VERSION 0.5.2

Documentation rev 1

-1-

End User License Agreement

ALL USE OF BOTTANGO IS AT YOUR SOLE RISK.

BOTTANGO IS IN BETA TESTING AND MAY CONTAIN ERRORS, DESIGN FLAWS, BUGS, OR DEFECTS. BOTTANGO SHOULD NOT BE USED, ALONE OR IN PART, IN CONNECTION WITH ANY HAZARDOUS ENVIRONMENTS, SYSTEMS, OR APPLICATIONS; ANY SAFETY RESPONSE SYSTEMS; ANY SAFETY-CRITICAL HARDWARE OR APPLICATIONS; OR ANY APPLICATIONS WHERE THE FAILURE OR MALFUNCTION OF THE BETA SOFTWARE MAY REASONABLY AND FORESEEABLY LEAD TO PERSONAL INJURY, PHYSICAL DAMAGE, OR PROPERTY DAMAGE.

YOU MUST REVIEW AND AGREE TO THE BOTTANGO BETA SOFTWARE END USER LICENSE AGREEMENT BEFORE USING BOTTANGO: http://www.Bottango.com/EULA

2-
What is Bottango?

A little background

I've been a professional game developer for over a decade, and one of the most important lessons I've learned is that nothing supercharges development and product creation more than creating tools and processes that empower people to build what is in their mind's eye. When an individual has the tools and power to visualize, quickly iterate, and autonomously build their vision, the investment in the tools to do so often has a far greater return than hard- coded rules and behaviors.

When I first started building robots in my spare time, the magic of making something move in real life was intoxicating. A single line of code to change the PWM value on a servo had immediate real world impact. However, quickly my game developer instincts kicked in. I didn't just want the servo to move. I wanted to be able to control how the servo moved: How long did it take? How fast did it move? What curve did the servo take to get from start to finish? Was it slow at the start? Did it bounce a little past the destination and move back?

The traditional answer to these desires in robotics is to write the code that creates the behavior you want, then examine the results. If you're not happy, write new code. However, in game development, I knew that I'd take a different approach. I'd advocate for a tool that allowed for easy, visual iteration on robotic movement, with all the tools and controls needed to express exactly what I envision for my robot.

What is Bottango? - The short version

Bottango is a visual tool for intuitive robot control, with high levels of control and precision . It is meant to replace line upon line of special case behavior and hard coded movements with a WYSIWYG editor, so that an individual's intent can be more quickly and accurately expressed.

Bottango operates in real time. If you move the eyebrow of an animatronic face in Bottango, the eyebrow of your robot moves too. If you scrub an animation of your robot arm in Bottango, your robot arm plays the animation at the same time.

Bottango is robot agnostic. It's not supposed to work with only this robot from that manufacturer. It is intended to allow you to define the parameters of your robot, and iterate on the machine itself, not just its behavior.

What can Bottango do now?

In this version of Bottango, you can:

- Define the basic structure of your robot.

- Create and configure motors (Servos and Stepper Motors currently supported).
- Control those motors with hardware drivers (for example, an Arduino).
- Create virtual joints to move with motors.
- Animate those joints on a visual timeline with keyframes.
- Control the interpolation curve of your keyframes
- Synchronize an animation to audio.
- Real time input (game controllers, keyboard, mouse, etc.) for controlling your robot, including recording and playback.
- Sending arbitrary, user-defined events from Bottango to hardware drivers on the animation timeline.
- Import 3D models to use as the structure of your robot.
- Play back an animation on your robot.
- Control the movement of your robot in real time.
- Baking animations to code that can be run on hardware without the Bottango app driving the animations.
- Creating animations via IK chains.
- Create key poses on your robot and blend between them.
- Cross Platform Support

What will Bottango someday do

This is an extremely early version of Bottango. There are a lot of things I have on the development roadmap; here are some of the highlights:

- Additional motor types such as brushless DC motors, etc.
- Animating lights, including RGB LEDs.
- Blending between animations so that a robot can smoothly move from one animation to another.

Ultimately, I see Bottango behaving like a "state machine." In this mode, I envision the user defining state machine logic for a robot, and then controlling the expressed state of that state machine via commands from both the user and the hardware driver. In a state machine, the robot blends between animations as the state changes.

As an example, imagine a robot that cycles through blends of a few different idle animations, and then smoothly blends to a surprise animation when a capacitive sensor on the hardware driver detects touch, and then blends back to its idle animation as defined in the state machine.

Who is Bottango for?

Bottango is for anyone who wants to move or control real life things. If you want to inspire an emotional response when someone sees your robot move, Bottango is the tool to allow for that kind of expression by providing tools for rapid creative iteration on robotic movement. If you want to quickly and intuitively create trajectories and motion plans for your robot, without writing line upon line of custom code, Bottango is the tool to allow you to intuitively craft those movements with a visual editor.

You don't need to know how to code to use Bottango, but you should be pretty computer savvy. There will be problem solving and general computer tom-foolery. Bottango runs on Windows, Linux, and Macintosh computers.

The more you're familiar with the concepts of 3D animation and robotics, the more Bottango will come naturally to you. You don't need these concepts to be successful, but they will give you a leg up to understanding how Bottango thinks.

Keeping this documentation up to date

Bottango is in early stages. A lot is changing, and fast. New features are being added, and I'm frequently iterating on existing features. Throughout this documentation, I'm trying to keep things up to date, but I can't make any promises. Screenshots and images may show old versions of features not directly relevant to the subject at hand. If I had infinite time I'd replace all screenshots every time I update Bottango, but something has to give! However, I will try and make sure that at least the subject in question has accurate visualizations.

-3-

How To Use This Documentation

This documentation is made up into three parts.

The first part is the next chapter (Chapter 4: A Crash Course in Bottango). For those who like to learn through experimentation, that chapter gives the conceptual foundation you need to go off and explore Bottango on your own. But buyer beware, it's a lot of concepts thrown out at rapid speed! That chapter does not teach HOW to do things, but instead the major concepts at work in Bottango.

After the first chapter, the rest of the documentation covers the same subjects again, but at a much more detailed pace. Subsequent chapters as well will cover the information through tutorials and "how-to."

There is *nothing* covered in the "Bottango Crash Course" chapter that won't be covered again and in more detail later on in this documentation. So it's up to you if you'd like to skip the crash course entirely, or even to only read the crash course and figure the rest out through experimentation

The second part of this documentation covers the basics of getting up and running with Bottango. The third part covers additional advanced features and more in Bottango.

Part 1 - Crash Course

-4-

A Crash Course in Bottango

What's in this chapter

In this chapter we cover the core concepts and workflow of Bottango. This chapter is not a "how-to" but instead a high level conceptual overview of how robot animations are created in Bottango.

In future chapters we will cover this same information again, but in a much more detailed level with tutorials.

How Bottango works

Bottango works by sending commands over serial to a hardware driver. Think about it kind of like the relationship between a client and a server. Bottango is the server, sending commands. Your hardware driver is the client, receiving those commands and acting on them.

A hardware driver can be anything you want that can create a serial connection to your computer. However, in the most common case, a hardware driver is an Arduino or a similar user programmable microcontroller.

In Bottango, A hardware driver is a user programmable microcontroller running Bottango compatible code.

Hardware drivers respond to serial commands from Bottango running on a computer.

Bottango provides not just the program that runs on your computer, but also a fully functional Arduino-compatible program to upload to your microcontroller. That program is able to open a connection to Bottango running on your computer, and execute the commands Bottango sends to it. In order for Bottango to do anything, you need to upload the provided Arduino compatible code to a microcontroller and connect it via USB to your computer.

You can modify the provided Arduino program, but you don't have to touch a line of code to get all of the out-of-the-box Bottango functionality.

In the Bottango application itself, you create the animation that will be sent to your robot via serial commands to hardware drivers.

The basic Bottango workflow

Bottango is a tool to help you control a robot, not to help you build that robot. So let's assume you have a robot that could move if given the right controls. The steps you will take in Bottango are:

1. **Model the basic structure and joints of your robot.** The accuracy of the model required is up you. Your goal in this step is not to create a picture-perfect representation of your robot. Instead, create just enough detail to represent your robot so you can manipulate its joints. If you're familiar with the concept of animation rigging in 3D modeling, this is approximately the same idea.

   You can also import a 3D model instead to use as the structure of your robot.

2. **Add and configure motors.** Each motor in your robot that you want to control needs an accompanying motor in Bottango to configure and control.

3. **Add and configure hardware drivers.** Bottango can't move effectors on its own; it needs a hardware driver to send commands to, to drive motors, etc. Upload the provided code to a microcontroller, and then set up Bottango to communicate with that microcontroller.

4. **Configure joints.** Joints are links between the structure of your robot and motors. With joints you'll indicate that by moving your robot in a particular way, your motors should drive to certain positions.

5. **Add audio.** If you want to synchronize movement or playback audio, add it to Bottango.

6. **Animate your robot**. Create keyframes on the animation timeline of joint movement. Edit the time, value, and interpolation of those keyframes.

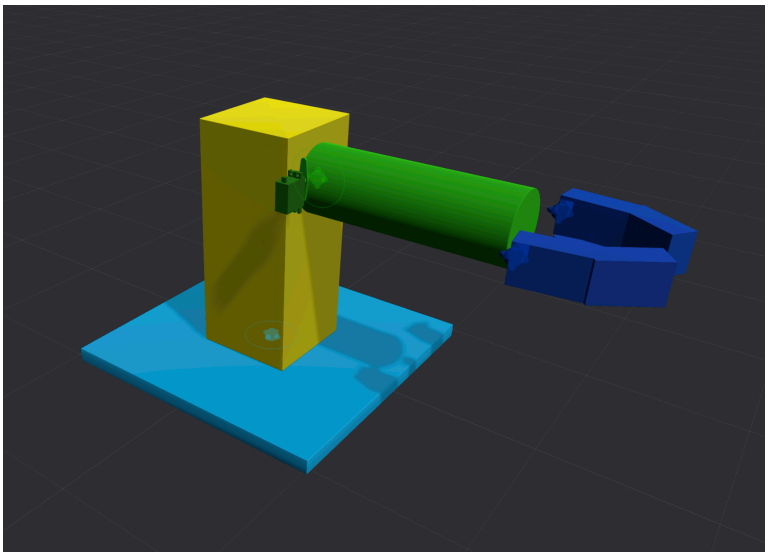
1. Model the basic structure of your robot

In order to move and animate your robot, first you need to create a virtual representation of it. Bottango provides very basic modeling tools to create the structure that represents your robot.

The structures you model in Bottango should not be a picture-perfect representation of your robot. Instead, they are just the virtual controls that you will manipulate to move your robot and visualize its movement.

> In Bottango, structures are the virtual representation of your robot's physical form.

Here is an example of a simple robot structure built in Bottango:



This robot has three points of articulation:

A. It rotates the yellow base extension on the Y axis.
B. It rotates the green arm up and down on the X axis
C. It opens and closes the blue gripper.

Obviously, the actual physical robot is far more complicated than this. However, this level of detail is all that's needed in Bottango to visualize and control movements.

• *Parenting*

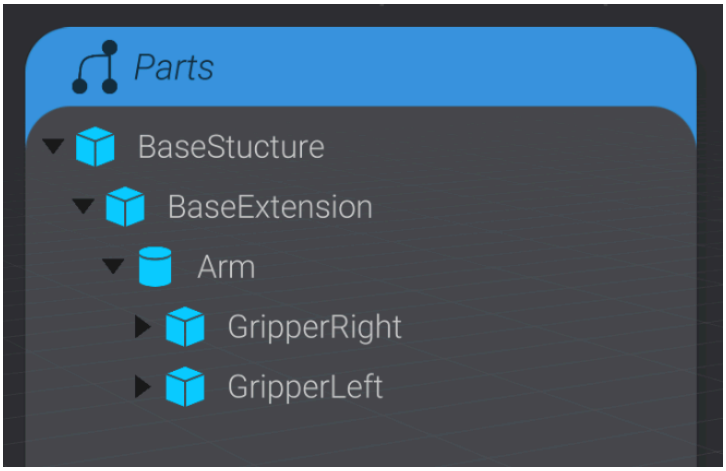
As you model the structure of your robot, it's important to think about parenting relationships.

When a part of your robot moves or rotates, any structure, motor, or joint that is the child of that part will move and rotate with it.

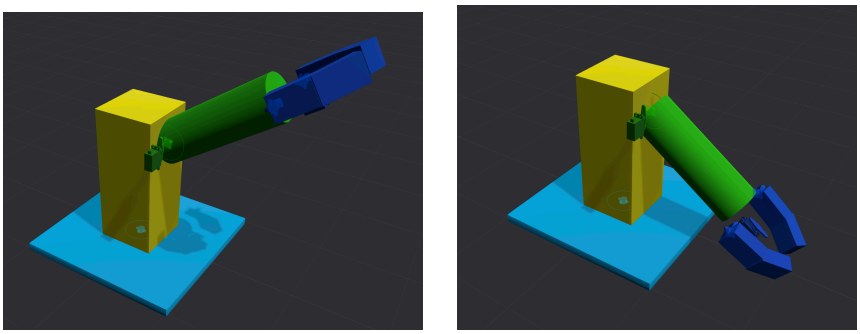
Looking again at that simple robot, we see the following required parent-child relationship.

1. The teal base is the top-level structure.
    2. The yellow extension is a child of the base.
        3. The green arm is a child of the extension.
            4. The blue gripper is a child of the arm.

Here is that same hierarchy as set up in Bottango:



For example, the green arm rotates, and the blue gripper rotates with it.



• *Structures are built at "home"*

As you model the basic structure of your robot, it should be modeled in the "home" position and rotation. This is the starting point from which structures will move while they are animated, and return to at rest.

> In Bottango, home is the position and rotation a structure will originate from when animated, and return to at rest.

This will be explored more when we talk about joints. But for now just remember that where you model a structure is its home, and when it moves, it will move away from home.

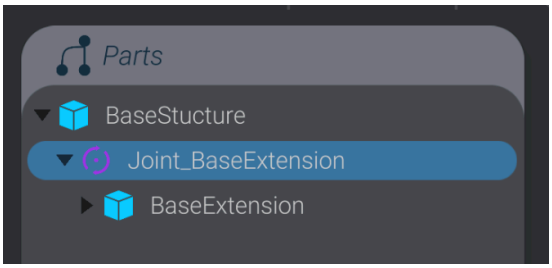
• *Add joints*

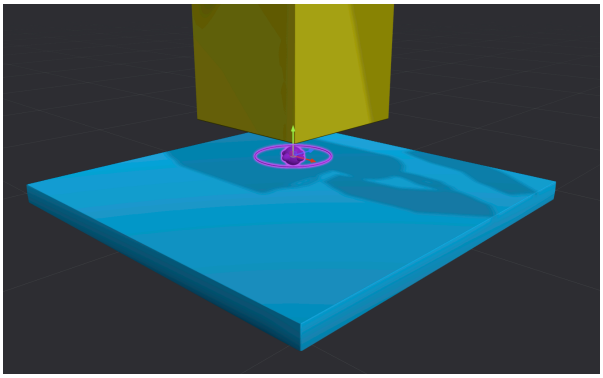
The structures you create are inherently static, just like how a support beam in your robot doesn't move on its own. While you are creating structure, you should also be creating joints in Bottango. Joints are the points of articulation that can move or rotate in a single axis.

> In Bottango, joints are the points of articulation that can move or rotate on a single axis.

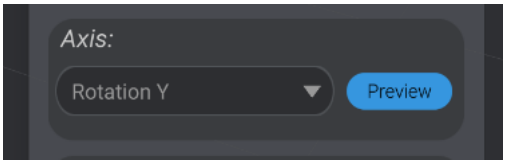
It's important to make sure the structures you build that would be moved or rotated by a joint are children of that joint. As a joint moves or rotates, just like any other parent-child relationship, the structures and other joints that are a child of the moving/rotating joint will move/rotate as well.

In this example, a joint has been created as a child of the BaseStructure, and as the parent of BaseExtension. This means that when that joint rotates, BaseExtension, and everything that BaseExtension is a parent of, will rotate.

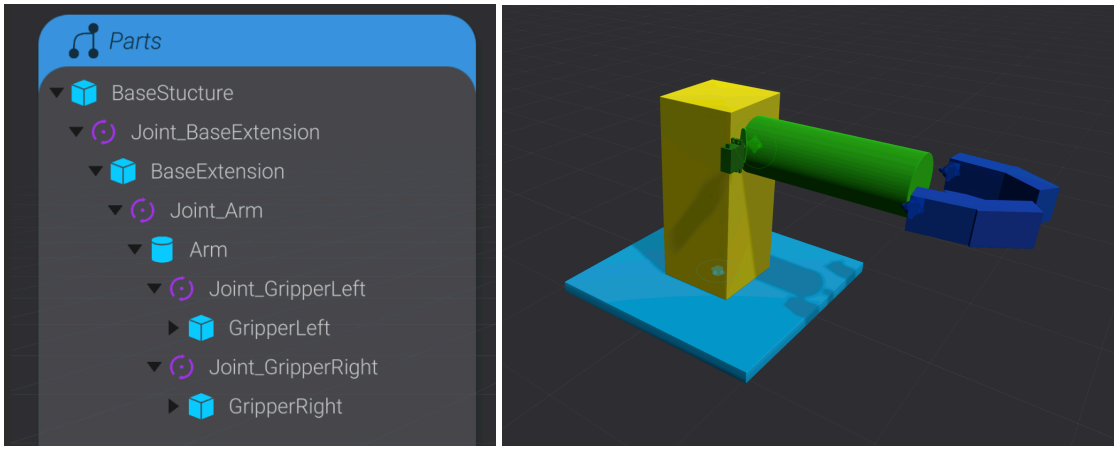




In addition, you can set the axis each joint you create. Here you can see this joint has been set to rotate in the Y dimension.



Finally, for our example robot, here you can see the full hierarchy of structure and joints:

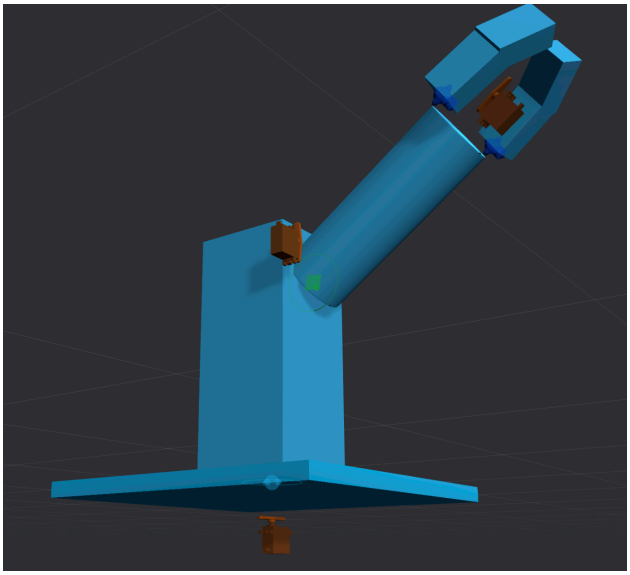


Each point of articulation in the robot has a joint that is the parent of the next part of the robot.

2. Add and configure motors

In order to create actual movement, you need to set up motors in Bottango. Right now Bottango supports servo motors and stepper motors. Because servo motors are the most simple, we'll be using servos for this portion of the documentation.

For each motor you want to control, you need to create a virtual motor:



In order to configure a motor, you need to provide a few things:

- The minimum and maximum signal you want the motor to receive. (ex: a servo motor that is configured get a PWM signal ranging between 1000 PWM and 2000 PWM.)
- The signal you want to send the motor to move the motor "Home" (ex: 1500 PWM is "Home" for this hobby servo)

- How you want to connect to the motor. (ex: connect to this hobby servo on pin 6 on the default hardware driver)

• *Movement: The most important concept!*

Throughout Bottango you will see this icon:

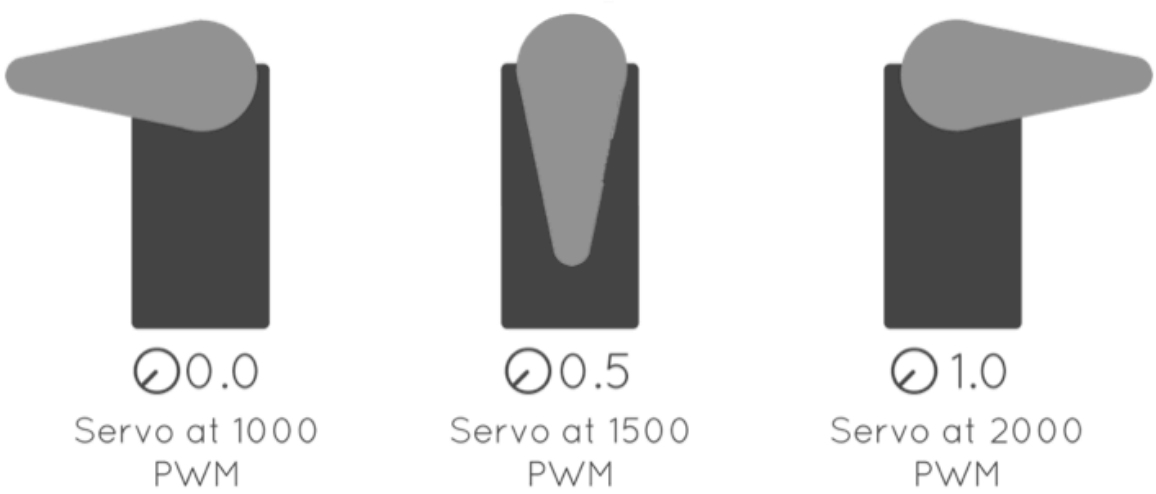
This icon is so important, you can see it's even in the logo for bottango:

This icon represents what I call "movement," and it stands in for the the value of an object between its minimum value and maximum value, on a 0 to 1 scale. A more "math-y" term for movement would be a "normalized value."

In Bottango, movement is the value of an object between its minimum value and maximum value, on a 0 to 1 scale.

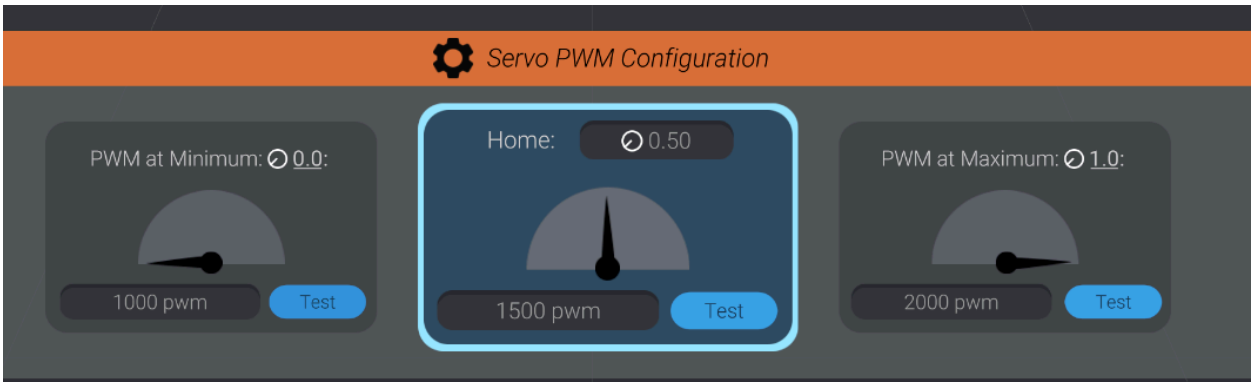
Movement is used in motors, joints, and animations.

Let's take, for example, a servo motor. A servo motor receives a PWM (pulse width modulation) signal from a hardware driver, and the servo then moves to an angle that maps to that PWM value. Let's say you wanted your servo to receive, at the absolute minimum, a value of 1000 PWM, and an absolute maximum value of 2000 PWM.



In Bottango, you would say that the minimum PWM signal (movement 0.0) is 1000, and the desired maximum PWM signal (movement 1.0) is 2000.

You can see movement referenced here in the setup menu for a motor:



The user is inputing that at a movement of 0.0, send a PWM signal of 1000. The user is also inputing at a movement of 1.0, send a PWM signal of 2000.

The user can input the home signal either in PWM or movement, and the corresponding value is calculated. Here the user has entered a home movement of 0.5, which calculates to a PWM value of 1500.

Why is movement important? Movement is the common denominator that allows you to manipulate not just different kinds of motors, but also different concepts. Additionally, movement allows you to change your robot at will and not worry about keeping the rest of your project in sync.

3. Add and configure hardware drivers

As mentioned before, Bottango cannot move motors on its own; it needs at least one microcontroller to communicate with and send commands to.

Before you can actually move and animate your robot, you need to upload the provided Arduino-compatible code onto your microcontroller. Provided with Bottango is a library of Arduino-compatible code that works out of the box with Bottango. It is located in the same folder that the Bottango application comes in. Additionally, there is a unique read-me with installation instructions for the Arduino compatible code.

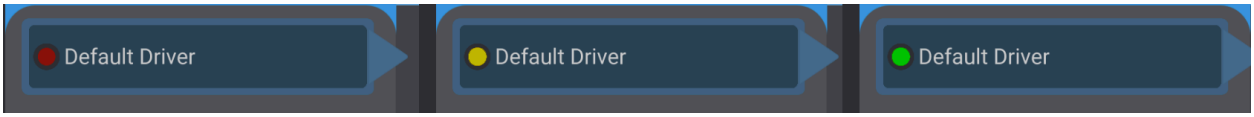
Once the microcontroller is running the supplied program, connect it to your computer via USB.

Bottango doesn't actually care what it's talking to though; it doesn't need to be an Arduino compatible hardware controller. You can and are welcome to rewrite the Arduino-compatible code to whatever microcontroller you prefer. All that matters is that Bottango talks to some kind of hardware driver over a serial connection that listens to and responds to its serial commands as Bottango expects. What happens after that is up to you.

Inside Bottango, you can set up hardware drivers. Each hardware driver has different ways you can connect to it, either through searching through available ports for a matching name, or through explicitly indicating a port with which to connect.

• *Driver status indication*

Whenever drivers are shown in Bottango, they are often shown with a colored circle next to them. That indicator is either, red, yellow, or green.



- A red indicator generally means that Bottango was unable to open a port for this hardware driver. Usually this is because it is not plugged in, or the port is busy.
- A yellow indicator generally means that Bottango opened a port for this driver, but wasn't able to start communicating with the hardware driver. This could be for a variety of reasons, including not actually running Bottango-compatible code on the device at that port.
- A green indicator means that everything is good to go, and communication between Bottango and this hardware driver is working.

You can see more detailed status messages in the drivers menu.

• *"Live" and "not live"*

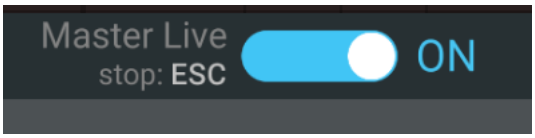
In order to let you control exactly what signals are sent, all drivers, motors, and Bottango itself have a "live / not live" switch.

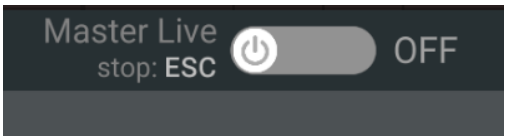
In Bottango, Live is whether the given motor, driver, or Bottango itself will attempt to send commands.

When a driver is set to "not live," Bottango will not send that driver any commands.

When a motor is set to "not live," Bottango will not send the driver associated with that motor commands for that motor.

You can stop all commands at the top level by setting Bottango itself to "not live."





Master live is shown in the top right of Bottango at all times. When master live is set off, Bottango will try and send a "STOP" signal to all currently live drivers, which pauses all movement. Until master live is set back on, no commands will be attempted to be sent.

• *Stopping Bottango with the escape key*

If at any point you want to try and stop your robot, press the escape key. This will set master live to off wherever you are in the application, and try to send the STOP signal to all live hardware drivers. Pressing escape again does not turn master live back on. Instead you must click the switch to re-enable.

IMPORTANT NOTE: Master live is *NOT* intended to be used as a safety mechanism. You should have hardware stops on your robot. Master live is intended as a convenience tool.

4. Configure your joints

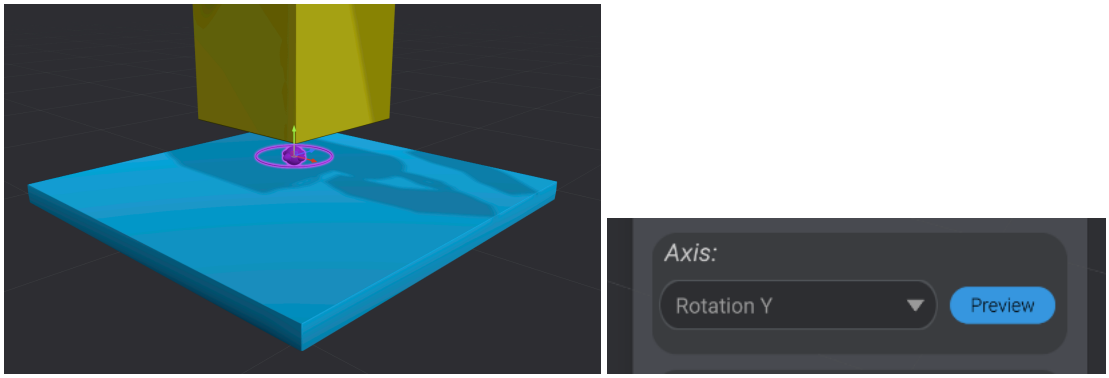
As a reminder, joints are the points of articulation that can move or rotate in a single axis. While setting up the structure of your robot, you placed joints at each point of articulation. The behavior of each joint needs to be configured as well.

In order to make a functioning joint, you need to configure the following:

- **Configure the axis that the joint moves or rotates on.** This defines if the joint moves or rotates in X, Y, or Z dimension.
- **Associate the joint with a motor.** When the joint moves or rotates, commands will be sent to that motor.

- **Set the joint minimum offset.** When the joint is at its minimum (0.0) movement, how much should it move or rotate away from home?
- **Set the joint maximum offset.** When the joint is at its maximum (1.0) movement, how much should it move or rotate away from home in the other direction?
- **Resolve any conflict if the motor and joint have different home movement.** If your joint has a home at 0.25, and your motor has a home at 0.75, which do you use when sending your robot home? You'll have an opportunity to either resolve the conflict to make the joint and motor have a matching home, or choose which home value you want to use.

Here's a specific example. This basic robot has a joint between BaseStructure and BaseExtension that has been configured to rotate in the Y axis:



The user has associated the joint with a motor, and is now configuring its minimum and maximum range of rotation.

You can set up a joint's range of movement or rotation without linking first to a motor, but it's easier to keep things in sync and set the joint up accurately if you link it to a motor first.

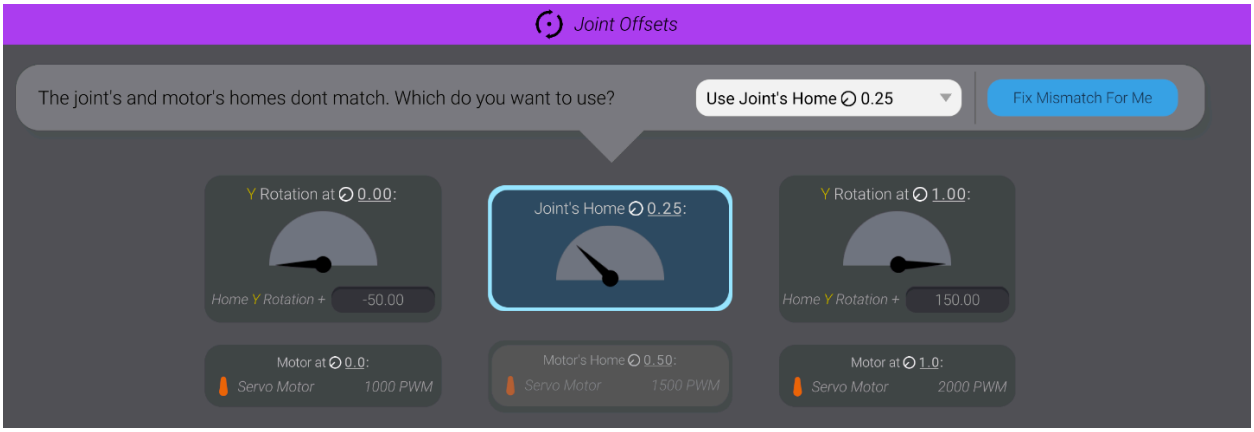
• *The best way to find your offsets*

There's a reason the basic workflow has you create and configure hardware drivers before configuring joints. If your hardware driver is live, while you are creating and editing a joint, Bottango will be sending commands to the associated motor for a selected joint at the same time. For example, while editing the minimum movement (0.0) offsets of a joint, Bottango will move that joint's motor to its current minimum movement hardware signal value.

This allows you to look at your robot in real life, and move its virtual representation to as close an approximation as you can. This is very helpful for making sure the movement of your physical and virtual robot are in close approximation.

In addition, this will hopefully start to illuminate one of the relationships of movement: the relationship between joint movement and motor movement. Moving a joint to its offset for a movement of 0.25 will send a command to the motor in that joint to move to its signal value for a movement of 0.25.

If your motor and joint don't have the same home movement, how does Bottango know which to use when sending your robot to home? The joint's or the Motor's? When a joint is linked to a motor and the home values don't match, you'll see a warning.



In that warning you can select if you want to use the joint's home or the motor's home. You can also click the "Fix Mismatch For Me" button, and Bottango will make its best guess as to what changes to make to get your motor and joint to match homes.

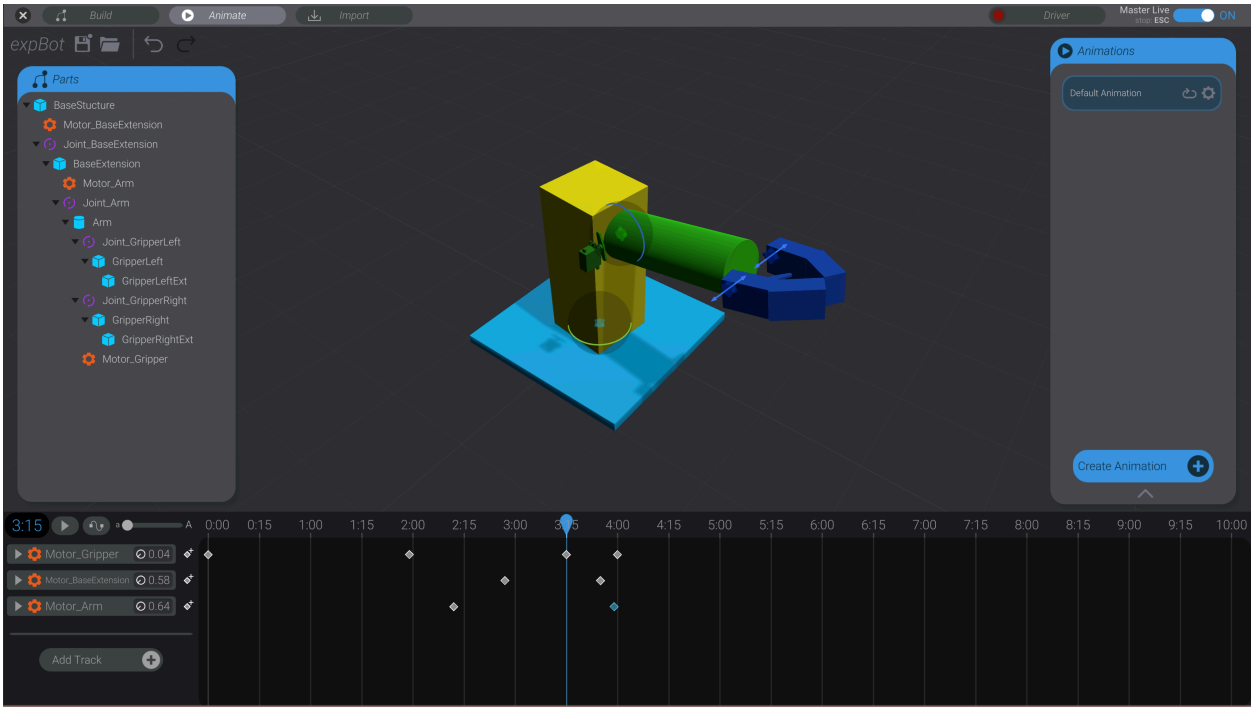
5. Add audio

If you want to animate your robot along with audio, you need to import audio tracks. Audio tracks are associated with a file path. When you reload a Bottango project, it will attempt to reload the audio tracks at that file path. Bottango will let you know if it cannot find the audio required, and will allow you to change the path if needed.

Bottango only supports audio tracks in .WAV or .OGG format. It is easy to find any number of free .OGG converters online if you need one.

6. Animate your robot

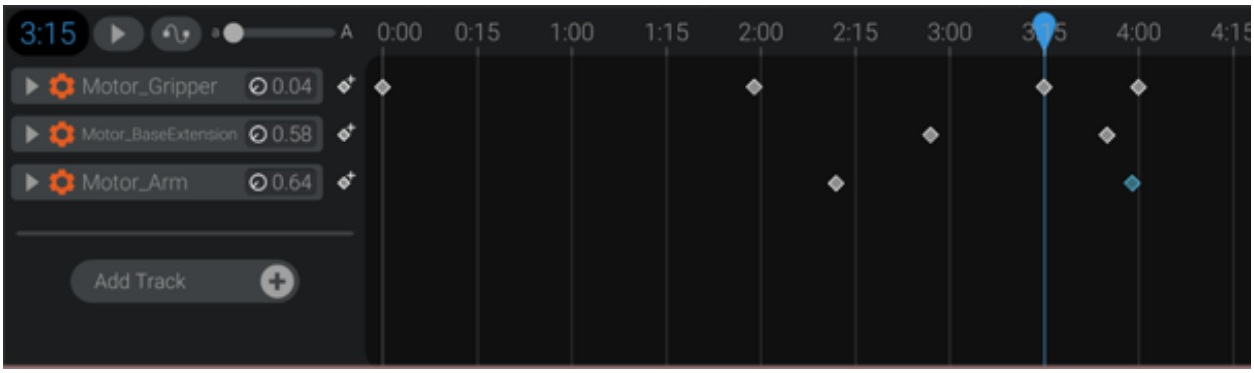
To animate a robot, you animate joints, moving them between their minimum and maximum movement values. You create the animation by creating keyframes along a timeline and setting values at those keyframes.



The scrubber in the above image is at 3 seconds and 15 frames. Bottango animation clips are 30 fps, so the scrubber is at 3 and 1/2 seconds.

(**Don't panic!** Animations on your hardware driver are at a MUCH faster frame rate than 30 fps. Animation frame rate on your microcontroller is determined by the speed of your microcontroller and how much else your microcontroller is doing. Bottango sends a curve to your microcontroller to execute as fast as the hardware can, not based on individual calculated values. 30 fps is just for the visual convenience of where to put keyframes on your timeline. Your motors will update their real world position on the order of 1000s of frames per second.)

To animate your robot, you simply move a joint to its desired position or rotation, and Bottango creates a keyframe wherever scrubber is located. Then, as Bottango plays back an animation, it will move the joint from one keyframe movement value to the next over the duration of the animation.



You can see in the above example that the user has set the scrubber to time stamp 3:15, and moved the base structure where he or she wants it. Bottango then created a keyframe and worked out the appropriate PWM signal to send to the correct servo motor.

You don't need to understand the exact math, or how the signals were calculated. That's the whole point of Bottango: you don't have to worry about that stuff. You just move the robot where you want it, and Bottango does the rest of the math based on your setup and sends the signal.

Instead, just think about the relationships that defined the above interaction:

1. The user moves or rotates a joint.
2. The joint calculates its movement based on its minimum and maximum offsets.
3. The motor associated with that joint calculates its PWM signal based on the joint's movement and its minimum and maximum signal.
4. Bottango sends the command to the serial driver that is associated with that motor.

Animation in real life

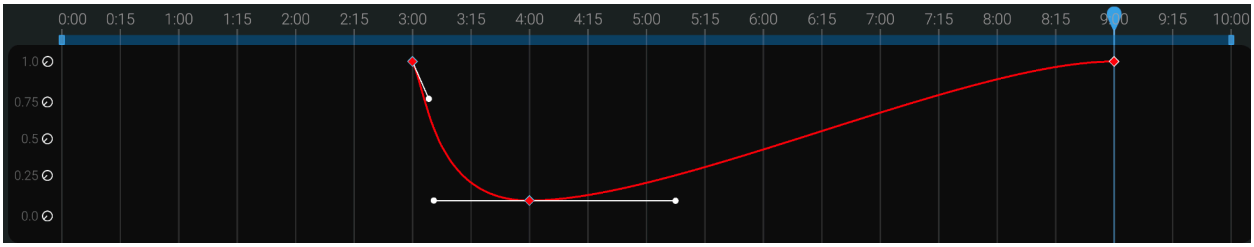
While you animate your robot—provided the motor, driver, and Bottango are all live—everything you do in Bottango will happen in real life at the same moment. This is one of the most powerful parts of Bottango!

This means you can scrub an animation and see it happen in real time on your robot. You can move your scrubber to an exact keyframe, and move and manipulate the joint until you're happy with the real world result. No need to compile, run, and see results. You are animating your robot in real time!

Keyframe interpolation

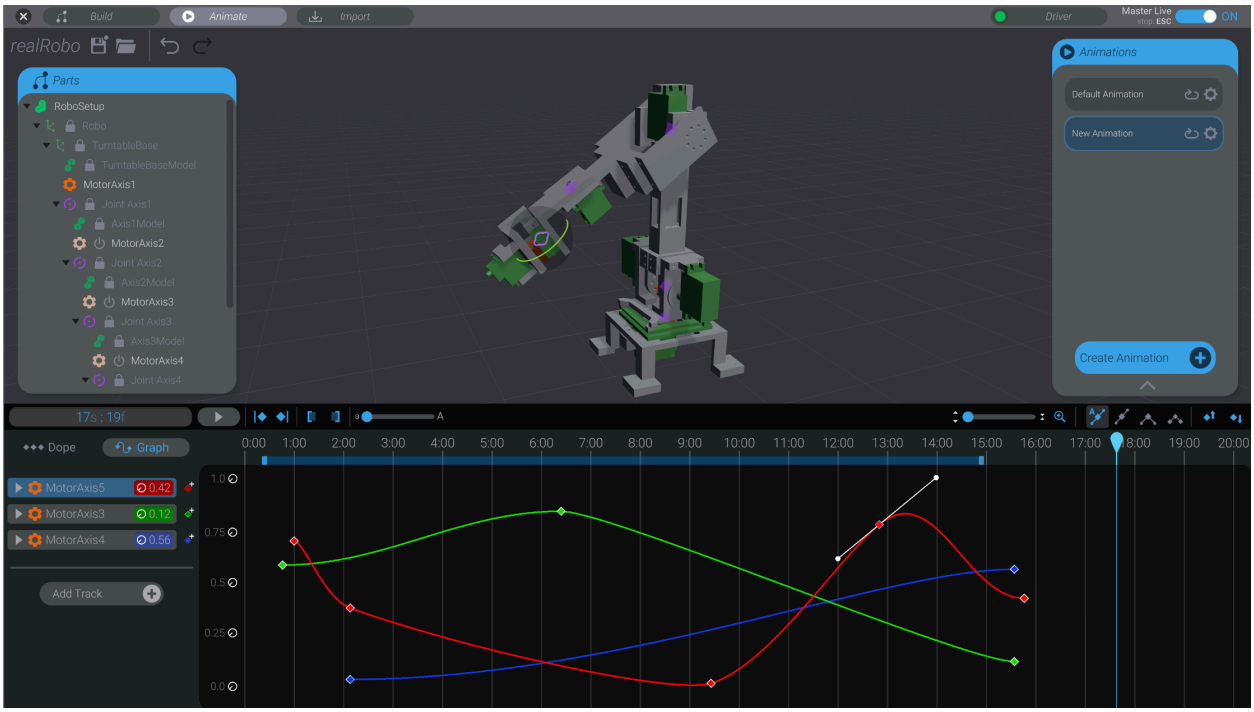
You don't have to settle for your robot moving from one keyframe to the next at a uniform speed. Bottango gives you the tools to sculpt the interpolation of your keyframes however you want.

As an example, here is a keyframe interpolation that starts at 1.0 movement around 3:00, moves quickly down to around 0.10 movement around 4:15, and then slowly works its way back up to 1.0 movement by 9:00.



Imagine how much more visually interesting a movement that is than simply moving from a start to end motor position at a uniform speed.

As you playback an animation, Bottango will send commands to the hardware driver that result in movement of the motor exactly as drawn in the curve. The Arduino-compatible code that you uploaded knows how to interpolate between the keyframes in the exact same curve and duration that you set in Bottango, so it will play back that animation between the keyframes at as high a frame rate as it can until the animation is complete or it receives a new command.



Take a breath

Congrats for making it this far. Don't worry if you don't feel like you've fully grasped the above concepts. That's OK! The goal of this chapter was not to turn you into an expert. This chapter exists to throw all the

big concepts at you and for each person to see what sticks! Now as you encounter these concepts in Bottango itself or in future documentation, these concepts will be a little more familiar.

Part 2 - Bottango Basics

-5-
Interacting With Bottango

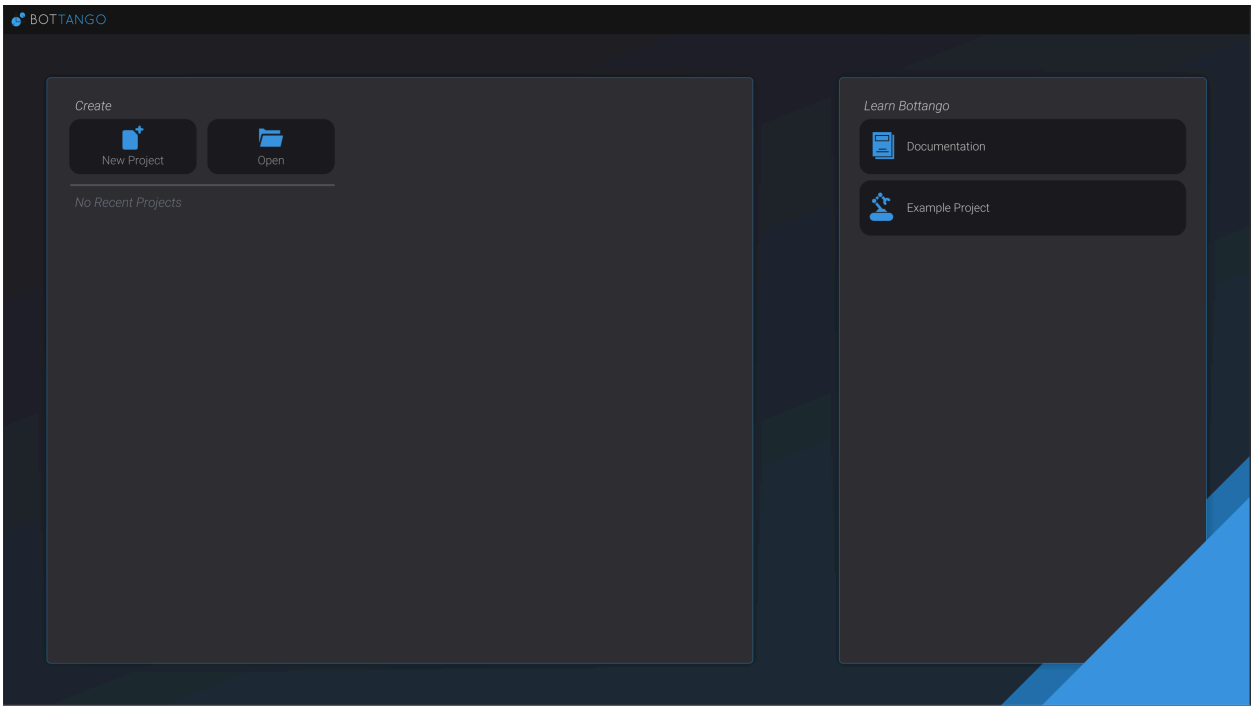
What's in this chapter

From this chapter on, we will be covering the same concepts presented in chapter 4 ("A crash course in Bottango") but in much more detail. This and all subsequent chapters are presented as tutorials and "how-to's" to help you learn the in's and outs of animating robots in Bottango.

This specific chapter covers the basics of interacting with Bottango:

- The home screen.
- Creating, opening, and saving projects.
- Creating and selecting parts.
- Controlling the camera.

Home



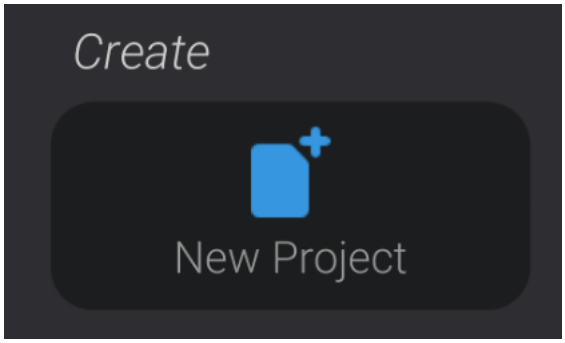
When you first launch Bottango, you will be presented with the home screen. From here you can:

- Create a new project.
- Open an existing project.
- Open a recent project.
- Visit learning resources.

- Open the example project.

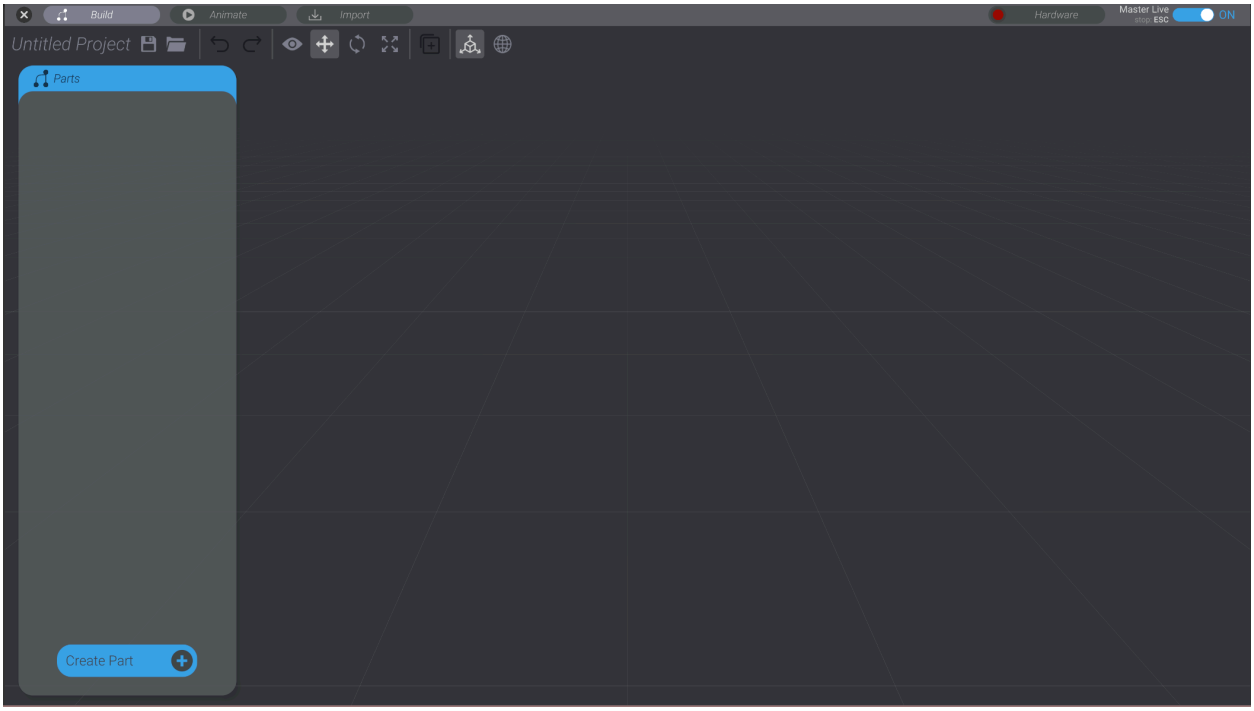
The example project is a project created by Bottango that shows off its basic functionality. It's also the example robot built in a lot of the upcoming chapters.

1 Click "New Project"



The Bottango Window

Here is the view of an empty, brand new project:



Bottango mode tabs

Along the top of the window, you will see a dark area with buttons like "Build" and "Animate." These are the mode tabs, and how you switch between the various modes of Bottango.

Bottango has currently the following modes:

- **Build.** This is the starting mode, and where you build and define your robot.
- **Animate.** Create the animation that will play back on your robot once it's built in configure mode.
- **Import.** Import audio tracks to use in animations and 3d models to use as structure.
- **Hardware.** Configure and establish connections between Bottango and hardware drivers, and view and control the status of motors.

In addition, you will see the current "Master Live" status of Bottango in the far right corner. We'll learn more about master live in Chapter 7: Hardware Drivers.

Simply click a mode button to enter that mode. The currently selected mode will be shown in a lighter gray than the others.

You can click the "X" button in the very left to close out of the current project and return to the home screen.

File Controls

Below the mode tabs, you'll see the file and tool controls.

The very first thing you'll see is the name of the current project. Since this is a new project, Bottango shows "Untitled Project."

To the right of the project name are save and load buttons. The disk button allows you to save the project. If the project has never been saved before, it will prompt you to choose a location and name for the project file. The folder icon allows you to load a Bottango project. Bottango project files have the ".btngo" extension.

Once you have saved a file, the name of the project will show up:

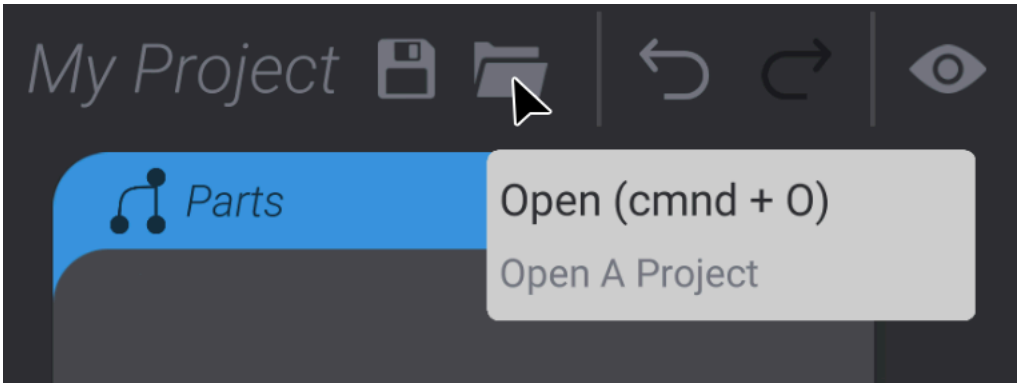
While you work on your project, if you have unsaved changes, the save icon will be badged to remind you:

Holding down shift while clicking the save button will allow you to "Save As" and create a duplicate version of the current project.

Bottango will as well create an "autosave" file while you have a project open. This autosave file will be located next to your original save file. If Bottango closes a project, the autosave file will be automatically deleted, unless Bottango encounters an error or crashes.

Tooltips

Many tools and options in Bottango have tooltips. Hover your mouse over a part of the UI to see a description and hotkeys for the element.



Tools

After the file name, save, and load buttons are a series of tools that you can use to configure and animate in Bottango. What tools are available depends on both the current mode you are working in, as well as your current selection. We'll talk about each tool more in the context that it is most commonly used, rather than cover them all now. Just know that you will only be presented tools that are useful to you in the context of your current actions.

However, two tools that are always available are the undo and redo buttons. The left facing arrow is Undo, and the right facing arrow is redo. Additionally, you can enter control/command + Z to undo, or control/command + shift + Z to redo.

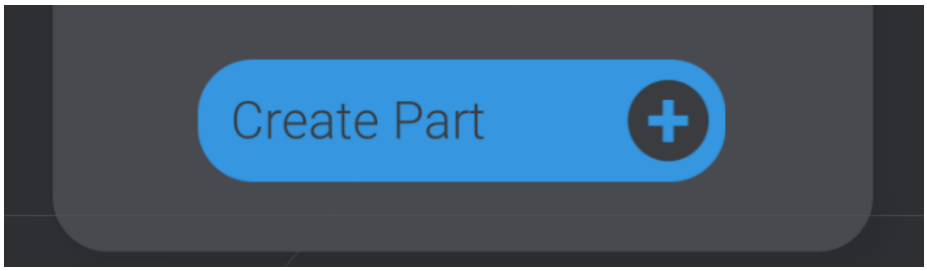
Selecting and Deselecting Parts

In Bottango a part is any 3D object you create to represent your robot. A part could be a piece of structure that makes up the bones of your virtual robot, a joint that rotates or moves, or it could be a motor that you animate with.

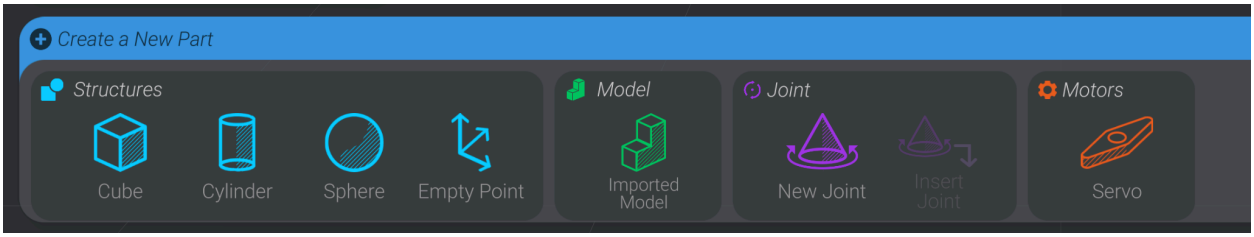
In Bottango, Parts are the 3D objects you create to represent your robot.

In order to try selecting parts, we first need to create a few parts. Don't worry too much about what these parts are and what they do; we'll learn more about them in later chapters.

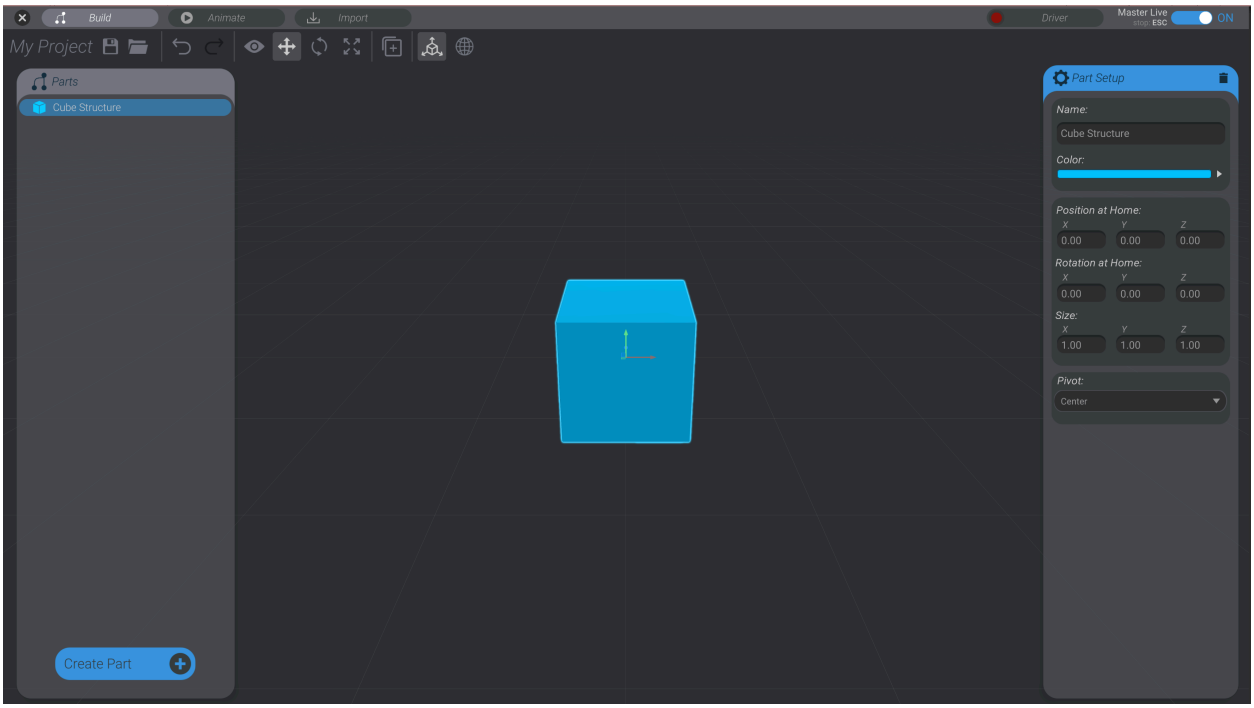
1. Click "Create Part" to create your first structure:



This will cause the create window to appear:



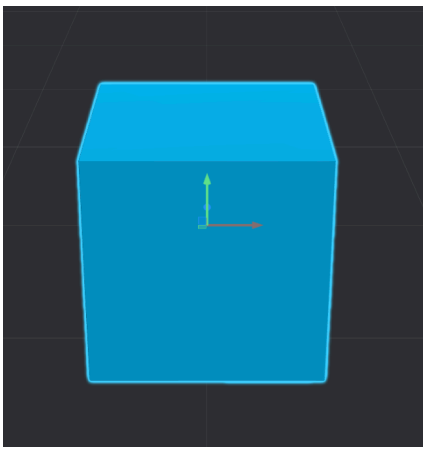
2. Click Cube to create a cube structure.



This will create a cube structure and add it to the project. The new cube structure is automatically selected. On the right side of the screen, you will see the Part Setup menu. Whenever you have one part selected, you will be able to configure that part.

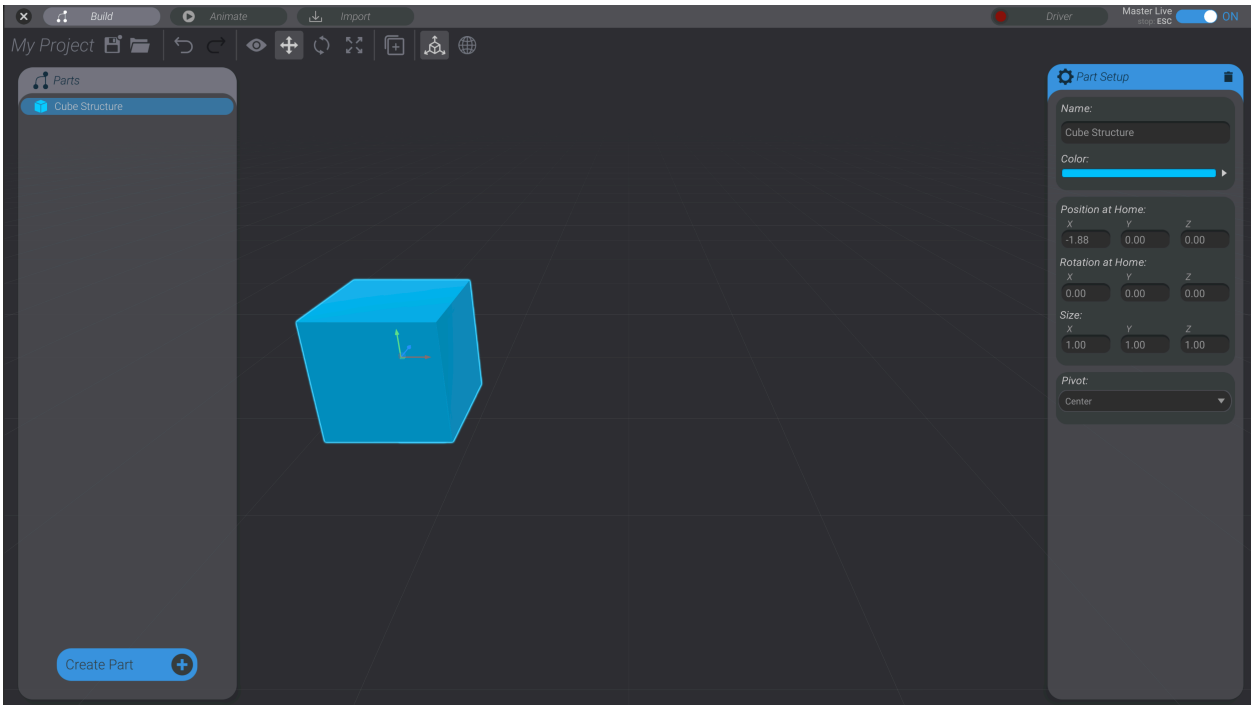
What shows up in the part setup window is different for every type of part. We'll learn more about the part setup window in later chapters.

3 Move the cube to the left a little bit.

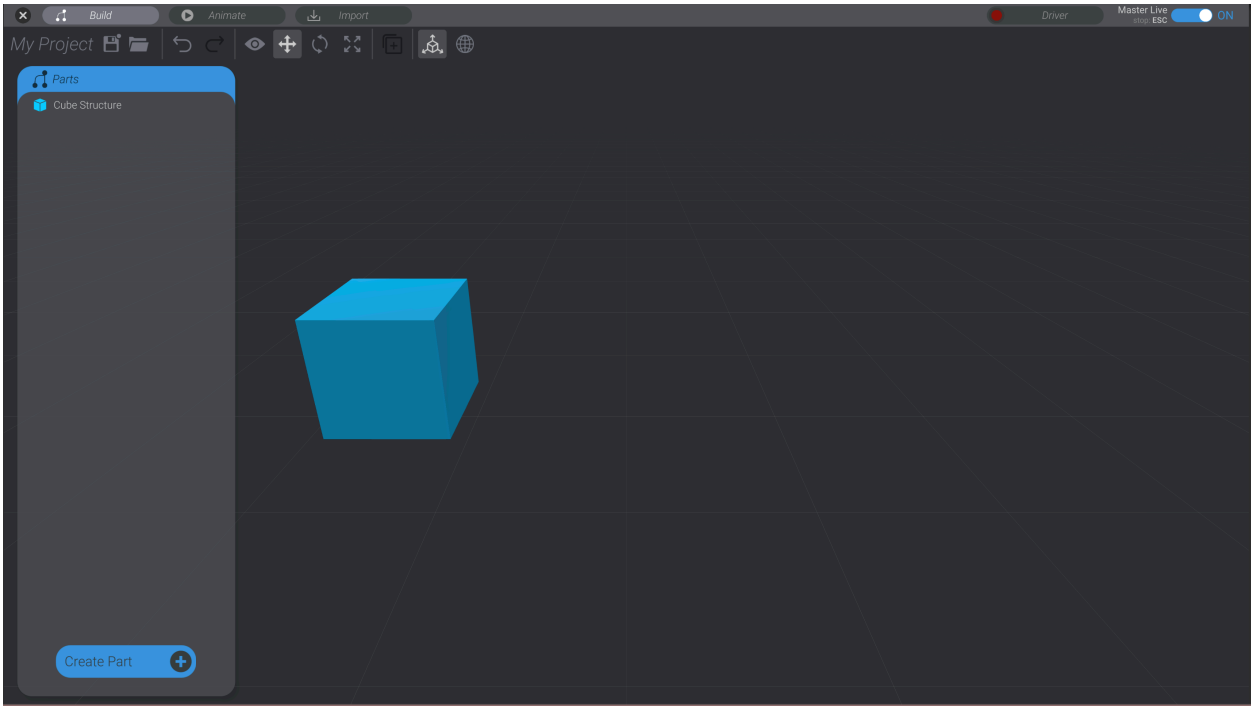


A selected part has a transformation handle to move it, which looks like a series of arrows. If for some reason you are missing the transformation handle, press the W key to select the “move tool.”

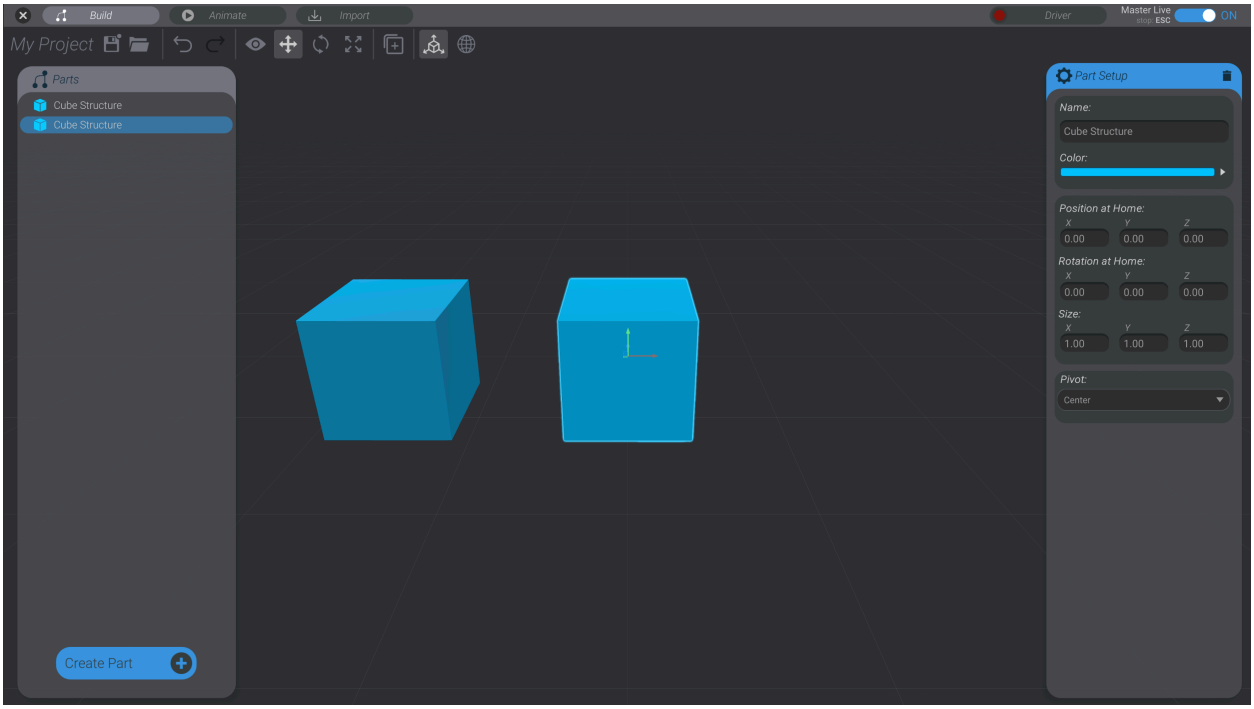
Click and drag on the red X axis handle arrow to move the created cube over to the left a bit.



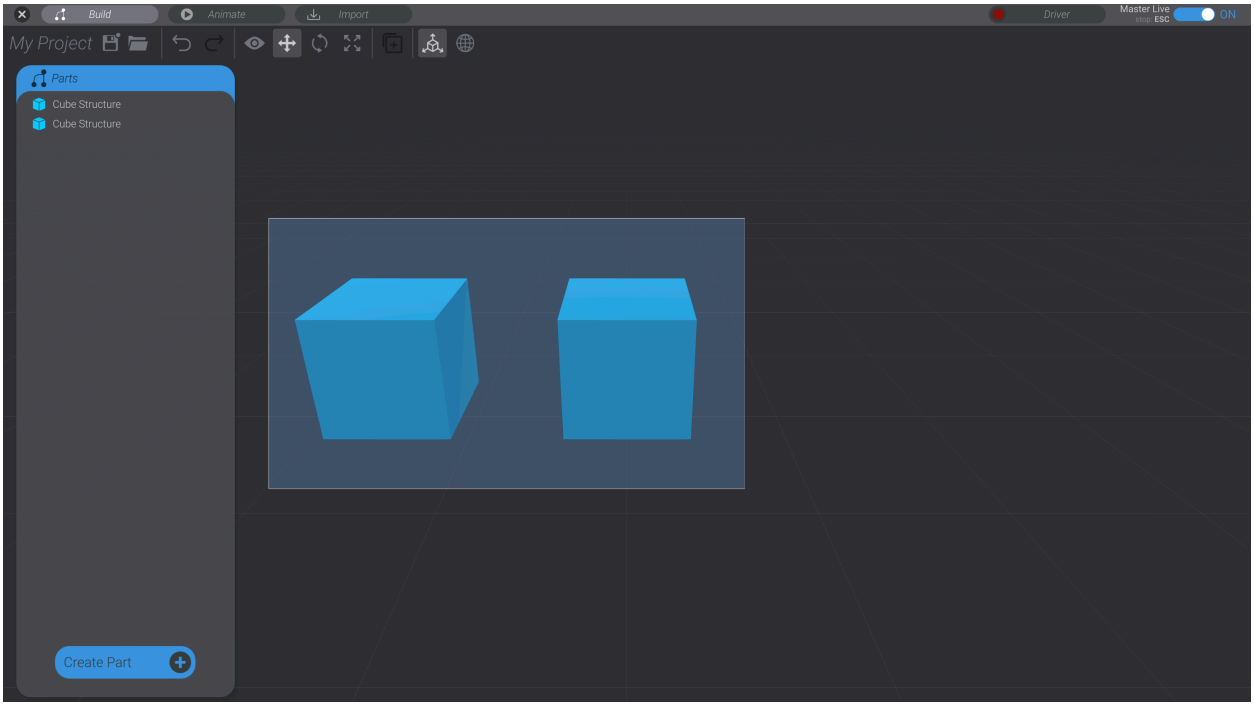
4 Click outside the cube on the background grid to deselect the cube. You will know you have deselected the cube when the transformation handle and part setup window disappears.



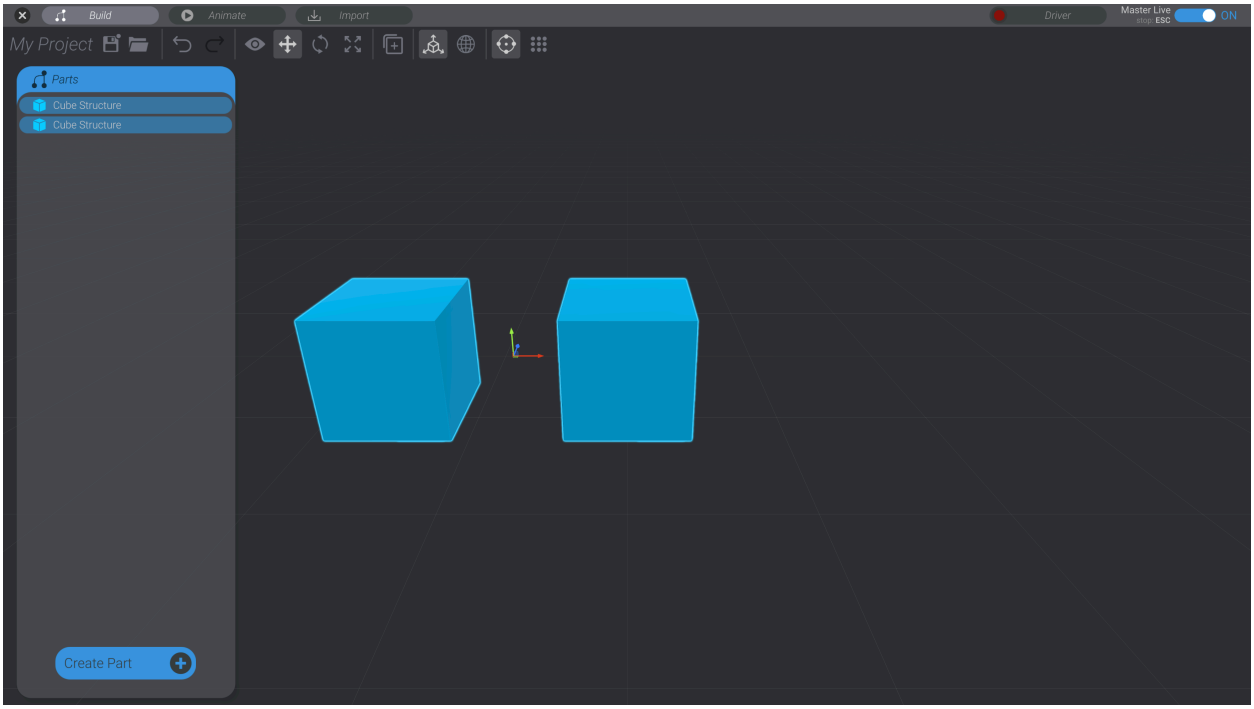
5 Repeat steps 1 and 2 to create a second cube.



6 Click and drag a box around both cubes to select both.

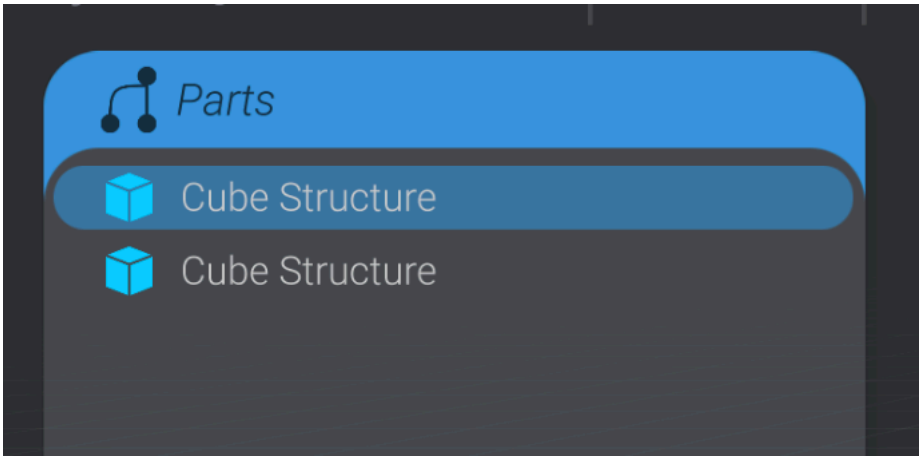


Both cubes are now selected. When more than one part is selected, you will not see the parts configuration window. You can only configure one part at a time. However, you can still move, rotate, and change the size of multiple parts if you have multiple selected.



Using the Parts Menu to select parts

Every part in your project will show up in the Parts Menu. You can use the menu to select and deselect parts.



Selection of parts in the Parts Menu follows the same rules as most file browsers. You can hold down shift to select or deselect in a range. You can hold down control/command to add and remove from a selection. Click in the parts menu outside of any part to deselect all selected parts.

Setting up Parts

When you select a single part, on the right side of the screen will be a menu to set up and configure the part. Each part has it's own set of controls, which will be covered in more detail in the following chapters. However, most all parts let you control the name, color, and visibility of the part.

You can edit the name of the part:

Change its color:

And toggle if it is visible or not:

If you make a part not visible, all of its children will be not visible as well.

You can as well delete selected parts by pressing the trash button, or hitting delete on your keyboard. All of the children parts of deleted parts will be deleted as well.

Camera Control

Bottango is a 3D application, so you can control your robot in 3D space.

1. Use the mouse scroll wheel to zoom in and out.

2. If you have deselected one of the cubes you created, reselect it. Press “F” while a part is selected to focus in on that part.

This will zoom in and center the camera in on the part. Additionally, the camera pivot will now be on that part, and rotate around it. If you have multiple parts selected, pressing F will focus the camera on the middle point between all selected parts.

3. Drag with right click OR hold down alt/option while dragging with left click to rotate the camera. The camera will rotate around the last set pivot.

4. Hold down alt/option + control/command while dragging with left click to pan the camera. Panning the camera will also move the camera pivot.

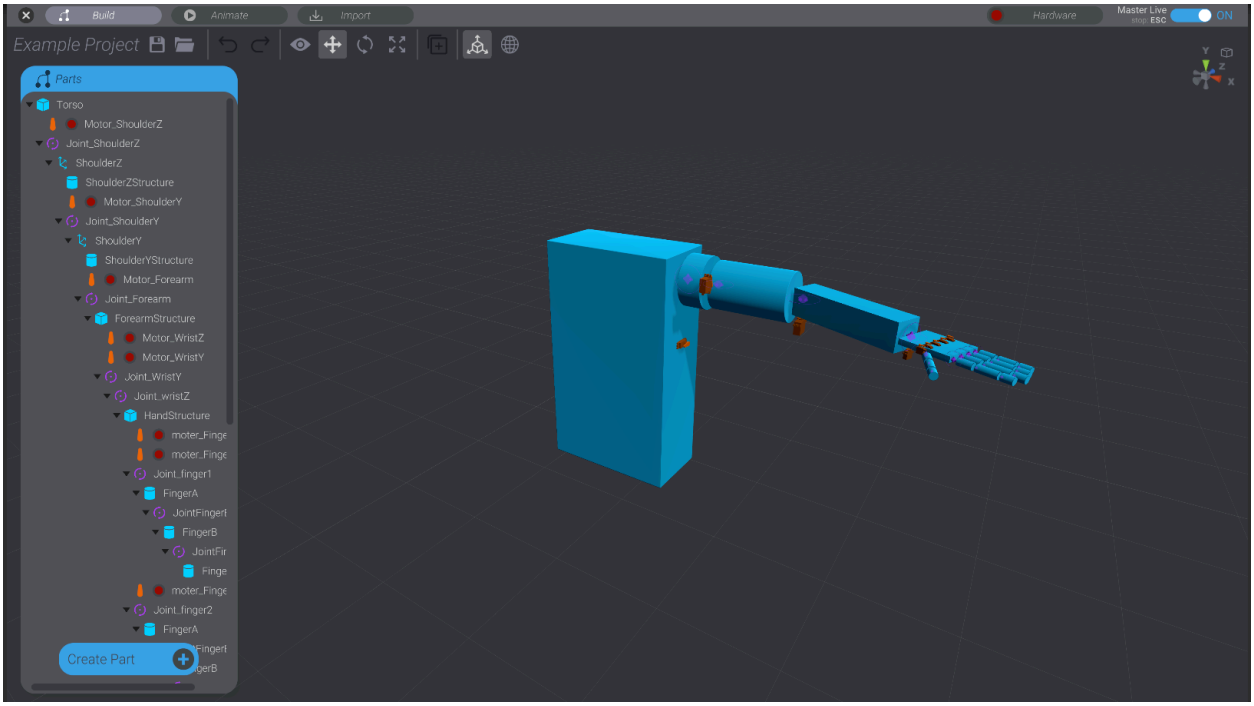
Camera Cube and Projection Switching

In the top right of the screen, you'll see a cube with axis buttons:

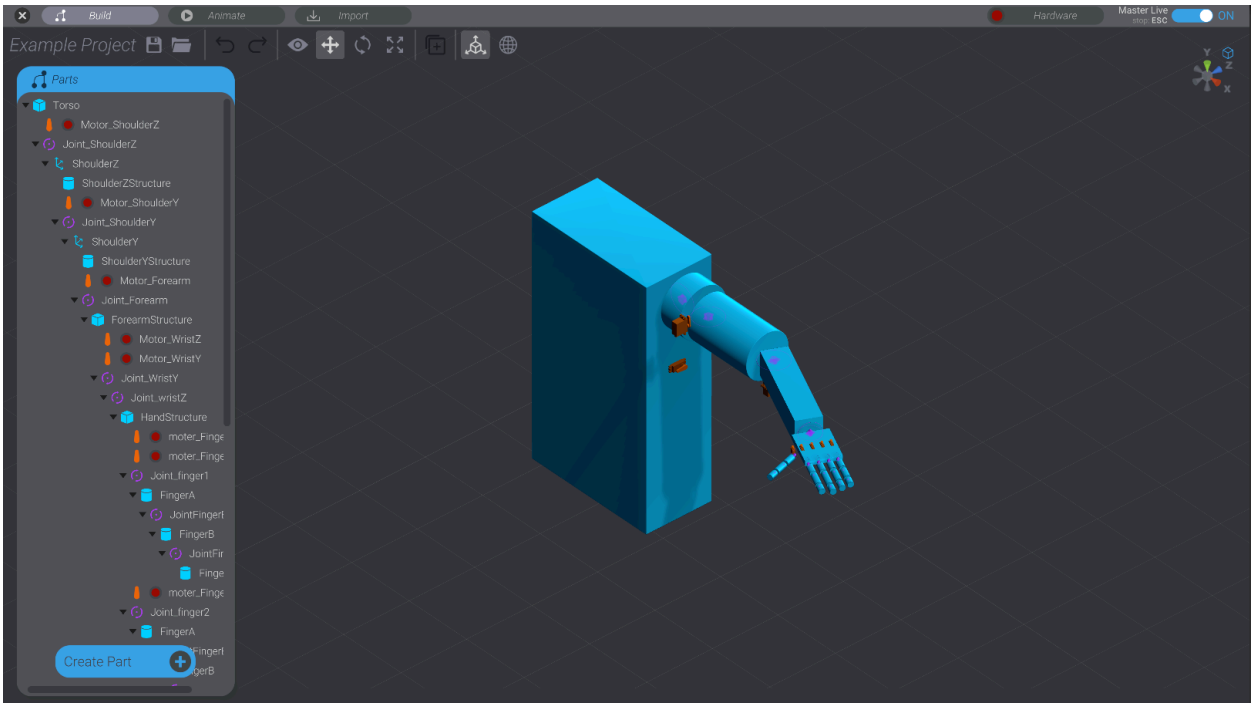
Click each axis to orient the camera in the clicked axis. You can click the center cube as well to return the camera to the home position.

As well, in the top right you'll see a cube icon. You can click the icon to toggle between a perspective projection and an orthographic projection.

Here is the standard perspective projection:



And the same view in an orthographic projection:



The orthographic projection can be useful for carefully aligning parts.

-6-
Creating Structure and Joints

What's in this chapter

In this chapter, we'll create the basic structure of your robot.

In order to move and animate your robot, first you need to create a virtual representation of it. Bottango provides very basic modeling tools to create a structure that represents your robot.

As well, you will create Joints. Joints are the points of articulation in your robot that allow you to configure how your robot moves.

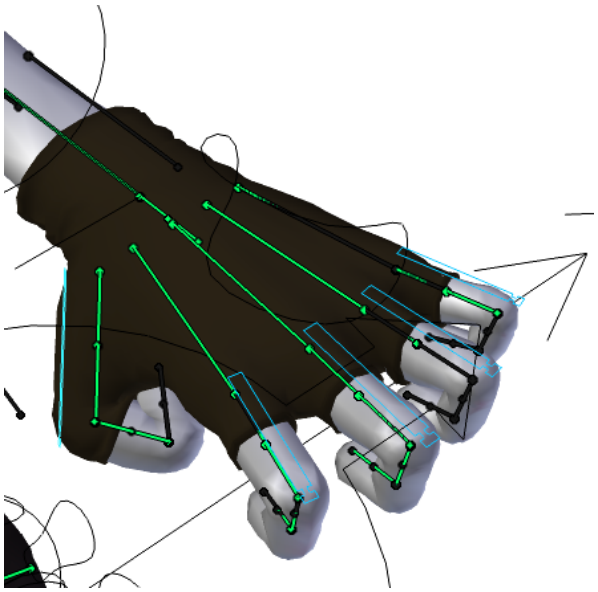
Structures should be approximations

The structures you model in Bottango should not be picture-perfect representations of your robot. Instead, they are just the virtual controls that you will manipulate to move your robot and visualize its movement.

In Bottango, Structures are the virtual representation of your robot's physical form.

3D animation involves a process called rigging, in which you create the bones and interaction points (known as a rig) that you then animate to move a 3D object. When animating a 3D model, very rarely are you actually manipulating the object itself; instead, you manipulate the rig that then moves the object.

The figure below shows a 3D model (the gray and black hand) and a rig (the green lines). An animator would animate the green rigging, which then moves the 3D modeled hand.

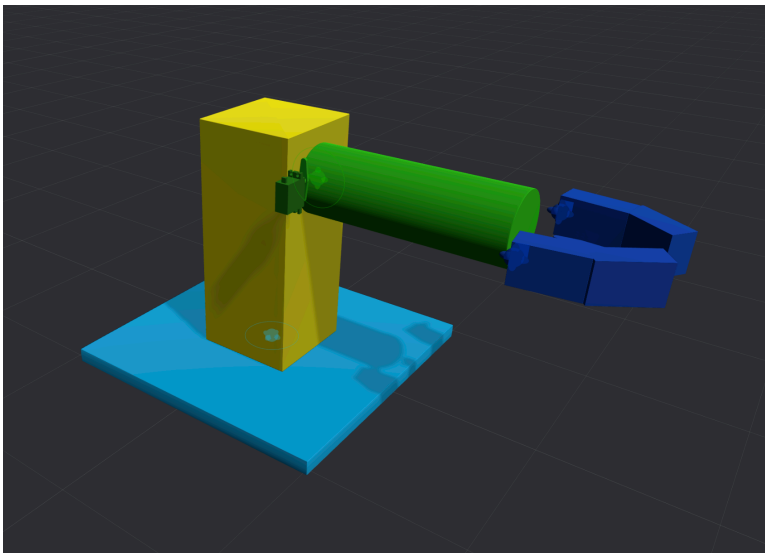


("Sintel-hand (cropped)" © Blender Foundation / www.sintel.org / CC BY-SA 3.0)

The structures you create in Bottango operate in the same mindset. Your goal is not to recreate your robot, but instead to create approximations that are just detailed enough to view and create animations.

In Bottango, the structures you make are the equivalent of rigging, and your real world robot is the equivalent of an animated 3D model.

Here is an example of a simple robot structure built in Bottango:



This robot has three points of articulation:

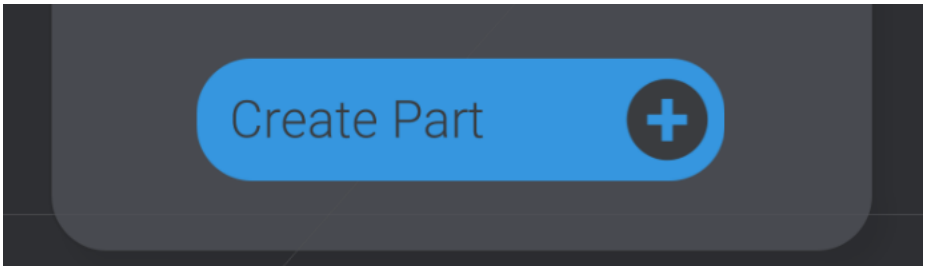
A. It rotates the yellow base extension on the Y axis.
B. It rotates the green arm its arm up and down on the X axis.
C. It opens and closes the blue gripper.

Obviously, the actual physical robot is far more complicated than this. However, this level of detail is all that's needed in Bottango to visualize and control movements.

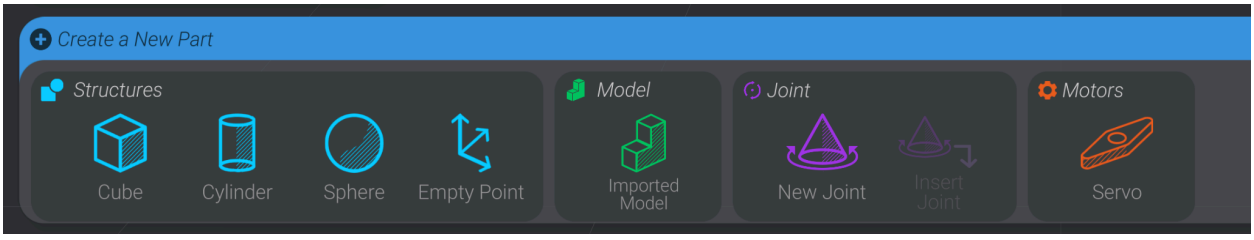
How to create structure - Creating the base

Let's build the structure of a simple robot that looks a lot like the one above. Open a new, blank project in Bottango.

1 Click Create Part to create your first structure:



This will cause the create window to show up:

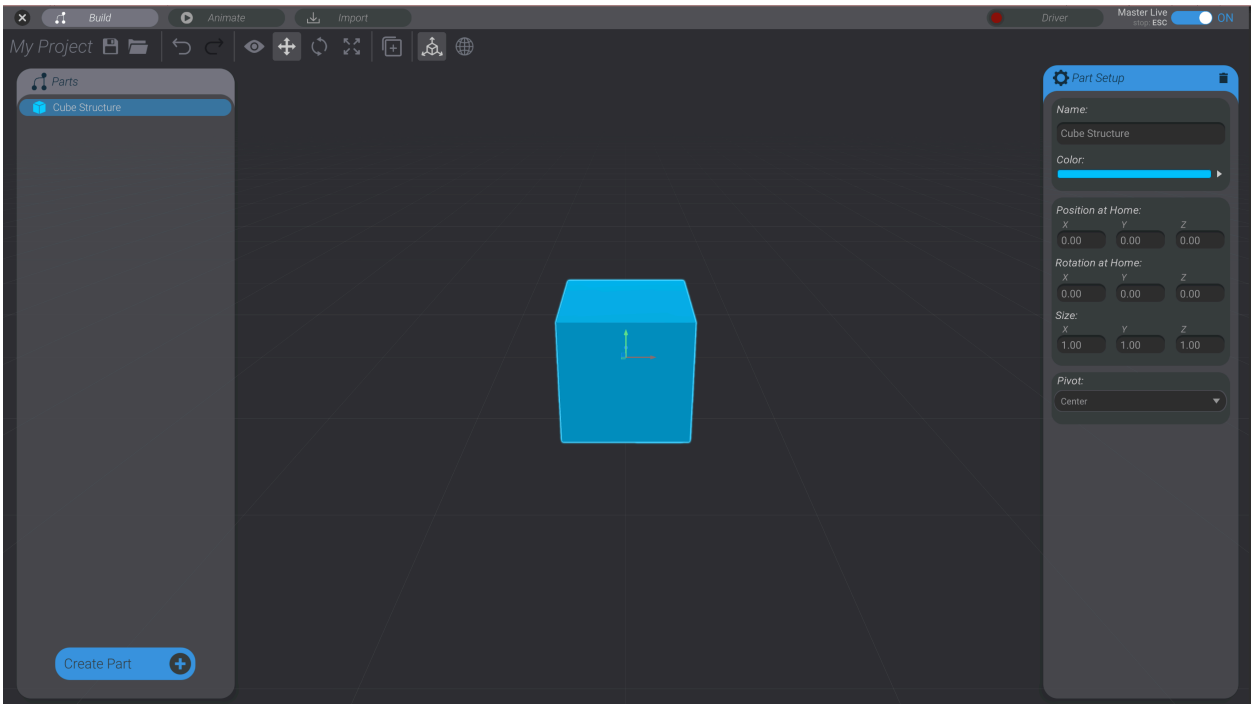


There are currently four types of structures:

- Cubes
- Cylinders
- Spheres
- Empty Points

Cubes, cylinders, and spheres should be self-explanatory. Empty points can be useful when you need to group a series of parts under one invisible parent, or when you want to change the local axis of child parts.

2 Click Cube to create a new cube structure.

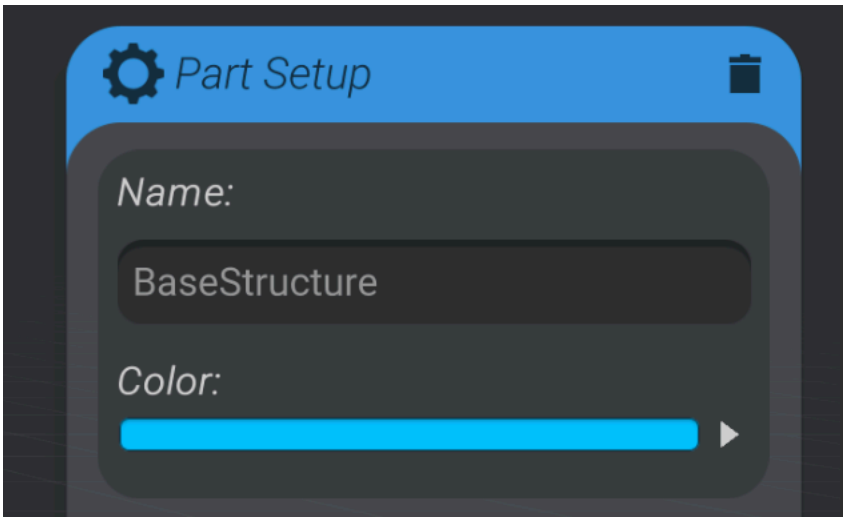


Every structure has the following characteristics:

- Name
- Color
- Position at Home
- Rotation at Home
- Size
- Pivot

You will see all these settings for the structure in the Part Setup window, whenever one structure is selected. Some structure types have additional settings. For example, you can change the sphere structure type to a hemisphere.

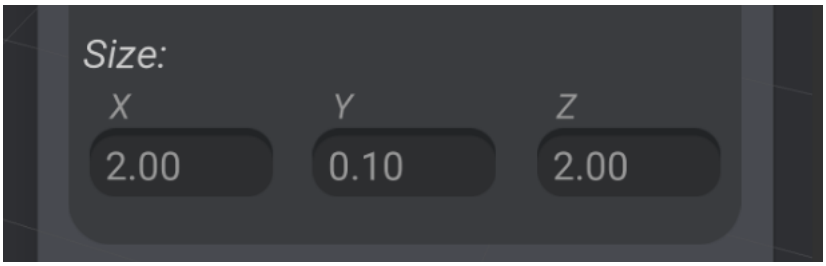
3 Rename the cube to “BaseStructure” using the Name field in the Part Setup window.



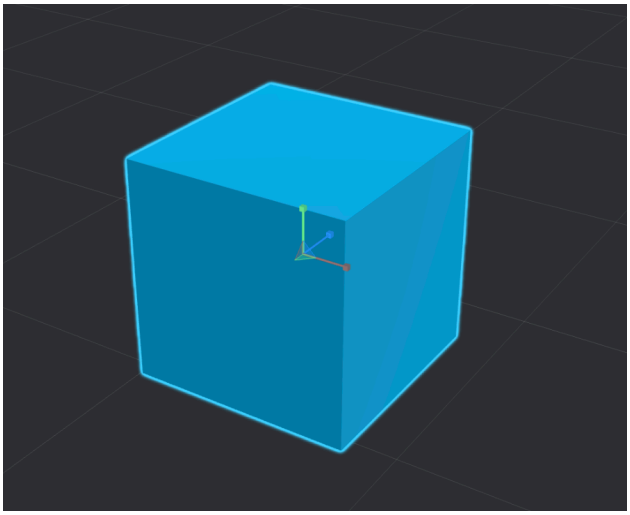
4 Resize the cube to 2.0 in the X dimension , 0.1 in the Y dimension, and 2.0 in the Z dimension.

There are three ways you can change the size of a structure:

1. You can enter the desired size in the text input fields under size.
2. You can click and drag left and right on the X, Y, or Z label to drag the number higher or lower. In fact, every number entry field in Bottango can be dragged in this way. This is often a great way to make quick adjustments to numbers.



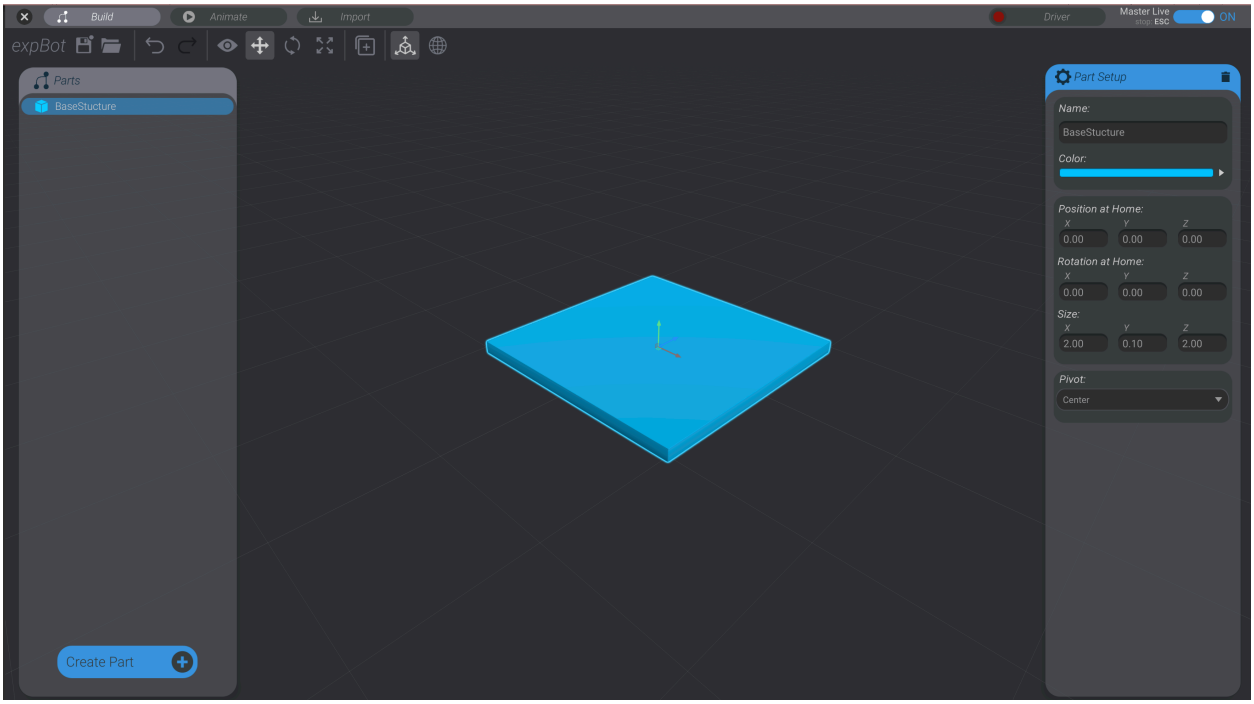
3. You can use the scale handles to adjust the size of selected structures.



Select the scale tool in the tools panel to use the scale handle:

You can also press the R key to quickly change to the scale handle.

At this point you should have a single cube structure named “BaseStructure” in roughly this size:



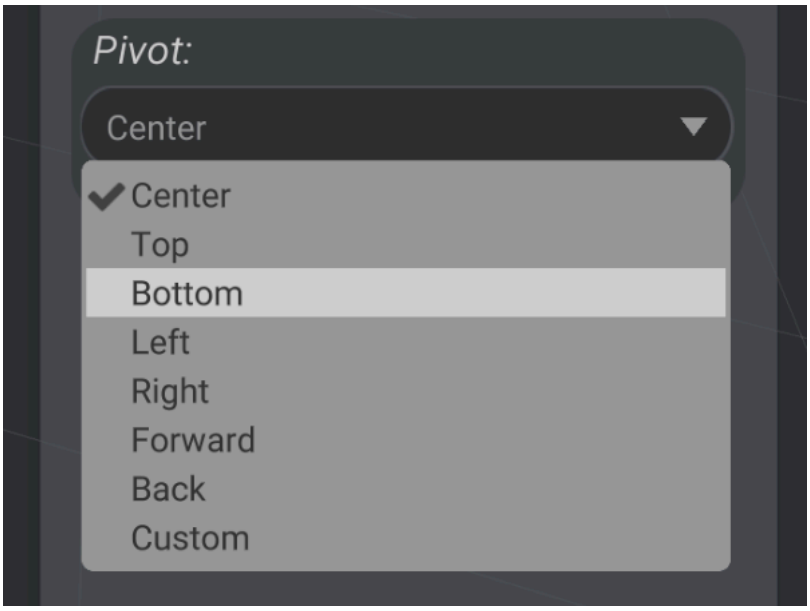
Creating the base extension

The base of this simple robot has two parts: the flat base we created already, and the taller extension.

1. Deselect the BaseStructure part so that nothing is selected.

2. Create a new cube structure, and name it “BaseExtension.”

3. Change the pivot of “BaseExtension” to “Bottom”.



Pivots

Every structure has a pivot. You’ll know where the pivot on a structure is by where observing where the move/rotate/scale handle originates from.

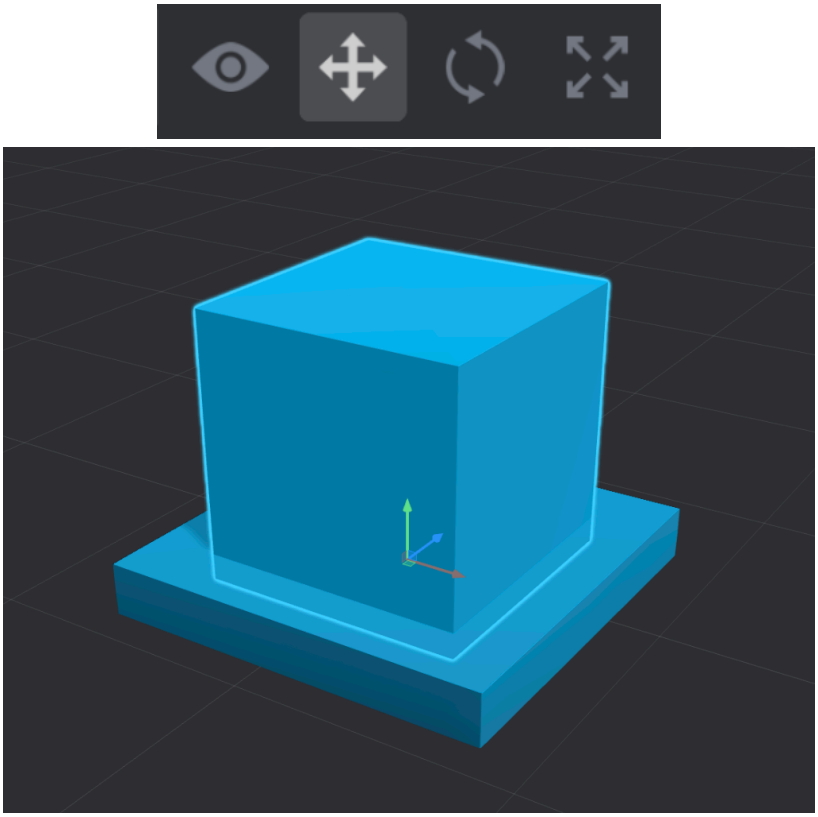
> In Bottango, every structure has a Pivot. The pivot of a structure is the point on the structure from which size, position, and rotation are calculated.

We changed the pivot of BaseExtension from center to bottom. This means when we resize it, it will change its size leaving the bottom where it is, and move the top up and down.

1 Set the position of BaseExtension to be right at the top of BaseStructure.

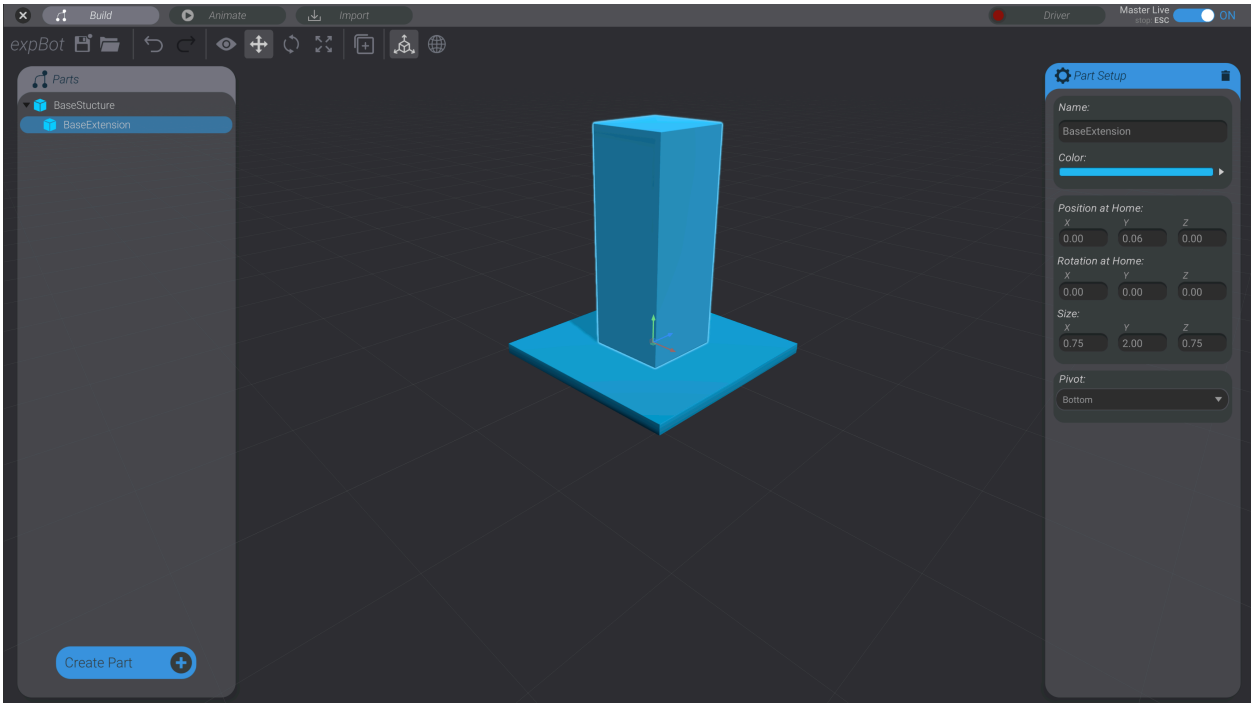
Just as with changing the size of a structure, there are three ways to move a part:

1. You can enter the desired position in the text input fields under position.
2. You can click and drag left and right on the X, Y, or Z label to drag the numbers higher or lower.
3. You can use the position handles to adjust the position of selected structures.



To change the active handles to position handles, click the tool icon, or press W.

2 Change the size of BaseExtension to 0.75, 2.0, 0.75.



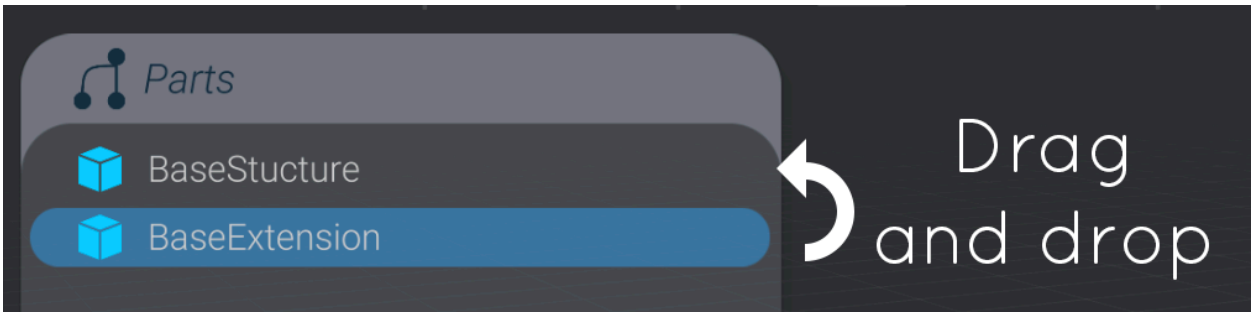
Again, because the pivot is bottom, the size changes leave the bottom of BaseExtension exactly where we want it.

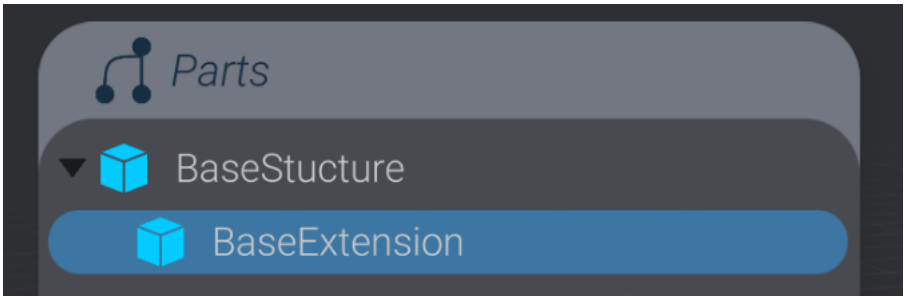
Parent-Child Relationships

Maintaining the parent-child relationships of your structure is essential to creating usable structure. Any structure can be a parent of other structures. When a parent structure moves or rotates, all of its children move and rotate with it.

> In Bottango, Structures can be Parents. When a parent moves and rotates, the Children of the parent move and rotate with it.

1 Using the Parts Menu, select and drag and drop “BaseExtension” on to “BaseStucture” to parent it to “BaseStructure.”





Now, if BaseStructure moves or rotates, BaseExtension will move and rotate with it.

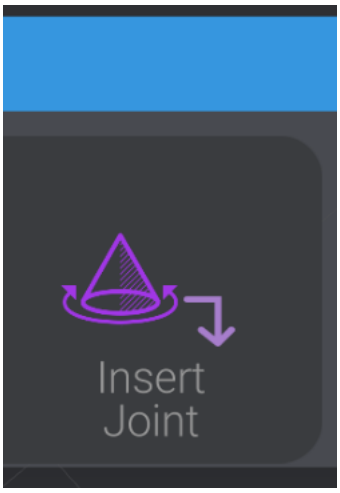
Initial Joints

Structures are the bones of your robot. But just like bones, they don't move on their own. They need muscles (i.e. motors) and joints. BaseExtension will not rotate on its own; it needs to be a child of a joint that rotates.

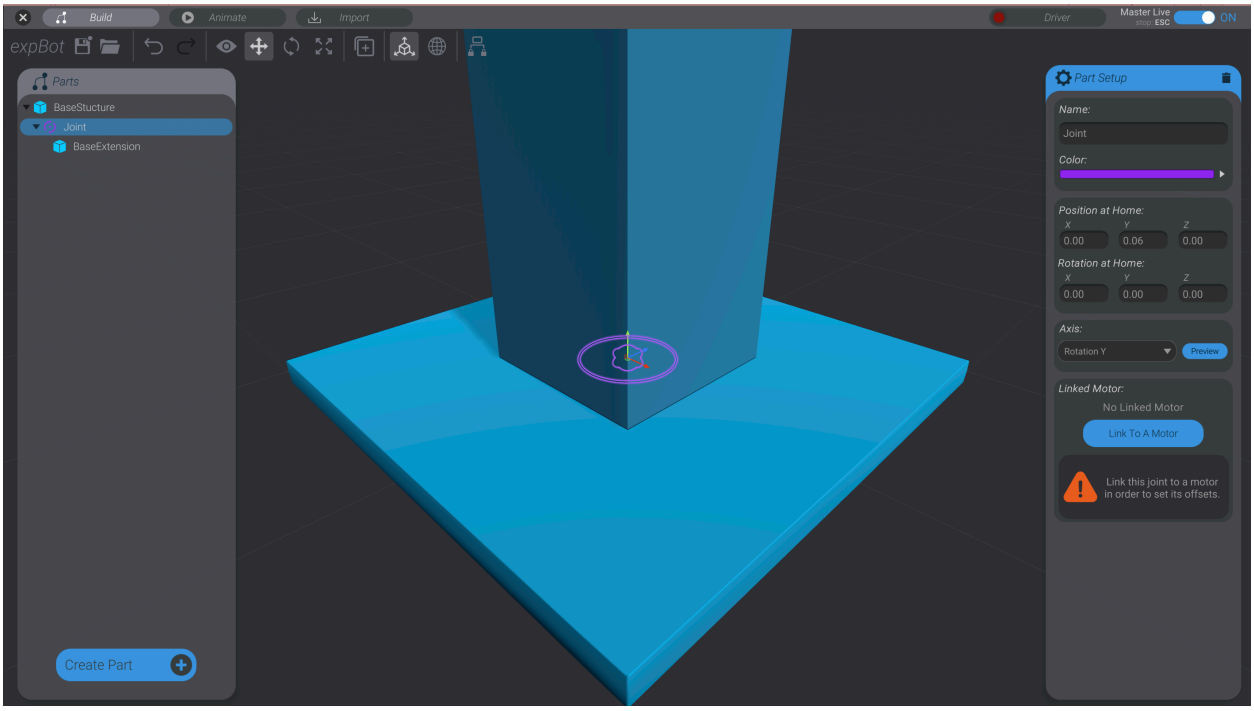
1. Select BaseExtension if it is not already selected.

2. Click Create Part.

3 Click Insert Joint.



This will reparent the selected structure under a new joint:

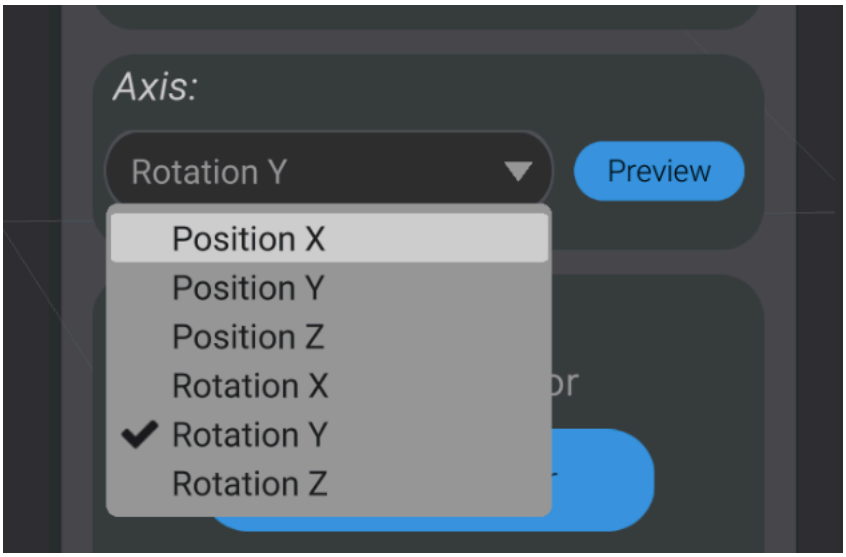


You can see in the above image that BaseExtension is now the child of a joint, instead of BaseStructure.

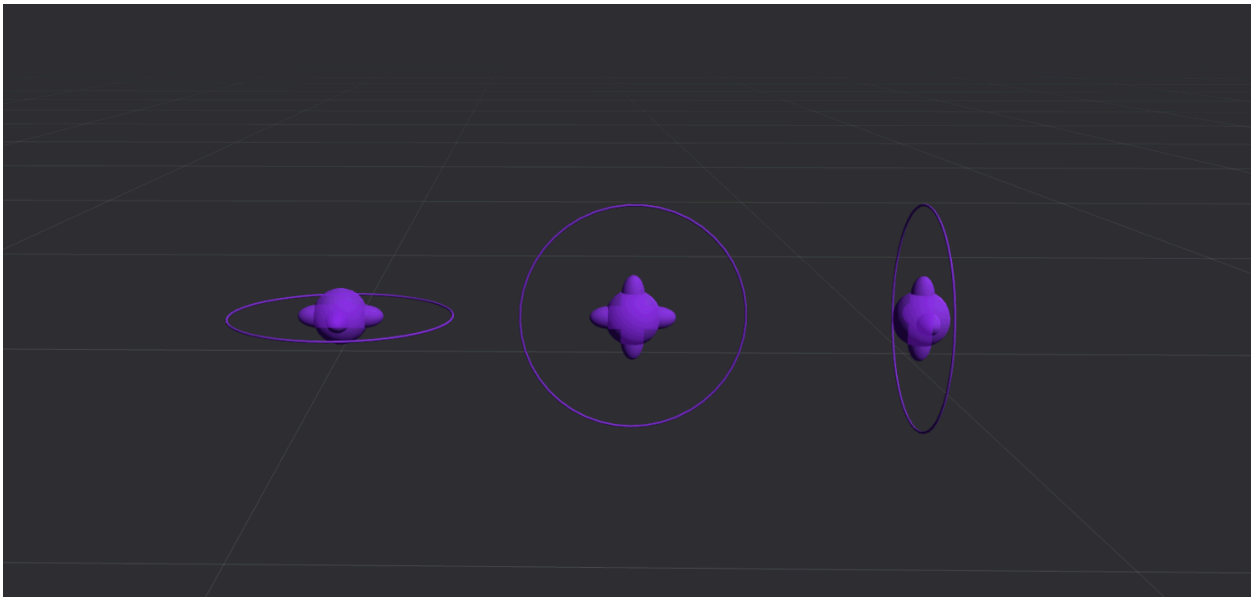
Joint Axis

A joint has a single axis of effect. A joint can either move or rotate, and in the local X, Y, or Z direction. In the above example, the joint has the default setting of Y rotation.

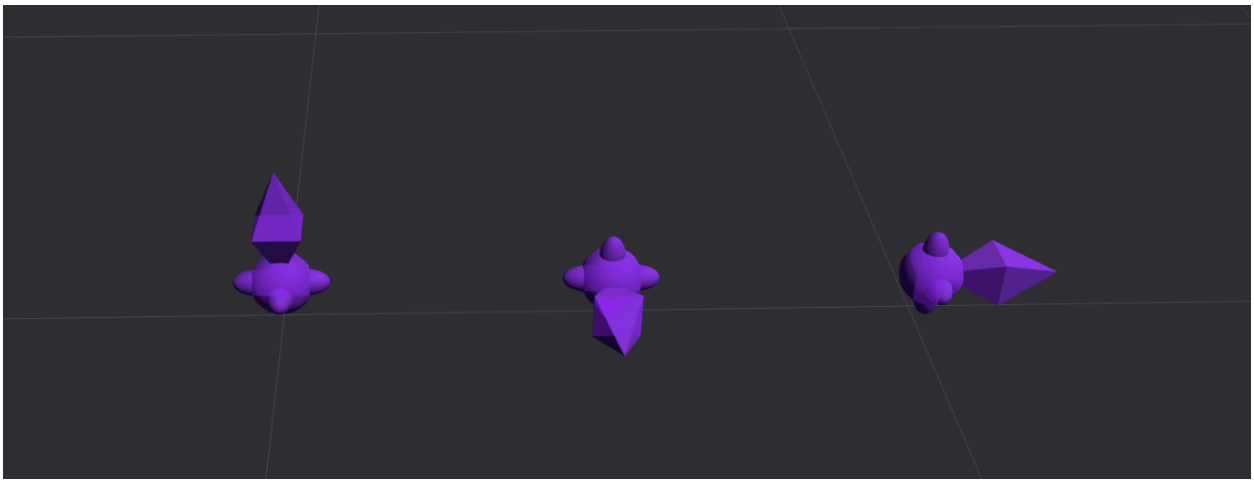
You can use the Axis dropdown to change the axis of a selected joint:



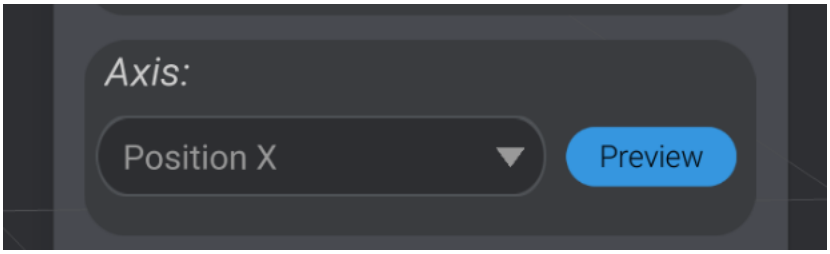
As you change the axis of a joint, its 3D representation changes as well. For a joint that rotates, the ring around the joint is oriented in the axis of the joint. Here, for example, are three joints, each configured to rotate in one of the three axes:



Joints can change position instead of rotate. Here are the same joints, with matching axis, that move instead:



If at any point you're not sure if you have the right axis for a joint, you can click the preview button, which will quickly animate the joint in the selected axis, to ensure that you are moving/rotating the joint as intended.



Why make joints now?

I thought we were making structure! Why are we making joints now? Because ultimately we want this robot to be able to move, so we want to create the structures of the robot as children of joints that will move. In a later chapter we will configure the joints to actually work, but for now, we just want to make sure the parent-child relationships of our robot is correct.

Joints can be parents of structures just like structures can be parents of other structures. Since joints move using motors, when parent joints move, all children move and rotate with it.

You can create joints, as we just did, by inserting via the create menu. You can also create new joints, and then create new structures as the children of the joint:

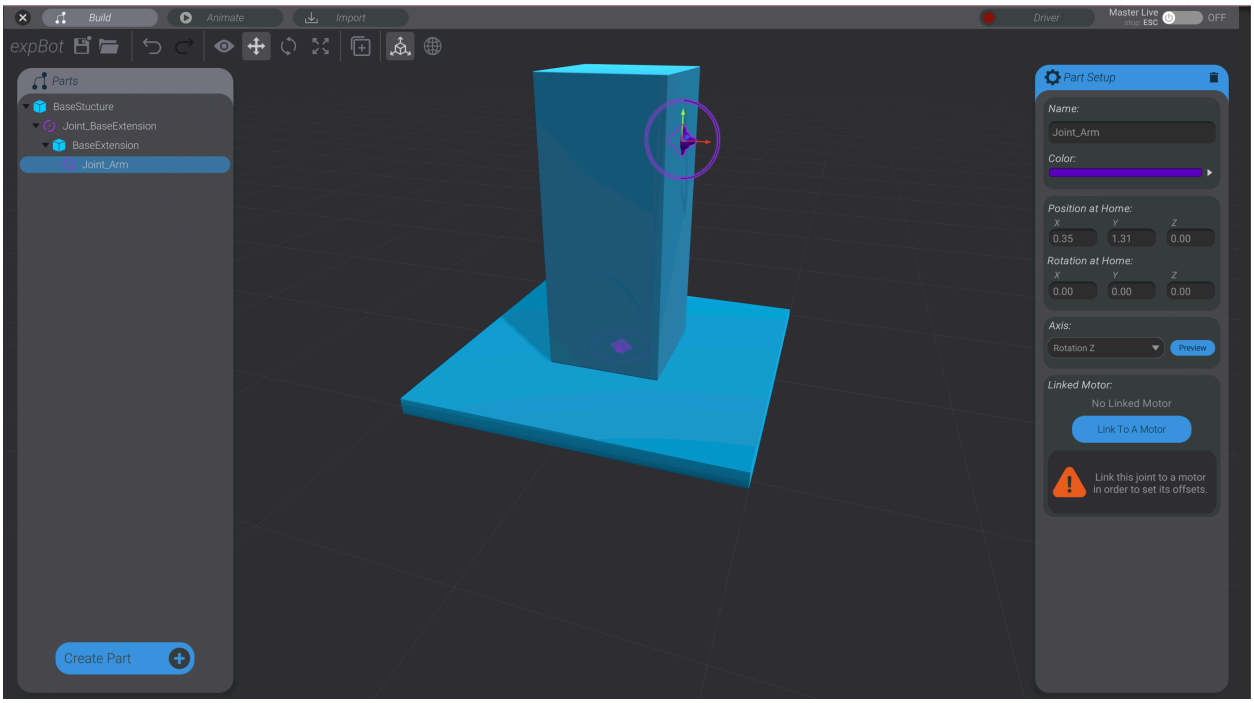
1. Select BaseExtension if it is not already selected.

2. Click Create Part.

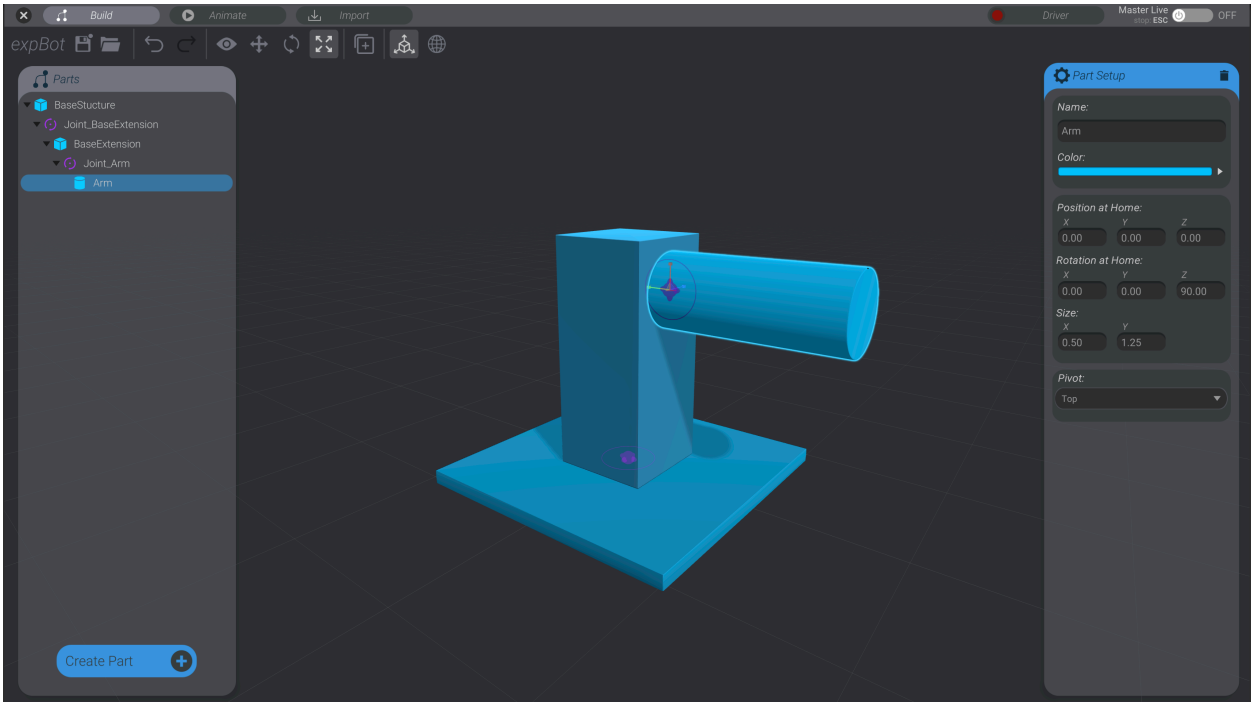
3 Click Create New Joint.

This will create a new, empty joint as a child of BaseExtension.

4 Move the new joint to be at the top and side of BaseExtension. Set the joint to rotate on the Z Axis.



5 Create and orient a cylinder so that it acts as the arm section of the robot, as a child of this new joint.



Home position and rotation

Robots move. That's the whole point of a robot! So since robots can be in all kinds of positions and configurations, how do we decide what configuration we should model when creating structure and joints?

The position and rotation you set up when creating structure is the "Home" position and rotation. This is the position and rotation the robot will return to "at rest." It's also the position and rotation that the structure will originate from when animated.

In Bottango, Home is the position and rotation a structure will originate from when animated, and return to at rest.

Let's take, for example, a simple animatronic eyeball, controlled by a servo, that can move the pupil left and right and also up and down. The home position of the pupil is probably looking straight forward. With a servo, nothing says it has to be that way; it just probably makes the most sense in your head to think about the neutral position as looking forward, and then as you animate and configure the joints that will move the eyeball, you'll define the how far left, right, up and down it can look.

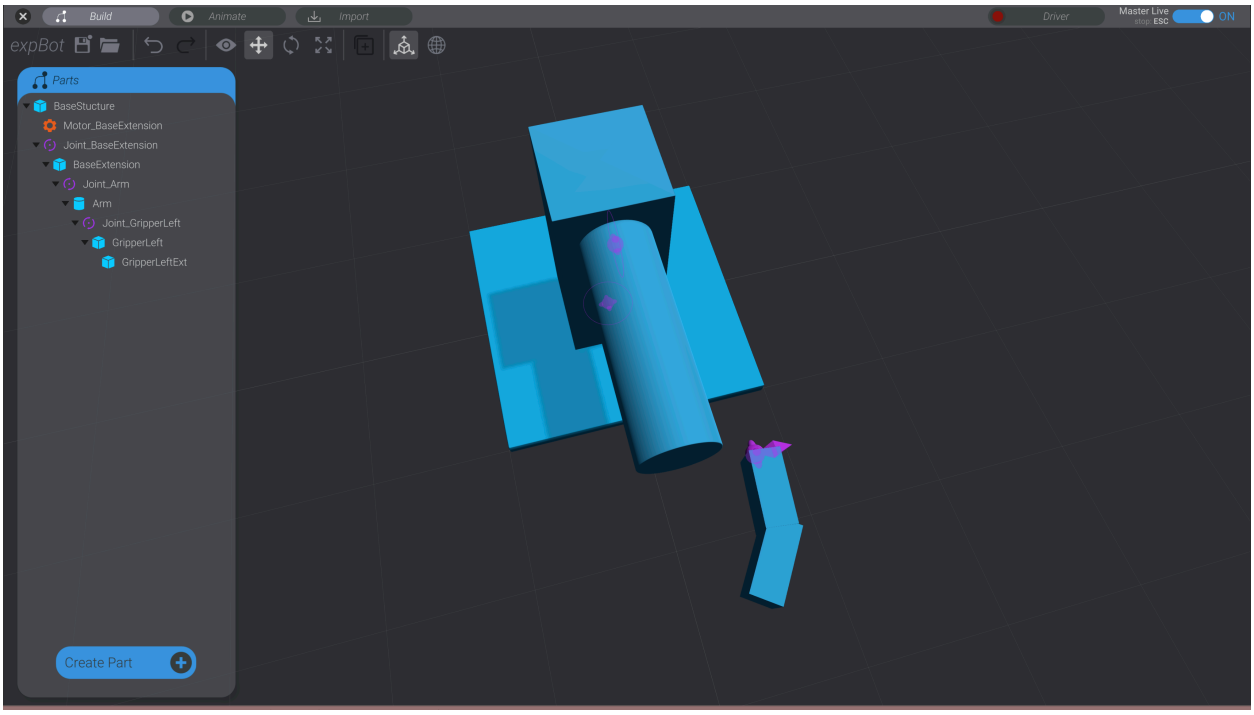
Let's take another example: A servo that opens and closes a mouth. In this example, halfway open, right in the middle, probably isn't what you'd think of as "home" for the mouth. Home for the mouth probably is all the way closed. As you model that structure, you would model it as a closed mouth, that you then open with joints and motors.

So there isn't always one exact answer for where home is on your robot. You'll have to decide what makes sense for you, and model towards that. You may also want to keep in mind the eventual home value of the motor you will link later to this joint. For example, a stepper motor may have a much more set in stone home value than a servo motor, and you may want to model your robot to have home joint positions the same as the home positions of your stepper motors.

Create the gripper

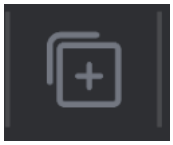
1 Using everything you've learned so far, create the first half of the gripper as the child of a joint that moves in the Z axis.

The gripper in this robot is made out of two parts that move in opposite directions. Because of that, each side should be a child of its own joint. That way we can configure each joint to move in opposite directions as the motor they both use is driven.



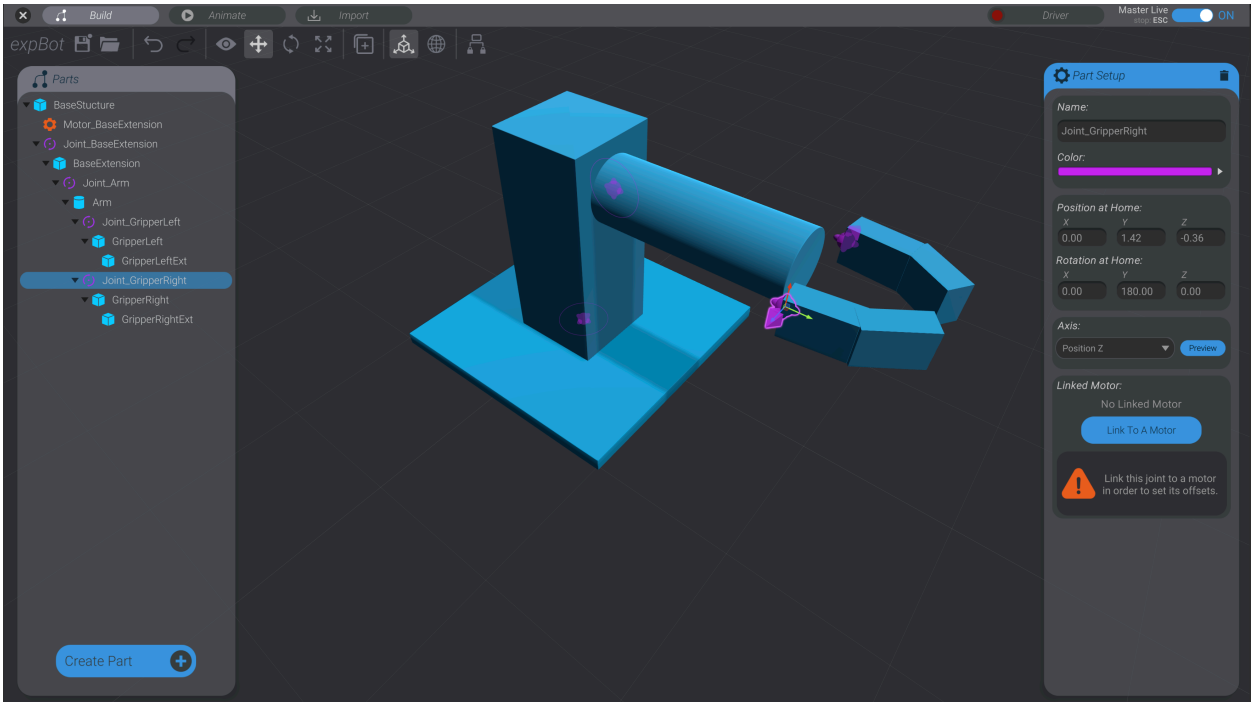
2 Duplicate the left gripper joint and associated child parts.

In the tools panel of Bottango, there's a "Duplicate" button. Click that, or press control/command + D with at least one part selected to duplicate the selected parts. When you duplicate a parent, its children are duplicated as well.



Duplicating LeftGripper should keep the newly duplicated structures as children of Arm. This is exactly what we want.

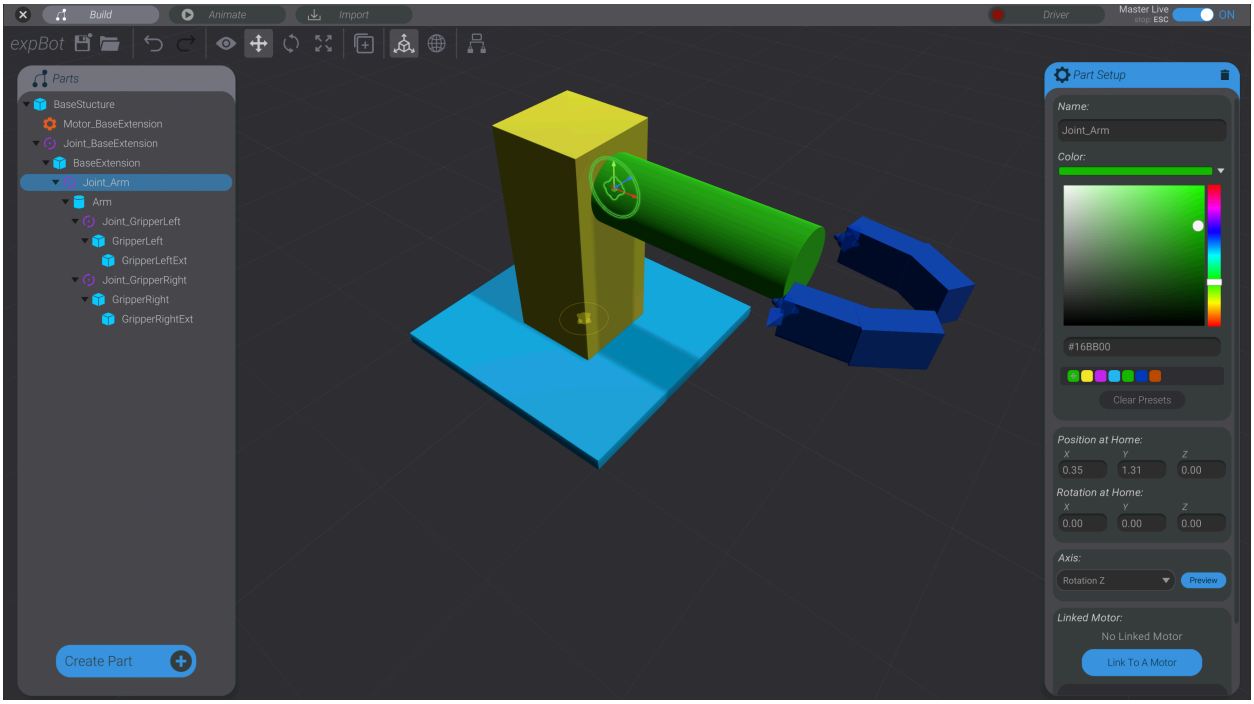
3 Move, rotate, and rename the duplicated gripper to form the other half of the gripper.



Part color

Every part has a color attribute. You can change the color to whatever you like; it's only used for your organization.

Here you can see I've colorized the structures and joints by point of articulation:



Child lock

Sometimes, you may want to move or rotate a joint, but keep the structures, motors, and other joints that are a child of the joint in place. Child lock allows you to do that.

In the tools panel, if you select any part that has any children, you will see the child lock toggle show up:

Click to enable or disable child lock. While enabled, any movement or rotation you apply to a part will not affect that part's children.

Wrapping up

That covers it for creating the structure and joints of your robot. Next up we'll get Bottango talking to a hardware driver. But before you do anything else, don't forget to save your project!

-7-
Hardware Drivers

What's in this chapter

Bottango can't move motors or control effectors on its own; it needs at least one microcontroller to communicate with and send commands to. In Bottango, a microcontroller that responds to commands from your computer and drives motors and other effectors is a hardware driver.

> In Bottango, a Hardware Driver is a microcontroller that responds to commands from your computer and drives motors and other effectors.

In this chapter we'll create and configure hardware drivers so you can communicate from Bottango to real world robots.

Preparing a hardware driver

A hardware driver can be anything you want that you can program and that can create a serial connection to your computer. If you are so inclined, you are welcome to create your own personal implementation of the Bottango serial protocol. However, you don't need to. In the most common case, a hardware driver is an Arduino, or a similar user-programmable microcontroller.

Bottango provides not just the program that runs on your computer, but also a fully functional Arduino-compatible program to upload to your microcontroller. This program is able to open a connection to Bottango running on your computer and execute the commands Bottango sends to it. In order for Bottango to do anything, you need to upload the provided Arduino-compatible code to a microcontroller, and connect it via USB to your computer.

Included in the archive that contains the Bottango application is a library of Arduino compatible code. In addition, there is a separate read-me and instruction set for installing the provided Arduino-compatible code onto a microcontroller.

If you're not really comfortable with coding, you can just follow the separate instructions to upload the provided program to a microcontroller and forget about it. However, if you are more code savvy, I encourage you to examine the program and get a sense of how it works. If you familiarize yourself with the kinds of commands the microcontroller is programed to respond to, you can modify how your microcontroller responds to Bottango commands away from the default, if your robot requires it.

As an example, you could set an LED to change from red to green whenever a particular servo is moving by modifying the provided microcontroller code… or really, whatever your robot requires.

In the future, I plan to add the ability for Bottango to send user-defined events to your microcontroller on the animation timeline. When that functionality exists, the provided code will handle signaling that the event happened, but it will be up to you to program what the event actually does.

But for now, if you're just looking for the out-of-the-box experience, follow the instructions to upload the code to a microcontroller, and leave the microcontroller connected to your computer via USB.

(Side note: make sure you close the serial monitor in your microcontroller IDE. Bottango requires an open serial connection with a microcontroller, and most microcontrollers can only have one open serial connection at a time. Bottango will warn you if you try and connect to a busy serial port.)

Keeping your hardware driver up to date

Bottango has a very exact communication protocol with connected hardware drivers. As such, as Bottango develops and evolves, so too does the communication protocol.

Every time you download a new version of Bottango, make sure to update the code on your hardware driver as well, using the microcontroller code included with the new version of Bottango. Until you do, Bottango likely won't be able to communicate at all with your hardware driver.

Connecting to a hardware driver

For the rest of this chapter, I'm going to assume that you're using an Arduino microcontroller as your hardware driver, and that it has stock Bottango code running on it. Feel free to do different and more advanced things, but where you deviate it's up to you to figure out how to make it work!

What happens when you're running Bottango, and have an up-to-date Arduino connected to your computer via USB, is a little different for Macintosh/Linux and Windows users. This is because the name and specificity of what Bottango can see in order to connect is different on both platforms.

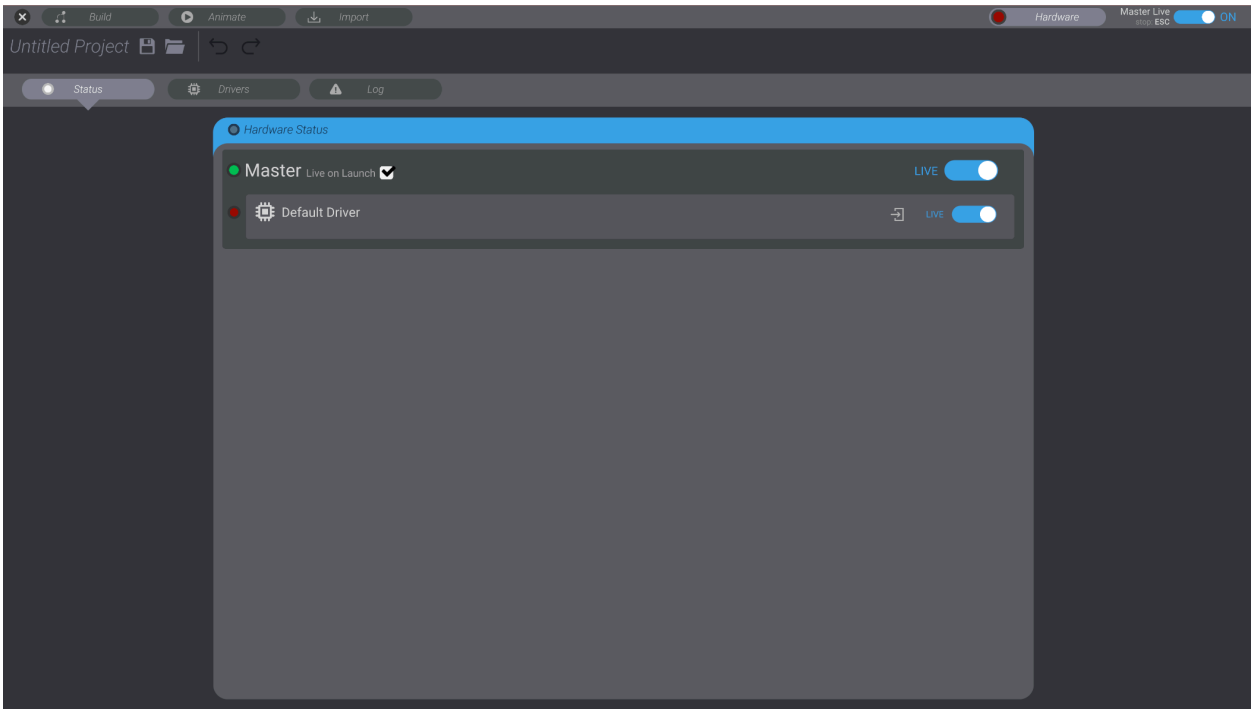
On Macintosh/Linux computers, it's likely that Bottango will find and connect to your Arduino right away, without any input needed from you. On Windows computers, you're going to need to tell Bottango what port your Arduino is on. We'll cover how to do that in later sections in this chapter.

In the top right corner of the Bottango window is the hardware mode button, to the right of the "Master Live" toggle. There will also be a red, green, or yellow circle in the button.

Go ahead and click driver to change to hardware mode, regardless of what state you see.

Hardware mode - Status

The first view you'll see in hardware mode is a view of the status of all your hardware:



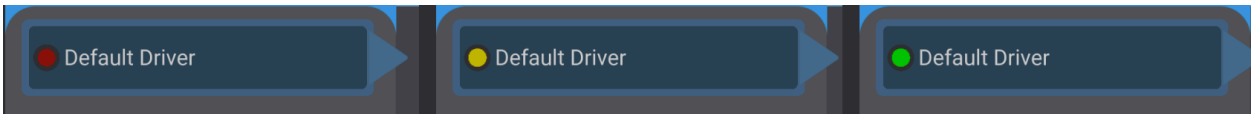
To start, you should only have two entities in this list:

- Master: The master control state of your project overall.
- Default Driver: Every project needs at least one hardware driver. The default driver was created when you created the project.

As well, each entry in the status list has a small colored circle to the left of it, and a live/not live toggle. We don't need to change anything in this specific menu yet, but let's still go over what each piece means.

Hardware Status Indicators

Most places hardware is shown throughout Bottango, a status indicator is also shown. That status indicator can be either red, yellow, or green.



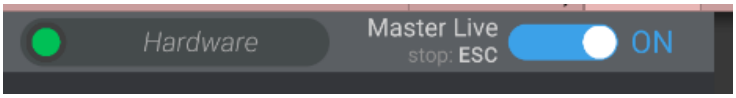
- A red indicator generally means that Bottango was unable to connect to that piece of hardware, or the hardware is set not live.
- A yellow indicator generally means that Bottango could communicate with the hardware, but that something isn't quire right. For example, the wrong version of Bottango is running on the hardware driver, or a stepper motor still needs to be synchronized.

- A green indicator means that everything is good to go, and communication between Bottango and this hardware is working.

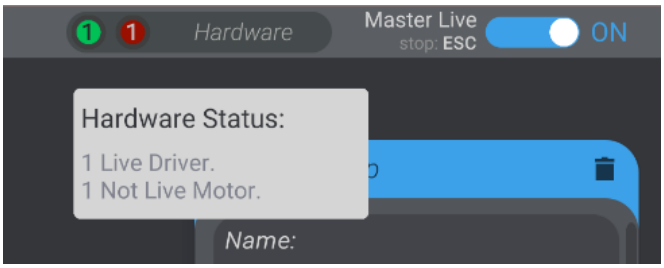
Hardware Status Indicators In the Top Menu

In the top menu, on the hardware mode button, you'll see a collected total status of all hardware in your project. The total number of hardware in each given state (red, yellow, or green) is shown.

If all hardware in the project are the same status, the count is hidden:



As well, you can mouse over the status indicator to see a quick tooltip of hardware status without having to visit the hardware status menu:



Live and Not Live

Every piece of hardware has a live/not live toggle (as does Master as well). When something is live, you have indicated to Bottango that you want if at all possible to be sending commands or moving that piece of hardware. A live hardware driver will execute curves and register motors if it can. A live motor will move to match what is visualized in Bottango if it can.

In Bottango, live indicates if the object in question should be allowed to send and/or receive commands.

You can toggle in the status menu the live status of every piece of hardware in your project. As well, when you configure or select an individual piece of hardware in other modes, you can set it's live status there as well. Changing live in one part of Bottango for a piece of hardware changes it everywhere.

IMPORTANT SAFETY NOTICE: Live and Not Live are not intended to be used for safety purposes. You should assume that as long as your robot has power, your robot could move, and treat it as such. Bugs happen. Unexpected behavior happens. Incorrect configurations happen. Live / Not Live is merely a convenience tool to help you program your robot.

Master Live and Not Live

The top most entry in the hardware status view is master. Here you can control if master is live or not live. When master is not live, all hardware in your project is treated as if it is not live as well.

If at any point you want to try and stop your robot, press the escape key. This will set master live to off wherever you are in the application, and try to send the "stop" signal to all live hardware drivers and motors. Pressing escape again does not turn master live back on. Instead you must click the switch to re-enable.

You can also always view the status of master live in the top right corner of the screen.

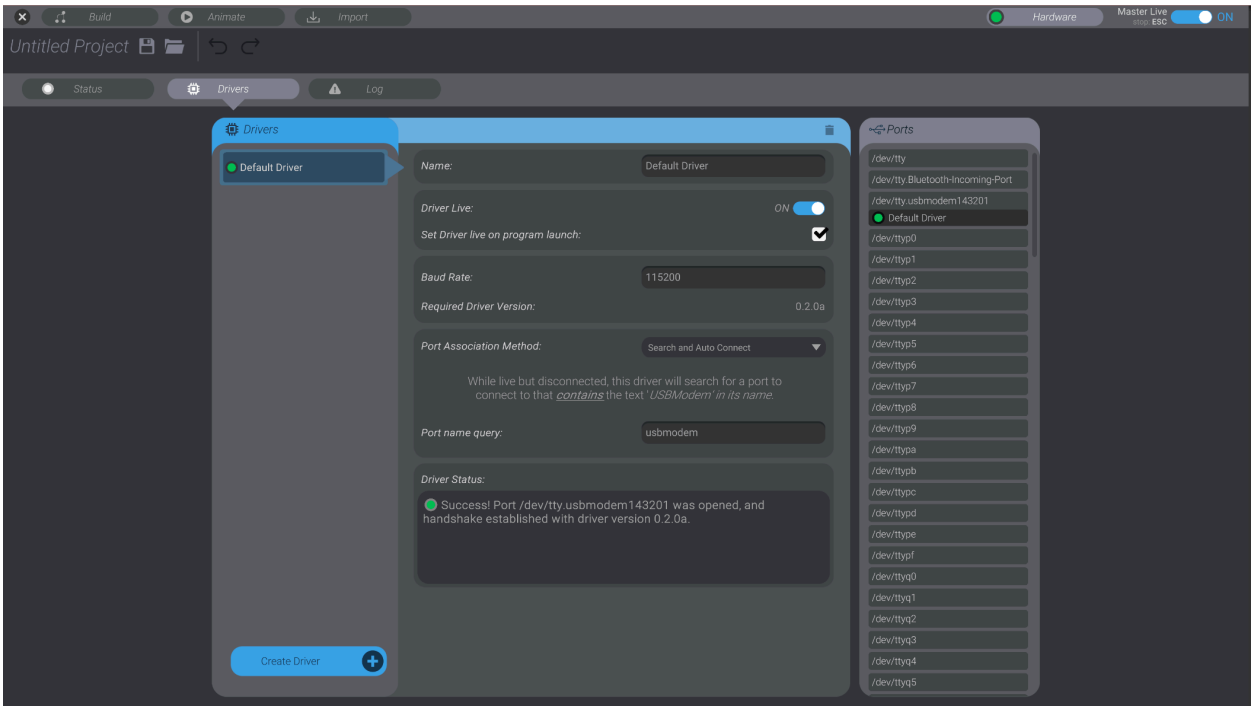
IMPORTANT NOTE: Master live is *NOT* intended to be used as a safety mechanism. You should have hardware stops on your robot. Master live is intended as a convenience tool.

As well, you have a checkbox to set if you want master to be live in your project automatically each time you load the project.

Hardware mode - Drivers

Go ahead and click on "Drivers" to in more detail configure the state of your hardware driver.

The drivers view should look something like this, although the state of your driver and the ports available will be different for every computer.



Driver mode is divided into four sections. On the left is the list of drivers in your project, where you can select and configure them. In the middle is details on the selected driver. On the right is all the ports available to you to try to connect to. Finally, on the top is the master live status of Bottango.

A Bottango project can have as many hardware drivers as you want. You could, if you were so inclined, have a separate driver for every motor in your robot. Or you can run everything through a single driver. This is really going to come down to how your robot works. But there must always be at least one hardware driver in a Bottango project. When a new project is made, a default driver is created as well.

Every effector / motor in Bottango has an associated hardware driver. When Bottango wants to send a command to a motor, it sends it only to the driver associated with that motor. When a new motor is created, it will by default associate itself with the first driver in the list of drivers in your project. Usually, that is the default driver. You can, however, change the driver associated with a motor. More on that in chapter 8 - "Motors."

Configuring a driver

Let's go step by step in configuring a hardware driver. We'll look at the middle section of the screen, where you see the details of the selected driver, and look at each section.

First of all, you can give a hardware driver a name. This has no effect other than helping you remember which driver is which, especially when associating a motor with a hardware driver.

The next section is Driver Live. If a driver is not live, Bottango won't be able to send a command to a motor associated with that driver. If a driver IS live, and then turns back off to not live, not only will the

driver stop sending commands, but Bottango will also try to send a stop signal to that driver in an attempt to stop all in-progress movement. Note that it will only try to send the signal; there's no way to guarantee that the driver receives it.

In addition, you can designate whether the driver will set itself "live" or "not live" when you launch Bottango.

Next you can set the Baud Rate with which Bottango will open the serial connection with your hardware driver. The default value here is the same value in the unmodified Arduino code provided with Bottango. Do not modify this value without also changing it in the Arduino code.

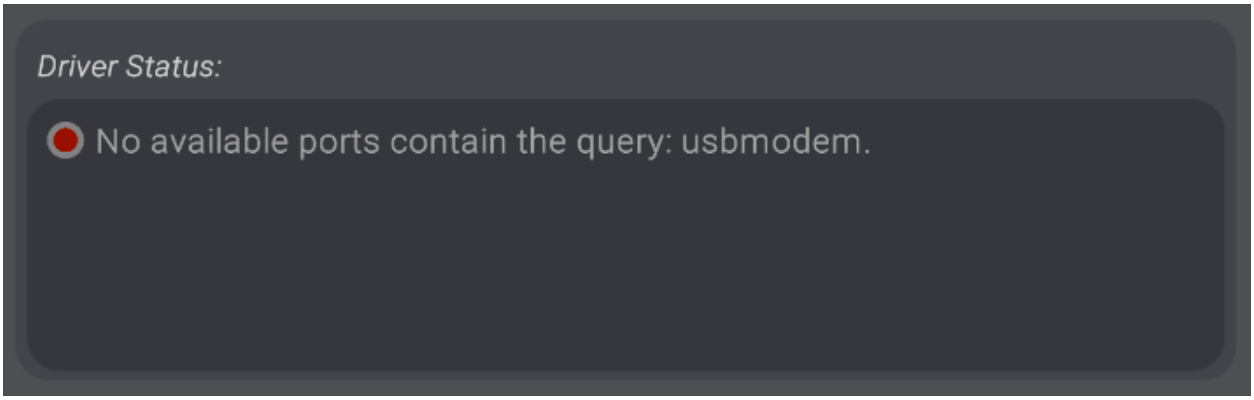
Next you can see the required version number of the driver software that this version of Bottango requires. If Bottango does open a connection with a hardware driver, it will make sure that the hardware driver has the software version number it is expecting, and will alert you if not.

The next section is how you configure which port Bottango will try to associate with the driver. Let's skip the next section for now, and look at the very last one. We'll come back to this section in more detail soon.

The last section is the status box, which tells you in more detail, as best Bottango can, the connection status of this driver.

Driver status Details

In the driver menu, you can see a more detailed explanation of the status of the selected driver.



- A red indicator generally means that Bottango was unable to open a port for this hardware driver. Usually this is because it is not plugged in, or the port is busy. A not live driver is also shown with a red status.
- A yellow indicator generally means that Bottango opened a port for this driver, but wasn't able to start communicating with the hardware driver. Some of the most common reasons for a yellow status are:
  - Connected to a device that isn't running Bottango compatible software.
  - Connected to a device with the wrong version number.
  - A mismatch in serial baud rate.

- A green indicator means that everything is good to go, and communication between Bottango and this hardware driver is working.

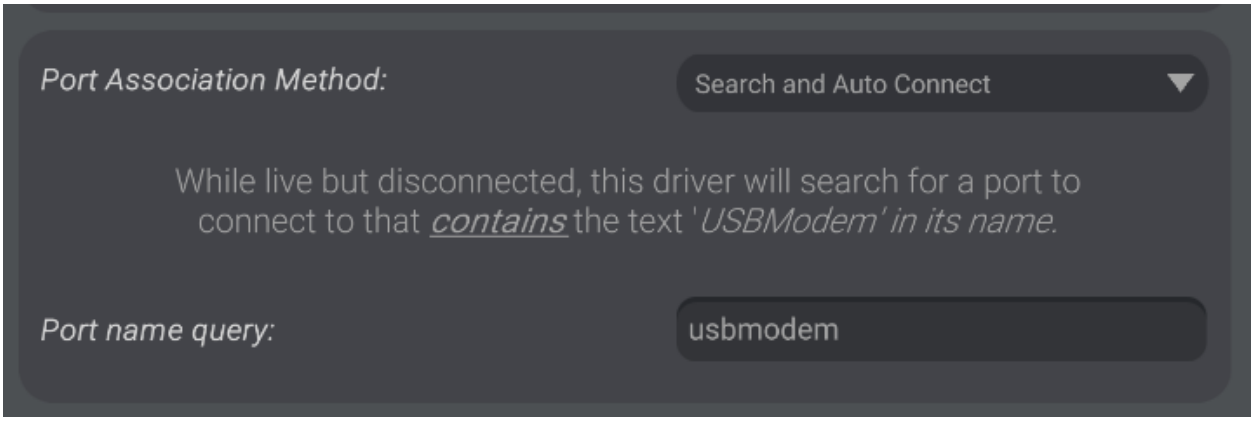
As well, in the driver status section at the bottom you'll see a detailed explanation of the current state of the hardware driver.

Opening a connection to a hardware driver

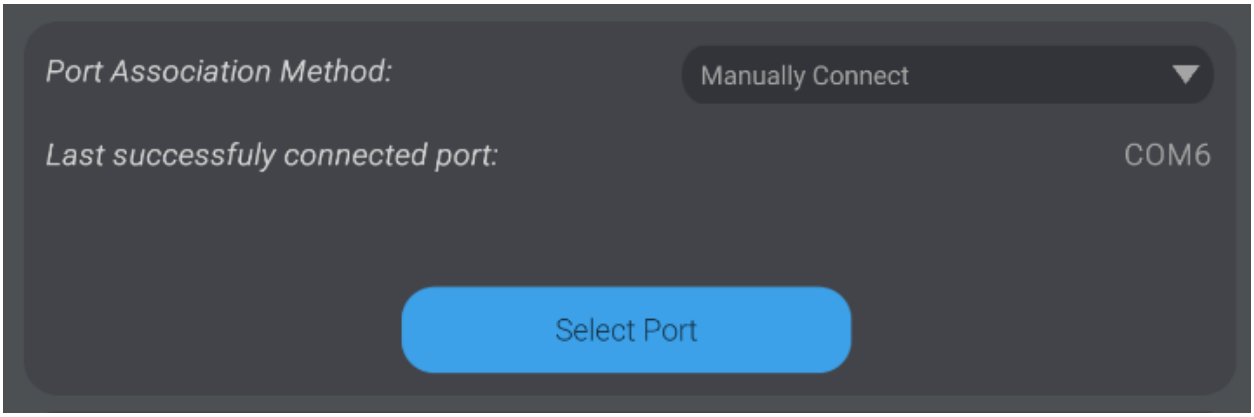
In order to open a connection to a hardware driver, the hardware driver needs to know what port to try to communicate with. There are two ways that a hardware driver can find the port to use:

1. Searching continuously for a port that matches a name you input until a match shows up.
2. Directly connecting to an available port from a list

On a Macintosh computer, an Arduino will show up with a port name that contains "usbmodem." Because of this, Bottango's default driver on a project created on a Macintosh computer searches for a port with the name "usbmodem" and connects to that port as soon as it sees one available. You can see this in the settings of the driver in the port association section.



On a Windows computer, an Arduino will show up on a COM port, but it's hard to know without more information which COM port to use. Here is helpful Arduino documentation (https://www.arduino.cc/en/Guide/ArduinoUno) on how to identify the port of your microcontroller.



However, both port association options are available on both platforms; the only difference is the default. You can switch between the two options by changing the Port Association Method dropdown between "Search and Auto Connect" and "Manually Connect."

Search and auto connect

When a driver is set to connect to a port via Search and Auto Connect, Bottango will continually scan for available ports that match the search query, and attempt to connect to a match if it is found. If for some reason that connection is broken or lost, it will begin the search process again, and automatically reconnect if the port reappears.

As you add more drivers, though, you will need to make your queries more specific. Two Arduinos will both show up in ports with "usbmodem" in their name, so you'll need to make the query more specific to the port you expect to connect to for that driver. And if you're frequently changing the USB port your drivers are plugged into, search and auto connect is probably not the right setting for you.

You can always see on the right side of the screen what ports are available on your computer in that moment. And if any driver is connected to a port, its name will be underneath the port name.

Manually Connect

To manually connect to a port, first make sure the port you want to connect to is listed in the ports panel on the right side of the screen. Again, if you are connecting to an Arduino, it will likely have the word "usbmodem" somewhere in the name if you are on a Macintosh, and "COM" if you are on Windows (use the device manager in Windows to figure out exactly which port you want.)

Click the Select Port button, and then click on the port you want to try to connect to. Bottango will then attempt to connect to the port in question, and give you the status along the way in the Driver Status section.

When you have successfully connected to a port, the name of the last successful port connection for this specific driver is saved. This can be helpful to remember which port you want to use in future sessions.

In addition, you have the option of whether or not you want Bottango to automatically reconnect to that port (if it is available), if/when an error occurs. An example would be if a corrupt message is sent over serial, and the hardware driver is not able to respond correctly. You can choose whether you would rather Bottango attempt to restart the connection automatically, or remain disconnected until you manually reestablish the connection.

Creating and deleting hardware drivers

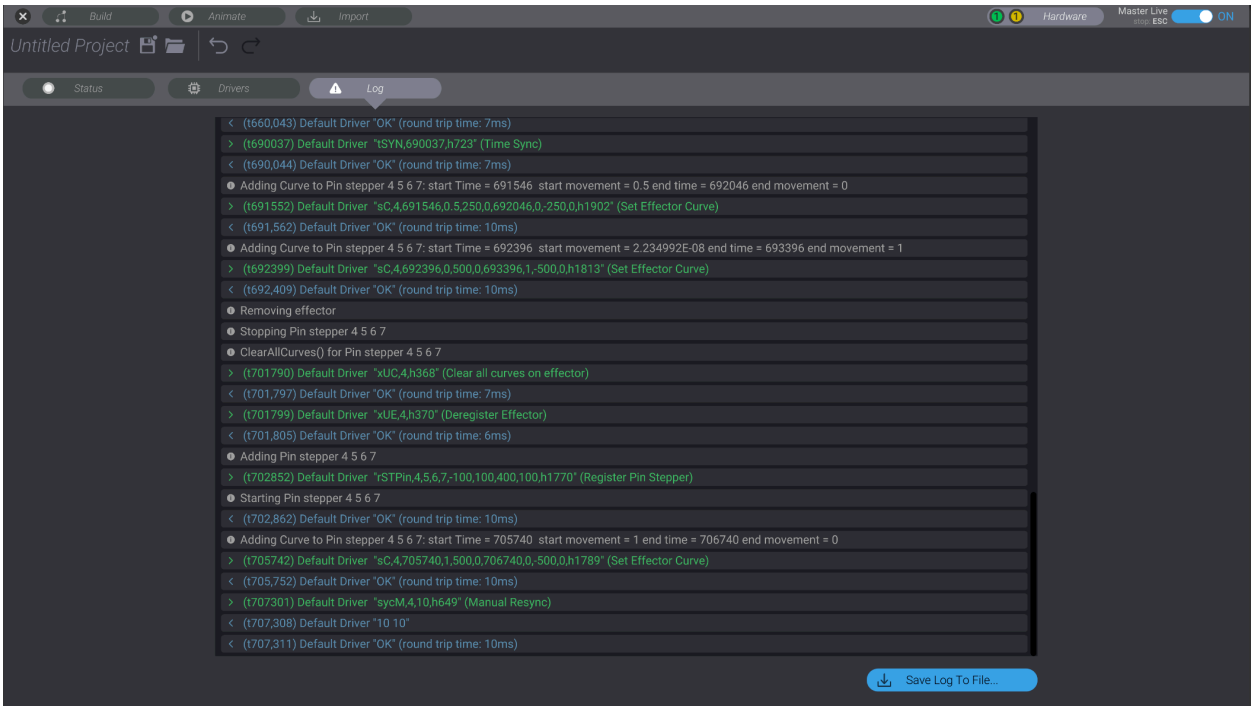
Although you always must have at least one hardware driver, you can add as many more as you want. Click the Create Driver button on the left side of the screen to create a new driver. This will automatically select the driver as well. Click the driver names in the drivers list on the left side of the screen to change which driver you have selected to configure.

The biggest limitation in having multiple drivers is making sure you don't have more than one driver that could try and connect to the same port via search and auto connect. Bottango will warn you if that's the case in the Driver Status window, but you'll need to come up with more unique search queries for each driver if you don't want to manually connect.

You can also delete a selected driver by pressing delete or clicking the trash can icon. When you delete a driver, all motors that are associated with that driver will change their associated driver to the default driver (i.e. the first driver in the drivers list).

Driver Log

While animating and configuring your robot, Bottango sends numerous requests back and forth to the hardware driver. For testing and support purposes, or just to learn more about the inner workings of Bottango, you can view a log of the sent and received commands. Press the log button above the driver details to switch to the log view:



The in-app log is limited to the most recent 250 messages. However you can export the most recent 10,000 messages to a text file by clicking "Save to file…"

Wrapping up

Hardware drivers are essential to the workflow of Bottango. From here on out, the design intent of Bottango is to make your real world robot express a state as close as possible to your virtual one. By setting up your drivers, you've made it so that as you create joints and animate your robot, you'll be able to see your changes in real time, and in the real world!

-8-
Motors Basics and Servos

What's in this chapter

If Bottango was only a 3D modeling program, it would be a pretty lackluster one. Motors are what make your robot more than just a statue. Every individual motor on your robot that you want to control in Bottango needs a virtual equivalent.

In this chapter, we'll cover the process of adding and configuring motors.

Currently Bottango supports two kinds of motors: servo motors and stepper motors. Servo motors are the simplest to configure and control. **In this chapter, we will exclusively use servo motors to teach the basics of motor control in Bottango. In later chapters we'll discuss how to control steppers and other motors.**

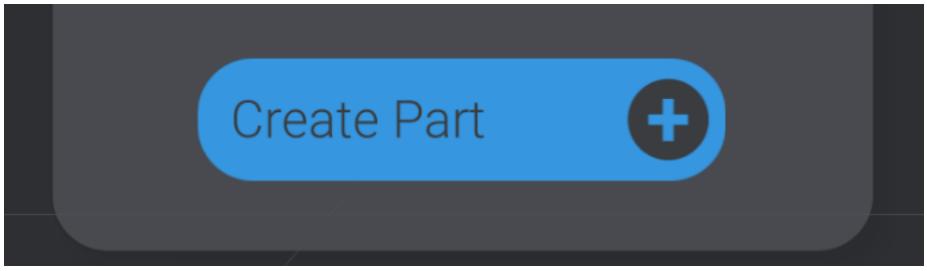
Tutorial vs your robot

In this chapter, we'll continue using the project started in the chapter six: "Creating structures and Joints." However, you almost certainly don't have the same robot in real life, with the same motors. From here on out, it's up to you if you want to follow along with the tutorial virtually only, without a real hardware connection, or if you want to start a new project, model the structure of your robot, and transpose the lessons of the rest of this documentation to your specific needs.

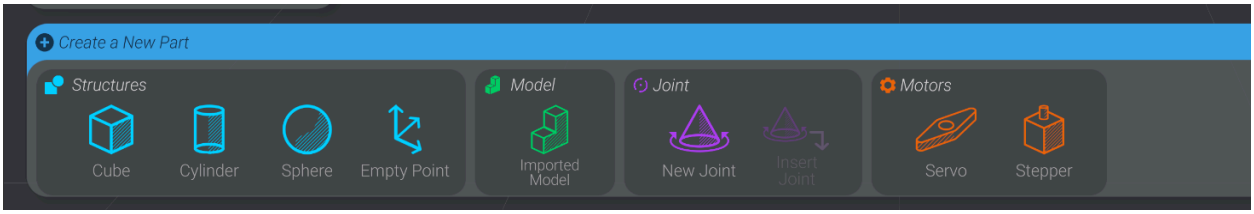
If you do begin working on your own robot from here on out, I really recommend you have your hardware driver(s) configured, connected, and wired up to your motors. One of the core design principles of Bottango is to closely match the real world configuration of your robot to the virtual one in Bottango. Only by having a live hardware driver wired up to your motors is that possible.

Create your first motor

1 Select BaseStructure, and click Create Part



This will cause the create window to show up:

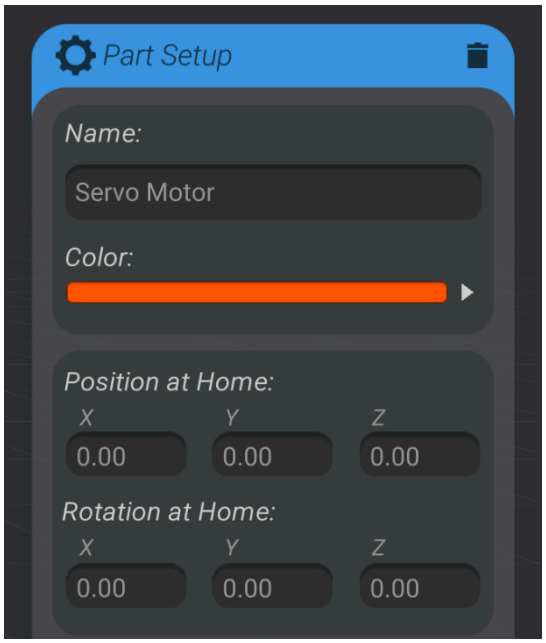


2 Click servo to create a new servo.

The location and hierarchy placement of a motor doesn't make any practical difference. It's just helpful for reminding you of which real motor is represented by a virtual one.

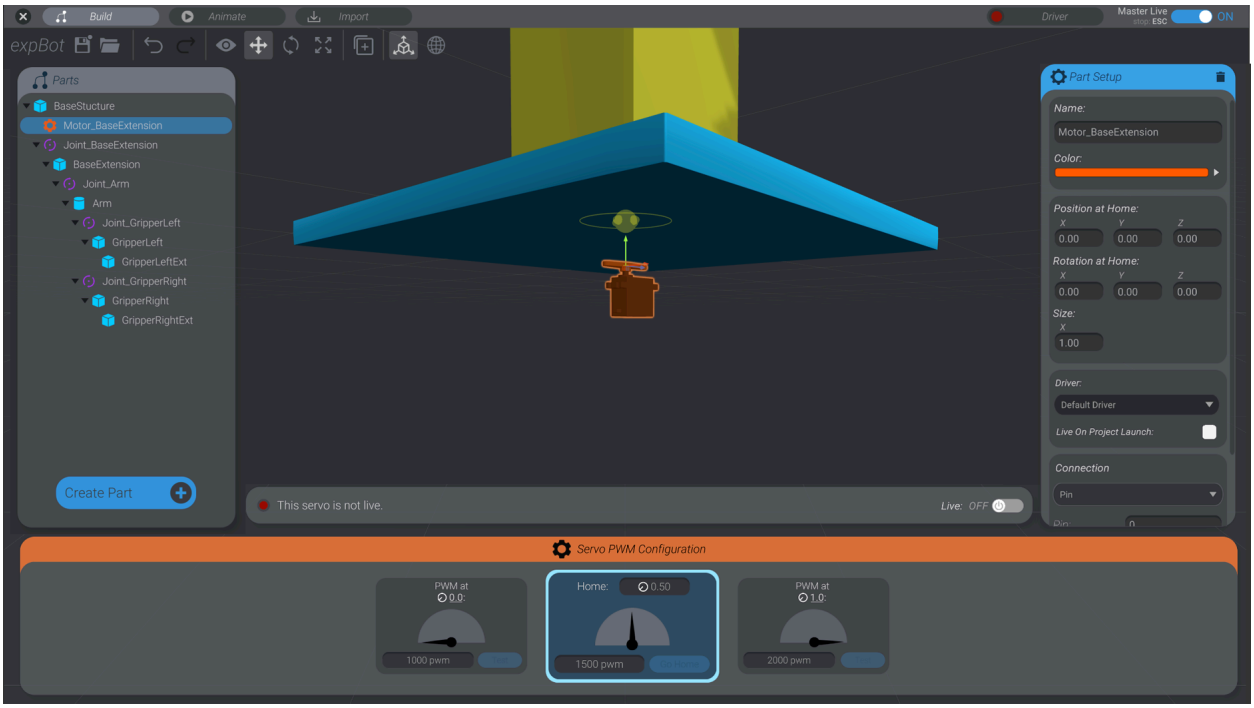
Motor setup

When you select a motor, the first few sections of the Part Setup window should look familiar to what you saw when creating structures.



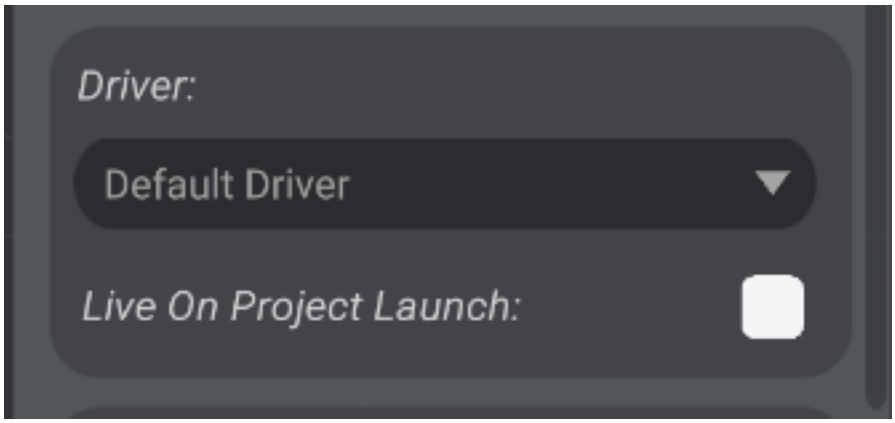
Motors have names, colors, and home position and rotations just like structures. However, motors do NOT have size.

Here, for example, the first motor we have created is named "Motor_BaseExtension" and is placed approximately where the real motor would be on the robot to rotate the equivalent real-world part.



Motor Driver and Live On Launch

Every motor needs an associated driver. The driver you associate with a motor is the driver that will receive all commands from Bottango when the motor is moved.



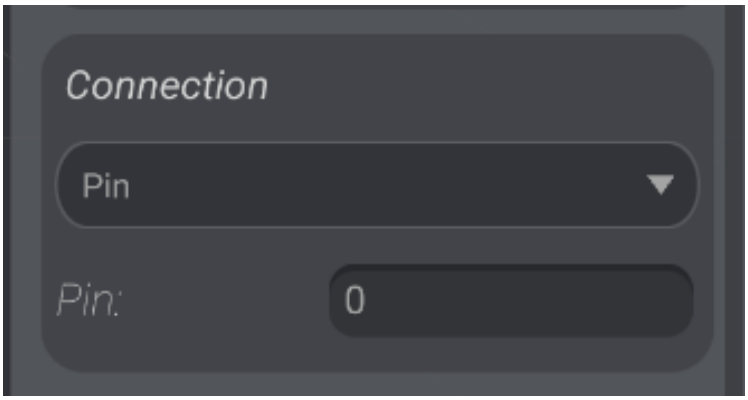
All created drivers in the hardware driver mode tab will be listed as options in the driver dropdown menu on the Motor Configuration panel.

If you associate a motor with a driver, then later delete that driver, the motor will associate with the default driver as the replacement.

Additionally, just like a driver, a motor can be live or not live on program launch.

Servo connection

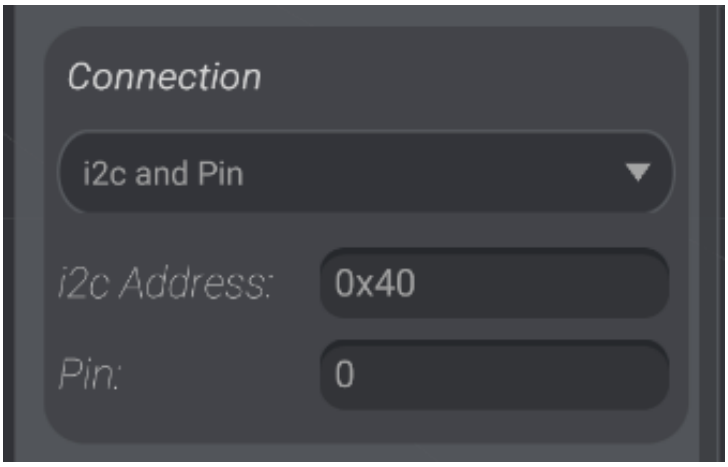
Your hardware driver has a particular way it is connected to your motor, and Bottango needs to know it in order to successfully communicate with your hardware driver. As a basic example, a servo is driven on a hardware driver via a PWM pin output on the hardware driver.



Here you can see this servo has been configured on the driver to be driven via a pin, and the pin is set to 5.

Only one motor can have the same connection configuration. For example, if two motors are both associated with the same driver, and have the same pin, one of them will be set to not live, and cannot be reenabled until the conflict is resolved. You will see this if you create two motors in a row without changing their connection and driver configuration, as both motors will have the default driver and default connection.

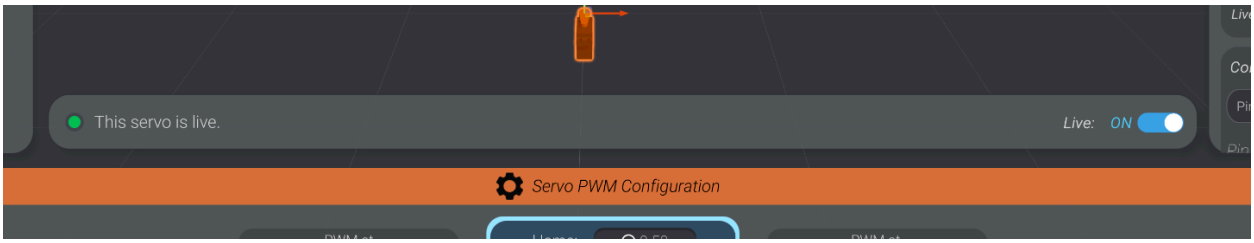
You can set a servo to connect via a Pin, or using the drop down, change to be driven over i2c using the Adafruit 16 channel PWM driver.



Please see the included ReadMe.txt with the Arduino driver code to take the steps required to enable support for various libraries (such as the the library for the Adafruit 16 channel PWM driver.

You will need to enter the i2c address in hexadecimal or integer format, as well as the pin for the servo.

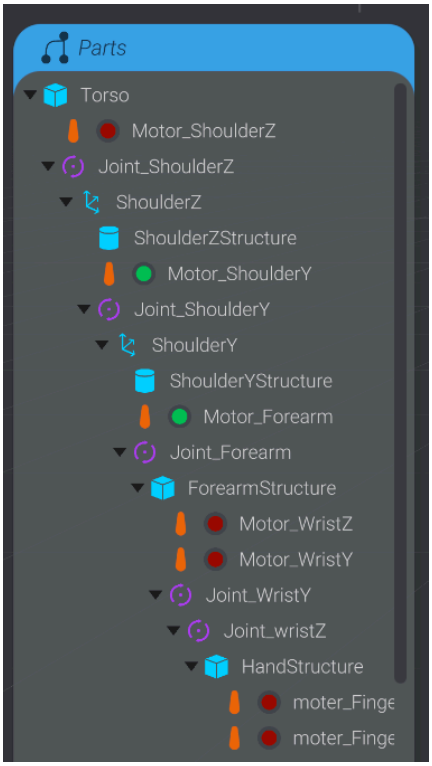
Servo Live and Status



Near the bottom of the screen, you'll see an indication of the status of the motor, and whether the servo is live or not live. This colored circle can be red, yellow, or green, just like described in the hardware status view. As well you'll see a text description of the exact status of the motor for more details.

You can change the live state of the motor here, just like you can in the hardware status view.

Finally, you can see the same status indicator next to each motor in the parts list:



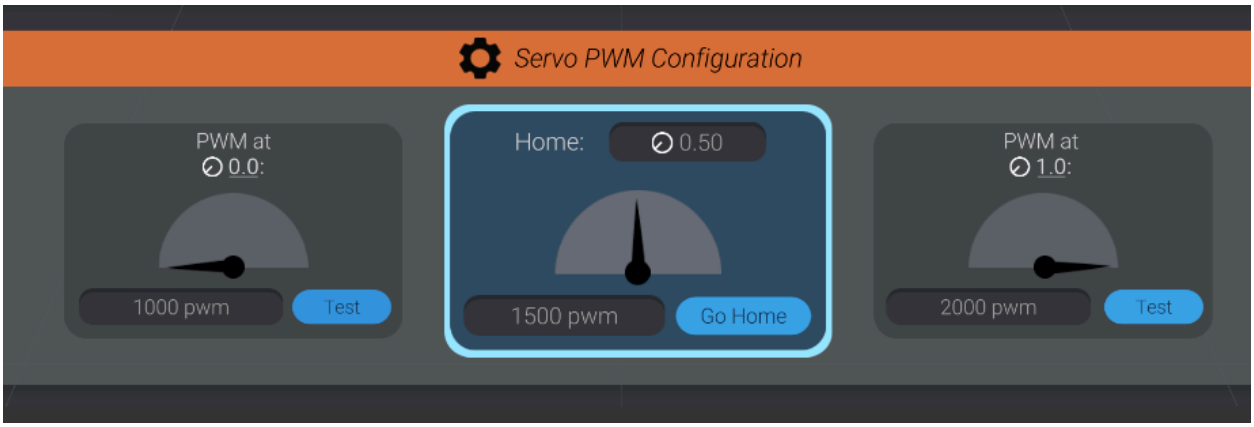
A newly created motor is not live by default. You must turn the motor to live once you have finished configuring it to begin driving that motor.

Servo PWM configuration

You set a servo to its desired rotation by having the hardware driver send a PWM signal in a range between the minimum and maximum PWM. Most every servo has a different hardware-bound range of acceptable PWM values, and in your robot, usually there's a subset of that range that you want the robot to actually use.

When you are configuring your motor, you should not use the theoretical maximum range that the servo could accept as the PWM values. Instead use the minimum and maximum PWM values you want the robot to actually use when animating it.

Shown here are the default PWM configuration values for a servo.



This servo will express 1000 PWM at its furthest in one rotation, and 2000 at its furthest in the other.

In the cell for Minimum, Home, and Maximum PWM, there is a test button. Press that button to send a signal to the servo to express that PWM. This can be useful if you're not sure what values you wan to use and you need to experiment a bit by setting your real world motor to the entered positions.

Our first encounter with movement: The most important concept

Throughout Bottango you will see this icon:

This icon is so important, you can see it's even in the logo for bottango:

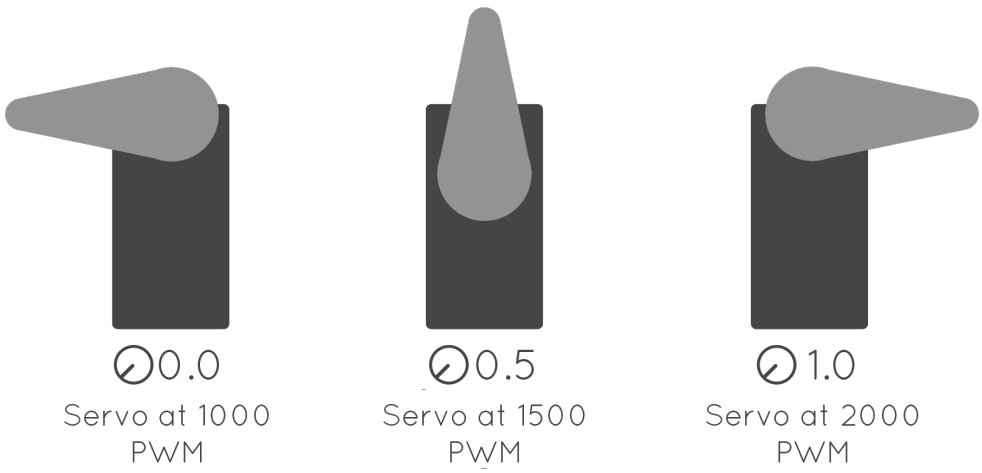
This icon represents what I call "movement," and it stands for the the value of an object between its minimum value and maximum value, on a 0.0 to 1.0 scale. A more "math-y" term for movement would be a "normalized value."

> In Bottango, movement is the value of an object between its minimum value and maximum value, on a 0 to 1 scale.
>
> Movement is used in motors, joints, and animations.

Why is movement important? Movement is the common denominator that allows you to manipulate not just different kinds of motors, but also different concepts. Additionally, movement allows you to change your robot at will, and not worry about keeping the rest of your project in sync.

We will cover movement a lot more when we talk about joints and animations. In this context, however, know that you are setting what PWM value the motor should be at when at minimum movement (0.0) and what PWM value the motor should be at when at maximum movement (1.0).

Here you can see how the currently configured PWM values of this motor map to what you would see on a real life servo, and the associated movement values:



Using movement rather than direct values allows us to have a single unifying number to keep multiple concepts in sync. For example, when we create a joint, we'll be defining the minimum and maximum

position or rotation offsets from home for that joint. Those minimums and maximum offsets will map directly the the minimum and maximum PWM values defined here on the motor. This allows us to make changes to one part of the project, and since we're using movement (normalized values), the rest of the project will stay in sync.

It's okay if movement doesn't make total sense yet. By the time you've created joints and animations, it'll be a lot more clear.

Home PWM

The middle PWM value to configure on the servo is the home PWM value. This is the value that the servo will be in "at rest." If you think back to when we created structure, we did so in its home position and rotation. When Bottango moves and rotates joints, it offsets the joint away from its home position and rotation, and then returns back to that home position and rotation when at rest. Bottango will do the same thing with motors, move them to the desired PWM value, and then return them to rest at their home PWM value.

You can see that in this example, the home PWM value is set to 1500. Because that is exactly halfway between the minimum value of 1000 and the maximum value of 2000, it results in a home movement value of 0.5 (halfway between 0.0 and 1.0). If you prefer to think of your configuration via movement, you could also type in the the movement value, and let Bottango calculate the actual PWM value. For example, you could set the home movement value to 1.0, and your home PWM will be whatever your maximum PWM value is.

Once a motor and a joint are linked, ideally they both have the same home movement, so that sending your robot home sends both the motor and the joint to the same home. If they are not the same, Bottango provides tools to help deal with that, which we'll talk about more in the next chapter on configuring joints.

Max speed and Jog Speed

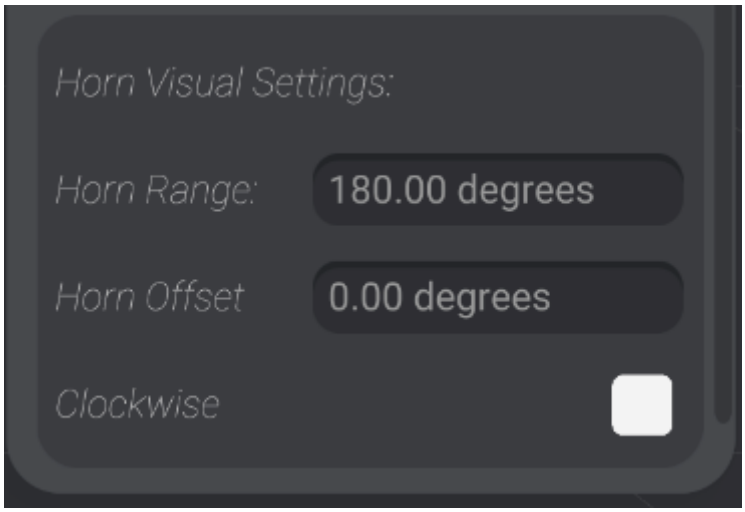
When you're moving a motor between positions, you often don't want it to go as fast as mechanically possible to its new destination. Max speed limits the speed with which a motor will transition from one value to another so as to not damage your robot.

When you are animating your robot, you'll see warnings as well when an animation moves faster than the given max speed. We'll talk about that more in chapter 13 - animation basics.

As well, you'll see a field to set the "jog speed" of a motor. This is the speed that Bottango moves the motor and associated joints when moved for any reason other than playing an animation. Jog speed is a percentage of the max speed of the motor. As an example, a servo defaults to 50% jog speed, which means if the maximum speed is 3000 PWM/sec, the jog speed will be 1500 PWM/sec.

Motor Horn Visual Settings

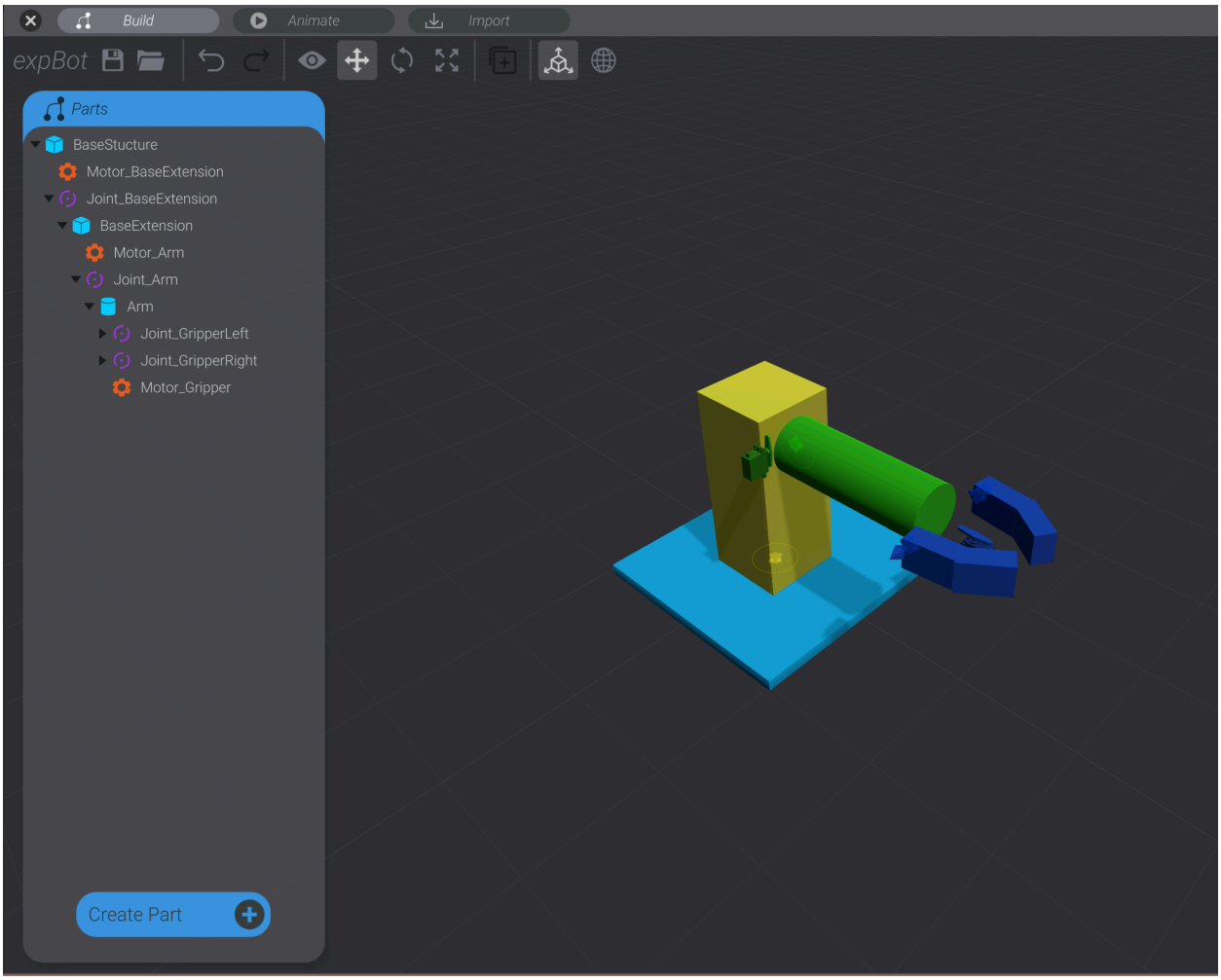
As your motor moves, as a helpful visual guide the horn moves as well. For each kind of motor, there is a default range of rotation, however, you can adjust that rotation however you'd like using the horn visual settings:



Note however that these settings only affect the 3d horn visual guide on your motor, they do not affect how your robot moves. To control how your robot moves, we'll cover that in the next chapter on Joints.

Creating all motors for the example robot

Here you can see I've created all three motors for the example robot: one to move the baseExtension, one to move the arm, and one to move the gripper:



I have also moved, rotated, and parented them in a way that made sense for my robot's real-life configuration. For example, the grabber motor is a child of the arm structure, just like the grabber structure, so that it all moves together when the arm moves.

Since you're not building this literal robot, there's no use in showing the process of configuring the driver, connection, and PWM values of these additional motors, as your robot will have a very different specific setup.

As well, remember to set your motors to live once you have finished configuring them, if you want them to move in real life.

-9-
Configuring Joints

What's in this chapter

In this chapter, we'll take the joints we created in chapter 6: "Creating structure and joints," and configure them to move correctly.

As a reminder, when we created the structure of our robot, we also created the structure pieces as children of joints at which the robot would move or rotate.

In Bottango, Joints are the points of articulation that can move or rotate on a single axis.

In order to make a functioning joint, you want to configure the following:

- **Configure the axis that the joint moves or rotates on.** This defines if the joint moves or rotates in X, Y, or Z dimension.
- **Associate the joint with a motor.** When the joint moves or rotates, commands will be sent to that motor.
- **Set the joint minimum offset.** When the joint is at its minimum (0.0) movement, how much should it move or rotate away from home?
- **Set the joint maximum offset.** When the joint is at its maximum (1.0) movement, how much should it move or rotate away from home in the other direction?
- **Resolve any conflict if the motor and joint have different home movement.** If your joint has a home at 0.25, and your motor has a home at 0.75, which do you use when sending the robot to home? You'll have an opportunity to either resolve the conflict to make the joint and motor have a matching home, or choose which home value you want to use.

We should have set the axis for the joints in our robot in Chapter 6: "Creating structure and joints". If you need a refresher on joint axes, refer to that chapter, in the section "Joint Axis."

What remains is to associate each joint with a motor, and set its minimum and maximum offsets, and resolve any conflicting homes if they exist.

Why link to a motor first?

In Bottango you can set up a joint without first connecting it to a motor. In some cases, that's actually the right thing to do (see the chapters on Target Poses and Pose Blends)! But for most cases, you'll find things easiest if you link a joint to a motor first.

Why is that? The point of setting up the motor is to configure it to move identically to the real world robot. By setting up hardware drivers and motors ahead of time, you can move your robot in real life, then match your virtual joints to look the same. Without doing that, you'll be left to guess or make assumptions on how your joints move without being able to look at the real robot to confirm what you got right and what needs tweaking.

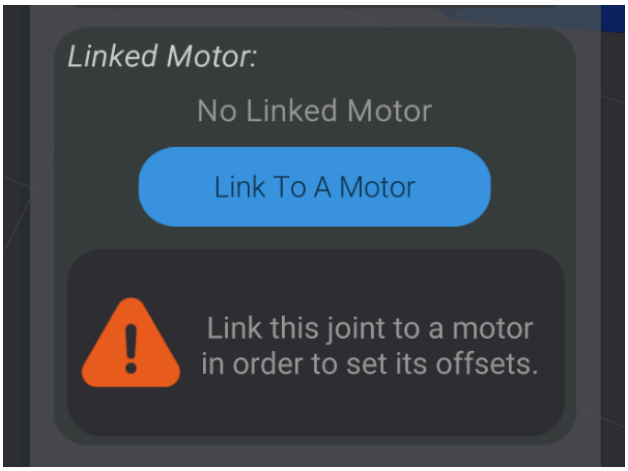
For example, while editing the minimum movement (↙ 0.0) offsets of a joint, Bottango will move that joint's linked motor to its current minimum movement hardware signal value.

This allows you to look at your robot in real life and move its virtual representation to as close an approximation as you can. This is very helpful for making sure the movement of your physical and virtual robot are in close approximation.

Link a joint with a motor

1 Select the joint that is the parent of Base Extension.

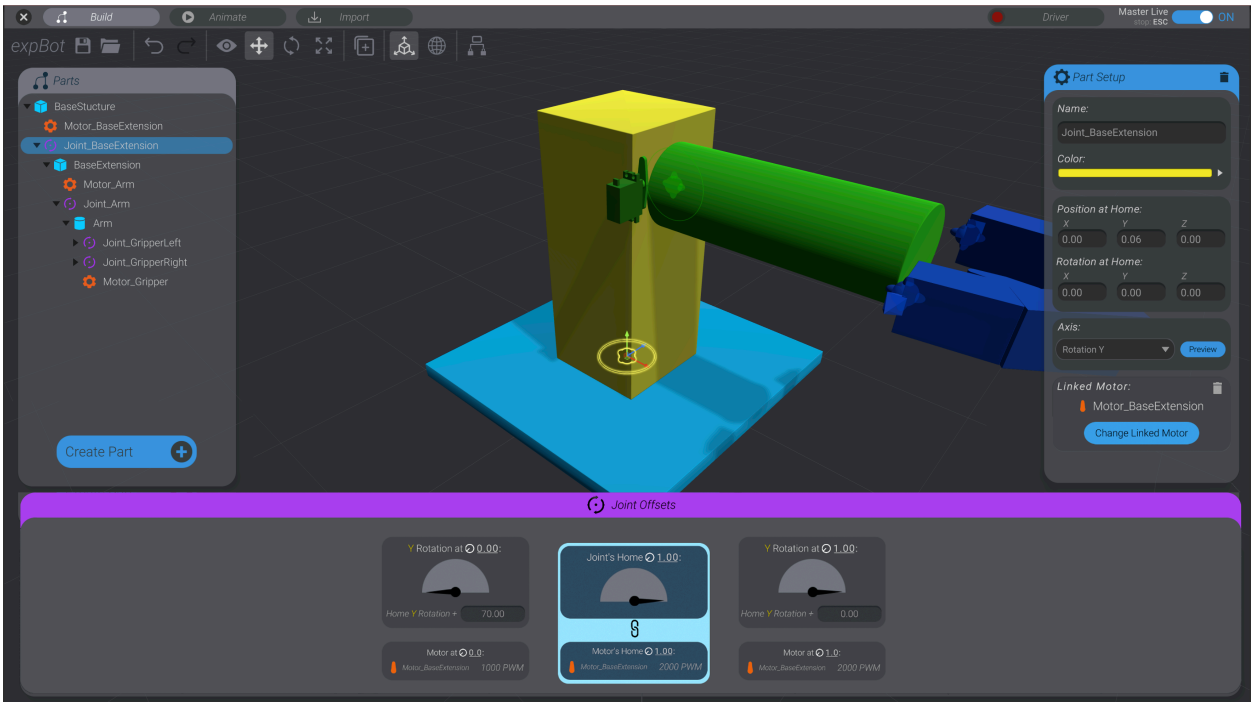
Notice how the joint says that there is no linked motor:



2 Click "Link To A Motor." All valid motors will begin to glow, allowing you to select the motor you want to use to drive this joint.

3 Select the Motor_BaseExtension. You can click either the 3D model or the motor in the parts list.

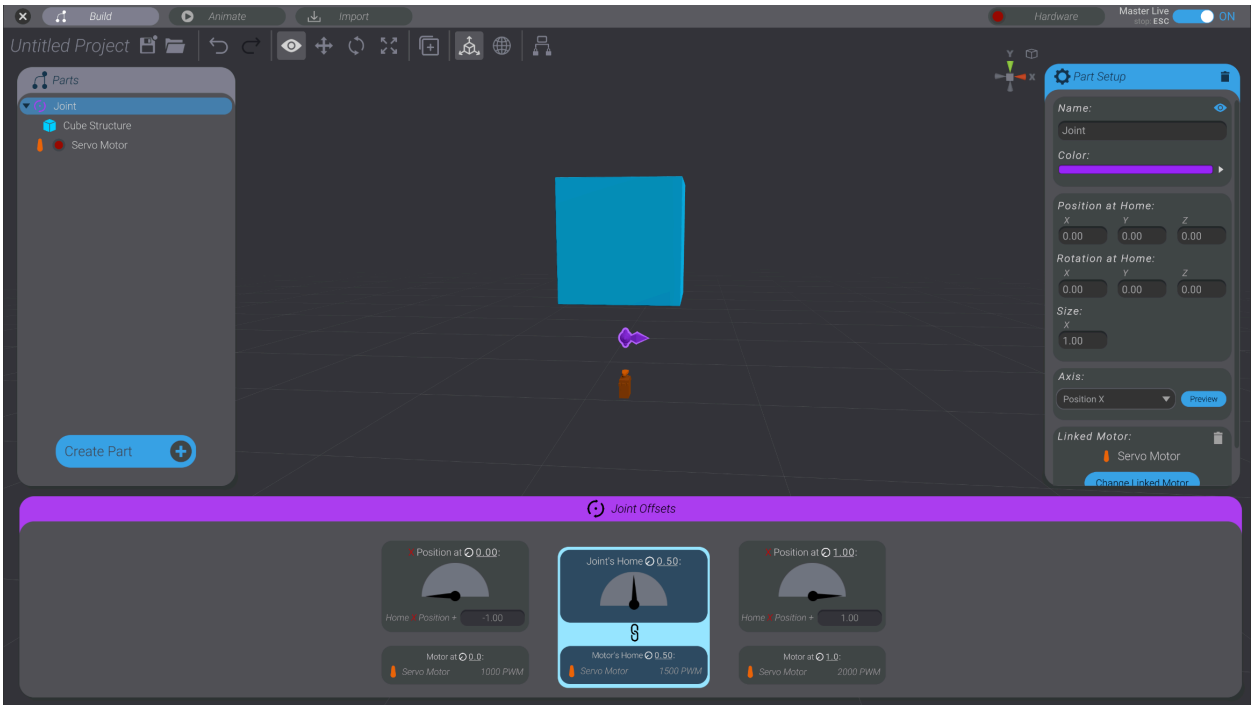
We have now linked this joint to Motor_BaseExtension, and can configure it further.



Configuring a joint's offsets

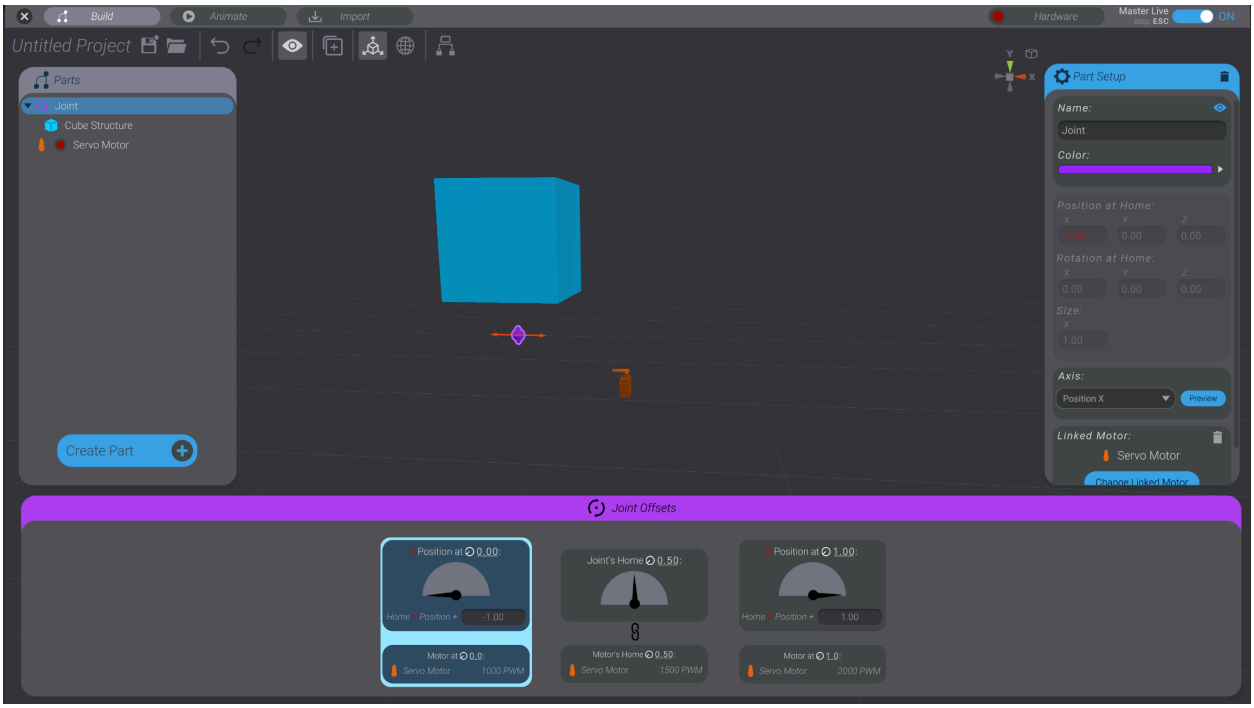
A joint operates by moving or rotating a structure away from its home position or rotation. When a joint is selected, you can configure those offsets.

Let's start with a simple example. Here I have created a project with a motor, a cube structure, and a joint that moves back and forth in the X axis:



Here you can see that the cube is a child of the joint. As the joint moves in the X axis, so too will the cube. The joint has a "home position" of 0.0 in the X dimension.

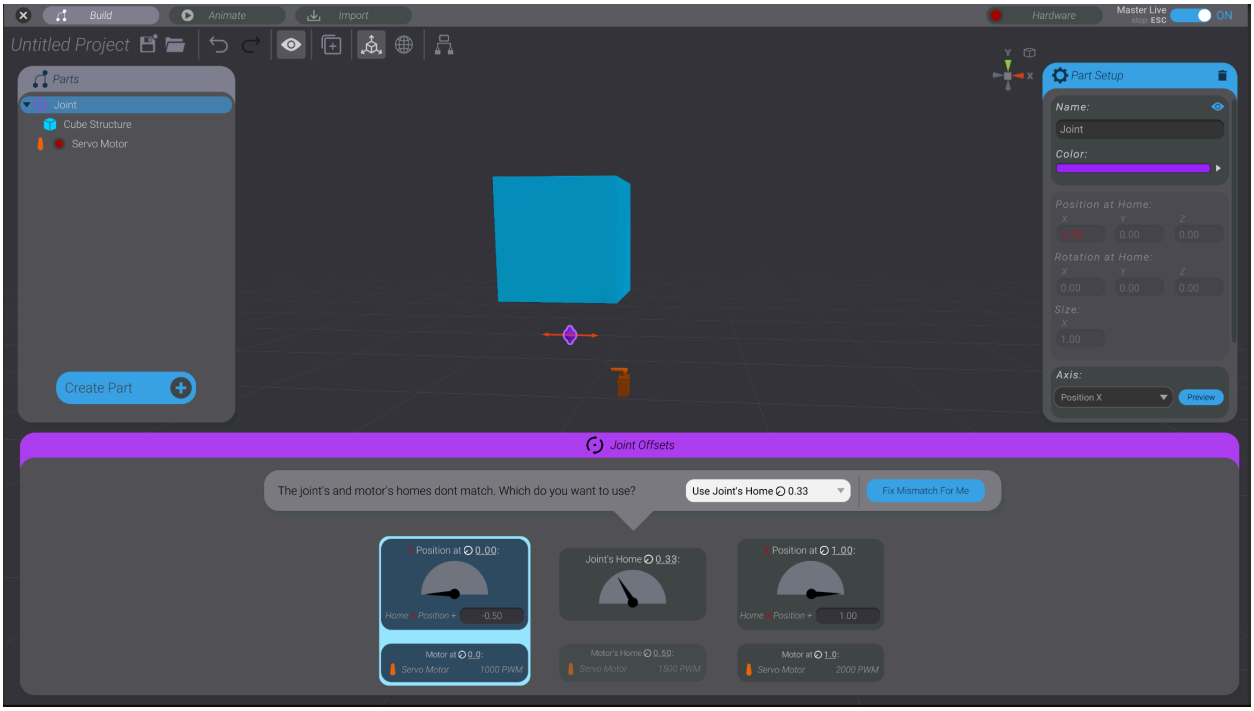
Now when I click on the left offset cell (offset at minimum movement 0.0), I will move the joint to its minimum position:



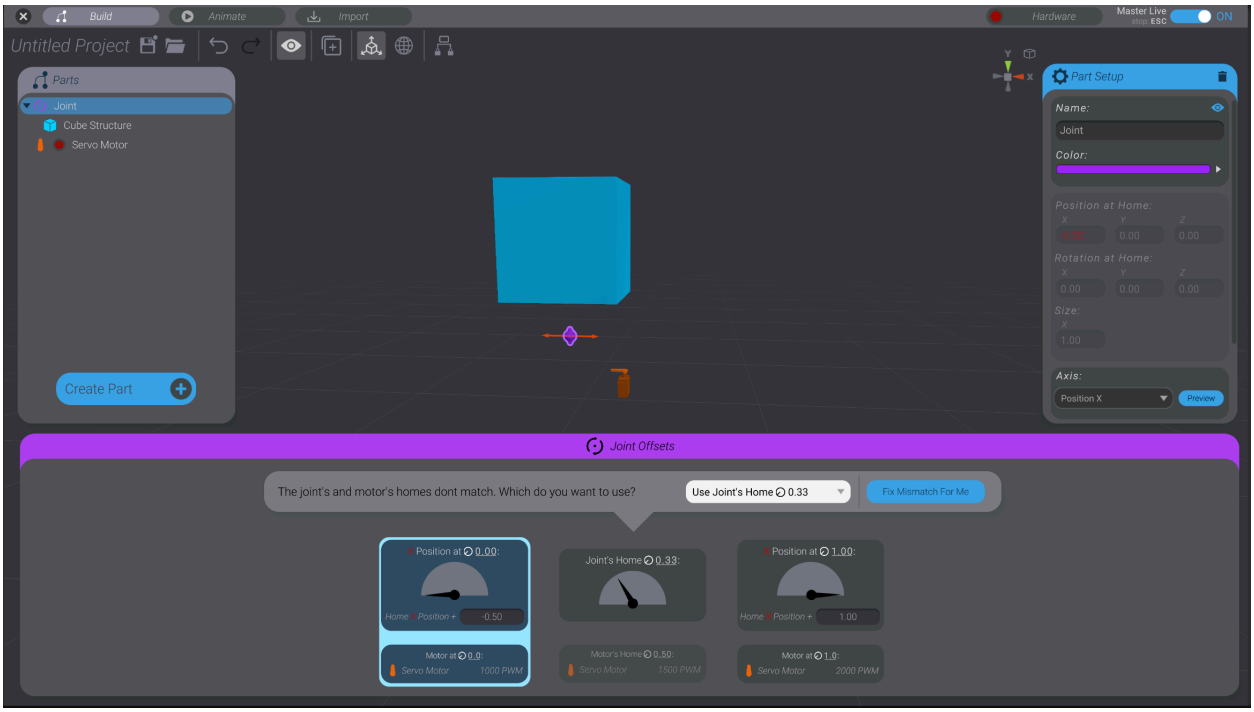
The joint moved to a position -1.0 in the X axis. You can see this by the movement of the joint, and also by seeing how its position field now says "-1.0." You'll also notice the X position field is red. Any time a joint is offset from its home position or rotation, the offset axes are shown in red.

Why -1.0? Well, home is 0.0 in the X, and the offset at minimum value is set to -1.0. So that means that when the joint is at minimum, it will move -1.0 from its home value (0.0) in the axis set (X position).

If I change the offset value to -0.5 instead, the joint moves -0.5 from home.



I can also click to select the offset at maximum movement (1.0), and configure and visualize that value as well:



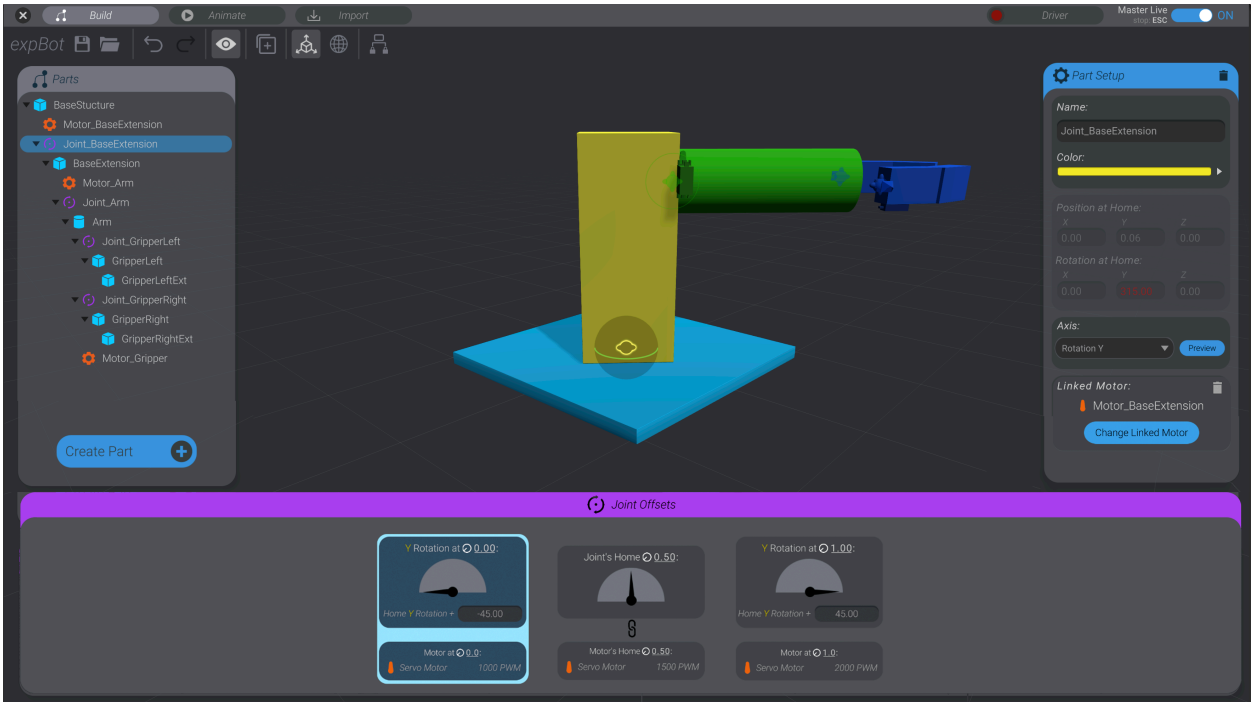
You'll notice as you change the minimum and maximum offsets, that you'll sometimes get a warning that the joint's home value and the motor's home value no longer match. This is because as you're changing

your joint's offsets, you're also changing what home is for that joint. We'll talk soon about what to do about that. For now, it's ok to ignore that warning.

Let's do it for real!

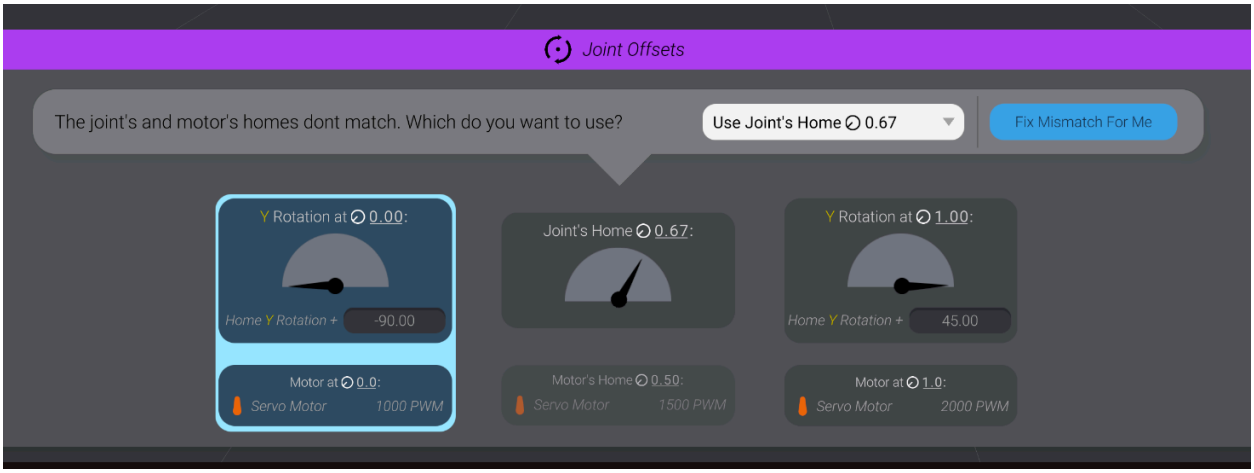
1. Select the joint that is the parent of Base Extension if it is not already selected.

2. Click "Offset at Minimum" to set the minimum offset for this joint.



The joint currently has the default value of -45 degrees.

3. Change the value to -90 degrees.



4 Select the maximum offset, and change that value from +45 degrees to +90 degrees.

We've set the joint to sweep 180 degrees total (from -90 degrees to +90 degrees), with home in the center (0.5).

5 Use the same technique to link a motor to the arm joint, and configure its offsets as you see fit.

Resolving a Joint and Motor Home Conflict

Once you've set up a joint to have the right offsets, it possibly doesn't have the same home movement anymore as the motor linked to it.

Let's look at an example:

	◴ 0.0	Home	◴ 1.0
Joint	-50 degrees	0 degrees ◴ **0.25**	+150 degrees
Motor	1000 pwm	1500 pwm ◴ **0.50**	2000 pwm

In this example, the home movement of the joint is ◴ 0.25, based on where you set up the offsets at minimum and maximum movement. The home movement of the servo is ◴ 0.5 based on how you set the motor up in the previous chapter.

When we're sending the robot to home, which home do you use? The joint's or the motors?

When the joints and motors home don't match you can select which home to use:

In the above example, we've set to use the Joint's home value ◴ 0.25 instead of the motor's ◴ 0.5.

As well, you can press the "Fix Mismatch For Me" button. Bottango will make the best guess it can on what changes to make, and will modify your joint offsets and/or motor home value to make them have a matching home.

Here you can see, after pressing the "Fix Mismatch For Me" button, the motor has had it's home value changed from 1500 pwm to 1250 pwm automatically:

	0.0	Home	1.0
Joint	-50 degrees	0 degrees **0.25**	+150 degrees
Motor	1000 pwm	*1500 -> 1250 pwm* **0.25**	2000 pwm

If this is a little confusing, that's ok. It's ok to leave the motor and joint with a mismatched home and just ignore this warning. You can also click the "fix it for me" button to see if you're happy with the automatic changes, and undo the changes if you're not.

Multiple joints on a motor

There are two joints that make up the articulation of the gripper, but only one motor. As the gripper motor moves, each joint moves to open and close the gripper by moving in opposite directions.

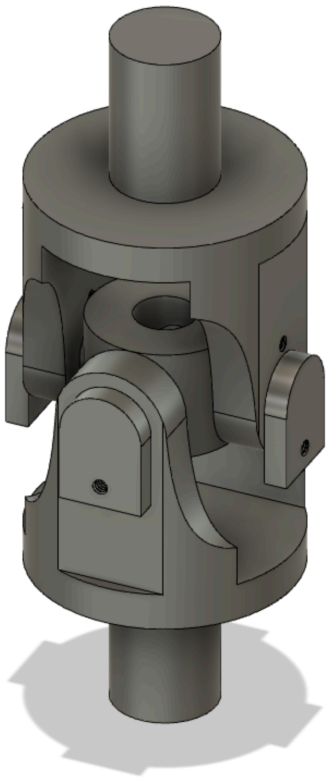
1. Link the left gripper to the gripper motor, and configure the left gripper's joint offsets to open and close.

2. Link the right gripper to the same motor, then configure the opposite offsets. If the left gripper is -1 at its minimum, for example, the right gripper will be +1 at its minimum offsets.

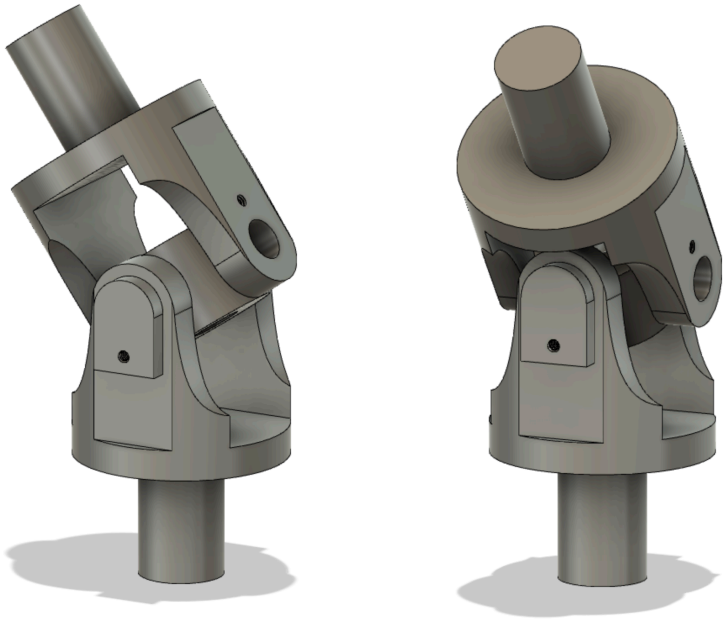
A motor can drive an unlimited number of joints; however, a joint can only link to one motor. If a joint linked to two motors, there would be no way to know how to resolve conflicting directions, and physically that would likely pull the joint apart.

What if a real world joint can move or rotate in multiple directions?

Let's take, for example, a universal joint:



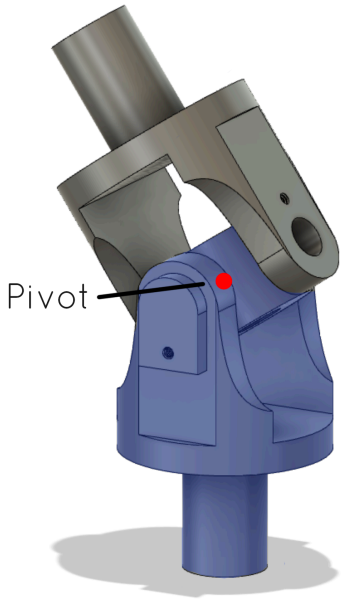
A universal joint can rotate in 2 axes. In the above example, the joint can rotate in the X and Z axes, and is static in the Y axis.



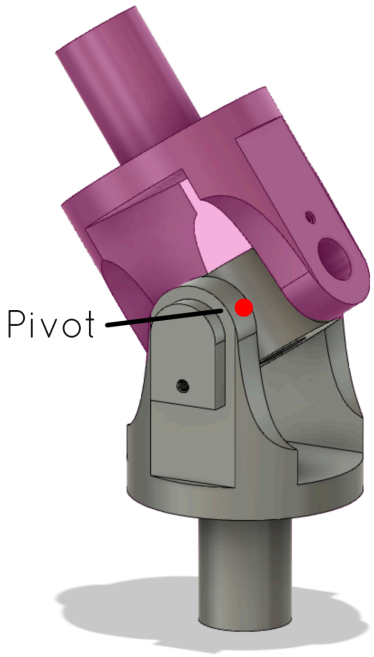
In the above illustration, you can see the joint rotated first in the X, and then in the Z axis.

So you might argue that a Bottango joint should also be able to rotate in two axes, not just one.

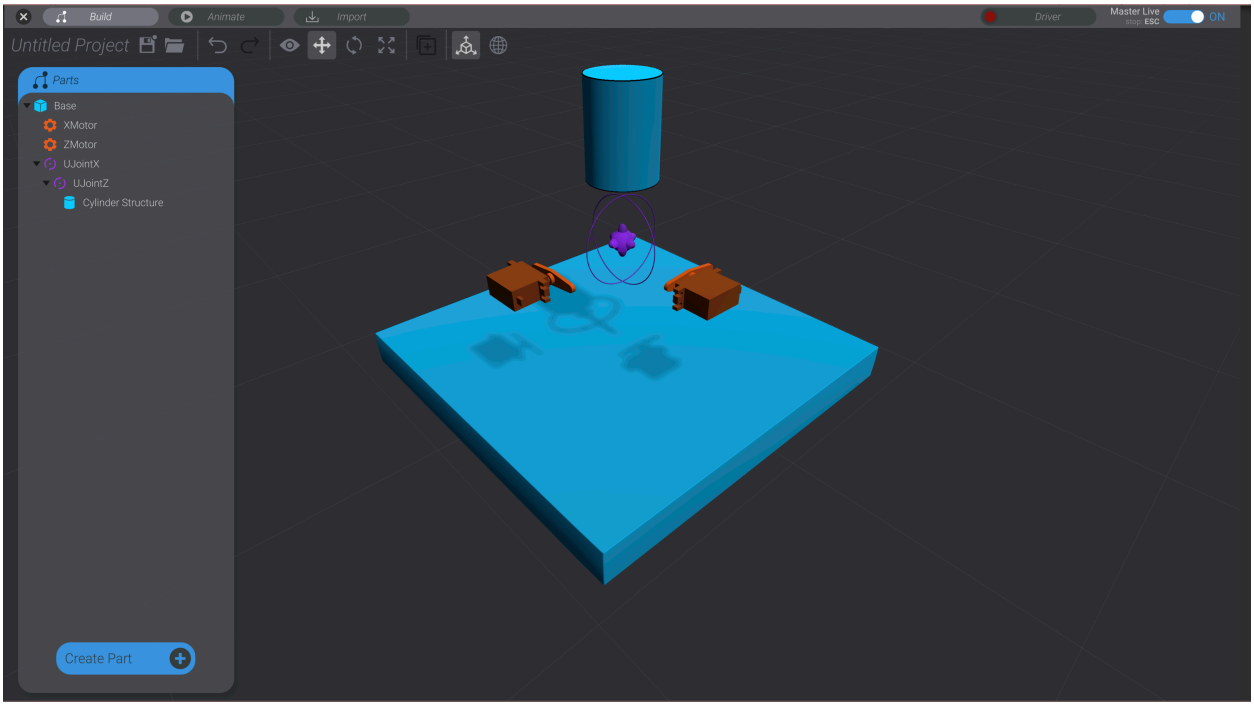
However, if we take a closer look at the example universal joint, it's really made of two joints combined that rotate in a single axis. There is first a joint that can rotate on the x axis:



Then, with the exact same pivot point and connected immediately to that joint, is a second one that can rotate on the Z axis:



To model this universal joint in Bottango, you would simply create the first joint, and set its axis to rotate in the X. Then you would create a second joint, as a child of the first joint, in the exact same position as the first parent joint (i.e. a home position and rotation of 0), with a rotation axis of Z. Subsequent joints, structure, and motors would be a child of this assembly, and you can rotate in both axes by rotating each joint.



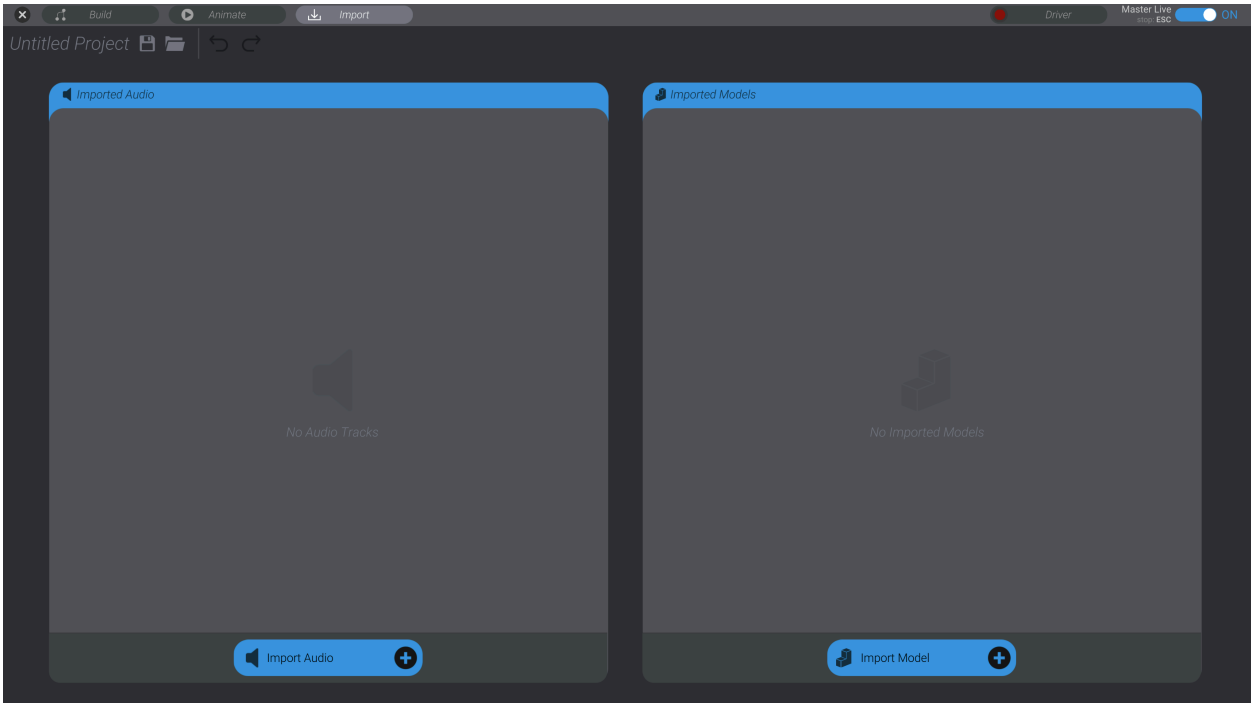
As for a true multi-axis joint, such as a ball joint, you would use the same setup. In a ball joint, it doesn't matter which axis is the parent of the other, so long as both joints share the same pivot point (i.e. location and rotation).

-10-
Audio

Importing audio

Bottango allows you to import audio tracks so that you can synchronize animations with audio. The current supported formats are .WAV and .OGG. If your audio track is in another format, there are countless free online transcoders to convert to either.

1 Click “Import” in the mode tabs section to switch to import mode.

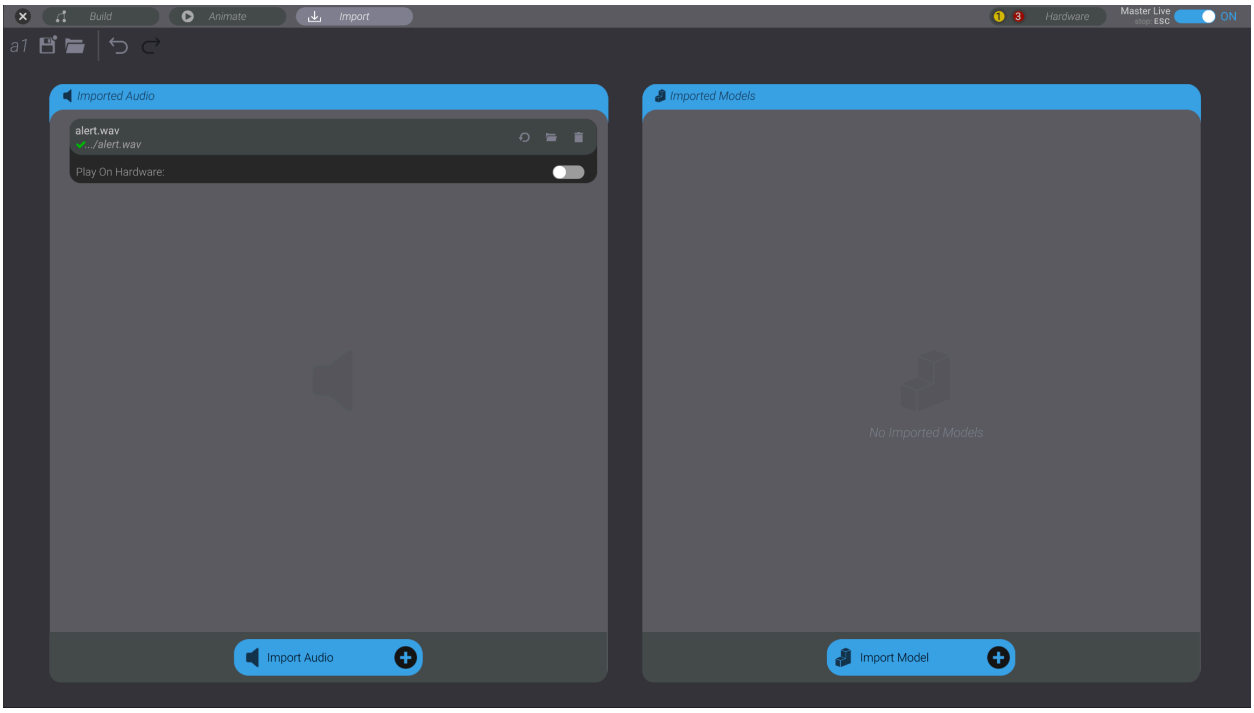


In Bottango, you can import two types of files: audio and 3D models. We’ll cover 3D models in a later chapter.

2 Click “Import Audio” to begin importing a new audio track.

3 Select the audio file you want to import.

The audio file will be imported into the project and listed.



Imported files (audio and 3D models) are saved to the project by either an absolute path or a local path. If a save file exists for the project, and the file is in the same directory or a child directory of the path for the project save file, the imported file will have a relative path. That means you could, for example, share a folder with a Bottango project file and an audio file, and open the project on another computer while maintaining the imported audio file.

If no save file exists for the open project, or the selected imported file is not in the same or a child directory as the imported save file, the absolute path to the file will be used instead.

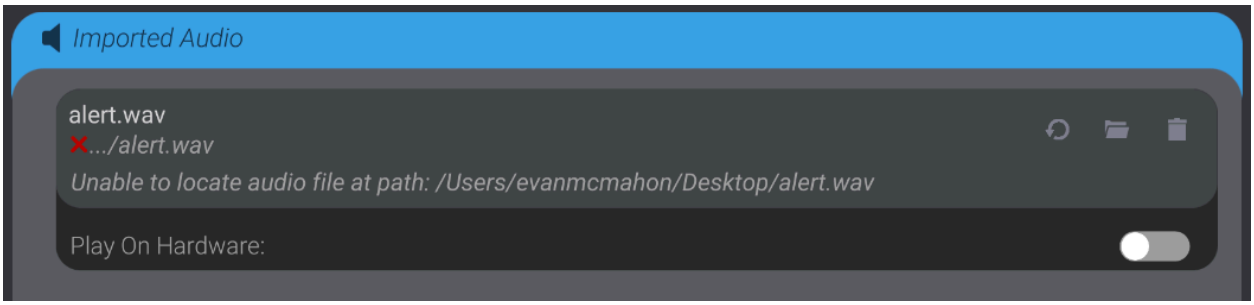
You can import as many audio tracks as you want in Bottango.

Imported file changes

If you change an audio file, it does not automatically reflect the change in Bottango. Bottango loads the audio file into memory on project load or first import, and keeps it unchanged there. However, if you make a change to the audio file on disk, you can press the reload icon to reload it and reflect the updated changes.

Missing or changing audio paths

When you open a project, if any audio tracks are not able to be located at the saved path, an "X" is shown in the audio list next to the path.



1. Press the folder icon to select a new path for the audio track. This will reimport the audio track at the new location and save that new path.

You can also do the same thing to change to a different audio track or set a new path for an audio track. If you want to use a different audio file for the track, you can change it by clicking the folder icon and selecting a new file path.

Finally, you can delete an audio track using the trash icon. However, be aware that any animation tracks that use this audio track will be deleted as well.

Triggering audio playback on hardware

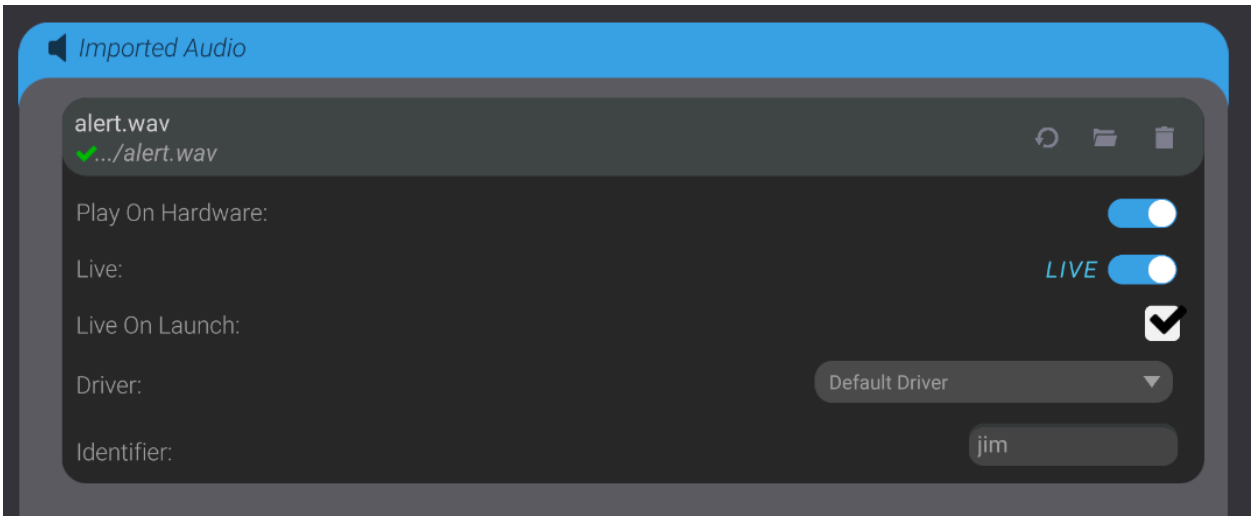
The easiest workflow for audio playback, is to simply use your computer as the audio playback device. Whenever audio plays in an animation, it will playback on your computer.

However, you can also setup an imported audio track to playback on Bottango controlled hardware as well. In order to do so, toggle on the "Play On Hardware" option on each track you want to play on Bottango controlled hardware

With this option on, you'll see some more configuration options. However, the big "idea" to understand, is that at this point, Bottango treats any animation track using this audio as if it were a trigger custom event. In order to understand how to respond to audio hardware cues, please read chapter 15 - Custom Events.

Just like any other custom event, you can set if the audio track event is live, if it should be live on launch, the associated driver, and the unique identifier for the trigger event. The custom identifier follows the same rules (alpha numeric, up to 8 characters).

Two limitations to note with playback of audio triggers on Bottango controlled hardware:



- If you enable "Play On Hardware" on an audio track, it behaves in every way like a trigger custom event. That means it will take up memory and count towards the maximum allowed registered effectors.
- Bottango driver code does not currently provide any audio playback functionality or code. There is not a single popular solution for audio playback on Arduino microcontrollers. You can listen to the trigger events to know WHEN to play audio and what track, but the HOW of coding for storing and playback back audio is still up to you.

-11-
Animating - Basics

What's in this chapter

We finally made it! The whole reason you're using Bottango: animating your robot! All the work we've done so far is to configure and set up your robot so that everything is in place to animate it.

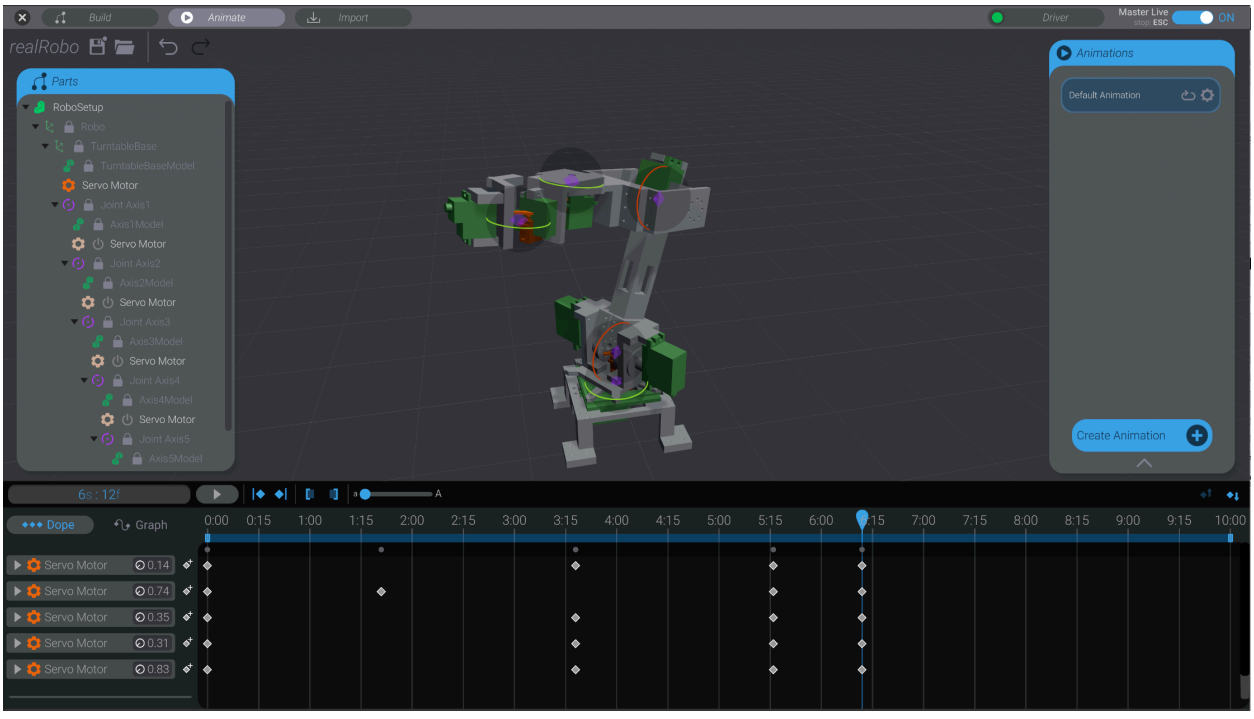
In this chapter, I'm assuming you have a project with a robot configured and ready to be animated, as outlined in all previous chapters. In this chapter we'll learn the basics of animating a robot.

You create animations in Bottango using the standard animation tools of keyframes and curves: create poses of your robot that are captured as keyframes, and then edit the interpolation curve that dictates how the robot moves from one pose to the next.

Animation mode

Click the "Animation" mode button in the top mode tabs section of the screen to switch to animation. While in animation mode you cannot create, edit, or destroy parts. Switch back to "Build" to perform those actions.

At the bottom of the screen is the animation window. At the right is the list of animations in the project. On the left, the parts list is shown just as in "Build mode"



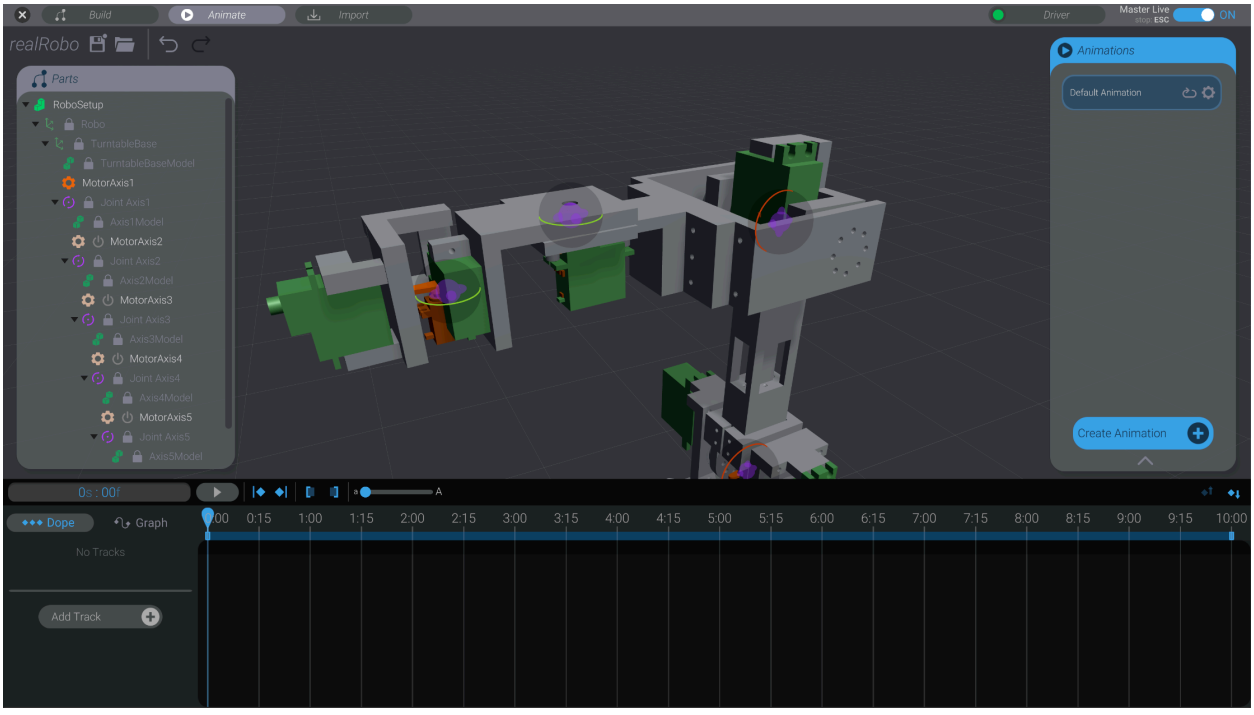
Project animations

Every Bottango project must have at least one animation. When you create a new project, an empty animation is created as well. You can create and duplicate animations, which we'll get into later. For now, just know that the animation we're creating in this chapter will be contained in the default animation.

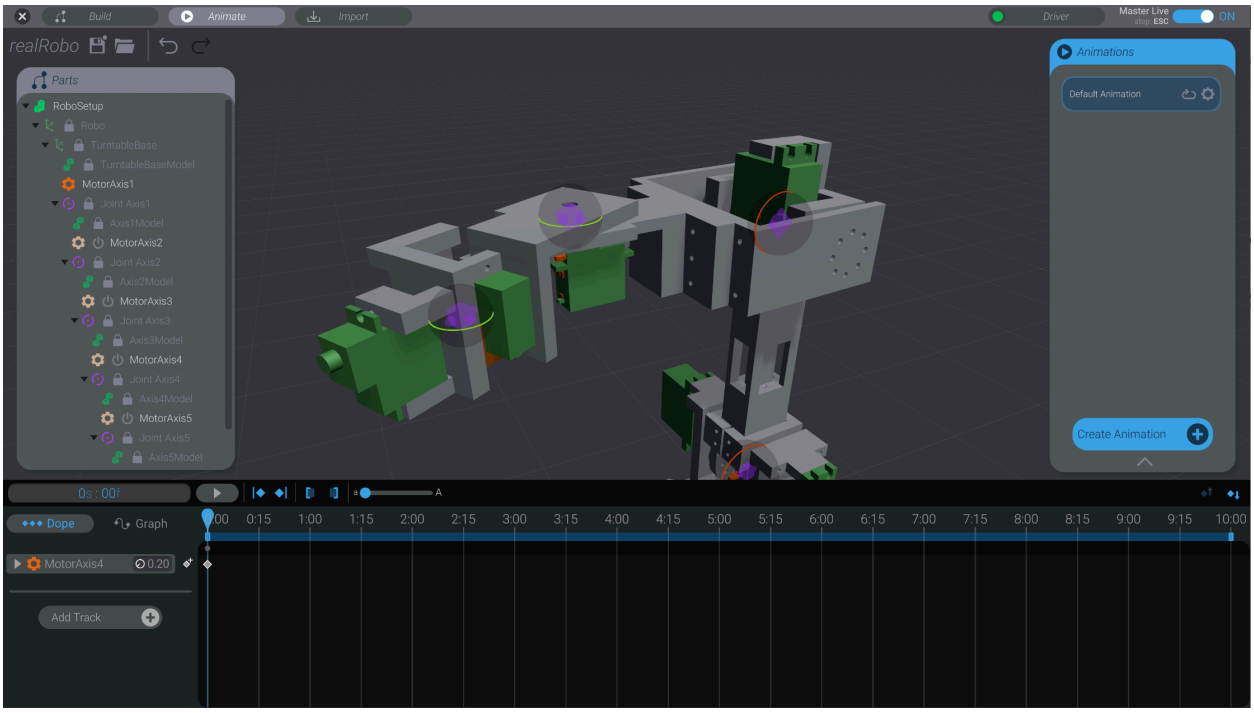
Animations are made of animation tracks. Each track contains a series of keyframes that are played back in the animation. Animations in Bottango can only have one animation track per motor. However, you can have as many animated motors in an animation as you have motors in your project. You can also have as many audio tracks in an animation as you'd like.

Creating keyframes

When you enter animation mode, you may notice that all configured joints are shown with their axis handles visible. The main way you animate and create keyframes in Bottango is to simply move or rotate the joints to the position or rotation you want. Bottango automatically captures the pose you create and stores it as a keyframe in the selected animation frame. Some animation programs refer to this behavior as "auto-key."



Drag on a joint handle to move or rotate it (as configured in build mode) and you'll see a new animation track created for the motor used in that joint, and a new keyframe on that track to capture the pose.

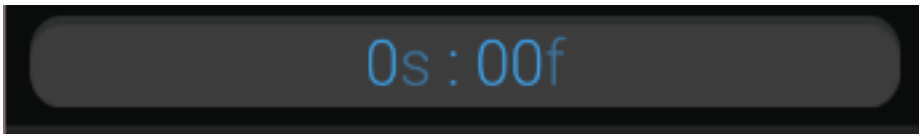


If you drag on that joint handle again, the new pose keyframe will again be updated automatically. Keyframes for motors are captured, like lots of things in Bottango, in terms of movement. You animate the keyframe between the min and max movement for the joint, which is expressed in the min and max signal for the associated motor.

Animation Time in Bottango

Animations have a duration. The default animation length in Bottango is 10 seconds; however, an animation can be as long as 4 hours.

On the top of the animation window, you'll see the timeline of the animation, from start to finish, expressed in the horizontal axis.



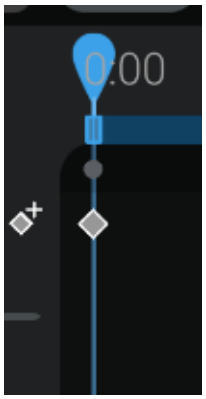
Time is expressed in the following format: Minutes : Seconds: Frames. So, for example, if you see on the animation window a time of "2s : 15f," that means 2 seconds and 15 frames. Or "3s : 00f" means 3 seconds and zero frames.

Animations in Bottango are authored at 30 frames per second. That means if you add 1 frame to "2s : 29f" (2 seconds and 29 frames) you'd end up with "3s : 00f" (3 seconds and zero frames.)

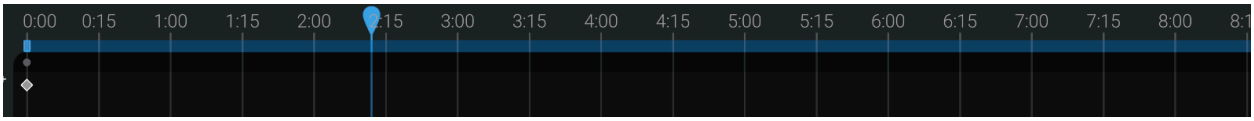
Don't panic! Animations on your hardware driver play at a MUCH faster frame rate than 30 fps. Animation frame rate on your microcontroller is determined by the speed of your microcontroller and how much else your microcontroller is doing. Bottango sends a start and end movement as well as a duration to your microcontroller to execute as fast as the hardware can, not individual calculated values. 30 fps is just for the visual convenience of where to put keyframes on your timeline. Your motors will update their real-world position on the order of 1000s of frames per second.

Controlling time

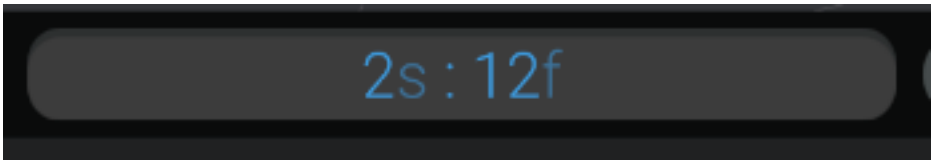
In the animation window, there is a blue scrubber.



The scrubber shows the currently selected time in the animation. Click and drag anywhere on the timeline to select a different time. When you do so, the scrubber will move to the clicked-on time.



You'll also see that there's a text version of the selected time:

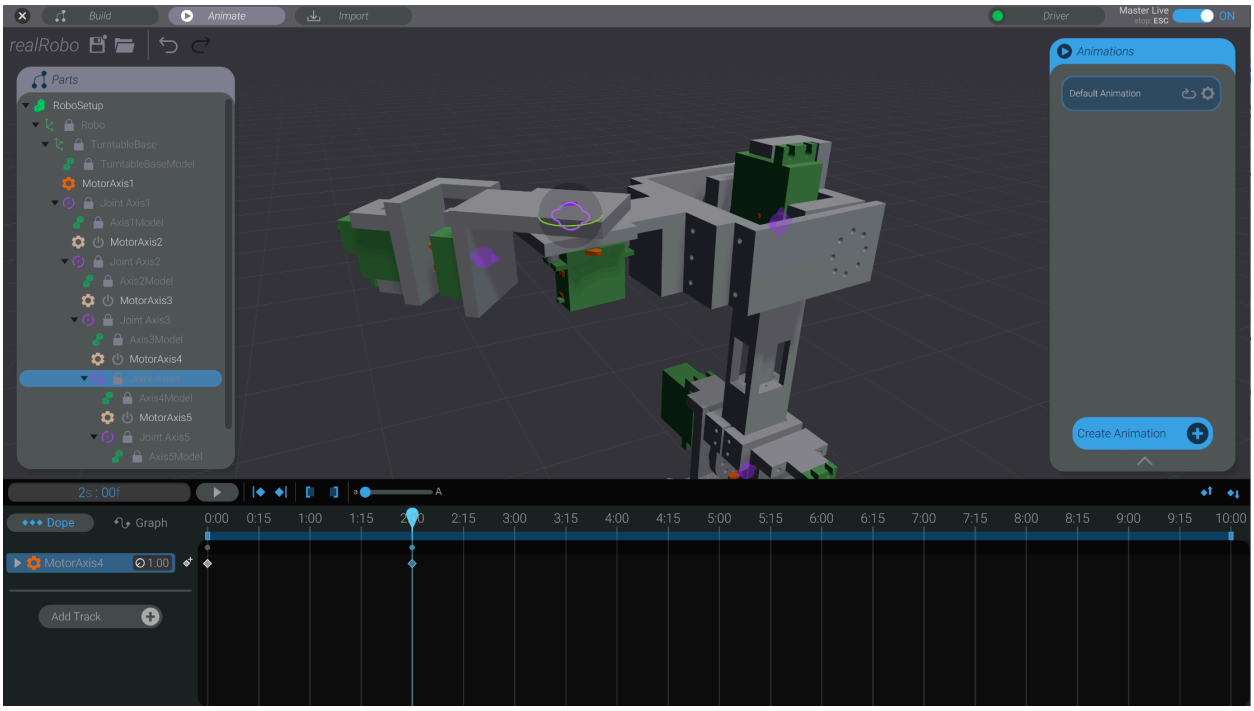


You can refer to this text to see what the current selected time in the animation is. You can also enter in a time manually to select, in the same Minutes : Seconds: Frames format. Again, for example, if you wanted to move the selected time to 3 seconds and 11 frames, you would enter 3:11.

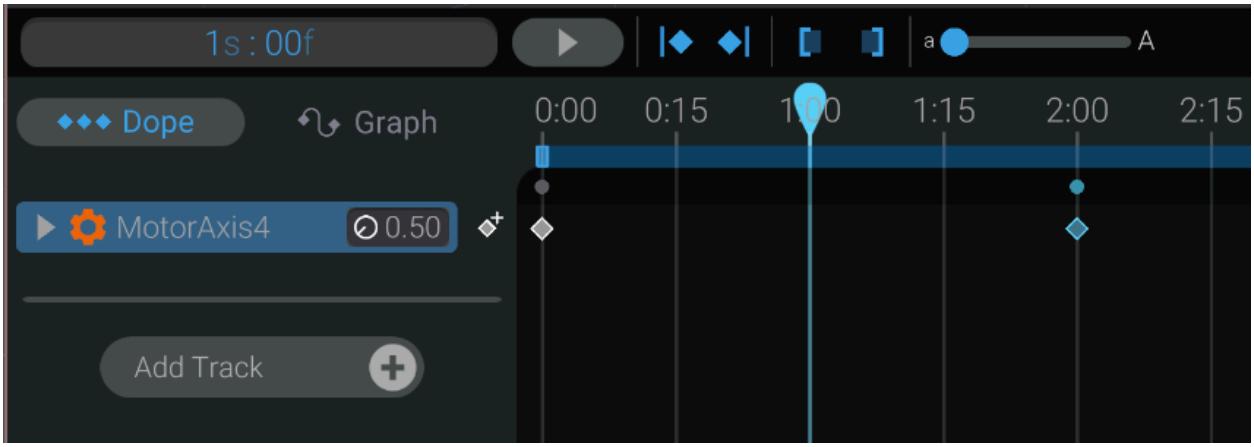
Adding more keyframes

Click on the timeline to select a time in the animation where you don't already have a keyframe. Then move or rotate the same joint you did previously. Because a track already exists for that motor, a new

track is not created. However, a new keyframe is added to that animation track. Now there are two poses of that motor at two different times in your animation.



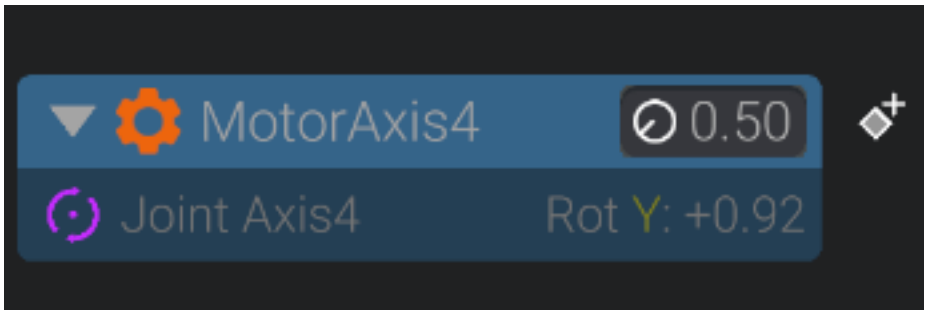
Drag on the timeline again now that there are two frames, and you'll see that the motor is interpolated between the value of the two keyframes. In this example there's one keyframe at movement 0.0 at time 0:00, and a second keyframe at movement 1.0 at time 2:00. The selected time is 1:00 (halfway between the keyframes' time), and the motor has been moved to movement 0.5 (halfway between the keyframes' values)



If you want to capture a keyframe at the selected time, you can press the manual Add Keyframe button as well.

This will insert a keyframe with the current motor value at the selected time.

If you want to see the end result rotation or movement of a joint, you can expand an animation track to see all the associated joints of that motor, and the derived value.



Scrubbing

As you move the selected time, you will notice that the motors in the animation move to the indicated values. The act of moving the selected time by dragging is called "Scrubbing."

When you first start scrubbing the timeline, you will notice that the robot transitions over time to where you first clicked down on the timeline. This is so that the robot does not immediately jump to the indicated time and possible damage your hardware.

However, once the transition to the selected time is complete, if you keep your mouse down and continue scrubbing, the transition delay is eliminated. As long as your mouse is down, the robot will immediately move its motors to the value as indicated by the selected time.

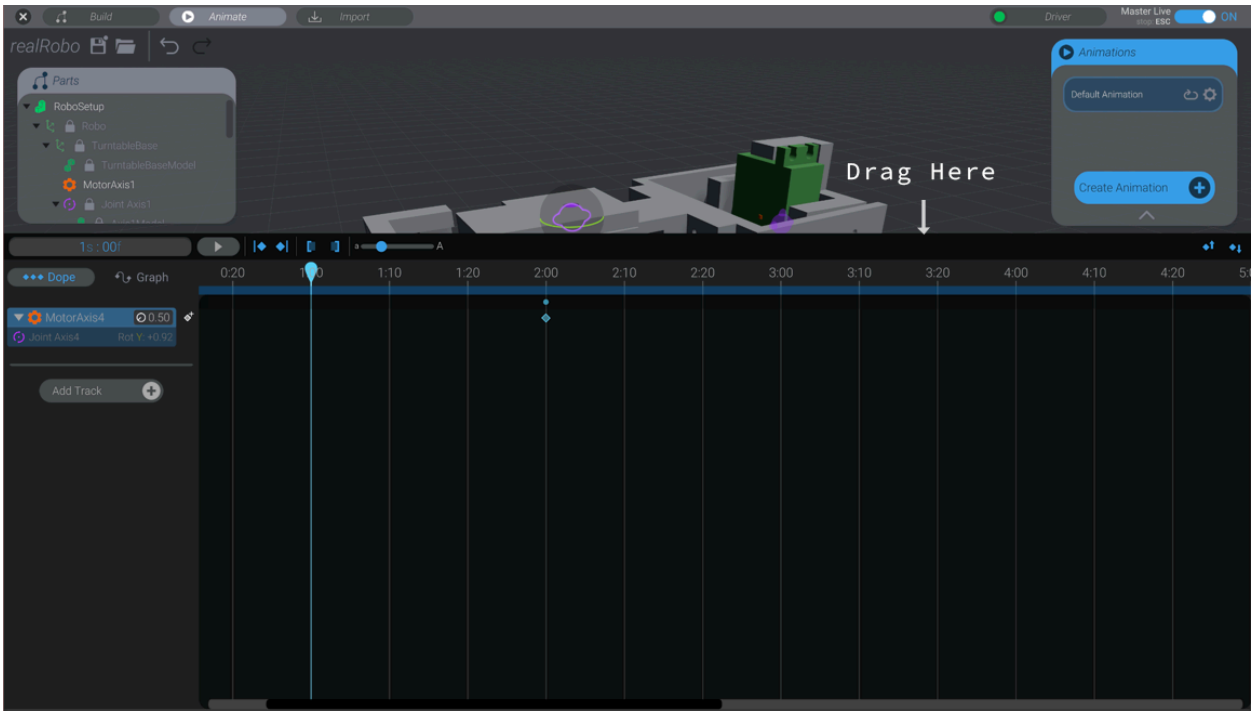
In order to best allow you to evaluate your animation, after the initial transition to the selected time, your robot will move as fast as possible while scrubbing. Be aware of this and don't scrub faster than your robot can handle.

If all of your hardware drivers are currently disconnected, Bottango will scrub instantly and skip the transition, since you are editing your animation "offline."

Zooming in and out

By default the animation window scales time so that the entire animation is viewable on screen. You can adjust the time zoom slider to zoom in and have more fine grained control of keyframes.

You can also resize the animation window by dragging on the top portion.



Editing keyframes

If you want to change the value of a keyframe, simply move the selected time to the same time as that keyframe, and then repose the robot how you want it. The keyframe will be updated automatically.

It can be a little finicky to get the time exactly where the keyframe is, so there are two buttons to help:

The button on the right will move the selected time to the time of the next keyframe after the current time. The button on the left will do the same except with the previous keyframe. You can also use the keyboard shortcuts J and K.

If you hold down option/alt while pressing J and K or pressing the above buttons, you will move forward or backwards a single frame. You can also move forwards or backwards a single frame using the arrow keys on your keyboard.

If you want an exact movement value, you can type it in on the track:

Enter a movement between 0.0 and 1.0 to modify the keyframe at that time (or to create a new keyframe if one doesn't exist).

Selecting and moving keyframes

Click on a keyframe to select it. You can change the time of the keyframe by simply dragging on it.

You can select multiple keyframes by holding down shift. You can add or remove the selected keyframes by holding down command on OS X or control on Windows.

With multiple keyframes selected, drag on one of them to move all selected keyframes.

Deleting

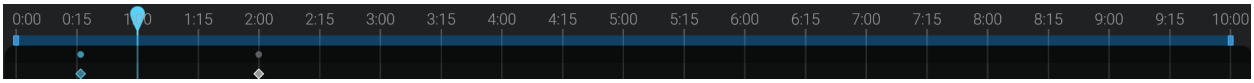
Press delete to delete any selected keyframes. You can also select animation tracks and press delete to delete them. When you delete an animation track, all associated keyframes in that track are deleted as well.

Playing the animation

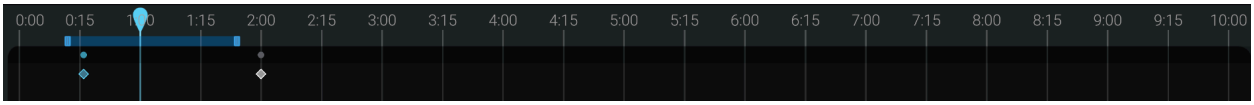
Once you've added a few keyframes, press the play button (or press the space bar) to play the animation. Press the play button again or press the space bar to stop the animation if you want to stop it before it finishes.

Play range

You may have noticed a blue bar right below the timeline in the animation. This is the play range.



By default, the play range covers the entire span of the animation. However, if you only want to play a portion of the animation, you can drag the two ends of the play range to cover the desired portion only.



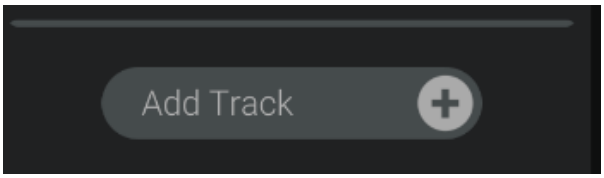
When you hit play, if your scrubber is in the play range, you will play the animation starting at where your scrubber is.The animation will stop when it reaches the end of the play range.

If your scrubber is outside (or at the very end) of the play range, Bottango will jog to the beginning of the play range first, and then begin playing.

Adding audio

It's likely you will want to play back the audio you imported in chapter 9 into your animations. Any animation clip can have as many audio tracks as you'd like. Additionally, an animation clip can have multiple tracks of the same audio.

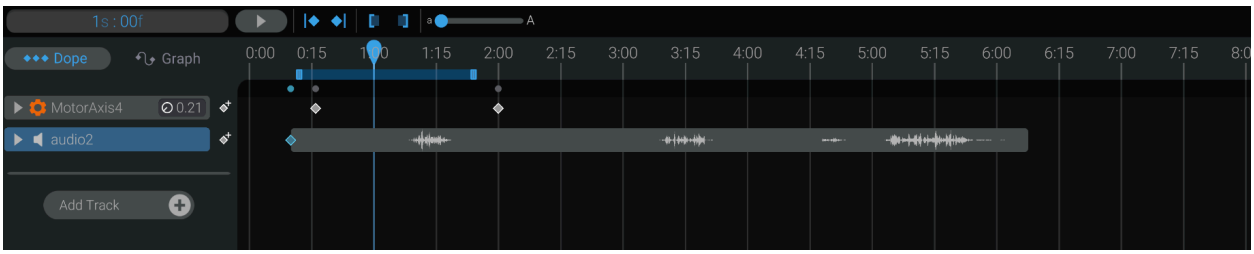
Click the "Add Track" button to select the kind of track you'd like to add.



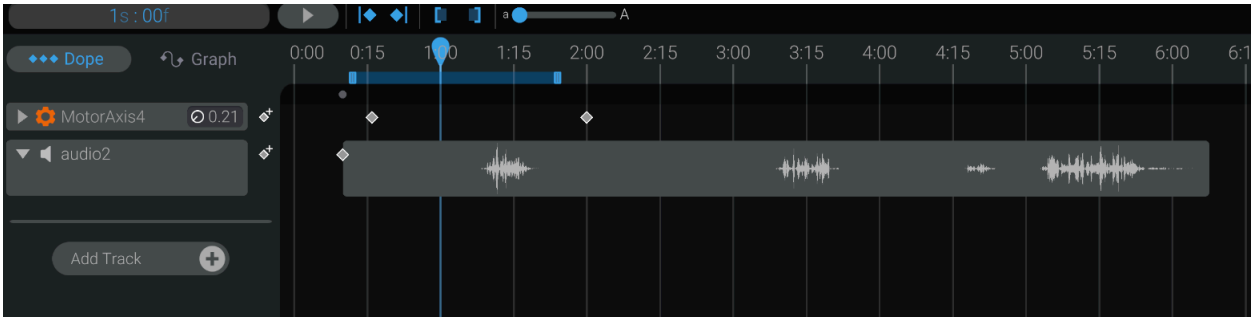
You'll see two options: Motor and Audio. (You added a motor track automatically when you moved a joint that didn't already have a track. You can manually add a track for a motor this way as well.)

Click on the audio track option, and then select an audio track to add to the animation.

A track and a keyframe for the audio will be created.



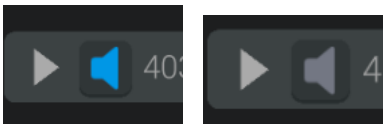
Audio tracks show the wave form of the audio to help you visually synchronize your motor keyframes with the sound of the audio. As you scrub time in your animation, the audio will play back as well. You can expand the audio track to see a larger version of the waveform.



Each keyframe in the audio track restarts playback of that audio clip. Click the add keyframe button to play the audio again (or restart it) in the same track.

Audio will play back not just when scrubbing, but when playing the animation as well.

You can click the speaker icon in the list of tracks to mute and unmute the audio track.



Right now Bottango only outputs the audio through your computer's audio output. However, we hope to add support for more advanced audio output options in the future.

Copy and paste

You can copy and paste keyframes. Select your keyframe(s) and use the standard command/control C and command/control V to do so, or use the copy / paste buttons:

When you paste keyframes, the keyframes start from the current selected time. So if you copy a keyframe from 2:00, and paste it while 3:00 is selected, it will be pasted at 3:00 (and any additional keyframes offset in time accordingly.)

The behavior of copy / paste behaves differently in different circumstances.

If you have copied keyframes from just one animation track, Bottango will attempt to paste those keyframes into any track selected. This will allow you to copy movements from one motor and paste them onto a track for another motor.

If you have copied keyframes from multiple tracks, though, Bottango will only paste those keyframes back into the original tracks (at the new selected time).

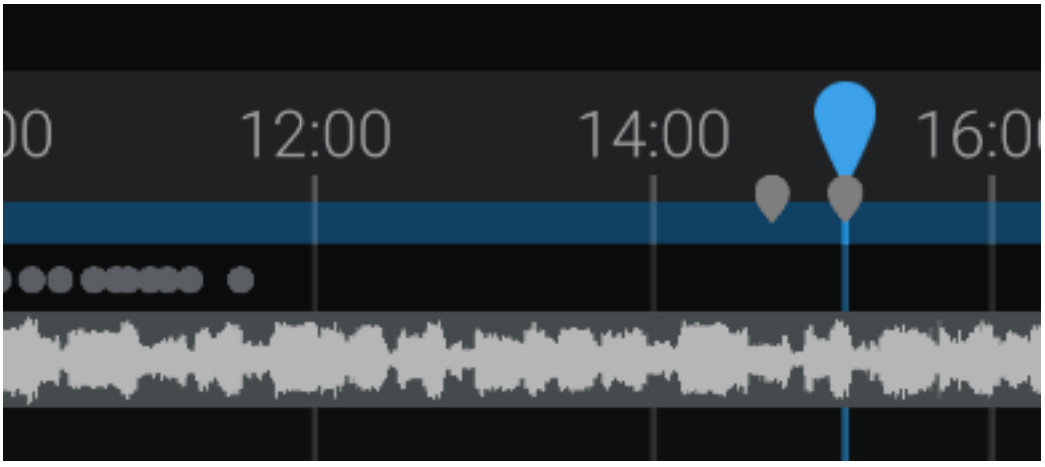
You can paste into missing tracks as well. For example, you could copy keyframes from one track, then move to another animation that doesn't have a track for that motor. As long as you have no other tracks selected, when you paste in that second animation, Bottango will create a new track for that motor and then paste the keyframes.

Adding Markers

Bottango allows you to add markers to animation tracks and clips themselves. Markers allow you to more easily synchronize keyframes with each other, or with audio. Markers also allow you to visually flag parts of your animation.

To add a marker, press the "add marker" button.

A new marker will be created at the current selected frame.



If you have any animation tracks selected, markers will be added to all selected tracks. Otherwise a marker will be added to the clip itself.

You can also add markers by pressing the "M" key. This can be especially helpful, as you can add markers while an animation is playing. As an example, if you wanted to add markers on the beat of an audio track, you could select an audio track, start playing the animation, and then press "M" at each beat. This will let you have visual markers at each beat in the audio track, for easier synchronization.

You can also select and drag markers to change their position.

The delete markers button will delete any selected markers. If no markers are selected, the delete markers buttons will clear all markers from the selected tracks (or animation clip if no tracks are selected).

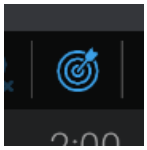
Markers added to tracks are only visible if the track is expanded.

Collapse a track to hide its markers.

Retargeting an Animation Track

After creating and animating an animation track, you may want to keep that track, but have it target a different part. Maybe you imported a new version of your 3d model. Or maybe you changed the setup of your robot and want to animate a motor directly instead of a joint.

With a single animation track selected, you can press the "retarget" button.



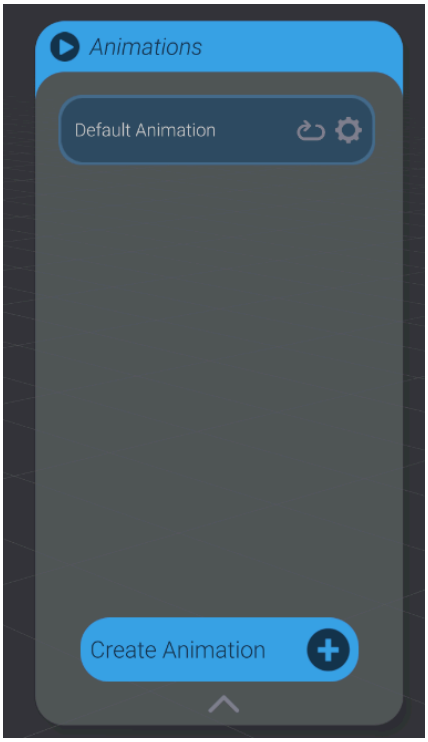
This will prompt you to select a part that doesn't already have an animation track in this animation. When you select it, the animation track will now animate that part instead.

After you delete a part, any animation tracks for that part will remain in the animation. They will instead be shown with in a missing part state, with the name of the last associated part.

You can retarget that track just like one that has a part, and select a new part to replace the missing part.

Editing and creating new animations

You can change some of the details of the animation itself using the Animations window on the right. You'll see that we have one animation to start: Default Animation.



If you want the animation to loop while playing (i.e. return to the start and play again after finishing playing), click to toggle the loop option.

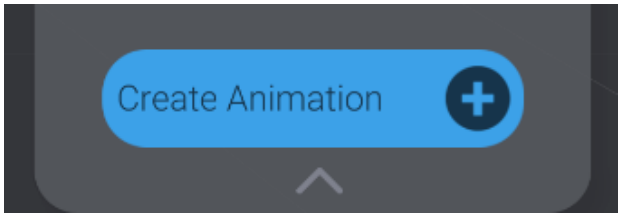
You can also click the settings button to edit a few more details of the animation.

Here you can change the name and duration of the animation. Input the desired animation duration the same way you would enter time when selecting time in the Animation window (Minutes : Seconds : Frames.).

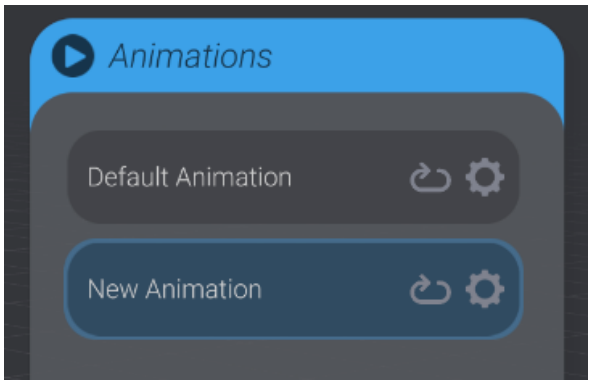
If you set an animation's duration to shorter than it is currently, any keyframes that currently exist beyond that time will be deleted.

You can also duplicate an animation by pressing the Duplicate Animation button.

Finally, you can create a new animation in the Animation window by pressing the Create Animation button.

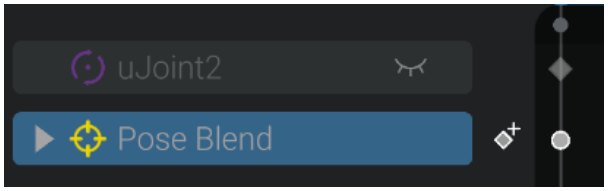


Click on an animation in the Animations window to select and begin editing that animation.



Obscured Animation Tracks

In some cases, an animation track may be shown in an "obscured" state. Obscured tracks are ignored when playing back an animation and are locked and can't be edited.



This happens when you create an animation track for a part, and then create a second animation track that takes priority of controlling that same part. As an example, you animate a joint, and then you animate a Pose Blend that also controls that joint (see the chapter on Pose Blends). In that case, the initial direct animation track of the joint is obscured by the track for the Pose Blend. If you remove the Pose Blend track, the joint track will no longer be obscured and you can animate it again.

In the above example, the track uJoint2 is obscured by the track Pose Blend, as the Pose Blend track animates the uJoint2 part as well.

Graph view

Bottango uses a default interpolation curve to interpolate between the two keyframes over time. The default interpolation curve accelerates in the beginning and decelerates in the end for a smoothed-out movement.

But in addition to controlling your robot's poses at keyframes , Bottango allows you to control *HOW* your robot animates between those poses. The next chapter is all about the graph view, which allows you to control the interpolation of your keyframes.

-12-
Animating - Graph View

Animation interpolation

Bottango has an animation graph view, which allows you to view and edit the interpolation of your keyframes.

Imagine the most basic movement between two poses: animating at a linear, constant speed. A hand, for example, could start in one position, and then move at a constant rate until it reaches its destination. It's easy to imagine all kinds of more interesting alternatives:

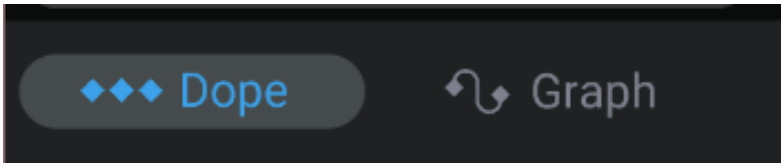
The hand starts moving slowly and then rushes to its final resting spot where it stops suddenly.
Or
The hand moves back slightly before springing forward to its final resting spot.

Both of the above involve moving from one point to another, but have so much more visual interest and character. You use the animation graph view in Bottango to control how a robot moves between each keyframe.

Switching to graph view

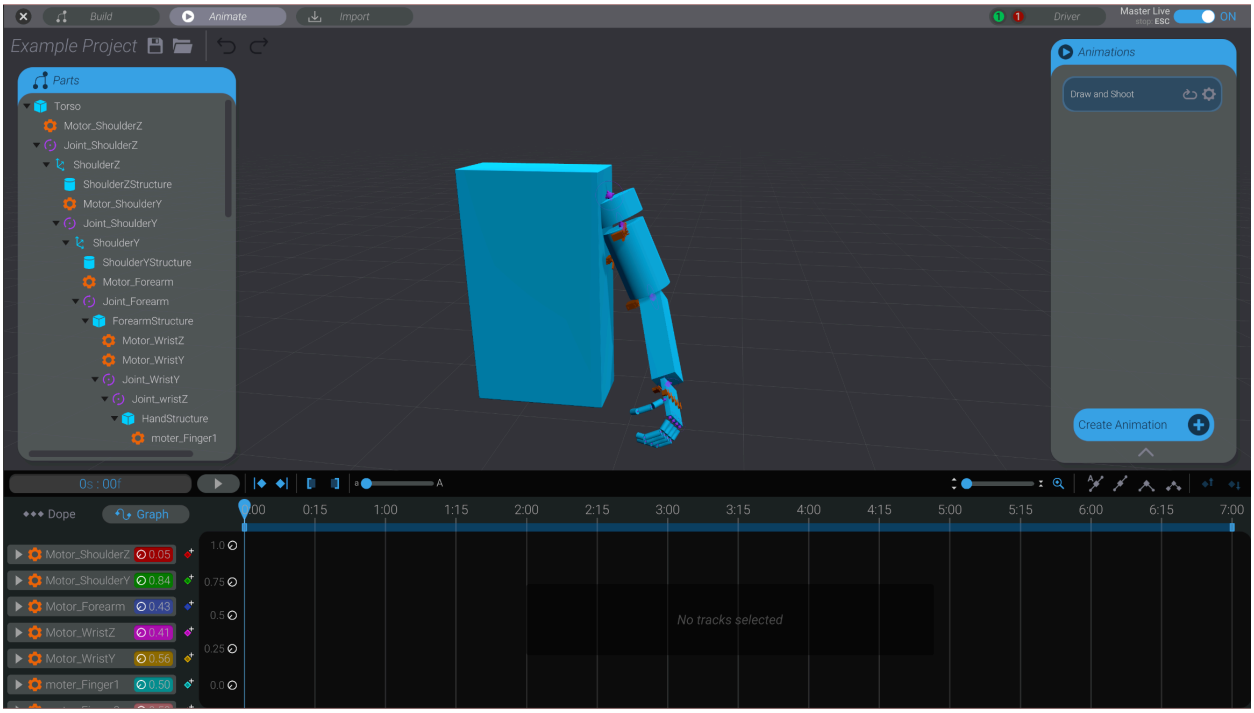
The default view in the animation window is the "dope sheet." This is the view we used throughout the previous chapter, creating and editing keyframes.

Press the graph icon to switch to graph view:

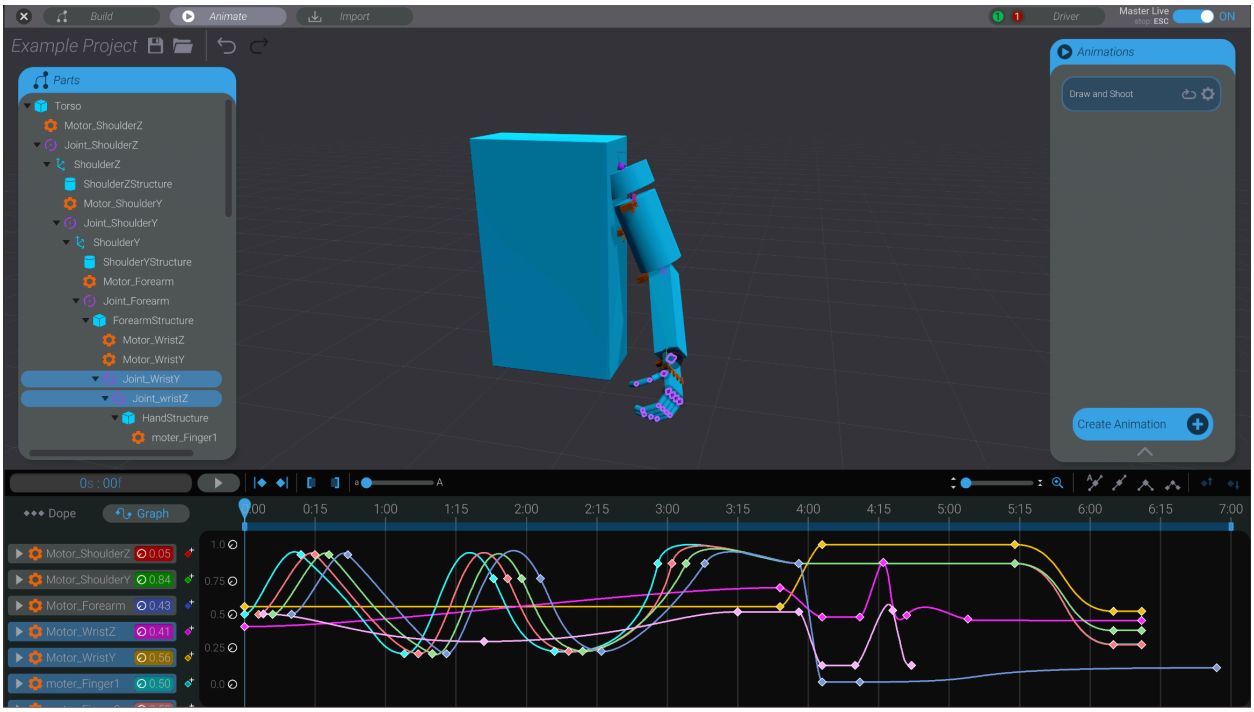


Showing and hiding graphs

When you enter the graph view, if you have no tracks selected, there will not be any graphs shown.



In order to view graphs, you need to select either tracks in the the tracks list or joints in the 3d view of your robot. The graph view will always show the graphs of selected tracks / joints (or nothing if there is no current selection).

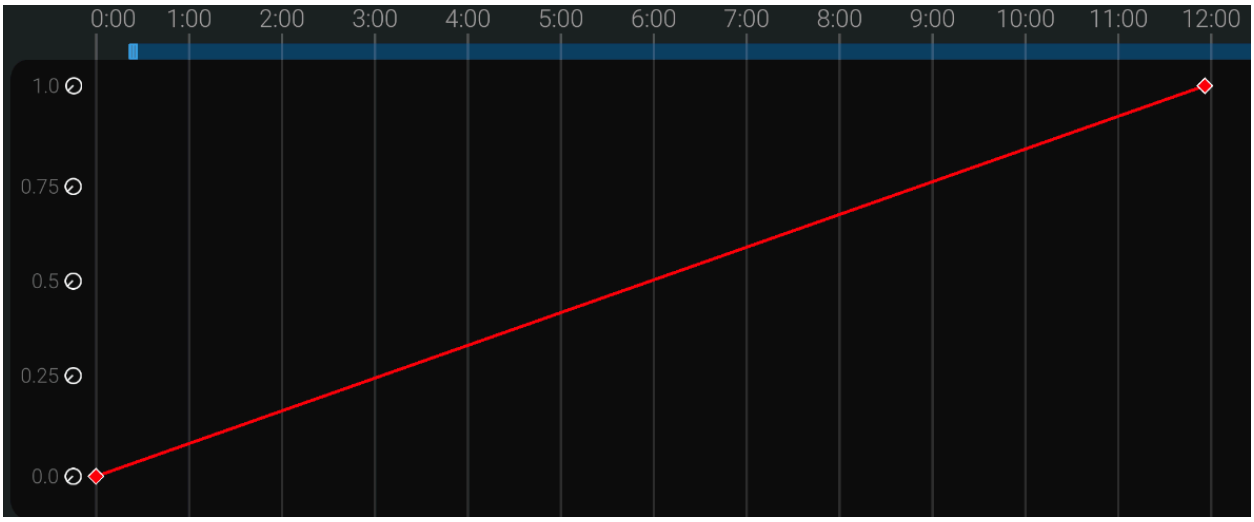


Reading the graph view

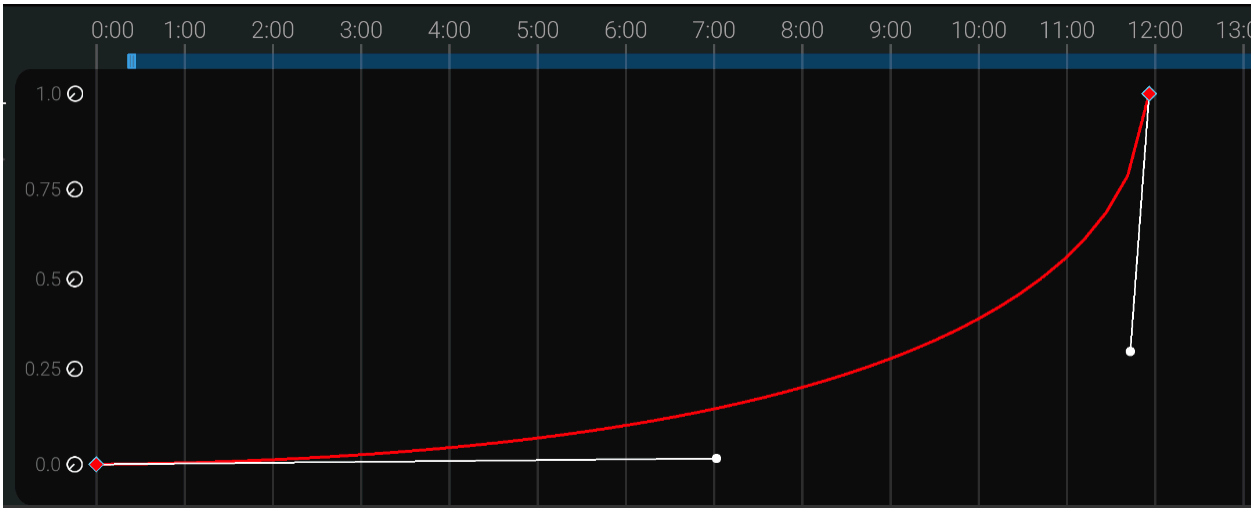
Just like in the dope sheet view, the graph view shows time in the X axis. However, the dope sheet view just uses the X axis. The graph view shows the calculated movement of each animation track in the Y axis over time.

What this means is best illustrated in examples.

To start, here's the most basic movement type: a linear, constant speed. This is the example of a hand moving at a constant rate until it reaches its final destination, then stopping:

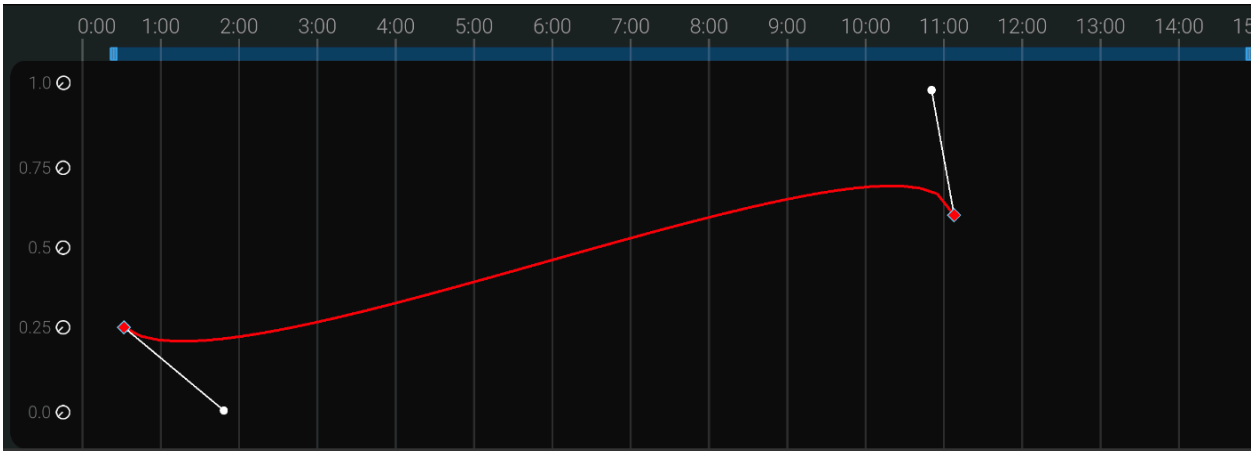


Here is the hand moving slowly in the beginning, and then rushing to a screeching stop:



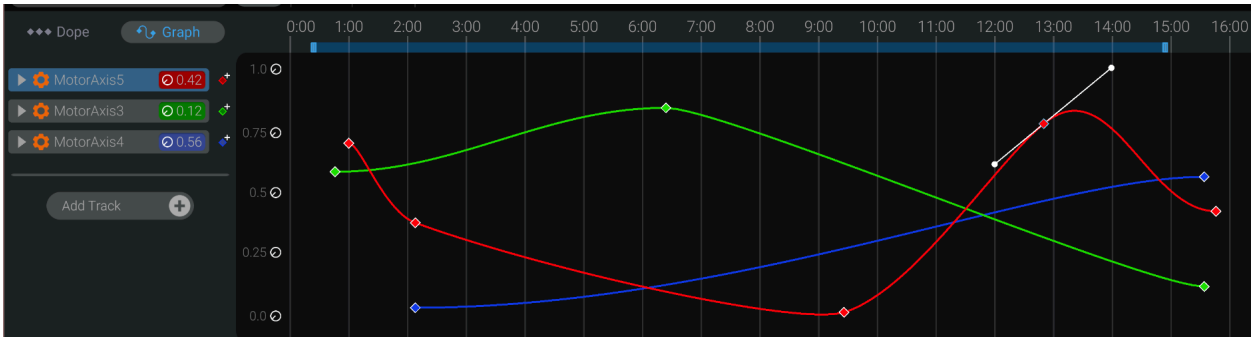
Notice how in the above example the slope of the curve is gradual and then suddenly stops. You'd see the same behavior in the movement of the robot.

Finally, here's a hand that moves back slightly before springing forward to its final resting spot (overshooting a bit on the way).



Track colors

Each track in the graph view is assigned a color. You can see the color of the track in the list of tracks on the left side of the animation window.



The corresponding chart in the graph view for that track shares the same color.

Audio is hidden

Only animation tracks that have movement are shown in the graph view. Audio tracks are not shown while in graph view. To view audio keyframes and wave forms, move back to the dope sheet view.

Editing in the graph view

Just like the dope sheet, you can select, delete, and move keyframes in the graph view. However, in this view, you can move keyframes in both dimensions: time and value.

Moving a keyframe left and right changes its time, just like in the dope sheet view. Moving a keyframe up and down changes its value.

You can also change the slope and shape of the curve. Select any keyframes in the graph view, and you'll see the interpolation handles for that keyframe.

The handles allow you to sculpt the line of the animation using Bézier curves. Dragging on the handles of each keyframe allows you to sculpt the shape of the curve as the motion moves from that keyframe to the next (or from the previous keyframe into this one).

Locking keyframe drag axis

If you want to only change the time of the selected keyframes when dragging them, or instead only change the movement value, you can lock the drag axis in the graph view:

Click the "left/right" lock to toggle only changing the time value of selected keyframes.

Click the "up/down" lock to toggle only changing the movement value of selected keyframes.

Only one lock can be enabled at a time. To go back to no locks, click the currently enabled lock.

Keyframe interpolation settings

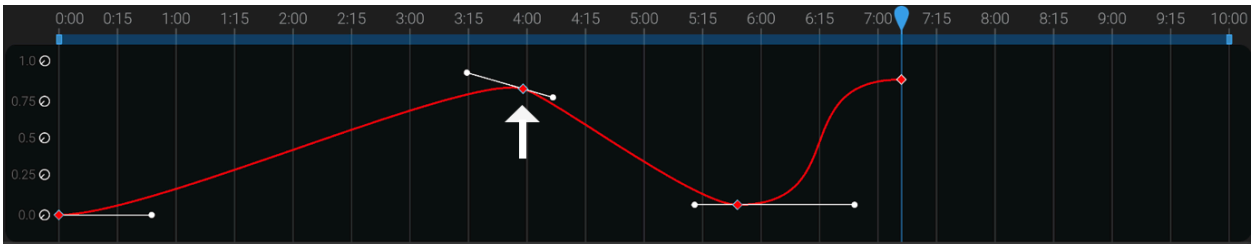
Each keyframe has one of four interpolation settings: auto smooth, smooth, broken, or linear.

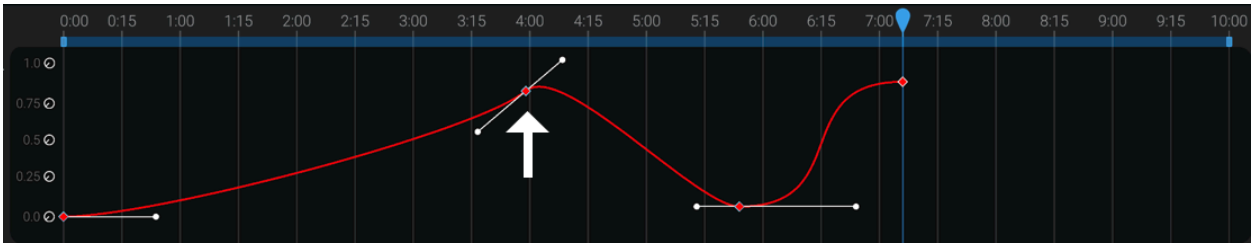
You can change the interpolation setting for a keyframe by selecting it and clicking one of the four interpolation type buttons.

Auto smooth interpolation

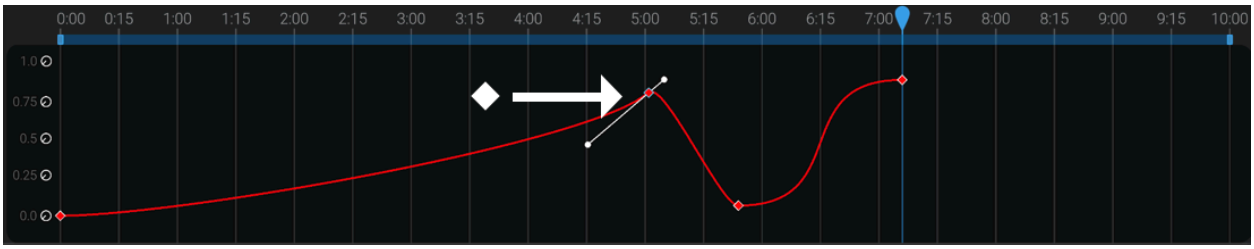
By default, keyframes are created with an auto smooth interpolation. This means as you drag one side handle of the keyframe, the other handle moves inversely, so as to result in a smooth curve. As you change the frame of a keyframe, the curve will stay the same shape.

Here you can see how the handle of the keyframe second keyframe is moved, both sides move together:





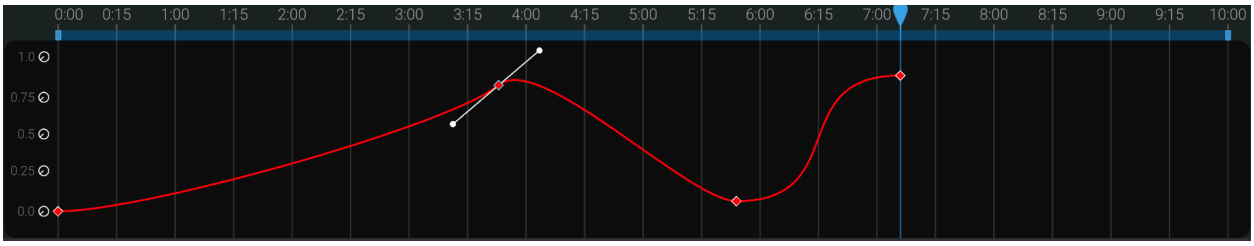
Here you can see how the handle changes its proportions as you change the keyframe's time in order to maintain the shape of the curve.

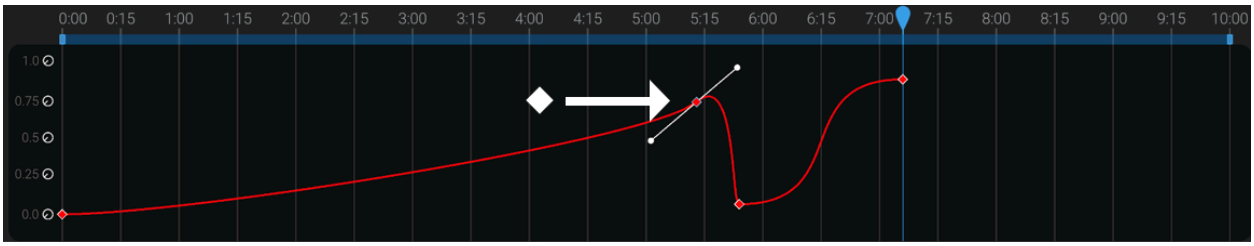


Smooth interpolation

Smooth interpolation behaves like auto smooth interpolation. As you move one side of a handle, the other side moves inversely.

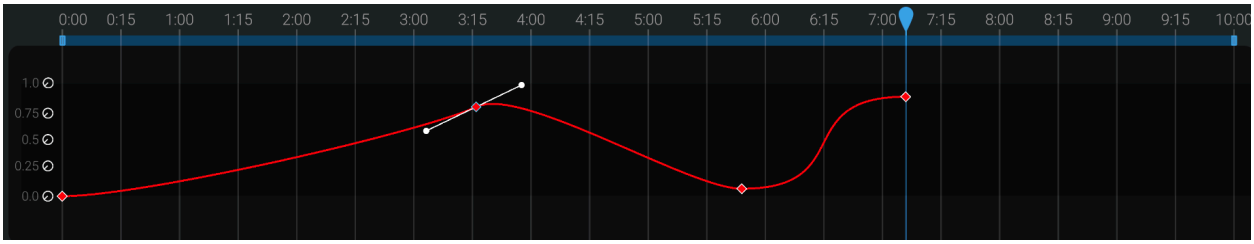
However, smooth interpolation will not change its proportions as you change a keyframe's frame.

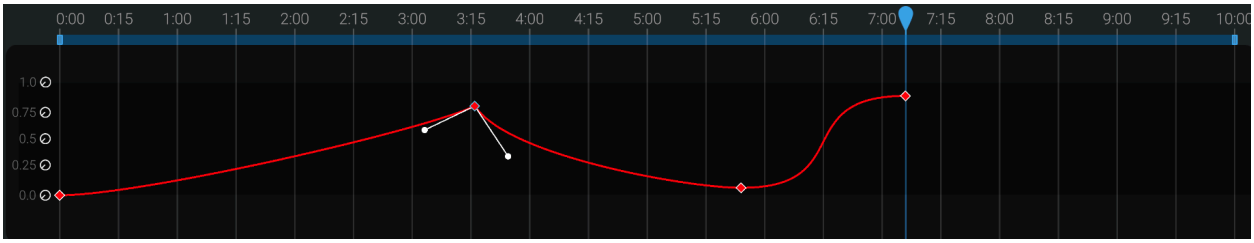




Broken interpolation

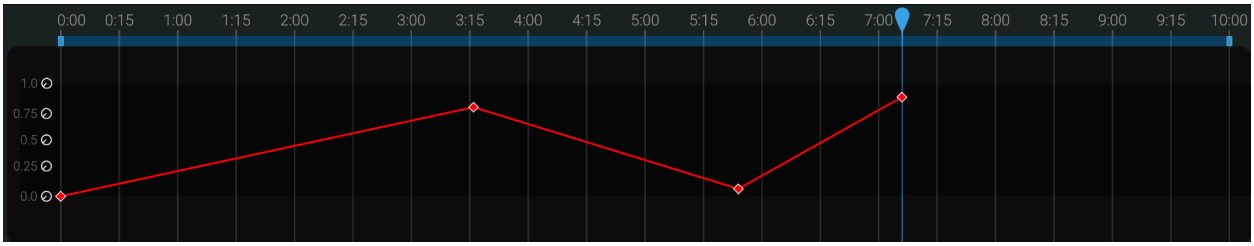
Broken interpolation does not adjust the handles at all as you manipulate them. This allows you to have a sudden change in direction.





Linear interpolation

Linear interpolation automatically adjusts the curve between keyframes to be at a constant rate.

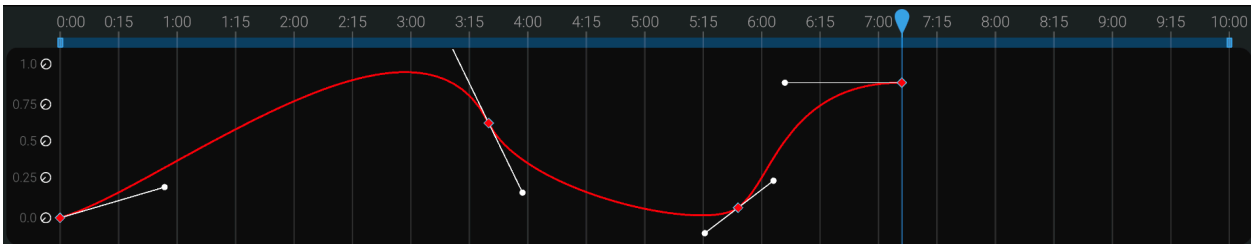


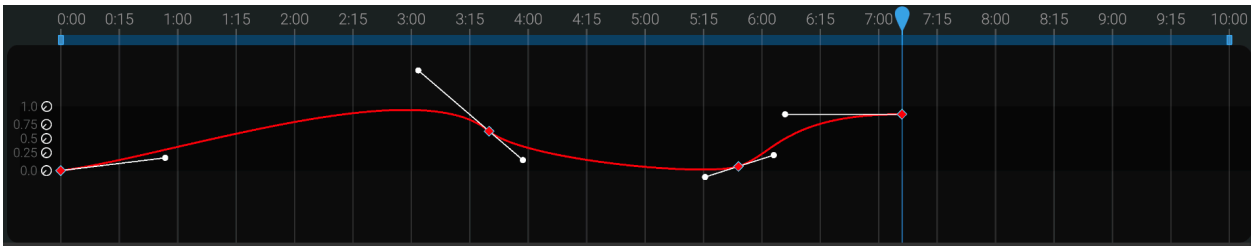
When you convert a keyframe to linear, you indicate if you want in (the left handle), out (the right handle), or both sides of the handle to be linear.

Adjusting graph Y size

As you edit keyframes in the graph, you'll be constrained to adjust the movement of the keyframe between 0 and 1. This means it will always be visible in the default graph view.

However, it's easy to move the handles of a keyframe out of the default view. You can change the scale of the graph view with the Y slider:

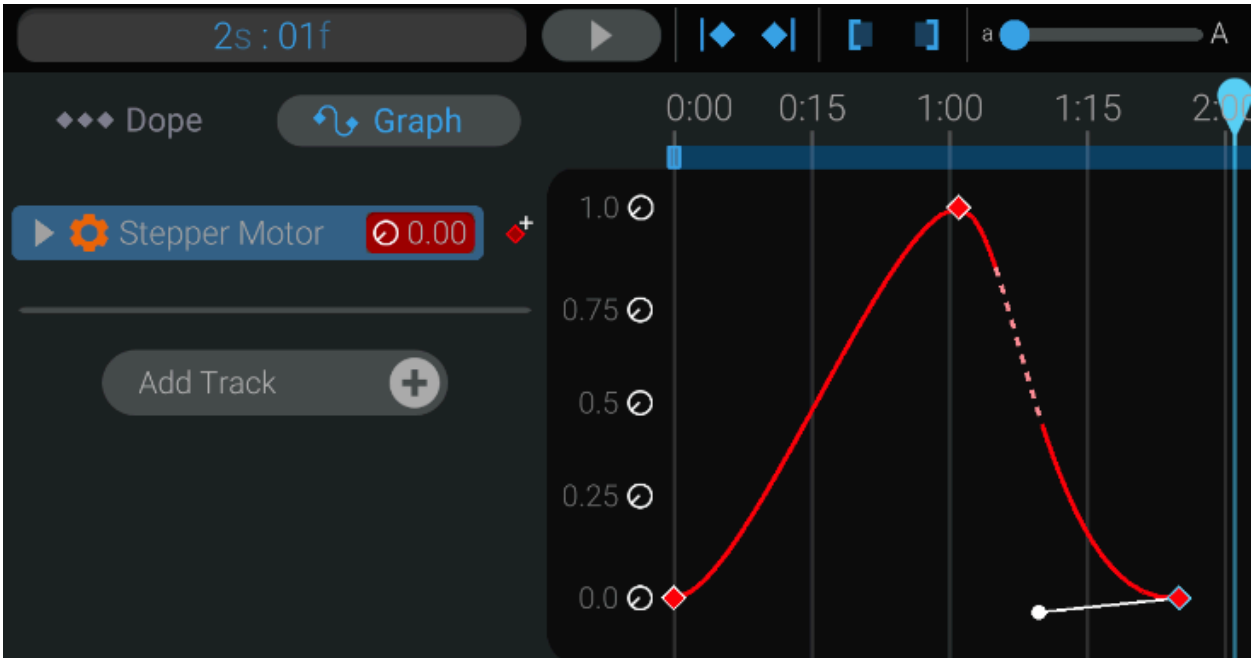




Additionally, you can press the Z key or press the auto zoom button to automatically adjust the zoom of the graph menu to show all selected handles.

Maximum speed in animations

While viewing the curves of an animation, Bottango will let you know if a movement is faster than the maximum speed you entered for the associated motor:



Here you can see that on the downward portion of the curve, the stepper motor would need to move faster than you allowed in the configuration of that motor. This is visualized by the dashed portion of the curve.

You can still play this animation, and your motor will move at its maximum speed (but no faster) to try and catch up. The end result will not move faster than your maximum speed, however it may not be in perfect synchronization with what you see and animate in Bottango.

You should, when possible, change the maximum speed of the motor, or slow down the movement when you see this warning.

Part 3 - Advanced Features

-13-
Stepper Motors

What's in this chapter

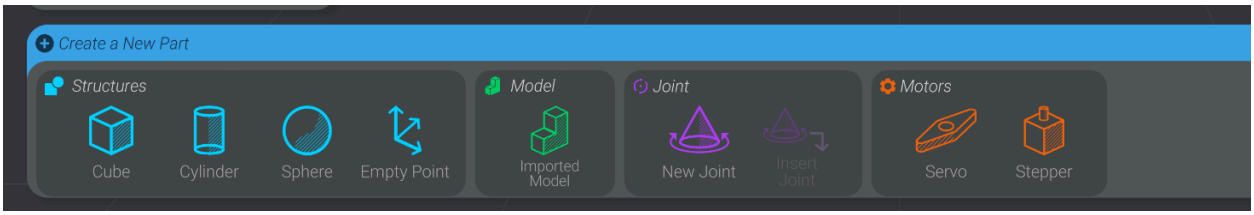
In this chapter, we'll cover the process of adding and configuring stepper motors.

Wherever possible, stepper motors behave the same as Servos. This chapter covers just what is different between stepper motors and servo motors. If you have not already, read chapter 8 - Motor basics and Servos.

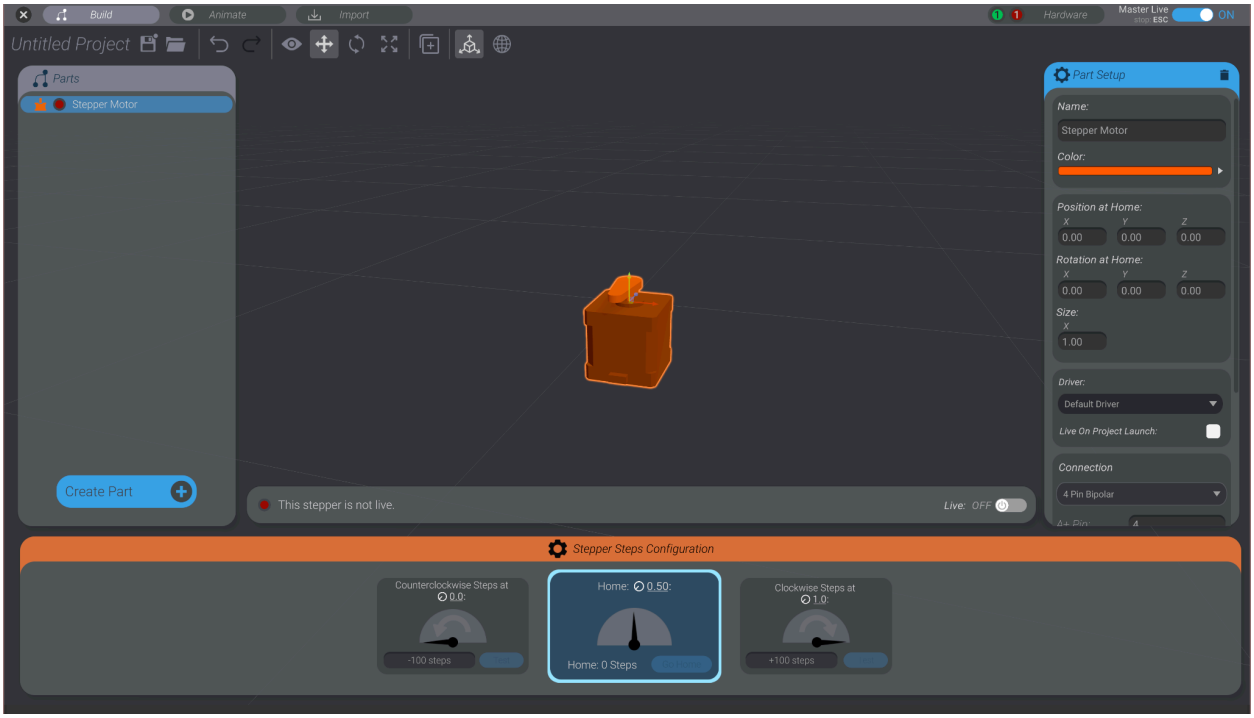
The biggest difference between stepper motors and servos is that stepper motors are "open loop" control, whereas servos are "closed loop" control. This means that while servos always know where they are, and can accurately go to a position indicated, stepper motors need to be brought to a known location first, and then move from there.

Creating a Stepper Motor

Stepper motors are created much the same way as servos are:



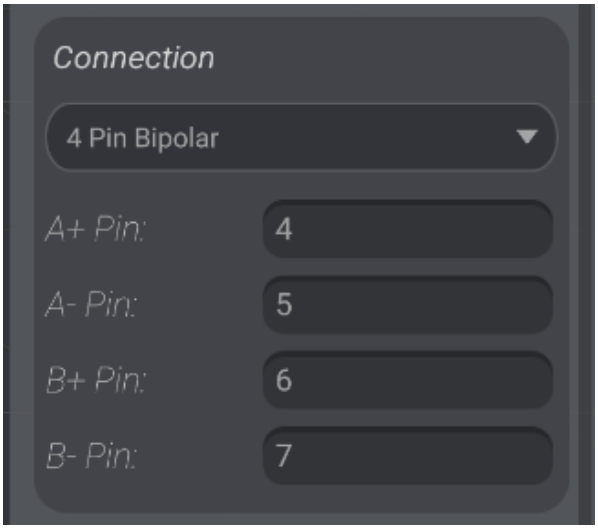
Click or drag out a stepper motor to create one:



Stepper Motor Connection

Bottango currently supports three kinds of connections to stepper motors:

1) 4 pin bi-polar



In this connection type, you supply four pins: two pins for the a coil, and two pins for the b coil of your bipolar stepper motor.

Here's an example of the kind of controller you would use for this connection type: The Adafruit DRV8833 DC/Stepper Motor Driver Breakout Board.

2) Step and direction

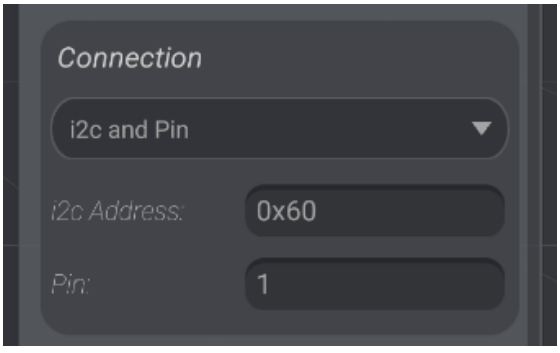
In this connection type, you supply two pins.

The first, is the step pin. This is the pin that will signal to the breakout board to step on a rising edge on that pin.

The second pin is the direction pin. This indicates to the breakout board if the step should be clockwise or counterclockwise. You can set whether a low signal on that pin should mean a clockwise or counterclockwise step. The setting here will depend on your exact board as well as the order you wired up the coils of your stepper motor. You may need to experiment with both settings.

Here's an example of the kind of controller you could use for this connection type: The EasyDriver - Stepper Motor Driver.

3) i2c and pin



In this connection type, you supply an i2c address (in hexadecimal or integer format) and pin, just like an i2c servo.

This connection type only works with one board: The Adafruit Motor Shield v2. Please see the included ReadMe.txt with the Arduino driver code to take the steps required to enable support for various libraries (such as the the library for the Adafruit motor shield v2).

IMPORTANT NOTE: Wiring up a stepper motor with the motor shield v2 is easy, but it comes with some drawbacks compared to the other two control types:

First, driving stepper motors requires very specific timing. However, the Adafruit Motor Shield v2 requires i2c communication, which can makes timing out step signals not as reliable. In general, controlling a stepper motor with the Adafruit Motor Shield v2 will not produce as responsive / timing accurate steps as the other two methods.

Second, the Adafruit Motor Shield v2 has built in overcurrent protection. If you are trying to draw too much current with a movement, some steps may be dropped. Make sure you're powering the board with an appropriate power supply that won't allow too much current to be drawn. As stepper motors are open loop control, there's no way for Bottango to know if steps are dropped, and you'll have to resynchronize the motor.

Configuring Stepper Motor Steps

You configure the steps values of a stepper motor in much the same way you configure the PWM of a servo motor:

On the right you enter the number of clockwise steps the stepper should take to move to it's maximum movement (1.0).

On the left you enter the number of counter clockwise steps the stepper should take to move to it's minimum movement (0.0). Note that this is always a negative number. You move positive steps from home clockwise, and negative steps from home counterclockwise.

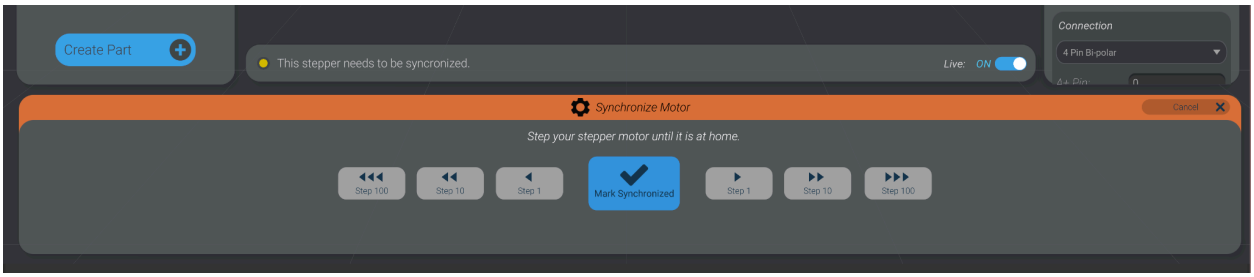
Finally, unlike a servo, a stepper's "home" is always 0 steps. This is the state the stepper motor should be in when it is synchronized, as we'll discuss in the next section.

Synchronizing a Stepper Motor

The biggest difference between stepper motors and servos is that stepper motors are "open loop" control, whereas servos are "closed loop" control. This means that while servos always know where they are, and can accurately go to a position indicated, stepper motors need to be brought to a known location first, and then move from there.

When you set a stepper motor to live, unlike a servo which goes to a green "ready state", a stepper motor will go to a yellow "needs work" state, and show that it needs to be synchronized.

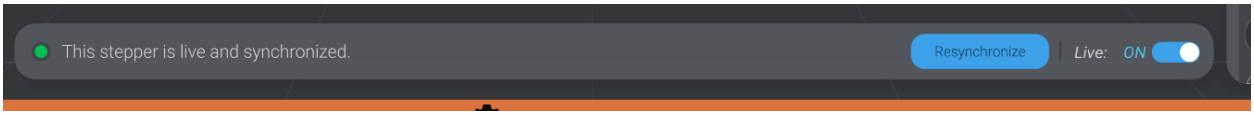
Click the Synchronize button to begin synchronizing the selected stepper motor.



While synchronizing, you have the option to manually step the selected stepper motor 1, 10, or 100 steps in either a counterclockwise or clockwise direction. Your goal is to move the stepper such that it is in the "home" position, that is the position

Home position is the value that the stepper will be in "at rest." If you think back to when we created structure, we did so in its home position and rotation. When Bottango moves and rotates joints, it offsets the joint away from its home position and rotation, and then returns back to that home position and rotation when at rest. When you are synchronizing a stepper, you need to move the stepper manually to that same position. All movements once synchronized will be from this starting point.

Press the blue "Mark Synchronized" button to mark the stepper synchronized and finish setting it live.



Resynchronizing a Stepper Motor

If for any reason, the stepper motor loses synchronization with what's in Bottango, you can click the "resynchronize" button. This will mark the stepper as not synchronized, and allow you to manually synchronize the stepper motor again.

As well, if for any reason a stepper motor goes not live after being synchronized, it will require resynchronization before it can fully go live again.

Future Synchronization Options

In the future, I plan to support more advanced synchronization options. As an example, it is common to synchronize a stepper motor with a limit switch. The motor moves until the limit switch is hit, and then the motor is marked as synchronized as its now in a known position. I hope to add support for this workflow in an upcoming version.

-14-
Importing 3D Models

What's in this chapter

You are not restricted to using the rudimentary 3D modeling tools Bottango provides to build the structure of your robot. If you modeled your robot in a 3D modeling or CAM program, this chapter will show you how you can create the structures of your robot using imported models instead.

Bottango currently supports models in the .OBJ and .FBX format. Bottango can import both the models and textures / materials of a model. A workflow to import animations from FBX files is planned, but not complete yet.

Import a model

1. Click "Import" in the mode tabs section to switch to import mode.

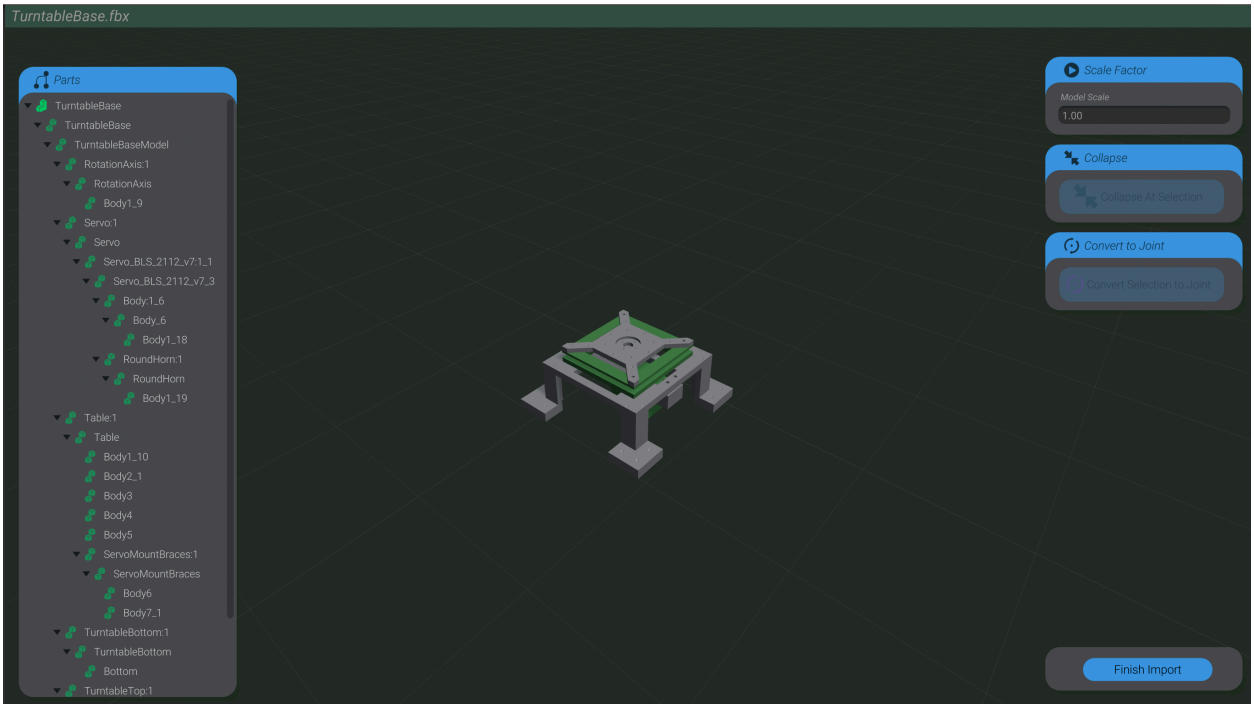
2. Click "Import Model" to begin importing a new 3D model.

3. Select the 3D model file you want to import.

Imported models follow the same file path rules as audio tracks. See the previous chapter for how absolute, relative paths, etc. are decided.

Immediately after importing a model, you are taken to edit its import settings. The edit imported model screen looks a lot like the basic building screen, but has a few key changes:

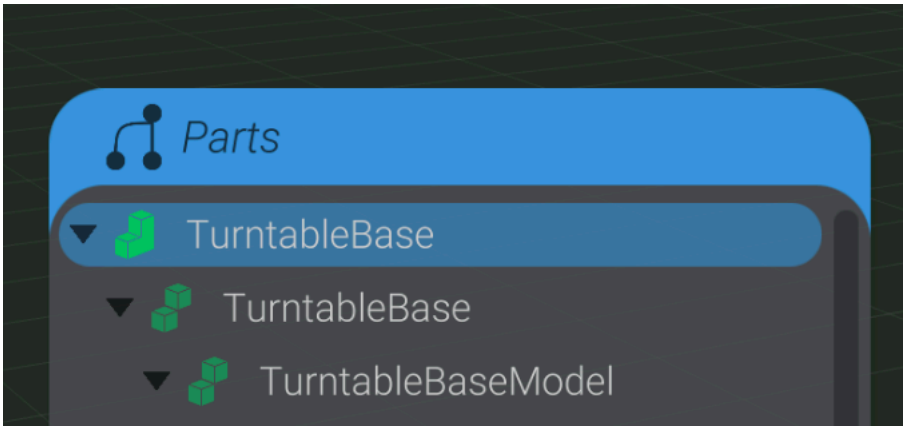
- Only the imported model is shown and selectable.
- Only the tools needed to allow you to set the import settings for an imported model are shown.
- The screen has a slight green tint to remind you you are in edit model mode.



In the most basic workflow for using 3D models, you would use a 3D model in a 1:1 replacement of a structure primitive (cube, sphere, etc.). You can see in the above example that I have imported a turntable base.

The imported 3D model is made of many individual parts, but if I just want it to act like a simple piece of structure, I really just want all of those imported pieces to act as one.

1. Select the root, top-level piece of the imported model in the part's list.

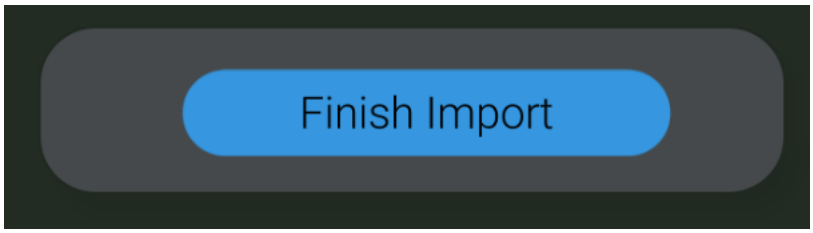


2 Click to collapse the model into one single part. All children pieces of the model will be imported and used, but treated as just part of the single, collapsed part.

As you can see, all the sub pieces of the model are now collapsed into a single part.

Create an instance of the model in your project

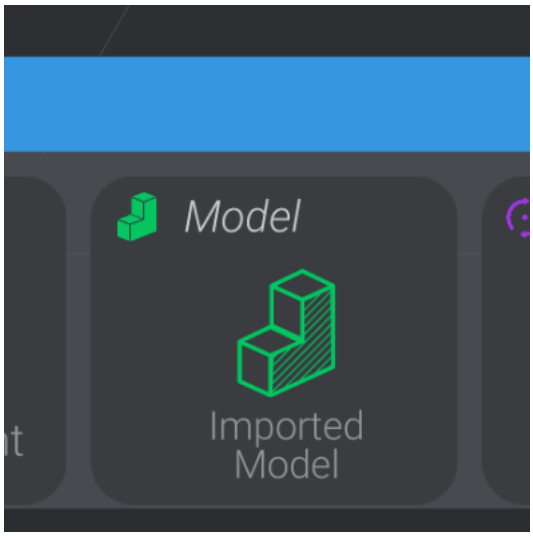
1 Click Finish Import to finish configuring the imported model.



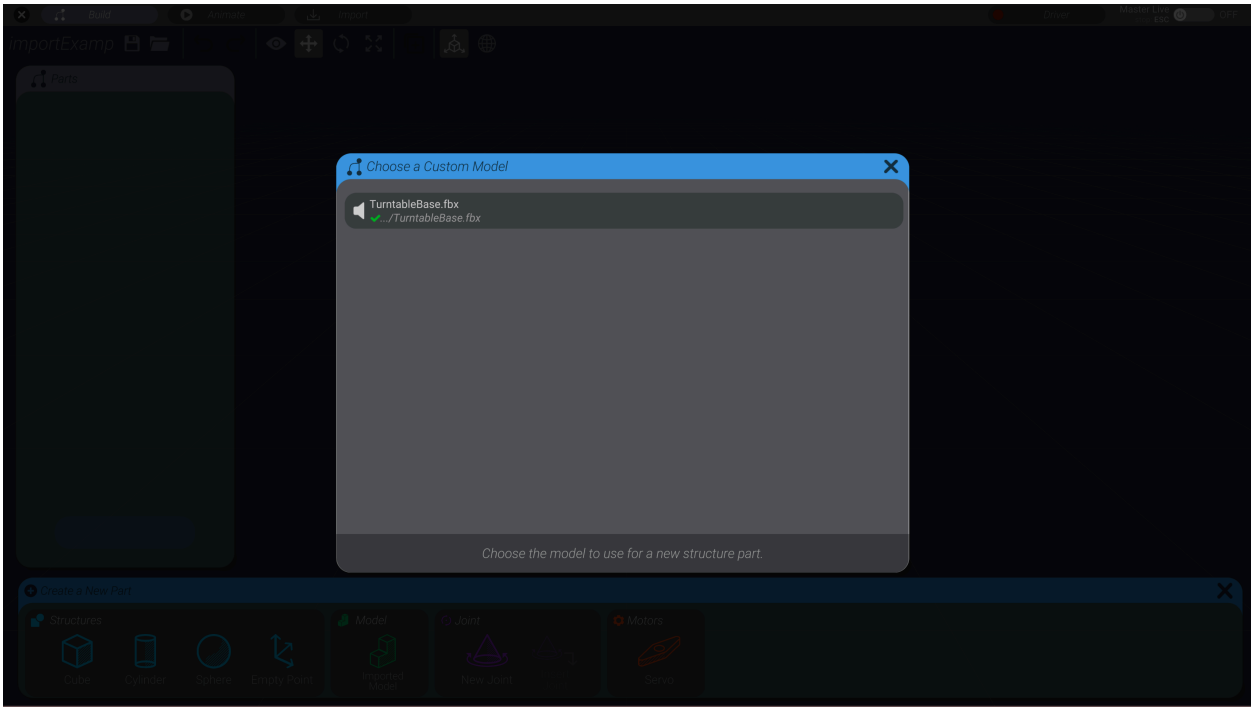
2 Return to “Build Mode.”

3 Click Create Part.

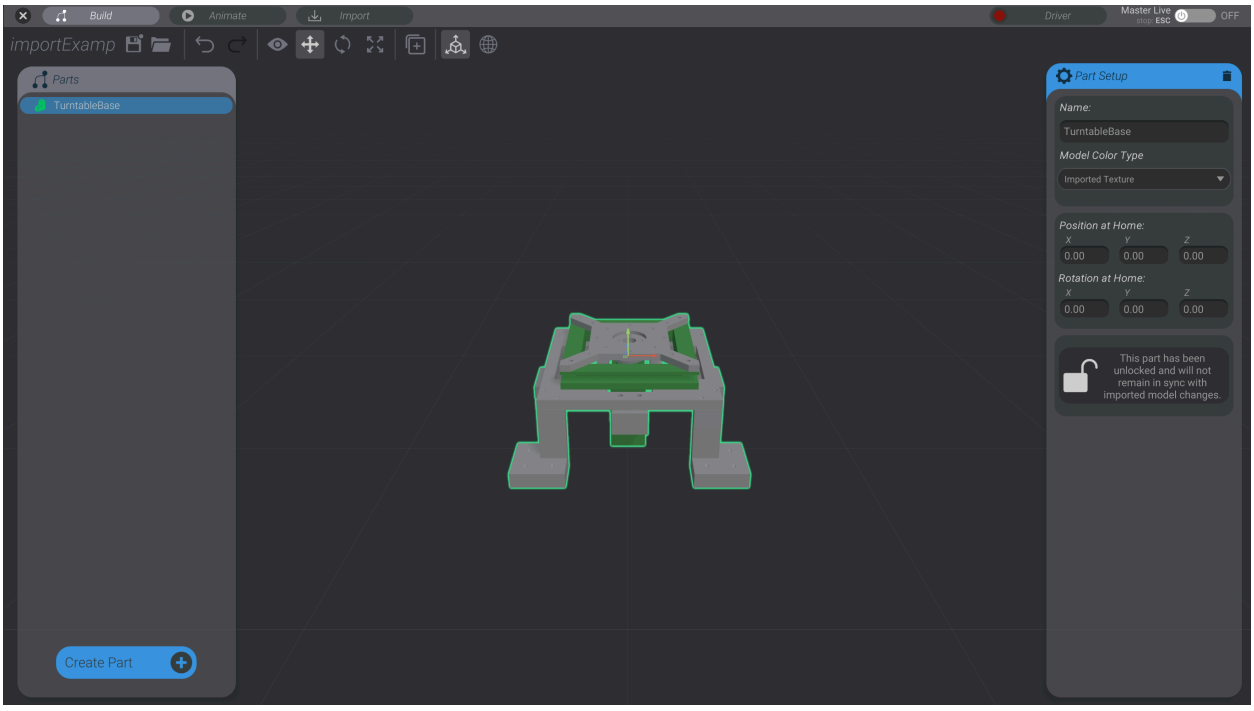
4 Select Imported Model.



5 You will be shown a list of all imported models in the project. Select the model you just imported.



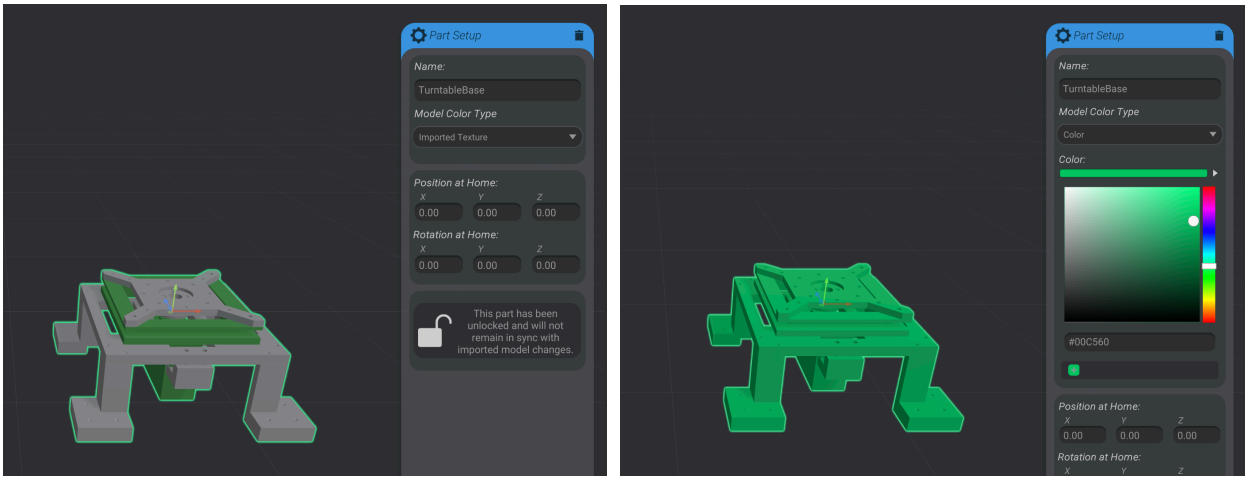
You will now have an instance of the model in your your project:



With a few exceptions that we'll get into later, imported models behave exactly like structures.

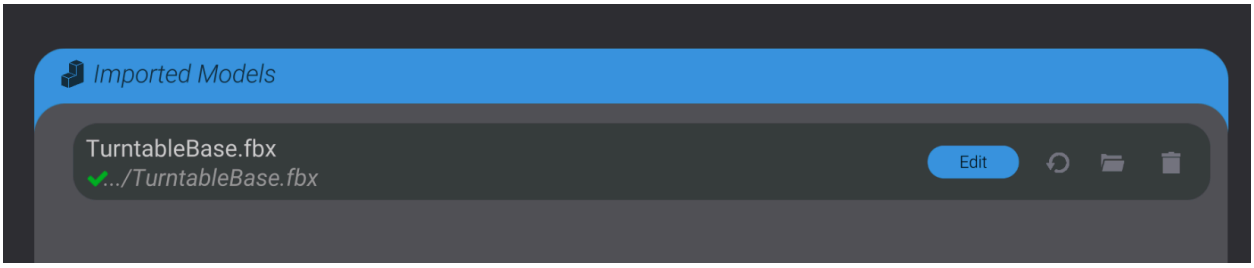
Model texture and color

By default, imported models use the imported texture that was part of the model file. You can select to change between using the imported texture and using a custom color:



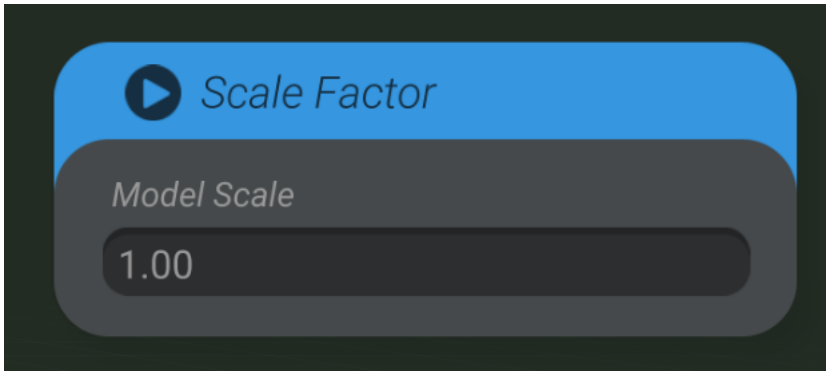
Changing model import settings

At any point, you can edit the import settings on a model. Return to import mode and click the "Edit" button to change the import settings on a model.



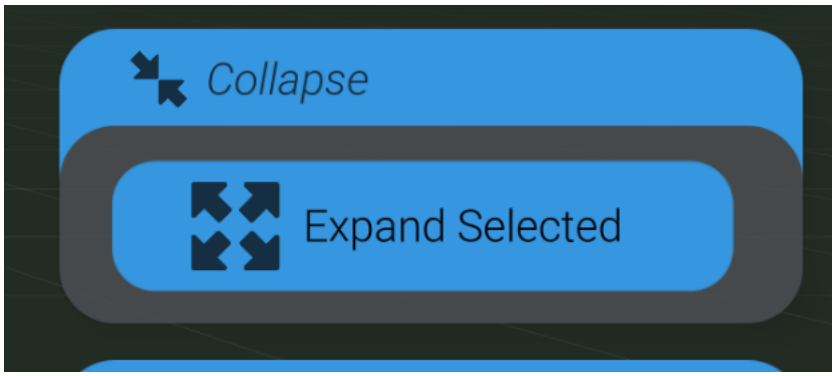
Model scale

It is difficult to have a consistent, single scale across all 3D modeling programs. To help compensate, you can adjust the import scale of an imported 3D model.



Expand a collapsed model piece

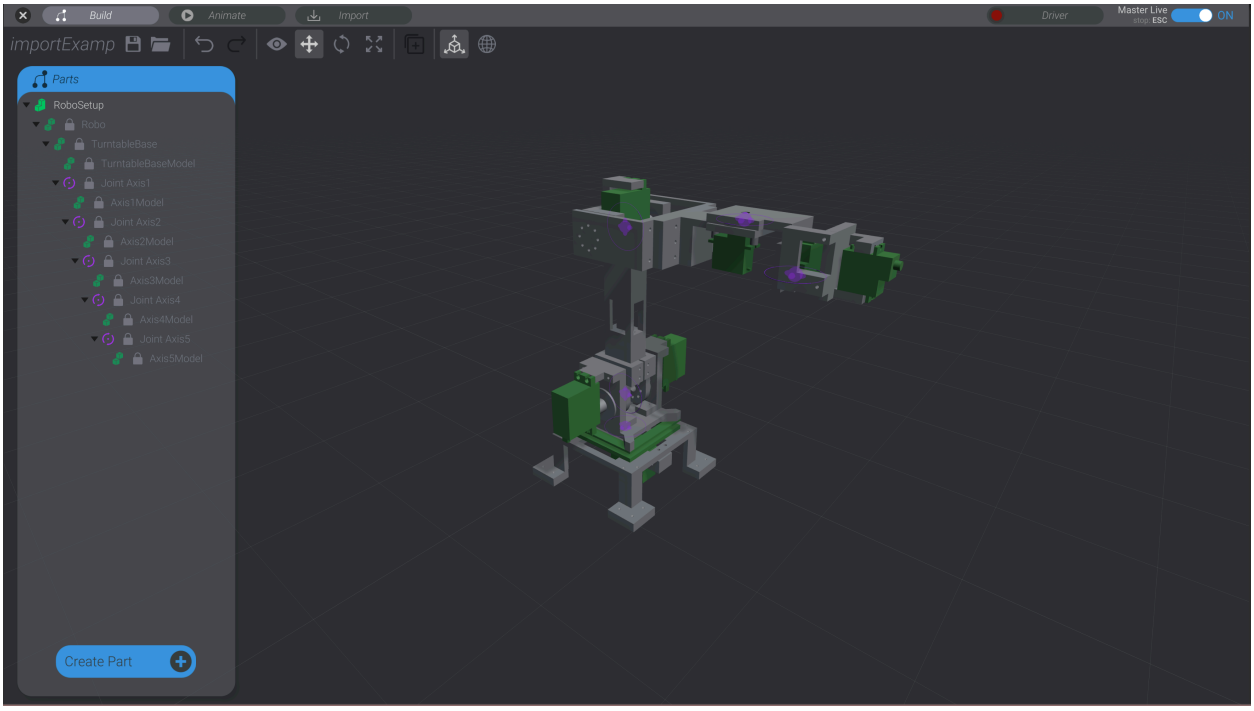
If you want to remove a collapse point while editing an imported model, select the collapse point. You will be able to select "Expand Selected" to remove the collapse.



Importing an assembly of multiple parts

As useful as it can be to import a model and use it as a piece of structure, it can be even more useful to import a fully composed robot and set up the assembled pieces in place from the imported model.

Here, for example, is a fully assembled 3D model that has been imported and configured to be used as the structure and joints in a Bottango project:



What's more, Bottango can keep the imported model in sync with changes to the model file. As parts of the model file move, are added, or are deleted, the pieces in Bottango will do the same.

However, you will need to construct your 3D model in such a way that it works with the logic of Bottango.

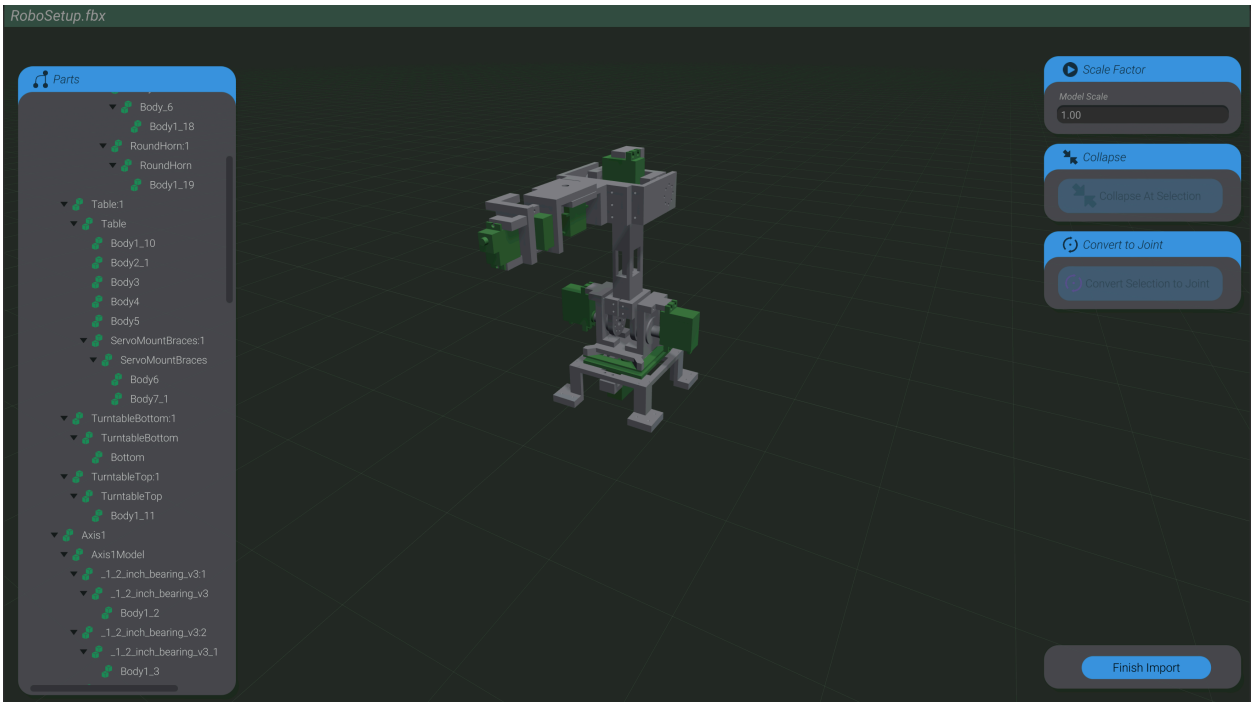
To make a 3D model file work easily with Bottango as a full assembly, you should do the following:

- Make sure the parent-child relationships are accurate. Once imported, you cannot reparent or change the hierarchy of an imported model.
- Have empty points as parents, in the correct pivot location and rotations, that you want to convert to joints. You can see that in the above model, 5 joints exist. In the import settings (as shown later), we have converted empty points that are in the correct rotation and position into joints.
- Optionally, have the visual model sections of the model parented in a way that makes for convenient collapse points.

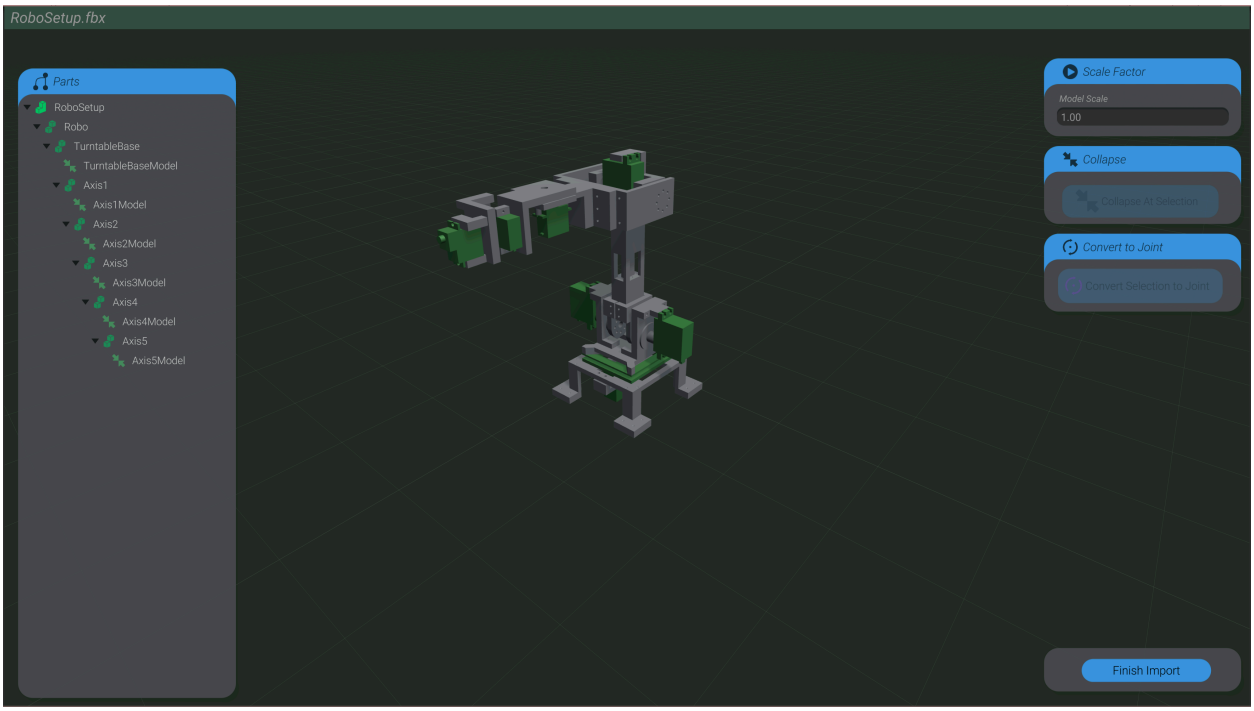
Let's go about creating an assembly of parts from a single model.

Converting points to joints

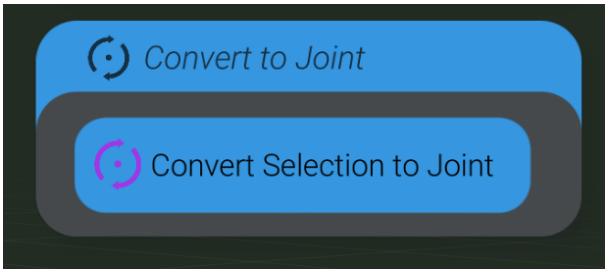
Here, for example, is a complicated model that has been constructed to work well with Bottango.

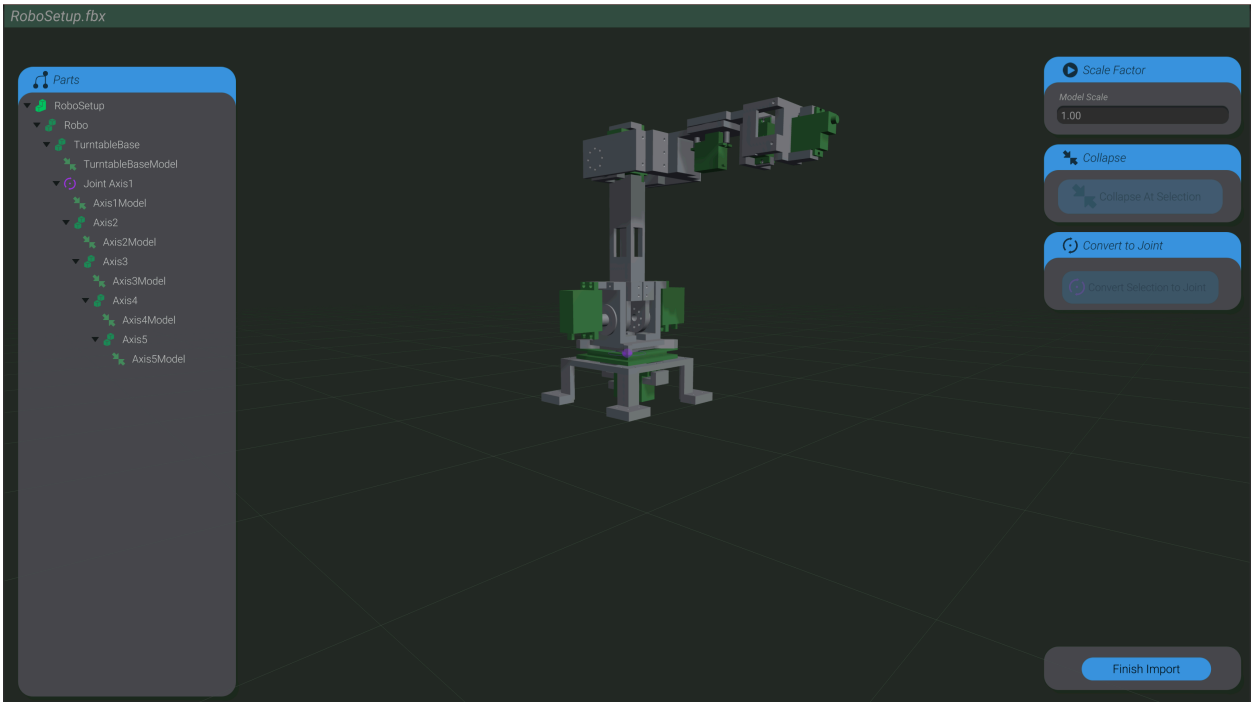


The first thing to notice is the many separate pieces that make up the model, most of which could be collapsed. When the model was created, the author anticipated collapsing so it was clear where to collapse pieces into into single parts. Furthermore, the model was created with a hierarchy that makes sense for animating the robot.

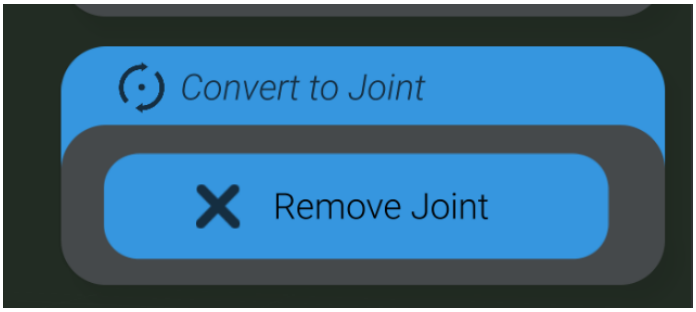


Finally, the parent of each collapsed part is placed and rotated so that it can behave as the pivot of a joint. Selecting a piece of the model and clicking "Convert To Joint" will convert this piece of the model to a joint.

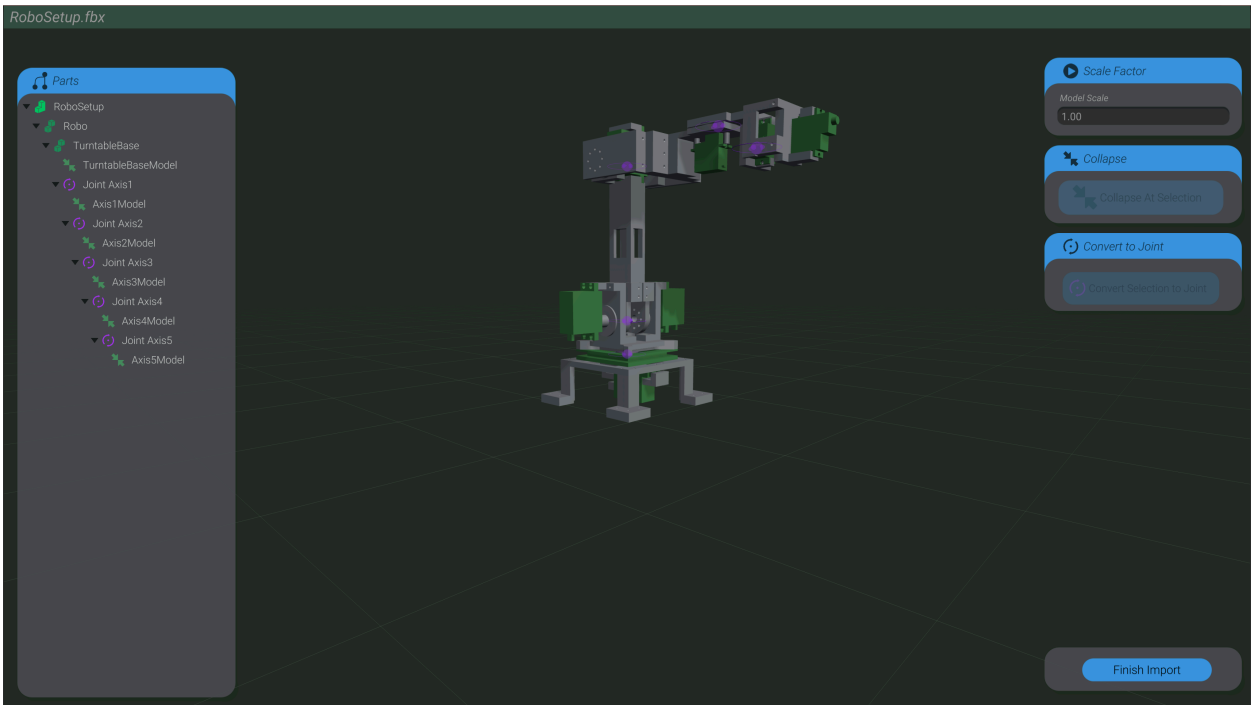




You can also select a converted joint and remove the conversion by clicking "Remove Joint."



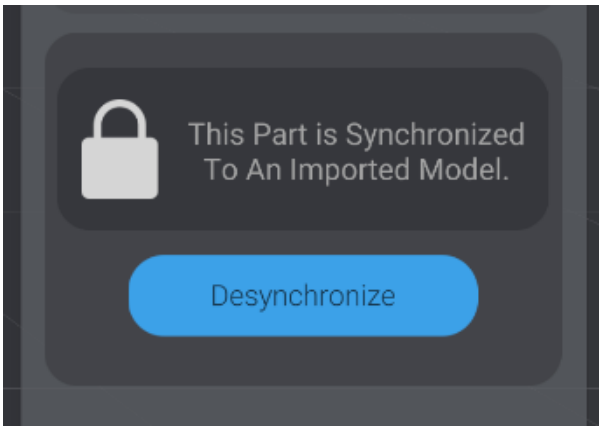
Here I have converted all the required pieces of the model into joints:



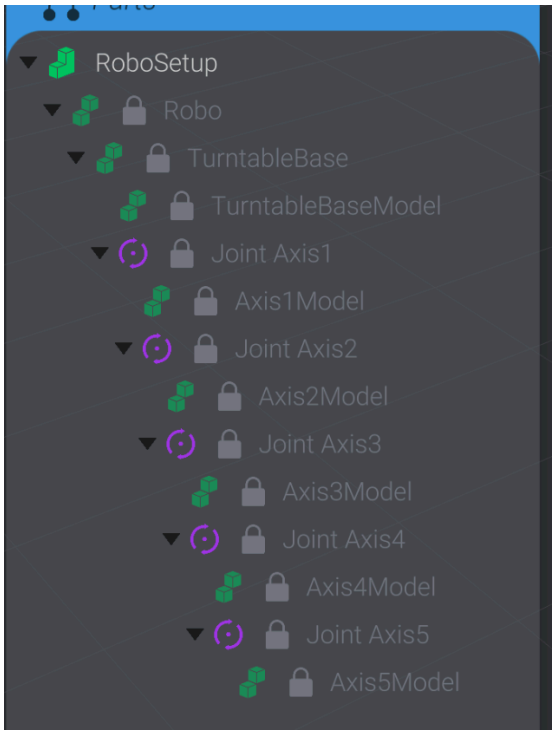
Joints created this way can't set their axes or offsets while importing the model. Instead, create a new instance of the model in your project via the "Create Part" menu, and then you can configure and use a joint created this way like any other.

Keeping imported models synchronized

When you create an instance of a model in your project, it is "synchronized" to the imported file. If you were to move, rotate, delete, or add elements in your imported file, the same changes happen in the instance of the model. Synchronized models (the default, starting state) show they are locked when you select any part of them:



Synchronized parts are shown in the parts list with a lock and a dim state:



However, keeping things in sync comes at a cost. You cannot change the following attributes of a synchronized model in Bottango:

- Name
- Home position
- Home rotation
- Hierarchy

As well you cannot delete or duplicate any child part of an imported model besides the top level root

You *can*, however, add drag parts to be the child of a synchronized, imported model (for example, making a piece of a model a parent of a motor).

If you want to make these kinds of changes to a model and keep instances in Bottango synchronized, you should make those changes in the 3D modeling program you used to create the model in the first place. That way, Bottango can always keep the instances of the model in sync with the latest version of the file.

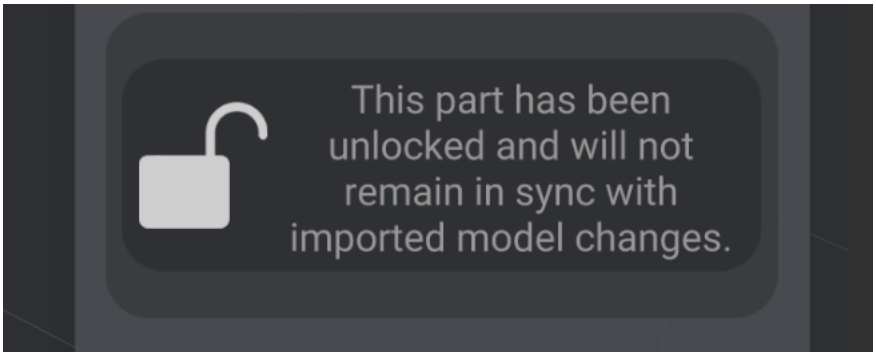
Just be aware that Bottango relies on the name and path of a model to build it out in the program. For example, if you rename"Root/Axis1/MyCube" to "Root/Axis1/MySquare," Bottango will act as if MyCube was deleted and MySquare was added.

Desynchronizing a model

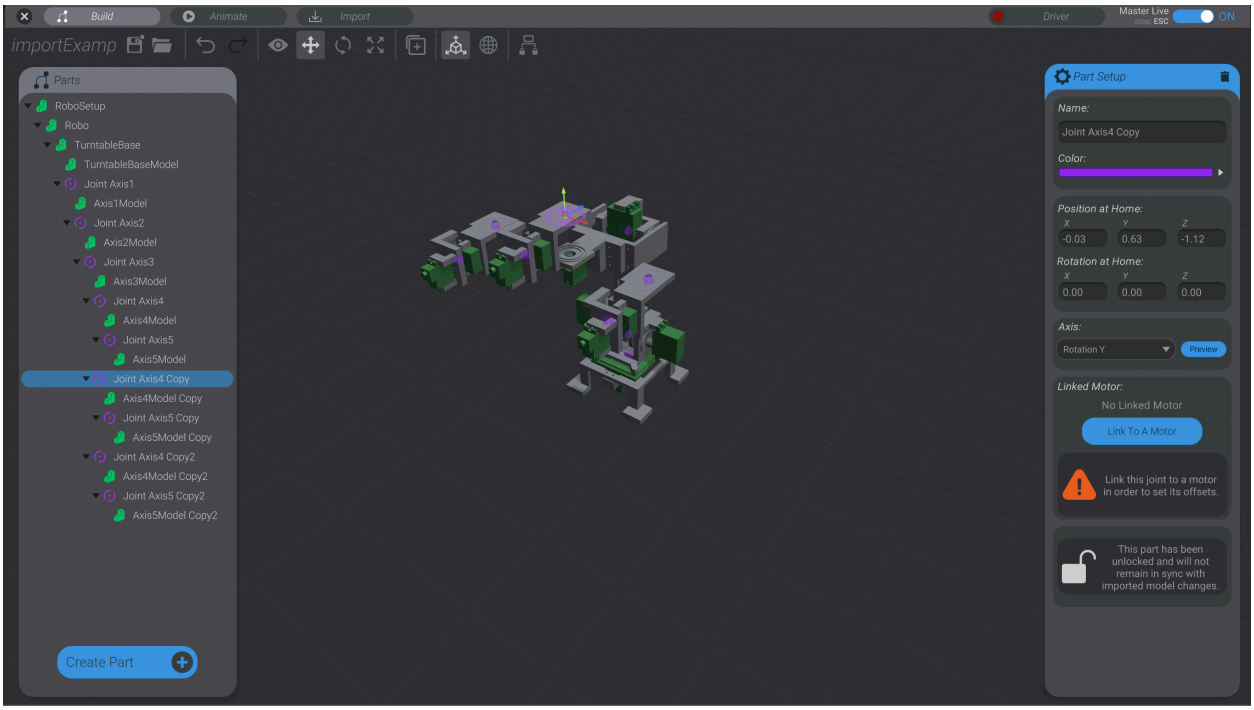
You can choose to desynchronize a model with the file. This means that you are able to edit all the parts in all the ways that were prevented above. However, it also means that this instance will no longer try to

stay in sync with the file as it changes. This is a permanent action; once you desynchronize and make changes, you can't resynchronize.

You can either select a piece of the model and click Desynchronize, or attempting to make a change to a synchronized model that would require desynchronization.



Here, for example, I have desynchronized and then duplicated and moved pieces of the model to modify it:



-15-
Custom Events

What's in this chapter

There are times you may want to control hardware with Bottango that is not one of the built in effector types. Maybe you have your own motor that Bottango doesn't support yet, or maybe you want to turn on or off a light at certain points in an animation, etc.

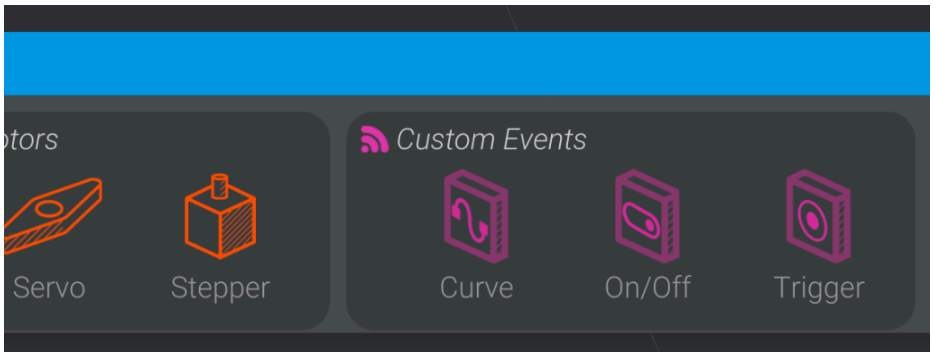
For any hardware effector that isn't one of the yet supported motors, you can use custom events. Custom events animate along the animation timeline just like motors and joints. However, they are not used as motors, and have a few additional signal types available.

Custom events require you to modify microcontroller code

Inherent to using a custom event, you're going to need to write some of your own code to handle what to do when the custom events fire. In the Bottango Arduino code, there's documentation of where and how custom events are fired, but what you do when that happens is entirely in your hands. Read the Bottango Arduino examples and documentation for more details on handling custom events on the microcontroller side.

Creating custom events

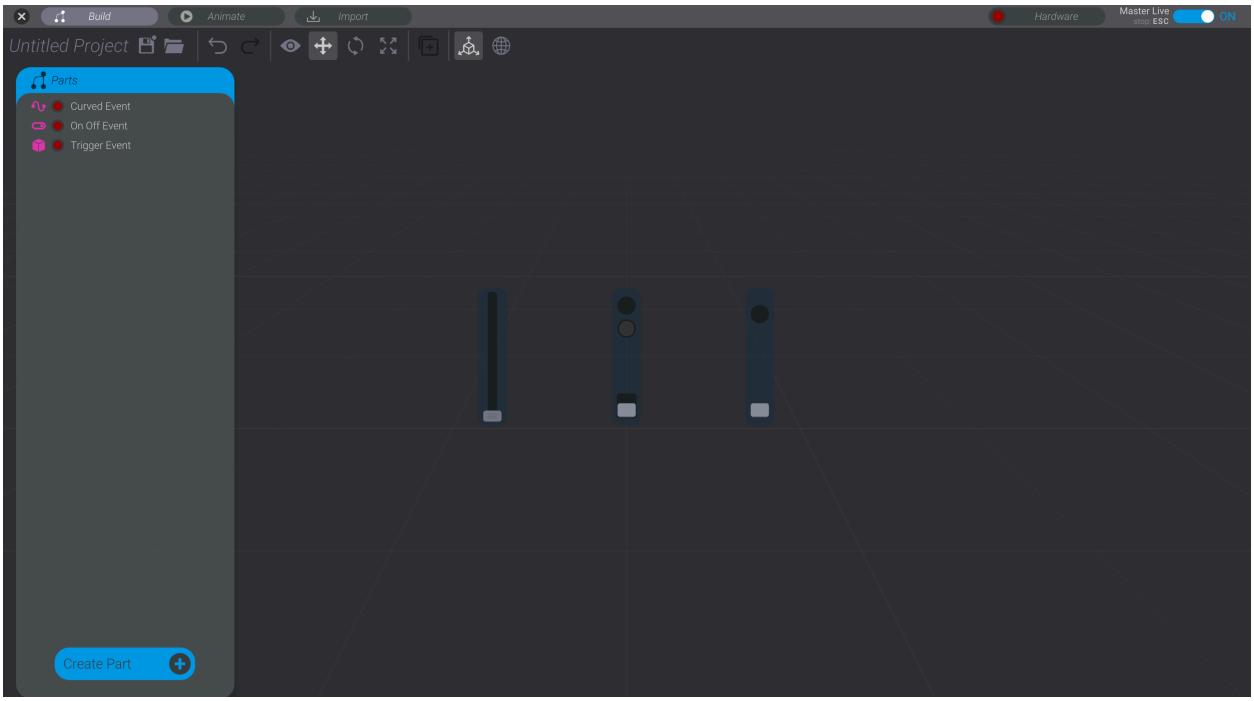
You create custom events in the Create Part menu, just like other parts. When you create a custom event, you can see that there are three to choose from:



- **Curved custom events**
  - Curved custom events are animated on the animation timeline with movement values, just like motors and joints. A curved custom event sends a value between a movement of ⊘0.0 and ⊘1.0. An example when you might use a curved custom event is dimming a light or adjusting the speed of a continuously rotating motor.
- **On off custom events**

  - On off custom events send an on or off signal, instead of a curved movement between 0.0 and 1.0. An example of when you might use an on off event is turning a light on or off, or turning a continuously rotating motor on or off.
- **Trigger custom events**
  - Trigger custom events fire on each keyframe for that event in the animation. They don't have any other data when the keyframe is expressed, just that a keyframe is at this moment. An example of when you might use a trigger custom event is playing a sound using your microcontroller, or releasing confetti from a closed container.

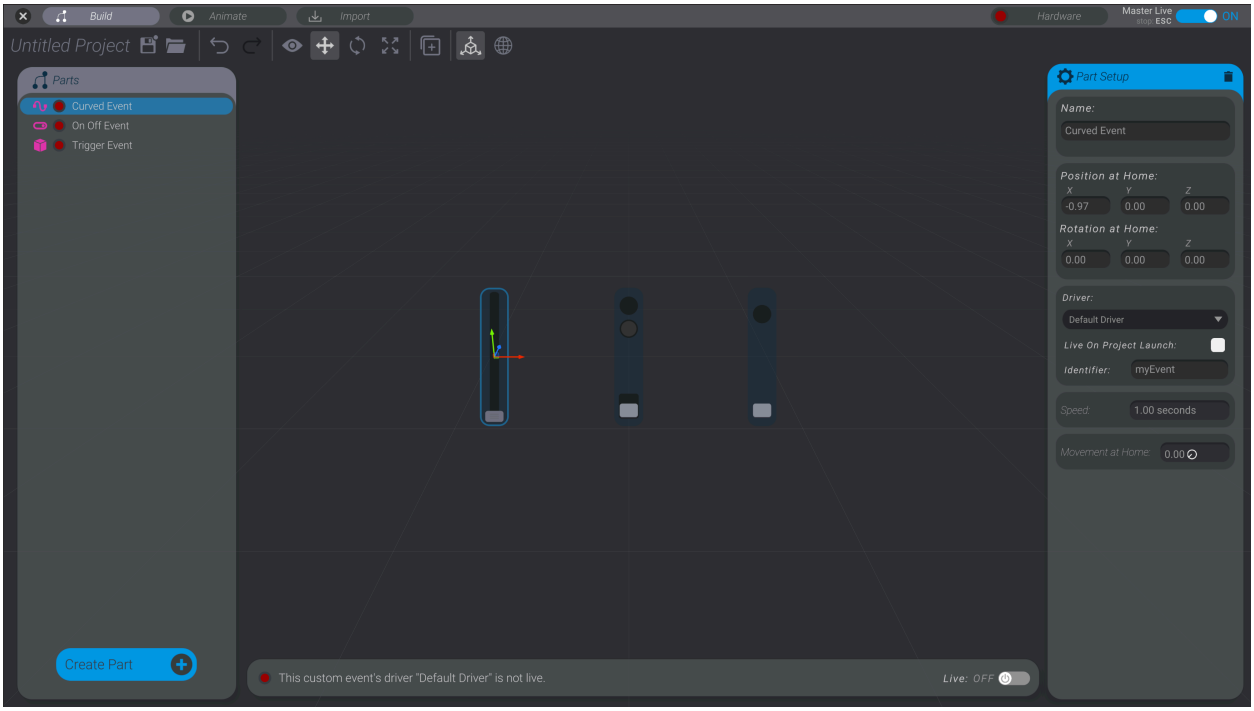
When you create a custom event, you'll see an interface for the custom event is created in the 3d scene:



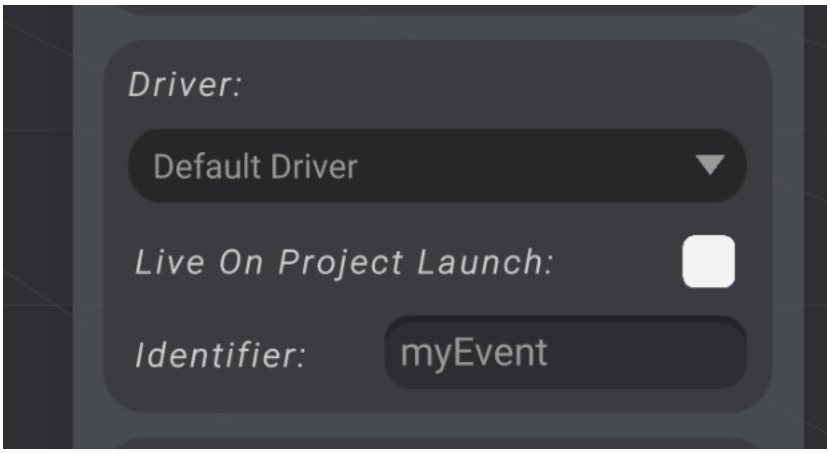
From left to right in the above screenshot is a slider that controls a curved custom event, a switch that controls an on / off custom event, and a button that controls a trigger custom event.

Configuring custom events

You can click on a custom event to select it, just like any other part. When it is selected, you'll be able to configure the custom event.



Custom events have an associated driver, just like motors. You can select the driver to use for the custom event in the same dropdown as a motor.



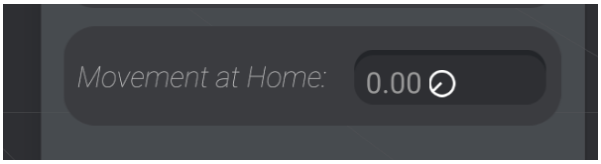
As well, you can set a custom event live / not live, and control if the custom event is live on project launch, in the same way as a motor.

However, custom events have one additional modifiable property: an identifier. All effectors in Bottango, including Motors, have an identifier. The identifier of an effector is how Bottango knows which motor is targeted in an animation curve. With motors, the identifier is automatically created based on the pins/

connection of that motor. With custom events, you set the identifier to whatever you want. However, each event controlled by a hardware controller must have a unique identifier.

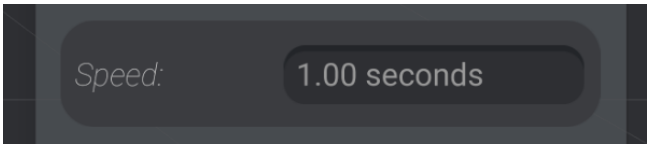
Identifiers are text based strings, and can have at most 8 characters. When you set the identifier of a custom event in Bottango, this is the name of the effector that will be sent to the microcontroller when it is animating.

As well, curved events and on off events let you set their home state:



A curved event's "movement at home" is where the curved event will return to when not being animated. An on off events "on at home" is if the event is on or off when not being animated.

Finally, curved events allow you to set the maximum speed.



The speed you set for a curved custom event is the maximum speed to travel from 0.0 to 1.0 while animating. For example, if you input 2.5 second, your custom event will animate from 0.0 to 1.0 in 2.5 seconds at the fastest.

Trying out events

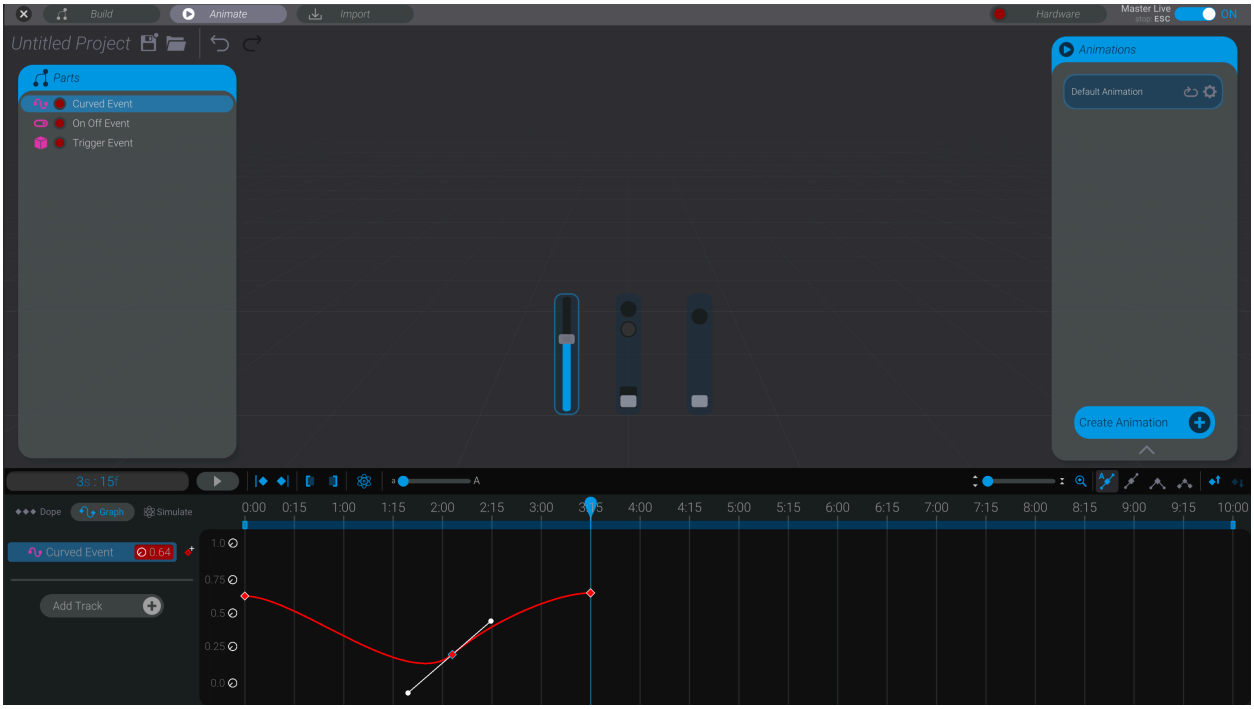
While in configuration mode, you can drive custom events to test them. If they are live, and their associated driver is live, events will be sent. To try out the event, you:

- Drag the slider of a curved event. It will return to home when you release the mouse.
- Press on the switch of an on off event. It will return to the home state when you release the mouse.
- Click on the button to send a trigger.

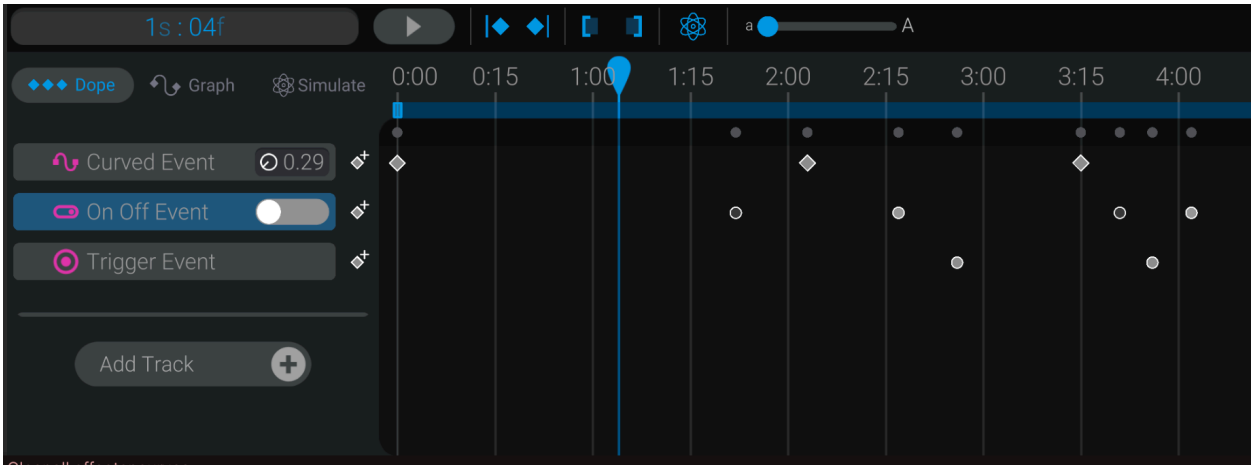
Animating custom events.

Custom events are animatable just like motors and joints. To add a custom event to an animation in animation mode, you can either use the "add track" button, or just start manipulating the custom event (it will automatically add itself to the animation).

For example, here I have dragged the slider of the curved event, and it keyframes itself on the animation in the same way that moving a joint updates and add keyframes for motors:



You can do the same thing with on/off events and trigger events. Press the switch to change the state of an on off event, or press the button to add a trigger. Because they do not have a curve, on off and trigger events will not show up in the graph editor. You can only see on off event keyframes and trigger keyframes in the dope view:



On off and trigger keyframes are shown as circles, instead of diamonds. As well, You can tell if an on off keyframe represents on or off by it's color. Darker on off keyframes represent off, and the lighter ones represent on.

You can drag, copy / paste, and manipulate custom event keyframes the same way you can other keyframes in Bottango.

-16-
Custom Motors

What's in this chapter

Bottango is a work in progress, and more supported effectors and motors are a top priority. In the meanwhile, Bottango provides a workflow for adding support yourself for a motor type that doesn't have official support yet.

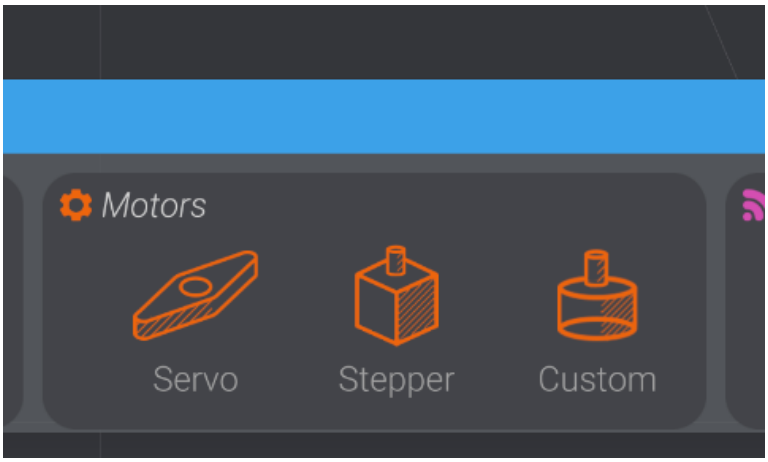
Custom motors allow you to drive and control motors that aren't yet built in to Bottango.

Custom motors require you to modify microcontroller code

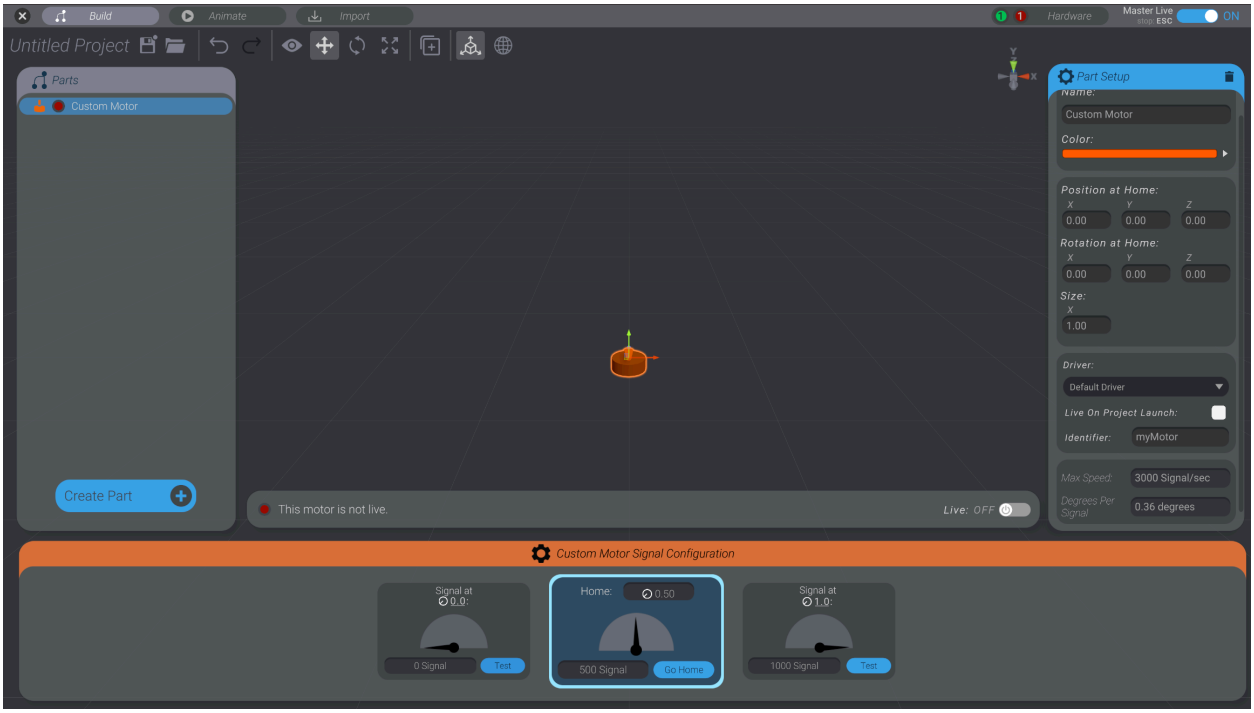
Inherent to using a custom motor, you're going to need to write some of your own code to handle what to do when the custom motor is registered, deregistered, and driven. In the Bottango Arduino code, there's documentation of where and how custom motors are controlled by the microcontroller. The exact details of how your motor responds to difference lifecycle events will be in your hands.

Creating and configuring a custom motor

Create a custom motor using the create panel:



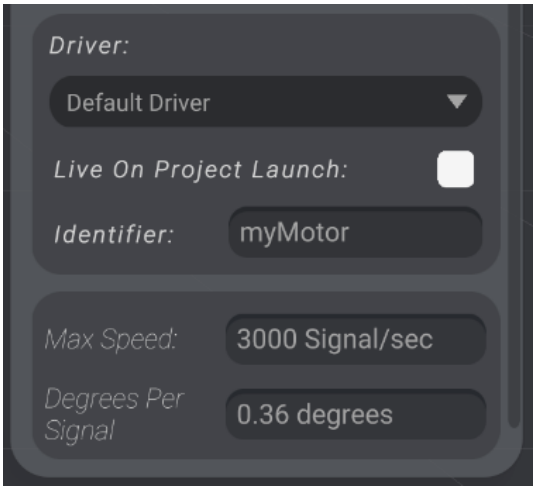
Then select the custom motor to configure it:



A custom motor can be attached to a joint just like a built in motor (servo, stepper, etc.) As well, custom motors have minimum and maximum signal you can define, just like a servo's minimum and maximum PWM, etc.

The biggest difference in configuring a custom motor require you to enter a unique identifier. With built in motors, the identifier is automatically created based on the pins/connection of that motor. With custom motors, you set the identifier to whatever you want.

Identifiers are text based strings, and can have at most 8 characters. When you set the identifier of a custom motor in Bottango, this is the name of the motor that will be sent to the microcontroller when it is animating.



As well, you can set the maximum speed that the custom motor will be driven at. In the above example, this motor will not move more than 3000 of the user defined signal units per second when animating.

Finally, you can set the degrees that the 3d representation of the motor will rotate per signal change. This does not affect the real world hardware driving of the motor, but is just helpful to keep the 3d representation and the real world motor in visual sync.

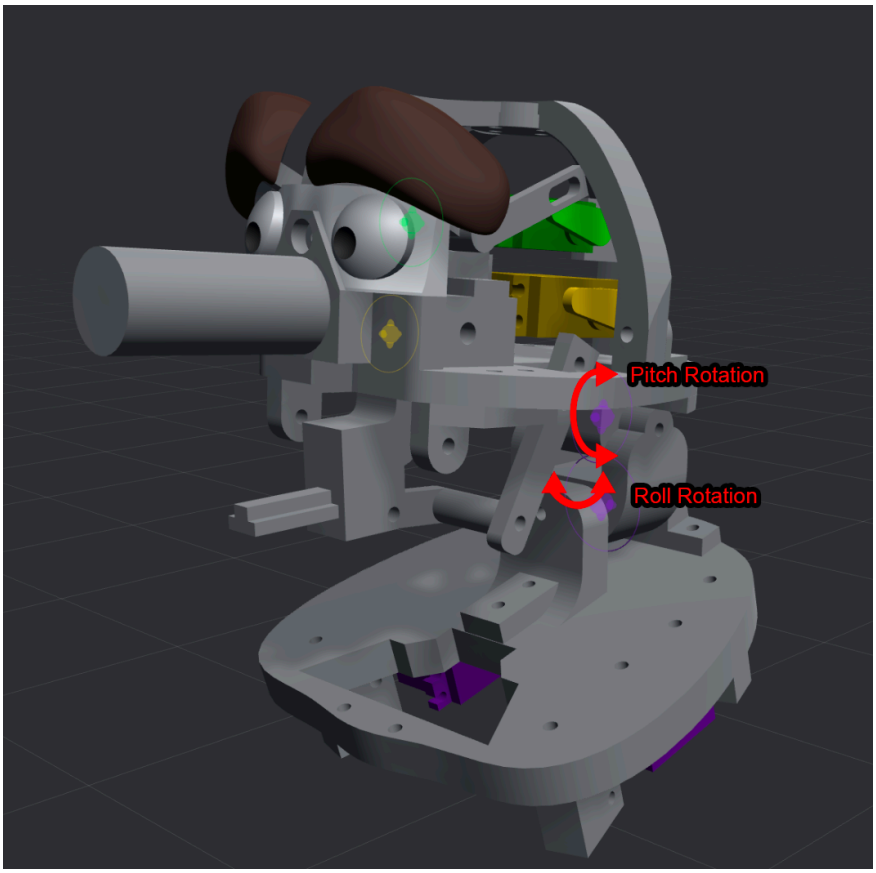
-17-
Advanced Mechanisms by Blending Target Poses

What's in this chapter

All of the joints and associated motors in Bottango so far are fairly straightforward mechanisms. As a motor rotates, so too directly does the linked joint in a 1:1 ratio. And for a lot of mechanisms, that's more than enough!

What if you want to keep multiple motors in lock step with each other, such as for example the sets of motors that control a pair of eye lids's movement, without having to animate each motor identically.

Or what if you have a more advanced mechanism, such as a neck on a u-joint, that tilts in rotates in different axis shared by multiple motors?



For advanced mechanisms, Target Poses and Pose Blending is the solution in Bottango. In this chapter we'll go over how to create Target Poses, and then how you can animate by blending between those target poses.

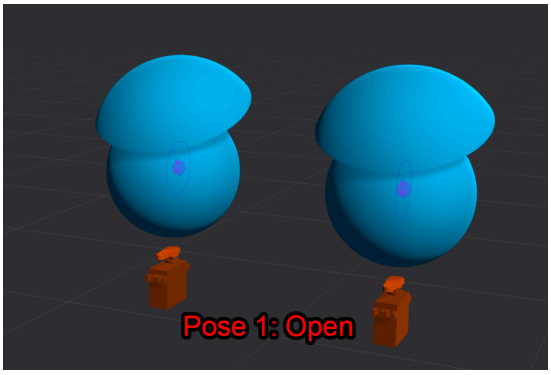
For the users that have invested in learning them, target poses are a clear favorite.

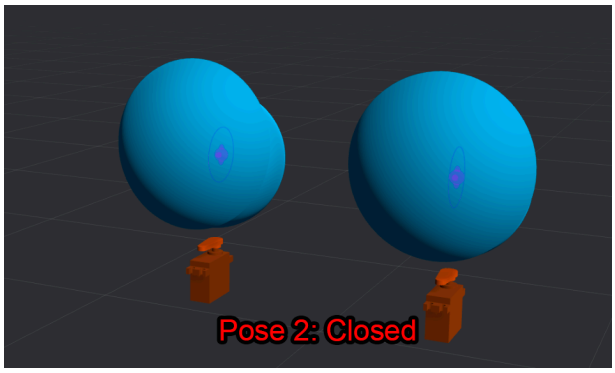
Target poses and pose blending may be the most powerful tool in the current Bottango suite of features.

What Target Poses should I make?

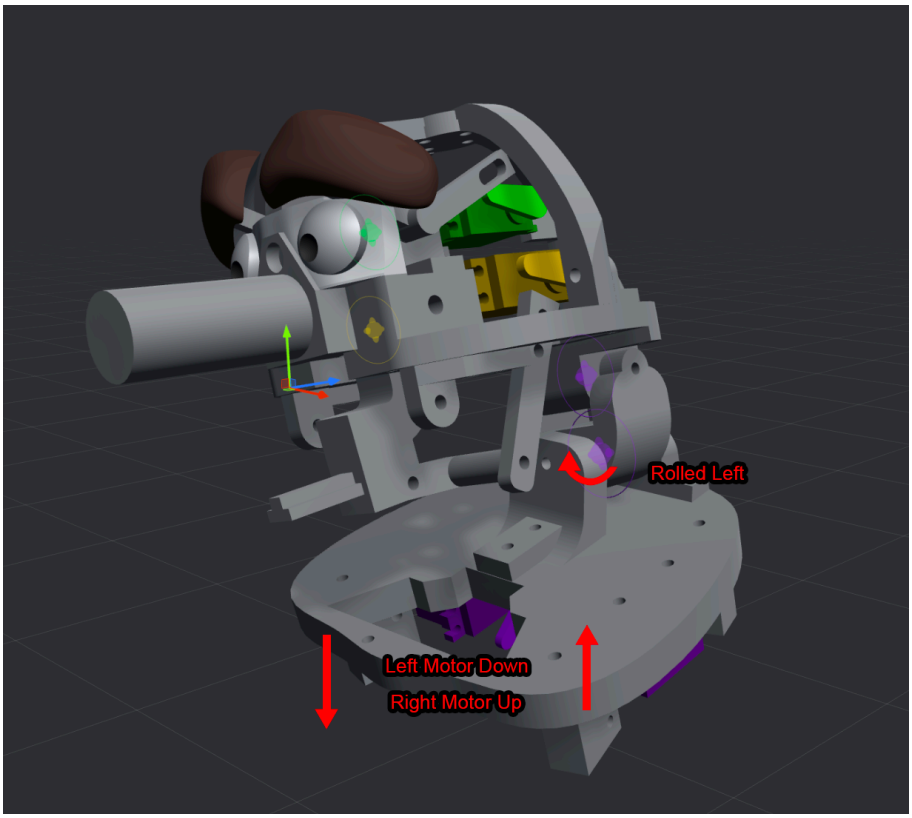
In order to blend between poses, you'll want to create and define the unique positions or poses your robot can be in for the mechanism or control you want to create.

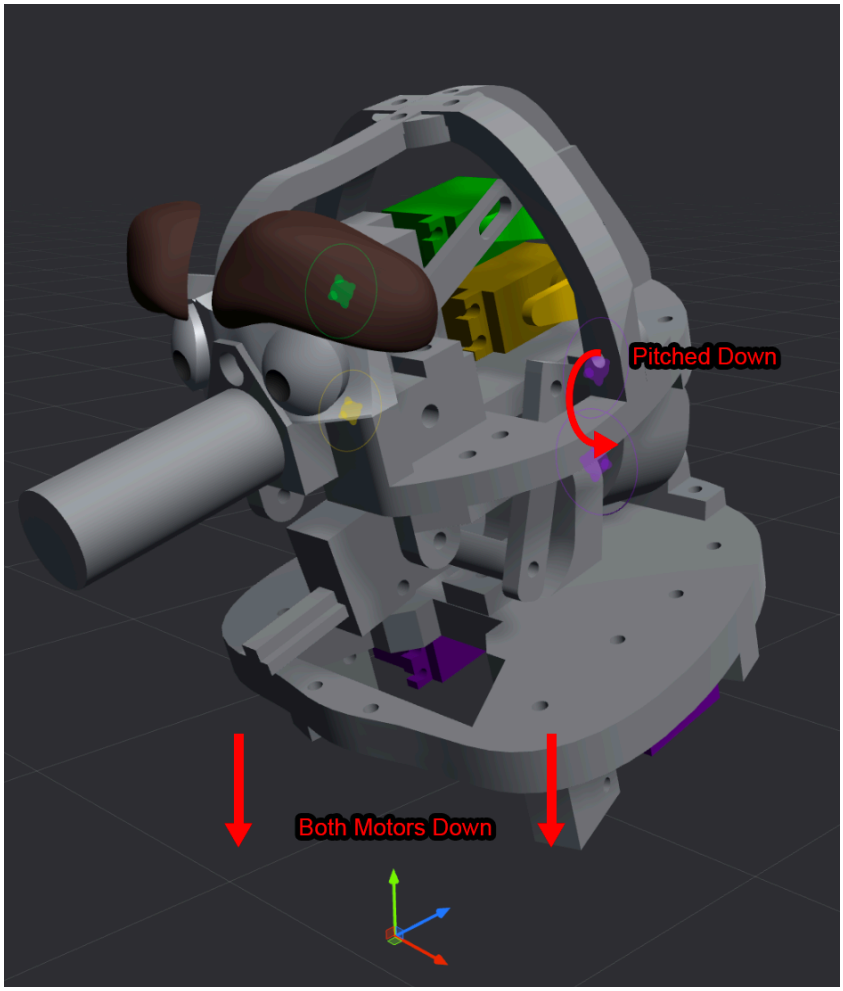
For example, let's say we wanted to animate a pair of eye-lids to always open and close together. Without pose blending, you'd have to animate each joint independently, and manually ensure that the animations for both eyelids match. In this case, we would need two poses: one pose for the eyelids fully closed and the associated joint and motor settings, and another for the eyelids fully opened. Once created, we'd blend between those two set up poses.



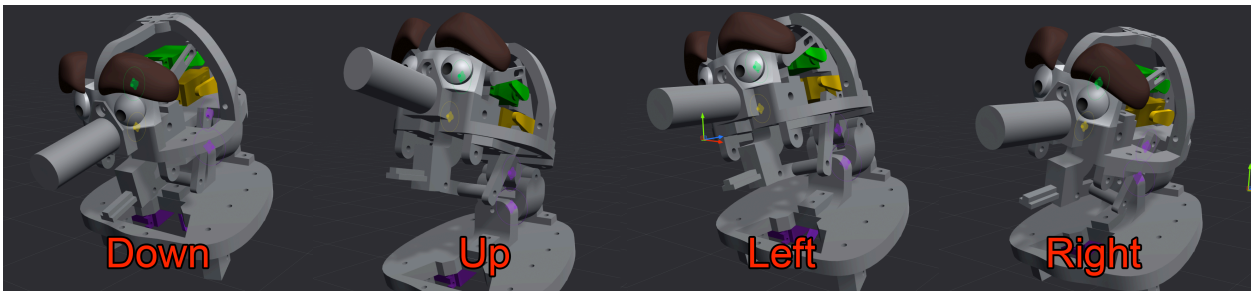


In a more complicated example, let's think about a neck that can rotate in two axis via a u-joint, with two motors pushing on either side of the head. In this mechanism, there isn't a single motor per axis, but instead each axis of the u-joint is partially influenced by both motors.





In this example, we'd want to create four total poses: one for looking up, one for looking down, one for looking left, and one for looking right. In each pose, we'd set up the joints and motor positions that achieve the desired head rotation.

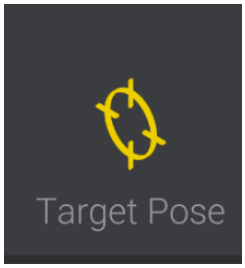


Creating and modifying target pose

When you create and edit a target pose, you move any joints, motors, events, etc. to the desired position and rotation for the pose you are editing.

One thing to consider is you may not always want joints and motors linked to each other in order to properly set up a target pose. Let's take the above neck example. If you were to link a neck joint to a motor, which motor would you pick. Neither motor is entirely responsible for the joint's axis. The answer is simple… don't link them! Create and configure the joints and motors range of signal and motion independently, and then set them to the desired positions instead via target poses.

1. Create a new target pose from the create part menu

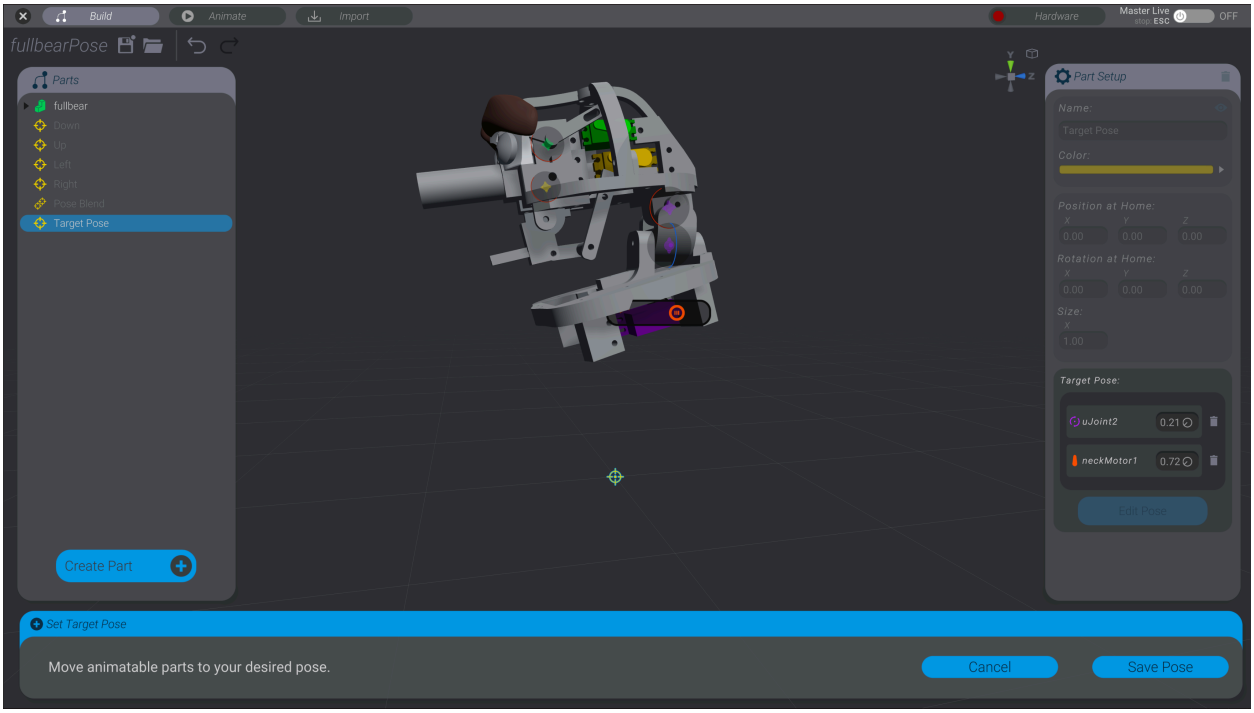


2. Click "Edit Pose"

3. Move any motor, joint, or custom event to the desired position or rotation.

If you have any motors that are not linked to a joint, you'll see a slider to adjust it's movement independently without needing a joint.

You can as well type in movement values directly once a part has been added to the target pose.



4 Save the target pose

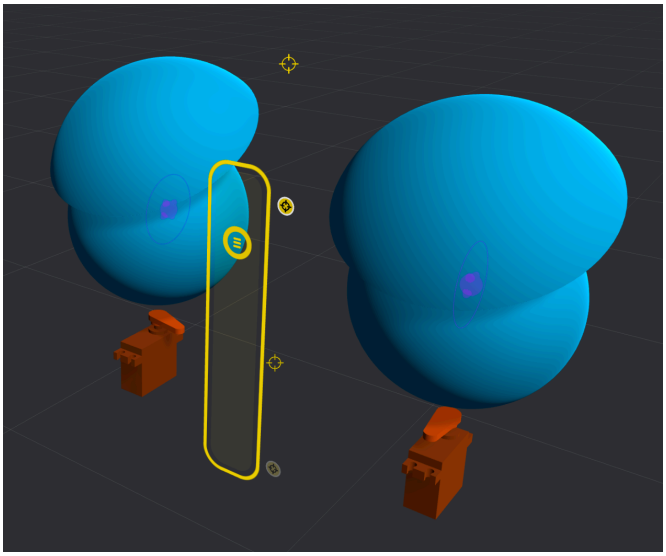
If you want to remove a part from a target pose, you can click the delete button in the list of parts to remove it from the target pose.

You can also edit the target pose to return to setting it up.

Two kinds of Pose Blends

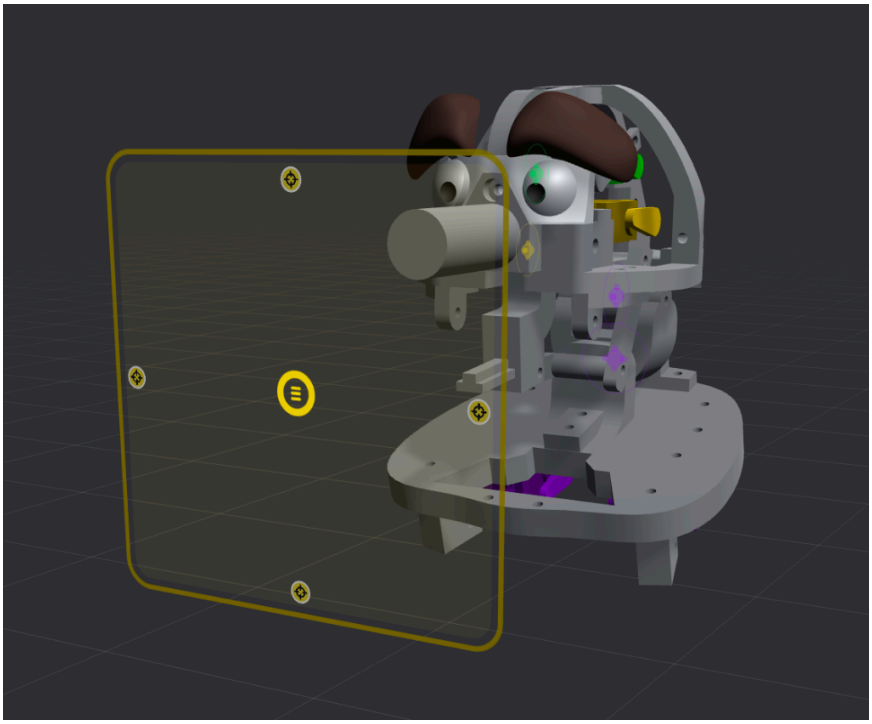
Pose Blends are the animation control you use to animate and blend between these target poses. When you create a pose blend, you'll set the dimensions of the pose blend, and then add Target Poses to the Pose Blend.

When you select a Pose Blend, you'll see a dropdown to select the dimensions of the Pose Blend. A Pose blend can either be one dimensional:



A one dimensional Pose Blend is good when you want to blend between poses that form a single kind of action. For example, open and closed eyes, happy or sad mouth, open or closed hand, etc.

Or a Pose Blend can be two dimensional:

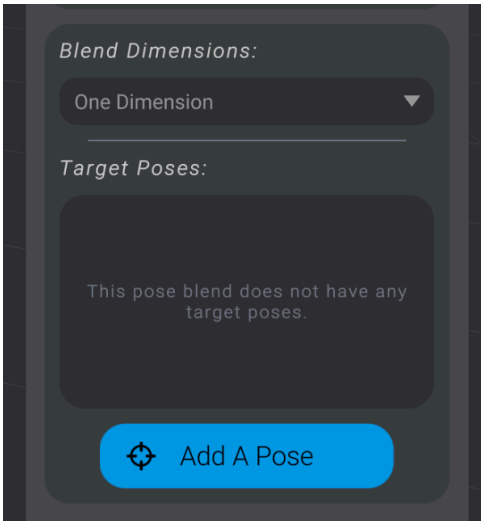


A two dimensional Pose Blend has more flexibility, and can be used to blend between multiple actions. For example, head turn left/right on the x axis, and head nod up/down on the y axis. Or as another example, different target poses representing different mouth shapes.

Adding Target Poses to Pose Blends

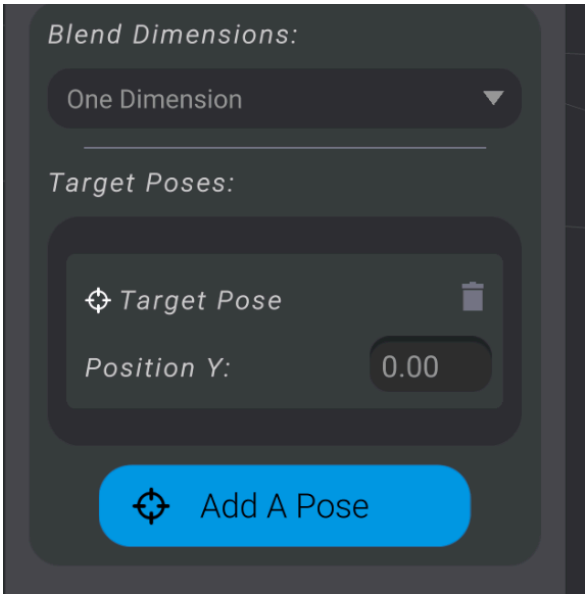
In order to blend between Target Poses, you need to add Target Poses to the Pose Blend.

1. With a Pose Blend selected, click "Add Target Pose"

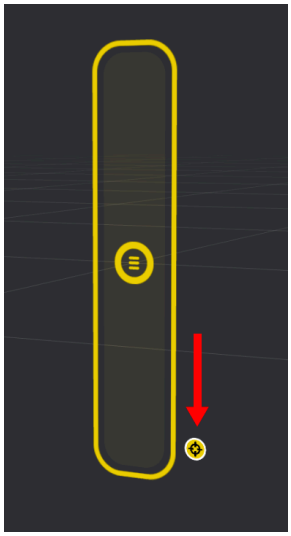


2. Select a Target Pose to add to the Pose Blend.

The selected Target Pose will now be added to the Pose Blend. You'll see it listed in the Target Poses on the Pose Blend:



As well you'll see it's location on the Pose Blend represented on the Pose Blend itself:



In order to change the location of a Target Pose on the Pose Blend, you can either drag the icon representation on the Pose Blend, or change its numerical position value.

On a one dimensional Pose Blend, 0 is all the way at the bottom, and 1 is at the top. On a two dimensional Pose Blend, the first value is the x position (0 is left, 1 is right) and the second value is the y position (0 is down, 1 is up).

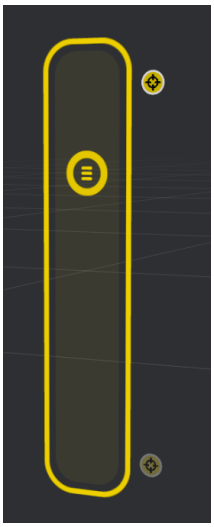
We'll cover why you'd want to change the Target Pose position in the next section.

Blending between Target Poses

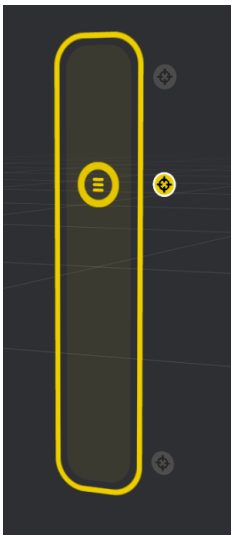
In order to blend between Target Poses on a Pose Blend, you move the handle. In a one dimensional Pose Blend, you can move the handle up and down, and in a two dimensional Pose Blend you can move it up, down, left, and right.

A Pose Blend will blend between the Target Poses on it based on distance from the Handle. You can see how much each Target Pose is contributing to the final calculated position based on the brightness of the Target Pose icon.

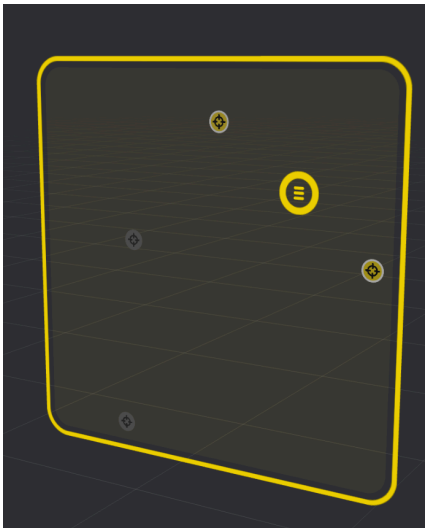
For example, in this one dimensional Pose Blend, the handle is 75% of the way to the top Target Pose, so the calculated final position will be a blend of 75% of the top Target Pose, and 25% of the bottom Target Pose.



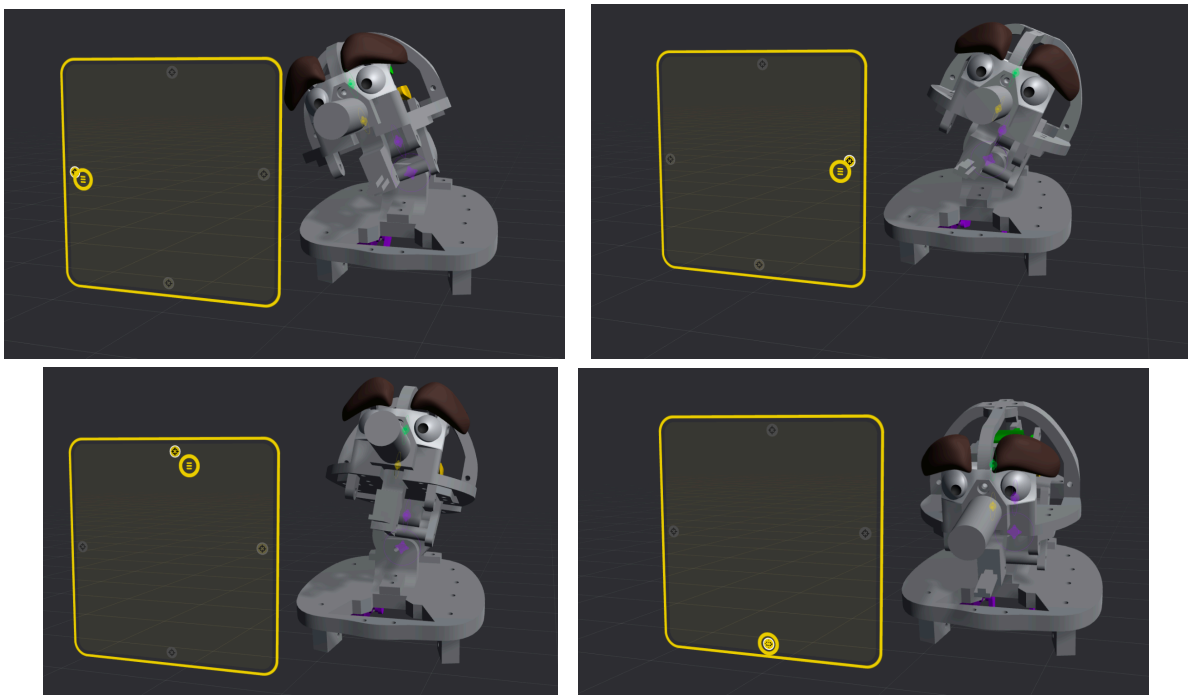
Or in this example, the handle is right on top of a Target Pose, so the final calculated position will be exactly that Target Pose and none of the others.



Here you can see a two dimensional pose Blend. The closer targets to the handle contribute more to the final calculated position than the further targets.



Finally, you can see in the head control example earlier in this chapter, as you move the handle around, you blend between looking up/down in the y position, and left/right in the x position.

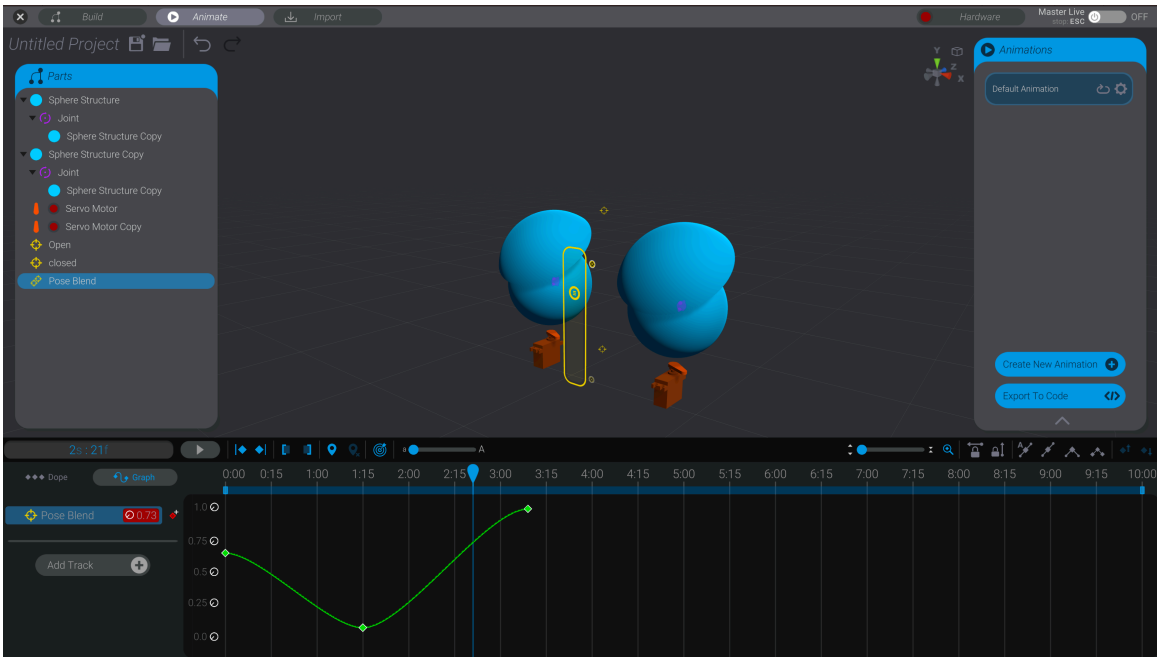


Animating Target Poses

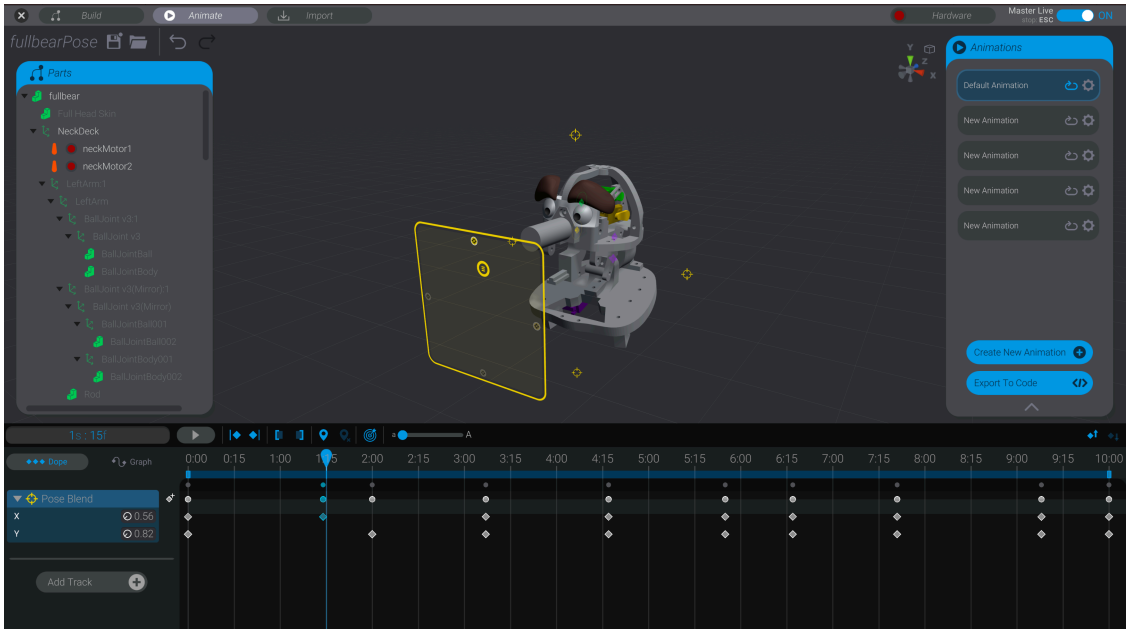
You animate a target pose much the same way you animate a joint. While in animation mode, you can drag the handle to positions, creating keyframes.

Bottango will animate between those positions, and interpolate the various intermediary positions as well.

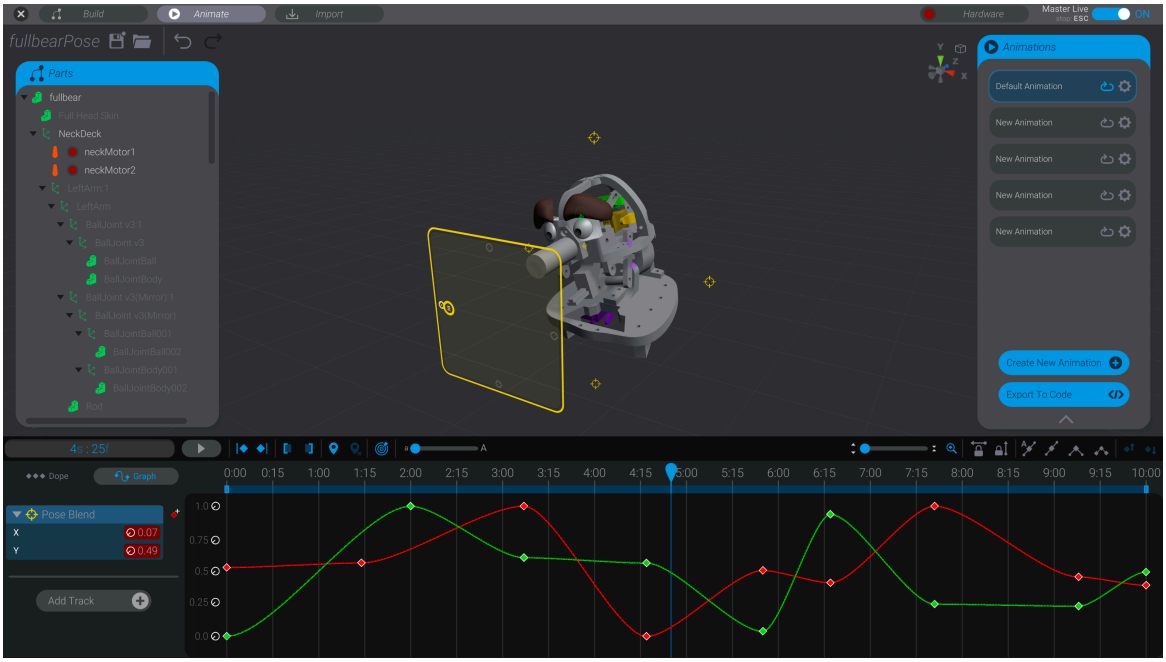
For one dimensional Pose Blends, you can control the keyframes and interpolation curves the same way you can a joint's keyframes.



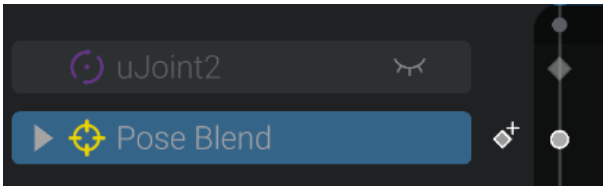
For two dimensional Pose Blends, you have keyframes for each dimension. As well, there is a master keyframe that lets you drag/control both dimensions at once on each keyed frame.



And as well you control the interpolation of each dimension independently.

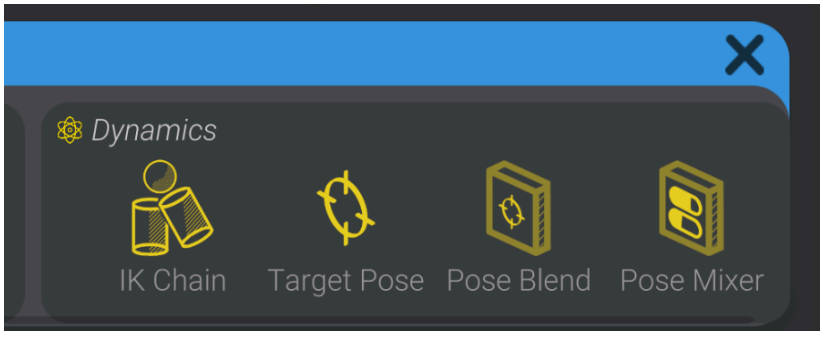


As discussed in the earlier chapter on Animating Basics, if you had animation tracks with target parts previously in an animation that are now animated via a Pose Blend, you will see the previous tracks locked and in an obscured state.



Build Any Mech with Pose Mixers

You may have noticed an entirely additional item in the create panel next to Pose Blend, the Pose Mixer:

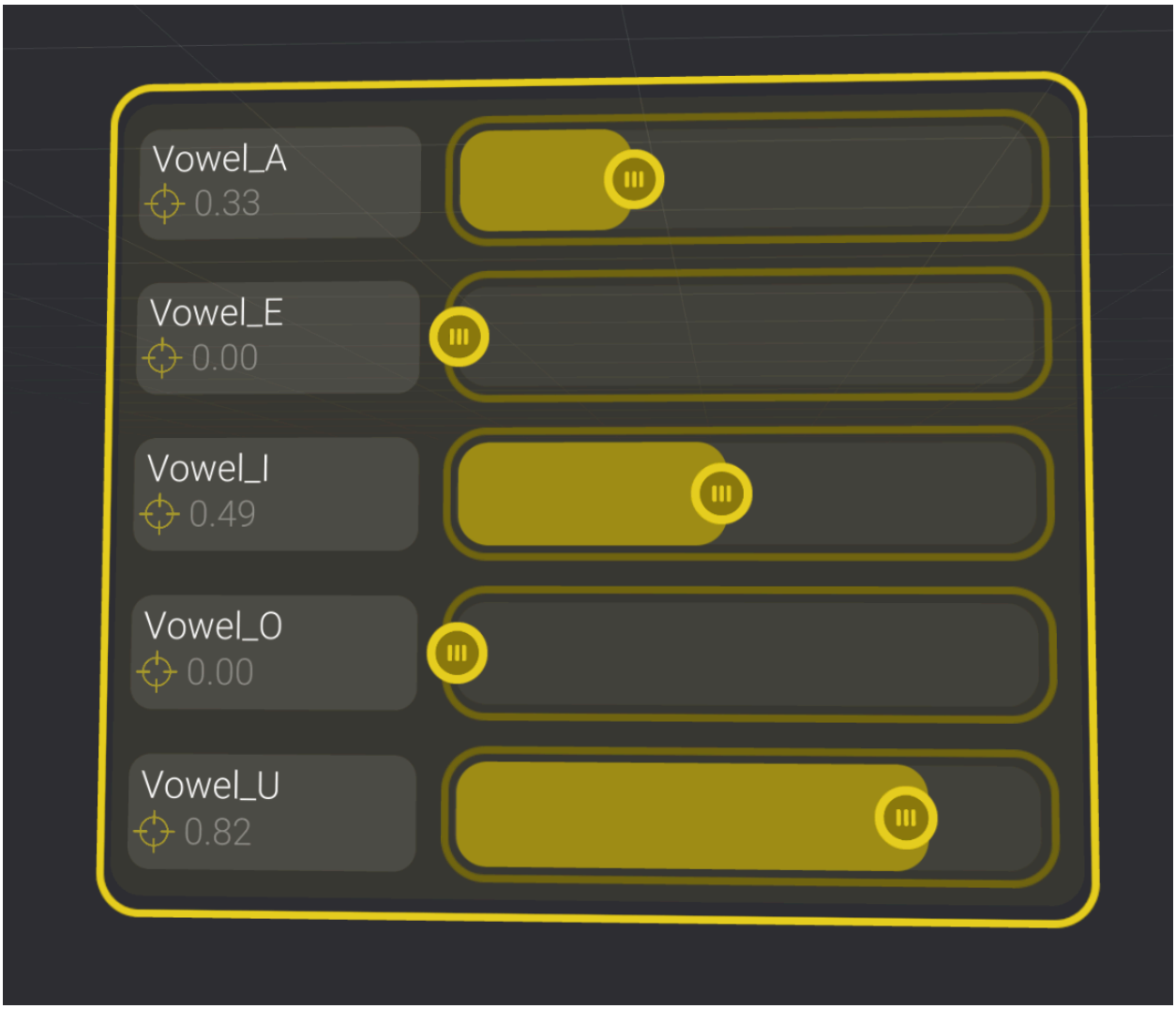


Pose Blends are powerful tools for blending between target poses in either a one or two dimensions. A one dimensional pose blend is a great fit for the opening and closing of eye lids, a two dimensional pose blend is a great fit for the movement of the pupils of an eye. But what if you have a mechanism that doesn't follow such positional mappings?

Pose mixers instead allow you to create blends between target poses that don't have an intuitive mapping to a one or two dimensions.

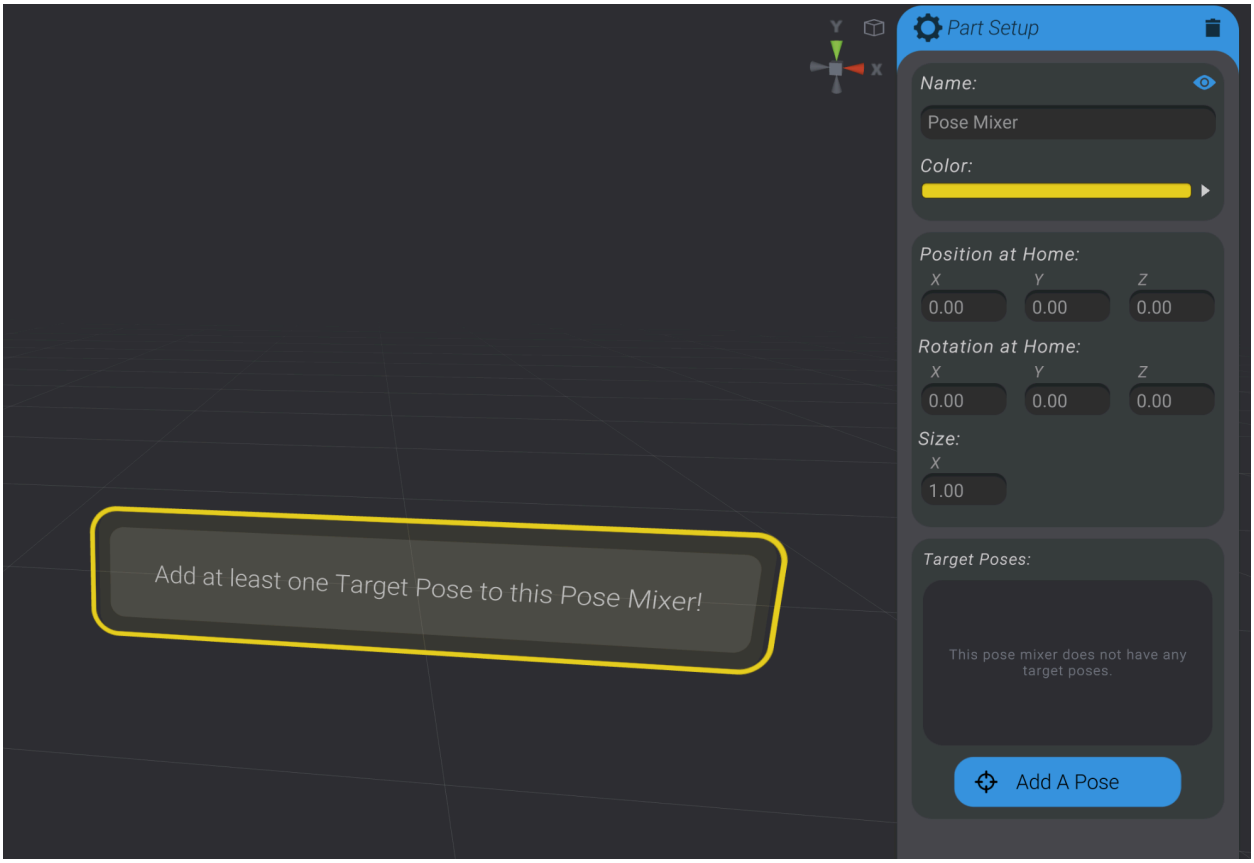
As an example, imagine that you had a target pose set up for the motors that controlled a robot's lips, and each target pose mapped to a vowel shape (A, E, I, O, U). Maybe in this case you want to mix in 70% E and 30% U, and then blend to 30% U and 70% A. Positioning those vowel shape poses on a 1 or 2 dimensional controller doesn't intuitively make sense and it restricts you to make choices about the relationship between the poses that may not be true.

This is where a Pose Mixer becomes the right tool for the job.

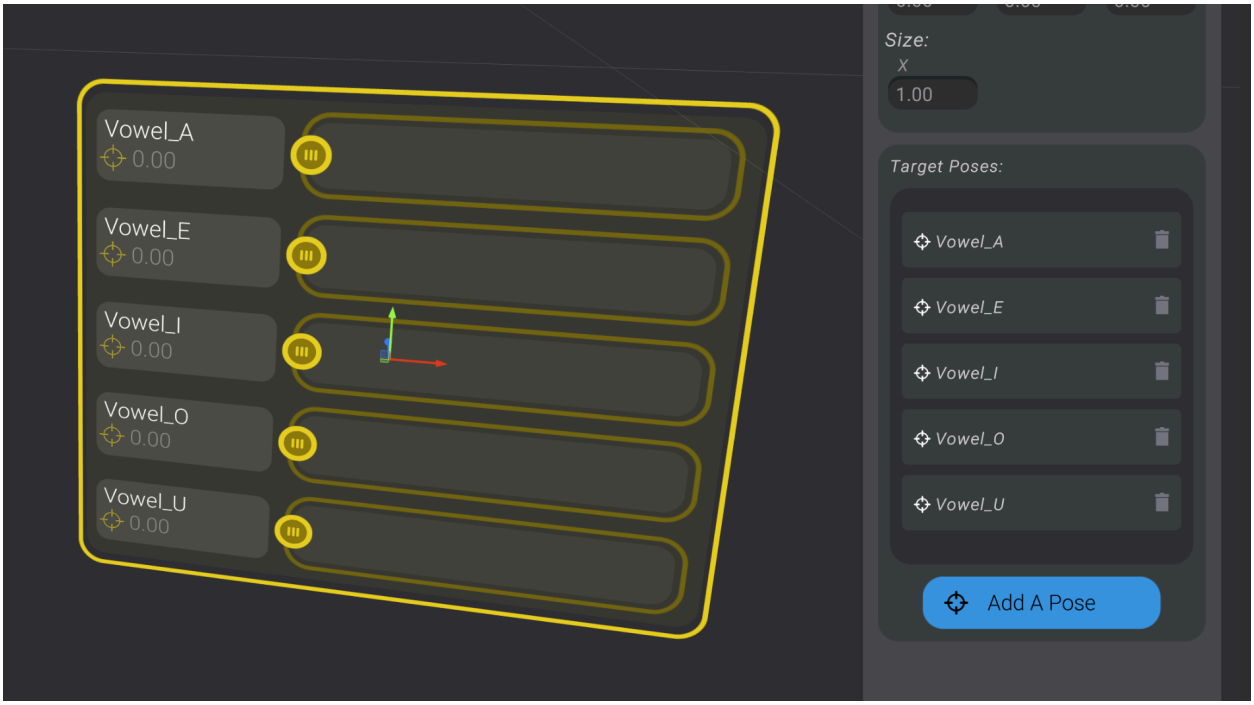


Creating and Setting up a Pose Mixer

When you create a Pose Mixer from the create panel, it will be initially empty:

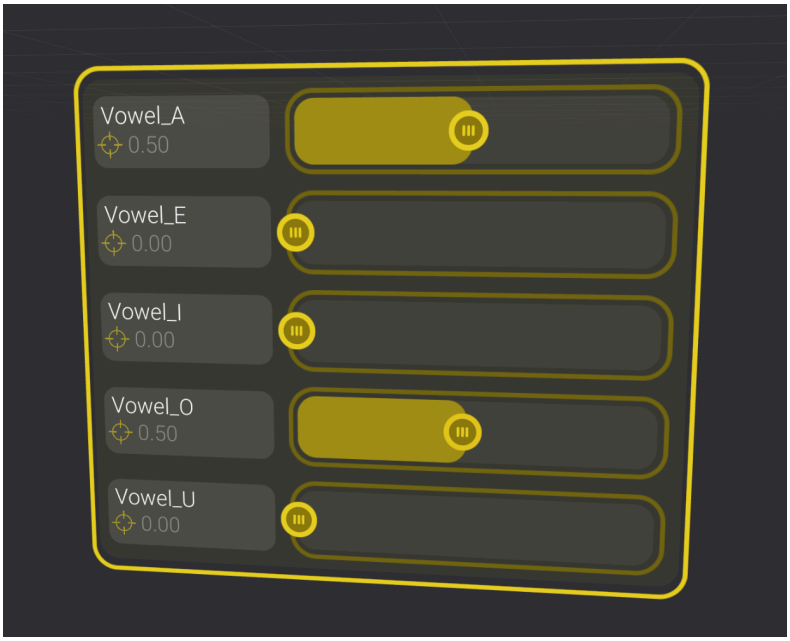


You can click the "Add a Pose" button to add target poses to the pose mixer in the exact same way as a Pose Blend.



Once you have added Target Poses to the pose blend, they will show up both on the pose blend itself, and in the part details menu. You can delete target poses from the pose mixer using the "trash" button, or drag to rearrange the order of the poses in the pose mixer.

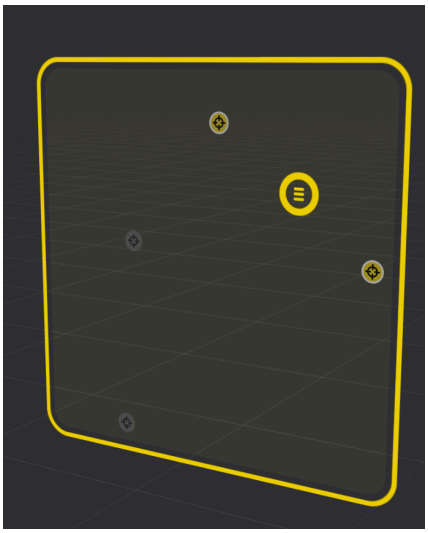
If I were to drag up the Vowel_A slider and the Vowel_O slider as shown:



The final pose will be calculated as a 50/50 blend of Vowel_A and Vowel_O

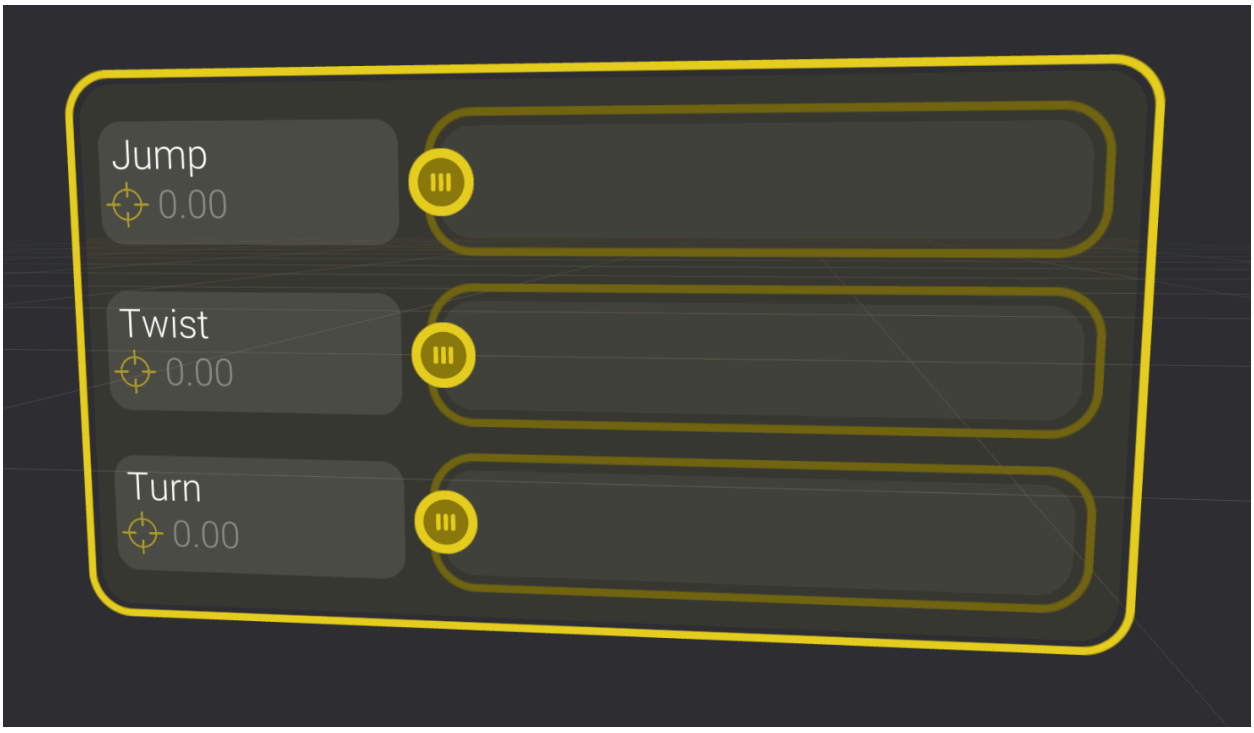
Pose Mixers Mix from Home to Fully Expressed

Let's look at a 2d pose blend for a moment:



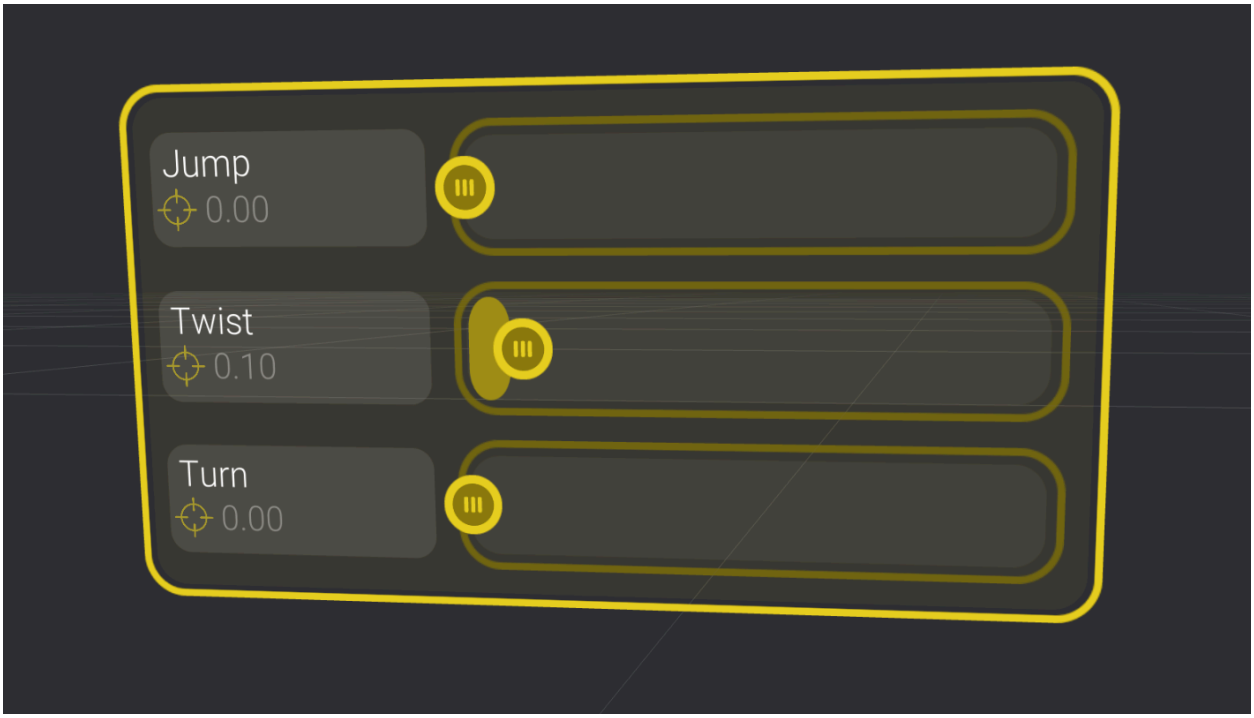
In this pose blend, the target pose is roughly a blend of 50/50 between the top and right target poses. Because there's always a "closest" target pose in the map, the final calculated pose will always be some mix of the target poses in the pose blend.

However, now let's look at this pose mixer:



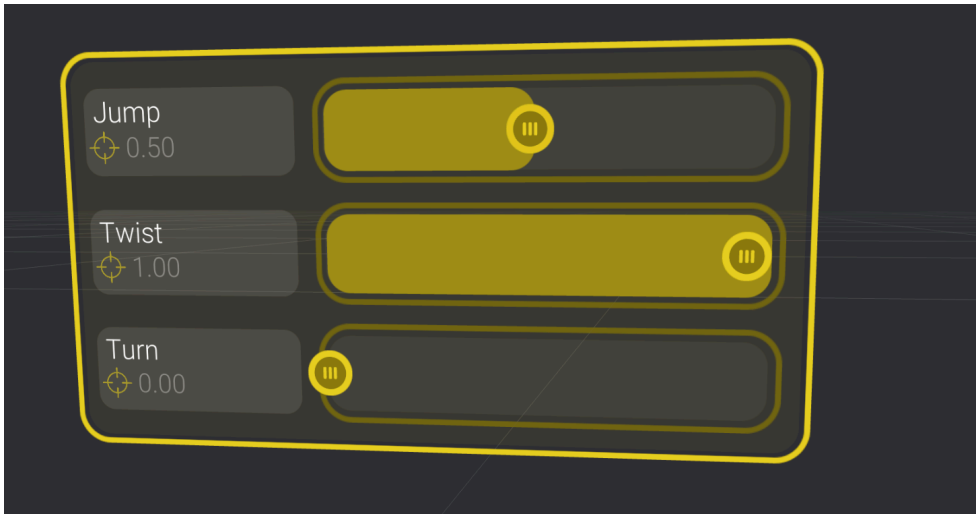
We would intuitively expect with this setting, we're not blending in any of Jump, Twist or Turn, instead of an equal mix of all three. This is exactly how a Pose Mixer works. **Fully left on a slider is the home position of all the parts in that target pose. Fully right on a slider is a full expression of the target pose.**

This mean on the following pose mix:



The calculated pose would be 10% twist expressed, 90% twist home, and the other poses are ignored.

You also can take your mix total percentage over 100%:



In this setup, the final mix would be 2/3rds “Twist", 1/3rd “Jump”, and no “Turn.”

If this is unintuitive to read through, try it out in practice and the end result will be come quickly predictable. This is a behavior the plays out as you would expect, but is difficult to describe in text.

Animating Pose Mixers

If you have any experience animating in Bottango, animating Pose Mixers should behave as expected. Drag on the sliders to set keyframes, and modify those keyframes using the dope and graph view.

Expanding the track in the tracks panel of the animation window will show the value of each target pose in the pose mixer, and allow you to set each target pose’s blend value explicitly.

-18-
Recording Live Input and Puppeteering

What's in this chapter

A common from of animating in the animatronics industry is to live capture input, and use that input to puppeteer an animatronic or robot. That input may come from an RC or game controller, a custom animation rig or control deck, or any number of sources. In this chapter, we'll look at how Bottango not only supports that workflow, but builds upon it with the power of the modern 3d animating workflow, by converting live recorded input into editable keyframes and bezier curves.

What kind of input devices are supported?

The short answer to this question is anything that acts as an HID device. If you're not immediately familiar with what that means, **any keyboard, mouse and most any USB game controller are supported**. Although I cannot test all the game controllers that exist, if it's a fairly common controller or joystick, it probably works with Bottango, and even if it's an esoteric one, it just might as well.

As well, any device you build yourself that can act as an HID device can be used as well, in case you want to build your own animator's console with potentiometers, switches, buttons, etc. An Arduino Leonardo is a great way to build your own hardware that can act as an HID device, look at tutorials on that specific board for how to do so.

The following input types are **not yet supported** but planned for future versions:

- MIDI.
- Socket based API for scripting input (similar to the Playback API).
- Audio clips as input sources (for automatic lip syncing, etc.).
- Motion capture via image processing of a camera feed.

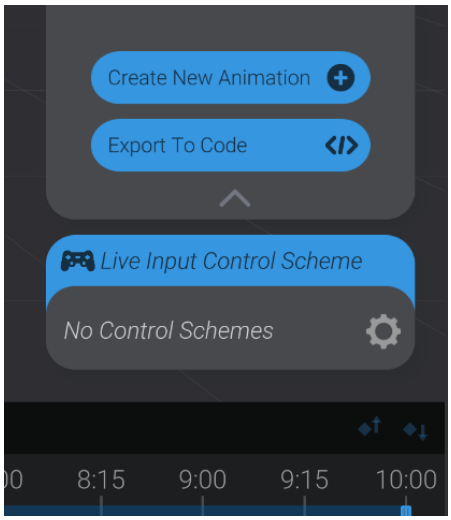
What can I live puppeteer and record in Bottango?

If you can animate it in Bottango, you can puppeteer it! Not only does that include motors and joints, but custom events and pose blends can be puppeteered and recorded as well! As more effectors are added to Bottango in the future, live control support will be included as part of those new features.

Getting started and setting up a Control Scheme

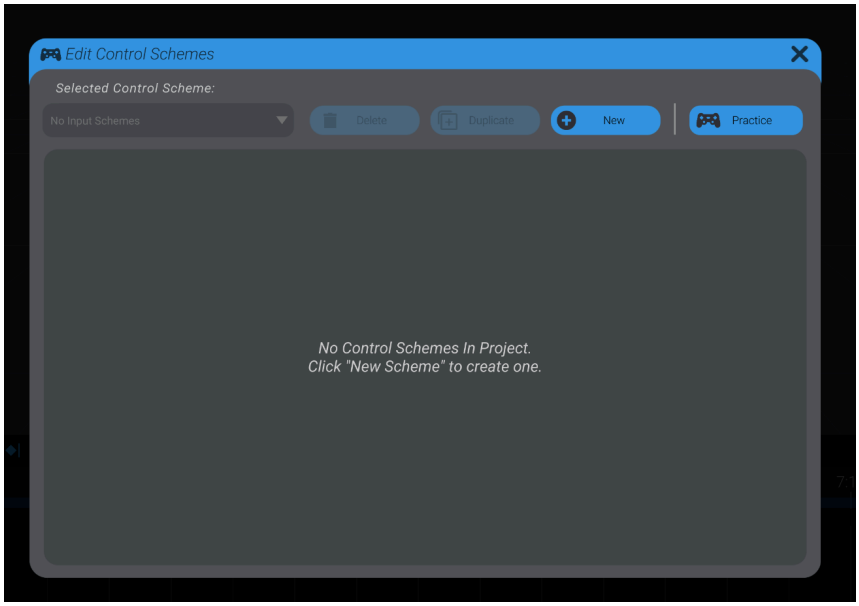
To start recording live input and puppeteering, you need to create a Control Scheme. In Bottango a Control Scheme is a set of controls that define the binding between the parts in your project (like a joint or pose blend) and specific inputs, like the "0" button on your keyboard, or the horizontal axis of the left stick of a game controller.

To create your first control scheme, go to the Animate mode of Bottango. Underneath the animations menu on the right of the screen, you'll see a menu to set up a "Controller Input Scheme."

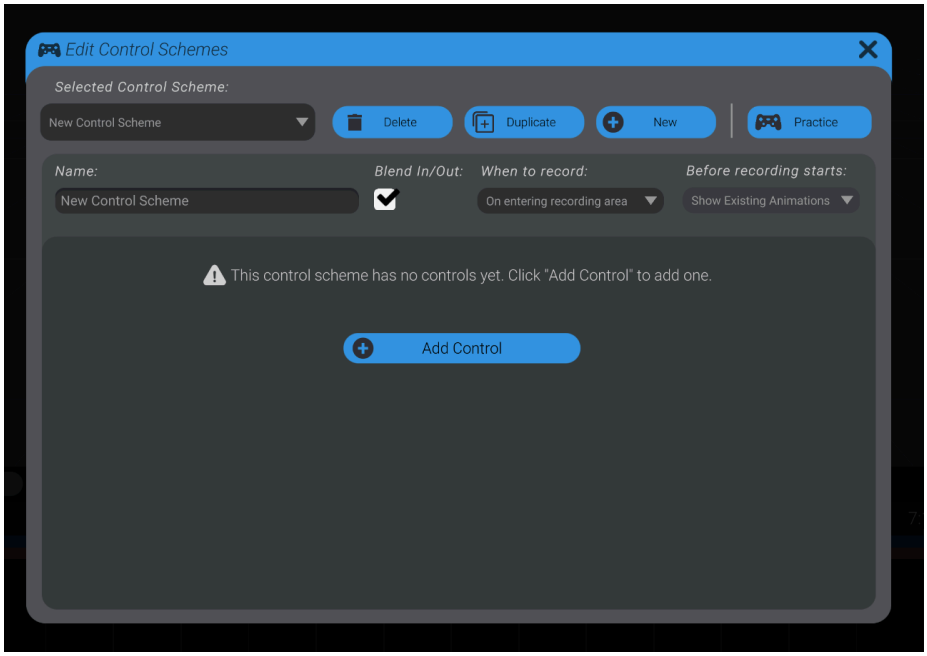


Unless you have previously created a control scheme in your project, you'll see the warning "No Control Schemes," indicating that this project does not yet have at least one control scheme in place. Hit the "gear" icon to configure and set up your first control scheme.

Here you'll see the control scheme editing window, which similarly warns you that you don't yet have a control scheme set up.



Click the "New" button to create a new, empty control scheme.



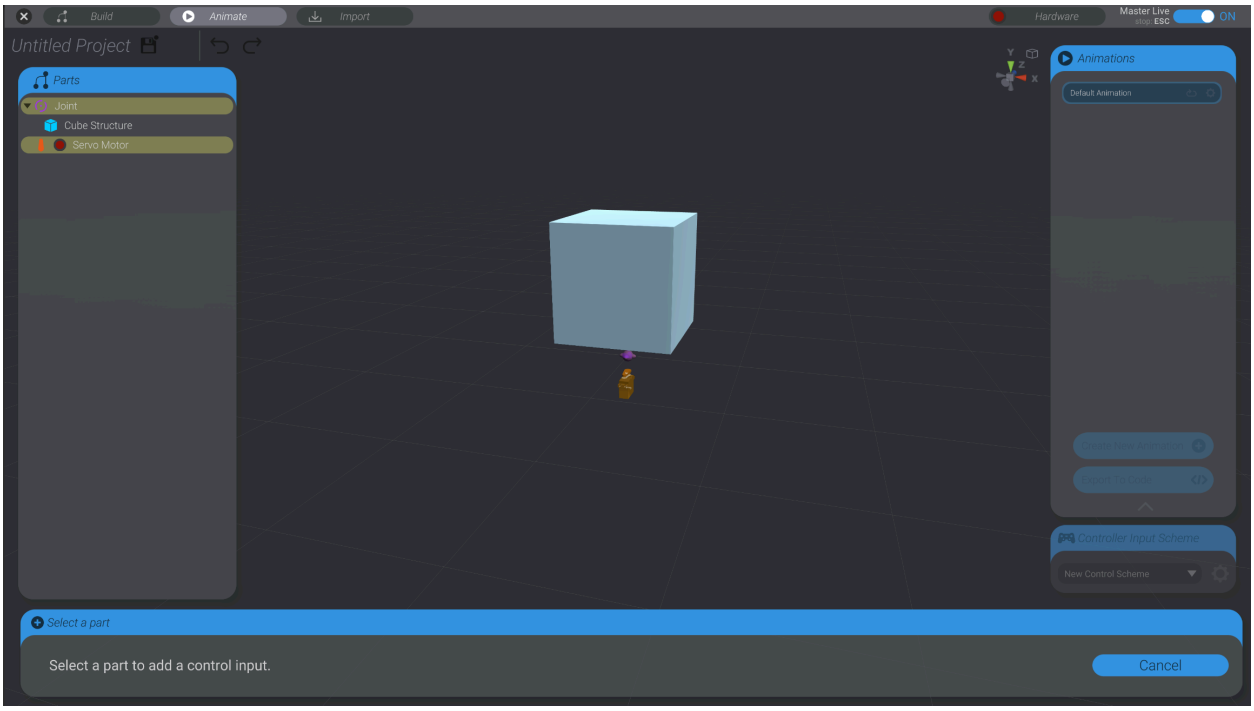
So we now have a new control scheme, which you can see is named "New Control Scheme." However, a control scheme is a container of controls, and this container is still empty.

Adding a control to a control scheme

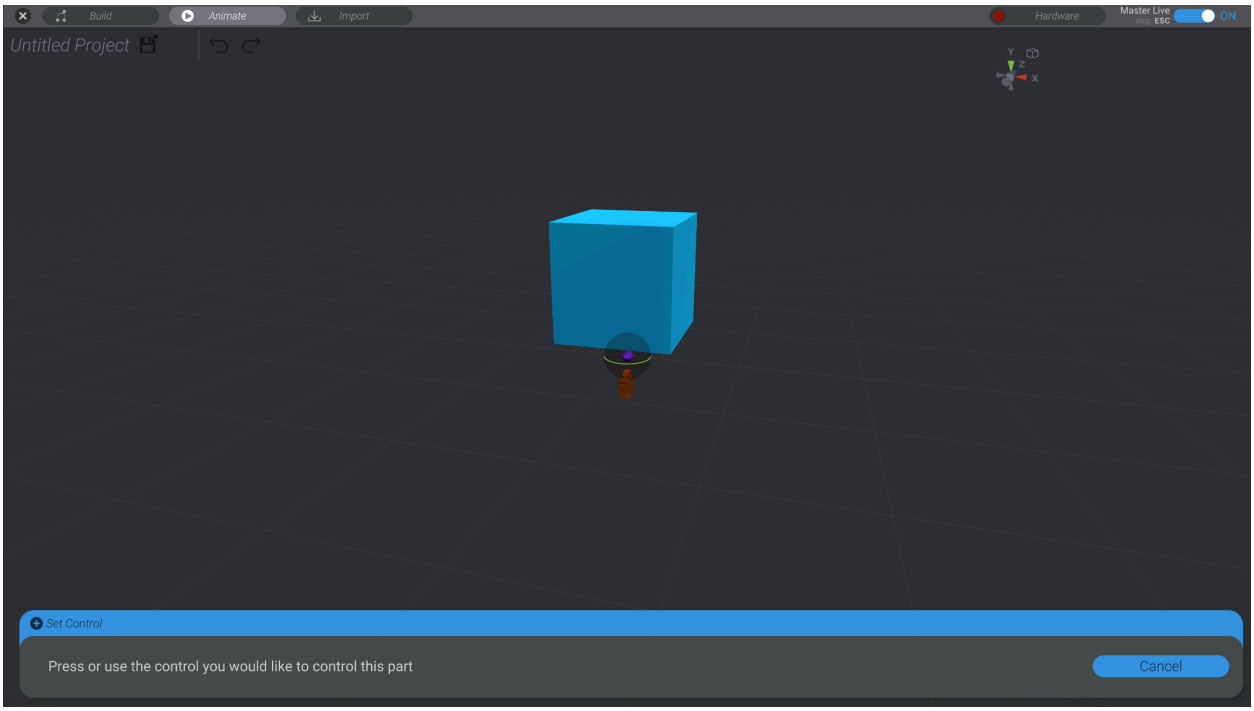
Let's add a first control, by clicking the "Add Control" button.

Adding a control has three major steps. First we select the part we want to control with live input. Then we press or use the input we want to control that part with. Finally, we tweak and configure the settings of how that input maps to control of the selected part.

For this documentation, I have a very simple project with a cube in a joint, and that joint linked to a servo motor. When I click add control, I'm prompted to click to select any animatable part in my project, that doesn't already have a control in this control scheme:



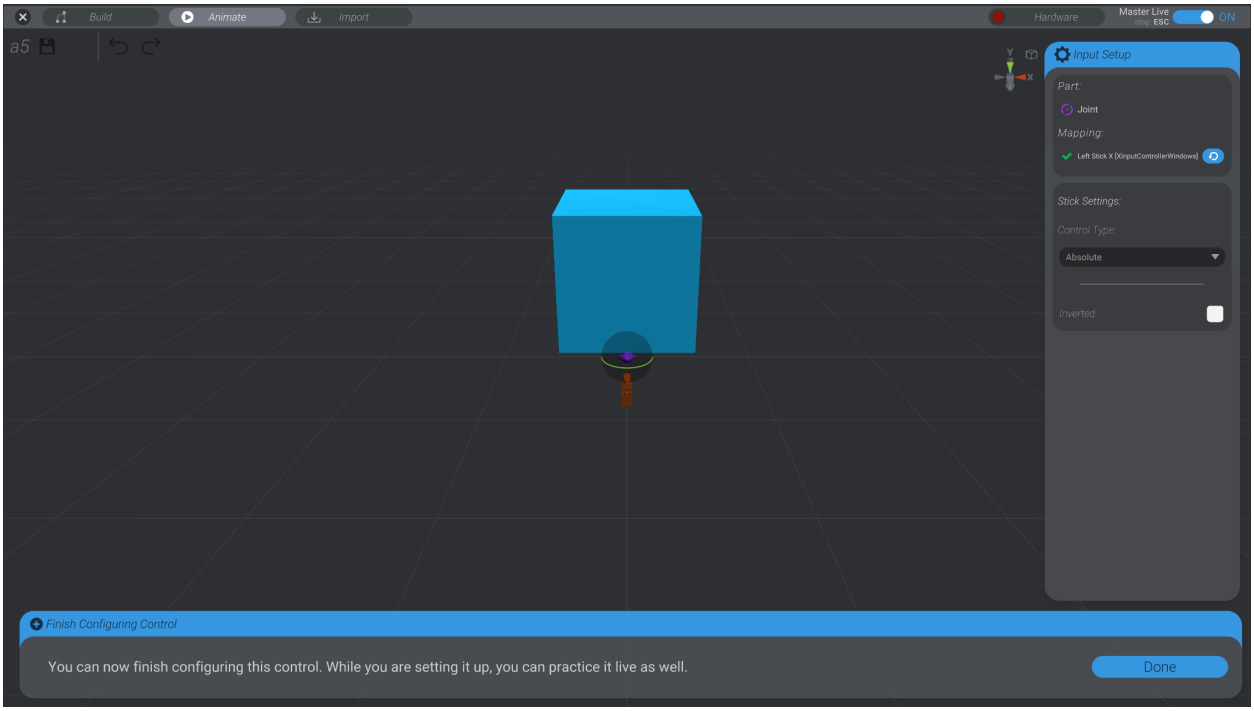
I have two options as shown, either the joint or the servo. Depending on my needs, I may want to animate the servo directly, but likely I would want to animate a joint, custom event, or pose blend, rather than the motors themselves directly. Your own use cases and project will define what parts of the project you want to animate and live control. I click on the joint, and am prompted to now press, nudge, or otherwise use the live input button/stick/etc. that is connected to my computer that I want to use to animate and control that joint.



In this first example, I'm going to use the horizontal axis of the left stick of my game controller, that is connected to my computer via USB.

This creates a control in my control scheme, between the selected joint and the left stick of any connected game controller. That control has the default settings chosen, but we can tweak them significantly. As well, we'll look in later sections at how I could have controlled the joint with other kinds of inputs like analog or digital buttons, etc.

After selecting the joint, and then pressing the horizontal axis of the left stick, I have created the control in my control scheme, and can now practice and configure that setting. You'll notice while in this menu, using the linked input will live control the selected part, so that you can test out configurations.

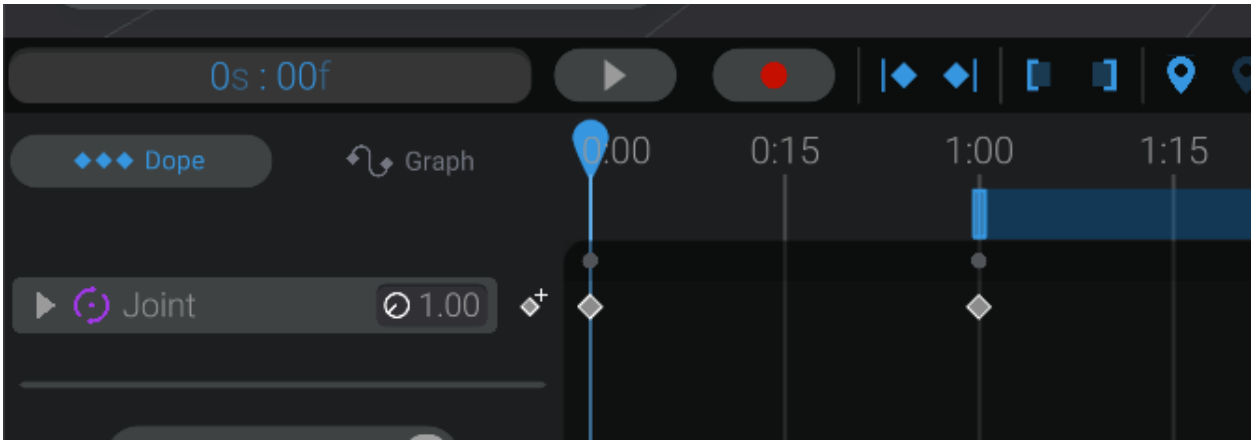


For now, let's just hit "Done" to save this control, and try our hand at recording. We'll look at configuration options in a later section.

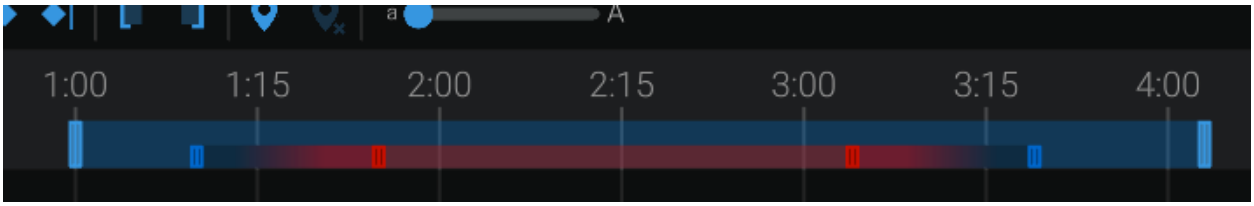
Recording Live Input into in Animation

Going back to animation mode, you'll see that now that we have selected a control scheme, our animation view has changed a little bit.

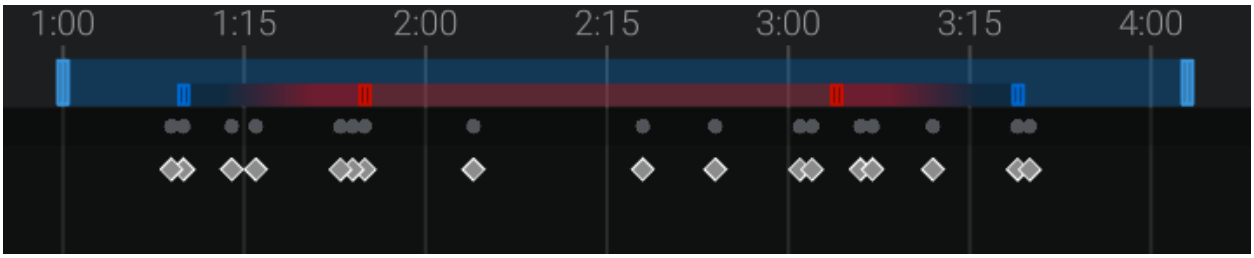
First, a record button has been added to the view:



And second, a new slider has been added to the work area. This slider represents where in the work area it's OK to capture and record input. Bottango will capture live input and insert keyframes only where the second smaller slider overlaps. We'll discuss the different colored handles of this slider in the next section.



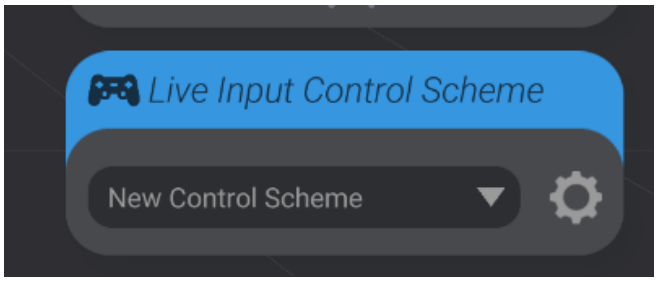
For now, I'll press the record button in the animation view. This will begin playing my animation. However, it will also begin using my live input, and showing that live input instead of the existing animations when recording. After the animation is finished playing, or I hit pause, my live input will be converted into keyframes, and replace any existing animation data over the record area time.



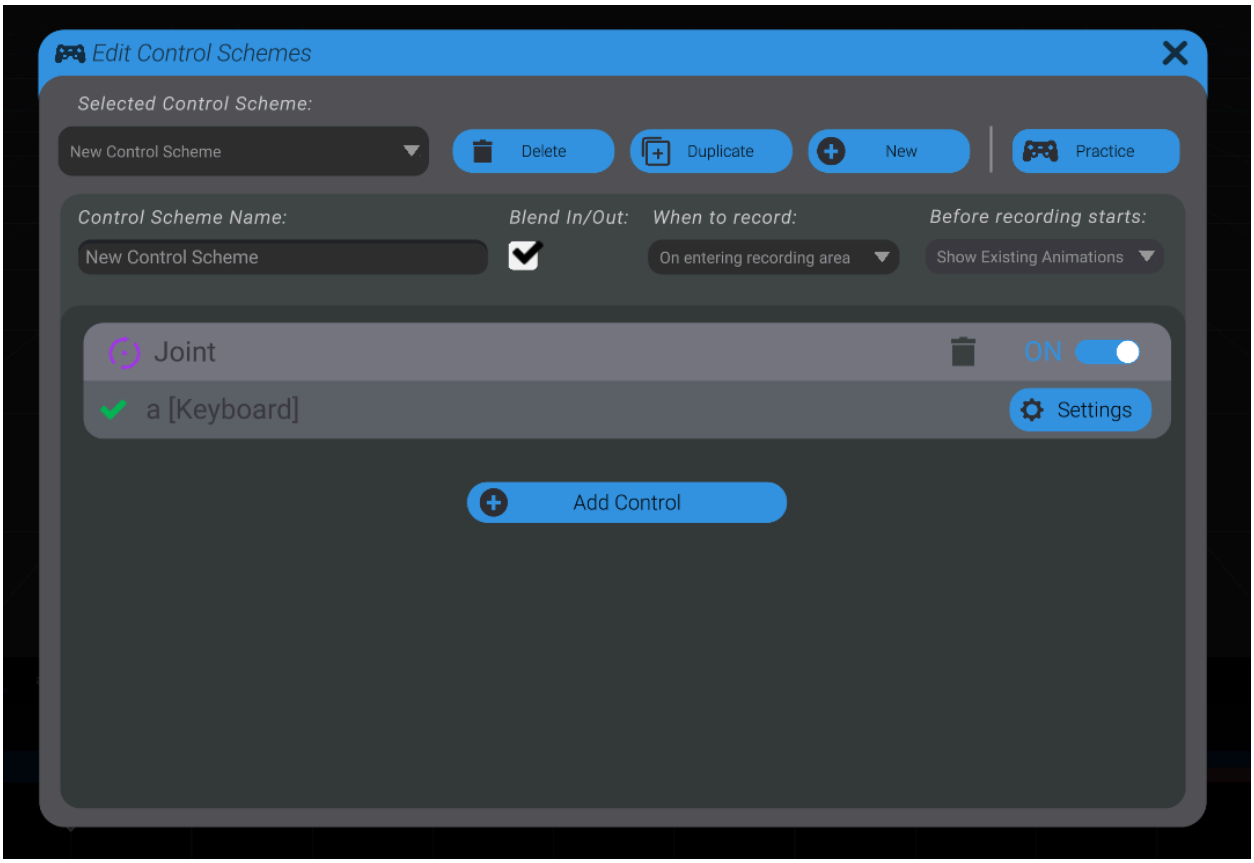
Here you can see Bottango captured my input, and added keyframes for all the recorded movements. These keyframes are editable just like keyframes you created yourself, if you want to tweak the resultant animation. As well, you can undo recording after it is applied.

Control Scheme Settings

Press the gear button again on the Controller Input Scheme menu, to edit the settings on your control scheme.



This will bring up the control scheme settings menu again.



At the top of this menu, you can see a dropdown of which of your control schemes you are configuring. As well, you can delete or duplicate the selected control scheme, as well as make a new empty control scheme.

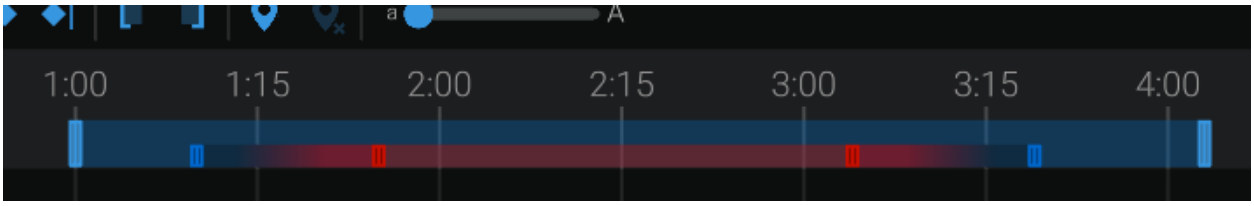
As well, at the top of the menu is a "Practice" button. This will turn all your live input on for the selected control scheme, and let you practice and puppeteer your project outside of an animating context.

The bottom portion of the scheme is where you configure the control scheme selected in the top portion of the menu.

Control Scheme Name lets you change the name of the selected control scheme.

"When to Record" lets you select when your animation will actually start converting your input into keyframes.

The default setting is "On entering recording area." This means, once you hit the record button, keyframes will only be captured once your playhead enters the smaller record area slider:

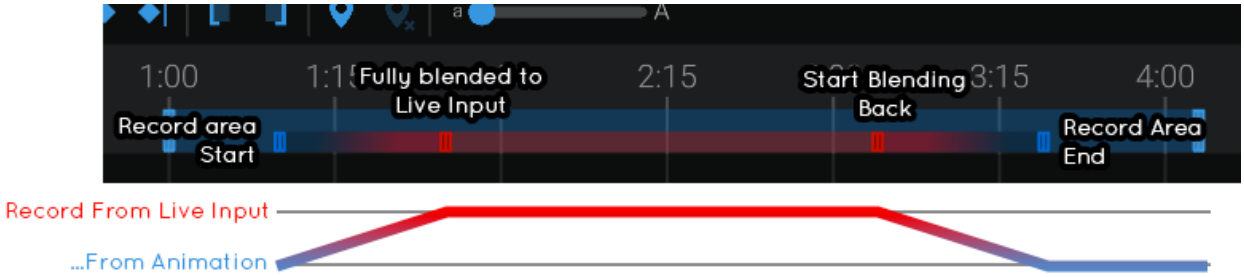


You can also select "On First Input." With that option selected, keyframes will be captured only once you first use any of the mapped inputs, while your playhead is in the smaller record area.

Finally, you can select "Practice (don't record)." This will still show live input when you hit the record button, but it won't actually change or add any keyframes to your animation.

"Before recording starts" lets you select what you want to see when you hit the record button, but your playhead hasn't yet entered the record area. By default, it will show any existing animation data, and then switch to live input once keyframe capturing begins. However, you can set it to "Show Live Input" and you will see live input even when keyframes are not being captured. This option is disabled if "blend in/out" is enabled.

"Blend in/out" is enabled by default and lets you blend smoothly from your existing animation to live input when recording starts. With this option disabled, your animation will jump immediately from any existing animation to whatever state is shown by your live input. However, by enabling it, instead, your capturing will over time blend from your existing animation to your live input. You can control the blend in and out time with the sliders on the record area in the animation view.

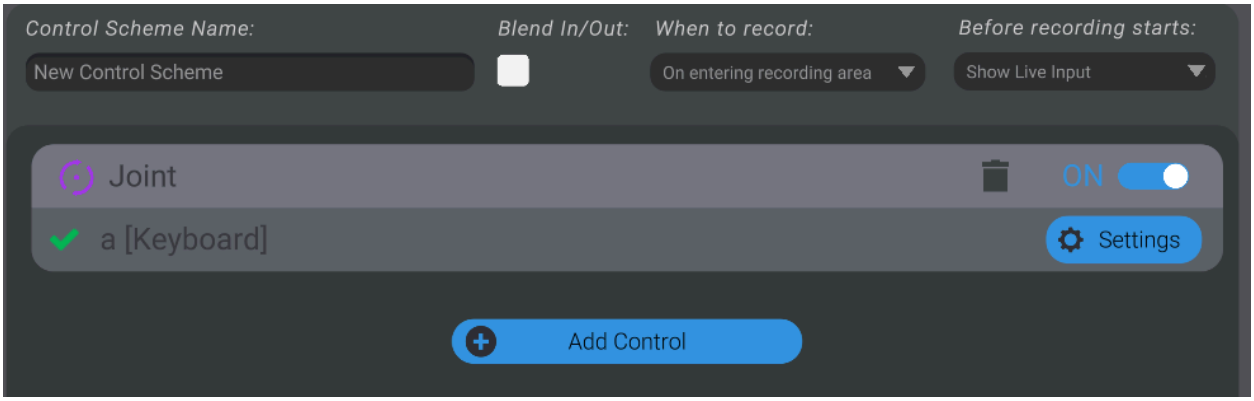


The outside sliders of the record area dictate where the record area begins and ends. The inside sliders dictate the time to blend from existing animation to live input.

If you disable "blend in/out," the inside blend time sliders go away, and you can only set the record area range.

Individual Controls in the Control Scheme

A control scheme is made of any number of individual controls, in which you map between a live input like a button or a stick, and an animatable part in your project. The controls that are in the selected control scheme are listed in the bottom of the menu:



For each control, you'll see the name of the part selected. As well you can press the trash button to delete this control from the control scheme.

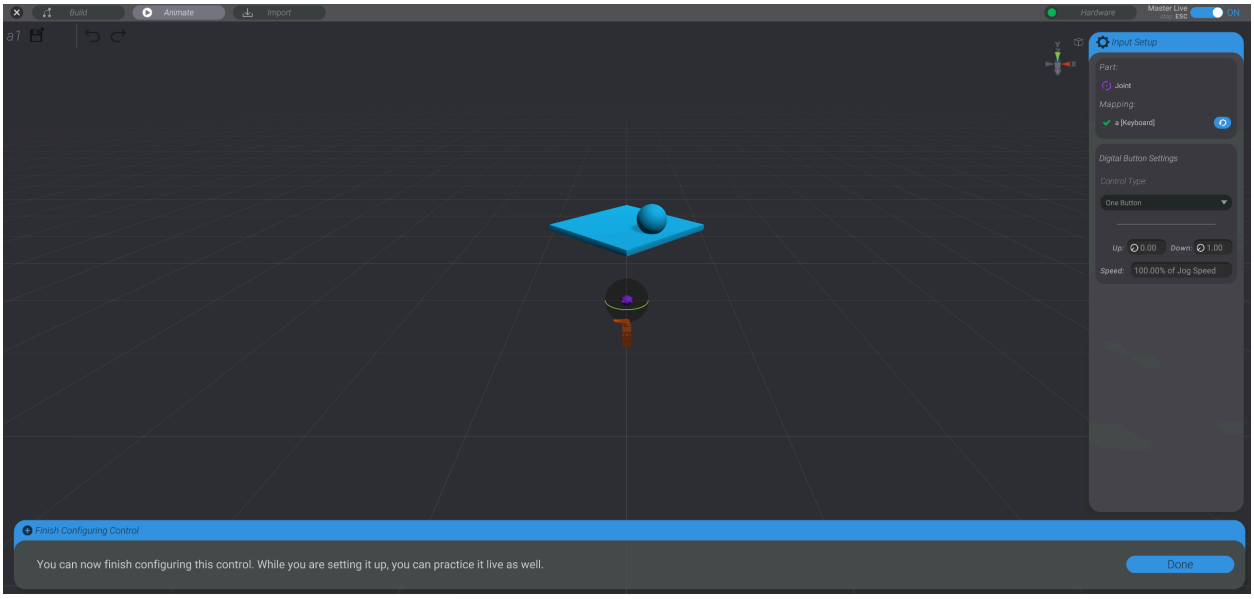
The "On/Off" toggle sets whether that control is enabled. When enabled, it will show live input and capture keyframes while recording. When "off" it will act as if the control is not part of the control scheme at all. This can be useful if you want to record just a certain part of your control scheme, and are happy with or don't want to effect the other animation tracks.

The bottom portion of each cell shows the current binding. Here you can see this joint is mapped to the "a" key on my keyboard. A check is shown if Bottango has that input currently connected to the computer, and an X is shown if it is disconnected. Finally, you can press the settings button to configure in detail how the live input maps to movement of the selected part.

Finally, you can always add another control to the control scheme using the "Add Control" button.

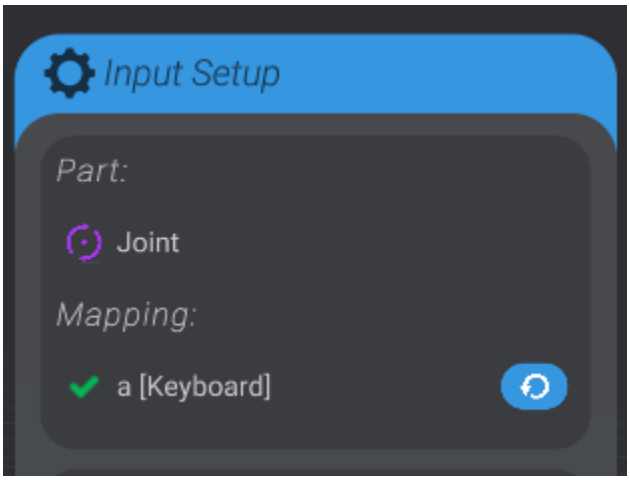
Detailed Control Settings

When you first create a control, and select an input to control the selected part, or you press "Settings" on a control, you will be shown the menu to set how using that button moves the selected part.



While in this view, the selected control is enabled, and live input is shown so you can test your configuration.

The first portion of the menu shows the name of the part as well as the current binding.



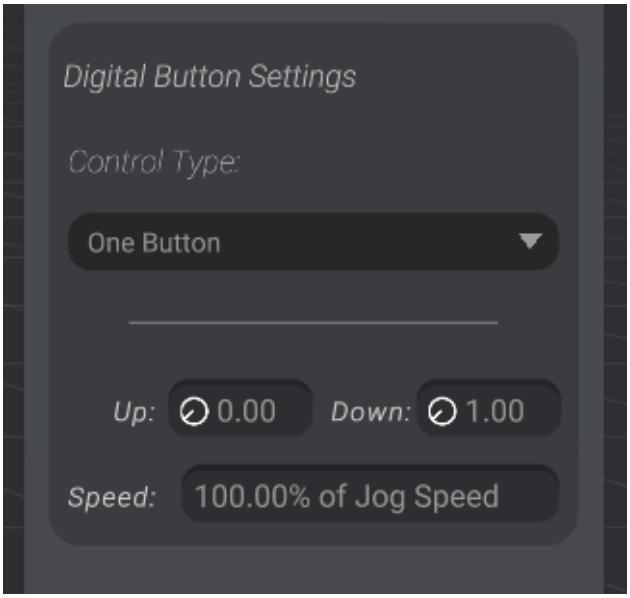
You can press the "refresh" button to select a new binding. You are not limited to the same kind of binding you initially chose. In this case, I could rebind from the "a" key on my keyboard to any other type of input, such as a stick on a game controller.

If you rebind the control, the rest of the settings will change to give you settings for the new kind of selected input.

Detailed Control Settings (Digital Button)

A digital button is a button that is either pressed or released, with no in-between value. An example would be a key on a keyboard, a button on a mouse, or the "A/B/X/Y" buttons on a game controller.

The rest of the menu shown here is the settings to configure how the selected digital button will move / control the selected part.

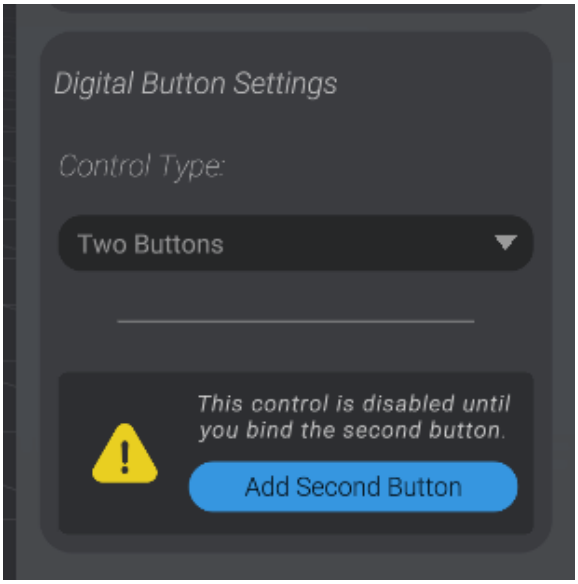


The first setting is a drop down, to select the control type. Each control type represents a strategy for how to map input to movement. Each control type has different settings, which we'll go over in detail:

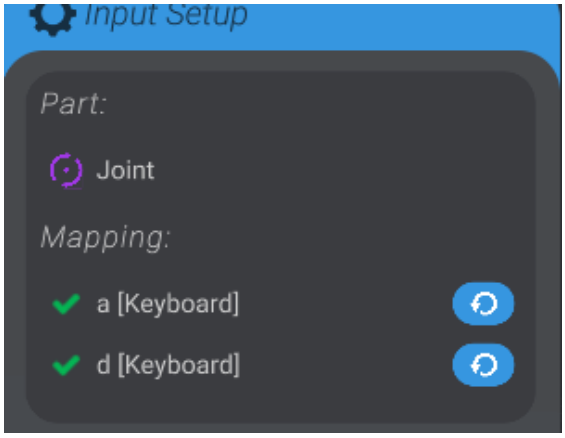
Control Type: One Button
With "One Button" selected, you can set the movement (⏲) value for the selected part when the button is pressed, and the movement (⏲) value when the button is released. The speed value sets the speed your part will move between the two movement (⏲) values.

Control Type: Two Buttons
Instead of setting the movement (⏲) value of a pressed and released single button, you map two different digital buttons to the range move movement.

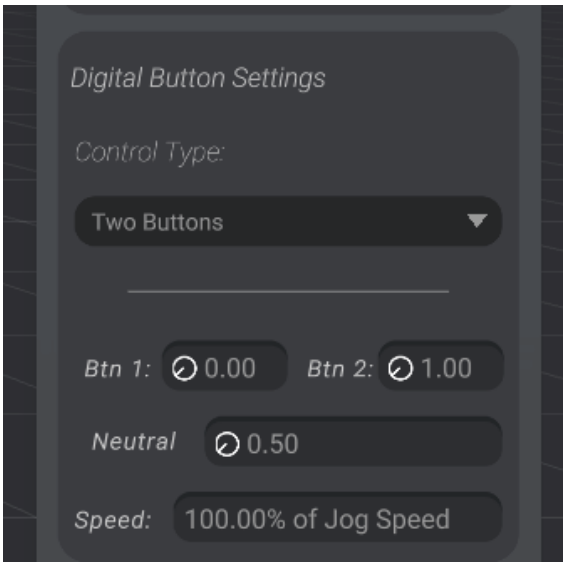
When you first select any input that needs two inputs, you'll see a prompt to select the second input.



When you press the "Add Second Button" button, you'll be prompted to select an input of the same type as the first. In this case, another digital button. Here you can see I've added the "d" key on my keyboard as the second input:



And now I can finish setting up "Two Buttons."



The "Btn 1" movement (↙) value sets the desired movement for when I press my first button (the "a" key in this example). The "Btn 2" movement (↙) value sets the desired movement for when I press my second button (the "d" key in this example). The neutral value sets the desired movement (↙) value when neither button is pressed.

Control Type: Two Buttons Incremental
In "Two buttons" when neither button is pressed, the part returns to a neutral value. In "Two Buttons Incremental" instead when neither button is pressed, the part stays where it was last inputed.

Detailed Control Settings (Analog Button)

An analog button is a button that is pressed or released, or some value in-between. An example would be a shoulder button on a gamepad, that has different amounts of pressed, in contrast with a digital button such as a key on a keyboard.

As with the other input types, there are different control types with their own settings.

Control Type: Absolute
In this control type, your input on the button will be mapped 1:1 to the movement range of the selected part. If you'd like to invert the mapping, you can check the "inverted" check box.

Control Type: Custom Range Absolute
In this control type, your input on the button will be mapped to a range of your choosing of the selected part.

Control Type: Two Buttons
See the two buttons control type in the previous section on digital buttons. This control type works the same, just for analog buttons instead.

Control Type: Two Buttons Incremental
See the two buttons incremental control type in the previous section on digital buttons. This control type works the same, just for analog buttons instead. Speed in this setting is the speed to move in the given direction when the button is fully depressed.

Detailed Control Settings (Stick)

A stick is an input that has an axis of movement, and can be actuated in either direction. An example would be a the left right movement of a stick on a game pad, or a joystick. Bottango only uses the actuated direction, so you can map the different axis of a stick however you like.

As with the other input types, there are different control types with their own settings.

Control Type: Absolute
In this control type, your input on the stick will be mapped 1:1 to the movement range of the selected part. If you'd like to invert the mapping, you can check the "inverted" check box.

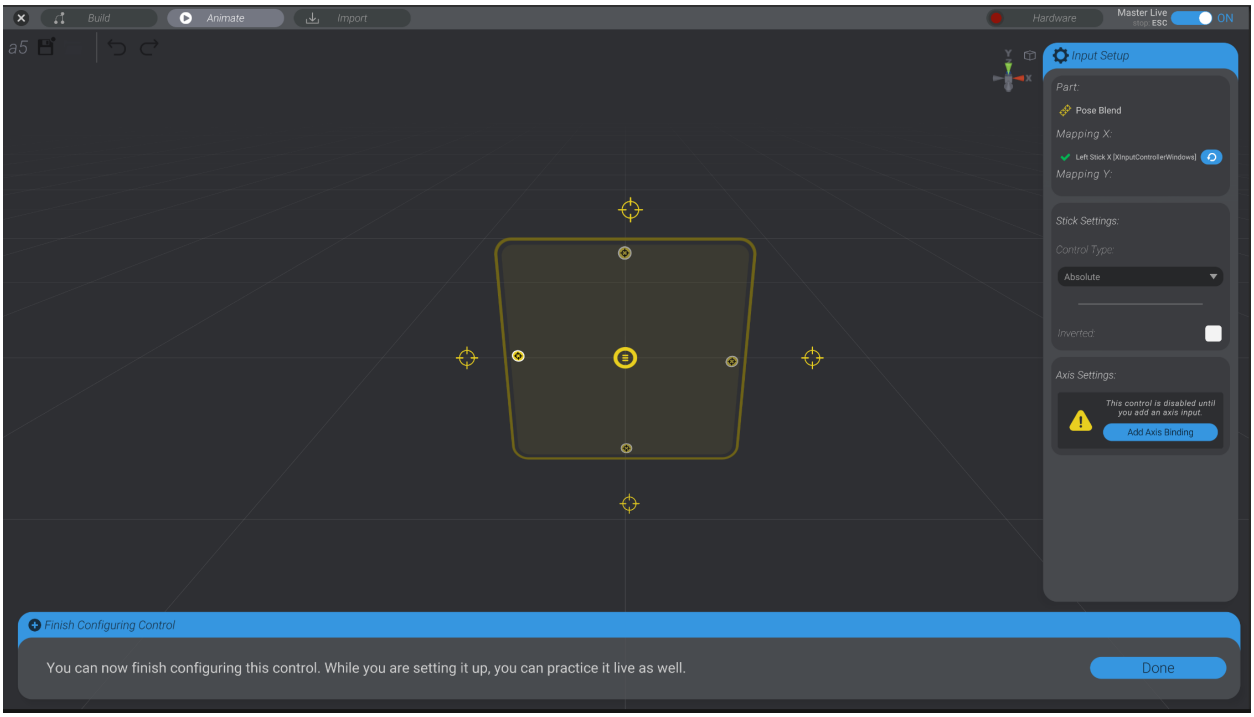
Control Type: Custom Range Absolute
In this control type, your input on the stick will be mapped to a range of your choosing of the selected part.

Control Type: Incremental
In this control type, your input on the stick will move in the given directions incrementally, and not return to neutral when the stick is at neutral. Speed in this setting is the speed to move in the given direction when the stick is fully actuated in a given direction.

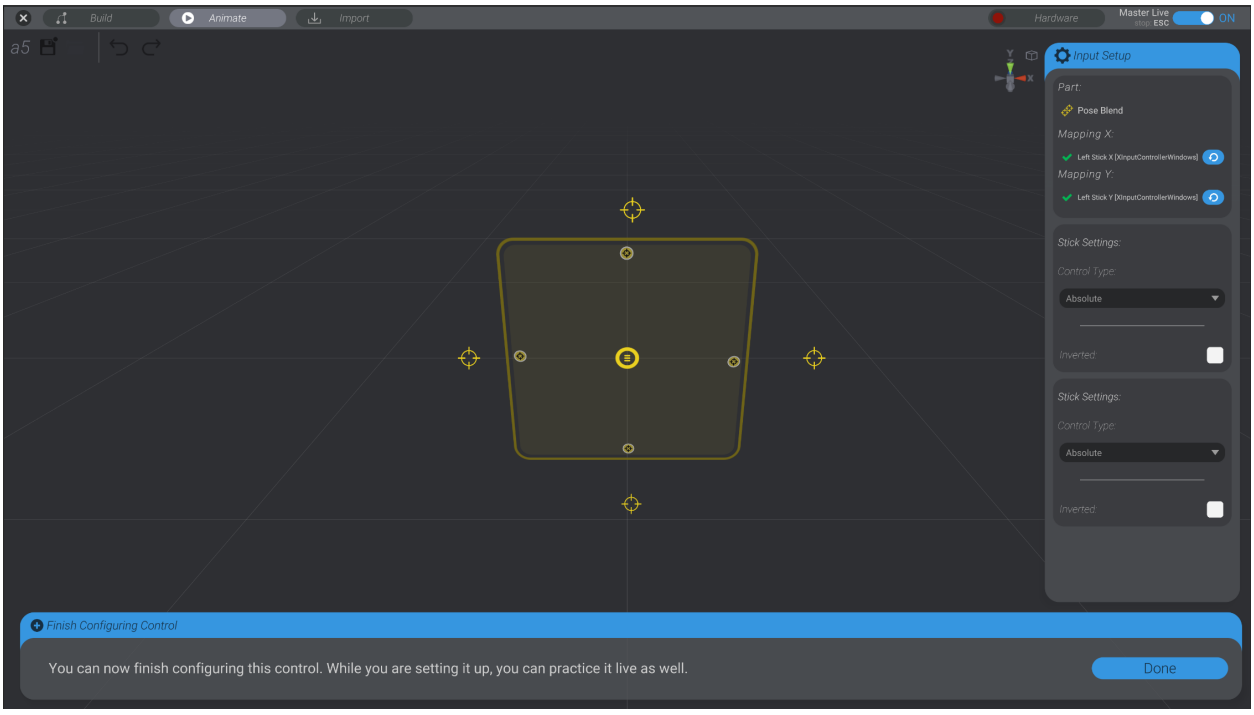
Control Type: Left/Right/Down/Up Only Absolute
In this control type, your input on the stick will be mapped to a range of your choosing of the selected part. However, only the given half of the stick input will be used. For example, if you use the x axis of a stick, and select left only. Any movement to the right past neutral on the stick will be treated as neutral.

Unique settings for 2D Pose Blends

When selecting the part for your control, if you select a 2D pose blend, you'll be able to set up both the x and y axis of the pose blend. The first input you set up will be for the x axis:



After which you can add an additional second binding for the y axis.



Each axis is independent of the other, so you can mix and match input types however you'd like to control your 2D pose blend.

Unique settings for On/Off and Trigger Events

Trigger events and on/off events behave a bit differently than motors, curved events, pose blends, etc. With a trigger event and an on/off event, there is really only an on or off state or a "fired" state. There is no analog or curved movement. As well, with on/off and trigger events, you can only use a digital or analog button, stick inputs are not supported for these part types.

For on/off events, you have the following control types:

Control Type: Hold
The event will be on while the button is pressed, and off when unpressed. Invert does the opposite of this behavior.

Control Type: Toggle
The event will alternate between on and off with each button press.

For trigger events, you have the following control types:

Control Type: Fire on Press
The event fire when the button is pressed.

Control Type: Fire on Release
The event fire when the button is released.

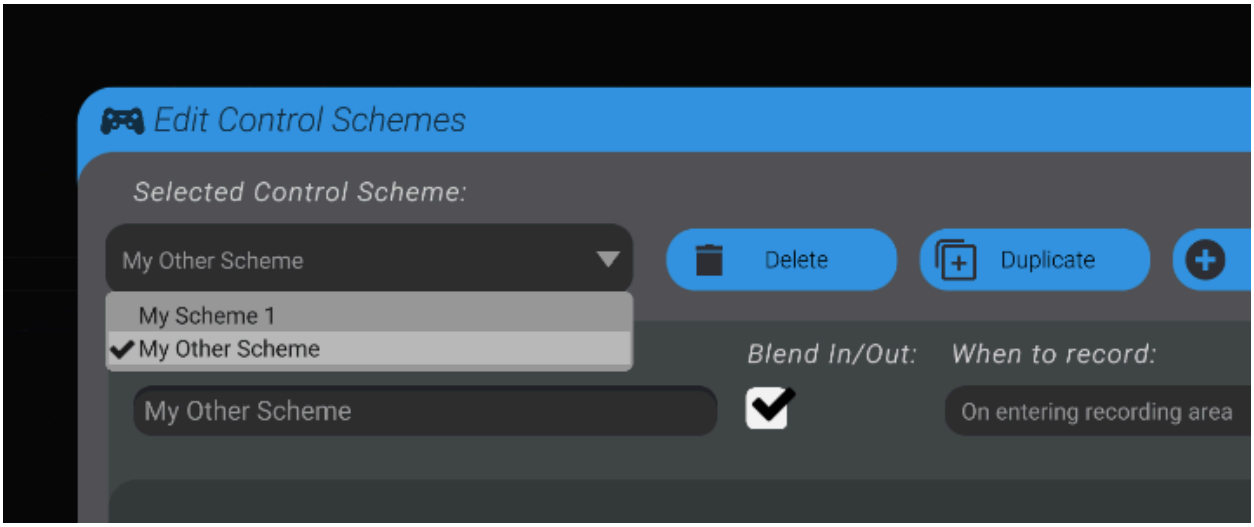
Control Type: Fire on Press and Release
The event fire when the button is pressed and when it is released.

For both on / off and trigger events, if you use an analog button, there will be an additional threshold value to determine at what point in the press of the analog button a "press" is counted towards changing on off state or firing the trigger event.

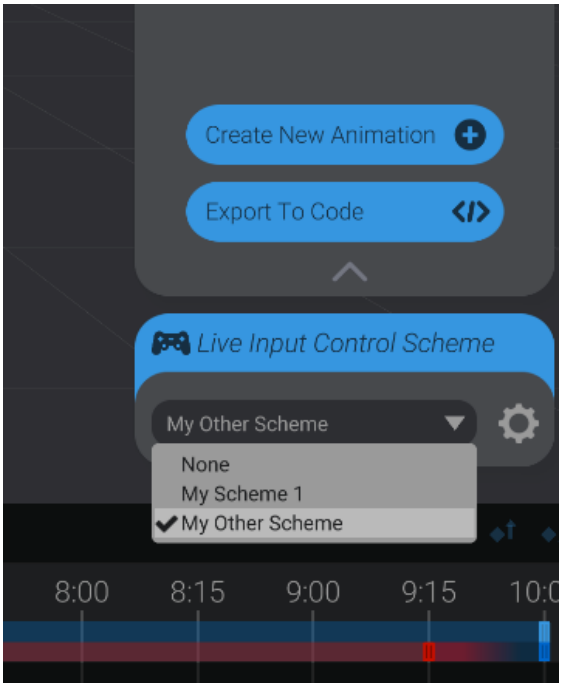
Using Multiple Control Schemes

As a reminder, a control scheme is a set of mappings between parts and inputs, and the associated settings. You can have as many different control schemes as you want in your project.

In the edit scheme menu, you can select which scheme is selected for editing:



And in animation mode, you can select from a dropdown which control scheme you'd like to have active for recording, or set it to none to disable recording entirely:



-19-
Animating with Inverse Kinematics

What's in this chapter

Every movement you've made so far in Bottango has been using something called "forward kinematics." Let's imagine a hand connected to an arm, and you want to reach the hand out to grab an apple. With forward kinematics, the way you've been animating so far, you would first position the shoulder, than the elbow, then the wrist, to put the hand in the right position.

Inverse kinematics is a tool to help you make movements like that more easily and naturally. With inverse kinematics, you place the hand where you want it, next to the apple, and Bottango figures out for you where to move the wrist, elbow, and shoulder for you to facilitate the desired position of the hand.

Inverse kinematics is the process by which Bottango calculates for you the parent joint positions needed to reach a desired child position.

The current state of inverse kinematics in Bottango

Though the above may sound pretty straightforward, there's actually a lot of complexity behind the scenes and divergent workflows available for using inverse kinematics (IK). What is released in Bottango now is just the first version of what will someday be a more robust inverse kinematics workflow.

As will be discussed later in this chapter, the current workflow uses IK to create keyframes on your joints. A future version will allow you to animate the target of an IK chain directly.

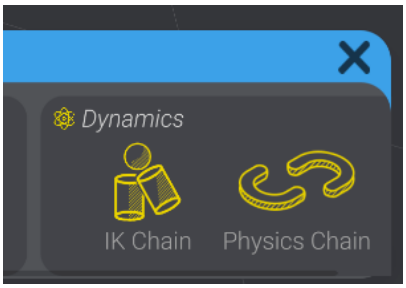
Setting up inverse kinematics

To set up an inverse kinematics chain, you'll need to identify the root of the chain and the tip of the chain. In the above arm example, the root would be the shoulder, the rotating point from which the chain originates, and the tip would be the hand, the part of the chain you want to try and get as close to the target position as possible.

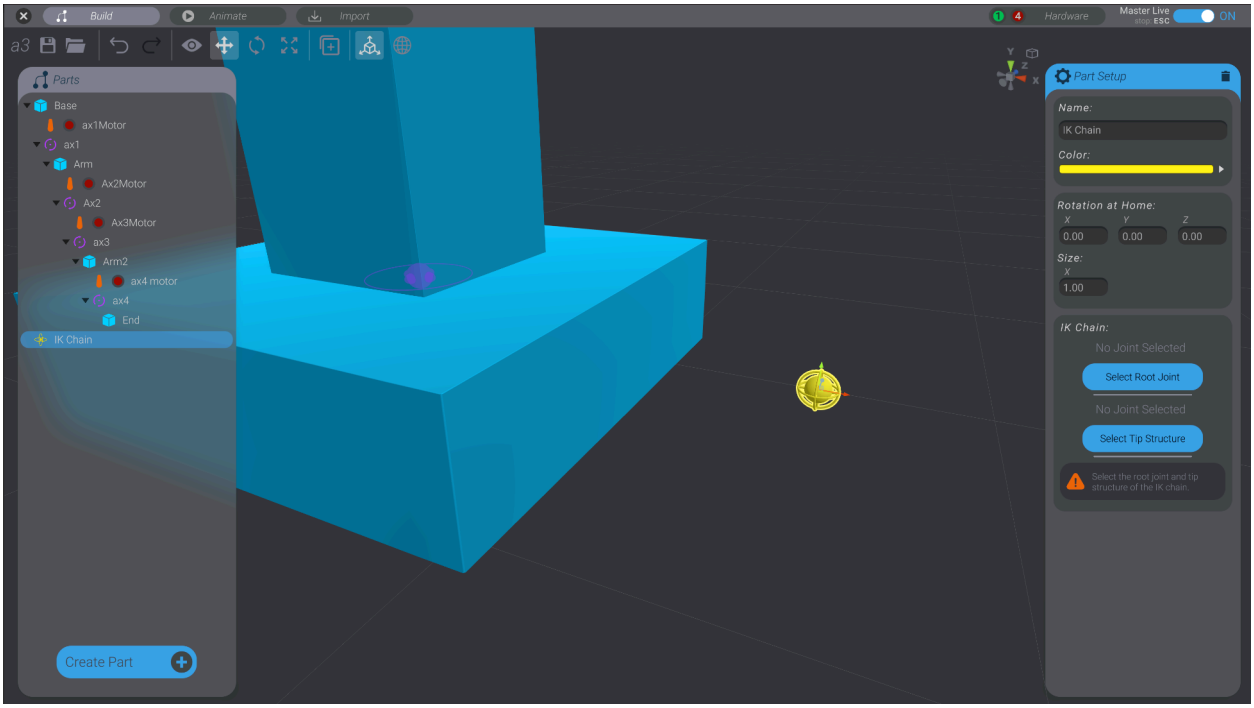
As well, you'll need a target, IE the movable point where you want your inverse kinematics chain to try and reach.

1. Click create part

2 Create an “IK Chain” from the “Dynamics” section.



This creates an IK Chain part in your project, which you can then configure.



3 Click “Select Root Joint” and click on the root joint of the IK chain.

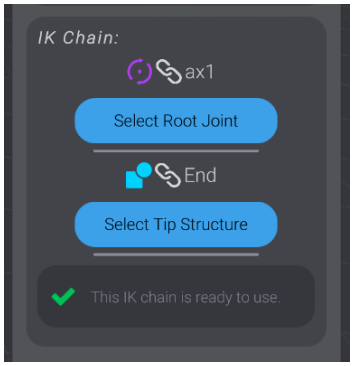
The root joint is the highest up parent that you want to move to try and reach the target when animating.

4 Click “Select Tip Structure” and click on the structure you want to act as the tip of the chain.

The tip structure is the structure in your IK chain you want to try and move to the IK target. The tip structure must have, at some level, the root joint as once of its parents.

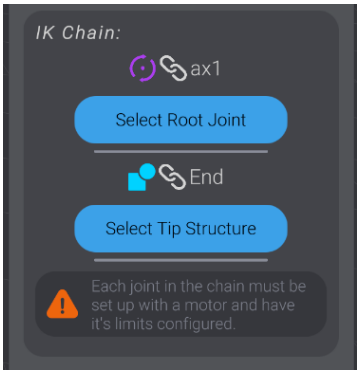
When you select a tip structure, all the joints between the root joint and the tip structure are automatically added to the chain. As well, the IK chain you created will move automatically to be in the same position as the tip structure.

If everything is correctly set up, the IK chain will show its status with a check and it's good to go status.



Fixing IK configuration errors

IK chains need to be set up in a very specific way, or else Bottango will report an error letting you know what changes need to be made to fix them. Let's go over what those errors are and how to fix them.



- ***"No joint can share a motor with another joint in the chain."***
  - Every joint in the IK chain must not be linked to the same motor as another joint in the chain. If two joints in the chain share a motor, change the linked motor on one of the joints.

- ***"Select the root joint and tip structure of the IK chain."***
  - You have not yet selected the root joint and/or the tip structure for the IK chain, or one of those previously selected parts was since deleted. Click the "select" button for the missing part.

- ***"The tip structure of an IK chain must be have the root joint as one of its parents."***

  - The tip structure in an IK chain must ave as one of its parents the root joint. Either select a new tip structure that is at some point a child of the root joint, or rearrange your project so that the tip structure is a child.

- ***"The IK chain cannot be a child of any part of the chain."***
  - The IK chain part is the target you will be moving around. If it is a child of any part of the chain, as the chain moves, the child will move, causing the chain to move, etc. This would be an infinite loop! Move the target to a place in your part's list that is not a child of the any part of the IK chain. For most cases, the IK chain part should be at the top level in your project.

- ***"The IK chain contains some of the same joints as another chain."***
  - Each joint in your project can only be in one IK chain. If your IK chain overlaps in joints with another IK chain, both will be invalid until that is fixed.

- ***"The IK chain contains some joints with a total angle > 360."***
  - Each joint in an IK chain must have a total range of movement at most equal to 360 degrees. Support for a wider range of movement will be added in a later version.

Animating with IK

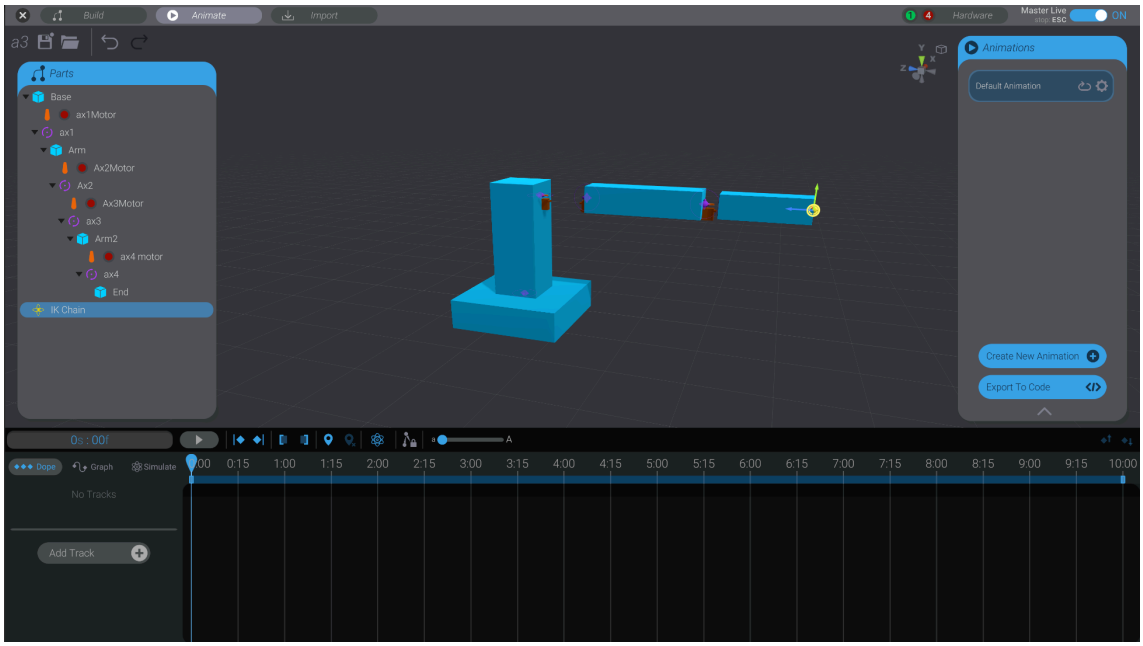
Once your IK chain is correctly set up, you can use it to animate your project.

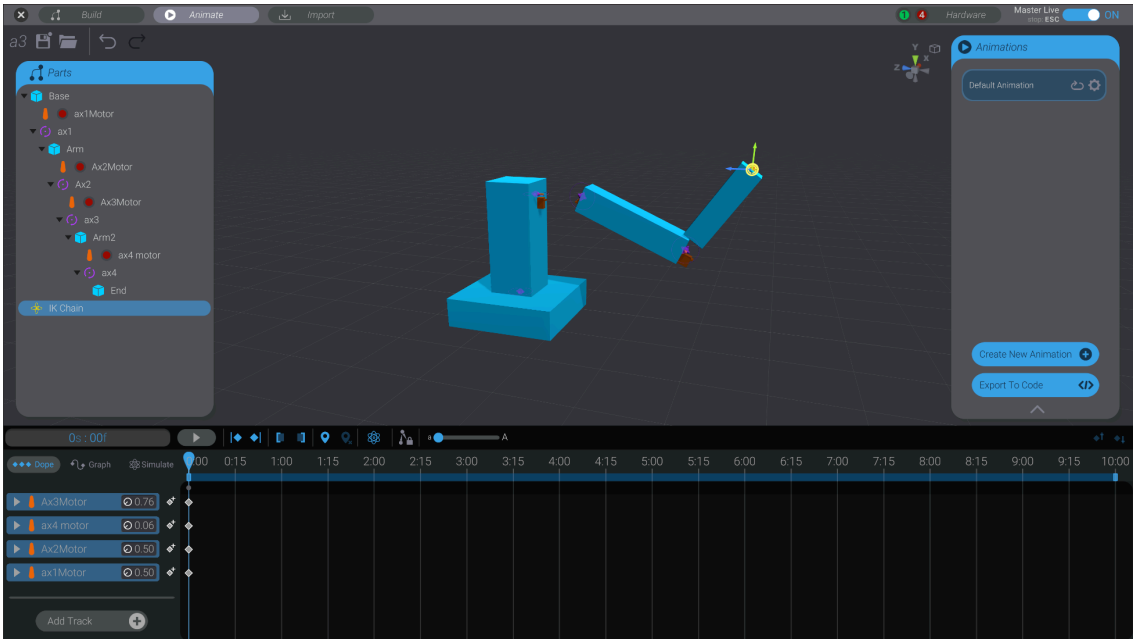
1. Enter Animation mode.

2. Select the IK chain part.

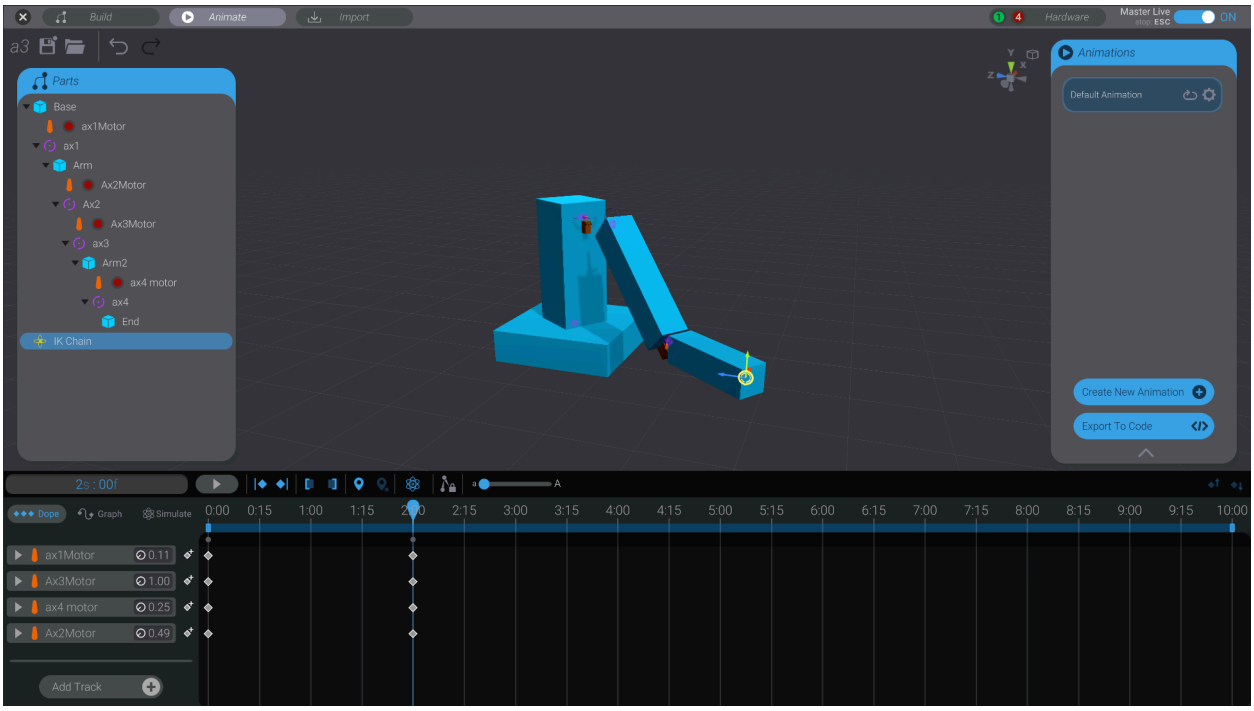
3. Move the IK chain part to where you want the tip structure of your IK chain to try and reach.

You'll see, as you move the IK chain part around, Bottango positions all the joints in the chain as best it can to try and place the tip structure in the same location as the IK chain part, and will create keyframes at the selected time as well.





If you move the animation scrubber to a different time, and then move the IK chain part to a new location, you'll create new keyframes at that new time.



You'll see as you scrub between the two sets of keyframes, you've created two target positions for your robot using IK and the IK chain part.

NOTE: In the current IK workflow, you create target poses with the IK chain part, and then animate between those poses. This is not the same thing as animating the chain target itself. Because of this, your robot may not, for example, follow a perfectly straight line while moving between the target poses. A future workflow in Bottango will allow you to animate the IK chain target itself.

The IK chain part only sets poses while you are dragging it. Unless you drag the IK chain part, you can animate identically to how you did previously (using forward kinematics). This means that even though you have an IK chain set up, you can still move individual joints. The IK chain will only try and set positions if you drag on the IK chain part itself. This means you can still edit keyframes and animation curves as you previously had, even if they were created by moving an IK chain.

Locking on to an IK Target Position

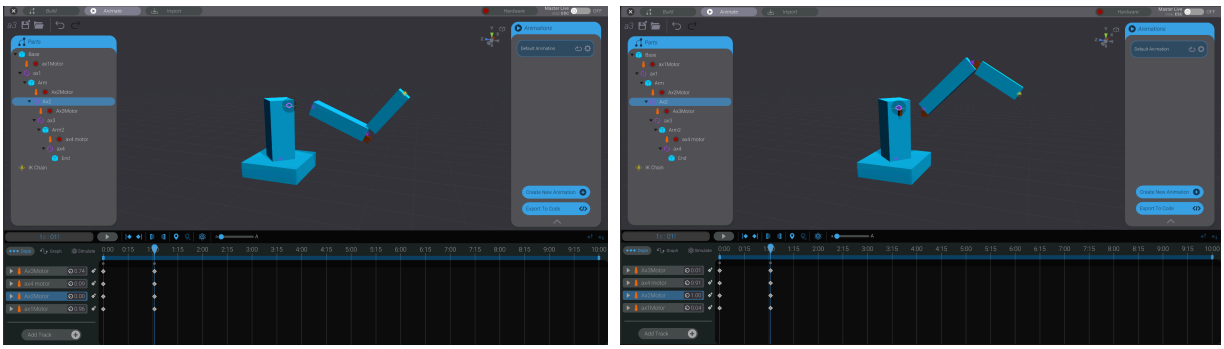
What if, for example you wanted to keep the tip structure of an IK chain reaching towards the target position, but move or animate the parent joints further up the chain? In the arm, hand, and apple example, this would be keeping the hand in the same position touching the apple, while moving the shoulder and elbow around.

You can do that in Bottango too, using IK!

1 Select the IK part you want to make "stay on."

2 Click the IK lock button in the animation tools to enable it.

While IK lock is on for an IK part, any movements you make on one joint in the IK chain, will cause all the other joints to move to try and reach the IK target.



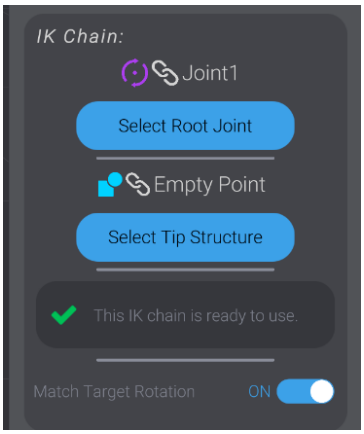
You'll also see a "Lock" icon next to the IK chain in the parts list when this functionality is enabled for an IK chain part.

Select an IK Chain part that has lock enabled, and click the lock button again to disable it.

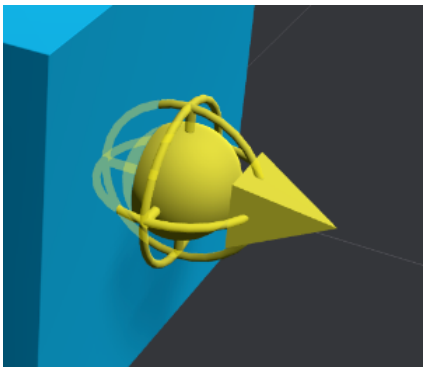
Setting IK End Point to Match Target Rotation

Another way you may want to constrain your IK chain is to have the end point of the rig try and match the rotation of the target. This can be useful for something like a camera arm, where you want to control not just the movement of the target, but keep the camera pointed in a certain direction.

At the bottom of the IK Configuration panel, when you have an IK Chain selected in configure mode, is a toggle to "Match Target Rotation."



This is disabled by default. When you enable it, you'll see an orientation hint show up on the IK chain target:



With "Match Target Rotation" enabled, in animate mode, when you rotate the target the end point of the chain will try and maintain that same rotation as the IK chain solves for a target pose.

-20-
Controlling Bottango with your own Scripts and Code

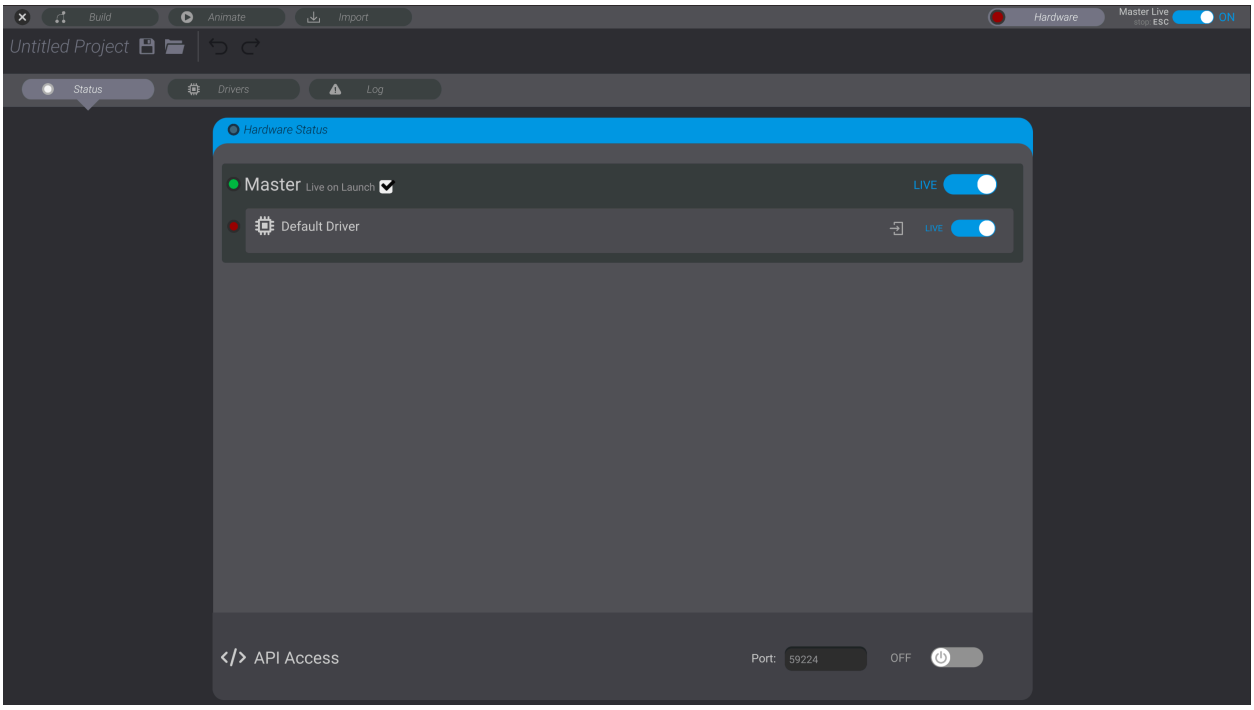
What's in this chapter

Bottango contains the beginnings of a scripting API to allow you to control Bottango from your own scripts, applications, and other related use cases. This could be used to integrate Bottango with your own or other robotic control applications. Or it could be used for you to make a hardware controller to start and stop different animations at a physical button press.

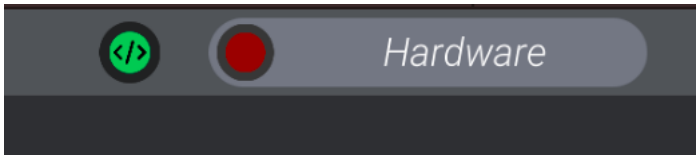
API control right now is oriented around getting information on and controlling the playback state of Bottango. But it's just an early version, and will continue to expand.

Enabling API control

You can enable the API access in a project by toggling the "API Access" toggle in the hardware menu:



Once it is enabled, you will see an "API" notification in the top bar as well:



Api control works as a local server, responding to GET and PUT requests in a REST API. You can change the port of the local server in this same menu.

More Information and API Documentation

Included in the download folder with Bottango is a "Playback API" folder. In that folder you'll find two things to help you use API control of Bottango:

- BottangoPlaybackApiDocumentation.pdf - Much more complete documentation of the API. This chapter in this documentation serves only to point you in that direction. The API has it's own documentation located in that folder.
- PlaybackAPIExample.py - An example API script making each supported call to the Bottango API.

-21-
Networked and Custom Hardware Drivers

What's in this chapter

The most common workflow when using Bottango is to upload the provided Arduino compatible C++ Bottango driver code onto a Arduino compatible microcontroller board, and then connect to that board via USB and Serial. However, if you're comfortable scripting yourself, there are other options for how to communicate between Bottango and your robot. Two options are as follows:

The Bottango driver API is fully documented and open source, and you're able to modify or entirely reimplement the driver code based on that API.

The Bottango Networked Driver allows you to communicate with the driver API over a networked socket connection instead of USB serial. Bottango provides a fully functional implementation of the networked driver API written in python. This can serve as both an example for writing your own driver, or as a functional driver implementation on it's own.

Implementing your own custom driver

Provided in the download folder of this application is a full documentation of the Bottango Driver API in the following directory: Bottango/AdvancedFeatures/BottangoDriverAPI.pdf.

You are welcome to create your own driver implementation in whatever language and target platform your use case requires, using that documentation and the C++ and Python drivers as examples. You can communicate with the driver you create from the Bottango application over a USB serial connection or through a socketed network connection.

Networked drivers

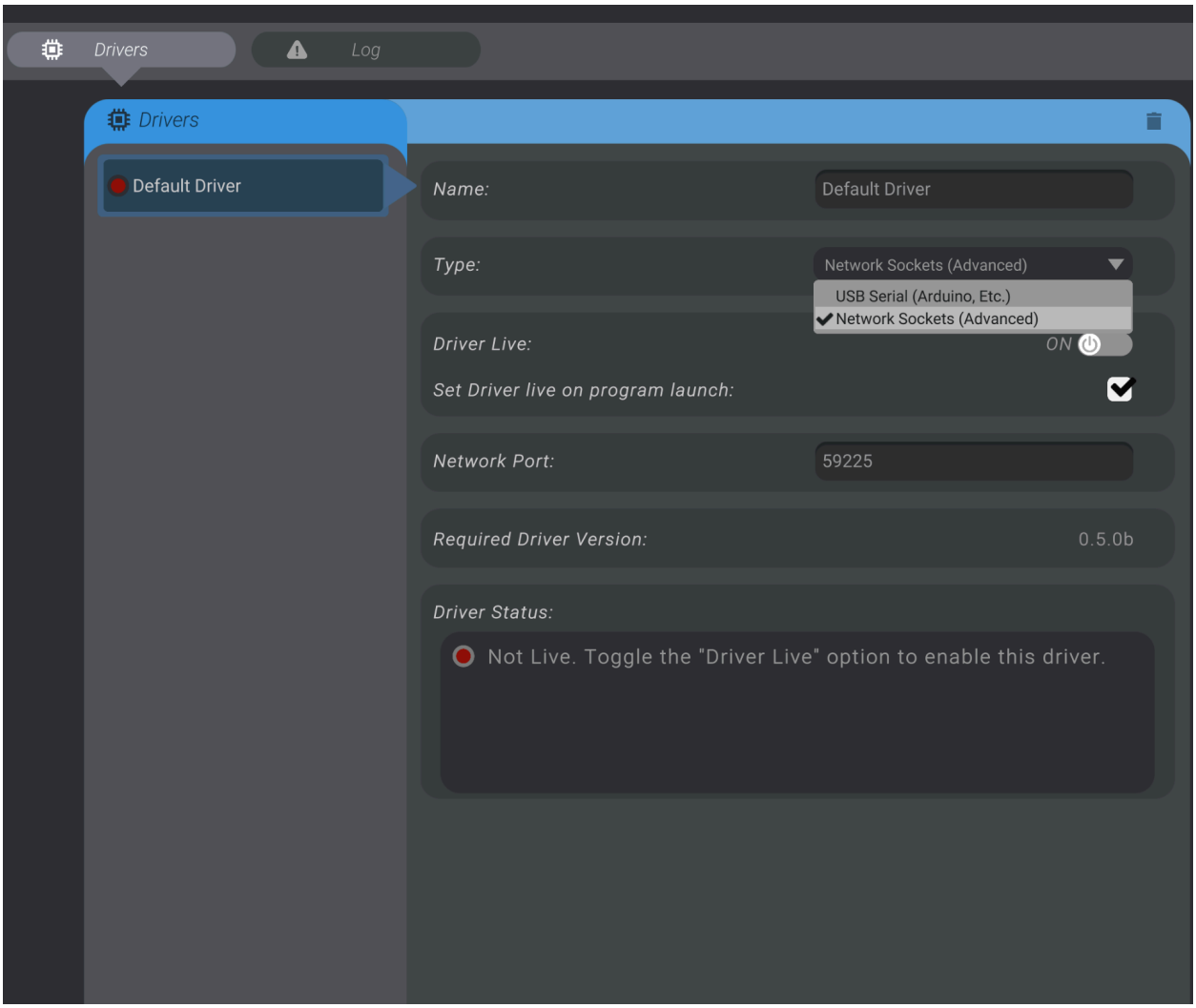
Besides serial over USB connections, you can change any driver in the Bottango application to communicate instead over a socketed network connection.

Some example use cases where the Bottango Networked Driver makes sense:

- I want to communicate from the Bottango app to my own code running on a computer.
- I want to control a motor that requires a connection to a laptop/desktop computer instead of a microcontroller.
- I want to control a robot with Bottango wirelessly, and I'm able to use a Python compatible microcontroller, or can rewrite the C++ code to create and maintain a socket connection for streamed data (providing a networked variant of the C++ code is on the Bottango roadmap, but not yet delivered)

Using a Networked Driver

In the driver menu, you can change any driver from USB serial to networked by changing the "type" dropdown from "USB Serial (Arduino, Etc.)" to "Network Sockets (Advanced)."



Once you do so, you'll be able to set the port that the socket server will be open on (59225 by default) when the driver is set live.

When you set the driver to live now, it will start a server on your local network and look for incoming client connections. In order to connect to this server, there is fully functional networked client code, written in Python. Provided in the download folder when you downloaded Bottango is a separate documentation PDF with instructions for how to use the Python code, and advice on how to modify it or create your own networked driver code from scratch.

For more information, the example Python code, and additional documentation, look in the following directory in the downloaded Bottango folder: Bottango/AdvancedFeatures/Driver_Networked.

-22-
Exporting Animations To Code

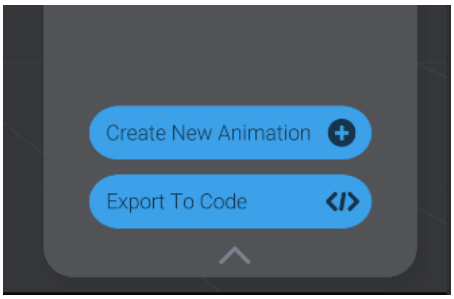
What's in this chapter

The most robust and easiest way to control your hardware with Bottango is to keep your hardware connected to your computer, and to control your hardware in real-time. A lot of care has been taken in the development of Bottango to ensure that live, real-time control is as performant as possible. In general, if keeping a computer connected is an option, that is the recommended workflow.

However, it may not always be practical to keep a computer controlling your hardware. Bottango offers the ability to export out animations to generated code, that you can upload to your hardware, and playback without computer control. Again, please not that this does NOT provide any performance benefits, but instead is intended as a convenience when live computer control is not possible.

Generating code

Code generation happens from the animation view. In the window with the list of all animations in your project, there is an "Export To Code" button.



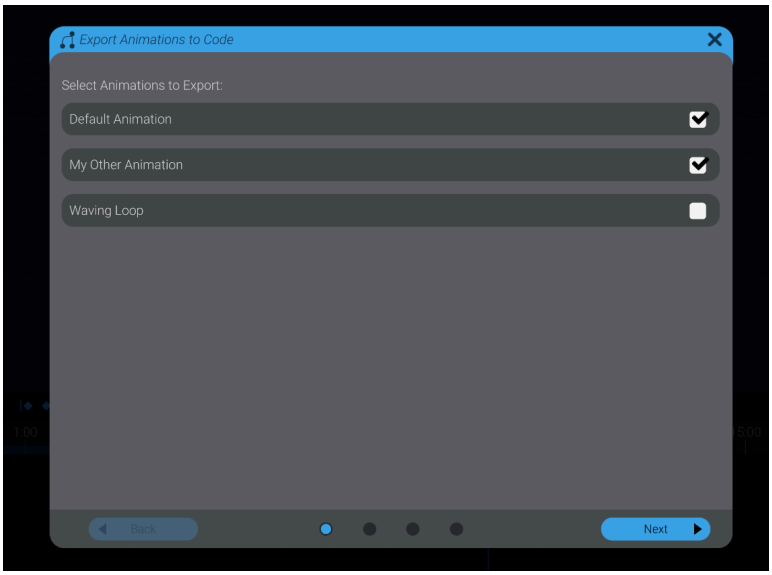
Clicking this button brings up the exporting workflow, which has four steps:

1. Select the included animations.
2. Select the included effectors.
3. Configure settings.
4. Export.

Generating code - Included Animations

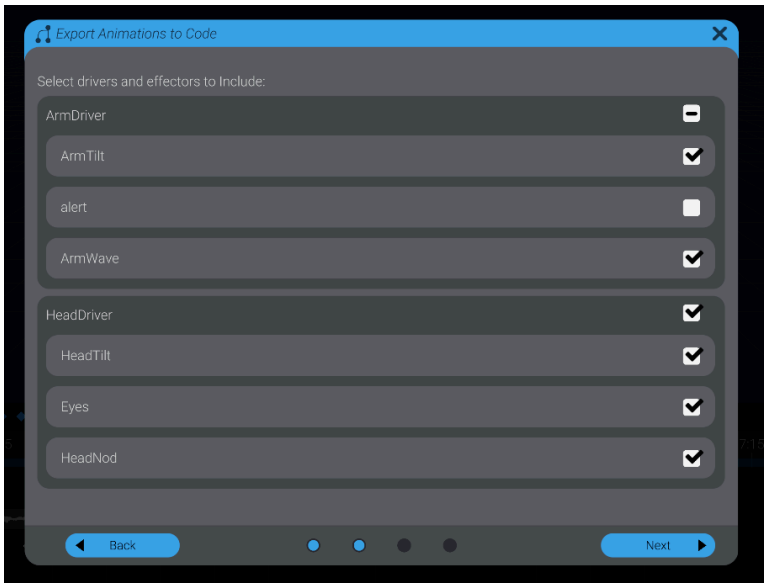
The first step of exporting animations to code is to select which of the animations in your project you want included in the code. Because of the limited memory size of hardware controllers, you should take care to only export the animations you care about.

Press next to continue to the next step.



Generating code - Included Drivers and Effectors

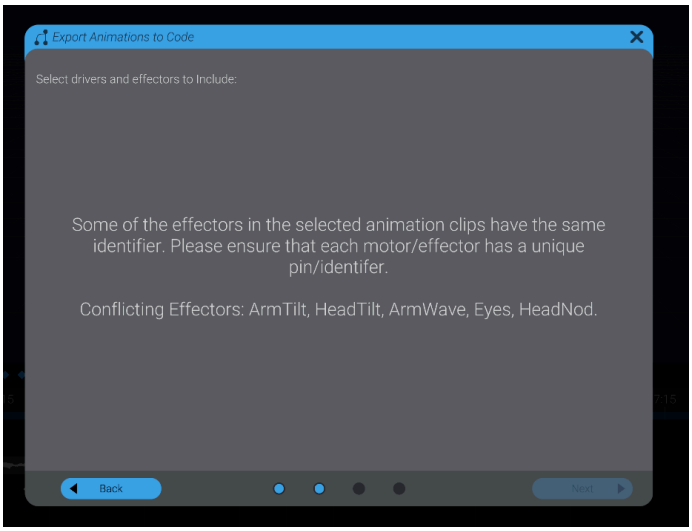
Based on the selected animations, all possible drivers and effectors are shown to select to include in the exported code. If you have a driver or animation that is not included in any of the selected animations, it will not be shown.



When you export out code in a project with multiple drivers, you will export unique code out for each individual driver. This is so that each driver only has the code it needs for the motors and other effectors that are registered to it.

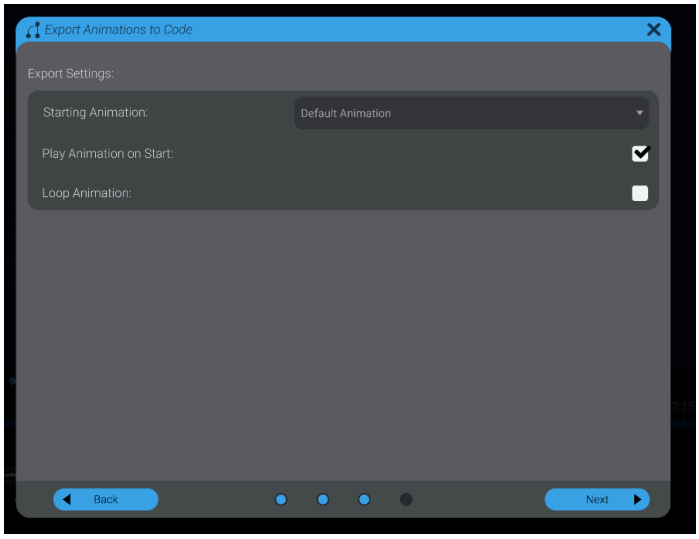
If you don't include an effector, none of the animations for that effector in specific will be included in the exported code. If you don't include a driver at all, you will not export code for that driver.

As well, Bottango will run a check to see if any of the effectors will conflict with another effector. This can happen if two effectors have the same pin or identifier. In Bottango usually, Bottango will report an error if you try and set a conflicting effector live. However, since Bottango won't be running, there is no opportunity to receive that error. Instead, we prevent you from exporting out faulty code:



Generating code - Configure Settings

There are a few helpful settings you can configure when exporting code.



The starting animation determines the starting position of all effectors when the code first begins running. Even if you are not going to play back an animation immediately at start, this tells the code where each effector should begin from.

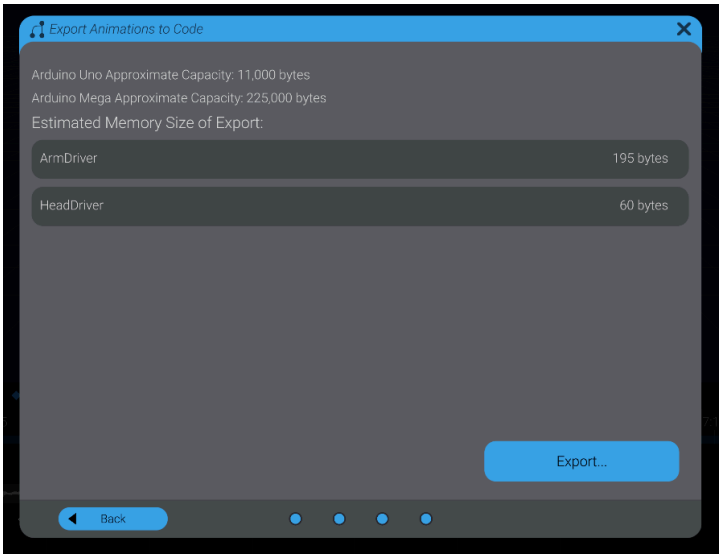
"Play animation on start" and "Loop animation" are helpful options to automatically configure the playback logic in the exported code. However, you are able to implement any logic you want on your

hardware, by following the examples in the commented exported code, and the exported "How To Use This Code" read-me.

For example, it's possible to respond to real world button presses to trigger a specific animation, or to start a cue of animations one after the other. Read the commented exported code and "read-me" file fore more information.

Generating code - Final Export

The final step is to export the code itself. Before you do so, you'll be shown the approximate memory size for each driver:



As well, you can see the current estimated capacity on two popular Arduino models (Arduino Uno and Arduino Mega) after taking in to account the rest of the Bottango driver memory needs. As you can tell, memory capacity is not infinite, and you really don't want to get too close to full for stability reasons. In a future version, Bottango will support exporting and playback from an external storage device, such as an SD card.

If you have only one driver to export, Bottango will place all exported files in the selected location. If you have more than one driver to export, Bottango will create a folder for each driver, and place the appropriate exported code for each driver accordingly.

Generating code - Some limitations to be aware of

As stated above, the preferred workflow whenever possible is to control your hardware in real-time with a USB connection. If you do export out code, be aware of some of the following limitations:

- You are limited in the storage capacity of a microcontroller. You can only put so many keyframes on a microcontroller in exported code. In a future version, this will be improved by allowing for exporting and playback from an external device such as an SD card.
- There is not currently a workflow for synchronizing stepper motors without using a Bottango USB connection. You should ensure on your own that each stepper motor is in the correct position for the start of an animation you intend to play.
- Looping animations is supported, the code is smart enough to move effectors back to start positions before repeating the animation if it is not a perfect loop. HOWEVER, there is not currently support for seamless blending from the end of one arbitrary animation to the beginning of another, if the two animations do not already line up. This functionality is planned for a future version. Until then, if you start playing an animation, the code will assume that each motor is where it should be right at the start.

How to use generated code

The result of the exported code is a handful of files. As well, there will be a "read-me" text file. Please always check that text file, in case the instructions have changed. If there is a conflict between this manual and that text file, go with the "read-me" text file that is exported.

In order to use this generated code, you will need to do two things:

1) In the Bottango Arduino folder, you will find the file "BottangoArduinoConfig.h." You will need to uncomment the line in that file that has the definition "USE_COMMAND_STREAM." This tells the code to read commands from the generated code, instead of from your computer over USB.

To uncomment a line, simply remove the "//" characters from the beginning of that line.

2) Drag the two generated files ("GeneratedCommandStreams.cpp" and "GeneratedCommandStreams.h") into the Bottango Arduino folder.

Once you have done these two things, re-upload the BottangoArduino.ino file to your Arduino. To go back to live mode, reverse these two steps and then upload to your Arduino again.

Monitoring over USB connection

When you uncomment the "USE_COMMAND_STREAM" definition, that puts your hardware in "listen-only" mode, and it will ignore all commands that it receives over USB, if you happen to plug it in. That means that you cannot issue new animation commands, motor registration commands, or stop commands. You should follow the example code and provide a safe way to execute the "Stop()" function on your hardware if you are using generated code.

While connected over USB in listen mode, you can still see the log of what the driver is doing in the hardware view. This can be helpful for troubleshooting or for support.

-23-
Wrapping Up

Thank you!

First and foremost, thank you! I am so excited to create a tool that enables creative people to better express their vision. There's a lot of work ahead for Bottango, but by reading through this documentation, participating in the beta, and helping shape Bottango's future, you're helping forge the path of what I hope becomes an invaluable tool for creative hardware expression.

You can actively participate in the future of Bottango by joining the official Discord group and discussing Bottango with me and other Bottango users: https://discord.gg/6CVfGa6

If you have support needs or questions, the Discord group is the best place to reach out.

I'm excited to see what you use Bottango for. Share your projects in the above Discord group, or by e-mailing me at Contact@Bottango.com. The roadmap for Bottango is still very much a work in progress. The more you reach out, share what you're working on, and make feature requests, the more I can shape Bottango into what you want it to be.