Nathaniel Paulus

Dale DeRoche

Electrical Engineering

3 May 2019

Design Considerations for an Automatic Fish Feeder

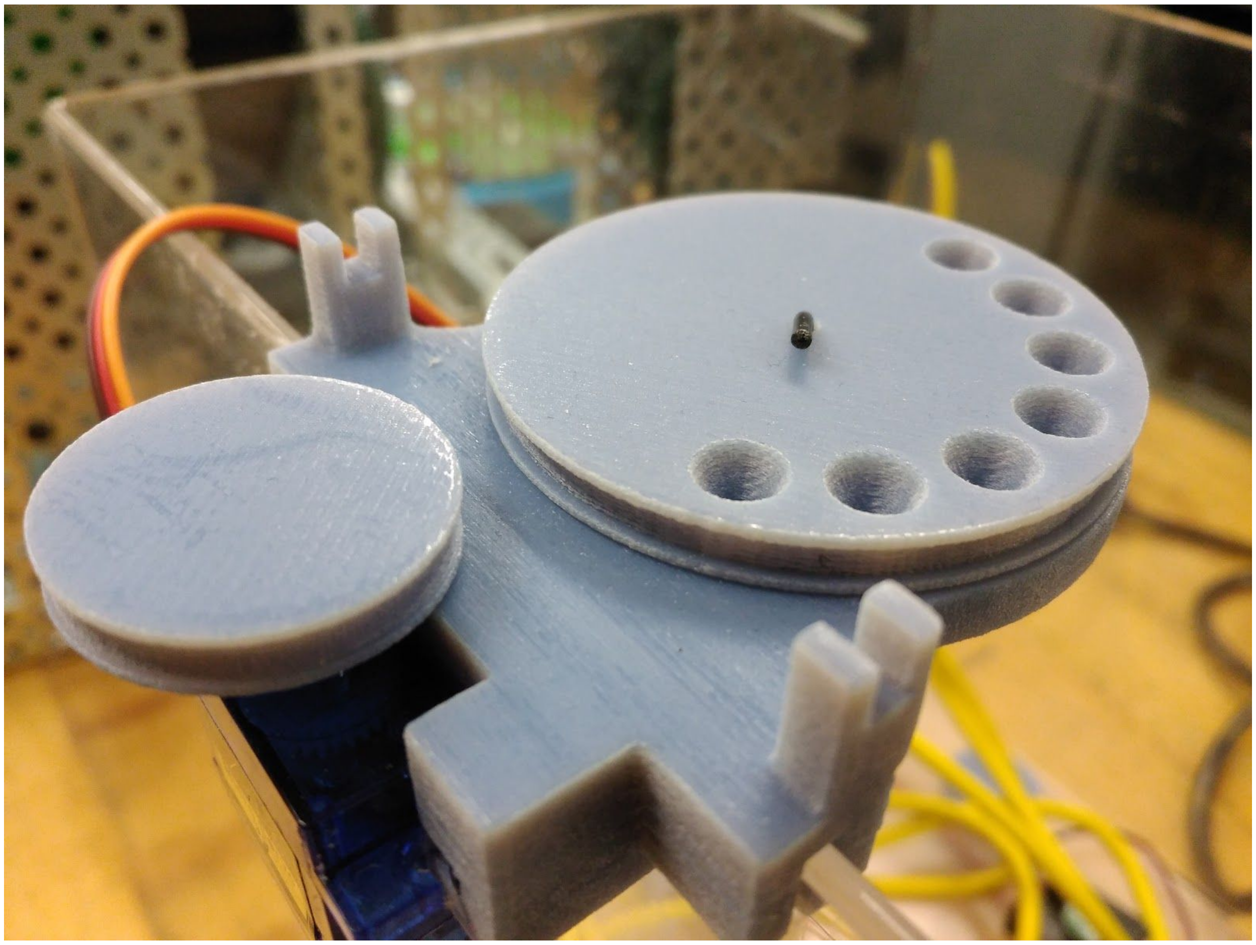
INTRODUCTION

During the fall 2018 semester at Ivy Tech, one of the projects I worked on in EECT 222 (Intro to Microcontrollers) was an automatic fish feeder. The design went through a couple of iterations during the semester and ultimately was not finished due to unresolved issues. This document details some of the challenges encountered, how some of them were resolved, and potential solutions to as-yet unresolved problems. It also considers issues that have not yet come up and had hitherto not been considered.

DESIGN OVERVIEW

The fish feeder is made to attach to the edge of a fish tank, with part of it overhanging the water. Supports are used to raise the lid of the fish tank around half an inch to make room for the feeder. The platform overhanging the water contains a hole for allowing fish feed to fall through. A horizontal wheel directly on the platform contains seven holes for holding fish feed, one hole for each day. As the wheel is rotated the holes will align with the hole in the platform, allowing the fish feed to drop into the water at controlled intervals. The wheel is driven by a servo motor via a rubber band belt. A bent piece of insulated solid-core electrical wire served as a surprisingly good shaft, running through the wheel and the platform it rests on.

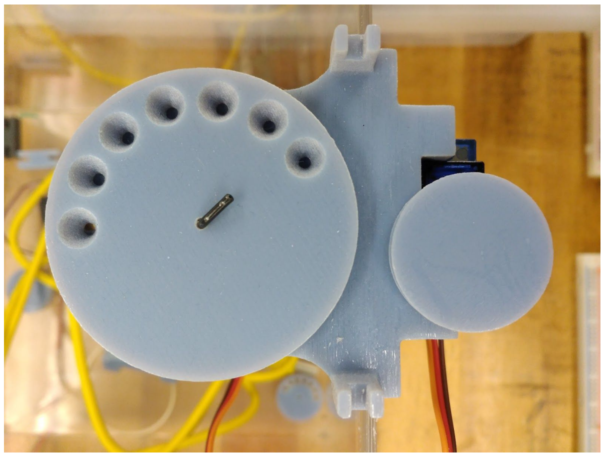
The entire structure was 3D-printed. The size of the holes proved to be too small in the original version, and the quality rather low, so it was reprinted with larger holes and a much higher quality print. The most recent prototype is shown in the image below.

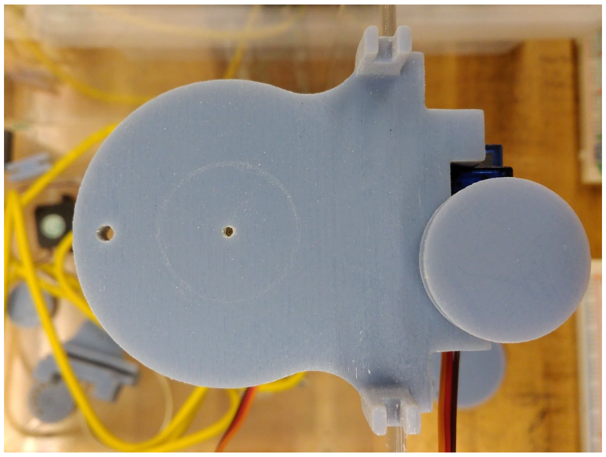


ISSUES WITH CURRENT DESIGN

The increase in hole size and print quality were dramatic improvements, but the larger holes necessitated a larger wheel. This was problematic because the wheel being driven was now significantly larger than the drive wheel mounted on the motor. The servo motor can only rotate slightly less than 180°. With approximately a 2-to-1 ratio in size between the two wheels, the servo motor could only rotate the wheel around 90°, about half the necessary range of motion.

Increasing the size of the driving wheel to match the size of the feeding wheel was impractical due to size constraints.

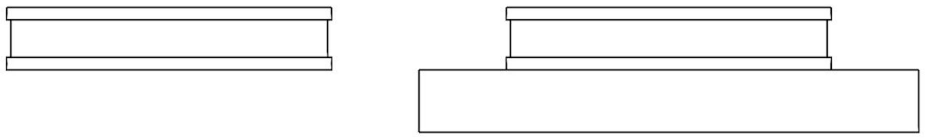




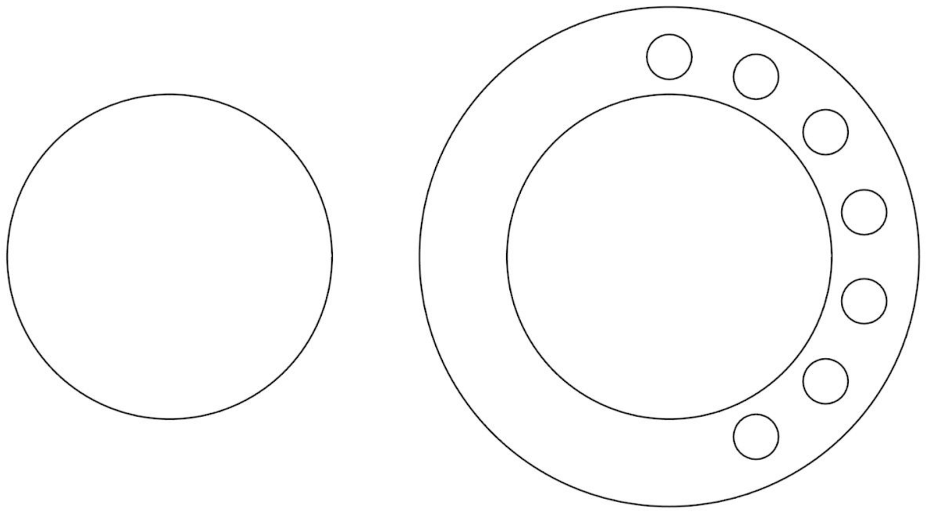
There are several possible solutions to this issue, listed below:

1) Replacing the servo motor with a motor that has about 360° of motion would be sufficient. Such motors do exist, but are more difficult to obtain and likely would cost significantly more than motors with the more conventional 180° of motion. Additionally, the size, power requirements, and ease of finding replacement parts make the current motor ideal for the job.
2) A change in design could allow the motor to drive the feeding wheel directly. This would most likely be done by mounting the motor upside-down directly above the feeding wheel. While not impossible, this would make the design more complex and require the motor's mount to overhang the feeding wheel. The overhang would likely create a mechanical stress point, increasing the chance of it breaking.

3) Mounting a smaller pulley on top of the feeding wheel would require minimal changes to the current design while allowing the motor to drive the wheel through about 180° of motion. The servo motor would need to be mounted a little higher, at a height equal to the current thickness of the feeding wheel. Among each of the possible solutions, this seems to me to be the most practical to implement. Shown below is a drawing of a side view of the proposed configuration.



The pulley on the left would mount directly on the servo motor and drive the pulley mounted on the feeding wheel. Because the wheel is no longer being driven by a rubber band, so it is shown without the groove has had in previous designs. The same drawing, rendered from above, is shown again below.



SUPPLYING POWER

The servo motor is controlled by an Arduino Uno. During testing, power was supplied to the board from a computer via a USB cable. Power for the servo motor was supplied directly from the board's 5V pin, which can provide sufficient current for a small motor. During actual use, powering the board directly from a computer is likely to be impractical. Depending on the setup it could even create a direct line for water to flow along the cable from the fish tank to the computer. Such a possibility should also be taken into consideration for any means of power that leads back to a power outlet. Caution should be taken to ensure the lowest point along the cord is not at the power outlet, so that the water would drip before reaching it.

What follows are several reasonable ways of powering the board.

1) Supplying power directly to the Arduino's 5V pin provides the greatest amount of flexibility, but also is more complicated. Supplying power to the 5V pin bypasses the Arduino's onboard regulator, so an external regulator becomes necessary. While this can be a disadvantage, it may be the best solution when power is supplied from a source that provides less than 7V, the minimum recommended voltage to be supplied to the Arduino's regulator. Additionally, applications that require that power consumption be very low may benefit by being able to use an external regulator that is more efficient than the onboard linear regulator. If power consumption is a major factor, however, it would likely be wise to do away with the board and use the Arduino Uno's ATmega328P chip directly, or even the 8-pin ATiny85. Extremely low power consumption does not appear to be of importance for this application, so the addition of an external regulator makes this solution unideal.

2) If powering the board from a battery is desired, a 9V battery makes an ideal choice. The recommended input voltage for the Arduino's regulator is 7-12V, making 9V fall squarely within those bounds. Additionally, 9V battery connectors that plug into the Arduino's 2.1mm power socket are readily available, providing an easy way to connect the battery. The biggest downside to using a 9V battery is that they tend to have relatively low mAh ratings. It's unlikely a 9V battery would power an Arduino for very long, given the amount of overhead power consumption on the board.
3) By far the simplest option is to power the board from a power adapter plugged into an outlet. A 9V power supply could connect to the 2.1mm power socket, or a 5V USB charger could plug into the board's USB port. Both are extremely easy to implement and should last indefinitely. One caveat is that the board will reset whenever the power goes off or the system is unplugged. This could be partially mitigated by periodically writing the program's state to the EEPROM. The progress of the program would then be delayed by the amount of time between the last writing of state, and the time the board receives power again. Since the exact time the fish are fed is likely of little importance, this seems to be an acceptable solution. Such logic has not been implemented, however, and it seems unlikely that it will be added in the future.

POTENTIAL BELT ALIGNMENT ISSUE

While driving the feeding wheel with a rubber band appears to work quite well, there is the potential that over time the two pulleys could become misaligned. This is especially likely to occur if the rubber band is not stretched evenly. While it’s not possible at this time to determine whether this will be an actual issue, it’s worth considering possible solutions if the problem does materialize, and it should be tested before considering the project complete.

The direct drive method listed as a potential solution to the wheel-size issue above would not have this problem, since it eliminates the rubber band entirely. However, despite potentially solving two issues at once, it remains a poor solution for the reasons previously mentioned.

Another potential solution is to connect a potentiometer to the wheel being driven, making it a closed-loop system. This would allow the software to automatically correct for any misalignment detected. This, however, raises its own challenges. If the two wheels become misaligned by more than a few degrees, the servo motor will not be able to correct for it, because it will no longer be within its range of motion. If the drive wheel is made larger than the pulley it connects to, then the range of motion will be larger, and it can correct for misalignments of more than a few degrees.

Alternatively, the servo motor could be replaced with a continuous-rotation servo motor. This eliminates the servo motor’s closed-loop control, but such control is no longer needed due to the feedback mechanism on the wheel that is being driven. This type of servo motor can be purchased, or it can be created by modifying an existing motor.

Both of these solutions require adding a potentiometer to the wheel being driven. If placed above the wheel, it would require an overhang, which would both create a potential source of weakness and interfere with the user’s ability to fill the fish feeder. Alternatively it could be placed below the wheel, embedded inside the platform, with the shaft of the potentiometer protruding upwards into the center of the wheel, replacing the current shaft. Between the two, this is the much better option.

Currently the depth of the groove in the pulleys is quite small, potentially allowing the rubber band to fall off with repeated use. Further iterations of the prototype could be improved by increasing the groove depth significantly. Testing whether this is an actual issue could be done by setting the system up to run repeatedly for days on end and observing whether the rubber band has a tendency to fall off. This would also be a good way to test whether misalignment issues are a legitimate problem, and what effect the tightness of the rubber band has on the tendency for alignment issues to develop.

MILLISECOND OVERFLOW

The logic used by the program running on the Arduino uses the `millis()` function to determine how much time has elapsed in order to determine when to dispense the next day's food. The data type of the return value is unsigned long, which on the ATmega328P is 32 bits. After 2^{32} milliseconds (a little less than 50 days) the value will roll over to zero. Since the board is very likely to run for more than 50 days at a time, this is an issue that needs to be handled properly. Two variables are used in the decision whether it is time to dispense food or not: `previousMovementTime`, and `msPerDay`. The first stores the time in milliseconds, that food was last dispensed. This is relative to either the time the board started, or the last time the milliseconds rolled over. The latter stores the number of milliseconds between each feeding, which for testing purposes is set to 2000, causing it to move forward every two seconds. The most obvious way to check whether the amount of time has elapsed is to check whether `millis()=> previousMovementTime + msPerDay`. After the last feeding before the rollover takes place, `previousMovementTime + msPerDay` will roll over and become quite small. At that point the comparison will always find that `millis()` is larger and start attempting to dispense food every millisecond. In the worse case this could kill the fish by overfeeding.

This rollover can easily be accounted for by algebraically subtracting `previousMovementTime` from each side of the comparison, changing it to `millis()- previousMovementTime => msPerDay`. This comparison is mathematically equivalent, but the comparison is handled very differently. It no longer has the potential to make `previousMovementTime + msPerDay` overflow, since the two numbers are never added. When `millis()` overflows, `millis()- previousMovementTime` will underflow, giving the actual time difference between the current time and the last time the motor moved. This can then be compared with `msPerDay` without any issues.

While it's possible to reason through what will occur when an overflow happens, it's much safer to test it directly. It's not very practical to wait 50 days to observe what happens, but it's quite simple to mock what would occur. For testing purposes, a function can be defined that returns `millis()` plus a number that is slightly less than the maximum unsigned long. This function should then be used everywhere in the program in place of `millis()`. This simulates the overflow that would occur after about 50 days, causing it to occur almost immediately. When testing is complete the function can be modified to just return `millis()`, effectively restoring the default behavior.

It's worth noting that each time `previousMovementTime` is updated, it is updated by adding `msPerDay`, not by setting it to `millis()`. This ensures that the feeding time does not gradually drift over time. While the magnitude of such a drift would be unnoticeable in this application, it’s best practice to write code that is correct and future-proof. If in the future it was decided that the microcontroller board needed to frequently sleep in order to save power, then the drift could become quite noticeable.

While this discussion of handling timing is quite lengthy, the actual code in question is quite short and simple. However, if the situation were any more complex than it is, it would be wise to consider using a small, well-tested library designed for accurately handling timing. Not doing so could unnecessarily introduce unexpected timing bugs that are extremely difficult to diagnose and resolve.

CONCLUSION

The fish feeder project has already been a great learning experience. It remains to be seen whether it will materialize as a genuinely useful and practical tool. Based on my analysis, I believe that if it is designed correctly, it can be everything it is intended to be.