# Using AnimatLab, the OpenCM microcontroller, and the Szerial protocol

## Motivation

[AnimatLab](http://animatlab.com/Download) is a neuromechanical simulation tool written by David Cofer and Don Edwards at Georgia State University [1]. AnimatLab enables the user to construct a dynamic model of an animal or robot, and control it with a network of dynamic neurons and synapses. Neural activity determines actuator forces, and sensors determine the current in sensory neurons. In this way, the user can build and simulate closed-loop neural controllers for animals and robots.

AnimatLab can also be used to control a real robot, by specifying which items in the simulation correspond to real components. For example, the user could specify that their simulated robot’s hip joint has ID 0 on their hardware robot, enabling AnimatLab to route information for the hip joint to hardware ID 0 instead. If the user also specifies a way for the servo to send feedback to AnimatLab, then they can perform closed-loop, neuromechanical control of a real-world robot.

AnimatLab has two main ways to set up such an interface to a hardware robot, and each has unique drawbacks. The first is called [ArbotixFirmata](http://animatlab.com/Help/Documentation/Robotics/Robot-IO-Controllers/Firmata/Arbotix-Firmata). It is based on the [Firmata](https://www.arduino.cc/en/reference/firmata) protocol, which is a generic program for handling communication between a desktop/laptop (“computer” from here on out) and a microcontroller that directly controls actuators and reads from sensors (a “tosser” for short). The ArbotixFirmata Tosser is convenient to use, but assumes that the user is using [the Vanadium Labs Arbotix-M microcontroller](https://www.trossenrobotics.com/p/arbotix-robot-controller.aspx). This controller is expensive, slow, and finicky to use, and is therefore not preferred over less expensive, faster, and more robust controllers like the [Robotis OpenCM](http://www.robotis.us/opencm9-04-a-no-connectors-onboard/). Additionally, the communication protocol itself is clunky, effectively centralizing control of each actuator on the computer, rather than on the microcontroller. This requires more communication overhead, thus slowing the rate that information can be exchanged. ArbotixFirmata and the Arbotix-M were used for early work with MantisBot [2]–[5], but they were not fast or reliable enough to control all of the robot’s actuators at once.

The second way to control a hardware robot with AnimatLab is the [AnimatSerial](http://animatlab.com/Help/Documentation/Robotics/Remote-Controls/AnimatSerial) protocol. This protocol establishes a format for a serial “sentence,” that is, a temporal list of bytes, and what each position in the sentence means. The user can program any controller to interpret and send sentences that fit this format, making it very versatile. This protocol was used to control Puppy and its braided pneumatic actuators [6], [7]. The main drawback is that AnimatSerial sends data as 5-byte floating point numbers. This is not ideal, because a 5-byte floating point contains more precision than a robot can make use of. Most actuators are limited to 8- to 16-bit resolution, and thus cannot distinguish between commands of 100.1 and 100.2, for instance. Therefore, data can be sent more compactly as an integer with fewer than 5 bytes, which would shorten the sentences sent between the computer and the controller. Additionally, such discretization increases the likelihood that subsequent commands may be identical, and thus do not need to be sent, reducing the number of sentences between the computer and the controller.

I modified AnimatSerial to address these issues. It now appears as a new robot interface type, called Szerial. It enables the user to establish sparse, efficient communication between AnimatLab and any digital actuator or sensor over a serial connection. My code is written for the OpenCM microcontroller, but the Szerial.ino sketch can be modified to run on any microcontroller, and using any other actuator or sensor libraries that the user wants.

## Goals and specifications

The goal is to produce a communication system between AnimatLab and a robot that is sparse and efficient; that can be intuitively reconfigured; and is not limited to a particular hardware setup.

### Specifications:

1. Szerial shall transmit data as integer values
2. Szerial shall enable the user to use the AnimatLab GUI to establish the mapping between neural values and robot values, to increase its applicability to different actuators and sensors
   1. As a result, Szerial shall enable the mixing of actuator and sensor types on the same robot
3. Szerial shall send as many updates as possible in one sentence, to reduce communication overhead
4. Szerial shall run in the Arduino IDE, to increase its applicability to other microcontrollers

## Instructions for setting up

### Adding OpenCM to the Arduino IDE

1. Open the Arduino IDE.
2. Click File>Preferences.
3. Change the “additional boards manager url” to https://raw.githubusercontent.com/ROBOTIS-GIT/OpenCM9.04/master/arduino/opencm\_release/package\_opencm9.04\_index.json
4. Click OK.
5. Click Tools>Board>Board Manager.
6. Search for “opencm” and install the OpenCM9.04 board package. This may take 10 or 20 minutes. This will also install the DynamixelSDK library, which you will need if you plan to control Dynamixel smart servos. The AXCM library, which will be installed next, depends on these libraries.
7. Now, you should be able to use the DynamixelSDK and DynamixelWorkbench packages. For example, you can click File>Examples>OpenCM9.04>08. Dynamixel Workbench>j\_Position, and control the position of a Dynamixel servo.

### Adding the AXCM and Szerial libraries to the Arduino IDE

Any library (i.e. C++ class foo, as contained in a folder containing foo.cpp and foo.h) can be added to the Arduino IDE by pasting it into the C:\Program Files (x86)\Arduino\libraries folder. This is **not** the same as the C:\Users\xxx\Documents\Arduino\libraries folder. Clone the repository from <https://github.com/nss36/axcm.git> into your libraries folder. Similarly, clone the repository from <https://github.com/nss36/Szerial.git> into your libraries folder.

### Installing the AnimatLabSDK

The user cannot use the release version of AnimatLab and Szerial together. Instead, they must clone the SDK version at <https://github.com/nss36/AnimatLabSDK-nss36.git>. This must be cloned directly to the C:\ drive to ensure that file paths are correct. Once cloned, Download the 5 .exe files from <http://animatlab.com/Download/AnimatLab-Windows-SDK>, and run them in order, making sure they are installed to C:\AnimatLabSDK\3rdParty.

If the user only wants to run AnimatLab but not modify it, they can run AnimatLabSDK\AnimatLabPublicSource\bin\AnimatLab2.exe. You can stop reading this section.

If they wish to modify it, they must be sure to clone it directly to C:\, and then install Microsoft Visual Studio with VS2010 compiler tools. First, download and install Visual Studio 2017 Community Edition (totally free). When you open C:\AnimatLabSDK\AnimatLabPublicSource\AnimatLabCode.sln, it will ask you if you want to “retarget” the solution. Say no. If you try to build the solution, 7 of the files will fail, because it is trying to use the most recent .NET libraries, but they aren’t compatible.

To fix this, download and install Visual Studio 2010 VC Express from this page:

<https://my.visualstudio.com/Downloads?q=visual%20studio%202010&wt.mc_id=o~msft~vscom~older-downloads>. Installing VS2010 anywhere on your computer will install the necessary older “build tools,” which will enable VS2017 to build AnimatLab.

To install, open the solution file AnimatLabSDK/AnimatLabPublicSource/AnimatLabCode.sln with Visual Studio 2017. You should be able to load that solution and then do a rebuild all with no errors. Next you will probably need to setup the debugging options because those do not carry over to new computers. Make sure the AnimatLabGUI project is set as the startup project, right click it and go to properties.

1. Select the "Debug" tab and change it to "Start external program."
2. Choose the "AnimatLab2.exe" from the bin folder as the application.
3. Compile the code, and then switch to "Debug\_Double" configuration and make sure that bullet is compiled with this as well.

You should now be able to run and debug the application.

If you still get errors, it may be because you do not have boost 1.54 or later installed on your system, or do not have BOOST\_ROOT configured in your environment variables and pointing to the root location of a boost 1.54 or later installation for the code to compile. You can rectify this by adding an environment variable in Windows 10 or Windows 8:

1. In Search, search for “environment variables” and then select “Edit the system environment variables.”
2. Click the Advanced tab.
3. Click Environment Variables. In the section System Variables, find the PATH environment variable and select it. Click Edit. If the PATH environment variable does not exist, click New.
4. In the Edit System Variable (or New System Variable) window, set the name of the PATH environment variable as BOOST\_ROOT, and the value as C:\AnimatLabSDK\3rdParty\boost\_1\_54\_0. Click OK, and try to build the solution again.

## Instructions for controlling a robot with AnimatLab

### Configuring Szerial.ino

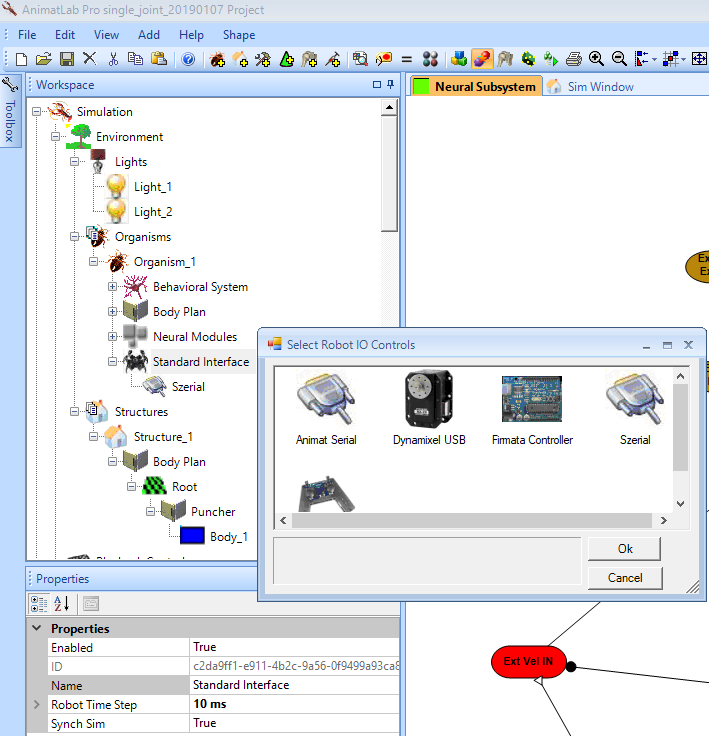
To use Szerial, the user will need to configure the Arduino sketch Szerial.ino. In it, the user will specify:

1. the number of Dynamixel smart servos, to write and read positions. The servos must have IDs in a continuous list starting at 0 and ending at numServos-1.
2. analog inputs from AnimatLab, which the OpenCM will use as commands for robot outputs, such as PWM signals for servos or LEDs. The analog pins must be a continuous list. The default list is in Szerial.h, but the user can change this to fit their particular application.
3. analog outputs to AnimatLab, which the OpenCM will collect from sensors, such as potentiometers, gyroscopes, or strain sensors. The analog pins must be a continuous list. The default list is in Szerial.h, but the user can change this list to fit their particular application.
4. the I/O loop duration. This must correspond to the value set in AnimatLab, in the Standard Interface>Szerial properties window.

Szerial should now interface with the modified AnimatLab SDK linked to above. If it does not work as intended, the user can use a USB to TTL debugging cable like [this one](https://www.adafruit.com/product/954) and uncomment the #define DEBUG on line 18 to read info from Serial3 (TX is pin 24, RX is pin 25). The complete OpenCM pin designations can be found in [the online e-manual](http://emanual.robotis.com/docs/en/parts/controller/opencm904/).

### Configuring Szerial within AnimatLab

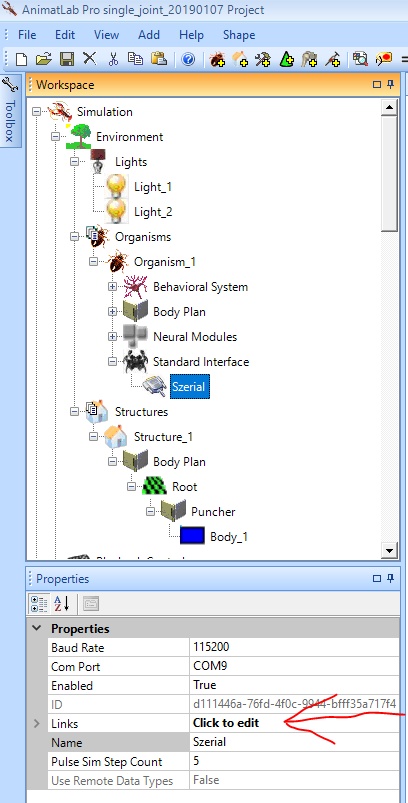
The user must configure Szerial within AnimatLab to ensure proper communication with the robot. First, the user must add a Standard Interface to their organism, and then right click to Add IO Control. The window will provide several options. If Szerial is not an option, as shown in the figure below, then the SDK has not been set up properly.



One can also see that the Robot Time Step is set to 10 ms. As noted above, this must match Szerial.ino. The speed of Szerial.ino will likely be the limiting factor, so I *highly* recommend that the user first calibrate their particular robot, by testing the controller and indicating (either with Serial.print, or an LED, etc.) when the loop is simply “waiting” after completing all writes and reads, and the end of the loop. If your program never waits, then it cannot read and write from all servos/peripherals as fast as needed. If this is the case, increase the Robot Time Step in AnimatLab, and the loopDuration in Szerial.ino. Of course, if your program spends a lot of time waiting, then shorten the loop duration until it waits just a tiny bit.

After instantiating the Szerial IO Controller, the user must set basic properties, shown in the image below. The Baud Rate should be as fast as possible while still providing error-free communication between AnimatLab and the OpenCM. 115200 is a standard value that is rather high. The user may of course attempt to use a higher Baud Rate.

The Com Port must match however Windows has installed the OpenCM. This can be checked in Device Manager>Ports (COM & LPT)>ROBOTIS Virtual COM Port (COM XX).



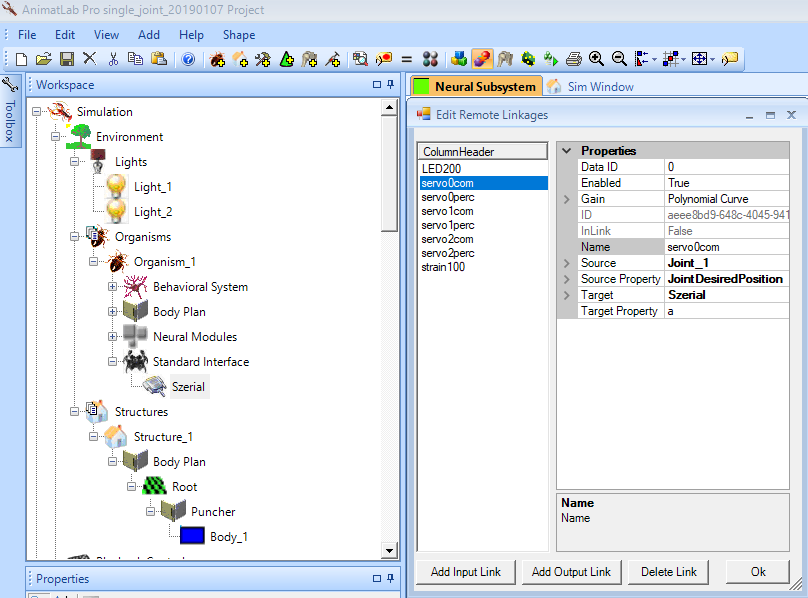
### Configuring a Dynamixel within Szerial

To set up particular I/O connections (e.g. with a particular Dynamixel), the user must edit the Links, by clicking on Click to Edit, and then clicking on the ellipses that appear. This will bring up a new window with a list of input and output links between AnimatLab and the OpenCM. The words “input” and “output” are from AnimatLab’s perspective; to command a servo’s position, specify an output link. To read its position, specify an input link.

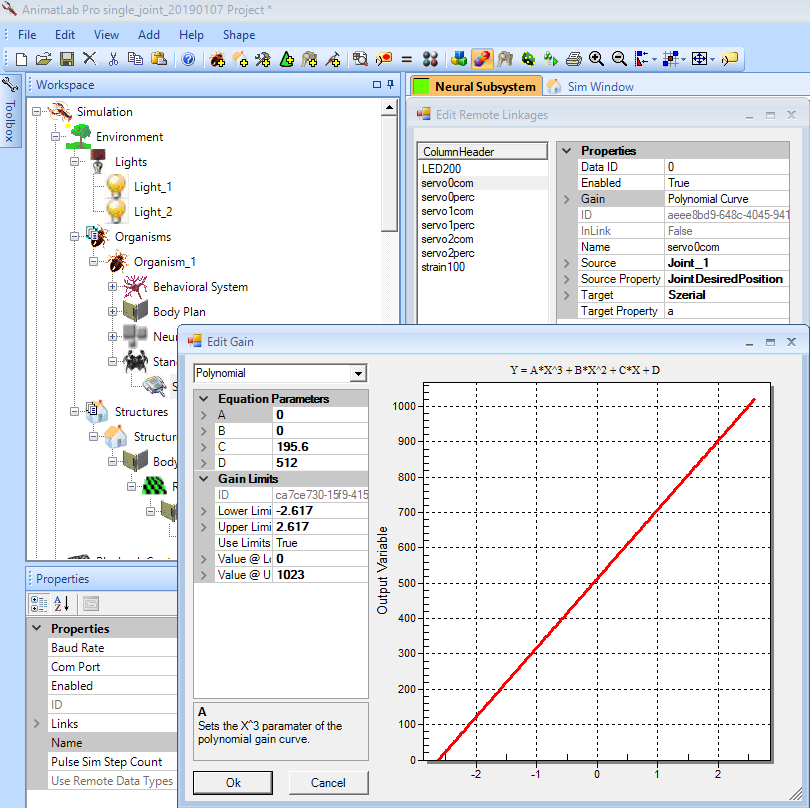
Szerial is set up to let you control a servo two ways: set its position, or set its position and velocity. First, setting up the position control is explained. This is the same for both methods. Second, setting up the velocity control is explained.

To control a servo’s position, first set the Data ID. For a servo or other actuator, this should be the actuator’s internal ID. The robot should be set up to list them from 0 to the number of actuators minus one. Next, the Source should be specified. For an output, this will be a part of the simulation. It could be a particular joint, a muscle, a neuron, or anything else. Once this is selected, the user must specify a Source Property. I prefer to link my servos to individual Joints and their Desired Positions, so I can make one neural controller and easily test it either in simulation or with the robot.

For an output link, the Target will be the name of your Szerial controller, in this case, simply Szerial. The Target Property **must** be unique for each item; I typically use single letters in alphabetical order. Using the same value twice will cause commands to be issued to the wrong robot component. For an input link, the Source and Target will be swapped; the source will be Szerial, and the user must specify a Target within their simulation. For servos, this can be the JointRotation, but it can be any neural or mechanical property.



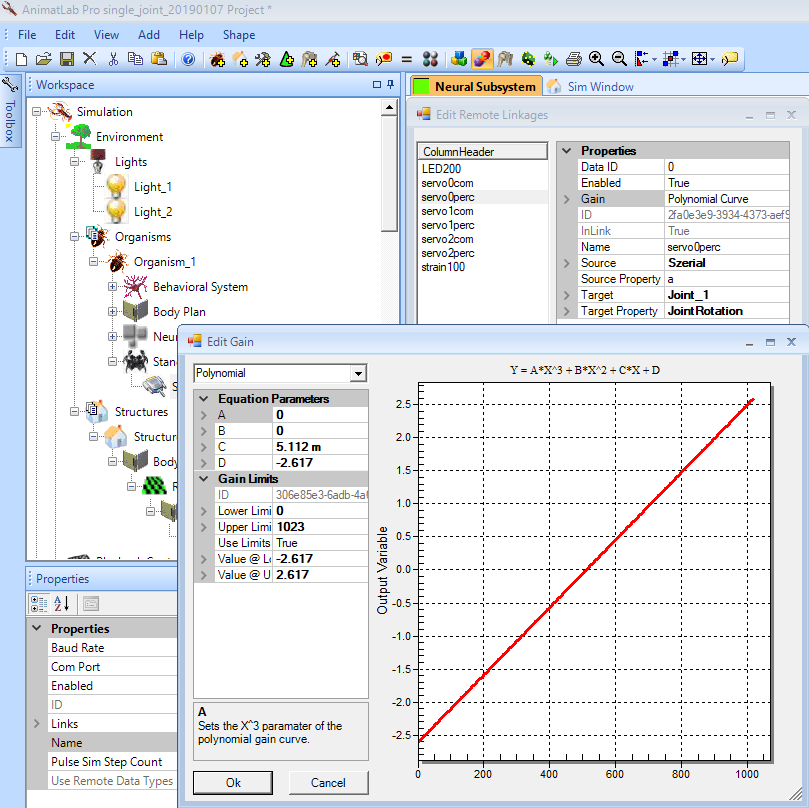
After specifying these basic quantities, the user must set the Gain between the Source and Target values. Clicking on the words Polynomial Curve, and then clicking the ellipses that appear, will bring up this window:



Here, the user must design a polynomial, sigmoidal, or Gaussian mapping from the Source to the Target. In this case, I am mapping the desired joint angle in radian into commands for a Dynamixel AX-12 smart servo. The AX-12 specifies its position as a 10 bit integer, where 0 is -150 degrees, and 1023 is 150 degrees. This enables the user to calculate the mapping from radian to AX-12 position. Note that all output values will be treated as integers. Also, note that I have elected to Use Limits, and have specified them such that values lower than 0 will be rounded to 0, and values greater than 1023 will be rounded to 1023. This prevents the actuator from receiving commands outside of its acceptable range. The user may also wish to add safeguards to Szerial.ino.

If the user wishes to also command a servo’s velocity, they can do so by adding another output link. This link must have the ID = servo’s ID + total number of servos. The variable “numServos” is set in SzerialTest.ino. So, for example, if you have two servos, their IDs must be 0 and 1, so to control their velocity, those output links will have IDs 2 and 3, respectively. The velocity is commanded much like the position; the user must specify a mapping between the simulation object (JointDesiredVelocity) and the physical object. The AX-12 servos specify speed on a 0-1023 scale, with 1023 meaning full speed (12 rad/s) and 1 meaning lowest speed. Specifying 0 also specifies full speed, so you may wish to set “use limits” to true, and make the minimum output velocity 1.

If the user wishes to read the position of the servo as well, then they must add an input link with the same ID. That is why the list in the above figure includes servo0com (servo 0 commanded position) as well as servo0perc (servo 0 perceived position). Such an input uses the inverse mapping of the output. For example:



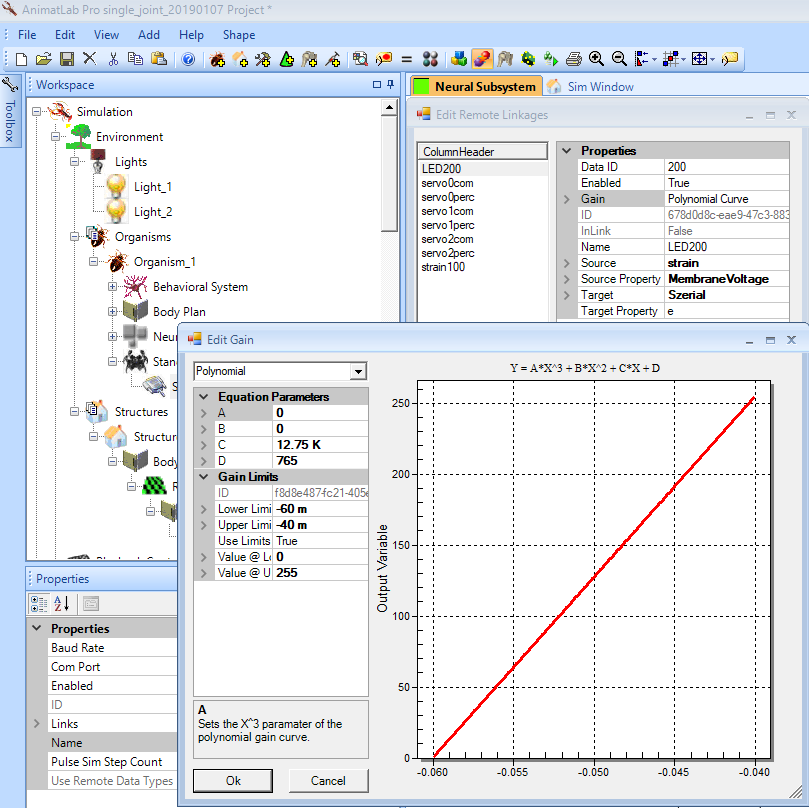
Reading servo velocity is analogous to reading the servo position: one must make a mapping from the digital values to the simulation values (JointActualVelocity). SzerialTest.ino is set up to return 1024 when the velocity is 0, 2047 when the velocity is 12 rad/s, and 0 when the velocity is -12 rad/s. This will need to be mapped by the user.

This arrangement may seem cumbersome at first glance. However, it is quite flexible, enabling the user to configure different types of actuators with different properties or polarities, without changing robot designs or low-level code. As examples, the MX series Dynamixels use 12 bit position encoding, and PWM outputs use 8 bit encoding. Rather than making a new control object, the user can simply configure the map to fit their application. The example project includes both a 12 bit MX series Dynamixel (servo 2), and an 8 bit PWM-controlled LED output (LED200).

Another advantage of this arrangement is that the direction or polarity of actuators can be easily modulated. For instance, biologists often specify joint rotation directions according to conventions that may make robot assembly cumbersome [8]. With Szerial, this is no longer a problem, because the user can easily change the sign of the slope of the input and output mappings to the actuator, and assemble the robot in whatever configuration is mechanically easiest.

### Configuring an analog input and output with Szerial

Configuring other inputs and outputs with Szerial is largely identical to actuators. The main difference is that they may not correlate 1:1 with objects in the simulation. For example, the brightness of an LED may be used to indicate a neuron’s membrane voltage or firing frequency. The image below shows the configuration of such an output link. The key difference is that read-only links should start with an ID of 100 and continue continuously (100, 101, 102, etc.), and write-only links should start with an ID of 200, and continue continuously (200, 201, 202, etc.). The example project includes both an input (a strain gage), and an output (a PWM-controlled LED).



## References

[1] D. W. Cofer, G. Cymbalyuk, J. Reid, Y. Zhu, W. J. Heitler, and D. H. Edwards, “AnimatLab: a 3D graphics environment for neuromechanical simulations.,” *J. Neurosci. Methods*, vol. 187, no. 2, pp. 280–8, Mar. 2010.

[2] N. S. Szczecinski *et al.*, “Introducing MantisBot: Hexapod robot controlled by a high-fidelity, real-time neural simulation,” in *IEEE International Conference on Intelligent Robots and Systems*, 2015, pp. 3875–3881.

[3] N. S. Szczecinski, A. P. Getsy, J. P. Martin, R. E. Ritzmann, and R. D. Quinn, “MantisBot is a Robotic Model of Visually Guided Motion in the Praying Mantis,” *Arthropod Struct. Dev.*, 2017.

[4] N. S. Szczecinski and R. D. Quinn, “Leg-local neural mechanisms for searching and learning enhance robotic locomotion,” *Biol. Cybern.*, pp. 1–14, 2017.

[5] N. S. Szczecinski and R. D. Quinn, “Template for the Neural Control of Directed Walking Generalized to All Legs of MantisBot,” *Bioinspiration and Biomimetics*, 2017.

[6] A. Hunt, N. Szczecinski, and R. Quinn, “Development and Training of a Neural Controller for Hind Leg Walking in a Dog Robot,” *Front. Neurorobot.*, vol. 11, 2017.

[7] A. J. Hunt, “NEUROLOGICALLY BASED CONTROL FOR QUADRUPED WALKING,” Case Western Reserve University, 2016.

[8] D. M. Chrzanowski, “MantisBot: A Robotic Platform for Development of Complex Neural Control,” Case Western Reserve University, 2015.