DEVELOPMENT OF AN AUTONOMOUS ROBOT

A Submersible Collecting of Coins within a Fountain

Final Report

EEL5666 – Intelligent Machine Design Lab

Instructors: Dr. A. Antonio Arroyo, Dr. Eric M. Schwartz

TAs: Mike Pridgen, Thomas Vermeer

Robot: COin IN Fountain RObotic Gatherer (Coin Frog)

http://plaza.ufl.edu/jeske/imdl/

By: Jonathon Jeske

TABLE OF CONTENTS

ABSTRACT

Coin Frog is a submersible driving robot that collects coins on the bottom of a fountain. It utilizes a 12VDC battery source to power a microcontroller, two motors, four LEDs strips, a centrifugal pump and various bump switches. It drives autonomously by reacting to bump sensors such that it avoids obstacles and finds the fountain edge. Coin Frog will collect all types of coins and maintain a large payload until it senses that its collection bin is full or the task has timed out. It was designed to be interactive with a person and fully deconstructible for servicing. All sensitive electronics are self-contained while other, less-sensitive electronics are completely exposed to the water. Coin Frog only works in clean, freshwater.

EXECUTIVE SUMMARY

Coin Frog is a submersible driving robot that collects coins from the bottom of a fountain. It moves using two sealed DC gearhead motors and has an inline bilge pump system to create a pressure difference that sucks the coins up into a removable bin. It uses bump sensors for obstacle avoidance as it drives randomly along the bottom of the fountain. It utilizes a customized Atmel Atmega128 microcontroller for processing behaviors and controlling its functions. The board and all other sensitive electronics are kept within a polypropylene container sealed with a face-surface o-ring. LEDs and an LCD screen are used as feedback to the supervisor of the process for when interfacing with it is necessary, i.e., bin is full, obstacle hit, bin connected, water is in the sealed container. It is powered using a 12VDC sealed lead-acid battery that is kept within the same dry container. All connections through the water are made using shrink-wrap and press-fit hole sealed with marine epoxy and/or high-temperature hot glue. Coin Frog can work in up to 16” of fresh water and collect coins as large as quarters for predetermined time intervals.

INTRODUCTION

The idea of a robot that could collect change out of a fountain came to me after researching previous projects on the IMDL course website. I came across *Change Collector* by Rafael Garcia in the Summer 2007 class. It was a robot that could pick up a coin, sort it and then display the amount of money it had stored. I decided it would be interesting to expand on his idea by completing a task I currently have not seen an easy manual solution for – collecting the coins thrown into fountains.

While sharing a basic money gathering concept, this task is inherently different from Rafael's in that it is performed underwater. Creating a robot that cannot be completely confined to a single sealed container (due to external controllers and sensors) means that power needs to delicately segue from the sealed environment to the wet one; other concerns involve added drag resistance, buoyancy and sensor limitations (i.e., infrared cannot be used for obstacle avoidance due to the bending of the light). Such complexities have been addressed in this paper and solutions surrounding the mechanic, electronic and software requirements have been outlined in what may not be the best way to reach the goal, but has been proven as effective.

INTEGRATED SYSTEM

Coin Frog uses a modified Atmel Atmega128 designed by the two TAs of this course – Thomas Vermeer and Mike Pridgen – as the driving force behind the entire system. The microprocessor is connected to every input and output and is in control whenever powered. Once the power and override switches have been enabled, the Atmel begins sending signals out to the motor driver (which in turn sends them to the motors) and feedback out to the multiple LED strings (two red, one yellow and one green) and LCD screen (for debugging purposes only). It constantly checks the system sensors using external interrupts – water sensor, obstacle bump switches, bin attachment switch and payload limit switch – waiting to change the system state and possibly therefore motor direction, current goal and system power. Figure 1 shows a high level overview of Coin Frog's components and how they interact.

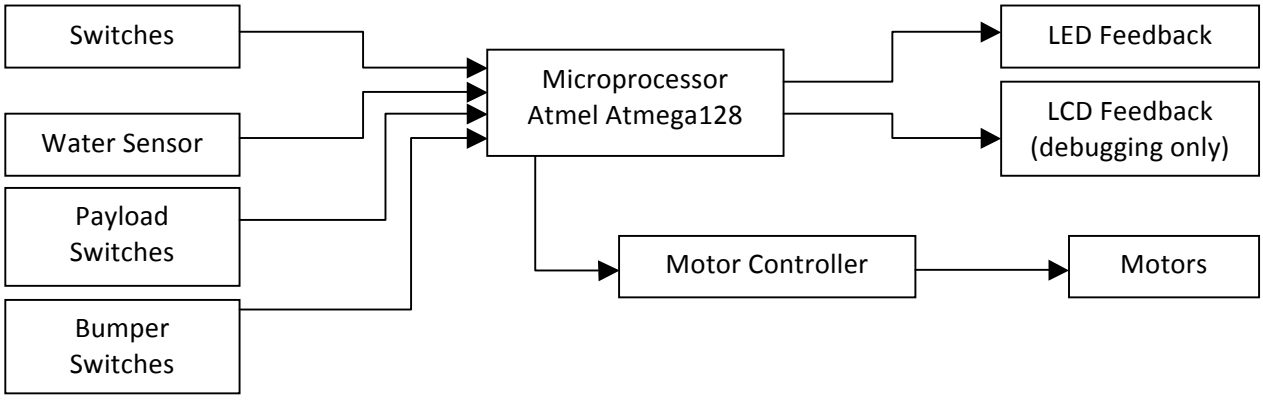

Figure 1. High level overview of Coin Frog's system

All permanent components are 12VDC source and required the use of transistors to control through the microcontroller.

MOBILE PLATFORM

The platform was designed around the requirement of a safe (dry) environment for the most sensitive electronics. To reduce complexity of the system, certain components were tested in an underwater environment without protection. It was determined that not all parts required an allotted location in the dry container and were thus removed from the packaging outline – mainly the DC motors and all switches.

Once the required space was determined, a suitable container needed to be found. Expensive submersible cases were researched, including Pelican Cases, but a transparent case with the

required size constraints could not be located. The idea to use a Tupperware-style box came to me after reading “air tight” on the Lock & Lock containers I use to seal my pasta in the kitchen. After initial depth testing, the provided o-ring seal proved to be submersible to at least two feet (it was not tested any deeper). The HPL826, at 248x180x93mm, was an ideal size, giving the battery a snug location while leaving adequate room for an electronics stack that sat separated from the interior wall – a necessary design constraint that might save the internal components if there was ever a slow breach (see Figure 2).

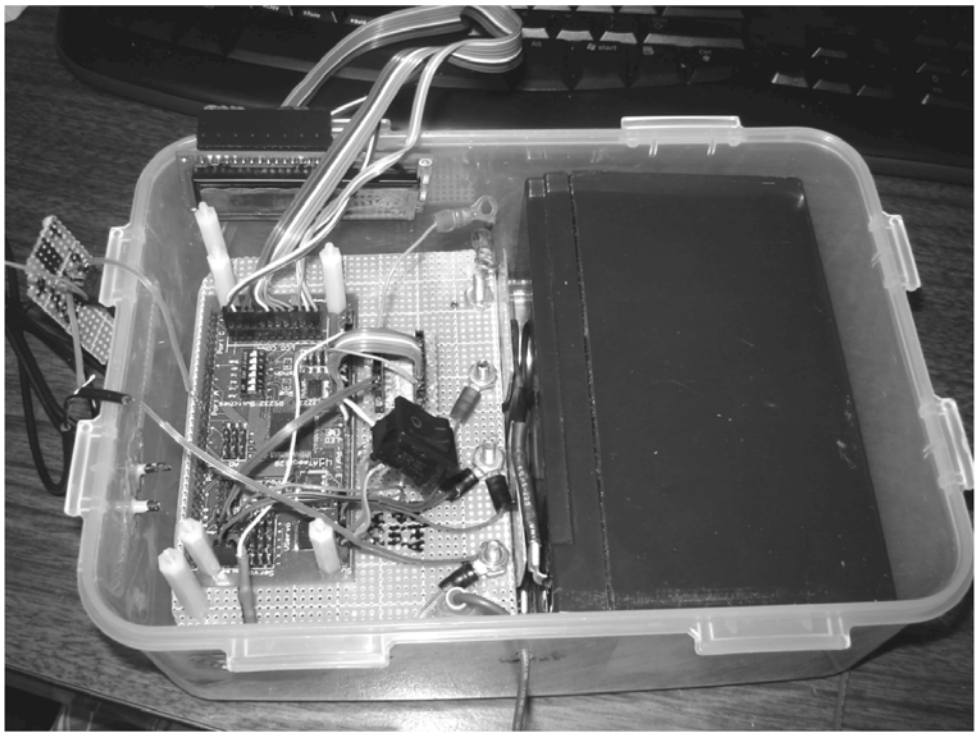

Figure 2. Internal structure early in the design stages

To allow connectivity between the external and internal components, press-fit holes were drilled for wires and sealed using both marine epoxy and high-temperature hot glue. To ensure that any component or wire could be inspected and/or replaced, the entire internal structure can be deconstructed and removed. This required careful planning when adding new pieces and routing wires and the use of multiple redundant connectors.

Once the electronics dry box was official, the remainder of the platform could be designed around it. The length constraint was a pump system requirement (to be further discussed in ACTUATION). The pump system had to have a completely removable bin so that the coins could be removed from the robot and this required a series of different PVC connections, all taking up their own length (see Figure 3). Once all testing of the pump system had been complete and it was determined that it had been optimized enough that it would pick up all change without a

problem, the base to the platform was drawn up and cut out of 3/8” acrylic sheet (Figure 4). The base was designed with a number of things in mind: it needed to work well for a bump-sense-only robot – allow room for a second tier such that no components would put stress on the top (and thus the o-ring) of the sealed container keeping the box free to be opened up for diagnostics – be very strong as a plastic material – and most importantly, look attractive.

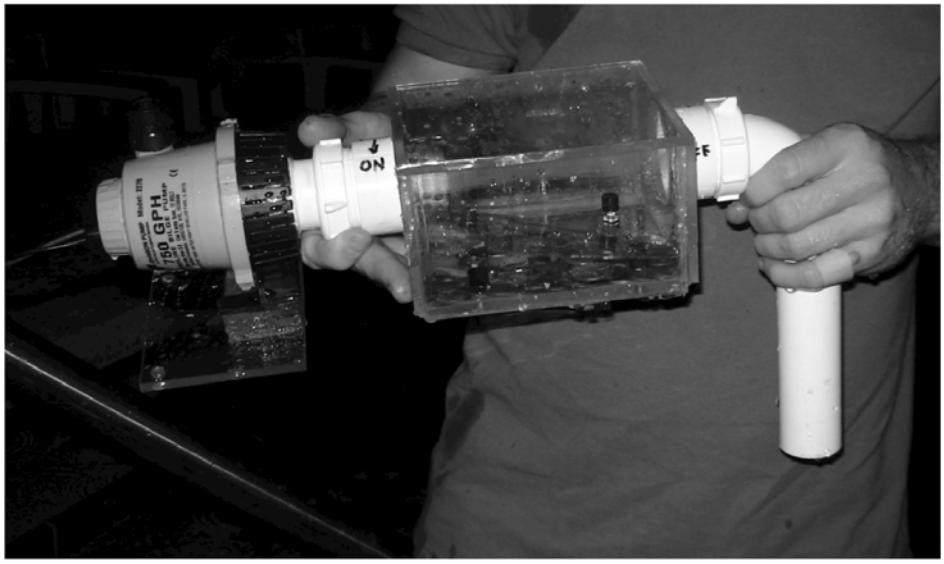

Figure 3. The pump system

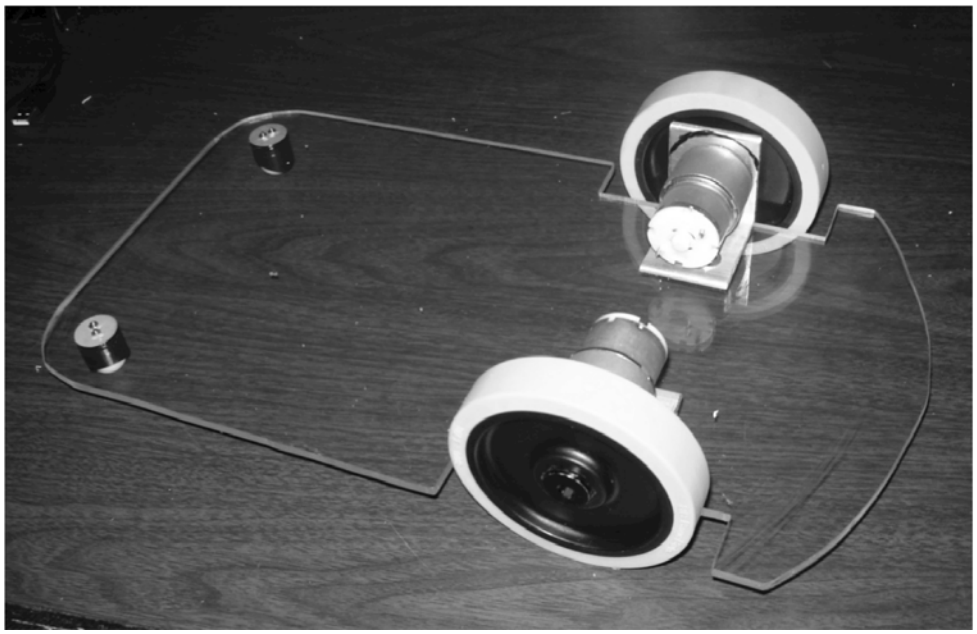

Figure 4. The base to the robot’s platform

Keeping the wheels inside the perimeter of the platform was a way to simplify bump sensor interference by allowing straight bumpers. Ball casters were chosen because the robot was very

heavy when not in water and required the stability of both corners while maintaining total rear-movement freedom.

To allow the second tier to be both adjustable during testing and removable, a set of six threaded rods were placed equally around the perimeter (see Figure 5). Stainless steel and aluminum was used were plastic components could not be located. Unfortunately, the threaded rod itself could not be located as anything but regular steel and thus requires quick drying after every submersion.

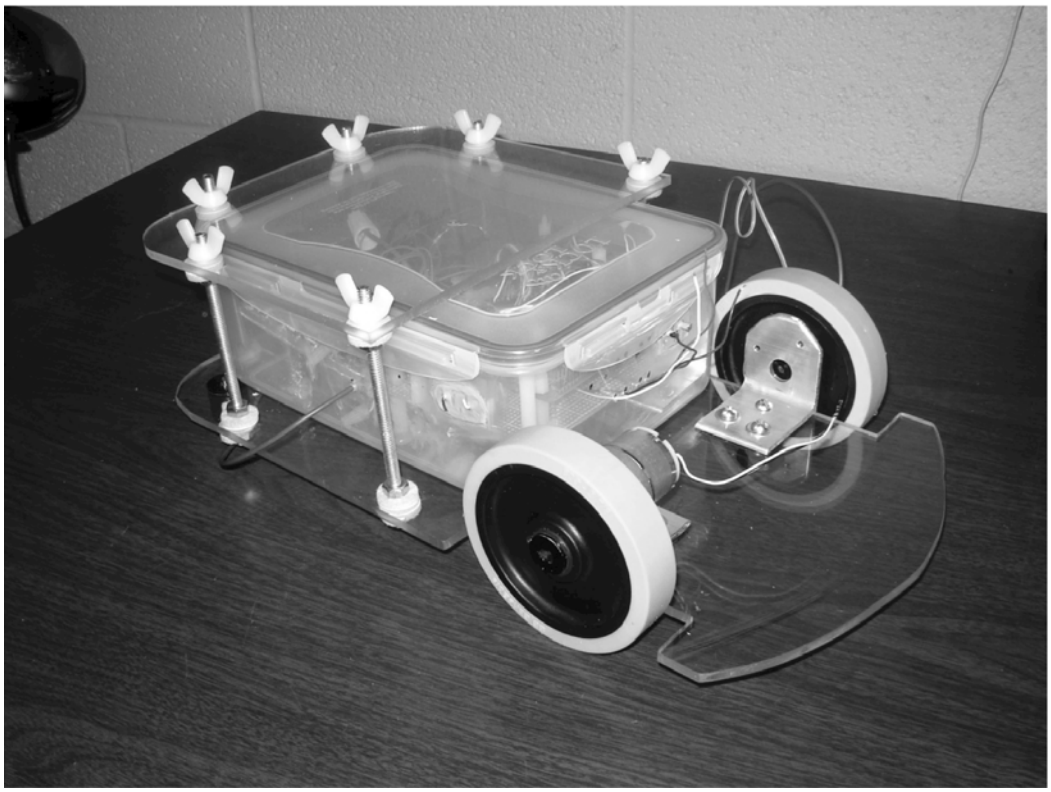

Figure 5. The robot's platform with the second tier attached

Nearly every part to the robot was designed in some form of plastic to increase longevity and buoyancy. While the robot may be heavy dry, it requires a 2.5lb weight to keep it negatively buoyant. This was designed on purpose so that the motors would not need to be incredibly powerful and so that the location of the majority of the weight could be placed over the drive wheels. As a drawback, it does inhibit accurate testing results out of the water since the motors have a much harder time driving the system.

Once the second level was mounted, the pump system could be permanently attached (see Figure 6). The design of the pump system also allowed for complete deconstruction. The pump was attached to its own platform so that it did not need to be fixed to the second level. A hole

in the bottom of the very front of the platform was carefully cut to allow the collection tube a press-fit. There is still enough movement that the tube can be turned sideways or removed if needed, but the friction holds the tube's adjustable depth, allowing for a precise mount and the best height for collecting the coins from the surface.

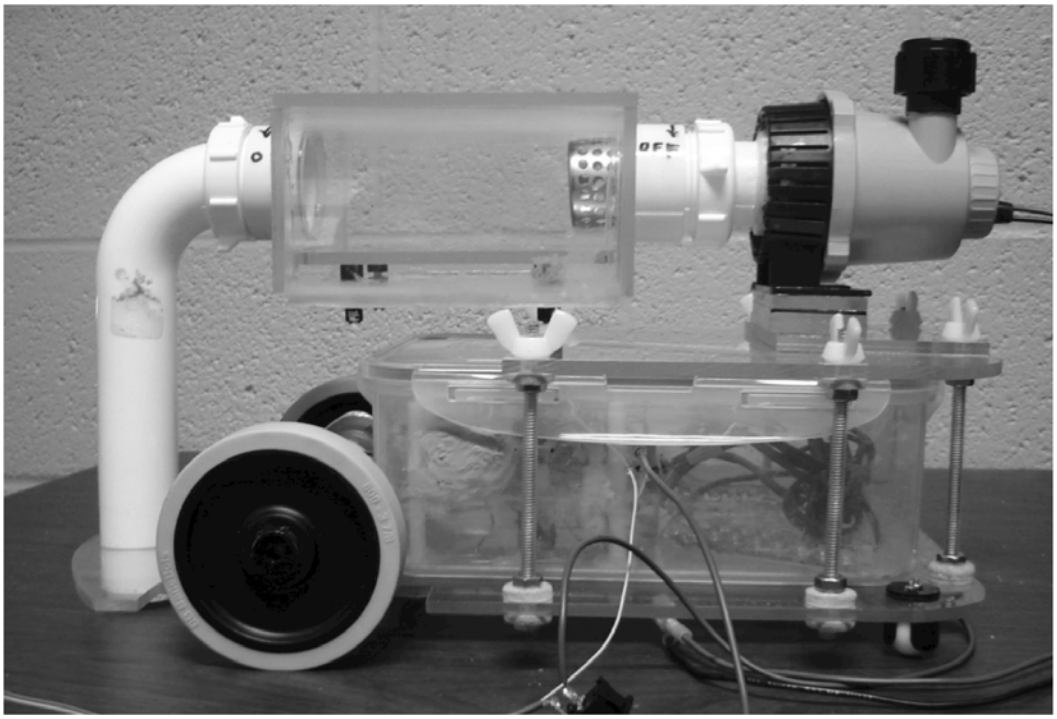

Figure 6. The robot's platform with the pump system attached

A handle was added after the initial bottom plate had been cut out (see Figure 7). The idea was that the robot needed a faster retrieval mechanism, instead of leaving it stuck on the bottom, possibly in the middle, of the pool during an emergency. It also gives an easy way to carry it and a dry location for the power switches – giving the user total control over it, once again, just in case. The most difficult part was that the handle needed to support about 20lbs at the mounting location while still being removable and allowing access to all other components. This was accomplished by splitting the handle before the second tier and making the wire connections permanent through the bottom but disconnectable through the handle itself. To ensure that it would not snap the acrylic base when being carried, 1/2" PVC pipe was used – allowing the load to be spread. Unfortunately, since the handle was an afterthought, there wasn't enough room to fully accommodate the handle at the base. It was instead connected across the base and then the gap between the two pieces filled with epoxy putty. This gave a really strong connection to the base. The handle was also located slightly closer to the middle of the robot than the battery. Since it was known that additional weight at the wheels would be

necessary to sink the robot to the bottom, the center of mass was designed to be almost exactly center – allowing a comfortable way to carry the robot.

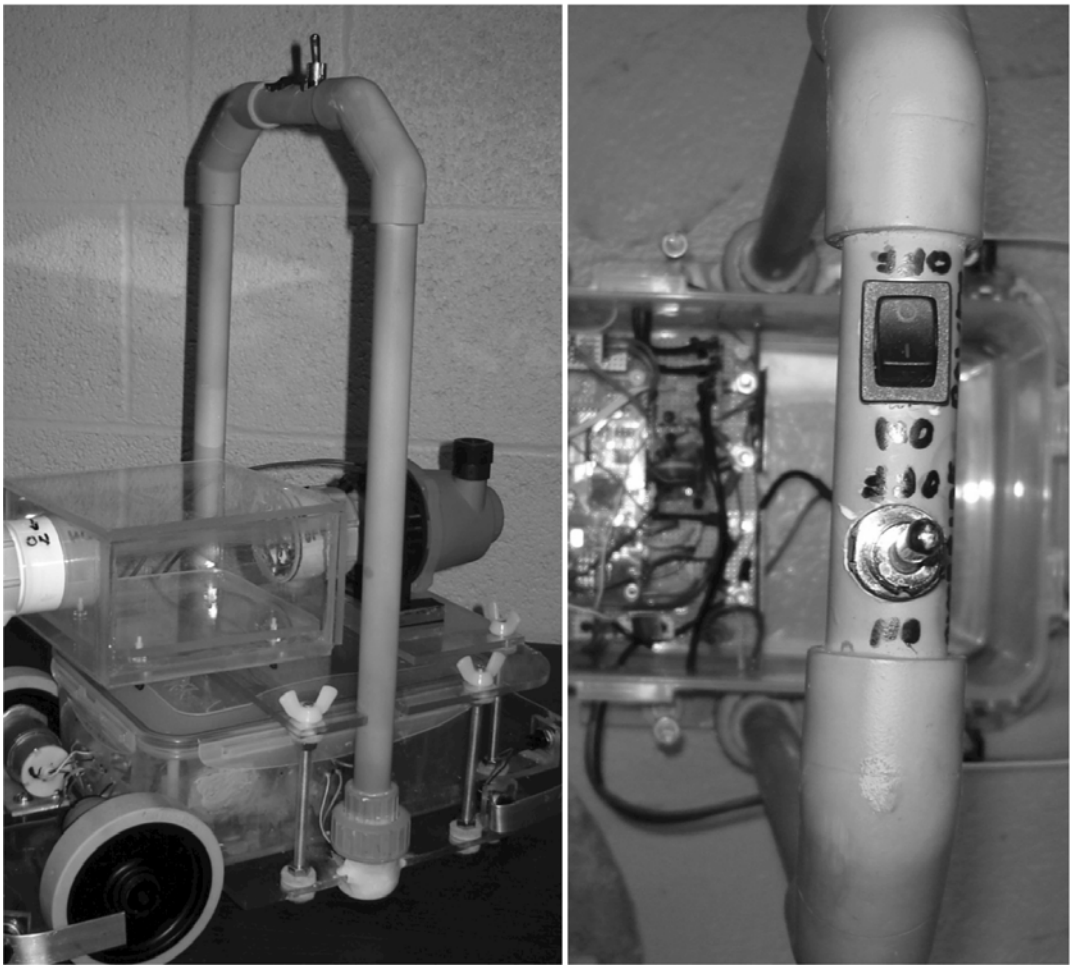

Figure 7. The handle and power switches of the robot

Finally the platform had some components painted a green color to match the wheels appropriately as the robot's name is Coin *Frog*.

Lessons Learned

Interestingly enough, the majority of the platform came together quite nicely without any CAD support. There was a long process leading up to its actual creation due to the sensitive nature of how it all fit together. One thing that severely delayed me was trying to use a manufactured bulkhead connector instead of press-fit holes for wiring. I spent 2-3 weeks with a manufacturer's salesman trying to get the correct pieces ordered only to find out that he didn't have a complete knowledge of the submersible series connectors or their availability. I ended

up scrapping the whole idea due to ridiculous lead time and the requirement of large-quantity orders.

Another small problem is that the plastic epoxy I used for a number of components seemed to lose adhesion when placed underwater. The first fountain Coin Frog saw was a very violent one and I had the bin break from its outlet to the pump. I replaced most of my permanent connections with hot glue and haven't had a problem since.

ACTUATION

Coin Frog is driven through the water using two Jameco 253526 12VDC motors. These motors have a low RPM (36) and high torque rating (4.7 in-lb) while pulling 2.9A at stall. These motors were chosen because it was known that the robot would be very bulky and require a speed no faster than 1in/second. The slow speed was determined from testing with the pump system – any faster and the coins could not become entrained in the flow.

The motors were controlled by the Atmega128 through a PB6612FNG dual channel motor controller (see Figure 8). This motor controller was determined by the stall current of the motor and the system power requirements. A PWM signal was sent from the Atmega128's PORT B using modified code provided by a classmate, Joe Bari. The motors were set up to run forward and backward together or individually.

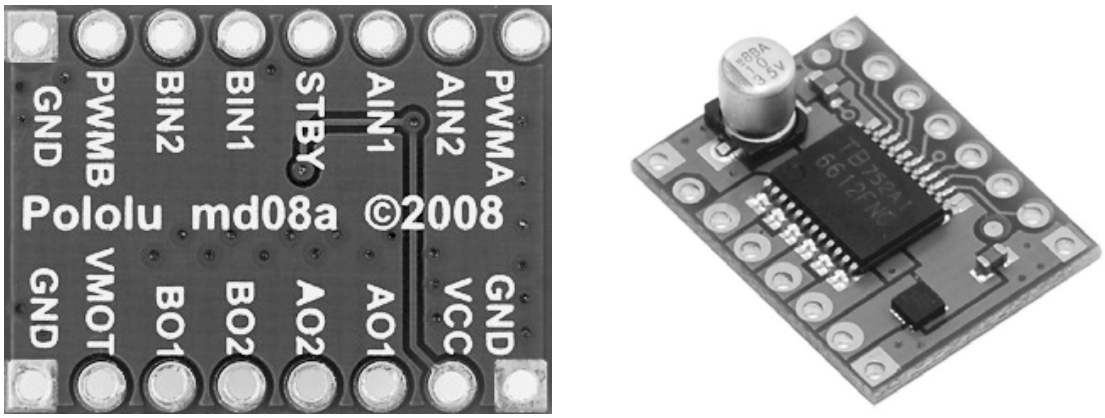

Figure 8. Diagram of PB6612FNG motor controller (from http://www.pololu.com/)

Wheels were chosen based on availability and adaptive capability with the motors. Since the motors came with a shaft terrible for transferring power to a wheel (circular with a through-hole), a mating hub that worked well with the wheel was very important. Fortunately, Robot Shop's online website has a huge variety of customizable wheel set-ups – including diameter, rubber hardness and mounting options. The required mounting height of the motor on the base

platform was measured for wheel clearance thus determining what height wheel was needed and the thickness of the wheel was determined by the clearance on the interior cut in the base's design – a matching ball caster also had to be found before ordering. The 3-7/8"x0.8", 3/4" hex mount with soft rubber wheel mated perfect to an available hex hub for a 6mm shaft, both at the perfect height for the 3/4" ball caster from Pololu. The softest rubber was chosen to allow for maximum grip.

To collect the coins, a pump system was developed with the help of one of Dr. Lear's graduate students, David Tiffany. Original plans were for a jet-pump style collection, but after discussions, it was determined to simply use a submersible, centrifugal pump inline with the system (see Figure 9). The size of the pump was determined from Bernoulli's equation. Knowing the flow rate and cross sectional area of the inlet, the velocity of the flow can be determined. This velocity in turn gives the pressure which can be related to the maximum weight pulled in by the pump. This amount from a 750gph pump in ideal conditions was determined to be around the order of a pound. Since the heaviest coin only weighs a fraction of an ounce, the Mayfair 22702 submersible bilge pump was chosen to match that flow rate. Unfortunately, this is not an ideal world and losses are encountered in nearly every aspect of the system. The calculations could only prove that that pump was far over-designed and could allow for those losses.

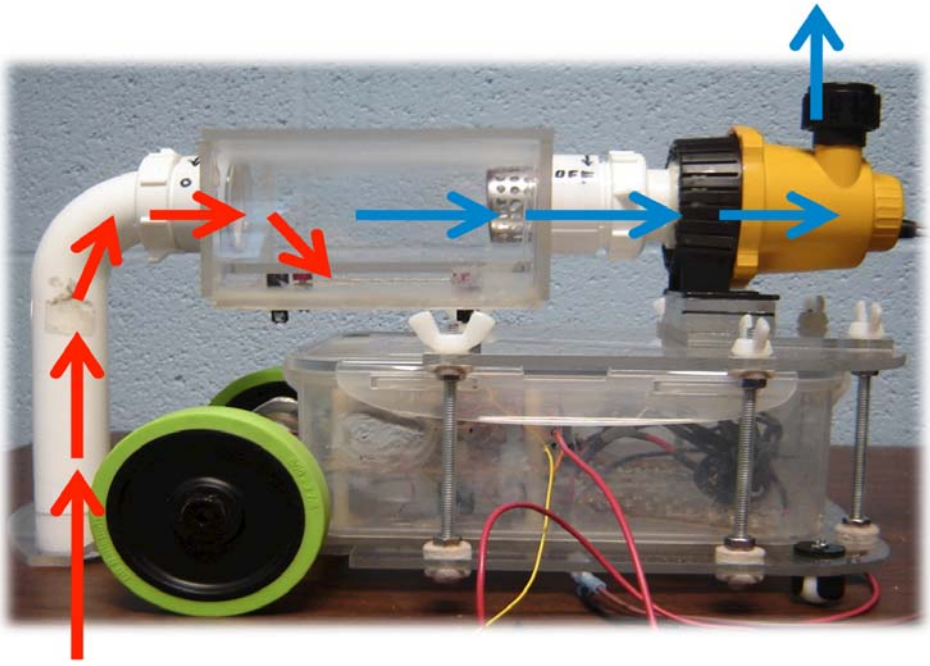

Figure 9. Schematic of how the water and coins flow through the pump system

Since the pump is specified to be powered at 12VDC at 3A nominal, a large battery was necessary to feed the entire system. A 12V 7.5Ah sealed lead acid gel-type was used and took up nearly half the entire electrical dry container. Since the pump would either be on or off, a motor driver wasn't necessary to control it – however an NPN type transistor acting as a switch had to sit between the pump, power and the microcontroller to allow the board to control its ability to turn on and off.

Lessons Learned

Some issues arose after initial testing of the pump system that were not keeping the coins entrained. In fact, most coins would only become completely trapped inside the bin when the suction side of the system was placed on the bottom of the pool being tested in. This was due to added head loss from the system design (long inlet, poor curve design and heavily restrictive filter) and an energy loss at the outlet. The pump has to try too hard once the coin was sucked up to keep it. To fix this, I shortened the inlet by a few inches, rapid-prototyped a new filter that had holes in it slightly smaller than a dime and rapid-prototyped a diffuser for the outlet of the pump. The diffuser helped to ease the water out of the pump, lowering the energy losses. This part of the design was also helped by David Tiffany. When all of this was put together, the pump system did a much better job collecting the coins.

Also, I burned up two motor controllers before finally making the system safe. The first one I learned that I should not probe pins on a board using a multimeter when ground and power sit next to each other. I shorted power across to ground and smoked the chip on the motor controller. The second time, the header pins fell slightly out of the header connector and crossed long enough to cause the chip to start on fire. Since I was now aware that my pin setup was working correctly, I hot-glued all the wires in their place and checked that there was no continuity across pins that shouldn't have it. There have been no problems since.

SENSORS

Obstacle Avoidance

All sensors used on Coin Frog are switch-style sensors. This was due to the expensive nature of underwater electronics.

For obstacle avoidance, it was necessary for the robot to have the ability to drive to the very edge of the fountain to collect the maximum amount of coins. This means that bumpers were used with momentary switches mounted directly to the front and rear of the robot (see Figure

10). Unless the robot was to be picked up by violent waves, there was no feasible reasoning for side object detection. The bumpers had a high enough spring constant to them that the force of the water wouldn't set them off but soft enough that a children's pool could be used as a demonstrative fountain without pushing the sides of the pool over.

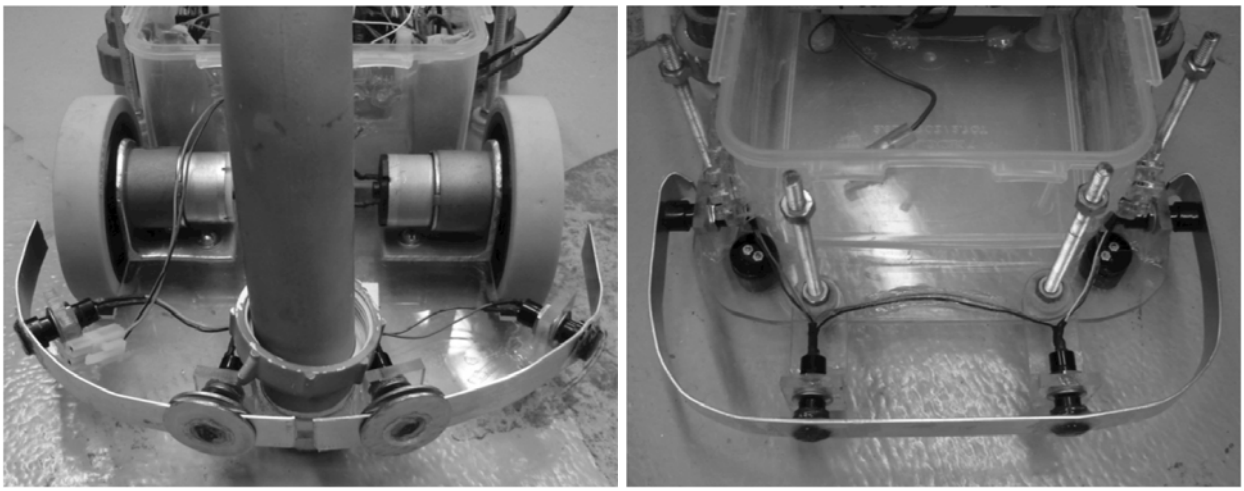

Figure 10. Front and rear bumpers for obstacle avoidance

The bumpers were fabricated from thin aluminum bar allowing for maximum rigidity while still being light-weight and virtually water-proof. The front was done in two separate bumpers because it required the added support from a second button. All momentary switches are mounted into the bumper itself for robust construction. The front-most buttons are placed within a slotted mount so that the center point on the bumper can still be depressed. The rear design is much simpler – the side switches are not attached. Instead, the ends of the bumper are folded over on the platform for support. This allows for free-movement with less hassle.

While there are eight switches, they act as only four. Precise obstacle avoidance was not as necessary as random avoidance behavior. Therefore, it is known if it hits the front right/left or rear right/left. There is some overlap with the rear sensing, but it does not affect the robot's performance.

The momentary switches themselves are Radio Shack of-the-shelf purchases and are not water-proof. They showed signs of internal rusting after the first submersion – however, they are inexpensive easily replaced and do not have connectivity across the switch while underwater.

The switches are given 5V power from individual pins on the Atmega128 and monitored by four separate external interrupt pins using low-level detection. When the switch is depressed, the

monitored pin goes to ground and sends the signal to the board to change the state (more about this will be discussed in the BEHAVIORS section).

Payload Limit

To control the maximum payload weight of the robot, a sensor plate was installed that will activate a "Bin Full" behavior when too many coins are in the bin (see Figure 11 and Figure 12). This task was created to not only allow for a better idea of what the motors' required torque rating would be, but to add an interesting and challenging twist to the robot's behavior beyond the requirements of the course that would not inhibit the core behavior.

Figure 11. Side front view of the sensor plate in the payload bin

Figure 12. The final sensor plate in the payload bin

The final sensor plate was designing in Pro-E and built in a rapid-prototype machine. It is secured to four small momentary switches (miniature versions of the obstacle avoidance momentary switches also bought from Radio Shack off-the-shelf) that are connected in series so that if any switch is activated by the weight of the payload pushing down, the external interrupt pin being monitored is polled (they work together in the same manner as the obstacle avoidance momentary switches). The plate was designed such that the thinnest coin – a dime – cannot become lodged between itself and the side of the bin or fall through any of the designs in the plate. To remove the container to empty the coins, the user simply has to unscrew the PVC connections on both sides and disconnect the molex-style connector attached to the bottom, right corner of the bin. The connector has been filled with hot-glue, so power will not be conducted across the pins while in tap water.

Bin Connected

Due to the fact that the container is removable during system use, it is desired to know when the bin has been placed back into its proper position. For this reason, another switch was mounted to the top of the second tier level of the platform that is depressed whenever the bin is in place (see Figure 13). This momentary switch does not require an interrupt but is instead used during the process of emptying a bin to let the robot know that it's ok to continue back to its fundamental goal.

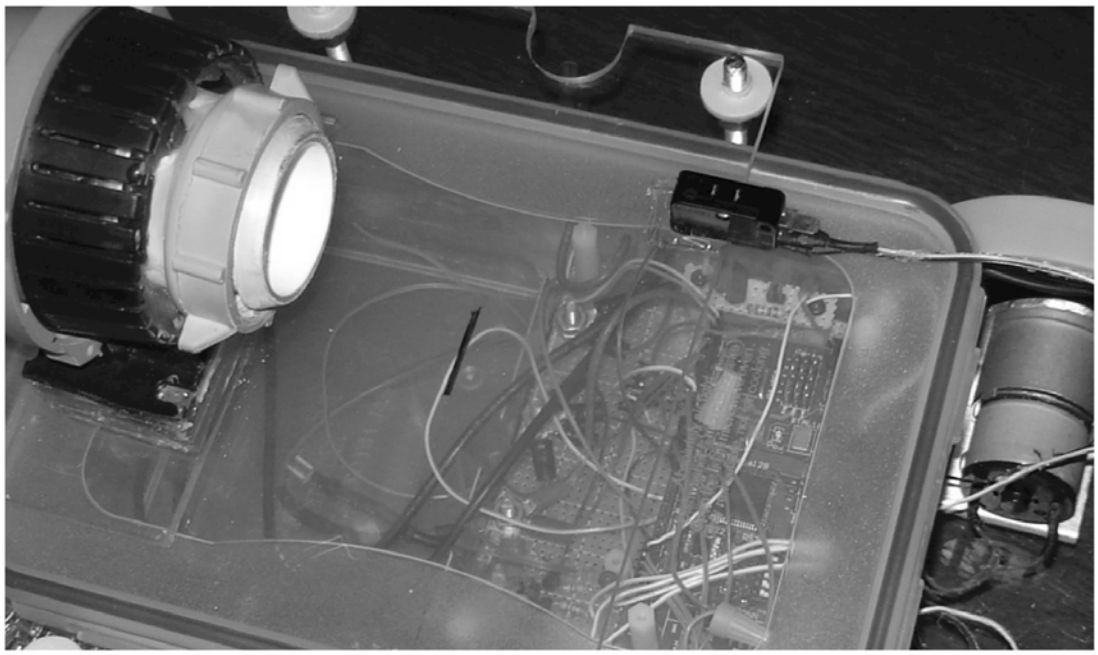

Figure 13. Momentary switch mounted to the top of the second tier to check for the bin

The switch was given enough wire to allow the second tier to be moved out of the way of access to the bottom levels, including the inside of the dry container.

Water Sensor

Probably the most important sensor to an underwater robot is the ability to check for a breach in the dry container. All of the power from the battery goes through a controlled, normally-open relay before reaching the terminal posts for any component – including the microcontroller. This allows the microcontroller itself to control its own power in case of emergencies. Two wires are laid out in a concentric circular pattern along the bottom of the dry, electronics container – one given VCC 5V power and the other attached to ground (see Figure 14). The wires have a 2mm gap between them that conducts current only when immersed in water. External interrupt 0 – the one with the highest processor queue priority – is connected to the 5V source and set to be polled during a low-level event, indicating connectivity across the wires due to a water breach. This interrupt is told to do only one thing – kill the power to the entire port powering that relay to stay closed. This cuts the power connection to the power bus, thus removing power from the system as an emergency effort to save the electrical components housed in the container.

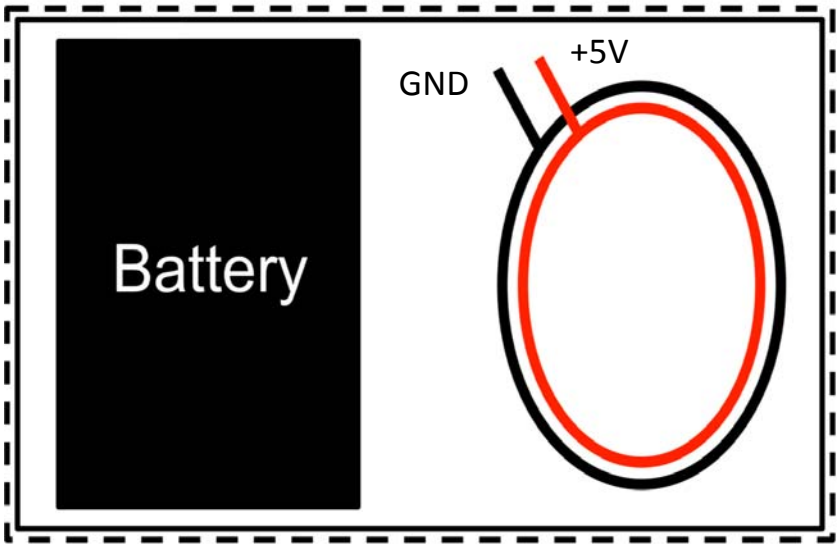

Figure 14. Location and shape of the water sensor located on the bottom of the container

Lessons Learned

Initial construction of the front bumper didn't have slots and therefore had an extremely stiff depression point at the center of the bumper. It was an easy fix, but I should have thought about it prior to constructing it.

The original sensor plate was cut from the 3/8" acrylic used to build all other flat surfaces. Unfortunately, I had made some errors when cutting the bin that caused imperfections in the dimensions. This meant that the plate was too small (and I was out of material) and coins were constantly getting stuck in the sides of the bin. I had to measure 10 times in multiple locations of the bin to ensure I sized the second plate in Pro-E properly for the rapid-prototype machine because it wasn't parallel or perpendicular to any other edge. I ended up making it the exact size of the hole and had to sand it down until it fit without rubbing. I'm quite proud of the design cut into it as well – my first time using styles to cut out an image.

During testing of the limits of the sensor plate, I found that four switches would produce connectivity with only a handful of coins because the switches were so soft. I assumed that would transfer well into an underwater environment where the coins weigh less and the plate has a positive buoyancy force against the coins. Unfortunately, it appears that those difference account for a very large change in the amount of coins necessary. It now takes a half full bin to set the behavior off which is unfortunate because the behavior works well when triggered. To fix this, I need to add a lot of static weight to the plate but it's very difficult to get a constant reading of how much because the switches fluctuate too much.

BEHAVIORS

Coin Frog runs through a quick initialization process and then works on a state based algorithm (see Figure 15). The initialization is a simple component life check – LEDs are working correctly, bin is connected, motors are working, pump is working and then begin. When it starts, its default state is to drive around with the pump on flashing the green LED strip. It stumbles over coins and collects them in the same manner an off-the-shelf vacuum robot cleans around your house. In this case, it wants to reach as close to the edge of the fountain as possible, so it goes until it hits it. The interrupt will let it know where it hit, change the state, disable the interrupt and then enter that new state where it moves away from the direction of the object and turns a randomly determined amount of time. After that time, it re-enables that interrupt and goes back to its default state. A bin full interrupt works the same way, but it disables all bump interrupts and stays in that state much longer. The bin full state is set to drive to the edge of

the fountain and stay there until the bin has been removed and then replaced into its location. Only then will it reenter the default coin collection state.

A breach is treated completely differently – once the interrupt is polled, the code is directly within the interrupt to turn the power off to the robot. This means the main function never sees it coming and the delay is not there.

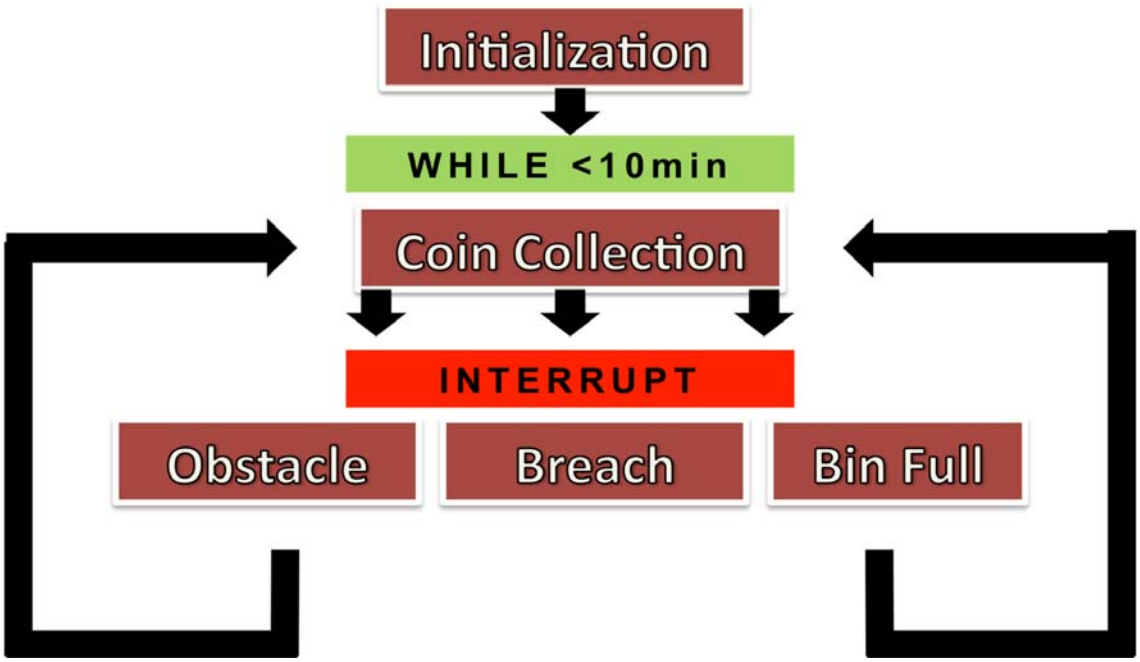

Figure 15. High level algorithm process of the robot

Once the timer counter has reached its given while goal (typically 10minutes, but reduced for demonstration purposes to as short at 2minutes), the while loop is exited and it run the same process of driving until it reaches an edge as it does for the bin full. This driving is to make retrieval of the robot easier. Once at an edge, the robot will sit there for 20 seconds simply flashing its lights letting the user know it has completed its task.

Lessons Learned

Getting the LEDs to flash was not a trivial task. My initial attempts failed since they would take over the microprocessor until done flashing. My thanks go to Nicholas Johnson for helping me through the process of a timer/counter interrupt. All four LED strips are controlled through an XOR state system check that allow me to control which LEDs I want on at any given time.

It's also very difficult to determine the timing for behaviors of an underwater robot when not testing underwater. Most of the behavior code was an educated guess, then tested above water and mostly worked when submersed.

EXPERIMENTAL LAYOUT AND RESULTS

Testing of an underwater robot is very difficult when without proper facilities. The robot underwent dry testing until the final week before demonstration. Systems were checked as most robots are by lifting up the wheels and watching its reactive behavior. All reactions were fully capable of being tested dry, however, minor tweaks were necessary after the initial underwater testing – mostly due to wheel slip and buoyancy.

Component testing was the key during development. The system was tested in pieces multiple times before a full system test was done. For instance, pump testing did not require the obstacle avoidance system and vice versa – it was much safer to approach the problem this way. Pump testing was done at one of the pools located at Rockwood Villas (see Figure 16). The initial pump designs were progressively checked for coin collections capability until finally perfected.

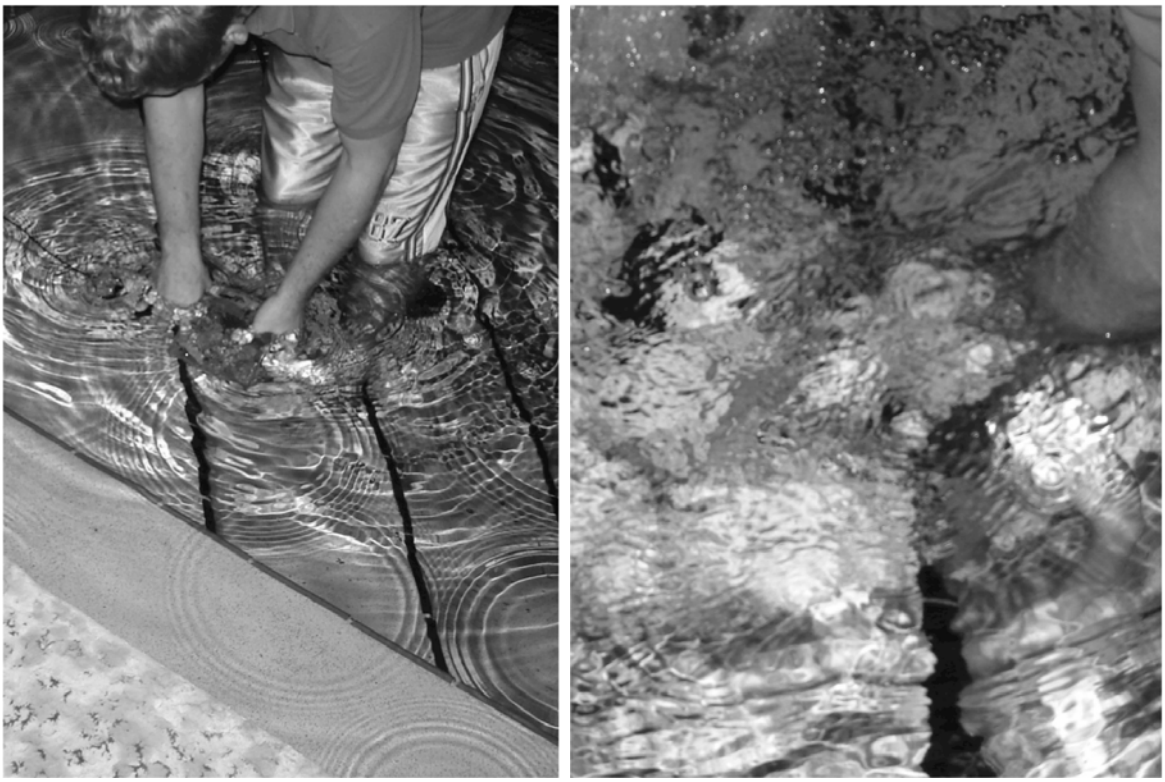

Figure 16. Initial pump system testing underwater

The initial plunge was taken in the Machine Intelligence Laboratory's water tub built for their underwater, autonomous submarine (see Figure 17). Coin Frog was dropped in multiple stages to test for breaches before a level of comfort was reached and real testing of all systems could be done.

Figure 17. Initial underwater test of the robot

The platform was first tested with all sensitive electronics removed from the dry container and checked for leaks. Once all leaks were removed, the microcontroller and battery were installed and code was allowed to be run – simple obstacle avoidance with LED feedback. When that was successful, the pump, then its system was added in stages. When everything proved to be working – the code was tweaked for slippage, the proper weight (2.5lbs) was added to sink it and it was taken out to a real fountain to be tested.

The only fountain available that was within Coin Frog's specifications was located by the Performing Arts Center (see Figure 18) – all others were too deep. As can be seen, the fountain it was tested in was very violent. The robot had a difficult time staying oriented and thus was losing coins a lot. By the time the testing was over, the outlet to the bin connecting to the pump inlet had been broken off. Aside from this, it was a complete success.

Figure 18. Testing at the Performance Arts Center fountain

Final testing was done the morning of the day of demonstration for class. This was the first attempt within the children's pool purchased for the purpose of the course. It was mostly a systems check due to previous submersion, but it was the first opportunity to check if the robot would bump sense the edges of the demonstration pool – and it did.

CONCLUSION

Overall, this project was a total success and I'm very proud of the end result. The entire semester was spent learning and I can honestly say I gained a lot by taking this course. A few of the things I had never attempted before: low-level soldering (which is much easier than previously thought), PC board development and creation, microcontrollers (this includes bitwise functionality, pin control, external interrupts, timer/counter interrupts, physical programming of a microcontroller, how low-level control works), using a transistor, using a MOSFET, using a capacitor to filter noise and using a rapid-prototype machine. This was not only my first robot, but I dare say my first real programming assignment. It's the most interesting feeling programming something that actually moves to provide feedback. I think my only real limitations during this project were finances. As it was, the robot cost over $600 to develop and build – but to do the job better and use underwater sensors, it would have been at least $2000. Even if the funding was possible, the time taken to build the platform would not have allowed for more advanced sensor development. The testing scenario would be very difficult without a dedicated facility. The only things I regret about this project are making the handle an afterthought (and therefore not leaving enough room for it and scraping the paint constantly) and not being able to use more attractive, bulkhead connectors. I don't regret my choice of submersible container, but if I had the opportunity, I would probably change it to a proper case designed for underwater use – if for nothing more than to reduce the stress involved with every submersion. I'm very lucky to have had CIMAR as a place for development and construction and I thank all those who helped me.

DOCUMENTATION

Atmel ATMega128 Manual
http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

Special Thanks:
Dr. Arroyo
Nicholas Johnson, CIMAR
Dr. Lear
Brandon Merritt, CIMAR
Mike Pridgen
Shannon Ridgeway, CIMAR
Dr. Schwartz
David Tiffany

APPENDICES

CODE

motors.h

```
#ifndef __motors_h__
#define __motors_h__

#include <avr/io.h>
#include <avr/interrupt.h>

void initMotors(void);
void driveStraight(int amnt);
void turnLeft(int amnt);
void turnRight(int amnt);
void brake(void);
void coast(void);

#endif
```

coinfrog.c

```
//-------------------------------------------------------------
// COINFROG.c is the main function run for CoIN FRoG
//
// Written by: Jonathon Jeske
// EEL5666 - Intelligent Machine Design Lab
// Spring 2009
//-------------------------------------------------------------

// PORT A = LEDs
// PORT B = Motor PWM
// PORT C = LCD
// PORT D = Interrupts
// PORT E = Power Relay
// PORT F = Pump
// PORT G = Unused

//-------------------------------------------------------------
// Includes
//-------------------------------------------------------------
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "LCD.h"
#include "sleep.h"
#include "motors.h"


//-------------------------------------------------------
// Main Function
//-------------------------------------------------------

volatile int state;
volatile int runtime=0;
volatile int shutdown=0;
volatile int edge=0;

int main() {

	// Set up PORT input/outputs
		// 1 = Output
		// 0 = Input

		// PORT E0 = Relay Source
		// PORT E1 = Bump Sensor

		DDRE = 0b00110011;;
		PORTE = 0b00110011;

		DDRD = 0b11110000;
		PORTD = 0b11110000;


	// Initialize Systems

		initSleep();
		initInterrupt();
		//initLCD();
		initMotors();
		initCoinFrog();

		// Set state to NORMAL = 6
		state = 6;
		runtime = 0;
```

```
while(shutdown == 0) {

switch (state) {

        case 1:

            // Front Left

            brake();
            ms_sleep(250);

            driveStraight(-85);
            ms_sleep(2000);

            turnLeft(-85);
            ms_sleep(randomTurnAmt());

            brake();
            ms_sleep(250);

            // Re-enable the triggered interrupts
            EIMSK = 0b00111111;

            state=6;
            break;

        case 2:

            // Front Right

            brake();
            ms_sleep(250);

            driveStraight(-85);
            ms_sleep(2000);

            turnRight(-85);
            ms_sleep(randomTurnAmt());

            brake();
            ms_sleep(250);

            // Re-enable the triggered interrupts
```

```
        EIMSK = 0b00111111;

        state=6;
        break;

case 3:

        // Rear Left

        brake();
        ms_sleep(250);

        driveStraight(85);
        ms_sleep(2000);

        turnRight(85);
        ms_sleep(randomTurnAmt());

        brake();
        ms_sleep(250);

        // Re-enable the triggered interrupts
        EIMSK = 0b00111111;

        state=6;
        break;

case 4:

        // Rear Right

        brake();
        ms_sleep(250);

        driveStraight(85);
        ms_sleep(2000);

        turnLeft(85);
        ms_sleep(randomTurnAmt());

        brake();
        ms_sleep(250);

        // Re-enable the triggered interrupts
```

```
        EIMSK = 0b00111111;

        state=6;
        break;

case 5:

        // Bin is Full

        pumpOff();

        while (edge == 0) {
                driveStraight(85);
        }

        brake();

        EIMSK = 0b00000000;

        // While switch closed, wait for removal
            while((PINE & 0b00000010) == 0) {
            }

        ms_sleep(500);
        state = 7;

        // Secondary check for bin reattachment
        while((PINE & 0b00000010) != 0) {

            // While switch open, wait for replacement
                while((PINE & 0b00000010) != 0) {
                }

            // Bin is now in place

            // Flash Green and Yellow signifying 10 seconds remaining
                state=9;
                ms_sleep(7000);

            // Flash Green and Red signifying 3 seconds remaining
                state=8;
                ms_sleep(3000);

// If the bin has been properly replaced, it will exit the bin removed loop
```

```
                        }

                        // Get off the wall
                        if(edge == 1) {
                                driveStraight(-85);
                                ms_sleep(1000);
                        }
                        if(edge == 2) {
                                driveStraight(85);
                                ms_sleep(1000);
                        }

                        // Re-enable the triggered interrupts
                        EIMSK = 0b00111111;

                        // Re-enter the pool area for resuming collection
                        if(edge == 1) {
                                driveStraight(-85);
                                ms_sleep(1000);
                                turnLeft(-85);
                                ms_sleep(2000);
                        }
                        if(edge == 2) {
                                driveStraight(85);
                                ms_sleep(1000);
                        }

                        state=6;
                        edge=0;
                        break;

                case 6:

                        driveStraight(85);
                        pumpOn();
                        ms_sleep(50);

}       // end switch
}       // end while

// Shutdown sequence:

state = 8;
```

```
        pumpOff();

        while (edge == 0) {
                driveStraight(85);
        }

        brake();

        EIMSK = 0b00000000;

        ms_sleep(20000);

        // Turn off power
        PORTE = 0;

}

//-------------------------------------------------------
// Interrupt Service Routine
//-------------------------------------------------------

// Timer for flashing LED lights and main loop duration

ISR(TIMER3_COMPA_vect) {

        // Increment time to 2400
        //      (0.1275 sec increments * 2400 increments = 300 seconds = 5min)
        //  (0.1275 * x ) = time in seconds

        if ((state == 1) | (state == 2) | (state == 3) | (state == 4) | (state == 6)) {
                runtime++;
        }

        if (runtime == 2400) {
                shutdown = 1;
        }

        // Reset counter for LED timer

        TCNT3 = 0;

        // Alternate LED power

        if(state == 1 | state == 2 | state == 3 | state == 4) {
```

```
		// For Flashing Red
		PORTA = ((PORTA & 0b10010000) ^ 0b10010000);
	}


	if(state == 5) {
		// For Flashing Yellow
		PORTA = ((PORTA & 0b00100000) ^ 0b00100000);
	}


	if(state == 6) {
		// For Flashing Green
		PORTA = ((PORTA & 0b01000000) ^ 0b01000000);
	}

	if(state == 7) {
		// For Flashing Red and Yellow
		PORTA = ((PORTA & 0b10100000) ^ 0b10100000);
	}

	if(state == 8) {
		// For Flashing Red and Green
		PORTA = ((PORTA & 0b01010000) ^ 0b01010000);
	}

	if(state == 9) {
		// For Flashing Green and Yellow
		PORTA = ((PORTA & 0b01100000) ^ 0b01100000);
	}
}

//  If INT0 is triggered low level, it will set the pins on
//	PORTD to 0, therefore opening the inline relay and
//  shutting off power to the entire system

ISR(INT0_vect) {

	PORTE=0b00000000;

}


// Obstacle Avoidance Interrupts for the Bump Sensors
```

```
ISR(INT1_vect) {

	// BUMPER_HIT_FRONT_LEFT

	if(state == 5 | state == 8) {
		edge=1;
		EIMSK &= 0b00111101;
	}

	else {
		state = 1;

		// Turn off INT1 while in INT1 so that the bump switch doens't
		//	continue to trigger the interrupt.

				// 76543210
		EIMSK &= 0b00111101;
	}
}


ISR(INT2_vect) {

	// BUMPER_HIT_FRONT_RIGHT


	if(state == 5 | state == 8) {
		edge=1;
		EIMSK &= 0b00111011;
	}

	else {
		state = 2;

		// Turn off INT2 while in INT2 so that the bump switch doens't
		//	continue to trigger the interrupt.

				// 76543210
		EIMSK &= 0b00111011;
	}
}

ISR(INT3_vect) {
```

```
	// BUMPER_HIT_REAR_LEFT


	if(state == 5 | state == 8) {
		edge=2;
		EIMSK &= 0b00110111;
	}

	else {
		state = 3;

		// Turn off INT3 while in INT3 so that the bump switch doens't
		//	continue to trigger the interrupt.

					// 76543210
		EIMSK &= 0b00110111;
	}
}

ISR(INT4_vect) {

	// BIN_FULL_LIMIT

	state = 5;

	// Turn off INT4 while in INT4 so that the bump switch doens't
	//	continue to trigger the interrupt.

				// 76543210
	EIMSK &= 0b00101111;
}

ISR(INT5_vect) {

	// BUMPER_HIT_REAR_RIGHT

	if(state == 5 | state == 8) {
		edge=2;
		EIMSK &= 0b00011111;
	}

	else {
		state = 4;
```

```
                // Turn off INT5 while in INT5 so that the bump switch doens't
                //      continue to trigger the interrupt.

                                // 76543210
                EIMSK &= 0b00011111;
        }
}


// randomturnAmt returns a value between 3 and 5 seconds for the
//      amount the robot will turn during obstacle avoidance to avoid
//      a patterned path

int randomTurnAmt() {

// TCNT3 is a timer/counter with a TOP of 1953
// To give one of five random turn amounts, 1953 is divided by 5 (almost equally)

        if(TCNT3<=390) {
                return 3000;
        }

        if(TCNT3<=780) {
                return 3500;
        }

        if(TCNT3<=1170) {
                return 4000;
        }

        if(TCNT3<=1560) {
                return 4500;
        }

        return 5000;

}
```

initCoinFrog.c

```
//-------------------------------------------------------------
// initCoinFrog.c initializes the robot's systems and
// gives feedback with colored LEDs
```

```
//
// Written by: Jonathon Jeske
// EEL5666 - Intelligent Machine Design Lab
// Spring 2009
//
// Green LED is held on during any component test.
// Yellow LED is held on to show a component test has been
//                  completed.
//
//-------------------------------------------------------


//-------------------------------------------------------
// Includes
//-------------------------------------------------------

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "LCD.h"
#include "sleep.h"
#include "motors.h"

//-------------------------------------------------------
// Functions
//-------------------------------------------------------

void initCoinFrog() {

        // 1.   Flash Red
        //              Flash Yellow
        //              Flash Green

                ledRedFlashing(3);
                ms_sleep(1000);
                ledYellowFlashing(3);
                ms_sleep(1000);
                ledGreenFlashing(3);
                ms_sleep(1000);
                allLedOff();

        // -------------------------------------------------

        ms_sleep(1000);
```

```
ledYellow();
ms_sleep(2000);
allLedOff();
ms_sleep(1000);

// 2. if  Bin Connected?
//                      Flash Green
//                      Keep Red and Fail Loop?

        if ((PINE & 0b00000010) == 0) {
                ledGreenFlashing(3);
        }
        else {
                while(1) {
                ledRedFlashing(1);
                }
        }

// ------------------------------------------------

ms_sleep(1000);
ledYellow();
ms_sleep(2000);
allLedOff();
ms_sleep(1000);

// 3.   Test Wheel Motors

        // Forward
        ledGreen();
        driveStraight(85);
        ms_sleep(2000);
        brake();
        allLedOff();

        ms_sleep(500);

        // Reverse
        ledGreen();
        driveStraight(-85);
        ms_sleep(1750);
        brake();
        allLedOff();
```

```
	// ----------------------------------------------------

	ms_sleep(1000);
	ledYellow();
	ms_sleep(2000);
	allLedOff();
	ms_sleep(1000);

	// 4.	Turn Pump on for 2 seconds

		ledGreen();
		pumpOn();
		ms_sleep(2000);
		pumpOff();
		allLedOff();

	// ----------------------------------------------------

	ms_sleep(1000);
	ledYellow();
	ms_sleep(2000);
	allLedOff();
	ms_sleep(1000);

	// 5.	Flash Red, Green and Yellow LEDs again as a READY

		ledRedFlashing(2);
		ledYellowFlashing(2);
		ledGreenFlashing(2);
		allLedOff();

	ms_sleep(2000);

}
```

initInterrupt.c

```
//-----------------------------------------------------------
// initInterrupt.c initializes the interrupt routine
//
// Written by: Jonathon Jeske
// EEL5666 - Intelligent Machine Design Lab
// Spring 2009
//-----------------------------------------------------------
```

```
//-------------------------------------------------------
// Includes
//-------------------------------------------------------


#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "LCD.h"
#include "sleep.h"

//-------------------------------------------------------
// Functions
//-------------------------------------------------------

void initInterrupt() {


	// Clear interrupts in SREG I bit
	cli();

	// Set interrupt type in register

		// External Interrupt Control Register A
		// Bits 1,0 are reserved for INT0
		// 0 0 = Low Level
		// 1 0 = Falling Edge
		// 1 1 = Rising Edge

			//	33221100
		EICRA = 0b00000000;
			//	77665544
		EICRB = 0b00000000;

	// DIEOE

	// Enable interrupts
	// INT	76543210
	EIMSK = 0b00011111;


	// Timer/Counter Interrupts for LED flashing
```

```
// Reset Timer/Counter 3
TCNT3 = 0;

// Set up Timer/Counter 3 Register
TCCR3A = 0b00000000;
        //      Bit 7:2 = OCR port operation (Normal = 0)
        //      Bit 1:0 = WGM31:0 determined by OCR3A = TOP

TCCR3B = 0b00001101;
        //      Bit 7:5 = Reserved
        //      Bit 4:3 = WGM33:2 determined by OCR3A = TOP
        //      Bit 2:0 = Clock Select
        //              001 = No Prescaler
        //              010 = clk/8
        //              011 = clk/64
        //              100 = clk/256
        //              101 = clk/1024

TCCR3C = 0;

// Set TOP of Timer/Counter 3 for Interrupt
OCR3A = 1953;
        // Frequency (f) = 16MHz
        // Prescalar (p)
        // Desired Interrupt Interval (t) = 0.25 seconds
        // Cycle @ 1024 = f*t/p = 1953

// Extended Timer/Counter Interrupt Mask
ETIMSK = 0b00010000;
        //      Bit 7:6 = Reserved
        //      Bit 5   = TICIE3
        //      Bit 4   = OCIE3A
        //              When OCIE3A = 1, the Output Compare Interrupt is
Enabled
        //      Bit 3   = OCIE3B
        //      Bit 2   = TOIE3
        //      Bit 1   = OCIE3C
        //      Bit 0   = OCIE1C


    // Enable interrupts in SREG I bit
    sei();
```

```
}
```

led.c

```
//-------------------------------------------------------
// led.c creates functions for the 3 LEDs used
//
// Written by: Jonathon Jeske
// EEL5666 - Intelligent Machine Design Lab
// Spring 2009
//-------------------------------------------------------
//
// PORT E is used for the LEDs
//
//        PIN 0 = Red 1
//        PIN 1 = Red 2
//        PIN 2 = Yellow
//        PIN 3 = Green
//
//-------------------------------------------------------


//-------------------------------------------------------
// Includes
//-------------------------------------------------------

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "LCD.h"
#include "sleep.h"

//-------------------------------------------------------
// Functions
//-------------------------------------------------------

void ledRed() {

   DDRA = 0b10010000;
        PORTA = 0b10010000;

}

void ledRedFlashing(int x) {
```

```
   DDRA = 0b10010000;
        int i;
        for (i=0;i<x;x--){
                PORTA = 0b10010000;
                ms_sleep(100);
                PORTA = 0b00000000;
                ms_sleep(100);
        }

}

void ledYellow() {

   DDRA = 0b00100000;
        PORTA = 0b00100000;

}

void ledYellowFlashing(int x) {

   DDRA = 0b00100000;
        int i;
        for (i=0;i<x;x--){
                PORTA = 0b00100000;
                ms_sleep(100);
                PORTA = 0b00000000;
                ms_sleep(100);
        }

}

void ledGreen() {

   DDRA = 0b01000000;
        PORTA = 0b01000000;

}

void ledGreenFlashing(int x) {

   DDRA = 0b01000000;
        int i;
        for (i=0;i<x;x--){
```

```
                PORTA = 0b01000000;
                ms_sleep(100);
                PORTA = 0b00000000;
                ms_sleep(100);
        }

}

void allLedOn() {

        DDRA = 0b11110000;
        PORTA = 0b11110000;
}

void allLedOff() {

        DDRA &= 0b00000000;
        PORTA &= 0b00000000;

}
```

motors.c

```
//-------------------------------------------------------------
// motors.c - is a list of functions designed to control the
// speed of two individual motors for forward and reverse motion
//
// Written by: Jonathon Jeske
// EEL5666 - Intelligent Machine Design Lab
// Spring 2009
//-------------------------------------------------------------
//
// This code is a modified version of what Joe Bari used on his
// robot.
//
//-------------------------------------------------------------
//
//      These funcitons use port B on the ATmega128
//
//      PIN Setup
//
//      0 = AIN2                SS
//      1 = AIN1                SCK
//
```

```
//        2 = Standby             MOSI
//
//        3 = BIN1                MISO
//        4 = BIN2                OC0
//
//        5 = PWMA                OC1A
//        6 = PWMB                OC1B
//        7 = Nothing             OC1C
//
//        PIN Configuration for Motion
//
//    BIN  AIN
//           21   12
//           43   10
//        000 01 1 01 = forward
//        000 10 1 10 = reverse
//        000 01 1 00 = left forward
//        000 00 1 01 = right forward
//        000 10 1 00 = left reverse
//        000 00 1 10 = right reverse
//        000 11 1 11 = brake
//        000 00 1 00 = coast
//
//------------------------------------------------------------

//------------------------------------------------------------
// Includes
//------------------------------------------------------------

#include <avr/io.h>
#include <avr/interrupt.h>
#include "motors.h"


//------------------------------------------------------------
// Functions
//------------------------------------------------------------


void initMotors(void) {

        // ICR = Input Compare Register
        // TCCR = Timer/Counter Control Register
```

```
ICR1 = 20000;
TCCR1A = 0b10100000;

        //      0b10100000
        //       76543210
        //
        //      7 = COM1A1   = 1
        //      6 = COM1A0   = 0
        //              Clear OC1A/OC1B/OC1C on compare match when up-counting
        //              Set OC1A/OC1B/OC1C on compare match when downcounting
        //      5 = COM1B1   = 1
        //      4 = COM1B0   = 0
        //              Clear OC1A/OC1B/OC1C on compare match when up-counting
        //              Set OC1A/OC1B/OC1C on compare match when downcounting

        //      3 = COM1C1   = 0      = Normal Port Operation
        //      2 = COM1C0   = 0 = Normal Port Operation
        //      1 = WGM11    = 0
        //      0 = WGM10    = 0

TCCR1B = 0x12;

        //      0b00010010
        //       76543210
        //
        //  7 = ICNC1    = 0 = Input Capture Noise Canceler
        //      6 = ICES1        = 0 = Input Capture Edge Select
        //      5 = Reserved
        //      4 = WGM13    = 1
        //      3 = WGM12    = 0
        //
        //              WGM11 = 0
        //              WGM10 = 0
        //
        //              This is mode 8 - PWM, Phase and Frequency Correct
        //              TOP = ICR1 = Maximum Sequence Counter
        //              Update of OCR1X = BOTTOM
        //              TOV1 Flag Set on BOTTOM
        //
        //      2 = CS12 = 0
        //      1 = CS11 = 1
        //      0 = CS10 = 0
        //
        //              clk_I/O / 8 (from prescalar)
```

```
	DDRB = 0b11111111;
}


void driveStraight(int amnt) {

	int maxspeed = 20000;
	int speed;

	if (amnt > 0) {

		PORTB = 0b00001101;
		speed = amnt*(maxspeed/100);
	}

	else {

		PORTB = 0b00010110;
		speed = amnt*(maxspeed/(-100));

	}

	OCR1B = speed;
	OCR1A = speed;
}


void turnRight(int amnt) {

	int maxspeed = 20000;
	int speed;

	if (amnt > 0) {

		PORTB = 0b00001100;
		speed = amnt*(maxspeed/100);
	}

	else {

		PORTB = 0b00010100;
		speed = amnt*(maxspeed/(-100));
	}
```

```
	OCR1A = 0;
	OCR1B = speed;
}


void turnLeft(int amnt) {

	int maxspeed = 20000;
	int speed;

	if (amnt > 0) {

		PORTB = 0b00000101;
		speed = amnt*(maxspeed/100);
	}

	else {

		PORTB = 0b00000110;
		speed = amnt*(maxspeed/(-100));
	}

	OCR1A = speed;
	OCR1B = 0;
}

void brake(void) {

	PORTB = 0b00011111;

	OCR1A = 0;
	OCR1B = 0;
}

void coast(void) {

	PORTB = 0b00000100;
}
```

pump.c

```
//-------------------------------------------------------
// pump.c controls the pump
```

```
//
// Written by: Jonathon Jeske
// EEL5666 - Intelligent Machine Design Lab
// Spring 2009
//-------------------------------------------------------

//-------------------------------------------------------
// Includes
//-------------------------------------------------------

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include "LCD.h"
#include "sleep.h"

//-------------------------------------------------------
// Functions
//-------------------------------------------------------


void pumpOn() {

   DDRF = 0b00000001;
        PORTF = 0b00000001;

}

void pumpOff() {

        DDRF = 0b00000000;
        PORTF = 0b00000000;

}
```

sleep.c

```
// sleep.c taken from class website
// Written by TAs

#include <avr/io.h>
#include <avr/interrupt.h>
#include "sleep.h"
```

```
volatile uint16_t ms_count;


// ms_sleep() - delay for specified number of milliseconds

void ms_sleep(uint16_t ms)
{
 TCNT2  = 0;
 ms_count = 0;
 while (ms_count != ms)
  ;
}

// millisecond counter interrupt vector

SIGNAL(SIG_OUTPUT_COMPARE2)
{
 ms_count++;
}

// initialize timer 0 to generate an interrupt every millisecond.

void initSleep(void)
{
 /*
  * Initialize timer0 to generate an output compare interrupt, and
  * set the output compare register so that we get that interrupt
  * every millisecond.
  */
 TIFR  |= _BV(OCIE2);
 TCCR2  = _BV(WGM01)|_BV(CS02)|_BV(CS00); /* CTC, prescale = 128 */
 TCNT2  = 0;
 TIMSK |= _BV(OCIE2);    /* enable output compare interrupt */
 OCR2   = 12;         /* match in 1 ms */
 sei();

}
```