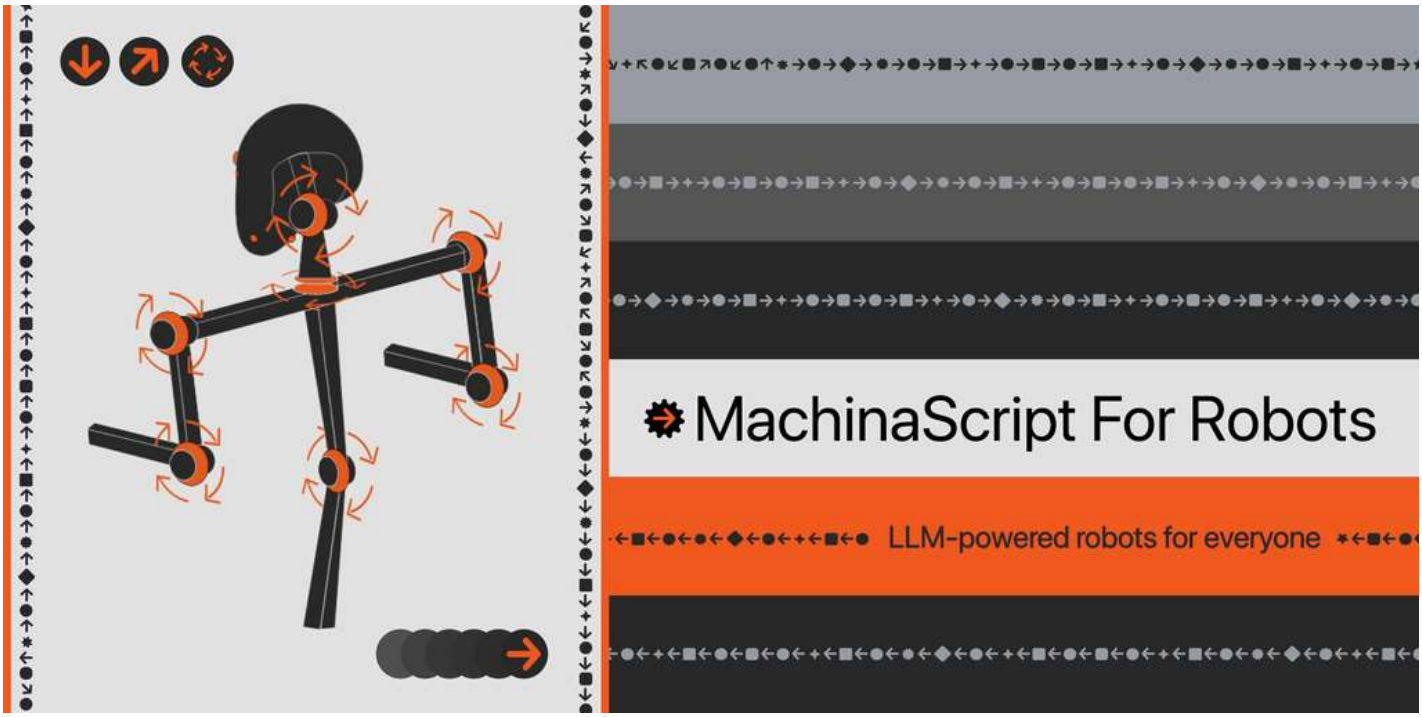
AUTODESK
Instructables

Build Fully Automated GPT-Arduino Robots With MachinaScript

By babycommando in CircuitsRobots

Introduction: Build Fully Automated GPT-Arduino Robots With MachinaScript



Howdy hackers! I am babycommando, robot maker and author of the MachinaScript For Robots project.

> *MachinaScript is a simple framework that enables anyone with a computer and an arduino to build a GPT or local-LLM p owered robot in their garages right now.*

What we will do:

In this project we are building a simple version of **MACHINA1**, a two-servos example that simulates the movement of a robot head (l ooking up and down, left or right) powered by GPT or any other LLM of your choice.

MachinaScript enables you to build robots that execute actions, make movements and using skills in style of function calling. In this example we are making a voice-command prompt that will provide the robot a starting point. From this model you can explore a self-prompting robot as well.

Anakin built C3PO when he was 9 years old, how about you?

Supplies

- Two servo motors
- An arduino uno
- A computer-like device (laptop or raspberry pi) with a microphone
- A protoboard
- The latest version of python3
- The latest version of the Arduino IDE
- The MachinaScript Framework from the github repo here (step 3)

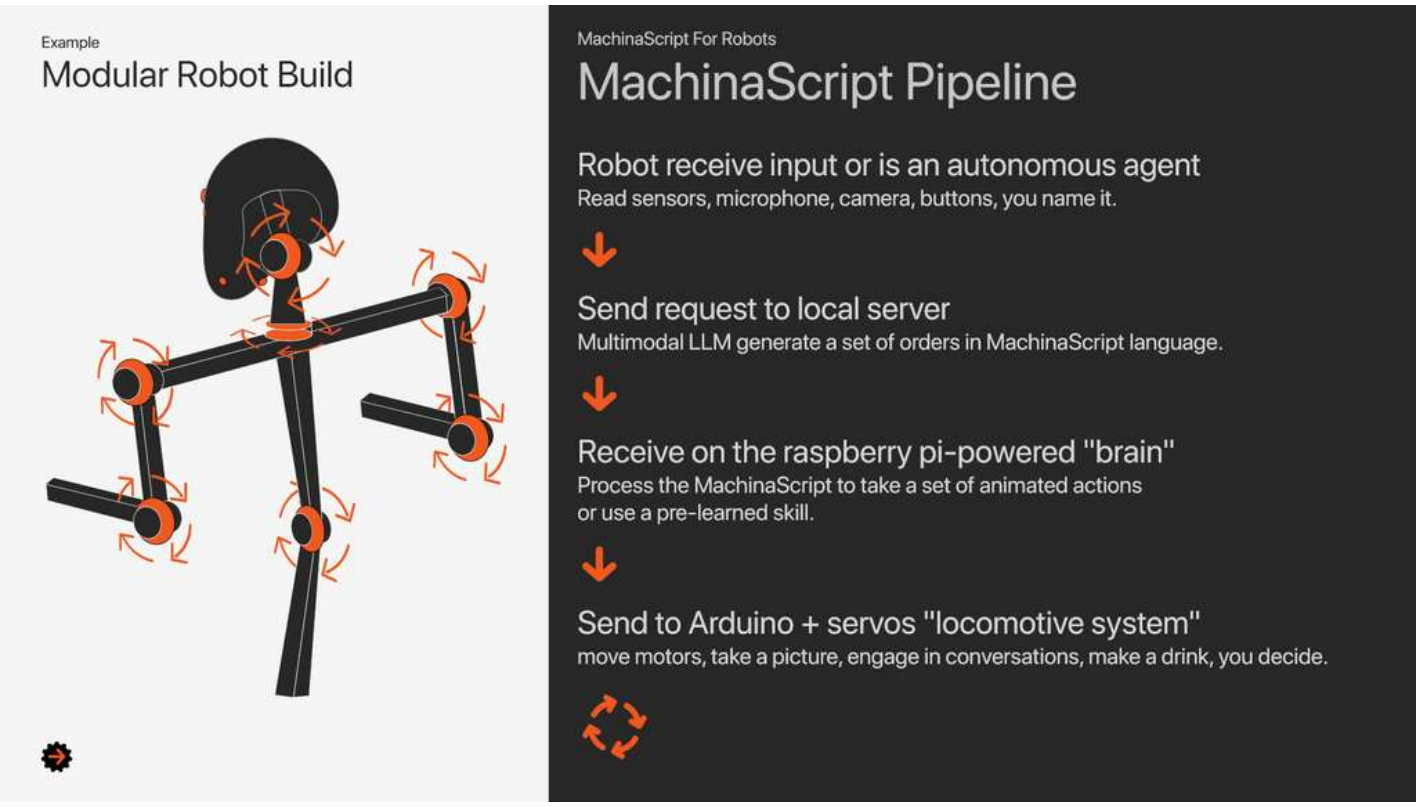
Step 1: Meeting MachinaScript for Robots

MachinaScript for Robots is a set of tools and a LLM-JSON-based language designed to empower humans in the creation of their o wn mechanical companions. MachinaScript facilitates the **animation of generative movements**, the integration of **motion persona lity**, and **teaching a set of instructions (skills)** with a high degree of autonomy. With MachinaScript, you can control a wide range of electronics like Arduinos, Raspberry Pis, servo motors, cameras, sensors, and much more. It makes a wide modular generative fo rmat more accessible, as no robot design is the same for anyone.

There's a lot of "***how to make arduino robots***" out there already. They work pretty much like remote-controller toys: robot dogs, hu manoids, R2D2 clones… We are just **switching the controls to a multimodal large language model**. And trust me, they perfor m **very** well on this. Motion design can display a lot of personality.

Before starting, notice this is a simple example easy to get started that can be expanded into any form or shape of robot, this is the beauty of MachinaScript For Robots.

Step 2: A Simple, Modular Pipeline



1. **Input Reception**: Upon receiving an input, the brain unit, (a central processing unit like a raspberry pi or a computer of your ch oice) initiates the process. For example listen for a wake up word, or a function to keep reading images in real time on a multi modal LLM.
2. **Instruction Generation**: A Language Model (LLM) then crafts a sequence of instructions for actions, movements and skills. T hese are formatted in MachinaScript, optimized for sequential execution.
3. **Instruction Parsing**: The robot's brain unit interprets the generated MachinaScript instructions.
4. **Action Serialization**: Instructions are relayed to the microcontroller, the entity governing the robot's physical operations like se rvo motors and sensors.

You can customize the complete pipeline according to your own designs.

Step 3: Getting MachinaScript

Getting MachinaScript

Clone this repository

```
git clone https://github.com/babycommando/machinascript-for-robots.git
```

Browse the code to understand the architecture

```
MachinaBody -> the arduino code for the robot's body
MachinaBrain -> the computer code for the robot's brain
```

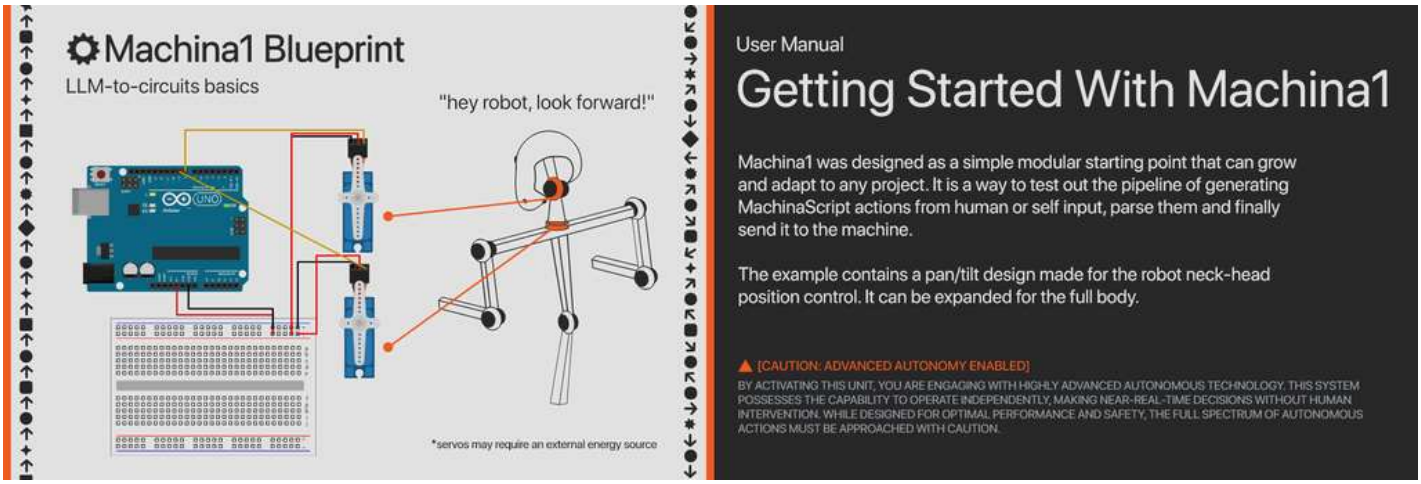
After cloning/downloading this repo, **make sure you have the latest version of Python3 and the Arduino IDE.**

Robot FileSystem: Body and Brain

The project is based in two different folders **-** the files for the "**brain**" (a computer) and the files for the "**body**" (the arduino).

```
MACHINA1
  MachinaBody
    machinascript_body.ino //Arduino code
    test_serial.py //Tests program
  MachinaBrain
    brain_openai.py //powered by GPT3.5/4/4Vision
    brain_local_llms.py //powered by Local LLMs
    brain_huggingchat.py //powered by huggingchat unofficial api
    machinascript_language.txt //system prompt
    machinascript_language_large.txt //system prompt large
    machinascript_project_specs.txt //project specs
```

Step 4: Assemble the Robot First



The easiest entry point is to start by programming your robot with Arduino code.

In this step we will:

- *Assemble your robot* and get it moving with simple programmed commands.
- *Modify the Arduino code* to accept dynamic commands, similar to how a remote-controlled car operates.

Let's go!

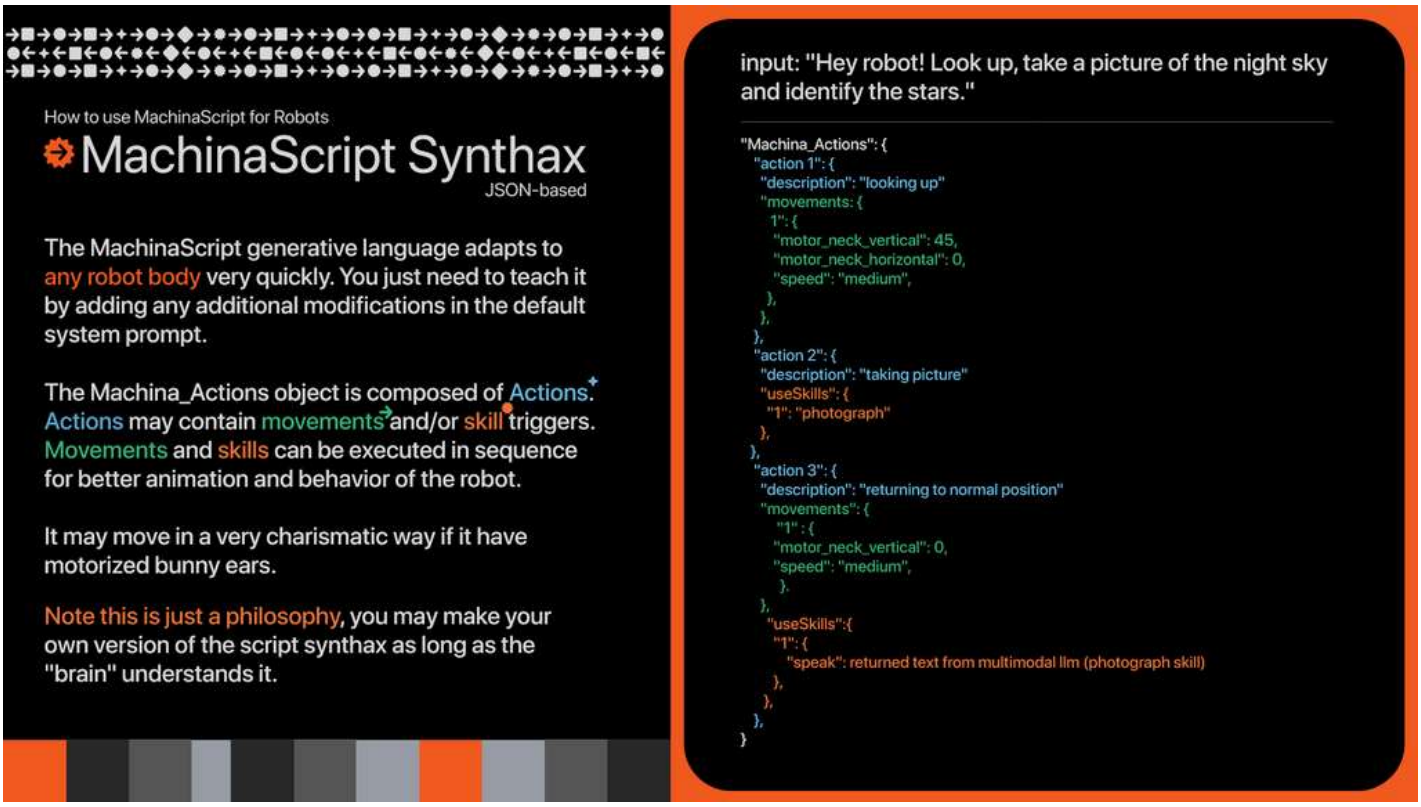
1. Hook two servos to your arduino as showed in the image above to be the neck_horizontal and neck_vertical servos, working a s a pan/tilt base. With it you can test commands like *"look up"* or *"say yes moving your head"*.
2. In your Arduino IDE choose the USB port you want to work and select your board, then test and inject the code in your little ba by.
3. Testing your robot build:

- Test A) Test the code by sending serial commands **via the arduino IDE** in this format: MotorID:degrees,speed;. For example:

```
A:45,10;B:0,10;
```

- ***A and B*** means the motors
- ***45 and 0*** means the position in degrees for the motor to move to
- ***10 and 10*** to be the velocity of the movement **-**; to be the separator that pipes multiple motors movements at the same time
- Test B) Test the code by sending serial commands via a **python script "test_serial.py"** (Note: edit the code making sure to se lect the correct USB port.)

Step 5: MachinaScript LLM-JSON-Language Basics



The MachinaScript language LLM-JSON-based synthax is incredibly modular because it is generative. It is composed of three major nested components: Actions, Movements and Skills.

Actions, Movements, Skills

Actions: a set of instructions to be executed in a specific order. They may contain multiple movements and multiple skill usages.

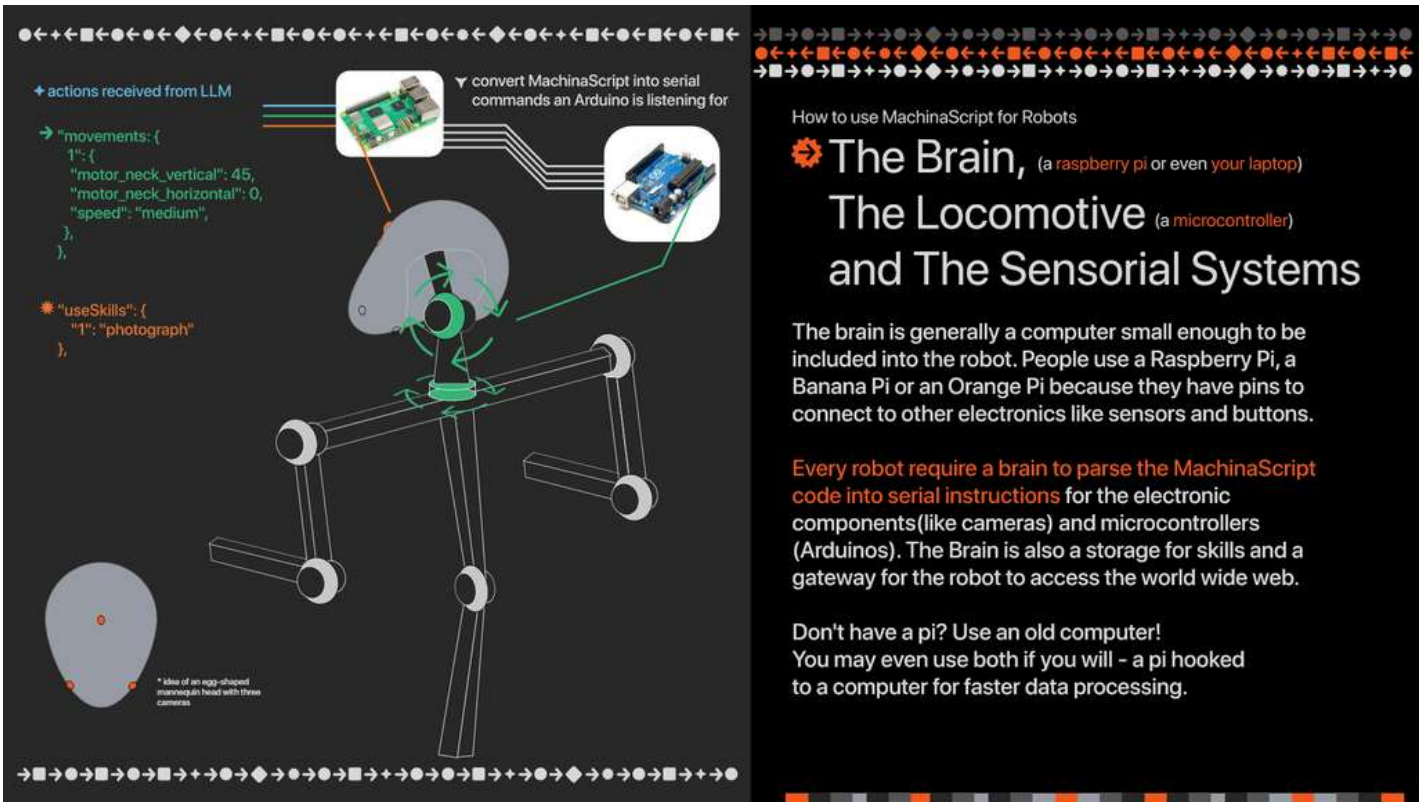
Movements: they address motors to move and parameters like degrees and the speed. This can be used to create very personal ani mations.

Skills: function calling the MachinaScript way, to make use of cameras, sensors and even to speak with text-to-speech.

As long as your brain unit code is adapted to interpret it, you have no ending for your creativity.

In the image there's example of the complete language structure in its current latest version. Note you can change the complete synt hax for the language structure for your needs, no strings attached. Just make sure it will work with your brain module generating, par sing and serializing.

Step 6: Choosing a Brain for Your Unit



There are three kinds of brains currently to power your robot:

```
brain_openai.py //powered by GPT3.5/4/4Vision
brain_local_llms.py //powered by Local LLMs
brain_huggingchat.py //powered by huggingchat unofficial api
```

Choose the correct brain for your project design.

The brain module consists of several components that make the complete pipeline possible.

```
receive input -> LLM generates machinascript -> parse the machinascript for acions, movements and skills -> serialize to the body
```

During the LLM text generation, a piece text composed of two parts is added to the system prompt:

```
machinascript_language.txt or machinascript_language_large.txt
+
machinascript_project_specs.txt //project specs
```

Choose the correct language file for your project. Larger may produce more accurate results, but a bit slower because have more w ords to be tokenized.

Teaching the LLM about your unique robot design - and personality.

No artisanal robot is the same. They are all beautifully unique.

One of the most mind blowing things about MachinaScript is that it can embody any design ever. You just need to tell it in a set of sp ecs what are their physical properties and limitations, as well as instructions for the behavior of the LLM. Should it be funny? Seriou s? What are its goals? Favorite color? The machinascript_project_specs.txt is where you put everything related to your robot person ality.

For this to work, we will append a little extra information in the system message containing the following information:

```
Project specs:
{
  "Motors": [
    {"id": "motor_neck_vertical", "range": [0, 180]},
    {"id": "motor_neck_horizontal", "range": [0, 180]}
  ],
  "Skills": [
```

```
    {"id": "photograph", "description": "Captures a photograph using an attached camera and send to a multimodal LLM."},
    {"id": "blink_led", "parameters": {"led_pin": 10, "duration": 500, "times": 3}, "description": "Blinks an LED to indicate action."}
  ],
  "Limitations": [
    {"motor": "motor_neck_vertical", "max_speed": "medium"}
    {"motor speeds": [slow, medium, high]}
  ]
  Personality: Funny, delicate
  Agency Level: high
}
```

note the JSON-style here can be completely reworked into any kind of text you want. You can even describe it in a single paragraph if you feel like. However for sake of human readability and developer experience, you can use this template for better "mental mappi ng" your project specs. This is all in very early beta so take it with a grain of salt.

Step 7: Declaring Specs: Teaching the LLM About Your Unique Robot Design - and Personality.

Teaching MachinaScript to LLMs

The project was designed to be used accross the wide ecosystem of large language models, multimodals and non-multimodals, locals and non-locals. Note that autopilot units like Machina2 would require some form of multi-modality to sense the world via images and plan actions by itself.

To instruct a LLM to talk in the MachinaScript Synthax, we pass a system message that looks like this:

```
You are a MachinaScript for Robots generator.
MachinaScript is a LLM-JSON-based format used to define robotic actions, including
motor movements and skill usage, under specific contexts given by the user.

Each action can involve multiple movements, motors and skills, with defined parameters
like motor positions, speeds, and skill-specific details, like this:
(...)
Please generate a new MachinaScript using the exact given format and project specifications.
```

This piece of code is refered as machinascript_language.txt and is recommended to stay unchanged.

Ideally you will only change the specs of your project.

To define a set of "rules" your robot MUST follow, declare them on machinascript_project_specs.txt:

```
{
  "Motors": [
    {"id": "motor_neck_vertical", "range": [0, 180]},
    {"id": "motor_neck_horizontal", "range": [0, 180]}
  ],
  "Skills": [
    {"id": "photograph", "description": "Captures a photograph using an attached camera and send to a multimodal LLM."},
    {"id": "blink_led", "parameters": {"led_pin": 10, "duration": 500, "times": 3}, "description": "Blinks an LED to indicate action."}
  ],
  "Limitations": [
    {"motor": "motor_neck_vertical", "max_speed": "medium"},
    {"motor_speeds": ["slow", "medium", "fast"]},
    {"motors_normal_position": 90}
  ],
  "Personality": ["Funny", "delicate"],
  "Agency_Level": "high"
}
```

Note that the synthax in this is still in very early beta, so there is a lot of exploration ongoing for this part. You may write literally anything in any format you want. The JSON formatting is just to make it more human-readable.

Finetuned Models

We are releasing a set of finetuned models for MachinaScript soon to make its generations even better. You can also finetune models for your own specific usecase too.

Animated Movements and Motion Design Principles

An action can contain multiple movements in an order to perform animations (set of movements). It may even contain embodied personality in the motion.

Check out Disney's latest robot that combines engineering with their team of motion designers to create a more human friendly machine in the style of BD-1.

You can learn more about the 12 principles of animation here.

Step 8: It's Alive! Now Give It a Goal

After successfully assembling your robot parts, making it work on the arduino "body" side and finally choosing your brain and editing your project specs, It's time to start the brain version of your needs.

Run the brain:

```
python3 brain_openai.py
```

Now wake it up with a wake up word and proceed by sending it a voice command related to your components, in this case a "robot h ead".

Try saying:

> Hey robot! (wait for it to listen)
>
> Look up!

This can grow very complex, for example giving a sequential set of instructions for the robot to perform, adding the use of skills or ev en by using a multimodal LLM that supports image analysis. You may modify the robot behavior to stay on a continuous generation "loop" and turn it into a singular automaton.

Step 9: Skills: Function Calling the MachinaScript Way

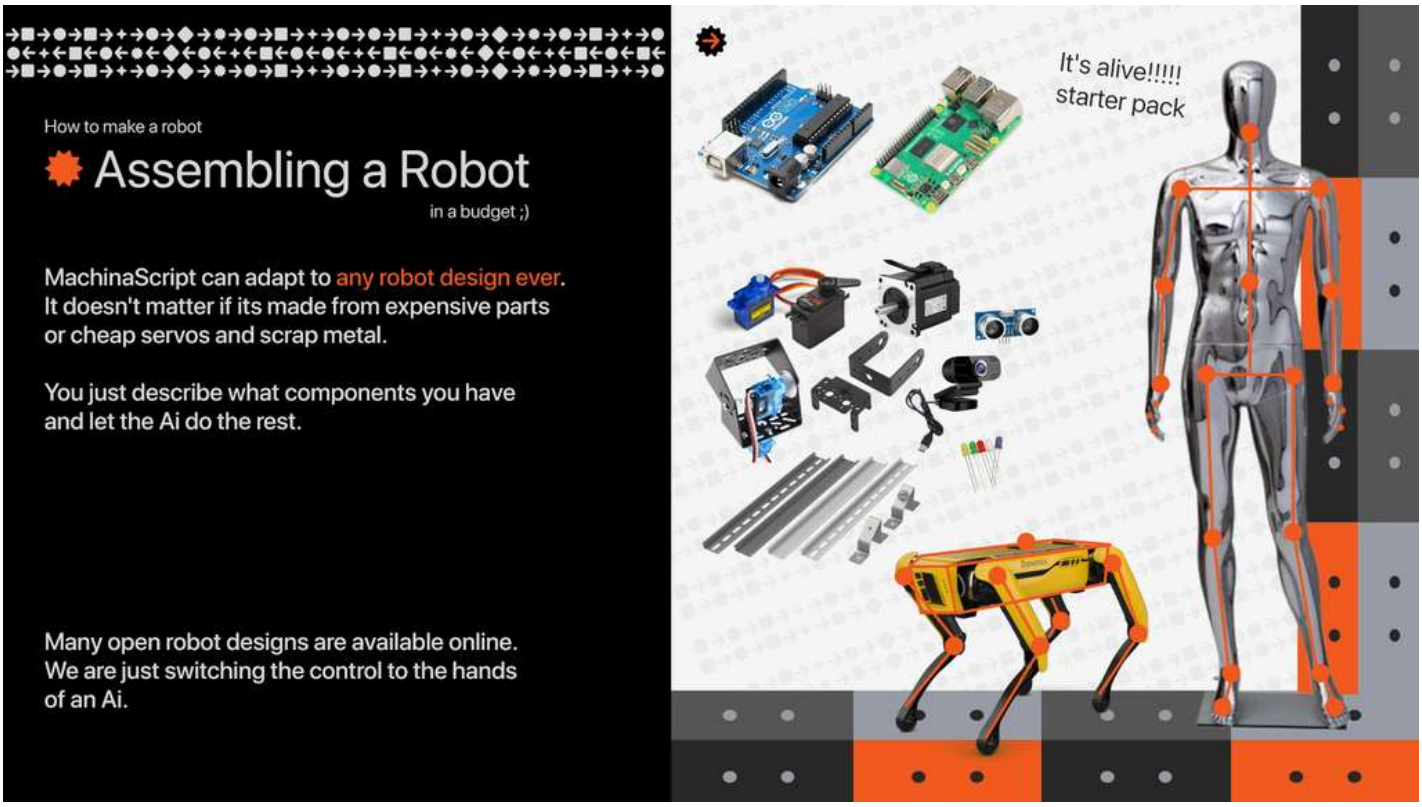
During the parsing of the actions in the python code, skills can be ordered to be executed.

You may declare skills as simple functions to be called when required.

```
def execute_machina_script(script):
    """Parses the MachinaScript and executes the actions by sending commands to Arduino."""
    actions = json.loads(script)["Machina_Actions"]
    for action_key, action in actions.items():
        # Check for 'movements' key and that it is not empty before execution
        if "movements" in action and action["movements"]:
            execute_movements(action["movements"])
        # Check for 'useSkills' key and that it is not empty before execution
        if "useSkills" in action and action["useSkills"]:
            execute_skills(action["useSkills"])
```

Check the brain code for a complete example of skill usage.

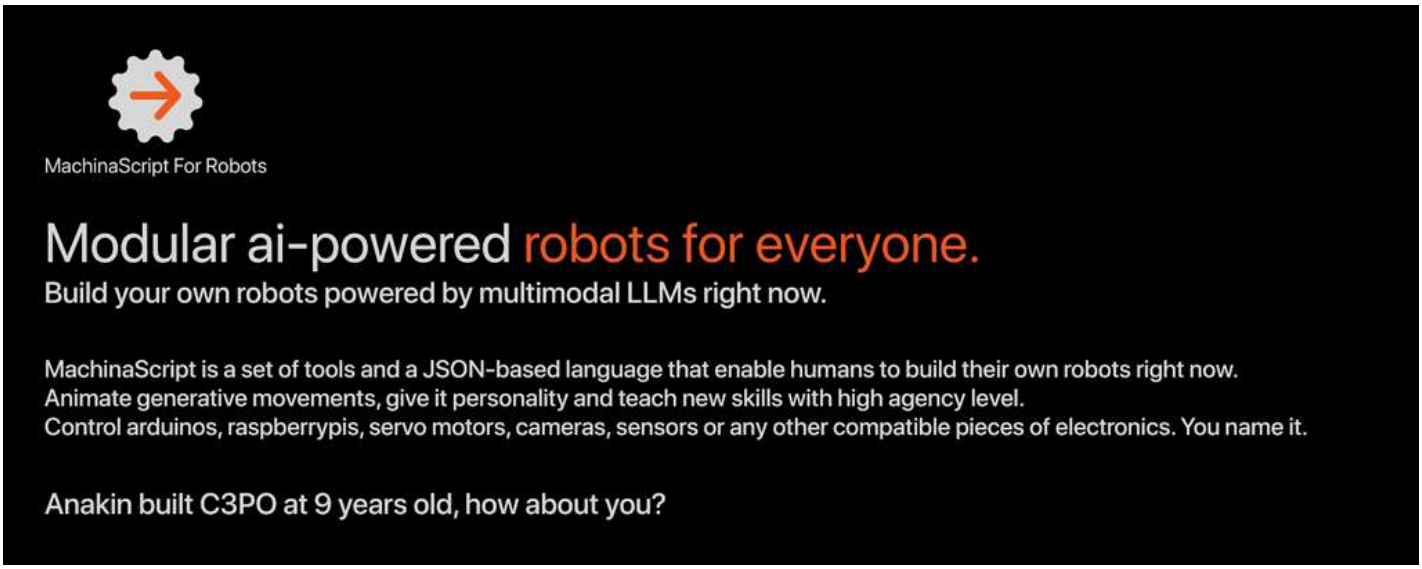
Step 10: Wrapping It Up



1. Starting with the arduino code, take a look at the template file included in this repo and modify it according to your project. Include any motors, ports and sensors as you like. In other words, start by making your robot move programatically before hooking it to an LLM to make sure the project works.
2. Proceed to editing the brain file and hooking with the arduino - map your project components and motors and pass them properly in the code. Then gently hook it with the serial arduino port. Try to make simple tests first, then you go complex. Explore new skills that only components could provide - for example radio frequency scan, RFID, infrared, web-related stuff... You name it.
3. Finally when you have the entire project set, teach the LLM how your robot works - pass all your specs in the MachinaScript_Project-Specs.txt and don't be afraid to explore new methods of doing it. In the file you will find a set of examples. We also recommend you having a quick read on the MachinaScript_Language.txt to understand better the synthax we initially came up with, however you may want to leave this intact for compatibility with the ready code parts in the body and brain.
4. If you are new to programming and have way too many questions, don't hesitate to paste the code on chatGPT-4 and ask about its structure as it may provide you some great insight for you to make your own modules. We really encourage you to get started debugging your code with the Ai pals.

You see, making Ai-powered robots is super easy now!

Step 11: Share Your Robots With the MachinaScript Community!



Join the discord community, share your designs, skills, 3D models, hacks and crafts with us! We really wanna see your robots :]

MachinaScript for Robots is and always will be free and open source for everyone.

Contribute to the github repo here. You can find all the code examples to start making your robots right now.

And suddenly the dream of building intelligent robots is not a dream anymore.

MachinaScript for Robots is in very early beta. Use at your own risk.