

NON-ROV-DEVICE (FLOAT) DESIGN DOCUMENT

RoseVG Robotics
For Rose-Hulman Institute of Technology
Terre Haute, IN, USA

Float Design

RoseVG Robotics' vertical profiling float was designed to have a small footprint, high performance, and easy serviceability (RoseVG Design Principle DP5). This section explains the design for our float's mechanical, electrical, and software components.

Float Mechanical

The float is housed within a 2" PVC pipe. The buoyancy engine, which is composed of a brushed dc motor and a lead screw, fits inside and controls the position of a 60 mL syringe to expel or draw in water, changing the density of the float. The output shaft of the motor fits snugly into a hole drilled into the 6.35 mm lead screw and is held in place with a set screw. The buoyancy engine parts are mounted and housed in a 3D-printed frame that slides into the PVC casing. A round, 3D-printed part with an embedded nut attaches to the lead screw, moving up and down when the screw turns.

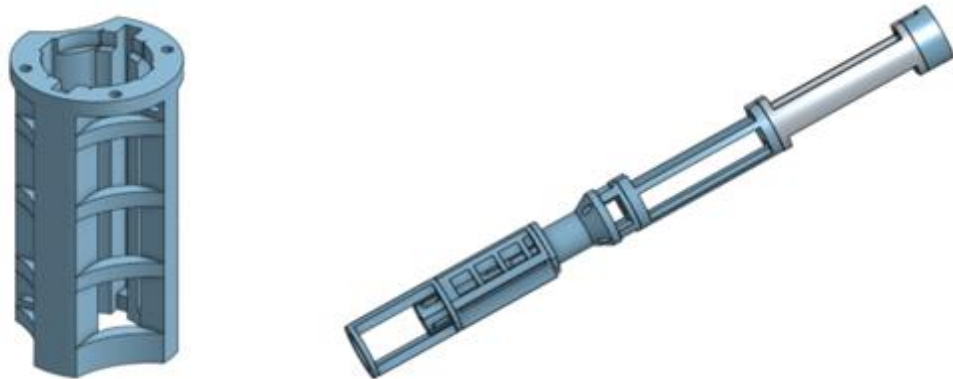


FIGURE 1: FLOAT MECHANICAL SYSTEM CAD MODELS

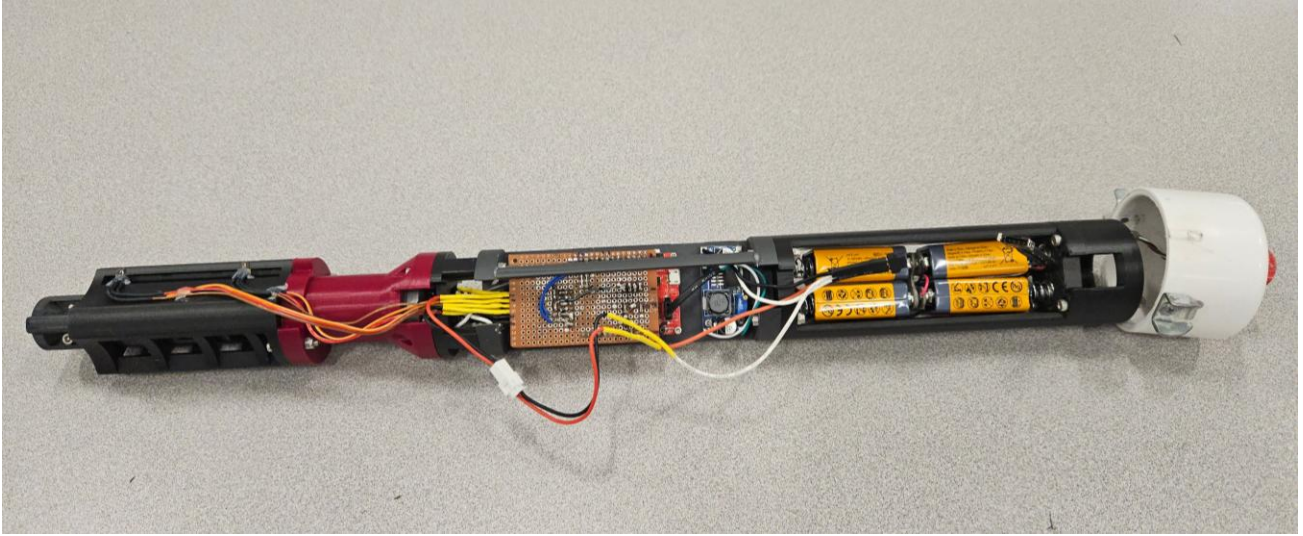


FIGURE 2: FLOAT SUBFRAME WITH ALL COMPONENTS MOUNTED.

Float Electrical

The fuse has not arrived yet; it would be spliced between the battery pack connector and the batteries.

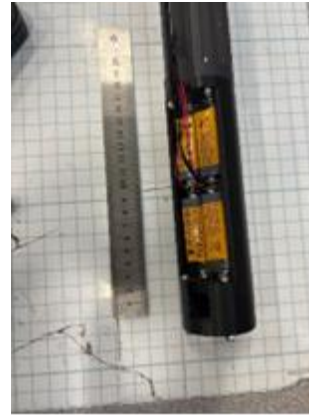
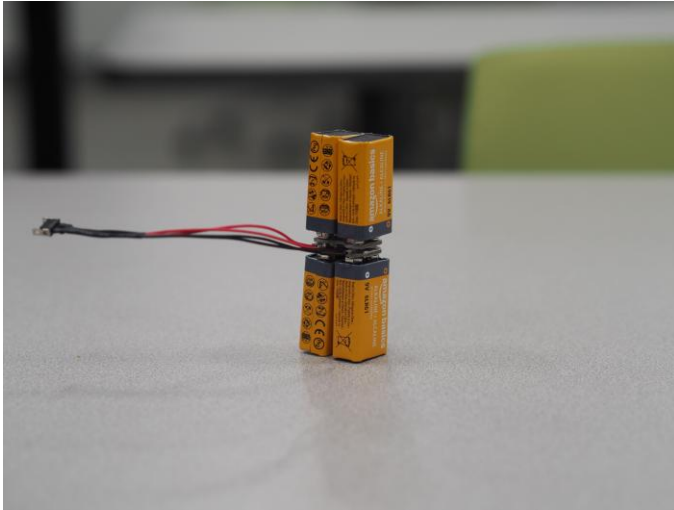


FIGURE 3: EXISTING FLOAT BATTERIES. ALKALINE BATTERIES ARE NOT FINAL; 9V NiMH BATTERIES WILL BE USED ON THE FINAL FLOAT.

The battery pack is made up of 4 9V batteries, so we could drive our motor at 9V with as little current as possible. We use 4 9V batteries with an above-average capacity for NiMH, so we can safely deliver 1A. And since our FLA is only 140mA, we have 860mA of headroom for motor in-rush current or any other possible power draws. Alkaline batteries are not final; 9V NiMH batteries will be used on the final float.

The Full Load Amperage of all float components were measured and are presented in the table below:

Component	FLA Measurement (mA)
Motor	100
Microcontroller	30
Sensors	10
Total	140

Float Software

Broad structure

Float control software can be broken into three components. The float controller, the relay, and the user interface. The float controller and relay use LoRa antennas to communicate with each other, and the user interface communicates with the relay via a USB serial port. The float controller and relay are both Arduinos with built in LoRa chips. The user interface is a Java program that runs on a laptop.

Relay

The relay facilitates communication between the user interface and the float controller. The relay does not automatically do anything.

User interface

The user interface is a CLI that executes commands, receives data from the float, and shows the float's status.

Float controller

The float controller receives commands, transmits data, and does the profiles. It communicates with the pressure sensor via I2C and controls the motor with a direction pin and enable pin.

Relay

The relay does not do anything special. It just relays commands and data between the float controller and user interface. The only notable thing it does is ask for dropped data packets to be retransmitted. See the communications section of the float controller for details on the communications protocol.

User interface

The user interface can be broken into a few parts:

- The CLI listens for user input and executes the command.
- The background executor executes any commands that should be run automatically (like setting the current time on startup) and retries eligible commands (like setting the current time).
- The relay listener listens for status updates from the relay.

Received profile data is written to a csv file.

Float controller

The float controller can be broken into several parts:

- The buoyancy control module is responsible for driving to buoyancy engine to reach the target depth.
- The sensor data module is responsible for reading the current depth and storing profile data.
- The communications module is responsible for receiving commands and transmitting data and status packets.
- The profile control module is responsible for determining when to record a data point and determining the target depth.

Buoyancy control

The buoyancy control module monitors the hall effect sensors in the buoyancy engine to ensure that the motor does not actuate the engine too far. The buoyancy engine is actuated to increase or decrease speed towards a target maximum speed, this maximum speed is calculated as follows:

```
If the float is close to the target depth, then the maximum speed is maximum_close_speed.
Otherwise, the maximum speed is:
max(maximum_close_speed, abs(target_depth-depth)/minimum_approach_time)
```

`maximum_close_speed` is a hardcoded maximum speed when the float is near enough to the target depth that the calculated approach speed would either overshoot or be too slow.

`minimum_approach_time` is a hardcoded value that specifies that if the float were to continue at its current speed until it reached the target depth, it must take at least that long.

When the Arduino is executing a command that may take a long time to complete (such as the transmit data command) the buoyancy engine motor is paused to ensure that it does not actuate too far.

Sensor data

The sensor data module just reads the pressure sensor and stores data points when directed.

Profile control

The profile control module is a state machine that does profiles until two were completed successfully. If a profile is unsuccessful, the float will automatically abort it and begin again. When all profiles are complete, the float will hold a position between the top and bottom interval depths to indicate that it is done.

Communications

Communications are done via LoRa radio and are hamming encoded. Commands are sent to the float in the following format:

```
COMPANY ID (8 bytes ascii)
COMMAND ID (8 bytes ascii)
OPTIONAL DATA (up to 20 bytes)
```

The float will only respond to commands that have the correct company id attached. The commands that the float can execute are as follows:

- Do profiles (`dprofile`) - causes the float to begin the profiles
- Transmit data (`transmit`) - causes the float to begin transmitting data packets (see below for details)
- Received packet status (`txrxstat`) - causes the float to retransmit any dropped data packets (see below for details)
- Set current time (`setctime` + 8 bytes) - causes the float to set the current time to the value in the 8 data bytes
- Query status (`qurystat`) - causes the float to transmit a status packet

Data packet transmission

When the float receives the transmit data command, it will transmit all data packets in order. Each packet contains the following fields:

```
company (8 bytes ascii) - the company id
profile (1 byte) - the profile that this packet belongs to
packet_number (1 byte) - which packet this is
total_packets (1 byte) - the total number of packets that will be transmitted
time (8 bytes) - the time that this reading was taken
pressure_1 (4 bytes) - the pressure at the time of this reading
pressure_2 (4 bytes) - unused
depth_1 (4 bytes) - the depth of the top of the float at the time of this reading
depth_2 (4 bytes) - the depth of the bottom of the float at the time of this reading
```

When the relay receives a data packet, it will note which packet it received. When the relay stops receiving packets, it will check to see if the number of packets received matches `total_packets`; if it does not, it will send the received packet status command with the optional data containing which packets were received. The float controller will then retransmit only the packets that were lost. This sequence repeats until all packets are received or the relay gives up.

Status packet transmission

Status packets are transmitted at a hardcoded interval or whenever the query status command is received.