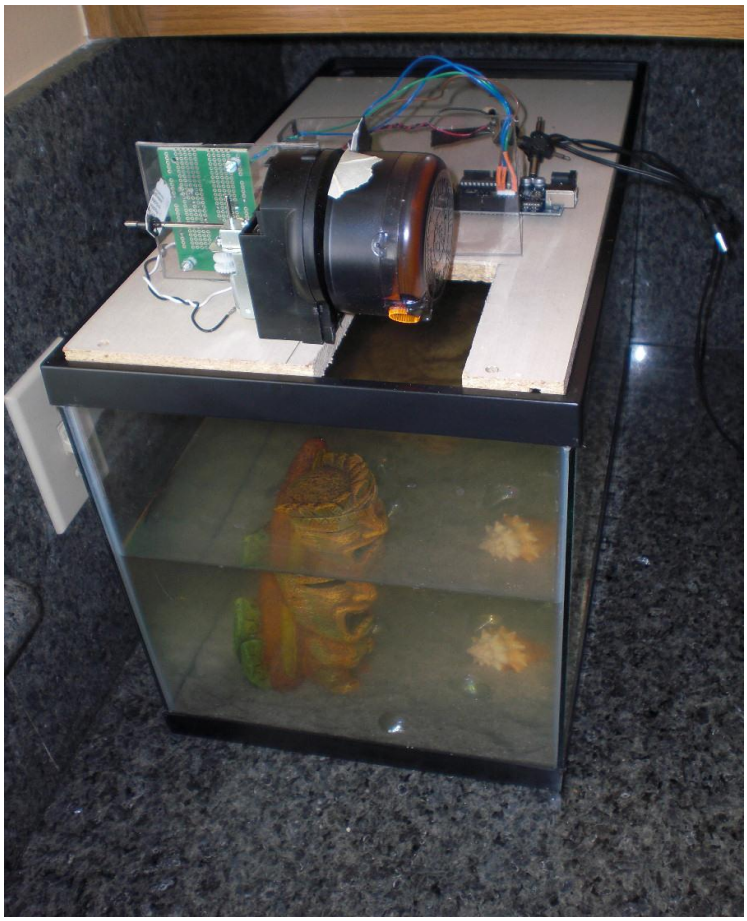
Frog Feeder

Overview



This project was designed to automatically feed my son's pet frogs while we was out of town for two weeks.

The frogs need to be fed once every two days and no commercial frog feeder was available. A fish feeder, purchased from *PetCo*, was considered as an "off the shelf solution", but it didn't work. The fish feeder was however cannibalised for parts for this project.

The project required software on an Arduino and some simple circuitry, soldered onto a protoboard. Some mechanical construction was required, to mount the motor, sensors and electronics, but the whole project can be done with basic mechanical skills.

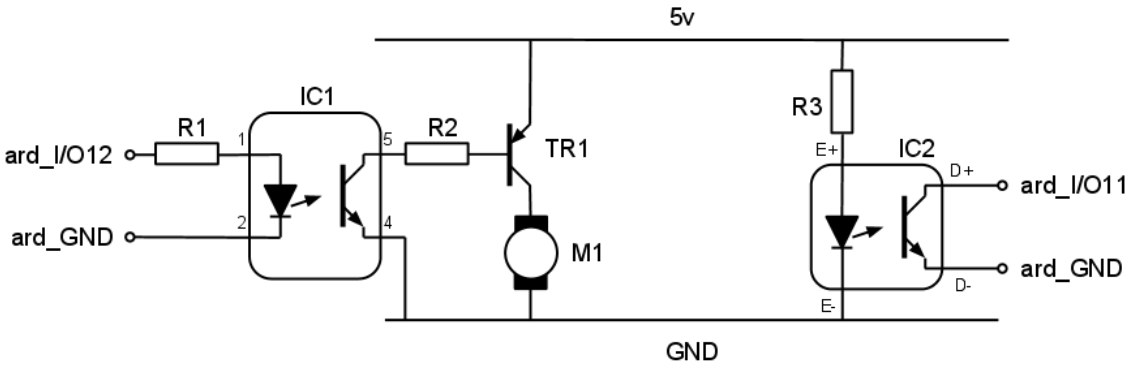
How it Works

An Arduino is used to feed the frogs once every two days. The Arduino counts down the necessary number of seconds for two days (configurable in the software). Once this time has passed an output pin on the Arduino drives a motor, which rotates a hopper to dump food into the frog tank.

A sensor, monitoring the food hopper, detects when the hopper has dumped the food and the Arduino turns off the motor. The cycle repeats indefinitely.

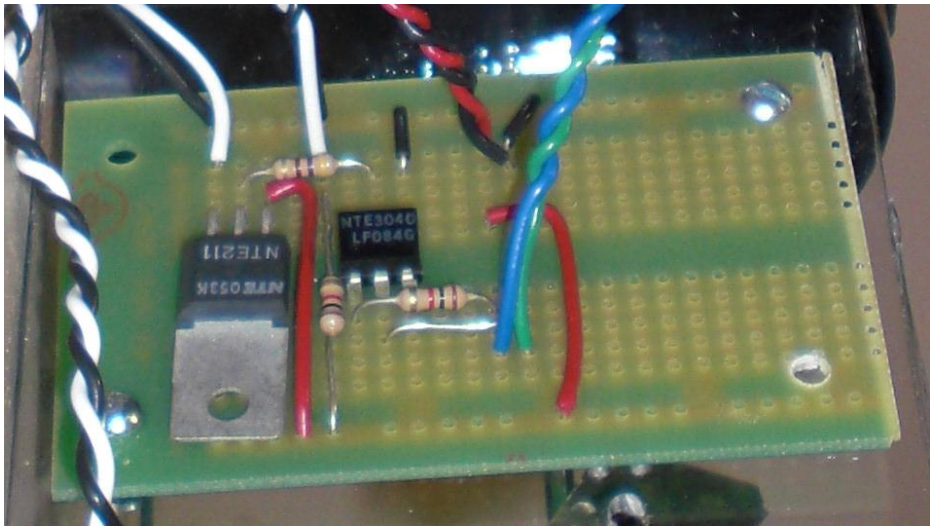
Circuit Design

Schematic Diagram



Motor Drive Circuit

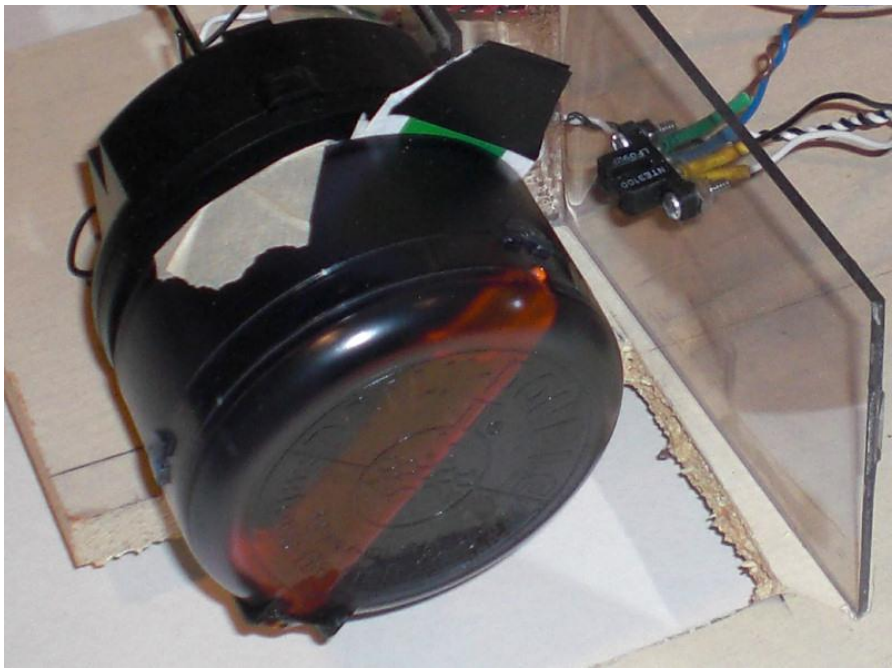
The motor and gearbox are an off the shelf, low voltage kit. The motor requires a voltage around 3-5volts and has low current requirement. The motor is driven through a medium power transistor (TR1). An opto-isolator (IC1) is used to interface between the Arduino and TR1. The opto-isolator provides noise isolation and reduces the risks of causing damage to the Arduino, due to short-circuits. I/O line 12 on the Arduino is used to drive the motor control circuit.



Position Sensor

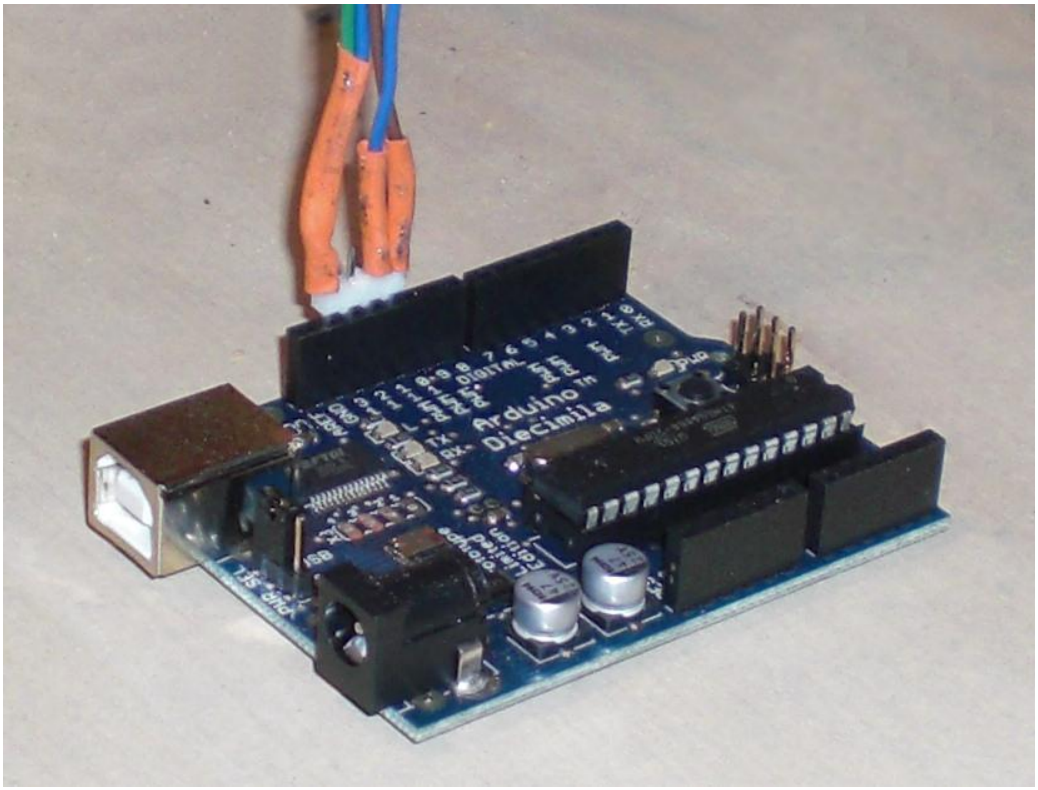
The small DC motor driving the hopper tends to start slowly when power is applied, it takes some time to get to speed, and also over-shoots when power is removed. So the time for a complete revolution is not constant. Additionally the food hopper tends to “slip” under load (a deliberate part of its design?) adding to the timing variability. For these reasons the only way to know when food has been delivered is to monitor the food hopper and wait until it completes a full revolution.

To monitor the food hopper a photo interrupter (IC2) detects a *tab* attached to the food hopper. The sensor is simply an IR emitter/receiver pair with an air gap between them. Anything passing through the air gap causes the sensor output to transition high-to-low. The output of IC2 connects between I/O pin 11 on the Arduino and the Arduino's ground. As with the motor drive circuit, this sensor is electrically isolated from the Arduino power supplies.



Assembly

The circuit was soldered onto the “Phenolic” prototyping board. Jumper leads from the circuit connect to the Arduino board, via a 4 pin header providing connections to I/O pins 11, 12 and GND.



Software Design

A short program on the Arduino implements a state machine. The state machine waits for 2 days, starts the motor, monitors the sensor, stops the motor, then resets the timer to repeat the process 2 days later. The program will loop forever.

As with all Arduino programs there is a **set-up** function (`setup()`) and a function for the **main-loop** (`loop()`). The set-up function executes once, when the software first starts, the mainloop is called repeatedly while the Arduino is in the normal running state. In addition to those two functions, there are **declarations** of a number of global variables, macros and some enumerations (custom data types).

Declarations

The first block of code is the declaration of variables and values needed for the application. This is done at the top of the file so the values can be used thoughout the application. This avoids using hardcoded values in the code, for example for things like seconds in a day or the pin numbers of the outputs.

- **Lines 10-12:** Variables identifying the pins used for inputs and outputs

- **Lines 17-30:** Macros and defines used to control timing.

  `FEED_INTERVAL` - defines how long between feedings

  `MAX_DELIVERY_DURATION` – limit the maximum time the motor will ever remain on

- **Lines 35-50:** Application variables and statemachine definition.

  `next_feeding/feeding_timer` – Control when the next feeding occurs and when it stops.

  `FeedState/feed_state` – enumerated type declaring the states of a statemachine used in the main loop to control the application.

  `motorDetect/lastMotorDetect` – Read the photo interrupter and detect the end of a revolution.

Setup

The `setup()` function sets the modes of the input and output pins, and initializes the outputs. The setup function initializes the application statemachine and sets an intial feeding for 10 seconds into the future.

Main Loop

The `loop()` function operates the state machine. The state machine has three states: *Waiting for Feeding*; *Delivering Food*; and *Finishing Feeding*.

In *Waiting for Feeding* the state machine checks the time against the next feeding time. Once the next feeding time is reached this state turns on the motor to deliver the food, then transitions to the *Delivering Food* state.

In *Delivering Food* the statemachine checks for the photo interrupter to generate a low to high transition, signalling the tab has passed though the interrupter. Once the low-high tranisiton is detected the state machine transitions to *Finishing Feeding*.

Additionally, in *Delivering Food*, the state machine checks whether the motor has been running for more than `MAX_DELIVERY_DURATION` seconds. If that is the case the state machine transitions to *Finishing Feeding*.

In *Finishing Feeding* the motor is stopped, the next feeding time is calculated (`FEED_INTERVAL` seconds after the last feeding time) and the state machine transitions back to *Waiting for Feeding*.

The state machine is implemented simply as a switch statement, with three cases.

Also in the `loop()` function the LED on the Arduino board is made to flash briefly once per second, to show that the application is still running normally.

TODO

Enhancements

The performance of the food hopper is, to say the least, disappointing. So, one alterations I am considering is to hopper with a *Screw Conveyor*. The food hopper is unnecessarily complex and unreliable. A screw conveyor should be a more accurate, less error prone and cheaper solution ($5 rather than the $20 I paid for the *fish feeder*).

Mechanical Design

As the pictures show, the mechanical design of the project are very much a prototype. Once the design is adapted to use the screw conveyor a better housing can be made, housing the electronics and more suitable for deploying on a variety of different tank sizes.

Links

http://sagar.org/workbench/frogfeeder

Website for this project, with source code and project document

http://www.arduino.cc/

http://www.nteinc.com/specs/3000to3099/pdf/nte3040.pdf

http://www.nteinc.com/specs/3100to3199/pdf/nte3100.pdf

http://www.nteinc.com/specs/200to299/pdf/nte210.pdf

http://www.jameco.com

http://en.wikipedia.org/wiki/Screw_conveyor

Table of Components

Component	Value	Notes	Supplier
IC1	NTE3040	Optoisolator	Fry's
IC2	NTE3100	Photo Interrupter	Fry's
TR1	NTE211	PNP General Purpose, Medium Power Transistor	Fry's
R1	1k0	Current Limiting	Fry's
R2	470R	Current Limiting	Fry's
R3	1k0	Current Limiting	Fry's
Protoboard		GCElectronics "Experiementer's Phenolic Proto Board"	Jameco
Motor & Gearbox	Item 70103**660	Tamiya Universal Gearbox	Fry's
Arduino	Arduino "Deicimila"		