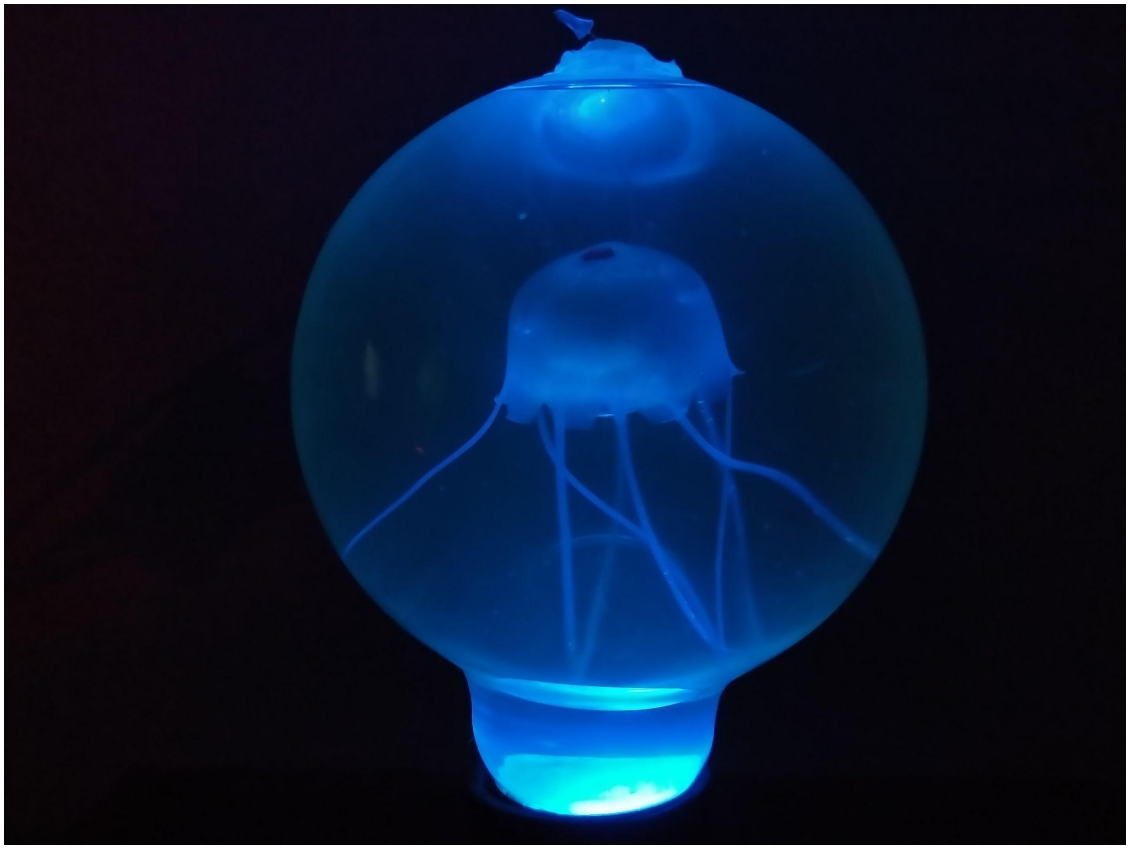
Electro-Jelly Box



Application Description

I initially started this project with the goal of making a jellyfish lava lamp, which is as crazy of an object as it sounds; a glass jar filled with little plastic jellyfish that swim around in patterns. As it turns out, a quick Google search yielded hundreds of images of such devices, and it was hardly unique.

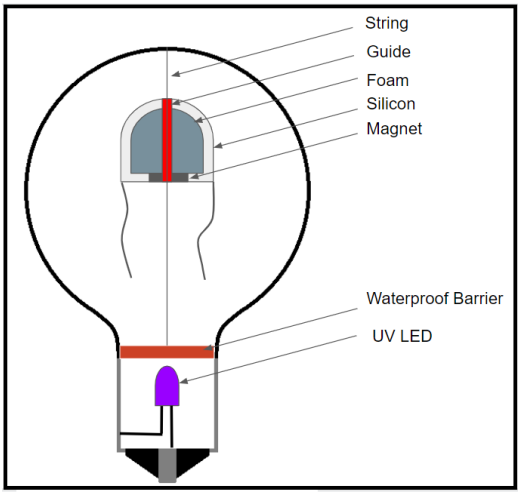
My problem with the vast majority of examples I found online was that they simply blasted the poor creatures from underneath with a jet of water, causing them to shoot violently to the roof before drifting haphazardly to the bottom of the jar. They were hardly realistic.

I set out to create a similar idea which would incorporate much more lifelike swimming patterns.

Hardware Description

The hardware, both mechanical and electrical, ate up the majority of my time on this project.

Mechanical

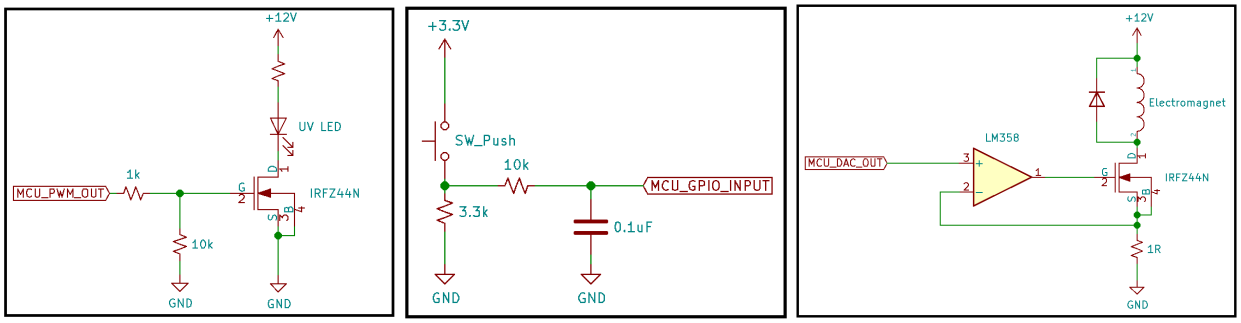


The heart of the project is the enclosure which holds the jellyfish. It consists of a large, globe light bulb that I carefully drilled into, removing both the bottom metal connector and top glass ceiling.

The jellyfish is a silicone mold (meant for fish tanks) which I bought on Amazon and reworked to look more realistic. I inserted a foam core to help with buoyancy, and glued a magnet on the bottom to provide the upward force to replicate swim patterns.

The entire bulb is filled wit ultraviolet blue dye and water, and a UV LED hidden in the base helps illuminate it in dark settings.

Electrical

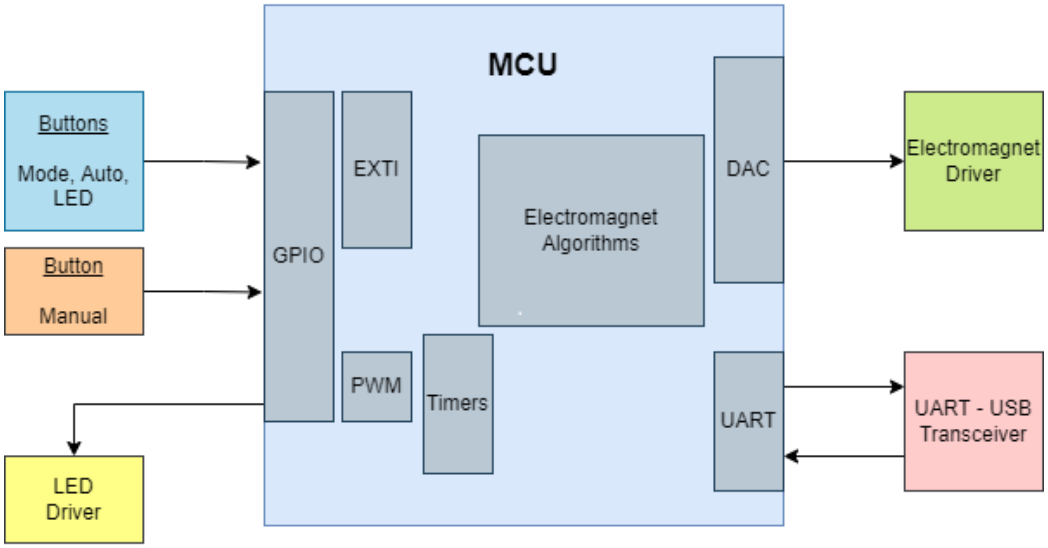


There are a few small circuits that I put together on a breadboard to help to improve the functionality of the system. The main electromagnet is driven by a basic voltage-controlled current source (right), which gets a bit too hot for my liking.

The LEDs are driven by an NMOS acting as a low-side switch (left) to handle the sizable current. Also, each button in the system contains a pull-down resistor and simple RC low-pass filter (center) for debouncing.

Software Description

At the highest level of the software is a simple state machine that keeps track of the different control modes for the jellyfish: idle, auto, and manual. Under the hood, the software consists, in large part, of peripheral drivers.



The following files/modules are the ones I most significantly modified for this project:

States.h/c
The heart of the state machine consists of two case statements. One transitions the state using the current state and event, while the other calls the corresponding function.

There's a quasi substate in the "auto_pulse" state, where the user can select different swim patterns for the MCU to generate.

Buttons.h/c
This module contains the code to initialize and read the system's external pushbuttons. There is no debouncing here, since that is done in hardware. Three of the four buttons utilize edge-triggered, external interrupts, which makes it easy to account for single user presses. The fourth button is meant to be held down, and therefore is continuously polled in one level of the main state machine.

Electromagnet.h/c
This code relies entirely on the DAC to feed a voltage-controlled current source for the electromagnet, and abstracts the details away from the main state machine.

led.h/c
The LEDs in the system are driven by a ~100Hz PWM which varies its duty cycle to dim their brightness.

Reuse
Apart from ST/Arm's native startup file and register definitions and Elecia's console module, a large portion of the codebase was reused from a project I created last year. This helped me avoid reinventing the wheel and focus on the important modules.

Build instructions

I left the vendor's files out of this repo and only included the ones written or edited by myself (apart from the console module). To build the software for this project, take the following steps:

1. Clone the project GitHub repo located at:
    - https://github.com/aaronv55/Making-Embedded-Systems-Class
2. Download STM32CubeIDE if you have not done so already.
    - https://www.st.com/en/development-tools/stm32cubeide.html
3. Create a new project and specify the target as STM32F410R8T6
    - This should generate the necessary vendor files, including *stm32f4xx_it.h, syscalls.c, system_stm32f4xx.c, stm32f4xx_hal_msp.*
4. Import the files from the project repo in step 1 by navigating the following menus:
    - Right click the directory "Core - Src" and select Import → General → File System. Include all ".c" files from the repo.
    - Right click the directory "Core - Inc" and select Import → General → File System. Include all ".h" files from the repo.
5. The project should now be ready to build and flash.

Hardware Required
The most critical part of the hardware system is the power supply. At its peak, the electromagnet will draw upwards of 2A at ~12v. For this I used a Power Force brand 13.8v, 20A supply which was capable of handling these large loads.

Apart from that, required hardware components can be found in the circuit diagrams above under Hardware Description - Electrical.

Future

- Consolidate Hardware - This project is currently living across two separate breadboards and a million jumper wires on my desk. In the future, I'd like to solder everything to a perf board and put it in the wooden enclosure. I'm also hoping to replace the power supply with one from a high-power laptop which is much nicer on the eyes.
- Add fancy algorithms - The current swimming algorithms are a bit basic. Moving forward, I'd like to tune them slightly and see how close I can get to making this look like a real jellyfish.
- Better Jellies - Over the course of this project, I did a lot of Googling. One thing that kept coming up which caught my eye was the many online examples of makers using magnetic filaments in their 3D printers. Creating the jellyfish in this manner would both improve longevity and decrease the need for a large, power-hungry electromagnet.

Grading

Project meets minimum project goals	**2**	The requirements of the project are met, but the state machine is fairly basic. The command line could use a few more complex commands.
Completeness of deliverables	**2**	The report is complete, however there could still be areas where greater depth may be needed, particularly system architecture.
Clear intentions and working code	**2**	The system seems to be described somewhat adequately in the report.
Reusing code	**2**	I did reuse code. I was slightly cheating by reusing some of my own code from an old project, though.
Originality and scope of goals	**3**	I wouldn't say that I went far beyond the requirements, but I think the idea is fairly unique.