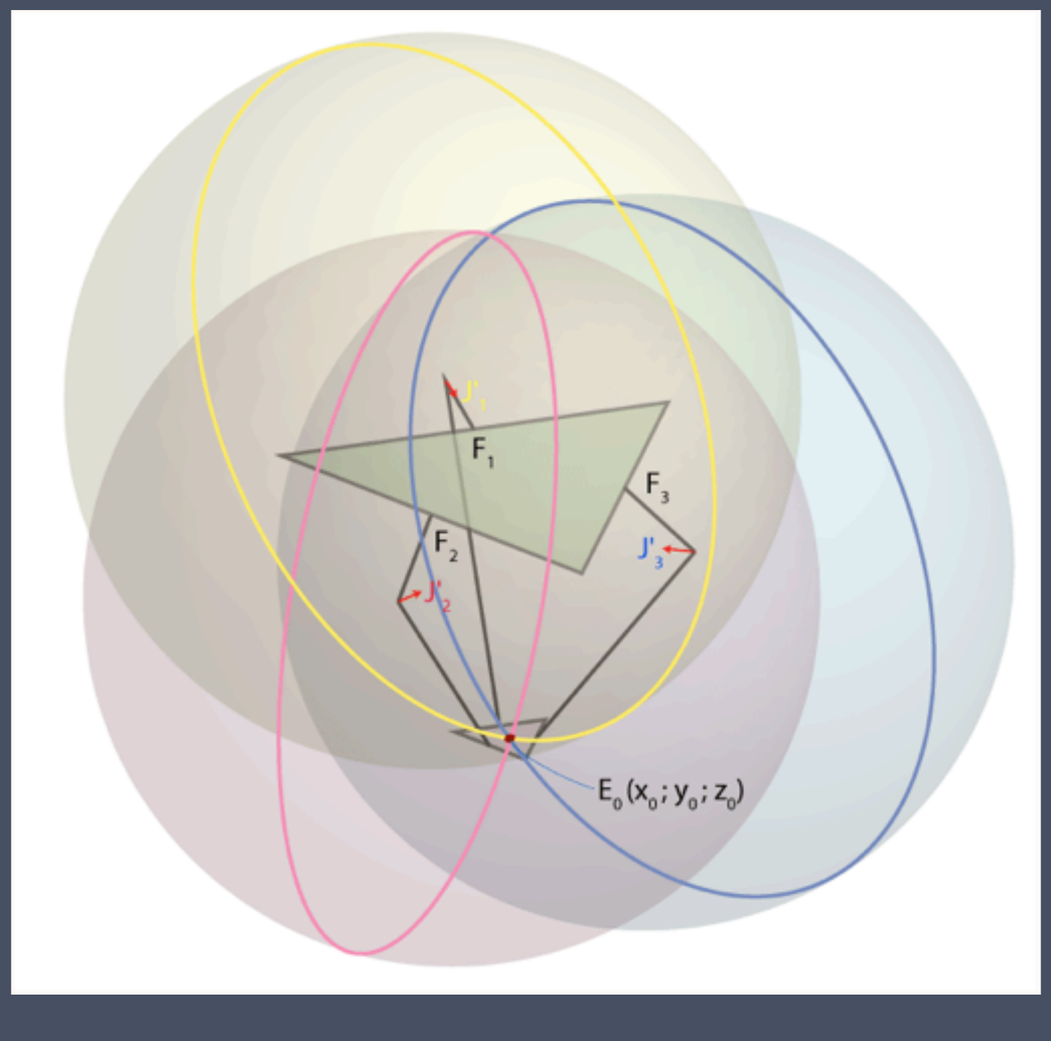


IMAGE: 'MZAVATSKY' ON TROSSEN ROBOTICS[5]

[5] HTTP://FORUMS.TROSSENROBOTICS.COM/TUTORIALS/INTRODUCTION-129/DELTA-ROBOT-KINEMATICS-3276/

A delta has three arms, so that means you have three spheres you can position along the y-z plane. To put the end effector at a specific XYZ coordinate, you make that the point at which the three spheres intersect.

J5-Delta

Delta Robot Component for Johnny-Five

Usage

```
var five = require("johnny-five");
var Deltabot = require('j5-delta');
var board = new five.Board();

board.on("ready", function() {
  var delta = new Deltabot({
    pins: [ 9, 10, 11 ]
  });

  delta.moveTo([20, -20, -150]);
  // other commands
});
```


Jason's tapsterbot project came with a JS example program to drive the bot adapted from some C code attached to the above Trossen Robotics post, but it wasn't very usable in another project as it was, so I decided to tidy it up and make some fixes. I cleaned things up and made it an npm module, made it more java-scripty and and less C like, added queueing of movements so that you could tell the robot perform a series of actions to perform in sequence. I added 'tap' and 'draw' methods. I fixed up the forward kinematics function so that you can also translate a collection of angles into an XYZ position. This is useful for figuring out where your end effector will really end up because servos aren't terribly precise. Finally it can be configured to control other kinds of delta robots. It's not up on NPM yet - I have a few things to finish up and a bit more documentation to write first, but it will be soon.

There's a usage example in the screenshot above - hopefully you can see it's not too intimidating. The coordinate system is based on the point of origin between the three motors, so the Z axis will always be negative unless something's gone horribly wrong.

Johnny-Five

The JavaScript
Arduino
Robotics & IoT Platform

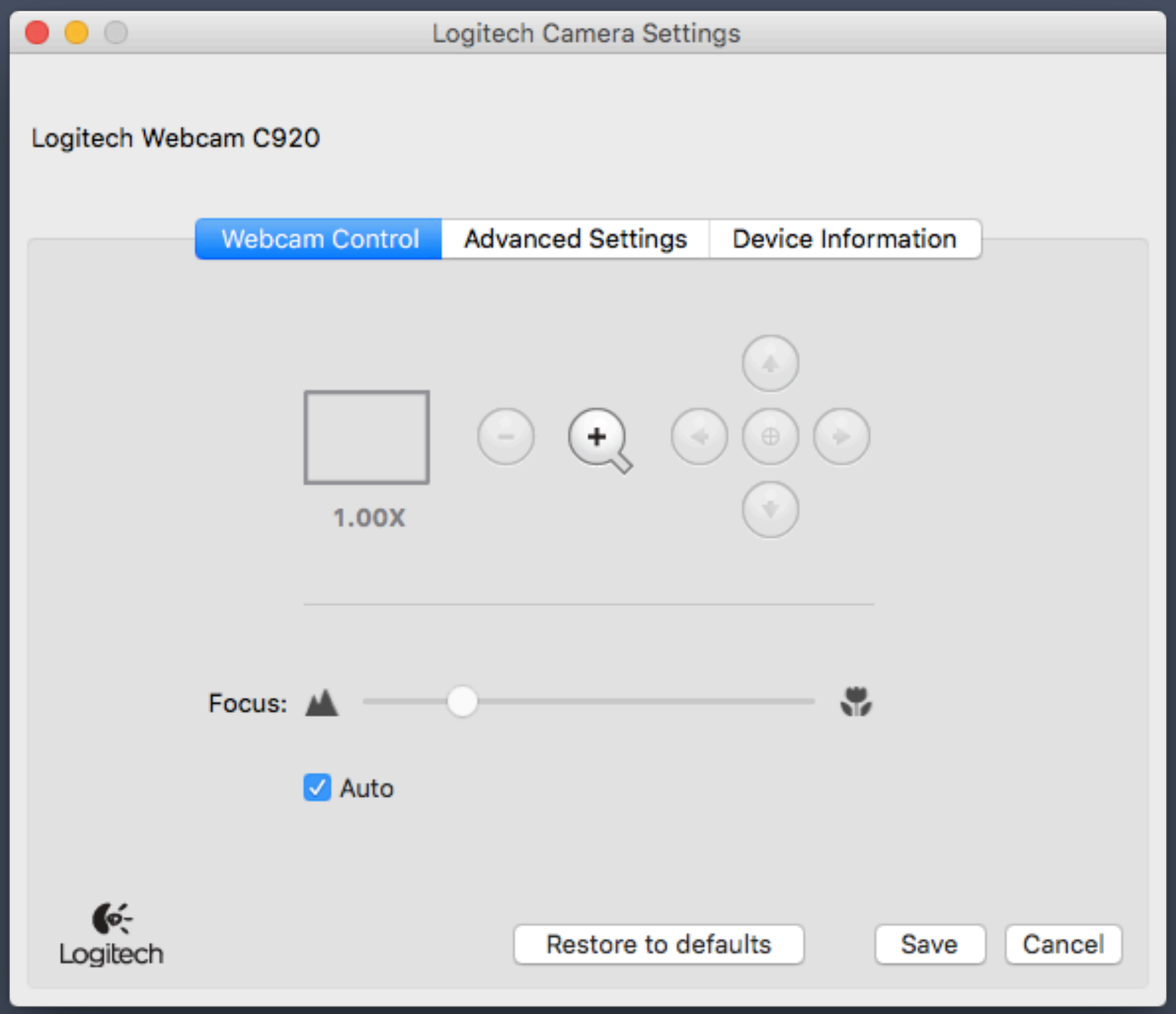
NPM INSTALL JOHNNY-FIVE

16 - MAKENAI@GMAIL.COM / TWEET @MAKENAI

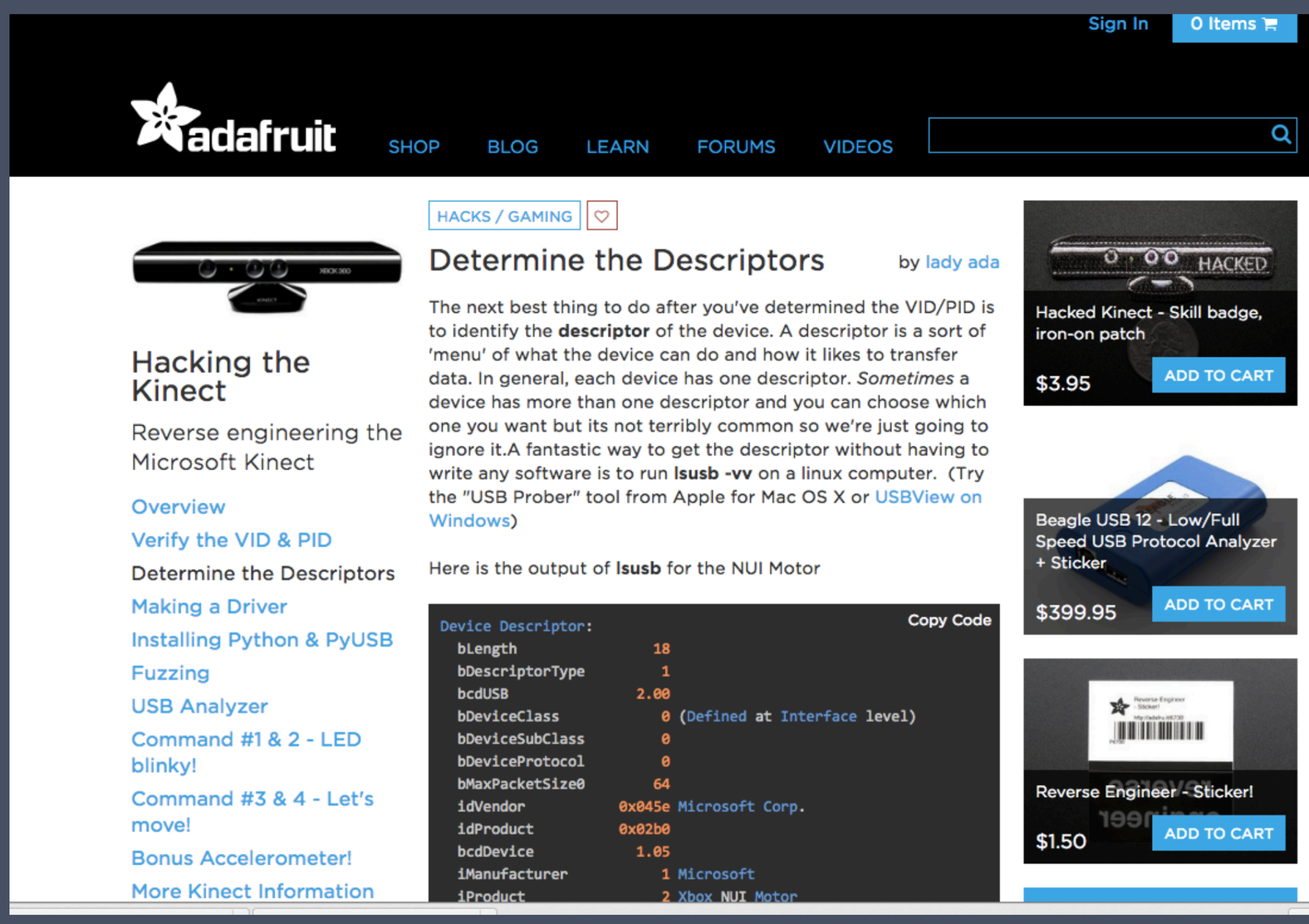
Naturally, both Jason's original program and this module are built on top of Johnny-Five, a great JavaScript library for hardware and microcontrollers of all kinds. Check it out.

Now we have a robot that can poke and swipe at things on command, but it can't see. That shouldn't be a problem - the webcam has also been invented. I had a few of these pretty decent webcams left over from a project and thought they would work fine. It turns out that most webcams autofocus, and having an end effector sweeping around in front of the screen means that you're going to have a pretty bad time keeping the screen in focus.

I did some googling around to find a webcam that supported manual focus and found this one. Seemed legit. Ordered it. Now, I've used an SLR before and I know you flip the switch to turn off autofocus, and turn the thingie to focus. This camera didn't have that - nor did any of the other ones I could find. What it did have is a little software control panel.

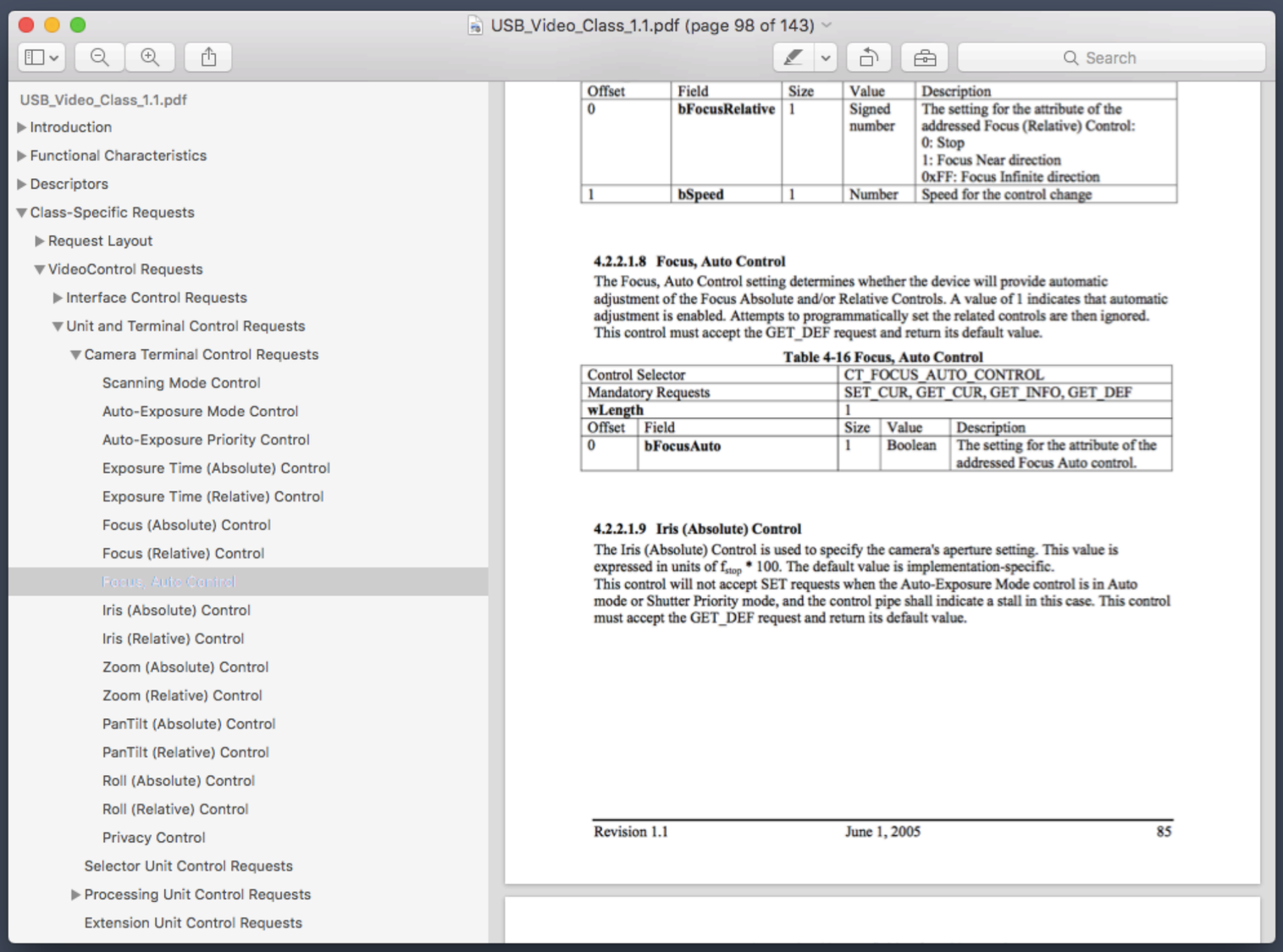


I thought it wouldn't be a problem - I could just adjust the settings before starting my program. However, it didn't remember the settings and it quickly got annoying to have to do this step every time while I was developing the rest of the code. I thought to myself "I can probably just reverse engineer this!"



HTTPS://LEARN.ADAFRUIT.COM/HACKING-THE-KINECT

I found a cool adafruit tutorial on reverse engineering and read up - it was all super useful until they get to the part about using a hardware USB analyzer which costs $$$.



Offset	Field	Size	Value	Description
0	**bFocusRelative**	1	Signed number	The setting for the attribute of the addressed Focus (Relative) Control: 0: Stop 1: Focus Near direction 0xFF: Focus Infinite direction
1	**bSpeed**	1	Number	Speed for the control change

4.2.2.1.8 Focus, Auto Control

The Focus, Auto Control setting determines whether the device will provide automatic adjustment of the Focus Absolute and/or Relative Controls. A value of 1 indicates that automatic adjustment is enabled. Attempts to programmatically set the related controls are then ignored. This control must accept the GET_DEF request and return its default value.

Table 4-16 Focus, Auto Control

Control Selector		CT_FOCUS_AUTO_CONTROL		
Mandatory Requests		SET_CUR, GET_CUR, GET_INFO, GET_DEF		
wLength		1		
Offset	Field	Size	Value	Description
0	**bFocusAuto**	1	Boolean	The setting for the attribute of the addressed Focus Auto control.

4.2.2.1.9 Iris (Absolute) Control

The Iris (Absolute) Control is used to specify the camera's aperture setting. This value is expressed in units of f_{stop} * 100. The default value is implementation-specific.
This control will not accept SET requests when the Auto-Exposure Mode control is in Auto mode or Shutter Priority mode, and the control pipe shall indicate a stall in this case. This control must accept the GET_DEF request and return its default value.

I googled a lot more, found Video4Linux and discovered that most modern webcams follow a standard called 'USB Video Class' put out by the USB implementers forum and there is a data sheet for all of the things that can be supported. Hooray! I also found simple implementation in Objective-C that was helpful.

GitHub, Inc. [US] https://github.com/makenai/node-uvc-control

uvc-control

Control a USB Video Class compliant webcam from node. Most modern USB webcams use a common set of controls standardized by the USB Implementers Forum. You can use this set of controls to change certain things on the camera, such as the brightness, contrast, zoom level, focus and so on.

Example

```
var UVCControl = require('uvc-control');

var camera = new UVCControl(0x046d, 0x082d);

camera.get('autoFocus', function(error,value) {
    console.log('AutoFocus setting:', value);
});

camera.set('brightness', 100, function(error) {
    if (!error) {
        console.log('Brightness Set OK!');
    }
});
```

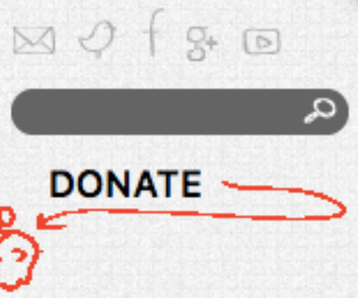
Finding Your vendorId / productId

Every USB device has a vendorId and productId. You can use Device Manager (Windows), System Information (Mac) or lsusb (Linux) to find these, or you can use the list-devices.js in this repo to find the right paramters.

```
$ node list-devices.js
HD Pro Webcam C920 [ vid: 0x46d  / pid: 0x82d  ]
Bluetooth USB Host Controller [ vid: 0x5ac  / pid: 0x828f  ]
```


So, I spent a couple of nights creating a node package for controlling UVC cameras that I could use in my project. Later on I ended up needing to use autoWhitebalance and brightness settings as well.

Now we can see clearly, but we need to interpret what we're looking at.



OPENCV (OPEN SOURCE COMPUTER VISION)

OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform. Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 9 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.

QUICK LINKS:

- Online documentation
- User Q&A forum
- Report a bug
- Build farm
- Books
- Developer site

LATEST DOWNLOADS

2015-12-21
VERSION 3.1

- OpenCV for Windows
- OpenCV for Linux/Mac
- OpenCV for Android
- OpenCV for iOS

WHAT'S NEW

2016-04-25
Itseez announces release of Accelerated CV library
Optimized specifically for ARM platform, ACV

2016-03-03
OpenCV in GSoC 2016!
Thank you Google for helping us out for the 6th year in a row! Awesome students will help make

2016-01-29
Adaptive Vision Studio 4.3 Lite
This data-flow based software makes it possible to prototype OpenCV

2015-12-21
OpenCV 3.1
Merry coming XMas & Happy New Year! OpenCV 3.1 has been just released, with lots of new

OpenCV seems to be the premier package people go to for solving computer vision problems. It was developed to speed the progress of computer vision research by providing a common and optimized base library of the most common computer vision algorithms. It has support for facial recognition, motion tracking, robot vision, gesture recognition and more. A lot of bells and whistles.

HuMoments

Calculates seven Hu invariants.

C++: void **HuMoments**(const Moments& **m**, OutputArray **hu**)

C++: void **HuMoments**(const Moments& **moments**, double **hu**[7])

Python: cv2.**HuMoments**(m[, hu]) → hu

C: void **cvGetHuMoments**(CvMoments* **moments**, CvHuMoments* **hu_moments**)

Python: cv.**GetHuMoments**(moments) → hu

Parameters:
- **moments** – Input moments computed with **moments()** .
- **hu** – Output Hu invariants.

The function calculates seven Hu invariants (introduced in [Hu62]; see also http://en.wikipedia.org/wiki/Image_moment) defined as:

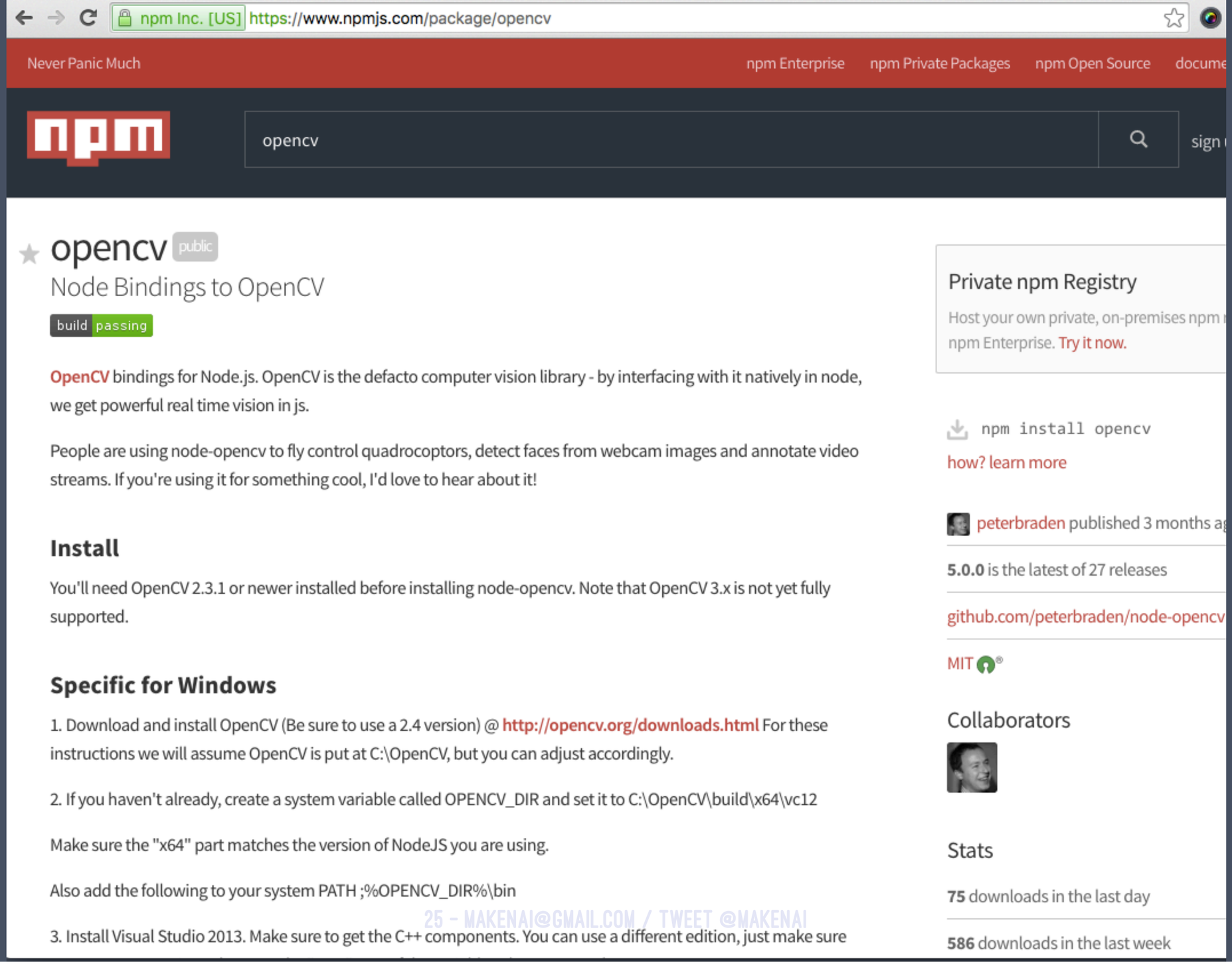
$$
\begin{aligned}
hu[0] &= \eta_{20} + \eta_{02} \\
hu[1] &= (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \\
hu[2] &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
hu[3] &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
hu[4] &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
hu[5] &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
hu[6] &= (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned}
$$

where η_{ji} stands for `Moments::nu`$_{ji}$.

These values are proved to be invariants to the image scale, rotation, and reflection except the seventh one, whose sign is changed by reflection. This invariance is proved with the assumption of infinite image resolution. In case of raster images, the computed Hu invariants for the original and transformed images are a bit different.

See also: **matchShapes()**

There is a lot of documentation for it, but it's pretty daunting if you don't already have a background in computer vision and if you think math formulas may as well be Elvish runes. This was one of the methods I was looking at for image recognition - basically, it assigned a unique-ish number score to the average of the pixel intensities in an image. I think.



OpenCV is written in C++ natively, so I found a nice looking opencv wrapper for node. It didn't have great documentation though, so whenever I was trying something I relied on the C++ docs and then would come back to see what was possible with the wrapper. I don't think this is an uncommon pattern for a library wrapper like this. But..

Matrix.cc

```
2291
2292  NAN_METHOD(Matrix::GetPerspectiveTransform) {
2293    Nan::HandleScope scope;
2294
2295    // extract quad info
2296    Local<Object> srcArray = info[0]->ToObject();
2297    Local<Object> tgtArray = info[1]->ToObject();
2298
2299    std::vector<cv::Point2f> src_corners(4);
2300    std::vector<cv::Point2f> tgt_corners(4);
2301    for (unsigned int i = 0; i < 4; i++) {
2302      src_corners[i] = cvPoint(srcArray->Get(i*2)->IntegerValue(),srcArray->Get(i*2+1)->IntegerValue());
2303      tgt_corners[i] = cvPoint(tgtArray->Get(i*2)->IntegerValue(),tgtArray->Get(i*2+1)->IntegerValue());
2304    }
2305
2306    Local<Object> xfrm = Nan::New(Matrix::constructor)->GetFunction()->NewInstance();
2307    Matrix *xfrmmat = Nan::ObjectWrap::Unwrap<Matrix>(xfrm);
2308    xfrmmat->mat = cv::getPerspectiveTransform(src_corners, tgt_corners);
2309
2310    info.GetReturnValue().Set(xfrm);
2311  }
```


It turned out that almost everything I wanted to do was possible, but not documented. The function signatures were often very different than the in the original, so I spent a lot of time in the wrapper source trying to figure out how to use everything. I now have a TODO to come back and contribute some more documentation to this project.

The first vision problem to solve has to do with image skew. When you put the webcam off on the side of the robot to afford the best view of the screen, it's not nice and rectangular like you're used to.

```
var cv = require('opencv');

var corners = [ 1347, 310,  // Upper Left
                1815, 463,  // Upper Right
                530, 770,   // Lower Right
                399, 448 ]; // Lower Left

var newCorners = [ 0, 0,
                   1080, 0,
                   1080, 1920,
                   0, 1920 ];

cv.readImage("in.jpg", function(err, image) {
  var matrix = image.getPerspectiveTransform( corners,  newCorners );
  image.warpPerspective(matrix, 1080, 1920, [255, 255, 255]);
  image.save('out.jpg');
});
```


It turns out that this is pretty easy to fix. You tell OpenCV to calculate a deformation matrix, giving it your 4 old coordinates, and 4 new coordinates and it stretch everything into the right dimensions for you.

Testing Stuffs

Close

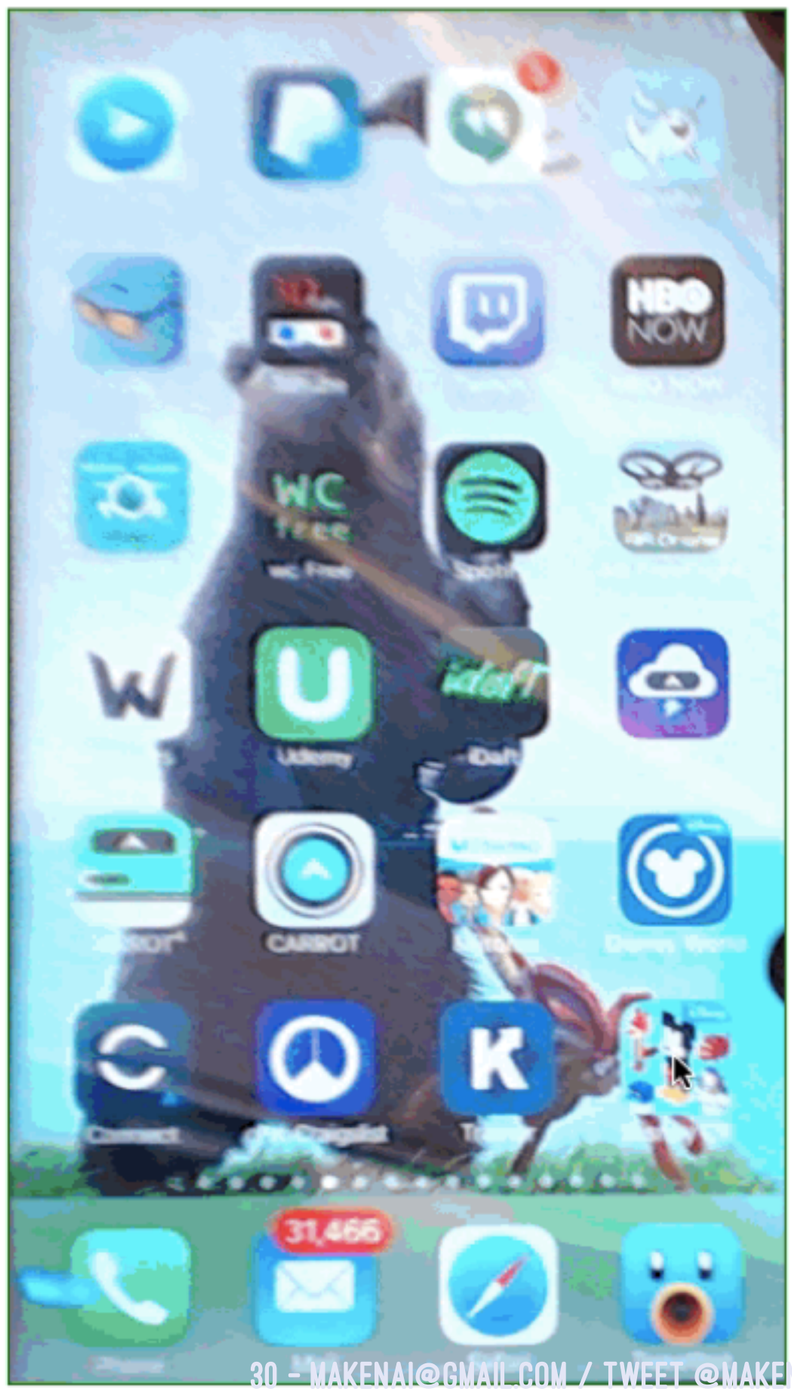
Reset Selection

Screen Dimensions: 1080 x 1920

Focus:

(34)

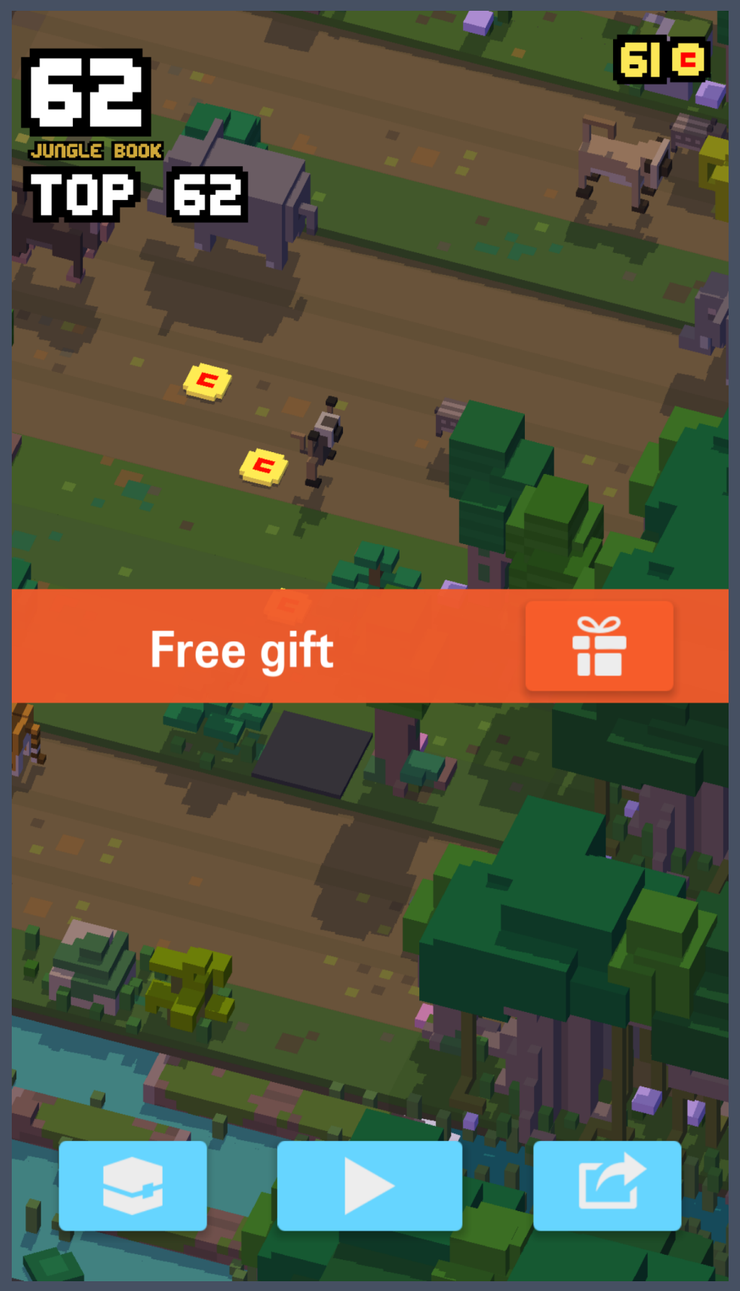
I spent a lot of time trying to detect the screen boundaries automatically using some of OpenCV's contour and line detection functions. In the end, I couldn't get it to work reliably so I took a shortcut and wrote a jQuery plugin with 4 draggable corners. If I can figure out how to make it generic enough, I'll probably package that one up.



This got me far enough to put together a demo app to control the phone from the browser. There is an express server on the back end that is capturing and deskewing frames from the webcam using OpenCV and pushing the resulting frames to a motion jpeg stream. It listens for Click and Draw events from the client over a socket.io connection and relays the commands to Johnny-Five and j5-delta for the robot to execute.

```
var camera = new cv.VideoCapture(0);
var imageEvents = new events.EventEmitter();
var cameraInterval = setInterval(function() {
  camera.read(function(err, image) {
    imageEvents.emit('rawImage', image);
    if (clientConfiguration) {
      deskewImage(image, clientConfiguration, function(newImage) {
        imageEvents.emit('processedImage', newImage);
      });
    }
  });
}, 250);
```

```
// MJPEG header
res.writeHead(200, {
  'Content-Type': 'multipart/x-mixed-replace; boundary=next',
  'Cache-Control': 'no-cache',
  'Connection': 'close',
  'Pragma': 'no-cache'
});

// Send an image every time we get the requested event
var sendImage = function(image) {
  res.write("--next\r\n");
  res.write("Content-Type: image/jpeg\r\n");
  res.write("Content-Length: " + image.length + "\r\n");
  res.write("\r\n");
  res.write(image.toBuffer(), 'binary');
  res.write("\r\n");
};
imageEvents.on( 'processedImage', sendImage );

// When the client disconnects, unsubscribe from the event emitter.
res.connection.on('close', function() {
  imageEvents.removeListener(eventName, sendImage);
});
```



Next, we want to be able to identify buttons automatically. As it happens, OpenCV has a lot of ways of doing this. Some, such as image moments, SURF and SIFT are very resilient to rotation and scaling. In our case, we are already deskewing the image and can expect it to be at a predictable scale and orientation, so we can use a simpler method - the template match. First we need to grab a sample of the part of the image we are trying to locate. That's the 'play' button on the right.

```
var match = image.matchTemplate('play.png', 3); // TM_CCORR_NORMED = 3
```


The code to initiate a template match looks like this. The first parameter is the image sample we are looking for. It's a little odd that it's taking a filename instead of an image. The second parameter is a match method. It should probably be defined as a constant somewhere in node-opencv but it's not at the moment.

a. **method=CV_TM_SQDIFF**

$$R(x,y) = \sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2$$

b. **method=CV_TM_SQDIFF_NORMED**

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') - I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

c. **method=CV_TM_CCORR**

$$R(x,y) = \sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))$$

d. **method=CV_TM_CCORR_NORMED**

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

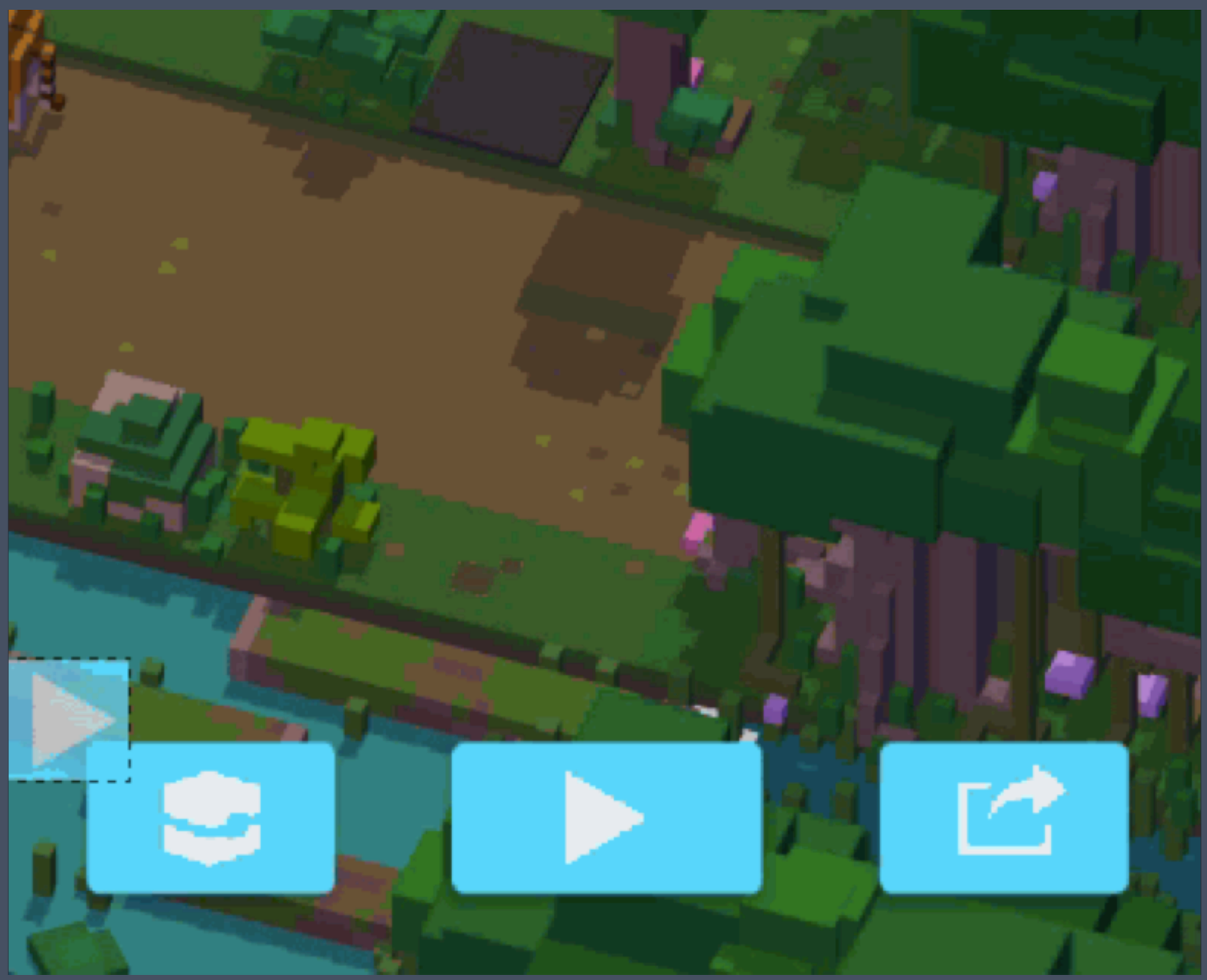
e. **method=CV_TM_CCOEFF**

$$R(x,y) = \sum_{x',y'} (T'(x',y') \cdot I(x+x',y+y'))$$

where

$$T'(x',y') = T(x',y') - 1/(w \cdot h) \cdot \sum_{x'',y''} T(x'',y'')$$
$$I'(x+x',y+y') = I(x+x',y+y') - 1/(w \cdot h) \cdot \sum_{x'',y''} I(x+x'',y+y'')$$

If you're curious what the possible match methods are and what they do, don't worry - the OpenCV documentation has you covered! Or you could do what I do and try them all out until you find one that seems to work OK.



What the templateMatch method actually does is pretty simple. It takes your sample image and slides it along the entire target image, running the match function on every pixel position. The output is a grayscale image where each pixel's intensity is the result of the match method.

```
var minMax = match.minMaxLoc();
if ( minMax.maxVal > 0.998 ) {
  console.log('Yay, a button at', [ minMax.maxLoc.x, minMax.maxLoc.y ]);
}
```


To find the position of the button, we then need to find the maximum or minimum value. For some reason, for the SQDIFF methods, smaller values are better. For everything else, larger are better. There's a convenience method that will get the maximum and minimum values. Finally, we look at the value and compare it to an arbitrary threshold to see how close the match was. If it passes, we can decide there's a button!

If you don't set a threshold, you're going to get a match every time and end up with a bunch of false positives.

Next, we need to add some logic. We can't just click every button we recognize. We need to click on the right buttons at the right time. I didn't even try writing this with ifs and thens, as I was pretty sure that it would become a mountain of spaghetti very quickly. Did you know that Flying Spaghetti Monster and Finite State Machine have the same acronym?

Even though I had never used one before, I was pretty sure that I needed a state machine.

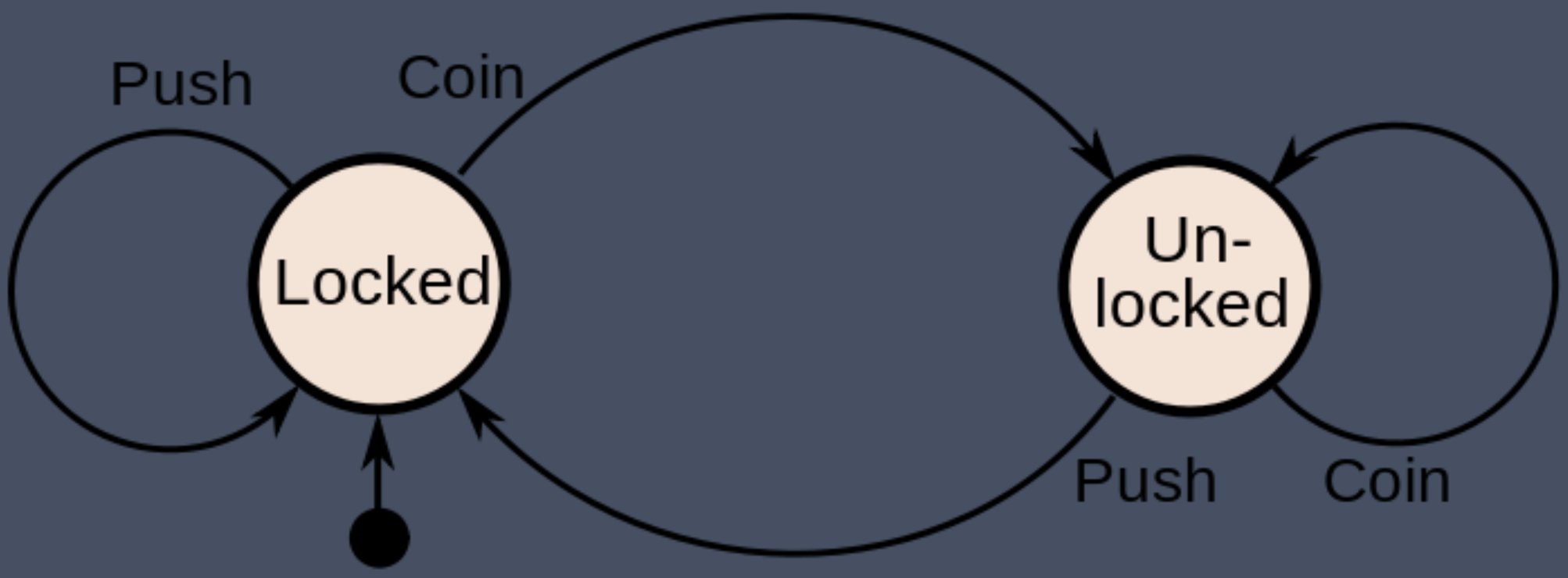
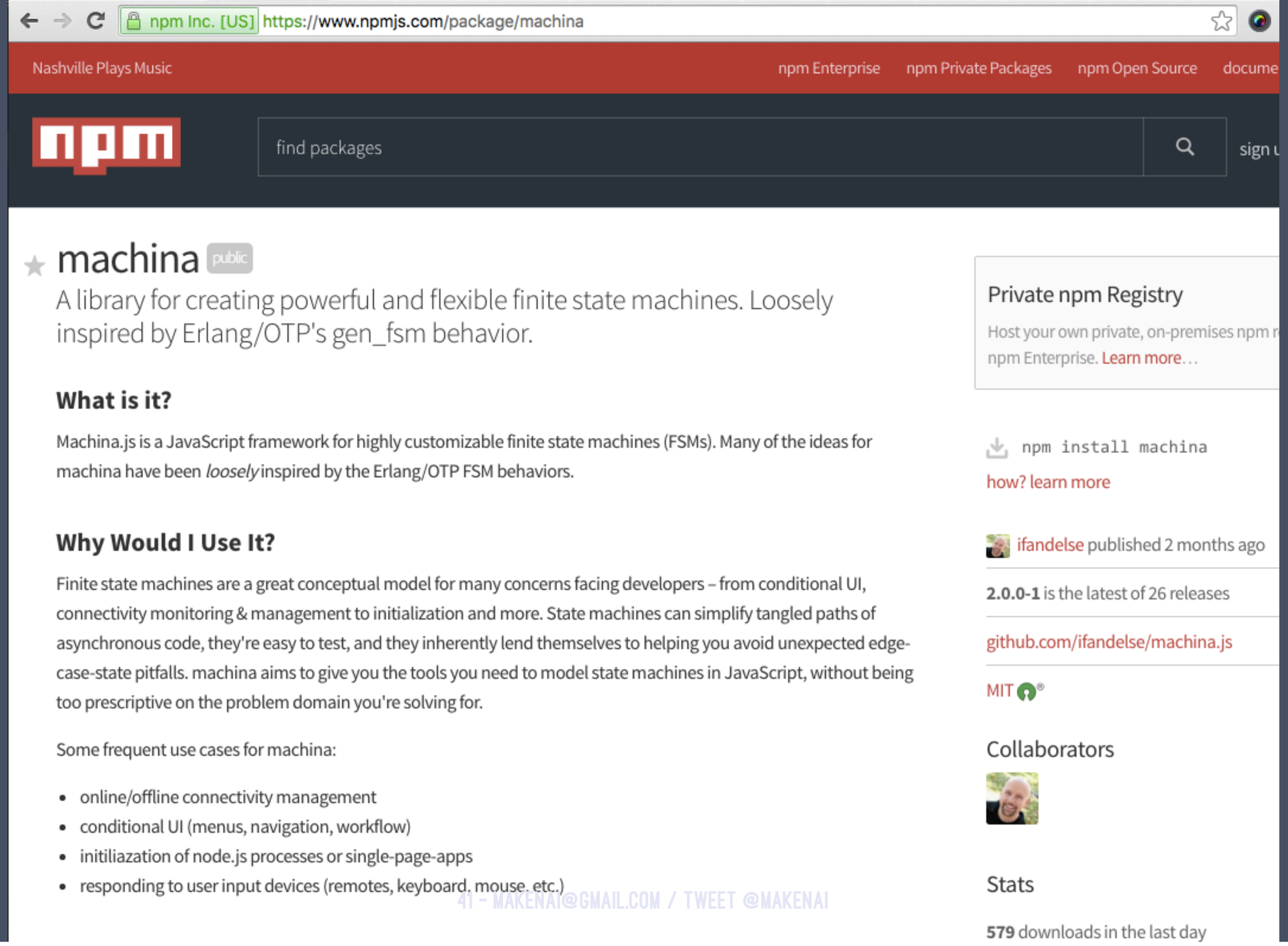


IMAGE: WIKIPEDIA [6]

[6] HTTPS://EN.WIKIPEDIA.ORG/WIKI/FINITE-STATE_MACHINE

This might be a bad explanation, but.. A finite state machine is kind of a conceptual model that dictates how something will behave depending on what state it's in. Various inputs can trigger transitions to a different states in a specific sequence, which defines the machines behavior.

The image here is a state machine diagram representing a coin operated turnstyle. You can push on it all you want, but if it's in the locked state nothing will happen. Inserting a coin will transition it to the unlocked state, where the same action will now cause the turnstyle to let you through before returning to the locked position. We can use this to keep our bot on track by reacting to only expected inputs based on our state, and ignoring things that don't matter at that time.



Machina is a really nice FSM implementation by Jim Cowart.
Its API has event hooks and input handlers that are very useful.

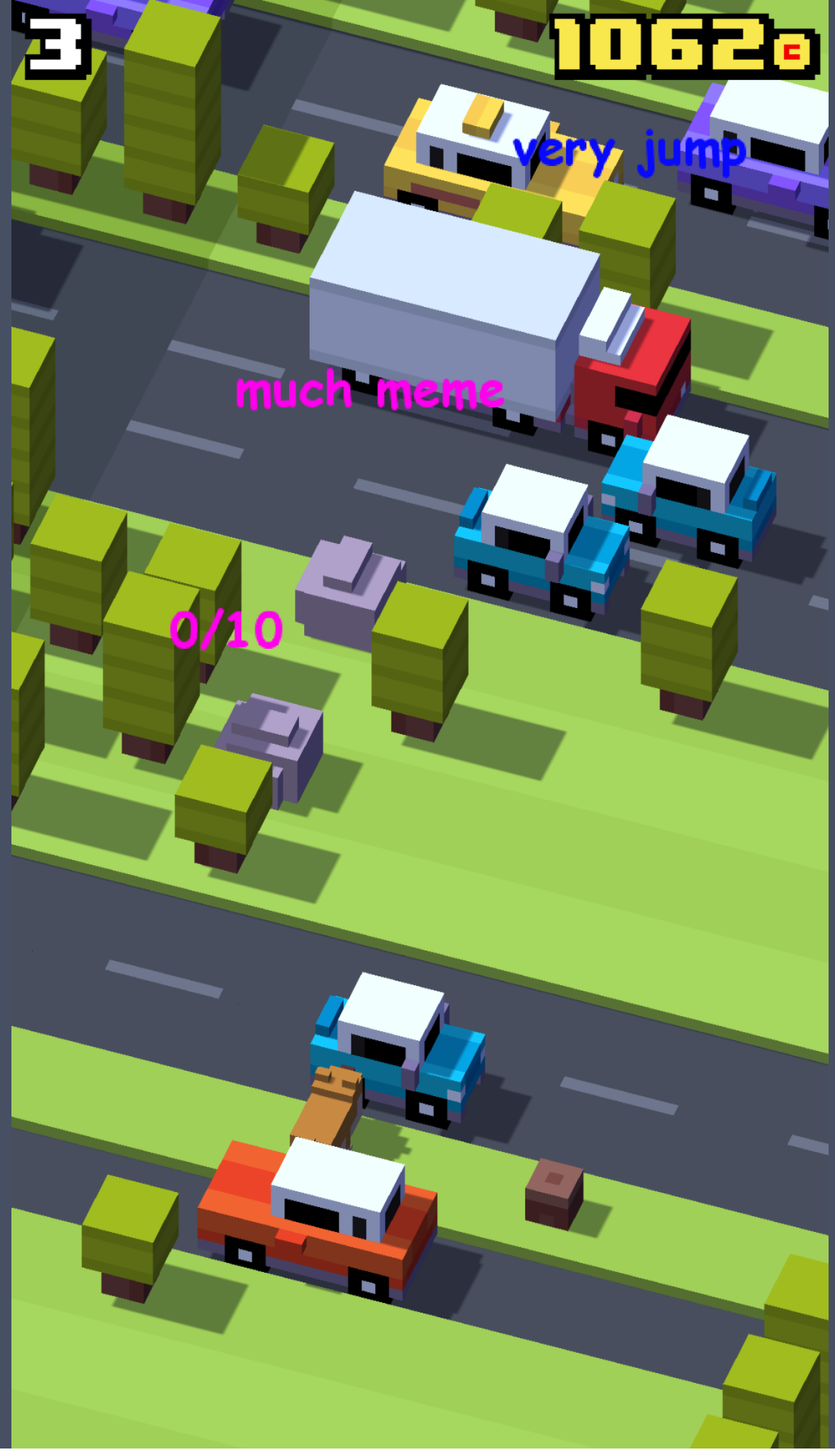
```
var stateMachine = new machina.Fsm({

  initialState: 'titleScreen',

  foundButton: function(button) {
    this.handle(button.name, button);
  },

  states: {
    titleScreen: {
      play: function() {
        // Do some things
        this.transition('anotherState');
      }
    },
    anotherState: {
      // ...
    }
  }

});

var buttons = detectButtons(image);
buttons.forEach(function(button) {
  stateMachine.foundButton(button);
});
```


This is the basic syntax for creating a machina state machine. foundButton is just a conveneince method to call an input handle. titleScreen and anotherScreen are two different states. If a button named 'cat' was found and titleScreen didn't define a handler for it, nothing would happen.

```
titleScreen: {

  finger: function(button) {
    device.tap(500,500, function() {
      this.transition('playing');
    }.bind(this));
  }

},
```

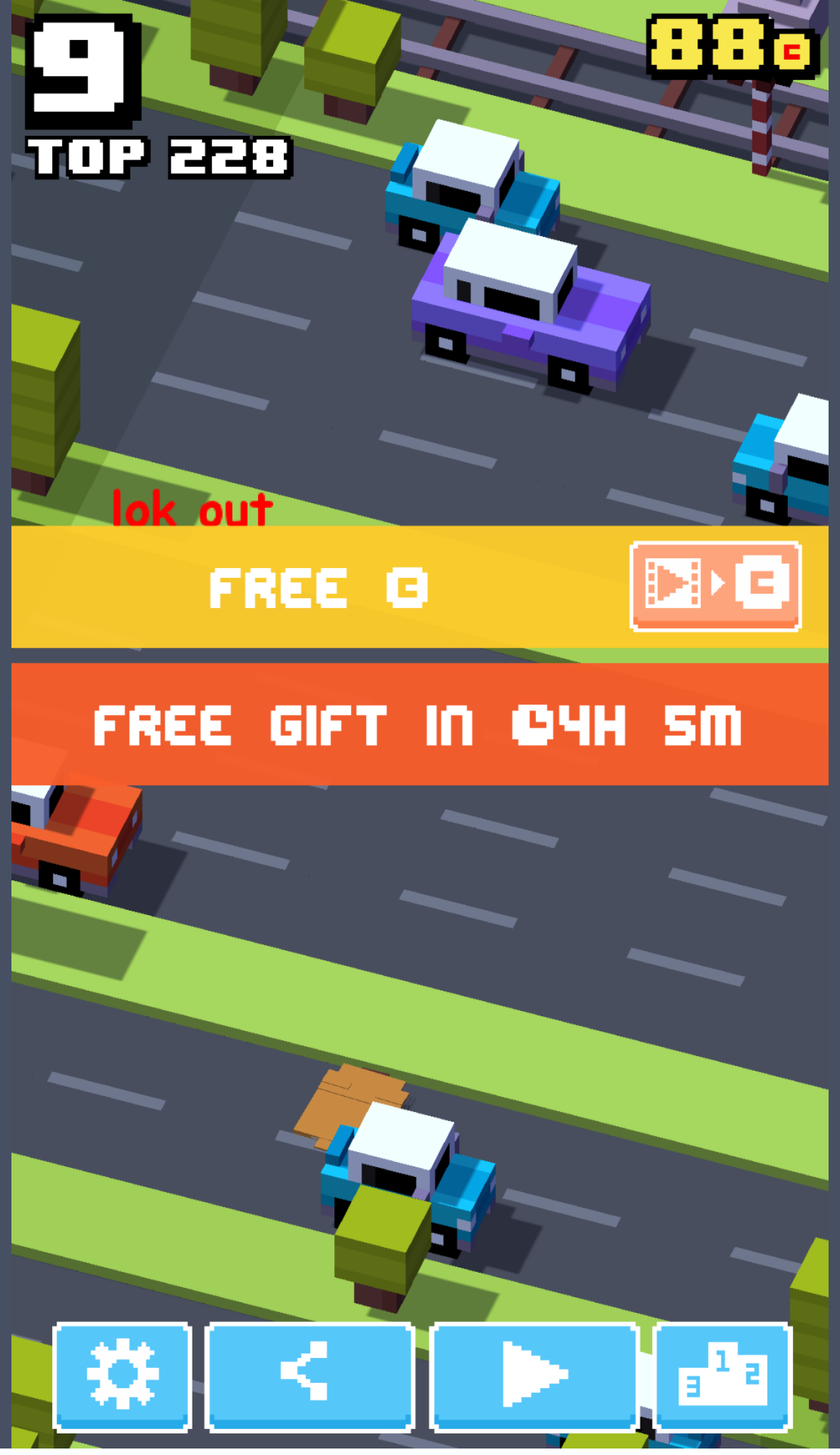
The first state is looking at the title screen. Here we just want to tap somehwere to start the game.

```
playing: {
  _onEnter: function() {
    this.timer = setInterval(function() {
      device.tap(500,500);
    }.bind(this), 2000);
  },

  _onExit: function() {
    clearInterval(this.timer);
  },

  play: function() {
    this.transition('gameOver');
  }
},
```



Now we are playing the game. Since programming the logic is pretty hard, we just keep hopping until death.

```
gameOver: {

  earn: function(button) {
    device.tapButton(button, function() {
      this.transition('watching');
    }.bind(this));
  },

  play: function(button) {
    device.tapButton(button, function() {
      this.transition('titleScreen');
    }.bind(this));
  },

}
```

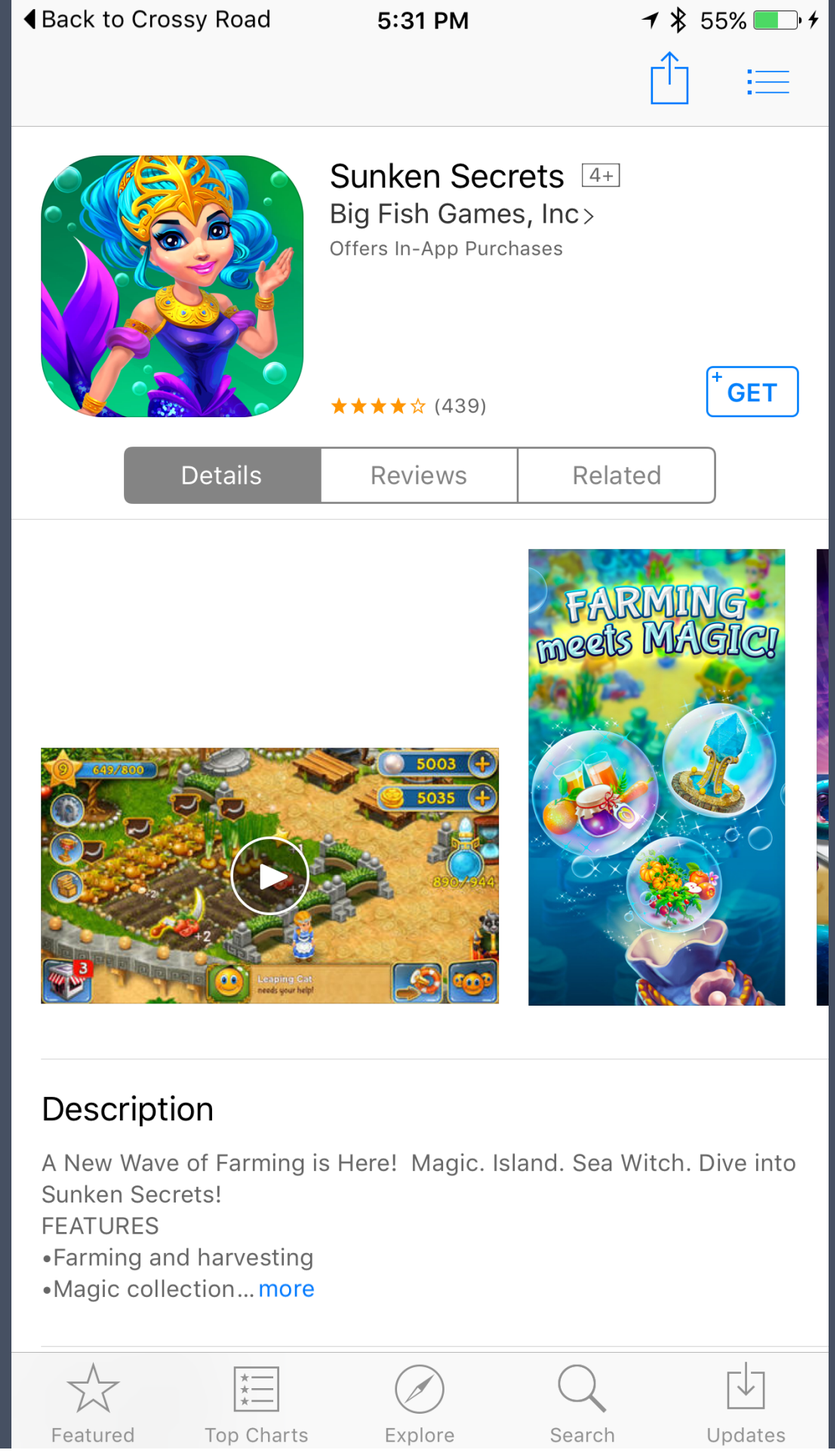



Oops, we died! There is a chance we will get a popup to earn some coins by watching a commercial, so if we see that tap it. Otherwise, tap the play button to play again.

```
watching: {

  _onEnter: function() {
    setTimeout(function() {
      this.handle('adOver');
    }.bind(this), 25000);
  },

  adOver: function() {
    device.tap(500,500, function() {
      device.tap(100,0, function() {
        this.transition('gameOver');
      }.bind(this));
    }.bind(this));
  }

},
```

Now we're watching a commercial. We just wait 25 seconds or so until the ad is done. Since the X target is really hard to detect and tap on accurately, I just tap anywhere to go to the APP store.

```
watching: {

  _onEnter: function() {
    setTimeout(function() {
      this.handle('adOver');
    }.bind(this), 25000);
  },

  adOver: function() {
    device.tap(250,250, function() {
      setTimeout(function() {
        device.tap(50,0, function() {
          this.transition('gameOver');
        }.bind(this));
      }.bind(this), 5000);
    }.bind(this));
  }

},
```

Finally, when we're in the app store, we just tap in the upper left corner to go back to the game and collect money.

DEMO?