Getting started with Elmo

What is Elmo?

Elmo V2, with official github available here, is the second version of IDMind's tabletop robot, with upgrades from the first version. This robot is designed for a variety of applications, including educational entertainment (Edutainment), human-robot interaction (HRI) research, and commercial uses. The source code for Elmo V2 is developed almost exclusively using Python 3.9 and is designed to run on a Raspberry Pi 4 using Raspberry Pi OS Bullseye. **We recommend you first follow this guide to acquaint yourself with the robot.**

What comes with the robot?

If you choose to use Elmo, you will be given access to a box containing:

- 1 Elmo
- 2 Ethernet cables
- 1 Router (shared by all Elmos)

All equipment underwent thorough testing and verification prior to its availability to students. Kindly exercise caution and ensure the return of each item enclosed in the box in the same condition as it was borrowed.

What can Elmo do?

Hardware Components: Elmo V2 has the following hardware components:

- Power Button
- Ethernet Port
- Microphone
- 13x13 Led Matrix
- Capacitive touch sensors (1 in chest, 4 in head)
- 2 servo motors for head movement (pan and tilt)
- Touchscreen (face)
- RGB Camera
- Battery
- Speakers

Initial Setup: Upon receiving Elmo, users can activate it by holding the power button for three seconds. Once booted, the robot displays the IDMind logo on its LED matrix, and its head starts moving. **Users can test the capacitive touch sensors by interacting with the robot's head.** Try it!

How can I program Elmo?

To effectively program Elmo, you have several options. Primarily, Elmo is operated via REST API commands. This flexibility means you can use any programming language to send and receive these REST requests. In this guide, we will introduce a Python interface pre-equipped with these commands, enabling you to control Elmo immediately.

First, visit the official GitHub repository to explore the available requests. You'll find that Elmo can perform various actions like moving its head, displaying images and videos on its monitor, and recording sound. While these functions are quite useful, it's important to note that Elmo currently lacks built-in requests for using its camera or generating speech. However, you have the opportunity to expand its capabilities. We'll guide you through modifying Elmo's internal code to add a new request for speech generation.

Now that you're familiar with Elmo's existing features and potential enhancements, let’s move on to powering up the robot and establishing your initial connection.

Getting started: Install the Companion App

Strat by turning the robot on (by pressing the power button for 3 seconds, you also turn it off that way) and wait for it to finish loading. After a few seconds, the robot will display a set of eyes on its monitor and start looking around. You can even pet it on its head and the robot will react to the petting.

You can not control the robot just by turning it on and looking at its default idle behavior, you need to send REST requests. To send REST requests you need the robot’s IP address.

How to get the robot’s IP address:

1. Download anaconda and create a python 3-9 environment (we called ours elmo2-env) and activate it
2. Then, use git to clone the official elmov2 github
3. After the cloning is complete, go to the elmov2/app folder
4. Execute the dev.sh script (we recommend ubuntu 18.04), in ubuntu and debian you can run the command *bash dev.sh*

   a. **Note:** The github does not contain any requirements file, this means you need to install every dependency using pip yourself. Just keep running the command until no more errors related to libraries and dependencies show. **Don't forget to update pip.**
   b. **IMPORTANT NOTE:** The app uses PyQT5 for its graphical aspects. If you are running a linux distro that is more recent than ubuntu 18.05 you probably need to follow the instructions on this website to correctly install PyQT5. Try to install everything first using pip, and then if the app does not run try installing PyQT following the website's instructions.
5. After getting the app to run, connect Elmo to the router with an ethernet cable and connect your computer to the router using another ethernet cable
6. In the app click on "Scan network", if everything is working fine, a button with your robot and its IP address should appear, click it.
7. Now you are connected to elmo, annotating its IP so that you can use it to send REST requests or connect via shh to the robot.

Send a REST request to the robot

1. Download the python files from here.
2. Look at the ElmoV2API class, you can see we used the requests library and created functions that create a POST or GET request, depending on the command.
3. Now, look at the elmo_test.py file. This file is supposed to be runned in the terminal with the command python elmo_test.py ROBOTS_IP_ADRESS_HERE. Run the command and the pythons script will automatically send requests to the robot, making it display videos on its monitor. But you can alter the elmo_test.py file call any command from the ElmoV2API

Using SSH to connect to the robot

If you need to create new behaviors that the rest API does not support you will need to update the code running inside elmo. To connect to Elmo using ssh you run the following command: *ssh idming@ROBOT_IP_ADDRESS_HERE*, and the password is *asdf*.

Implementing speech:

To implement a new behavior we have to:

- Implement the new behavior and send it to the robot
- Implement a REST request that allows us to run the new behavior

Implementing new behavior for speech

Go to the elmo-v2/src folder. In there you will find (among other things) drivers to use the hardware or make elmo do something, a middleware.py script and a robot_api.py script. If you look closely, there is already a driver for speech, so we don't need to implement it. To know more about middleware.py, go to the official github. In sum, this file allows you to control the hardware of the robot (it's a middleman). Finally the robot_api.py implements the available REST requests and the functions each request should call when made. To create new behaviors we typically need to create a driver and change the robot_api.py file.
NOTE: you will all share the robots, make sure to reset the robot (i.e. make sure the robot is running the original code it came with - the one on github).

Speech Driver:

In our case, there is already a speech driver. Open the driver_speech.py file. If you inspect it you see that this driver makes the robot produce speech from text by sending a gtts-cli command to the terminal. This means we need to install this command (because the command does not come pre-installed on the Elmos). To install the gtts-cli command, shh to the robot and follow the installation instructions on the official gTTS github.

Modifying robot_api.py

At this point you've seen the speech driver's code and have already installed the gtts command. Now that the robot can actually execute the command we are ready to implement the rest of the speech behavior. To do this, go to the robot_api.py file. You will need to do three things:

- Update the __init__ and update functions so that the information about speech is saved and provided to you when you send a get_status command. This will allow you to see what portion of the text has been already said by the robot for example.
- Create a new command under the command() function
- Create the function that's going to be called when the rest command for speech you previously defined is received

Update __init__ and update functions:

The __init__ function is called when robot_api.py runs, in our case it is called when we turn on the robot. If you look at the function you will see that it creates attributes that store information about the robot, such as its current battery level, the speakers volume, the available video list, etc. This information is then updated in the update function right below. The update function is called every x timesteps by the robot OS so that it maintains the most update information stored.

Now let's look at line 206, this function, called status, is the one that will be called when we send a get_status rest command to the robot. You can confirm that by looking at the url ar line 205, it’s the same for the status command as explained in the official github. So it is important for any new behavior that you implement that you update these two functions accordingly so that you can then access useful information via a get_status command if you need it.

In our case, we will go to the middleware.py file and look for the Speech class. It is at line 511. You can see that the Speech class has four fields: ready, language, say and saying. Check ready to see if the speech driver is ready, set language to a language code to set the language, set say to a string to say something and check saying to see what is being said. These are the fields that you will use to update the __init__ and update functions. As such, first define an attribute mw_speech and initialize an object of the Speech class (defined in the middleware.py file). Then, create attributes corresponding to each field in the update and __init__ functions in the robot_api.py.

Create a new command:

Still on robot_api.py, go to the command function (line 214). This is the function where the rest command is going to be implemented. Create a new elif op == “speak”, then inside this elif retrieve the necessary arguments. In our case we will define that the command speak receives text and language as arguments, so you need to retrieve those from the req variable. See the original code for guidance. Then, after having the text and the language we need to call a function that will execute the behavior (in our case have Elmo execute the gtts command). To do this we first add the line success, message = robot.speak() and then we will define this function.

Create the speak function:

Now you’ve implemented the command but the function the command calls does not exist. We need to define it. Inside the robot class on robot_api.py create a new function called speak (see the other functions for guidance, for example set_screen). The speak function needs to receive two arguments, text and language as per we defined earlier, and it also needs to return a bool and a message. Besides that, the function needs to set the mw_speech.language and mw_speech.text attributes.

Send the updated code to the robot:

Now you need to update the code running inside Elmo. You can send the files using the scp command. The files need to go inside the elmo-v2/src folder that exists in Elmo.

Now you just need to update the python api we provided and then call the command on elmo_test.py.

Having trouble? The files resultant from this exercise can be accessed here.

Further documentation and guides:

- Official elmov2 github
- Previous elmo version github
- A wiki we made when working with the previous elmo version, some tips might be useful. However the wiki was created for the previous version.

FAQs: