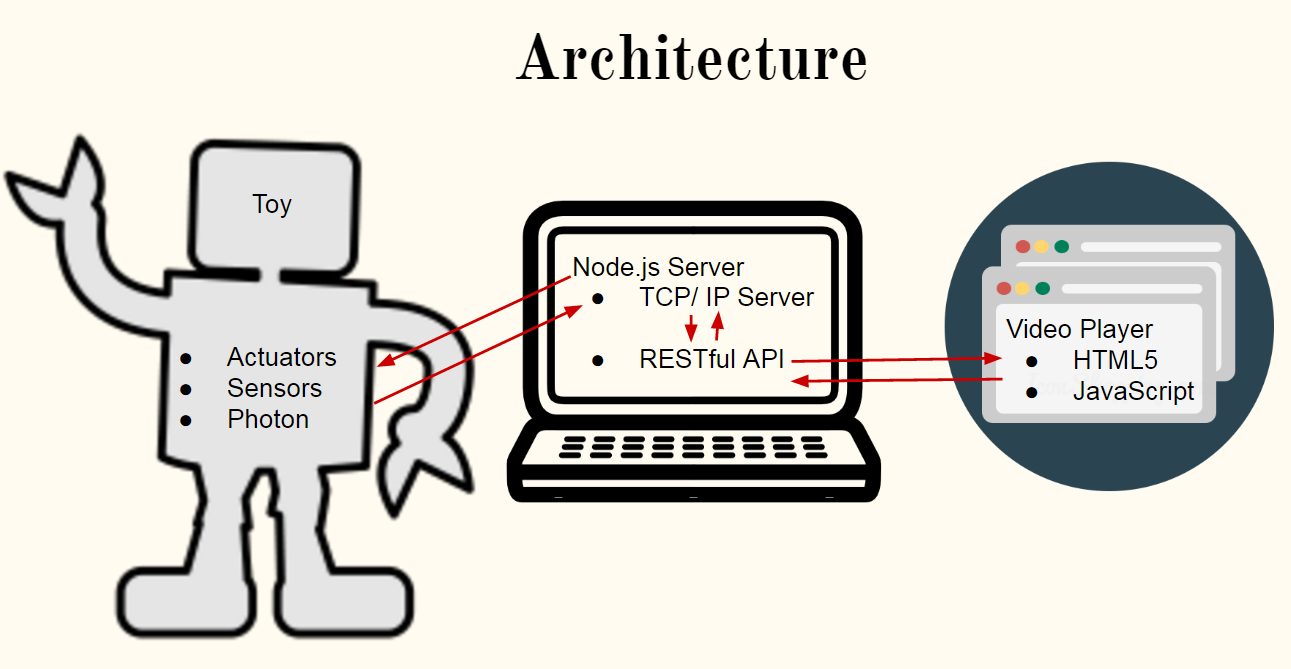
**MITE - Lab Assignment**

* **Ryan Spicer, Nichin Sreekantaswamy, Archana Ramachandran**

**Objective:**

The goal is to design an interactive toy which forms a closed loop system with the media displayed on the television. The viewer is required to perform certain actions and actuations with/on the toy in order to potentially decide the sequence of clips that will play on the TV. The toy also will have some form of actuation that occurs corresponding to the content being shown. The viewer gets to decide how an episode will play out and also be more physically involved with their favorite characters. This makes watching television more entertaining and a fun activity, unlike the existing linear playback format.



# Week 1-2: Hardware Integration with Photon (TA: Archana)

Goals:

1. Become familiar with Photon, various sensor modules and wireless communication
2. Establish a wireless connection between the Photon and the TCP/IP server in order to have a two way communication link to communicate the sensor acquired inputs and output signals for the actuators.

**Pre-Lab**

You will need to use the Particle Photon online compiler software for this lab. Follow the steps below in

the PRE-LAB section to complete the pre-lab. Make sure to do this before lab.

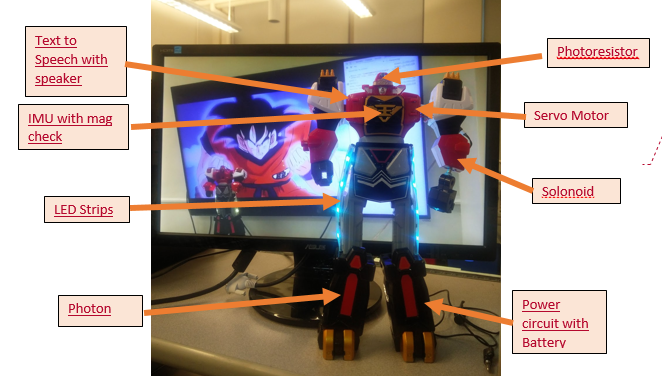
Create an account with Particle at [Particle Build.](https://build.particle.io/) For more information: <https://docs.particle.io/guide/getting-started/build/photon/>

Install the Particle CLI and claim your device: <https://docs.particle.io/guide/getting-started/connect/photon/>

Flash the “Blink an LED” example to your Photon to verify you are able to flash successfully.

**Part 1: Hardware connections**

A significant part of this lab is figuring out how to combine many off-the-shelf components inside of a standard toy. The image below shows how the MITE team fit everything into their toy.



As an Embedded Engineer, it’s important that you’re able to figure out how to use lots of devices with your microcontroller. The main goal of this section of the lab is to figure out how all of these devices work, and how to connect them to the Photon.

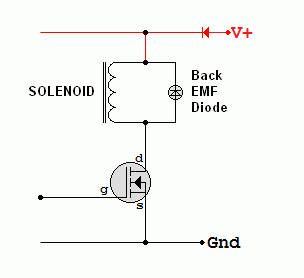
**These are the components you need to integrate:**

**Photoresistor:** This connects to Pin A0 of the Photon, and is used to detect if the toy’s face has been covered up. In the toy’s firmware, the photoCheck() function returns a 1 when the sensor is covered up. Look at the condition for this to be true, and figure out how the photoresistor should be connected.

**Vibration motor:** This provides haptic feedback for the toy, and can be fastened on virtually any internal surface. The motor is activated when Photon pin D2 goes high. The pin cannot provide enough current to run the motor. Using a MOSFET, come up with a simple circuit to drive the motor.

**Servo:** The servo allows the arm to rotate. Depending on the servo you have available, you may run it either off 3.3V or 5V. The servo is controlled by pin D3. Figure out how to secure the servo inside the toy and then connect it to the toy’s original arm. We have provided a 3D model of an adapter to print to connect the arm to our original servo.

**Solenoid:** The solenoid attaches to the toy’s fist, to mimic a punching action. Take apart the arm (the servo arm, not the normal one) and find a way to insert the solenoid between the forearm section and the fist. The solenoid draws a large amount of current, and can produce damaging back EMF during switching. Build a controller circuit such as the one shown below, and attach it to the solenoid and pin D7 of the Photon. Be careful with the diode polarity!



**IMU:** The IMU is responsible for motion sensing, and detecting magnetic fields. The MITE team used the Adafruit 10DOF IMU breakout, but yours may be different. Make connections to your IMU for power and I2C (Photon pins D0, D1), and mount it in the toy’s chest. Add some magnets to the chest piece so the IMU can see its field when it gets close.

**Power supply:** The toy needs a +5V regulated power supply. Using a linear regulator and a 9V battery, build a small supply and install it in one of the legs. If you have time, mount a power switch on the back of the leg to shut the supply off.

**Text to Speech module:** The text to speech module (EMIC2) communicates with the Photon over the serial TX/RX lines, and has a small audio amplifier built in. Connect the module to the Photon and a speaker, and install it in the toy. **Extra credit:** make your module louder by amplifying the audio from the TTS module.

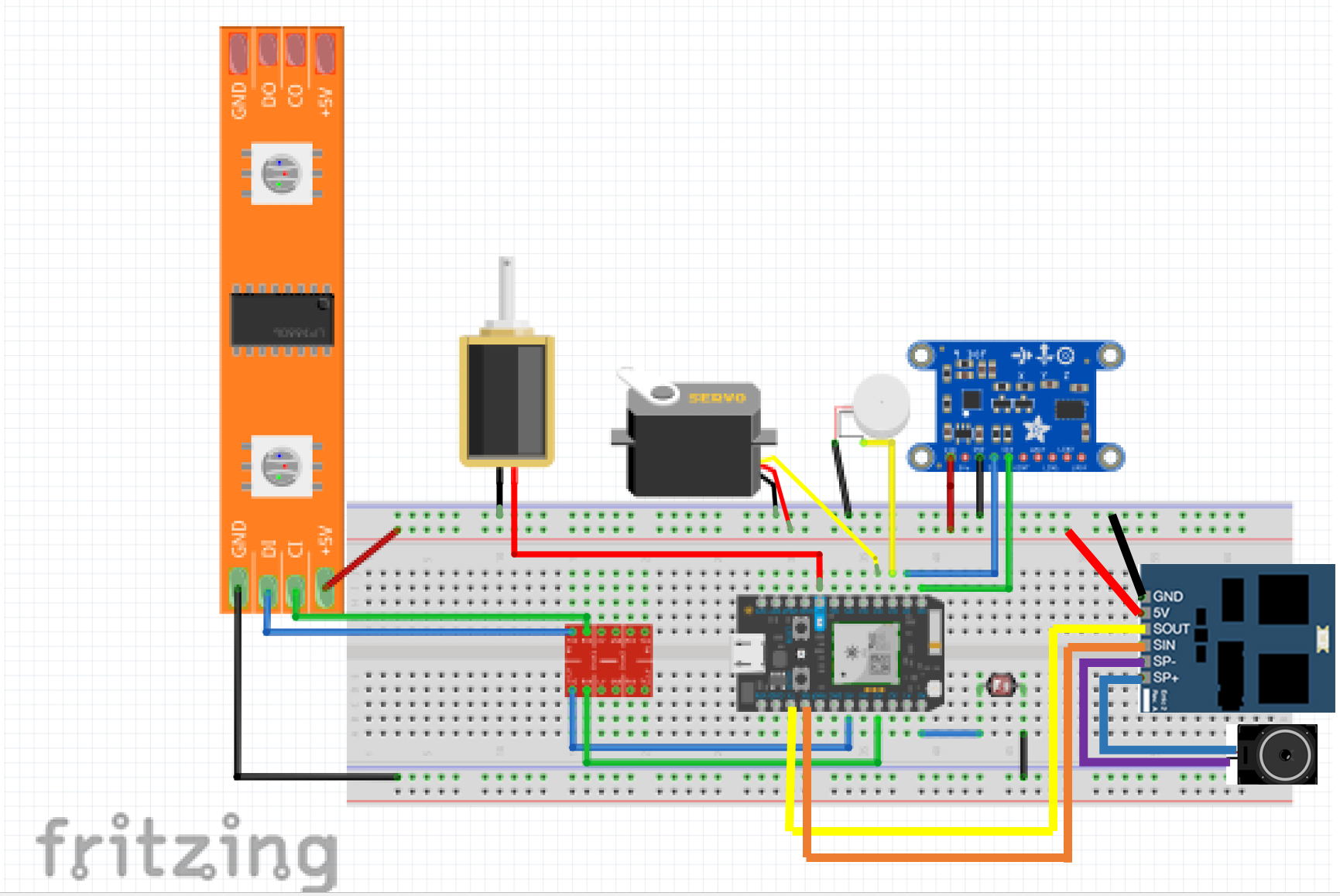
**LED strips:** A pair of LED strips runs down the side of each leg. These use a pseudo-SPI connection for control, and run off +5V power. One strip will connect to the Photon and run up one leg, while the other strip will be daisy chained off of the first strip. Ensure that signals flow from the “1” end of the strip to the “0” end of the strip. Since these are 5V devices, and the Photon uses 3.3V logic, a level shifter IC between the Photon and strips is necessary. Via a pair of TX connections on the level shifter, connect the first strip’s C1 line to Photon A3 (SCK) and the D1line to Photon A5 (MOSI).

The MITE team found that we needed more space in the toy than was available. You may need to carve out sections. We removed a significant part of the upper legs and covered it with a pair of plates, which you may laser cut from acrylic as we did. This part, as well as the servo adapter, can be found on our Github at <https://github.com/ese-519/MITE/tree/master/labAssignment>

**Summary of Photon signal pins:**

|  |  |
| --- | --- |
| **Photon Pin** | **Device Pin** |
| D0 | IMU SDA |
| D1 | IMU SCL |
| D2 | Vibration control |
| D3 | Servo control |
| D7 | Solenoid control |
| A0 | Photoresistor signal |
| A3 | LED strip C1 |
| A5 | LED strip D1 |
| TX | EMIC2 RX |
| RX | EMIC2 TX |

**Basic device connections (this is not the real schematic! You have to design parts of it as noted above!)**



**Part 2: Firmware**

We have provided the firmware for the toy in our Github repo: <https://github.com/ese-519/MITE.git>

Download the **toyclient** directory, and flash it using the Particle compiler.

To verify that your hardware is working, rewrite the main loop as you see fit to call various functions.

Demonstrate that each piece of hardware is working for credit.

# Week 2: TCP Server with RESTful API (TA: Nichin)

Goals:

1. Get started with Node.js and host a simple TCP server to receive and forward TCP traffic on different ports.
2. Add a RESTful API feature to the server, to effectively replace one side of the TCP communication with an exposed API that can respond to HTTP GET and POST calls from a web page.

Pre-lab:

This part is software intensive. For those who don’t have Node.js setup on their laptop in their preferred operating system, please follow the following links. Also, it would be great to have cURL tool on your laptop to be able to test the system easily. cURL is usually already present on Mac and Linux systems. The API relies on a few Node.js packages which need to be installed. Using npm or your preferred method, install the following packages: body-parser, cors, express.

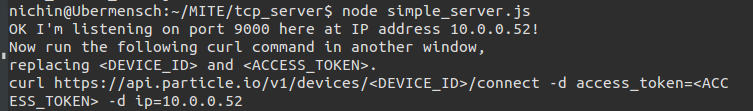
1. <https://howtonode.org/how-to-install-nodejs>
2. <https://help.zendesk.com/hc/en-us/articles/229136847-Installing-and-using-cURL>

PART-1: Setting up and testing the TCP server

1. After installing node.js, download [simple\_server.js](https://github.com/spark/local-communication-example/blob/master/simple_server.js).
2. Start the server from the command line

>> node simple\_server.js

1. Using the online Particle Build or Particle Dev, flash [firmware.cpp](https://github.com/spark/local-communication-example/blob/master/firmware.cpp) to your photon device.
2. Now you should have got a prompt similar to this that announces the IP address of the machine that node.js is running on.



1. Follow the instructions and run a command on cURL like so:

curl [https://api.particle.io/v1/devices/<device\_id>/connect\](https://api.particle.io/v1/devices/0123456789abcdef01234567/connect/)

-d access\_token=<access\_token>

-d "args=IPADDRESS"

1. The photon should be connected to the TCP server now. Test it by typing 7h or 7l on the command line and you will see the LED on the Photon turn on or off.

Explanation of what’s happening:

The node.js instance running on the machine is looking for TCP traffic on the port 9000. So it will receive any connections that are directed to it’s IP address and port 9000. The curl command uses photon’s built in API to send it the IP address and the photon is now able to connect to it.

If you take a look at the code inside the photon, there is a fairly simple code that parses the strings that the photon receives from the node.js server and controls the output at the pins.

PART-2: Understanding and integrating the RESTful API with the server.

Here we are referring to an HTTP API, which can be a way of sharing application data over the internet.

REST stands for Representational State Transfer. This is a term invented by Roy Fielding to describe a standard way of creating HTTP APIs. He noticed that the four common actions (view, create, edit, and delete) map directly to HTTP verbs that are already implemented: GET, POST, PUT, DELETE.

Use the following link to get a fundamental understanding of a RESTful API: <http://www.andrewhavens.com/posts/20/beginners-guide-to-creating-a-rest-api/>

We will now create a RESTful HTTP API on node.js to provide a simple and elegant a way for the HTML5 video player to interact with the server using just GET and POST requests.

To get you started, look at this simple code:

|  |
| --- |
| var express = require('express'); var app = express(); var fs = require("fs");  app.get('/listUsers', function (req, res) {  fs.readFile( \_\_dirname + "/" + "users.json", 'utf8', function (err, data) {  console.log( data );  res.end( data );  }); }) var server = app.listen(8081, function () {  var host = server.address().address  var port = server.address().port  console.log("Example app listening at http://%s:%s", host, port) }) |

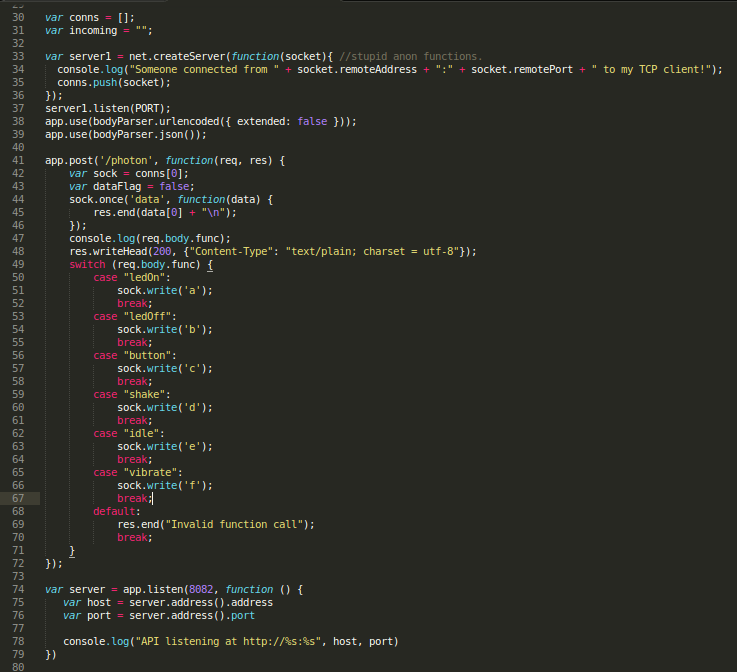
Create a JSON file with the following data in the local directory and name it users.json

|  |
| --- |
| {  "user1" : {  "name" : "mahesh",  "password" : "password1",  "profession" : "teacher",  "id": 1  }  } |

Now try to access the defined API using URL: http://127.0.0.1:8081/listUsers and HTTP Method : GET on local machine using any REST client, e.g cURL or just any web browser will do. It should show you all the data in the file. This should give you an idea of how to create a simple API endpoint on node.js.

Now we look at the full server code that uses both the TCP server and the API concepts we learnt.

Download the code from <https://github.com/ese-519/MITE/blob/master/tcp_server/RESTAPI.js>



In this code snippet, we see the TCP socket being declared first and have it announce when the Photon connects to it. We then create an API node ip\_addr:8082/photon. We only define a POST action to this URL and enable the video player to post the name of the function it wants to call in the HTTP request. The server then sends the appropriate codeword to the the photon on the TCP client in the big switch statement that is defined inside the API’s function.

# Week 3: HTML5 Video Player (TA: Ryan)

**Goals:**

1. Get familiar with JavaScript and the video metadata objects.
2. Write code to process the video metadata for actions and sensing.

**Pre-Lab**

Download the video player directory from <https://github.com/ese-519/MITE/tree/master/labAssignment/html5> to your project folder. Ensure that the “videojs” and “vids” folders are within the “html5” folder.

If you don’t already have it, download and install Google Chrome.

**Part 1: Intro to the player and initial setup**

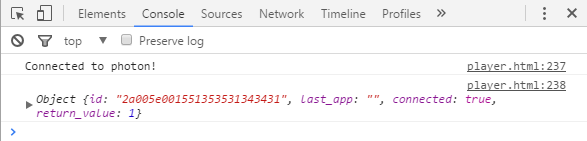
The video player runs on the Video.js framework. The framework allows for programmatic control of playback via JavaScript, as well as many advanced features implemented in plugins. We will be using videojs-overlay to write overlays on the video at specific times. The video player follows a simple scheme:

1. Start playing the first video clip
2. Send commands to the Photon to actuate and sense when the video clip’s metadata says to
3. At the end of the video, determine what clip to play next, load it, and start playback

Take some time to go over the code, ignoring the video objects for now. At the top of the script, there is a block of environment variables for you to set up. Enter the information necessary in each one. Make sure you use the correct (local) IP for your Node.js server, and get the proper device ID and access token for your Photon.

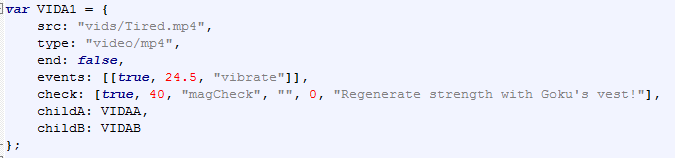
After entering the environment variables and saving, open the player in Chrome. Open the Developer Tools (Ctrl + Shift + J in Windows), start your Node.js server, and power up your Photon. Click on the “Connect to Photon” button and observe the output in the Developer Console and the Node.js window. Take a screenshot of both to demonstrate a successful connection between the player, API and Photon. Your output should resemble the examples below.





**Part 2: Video objects**

Each video clip has an object we define for it in the JavaScript. For example, this is the object for clip titled “Tired.mp4”



There are a number of fields that must be defined for each video:

*src:* Location and filename of the video

*type:* The video format

*end:* Denotes whether this video is the last in a branch of the content tree

*events****:*** A two-dimensional array of all of the actuation events that are triggered in the video

*check:* A one-dimensional array with details about the sensing to be done for the video

*childA:* The child video object for this video that is played when a “good” input is detected by the sensor

*childB:* The child video object for this video that is played when a “bad” input is detected by the sensor

The events and check arrays have similar formats. For events, the format is *[execute?, timestamp, function name]*. Execute is a boolean that defaults to *true*. If false, the player ignores that event. The timestamp tells the player when the event should fire, and the function name is the name of a function as it is registered in the API. In the above example, there is a single event that fires at t=24.5sec which tells the API to call the “vibrate” function.

For check, the format is *[execute?, timestamp, function name, argument, result, overlay text].* “Argument” is a reserved space in the array for any future sensing events that might require an argument in addition to the function name. “Result” holds the returned value from the sensor, and defaults to 0. “Overlay text” is the message overlaid on the video as a prompt to take some action when the sensing event fires.

Finish creating the metadata for the first video in the tree, VIDROOT. We will leave the events up to you. Keep in mind that Goku is getting hit a few times, and is doing a little punching of his own. The sensor event should be a “shake” at t=40sec. Be sure to include an overlay that makes sense. The “good” child video should be VIDA, and the “bad” child video should be VIDB. For credit, copy/paste your VIDROOT object and answer the following question:

*VIDROOT is the first video in the tree, but is the last object created. Why are the videos declared in reverse order?*

**Part 3: Metadata processing**

While the video plays, an event called ‘timeupdate’ is firing at regular intervals (depending on the browser - typically at least every 100 milliseconds). The code *vid.on('timeupdate', function() { });* is attached to this event, and the inline function defined within the curly braces is evaluated every time ‘timeupdate’ fires. We can use this as a way to process our video metadata in a timely fashion.

Following the prompts from the comments, write code that will fire events and sense at the right time for the current video object. A few hints:

1. You can find out the current time of the video by calling *vid.currentTime()*
2. There are *actuate()* and *sense* functions already written for you to call.
3. You only want to act on each event or sense once. Make sure they can’t be called again.

If your code works, you should be able to run through a whole sequence of clips with your toy. If you’re having trouble, pay attention to the Developer Console and make sure the API is working. For credit, demonstrate your project to a TA.