MARV

MARINE AUTONOMOUS RECOVERY VEHICLE

JOSHUA PHILLIPS

FINAL REPORT

7 DECEMBER 2009

EEL 5666C: INTELLIGENT MACHINE DESIGN LAB

INSTRUCTORS:	TA:
DR. A ANTONIO ARROYO DR. ERIC M SCHWARTZ	THOMAS VERMEER MIKE PRIDGEN

TABLE OF CONTENTS:

ABSTRACT:

MARV (Marine Autonomous Recovery Vehicle) is an amphibious assault and recovery vehicle. He supplies two main functions: (1) Hostile target removal, and (2) Sensitive cargo recovery. When not performing these two functions, MARV will remain in his home base. Upon activations he will leave the base and move toward the designated body of water. After reaching the shore line, he will locate any hostile installations and remove the threat. After clearing the area of hostiles MARV will enter the water. He will then locate the sensitive cargo which will be on the bottom of the lake. After locating the cargo MARV will recover it and return it to home base, where he will remain until activated again.

EXECUTIVE SUMMARY:

MARV was originally designed to perform in and out of the water. After getting about ¾ through the semester it was decided to limit focus to the out of water functions. If time remained underwater functions would be completed. Despite this fact all functions, devices, and elements added to the robot were designed and/or constructed with underwater capabilities in mind.

An RC tank was purchased and hollowed out to house the electronics and servomotors for continuous drive capability. The tank was designed to be amphibious and is approximately watertight unmodified. There are some leaks but they can be easily patched.

The robot contains a central controller: an ATxmega128 board constructed by PVR robotics. The board controls the PWM output signals for the motors/servos used for land and underwater motion, in addition to the gripper arm. In addition the board is used for RS232 serial communication with the CMUcam1 used for tracking and targeting functions. In addition it would also be used for general I/O function. The I/O ports control the underwater propellers in addition to several shutdown functions and the gripper arm pushbutton control.

Original specifications called for the use of continuous track driven by the original motors that came with the chassis. With two weeks remaining in the project it was discovered that the extra weight from all the intended devices was too much for the original motors driving the continuous track. If the underwater capabilities had been left off the design this would not have constituted a problem. Due to the current situation a new motor construct had to be devised. This was done using ultra high torque servomotors. The ultimate failure of this system precluded disappointing results for MARV.

The underwater propellers were originally going to be modified servos, but it was discovered that the RPM of these motors were far too low to drive MARV through the water. Rule 360GPH bilge pumps were purchased to solve this problem. They were modified and propellers were attached and they work marvelously in addition to being waterproof.

The main functions of MARV are driven by a set of sensors. These sensors include IR sensors for line following. This is used during the first stage where MARV makes his way to the water's edge. The other main sensor used is the CMUcam1 for color sensing. This system is used in order to track in the water in addition to locating the on-land target.

When the intended cargo is found there is a gripper arm added in order to pick up the object. In order to find the object the CMUcam1 mentioned above is used to locate it and a push button is located between the gripper arm to identify when the target is acquired.

The on-land functions all work, in addition to the underwater functions. The only problems arise in a broken output shaft for the drive motor late in the process. Due to lack of time necessary to work on the underwater function this was abandoned and saved for a later date.

INTRODUCTION:

One objective of warfare is the completion of tasks with minimal loss of friendly and civilian lives. One excellent way of doing this is by having intelligent machines perform tasks such as hostile removal, reconnaissance, and cargo removal. Currently this sort of function is performed by military departments such as the United States Naval Explosive Ordinance Disposal unit.

The purpose of MARV is to provide a functional way of locating and recovering underwater cargo for return to friendly base. It is assumed MARV will encounter hostiles along the way and in order to complete his task he will have to remove the hostile installations.

MARV's platform is constructed of an RC tank platform. The tank was gutted and servomotors were added to replace the insufficiently powered original motors. He will be equipped with propellers for underwater propulsion and continuous track for above ground motion.

Underwater propulsion units will be powered by modified bilge pump motors. These motors will be arranged in order to provide MARV will full 3-D motion while underwater. The continuous track is powered by continuous motion servos.

Sensors are a tricky thing on underwater vehicles. Many things that work above ground work differently or not at all when submerged in water. The main sensor used on MARV will be an CMUcam1 for color detection. The color detection is used for navigation and targeting activities. In addition IR sensors are used for line following to get MARV from the starting point to the water's edge.

INTEGRATED SYSTEMS:

At the heart of the system is a PVR board, designed and constructed by PVR. This board contains an ATXMEGA128. This board provides all the necessary controller functions for the board including: servo control, UART for the webcam, ADC for the sensors, and I/O functions such as for the artillery. The following is a system level block diagram:

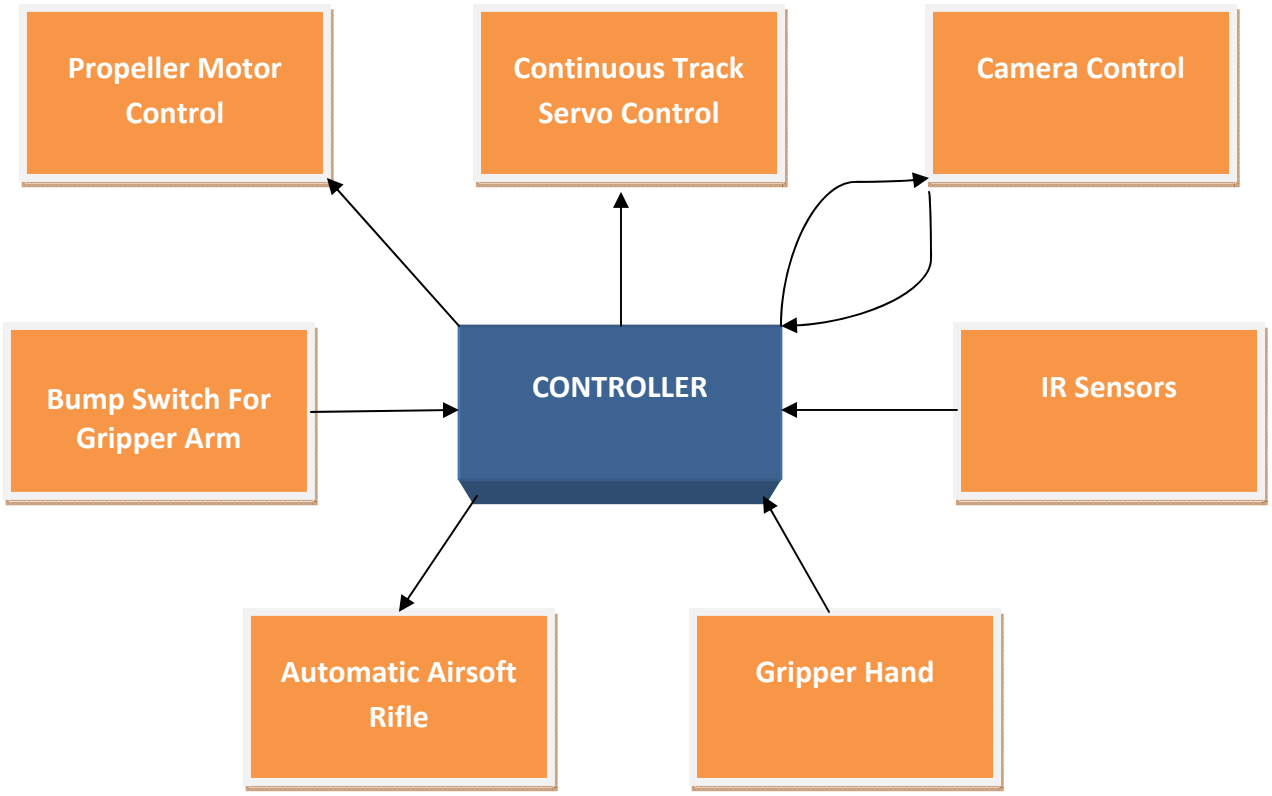


Figure 1: System Block Diagram

One nice feature of this set up is that all the features are directly controlled by hardware. All data is retrieved from the devices and interpreted by the XMEGA controller and instructions are sent out to the peripherals. This arrangement allows for redundancy and maintains system stability in case of contradictory sensory output.

MOBILE PLATFORM:

MARV's platform was constructed from an RC tank purchased online from XenonProject.com. This tank was chosen because it was light weight, small, and was an amphibious tank and already contained the necessary property of being watertight. A picture of the tank prior to modification can be seen to the right in figure 2.

Figure 2: RC Panzer Amphibious tank

The top part of the robot containing the fake artillery and the camouflage is removed and the inside of the housing is gutted, removing any excess plastic to make the mounting of electronics inside the robot easier. Some pictures of the robot are shown below in figures 3 and 4:

Figure 3: MARV main chassis exterior

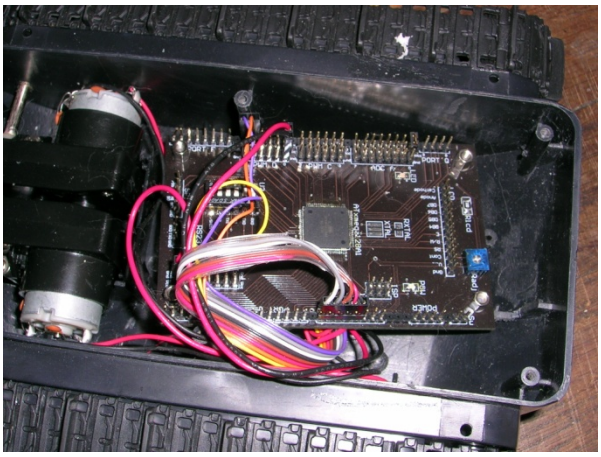


Figure 4: MARV interior of main chassis

Above this main chassis two electrical conduit boxes were added to hold the motor drive transistors for the propeller system in addition to the main battery line terminals, to supply battery voltage to any system that requires it, rather than the 5V or 3.3V regulated lines from the PVR board. The conduit boxes can be seen in figure 5 to the right. Above these conduit boxes is a platform constructed out of 8x12 piece of Plexiglass. The platform is very flimsy so two aluminum rods are added for support underneath the platform. This construct can be seen in later figures.

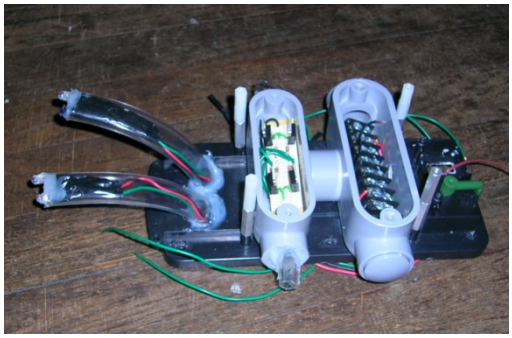


Figure 5: Conduit boxes above main chassis

This chassis is the 2nd one constructed. The first one was scrapped due to balancing issues. The front of MARV was far too heavy. The battery has

significant weight and was shifted far to the front while the camera was mounted in the middle causing the whole robot to shift forward. Eventually this idea was scrapped in favor of the current design. This design is not without it's flaws though. Rather than being front heavy this new design became too back heavy. To account for this a makeshift caster was constructed out of a flexible piece of copper and acts somewhat like a sled blade. The flexibility of the caster is nice as it maintains proper traction from the tank treads.

In addition to the weight issues the platform has other problems:

- Plexiglass for instance works really well, but if done again more rigid Plexiglass should be used for better structuring.
- As can be seen in figure 5 the platform level is held up by nylon spacers. These don't work very well and strip easily with some pressure applied to them. Metal spacers should have been used for this, but none were available at the time and never ordered.
- As seen in figure 4 there is not a lot of room for extra electronics and wiring inside the chassis, if done again it would be wise to choose a main chassis with more room for electronics. A larger chassis would also likely solve a lot of the balance issues.

In order to account for these shortcomings several things were done. For better structure aluminum rods were added along the length of the robot to make the platform more rigid. To make up for not having enough room for electronics expansion space was made in the way of conduit boxes pictured in figure 5. The nylon spacers never posed enough of a problem to worry about, though they were glued together late in the process just in case.

ACTUATION:

As stated in the introduction, MARV has both under water and dry land behaviors. As such an actuation method for both locations is necessary. A continuous track method is used above ground and a propeller propulsion method is used underwater.

Continuous track works by keeping as large of a flat surface continually on the ground as possible. An example of a configuration used is located to the left in Figure 1 For the purpose of MARV it was decided that the continuous track would make it easier to get out of the water due to the higher amount of surface area on the ramp. The continuous track motion will be driven by two independent hacked Hi-Tec HS 645MG servos. Originally the motors and gears that came with the chassis were going to be used but they didn't provide sufficient torque to move the tank so they were replaced with the aforementioned servos which provide 133 oz-in torque at 6V. The hack for these motors was particularly hard due to their metal gears. The pin that limits the rotation was particularly difficult to remove and had to be grinded off with a Dremel paying special care not to damage the gear.

Figure 6: M60 Patton, example of continuous track

Figure 7: HS-645 High Torque Servo

Underwater motion will be controlled by propellers. They will be driven by 360 GPH bilge pumps hacked for the motor. The pump can be seen in figure 8. To hack the pump the encasings below the red part at the top were removed exposing the motor output shaft and the impeller. The impeller was removed and a propeller was added. If done properly the motor is still waterproof.

Figure 8: Rule 360 GPH Bilge Pump

This motor runs most efficiently at 12 V but runs as low as 4V. In MARV the pump will be run at 8V. At this voltage the continuous current draw is 1.1A per pump with spikes in the 2.5A range during start and stop. These motors will be driven by TIP120 Darlington Pair transistors rated at 5A purchased from RadioShack for convenience. The circuit board for the transistors is located in the conduit boxes pictured in figure 5.

The gripper hand is powered by a servomotor as well. The servo in this case is the Traxxas 2065 waterproof servo. Two servos were used to construct the gripper hand.

The airsoft rifle is also powered by a DC motor which similarly to the bilge pumps are controlled by a darlington pair switching transistor circuit.

Like the mobile platform portion of the robot, the actuation devices presented considerable difficulty and were likely the most expensive devices, as a result the most costly mistakes. The following difficulties were encountered with the actuation system:

- Not enough torque for the tank tread motors originally used
- Motors in the water must rotate very quickly and servomotors only have around 50RPM, while RPM needs to be more like 500 in the water for proper propulsion to take place

The torque issue was not recognized until very late in the process. The motors lasted through the addition of all the items except the camera housing. When the camera housing was mounted it was noticed that one of the tank motors didn't have enough torque to pull the tank anymore while the other one barely did. This was after switching the motors off the 5V PVR board supply to the 8V battery supply. It still didn't make enough of a difference. These motors were then stripped out of the chassis and high torque servomotors purchased at HobbytownUSA. These servos have plenty of torque to push the tank as mentioned above with well over 130oz-in of torque.

Unfortunately due to the late acquisition of these motors, proper output shaft attachments were not acquirable and makeshift shafts were constructed. The rightward tank motor attachments works, while the leftward one worked temporarily before breaking. If given more time to work on this it would be an easy fix and significantly improve the performance of MARV.

The issue of RPM for the underwater motors was an easy fix, though very costly as the original servos cost about $120 and the new motors cost about $75, resulting in a cost of nearly $200 for this system. The motors used were from Rule 360GPH bilge pumps mentioned previously in this section.

One other noteworthy issue is that the additional weight of these motors over the servos originally set to be used is likely what caused the unexpected heavy weight of the robot leading to the inability of the main motors to perform their function.

SENSORS:

Three kinds of sensors were used in this project: (1) Infrared Proximity Sensor (2) Push Button Bump Switch (3) CMUcam1 for color tracking.

The Infrared Proximity Sensor is a simple voltage divider circuit. The circuit is arranged as in figure 9 below.

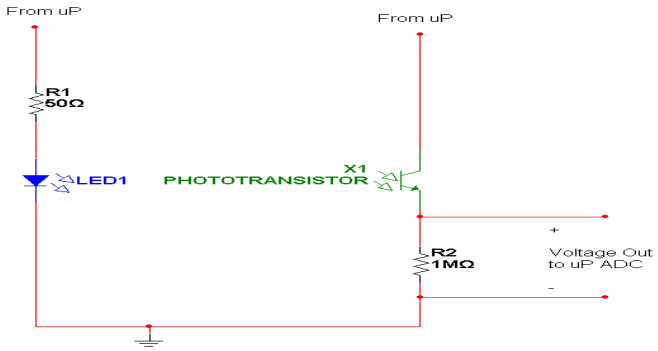


Figure 9: IR Emitter/Receiver Pair Connections

Though Sharp IR sensors are more accurate they are also more expensive and take up more space. The housing for these sensors needs to be watertight as it will be outside the chassis. As such a housing was constructed out of clear rubber tubing sealed to the chassis with epoxy and the IR emitter/receiver are sealed with epoxy where they exit the tube, to keep from putting the water's resistance in parallel across the LED/phototransistor terminals. In actuality this would likely not damage the robot, but just to be safe this precaution was taken. The full arrangement of the IR sensor can be seen in figure 10 and 11 below. Figure 11 shows the mount structure that the IR housing is placed in to direct them at the ground for line following.

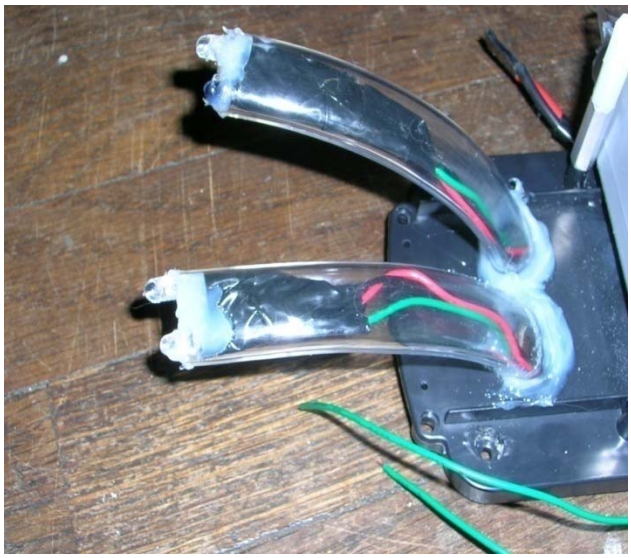


Figure 10: IR Housing

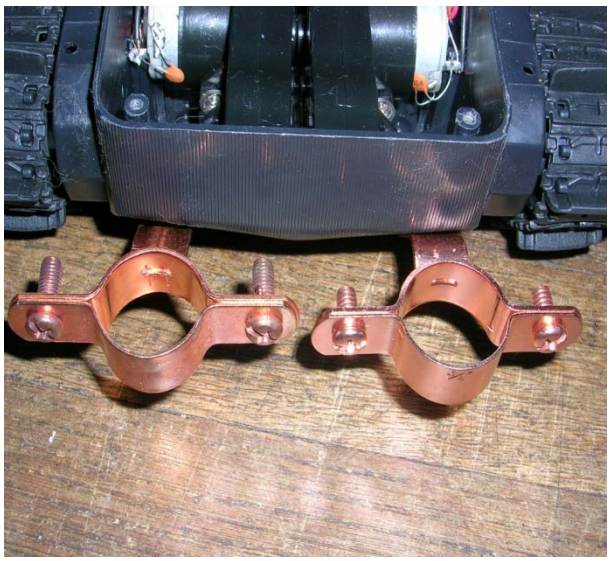


Figure 11: IR Mounts

The next system which there currently is no picture for is the pus button switch used for the gripper hand. It is used to indicate the presence of the cargo to be recovered to notify the gripper hand when to close in order to capture the cargo. It is a simple SPDT switch, with the normally closed input connected to ground, the normally high input connected to +3.3V, and the common output connected to an input port on the microcontroller through a 2.5kΩ resistor.

The final system and the most important system on MARV is the CMUcam1. The CMUcam is a CMOS IR camera designed by Carnegie Mellon University and distributed via Seattle Robotics. It comes fully constructed and ready to go out of the box. It can be interfaced with either a microcontroller via level shifted or TTL serial communication. In addition for purposes of debugging and testing it comes with the ability to interface with standard terminal programs such as Hyperterminal in addition to its own JAVA GUI. The JAVA GUI is buggy but is very useful in learning how the CMUcam1 works.

The CMUcam1 has the following properties:

- Tracks user defined color blobs at 17fps
- Finds centroid data, mean colors, and deviation
- Can be used to track RGB data or YCrCB
- 80 x 143 resolution
- Ability to control a pan servo directly for tracking colors
- 115200, 38400, 19200 and 9600 baud serial communication rates via hardware jumper

Below in figure 12 is a Diagram of the CMUCam1 control board

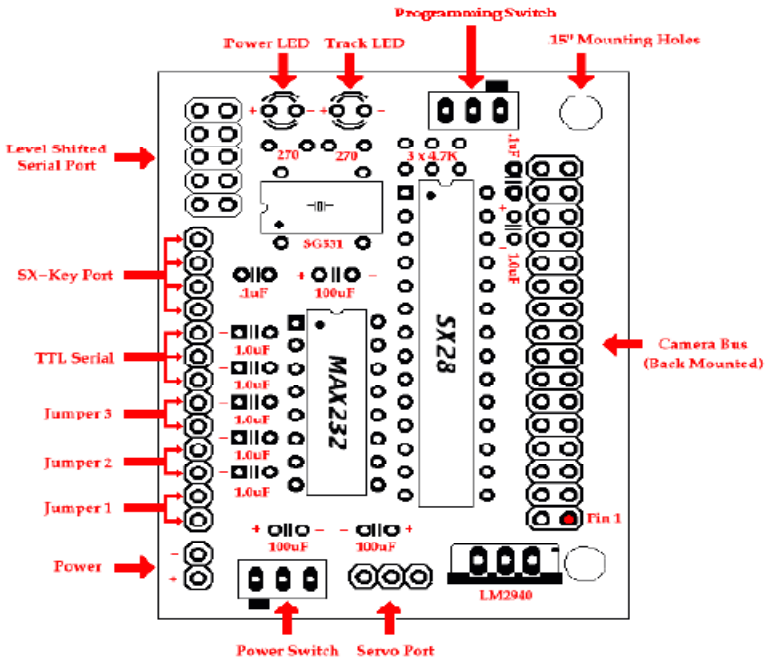


Figure 12: CMUcam1 control board schematic

By default the board is set up to run at 115kbps. A lot of errors occur at this speed due to the incompatibility of the XMEGA's USART system to run at a frequency within 2kbps of the required frequency, so it was reduced to 9.6kbps to positive results.

Color tracking is the main function of this device. It was found that depending on the environment that the colors vary wildly due to the nature of the camera to shift to red in heavy light, especially outside. Due to this the color range had to be selected overly wide, and the auto white balance and auto gain had to be adjusted. In this case the white balance was turned off and the auto gain was allowed to set on initialization before being turned off to prevent problems. All UART and CMUcam1 code is listed in the appendix.

Pictures of the CMUcam1 setup are shown in the figure13(a-d) below.

Figure 13: (a) top left is the airsoft gun with the camera attached below it (b) top right is the camera attached below the gun from the front (c) bottom left is the servo mount for the camera housing (d) bottom right is the camera housing itself fully sealed and ready to go.

As can be seen in the pictures above the CMUcam1 is directly attached to the airsoft rifle to make the targeting system more accurate as the camera will rotate and point directly at the same spot as the airsoft rifle. This rotation results from the servo in figure 13(c) which is directly controlled by the camera's controller board.

BEHAVIORS:

MARV will have two main phases. The first is above ground and involves leaving the base, moving down a path, stopping at the waters edge and removing hostile installations using an airsoft gun or other artillery. The second behavior involves entering the water, locating cargo, capturing it in some manner, and returning it to the home base (out of the water).

The ground phase dubbed "Hostile Removal" involves the need for MARV to be activated by a friendly troop in some manner. This will be accomplished using the bump switch also used by the gripper arm to indicate presence of cargo in the gripper hand. After leaving the base MARV is charged with the duty of following a predefined line on top of a bridge structure. In addition to following the line, MARV will require some sort of edge detection to make sure he doesn't fall off the bridge on his way to the lake, though this will not be implemented at this time. Upon arriving at the lake MARV will make about 90 degree scan of his surroundings for hostiles and take them out one by one. The hostiles will be designated a specific color, in this case red. By default, MARV will continue firing until all installations can be verifiably destroyed. After completion of this task the hostile removal phase is complete and MARV will continue down the ramp and into the water.

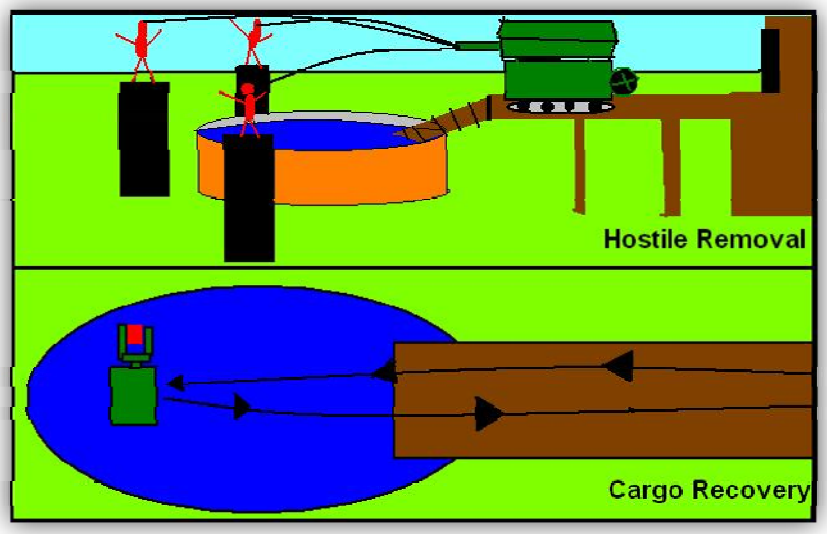
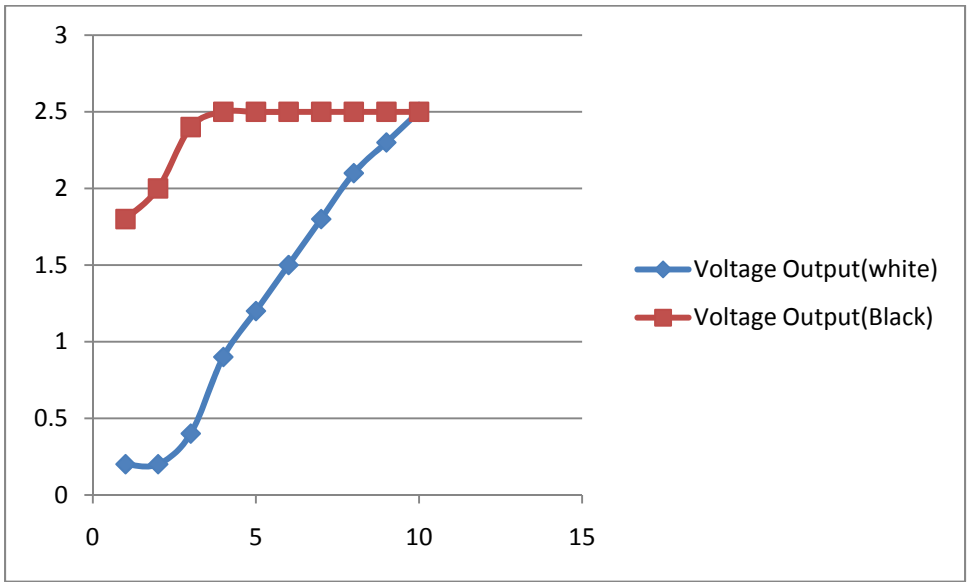


Figure 5: Proposed course and behaviors

The cargo recovery phase begins directly after the hostile removal phase. It will start with the behavior of MARV to detect that he is in the water and turn off the land actuation devices and use strictly his water propulsion systems. Due to the nature of his buoyancy structure, MARV simply needs to wait till he arrives at the bottom, without the necessity of propulsion to get to the bottom. Once at the bottom he will proceed to do another scan for the cargo which will be designated by a different color from the path followed as well as the hostile targets. Using the color target for navigation direction MARV will proceed to move toward the cargo. MARV will use a robotic claw to grasp the object. He will proceed to turn around and locate the ramp, still designated by the same color used earlier. Upon arrival at the ramp, MARV will continue to run his underwater propulsion system until completely out of the water. During this time the dry land propulsion system will start running to assist in moving out of the water, and continue after the underwater propulsion cuts off. MARV will then follow the line back to his home base.

EXPERIMENTAL LAYOUT AND RESULTS:

IR Sensor Data:



The sensor data above was for the IR used for the line following behavior between MARV's outpost and the edge of the water. As can be seen the output voltage for a black line would be far different from the output voltage from a white patch. Therefore it is possible to discern between these two high contrast colors using IR. In this case the sensors are arranged about 2 inches apart and are used to follow a one inch wide line. The sensors are used to detect the presence of the white space. If the white space is detected the motor on that side stays on, if it is not present (or alternatively the black line is present) that motor turns off allowing MARV to turn and correct himself. This particular function works flawlessly. If the motor output shaft had not broken there is no doubt that this particular function of MARV would work without any problems.

CONCLUSIONS:

The project accomplished many of the goals set out at the beginning such as:

- Line following procedure worked very well
- Underwater propulsion systems work despite inability to put robot in water
- Targeting and overall gun system worked well, except it wasn't entirely accurately, mostly due to the deficiencies of the CMUcam1 itself
- Propeller control tracking system using the CMUcam1 worked well
- Gripper hand activates and holds well enough to carry an object back

On the other hand there were some disappointments such as:

- Inability to get to a point where MARV could be placed in the water all the way and his underwater functions tested there
- Insufficient torque in motors resulting in new motor system too late to purchase proper mounts and output components
- Balancing issues due to lack of proper mechanical training
- Inability to have time to implement the original idea of starting MARV with voice activation

Despite the flaws I consider MARV to be a success as a work in progress. If given a bit more time I believe all of MARV's flaws could be reasonably worked out. That being said, MARV is not perfect and there are many things I would change if I could do things over such as:

- Either construct my own chassis using easily obtainable tank tread parts and USE PROPER CONNECTORS AND COMPONENTS or use a much larger base
- Select proper DC motors and gear box for the heavy load of MARV (~ 8-10 lbs)
- Construct everything out of more rigid materials (i.e. no plexiglass/plastic)
- Attach everything with screws, no glue
- Use only servos with easy to find servo horns, parts, and components such as Futaba or Hi-Tec rather than harder to find brands such as Traxxas which make servos for specific functions and specific vehicles
- Construct my own controller board devised specifically for what I need
- Getting PCBs made on campus was very difficult for this class, and not of terribly good quality, anything that needs to be milled should be shipped off even if it costs
- Don't cut corners get quality parts, in the end cheap stuff might end up costing more

It goes without saying that this project was overly ambitious. On the other hand, isn't that what college is for, testing your limits and seeing what you can do. I think anyone can slap on a few IR and SONAR to follow a line or avoid a wall given enough time. Designing a complex system like I attempted to do this semester pushed me to the limit, and even though I didn't accomplish all I hoped to, I think I'm a better engineer for trying and I encourage other people to do the same.

DOCUMENTATION:

[1] "Actuators: How to Modify a Servo," *societyofrobots.com*, 2005. [Online]. Available: http://www.societyofrobots.com/actuators_modifyservo.shtml. [Accessed: Sept. 12, 2009].

[2] “The CMUCam1 Vision Sensor,” 2001. [Online]. Available: http://www.cs.cmu.edu/~cmucam/home.html. [Accessed: Sept 28, 2009].

[3] "Instrucutables: Underwater ROV," *insructables.com*, 2009. [Online]. Available: http://www.instructables.com/id/Underwater_ROV/. [Accessed: Sept. 8, 2009].

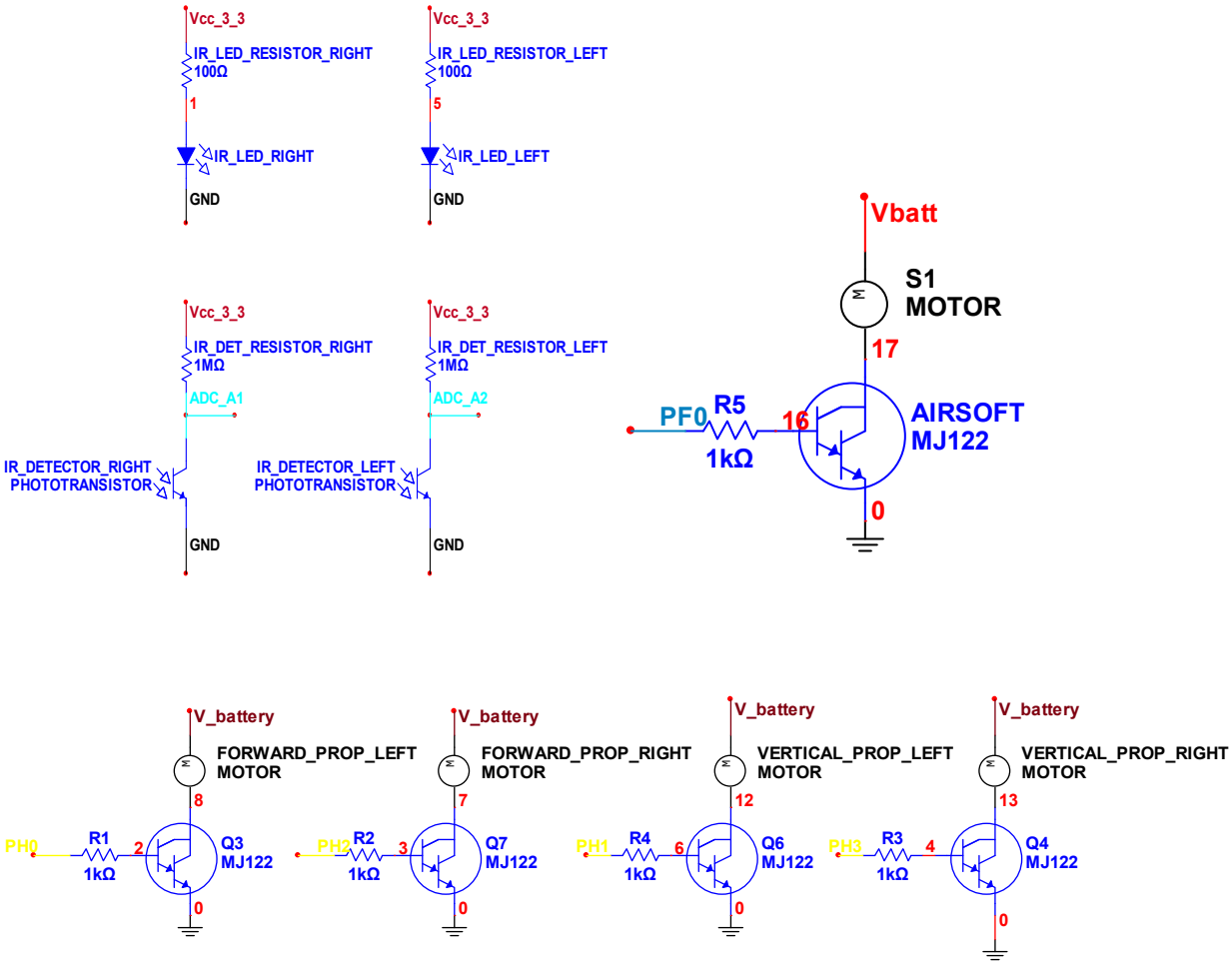
[4] "Homebuilt ROV’s," 2009. [Online]. Available: http://www.homebuiltrovs.com. [Accessed: Sept. 8, 2009].

[5] "Robot Airsoft," 2009. [Online]. Available: http://robotairsoft.com. [Accessed: Sept. 20, 2009].

APPENDICES:

Appendices include all schematics and code used in the design of MARV

APPENDIX A: SCHEMATICS



Appendix B: Motor Function Code

```
/*****************
 * Motor Library *
 *****************/

/*************************************************************
 * Code written by Joshua Phillips for DC Motor operation    *
 * during Fall 2009 session at the University of Florida for *
 * Intelligent Machine Design Lab.  The code is written      *
 * specifically for Atmel's XMEGA128,                        *
 * ----------------------------------------------------------*
 * Included are the following functions:                     *
 *              tankInit() - Initialize PWM ports for tank motors *
 *              tankL(value) - left tread forward at speed 0-8 *
 *              tankR(value) - right tread forward at speed 0-8 *
 *              tankStop() - stops all tank tread motion     *
 *                                                           *
 *              PropInit() - Initializes Propeller Functions *
 *              Prop(L,R)_ForwardOn() - Turns on L or R forward prop *
 *              Prop(L,R)_ForwardOff() - Turns off L or R forward prop *
 *              Prop(L,R)_UpwardOn() - Turns on L or R upward prop *
 *              Prop(L,R)_UpwardOff() - Turns off L or R upward prop *
 *              PropForwardOn() - Turns both forward motion props on *
 *              PropForwardOff() - Turns both forward motion props off *
 *              PropUpOn() - Turns both rise props on        *
 *              PropUpOff() - Turns both rise props off      *
 *              GunInit() - Initializes Airsoft Port         *
 *              GunFire() - Fires gun for 500ms              *
*                                                           *
 *************************************************************/

/**************
 *  INCLUDES  *
 **************/
#include <avr/io.h>
#include "Motor.h"
#include "xmega.h"

/**************
 *  DEFINES  *
 **************/
 #define PropControl            PORTH_OUT
 #define PropControlDir                 PORTH_DIR
```

```
#define PropLeftFwdOn           0x01
#define PropLeftUpOn            0x02
#define PropRightFwdOn          0x04
#define PropRightUpOn           0x08
#define PropLeftFwdOff          0xFE
#define PropLeftUpOff           0xFD
#define PropRightFwdOff         0xFB
#define PropRightUpOff          0xF7
#define GunControl              PORTF_OUT
#define GunControlDir           PORTF_DIR
#define GunOn                       1
#define GunOff                      0

/*********
 * TANK  *
 *********/

void tankInit(void)             //Initialize motor set up motor with PWM (Servo lines D4
& D5)
{
        TCD0_CTRLA = 0x05;      //set TCC0_CLK to CLK/64
        TCD0_CTRLB = 0xF3;      //Enable OC A, B, C, and D.  Set to Single Slope PWM
                                //OCnX = 1 from Bottom to CCx and 0 from CCx to Top
        TCD0_PER = 10000;       //20ms / (1/(32MHz/64)) = 10000.  PER = Top
        TCD1_CTRLA = 0x05;      //set TCC1_CLK to CLK/64
        TCD1_CTRLB = 0x33;      //Enable OC A and B.  Set to Single Slope PWM
                                //OCnX = 1 from Bottom to CCx and 0 from CCx to Top
        TCD1_PER = 10000;       //20ms / (1/(32MHz/64)) = 10000.  PER = Top
        PORTD_DIR = 0x3F;       //set PORTC5:0 to output
        TCD0_CCA = 0;           //PWMD0 off
        TCD0_CCB = 0;           //PWMD1 off
        TCD0_CCC = 638;         //PWMD2 off
        TCD0_CCD = 813;         //PWMD3 off
        TCD1_CCA = 0;           //PWMD4 off
        TCD1_CCB = 0;           //PWMD5 off
}

void tankR(int value)           //define left motor speed range
{
        if (value > 100)        //cap at +/- 100
                value = 100;    //  -100 => 1ms
        else if (value < -100)  //  0    => 1.5ms
                value = -100;   //  100  => 2ms
```

```
        value *= -5;                    //multiply value by 2.5
        value /= 2;                     //  new range +/- 250
        TCD1_CCA = (750 + value);       //Generate PWM.
}

void tankL(int value)
{
        if (value > 100)                //cap at +/- 100
                value = 100;            //  -100 => 1ms
        else if (value < -100)          //  0    => 1.5ms
                value = -100;           //  100  => 2ms
        value *= 5;                     //multiply value by 2.5
        value /= 2;                     //  new range +/- 250
        TCD1_CCB = (750 + value);       //Generate PWM.
}

void tankStop(void)
{
        TCD1_CCA = 0;
        TCD1_CCB = 0;
}

/*************
 * PROPELLER *
 *************/
void PropInit(void)
{
        PropControlDir = 0x0F;
        PropControl = 0x00;
}

void PropL_ForwardOn(void)
{
        PropControl |= PropLeftFwdOn;
}

void PropL_ForwardOff(void)
{
        PropControl &= PropLeftFwdOff;
}

void PropR_ForwardOn(void)
{
        PropControl |= PropRightFwdOn;
```

```
}

void PropR_ForwardOff(void)
{
	PropControl &= PropRightFwdOff;
}

void PropL_UpwardOn(void)
{
	PropControl |= PropLeftUpOn;
}

void PropL_UpwardOff(void)
{
	PropControl &= PropLeftUpOff;
}

void PropR_UpwardOn(void)
{
	PropControl |= PropRightUpOn;
}

void PropR_UpwardOff(void)
{
	PropControl &= PropRightUpOff;
}

void PropForwardOn(void)
{
	PropR_ForwardOn();
	PropL_ForwardOn();
}

void PropForwardOff(void)
{
	PropR_ForwardOff();
	PropL_ForwardOff();
}

void PropUpOn(void)
{
	PropR_UpwardOn();
	PropL_UpwardOn();
}
```

```
void PropUpOff(void)
{
	PropR_UpwardOff();
	PropL_UpwardOff();
}

/***************
 * Airsoft Gun *
 ***************/

void GunInit(void)
{
	GunControlDir = 0x01;
	GunControl = 0;
}

void GunFire(void)
{
	GunControl = GunOn;
	delay_ms(1000);
	GunControl = GunOff;
}

/*************
 *	CLAW   *
 *************/
void ClawR(int angle)
{
	if (angle > 94)				//cap at +/- 100
		angle = 94;				//  -100 => 1ms
	else if (angle < -45)			//  0    => 1.5ms
		angle = -45;				//  100  => 2ms
	angle *= 5;					//multiply value by 2.5
	angle /= 2;					//  new range +/- 250
	TCD0_CCC = (750 + angle);		//Generate PWM.
}

void ClawL(int angle)
{
	if (angle > 25)				//cap at +/- 100
		angle = 25;				//  -100 => 1ms
	else if (angle < -94)			//  0    => 1.5ms
		angle = -94;				//  100  => 2ms
```

```
        angle *= 5;                                     //multiply value by 2.5
        angle /= 2;                                     //  new range +/- 250
        TCD0_CCD = (750 + angle);               //Generate PWM.
}

void ClawClose(void)
{
        ClawL(-100);
        ClawR(100);
}

void ClawOpen(void)
{
        ClawL(100);
        ClawR(-100);
}
```

APPENDIX C:

```
/***************
 * UART Library *
 ***************/

/*****************************************************
 * Code written by Joshua Phillips for CMUcam operation          *
 * during Fall 2009 session at the University of Florida         *
 * for Intelligent Machine Design Lab.  The code is              *
 * written specifically for Atmel's XMEGA128,                    *
 * ----------------------------------------------                *
 * Included are the following functions:                         *
 *                uart_init() - Inititializes USART for 8N1 with no      *
 *                        parity at a frequency of 115.2 kbps.           *
 *                uart_getchar() - returns an unsigned character         *
 *                uart_sendchar() - sends an unsigned character          *
 *                uart_getsring() - returns a character array            *
 *                uart_sendstring() - sends a character array            *
 *****************************************************/

/**************
 *  INCLUDES  *
 **************/
#include <avr/io.h>
#include "uart.h"

/***************
 *  FUNCTIONS  *
 ***************/
void uart_init(void)
{
        PORTE_DIR   = PIN3_bm;                          //set TX pin as output
        PORTE_OUT   = PIN3_bm;

        USARTE0_CTRLC = 0x03;                           // set 8N1 asynchronous serial tx/rx
        USARTE0_BAUDCTRLA = 0xCF;                       // set to BSEL = 0xCF, BSCALE = 0x00
        USARTE0_BAUDCTRLB = 0x00;                       // fbaud = 9615.4 bps
        USARTE0_CTRLB |= 0x08;                          // turn on TX system
        USARTE0_CTRLB |= 0x10;                          // turn on RX system
}
```

```
unsigned char uart_getchar(void)
{
	static char rx_char;				// initialize received character buffer
	while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp)));	// wait for RXCIF to be set
	rx_char = USARTE0_DATA;				// data register in variable rx_char
	return rx_char;					// return value from data register
}

void uart_sendchar(unsigned char tx_char)
{
	while (!(USARTE0_STATUS & (1<<USART_DREIF_bp))); // check if data register is empty
	USARTE0_DATA = tx_char;					// store data in tx_char
	while (!(USARTE0_STATUS & (1<<USART_TXCIF_bp))); // wait for RXCIF to be set
}

void uart_sendstring(unsigned char *tx_string)
{
	int CR = 0;				// initialize CR check value to zero
	unsigned char *string = tx_string;	// store tx_string into pointer variable string
	unsigned char tx_char;

	tx_char = *string++;		// tx_char saves character pointed to by string
					// and increments the string pointer
	while (!CR)			// while not line return  continue sending serial string
	{
		uart_sendchar(tx_char); // while tx_char isn't carriage return send tx_char to
		tx_char = *string++;		// serial line then increment string pointer

		if(tx_char=='\r')			// if tx_char is carriage return exit function
		{
			CR=1;
			uart_sendchar(tx_char);
		}
	}
}

unsigned char uart_getstring(void)
{
	static char *rx_string;		//define pointer to array rx_string of unknown length
	int i = 0;
	int CR = 0;

	while(0)
	{
```

```
                rx_string[i] = uart_getchar();  //continue receiving string from peripheral till

                if ((rx_string[0] != ':'))          //check for colon and ignore it
                {
                        if(rx_string[i] =='\r')    //wait for carriage return signifying end of string
                        {
                             rx_string[i] = '\0'; //change end of string character to null character
                            return rx_string;                   //return the string value
                        }
                        i++;                                    //increment i for while loop
                }
        }
        return (rx_string);
}

unsigned int uart_getbyte(void)
{
        unsigned int rx_byte;                           // initialize received integer buffer
        while (!(USARTE0_STATUS & (1<<USART_RXCIF_bp)));        // wait for RXCIF to be set
        rx_byte = USARTE0_DATA;                                 // data register in variable rx_byte
        return rx_byte;                                 // return value from data register
}
```

APPENDIX D: CMUCAM1 LIBRARY

```
/******************
 * CMUcam Library *
 ******************/

/*****************************************************************
 * Code written by Joshua Phillips for CMUcam operation          *
 * during Fall 2009 session at the University of Florida for     *
 * Intelligent Machine Design Lab.  The code is written          *
 * specifically for Atmel's XMEGA128,                            *
 * ------------------------------------------------------        *
 * Included are the following functions:                         *
 *          CMUcamInit() - Initialize CMUcam Registers.  Set delay.  *
 *                              Toggle the light for debug. Start poll  *
 *          mode.  Start Raw mode and suppress ACK/NCK           *
 *          CMUcamServo(char *value) - takes three byte char and  *
 *                                     sends to CMUcam for servo  *
 *                pan start value.                               *
 *          CMUcamTC_active_color() -  Starts active color tracking  *
 *                                                 of color specified.  *
 *          CMUcamTC_passive_color() - Starts passive color tracking.  *
 *                                                 of color specified.  *
 *****************************************************************/

/**************
 *  INCLUDES  *
 **************/
#include <avr/io.h>
#include "uart.h"
#include "cmucam.h"
#include "xmega.h"
#include "lcd.h"

/*************
 *  DEFINES  *
 *************/
/** COMMAND STRINGS **/

 #define RESET                  "RS\r"
 #define POLLMODE_ON            "PM 1\r"
 #define POLLMODE_OFF           "PM 0\r"
 #define RAW_MODE               "RM 3\r"
 #define AUTOGAIN_OFF           "CR 19 32\r"
```

```
#define MIDMASS_ON_N            "MM 10\r"
#define MIDMASS_ON_M            "MM 2\r"
#define MIDMASS_NOSERVO         "MM 1\r"
#define NOISEFILTER_OFF         "NF 0\r"
#define NOISEFILTER_ON          "NF 1\r"

/** DEBUG/TRACKING LIGHT **/

#define LIGHT_OFF           "L1 0\r"
#define LIGHT_ON            "L1 1\r"
#define LIGHT_TRACK         "L1 2\r"

/**     COLOR TRACKING STRINGS **/
#define TRACK_RED           "TC 140 255 0 80 0 80\r"
#define TRACK_GREEN         "TC 70 140 70 110 50 90\r"
#define TRACK_BLUE          "TC 50 98 40 80 60 100\r"

/** LCD COMMANDS **/

#define CLR                         0x01



/***************
 *  FUNCTIONS  *
 ***************/
void CMUcamInit(void)
{
        int i;

        uart_sendstring(RESET)      ;           //reset camera
        lcdData(CLR);                           //send LCD feedback
        lcdString("Resetting the Camera");
        lcdGoto(1,0);

        for(i=0;i<5;i++)                    //toggling the light to make sure serial working
        {
                lcdChar('.');
                uart_sendstring(LIGHT_ON);
                delay_ms(500);
                lcdChar('.');
                uart_sendstring(LIGHT_OFF);
                delay_ms(500);
        }
```

```
        uart_sendstring(NOISEFILTER_ON); //Turn off noise filtering and AUTOGAIN
        delay_ms(50);
        uart_sendstring(AUTOGAIN_OFF);
        delay_ms(50);

        uart_sendstring(LIGHT_TRACK);       //put light back in normal tracking mode
        delay_ms(50);

        uart_sendstring(POLLMODE_OFF);   //put camera into poll mode
        delay_ms(50);

        uart_sendstring(RAW_MODE);              //put camera into raw transfer mode
        delay_ms(50);                           //and suppresses ACK/NCK responses

        uart_sendstring(MIDMASS_ON_N);  //put in middle mass mode and return N-Packet
        delay_ms(50);                           //also in auto-track servo mode


        lcdData(CLR);
        lcdString("CMUcam Init Complete");
        delay_ms(2000);
        lcdData(CLR);

}

void CMUcamTC_active_red(void)
{
        uart_sendstring(MIDMASS_ON_N);
        delay_ms(20);
        uart_sendstring(TRACK_RED);
}

void CMUcamTC_passive_red(void)
{
        uart_sendstring(MIDMASS_NOSERVO);
        delay_ms(200);
        uart_sendstring(TRACK_RED);
}

int GetServoPos(void)
{
        int i=0;                                // redundant loop variable initialization
        int BUFFER = 0;
```

```
	unsigned int CAMERA_NDATA[12];

	while(BUFFER!=255)
	{
		BUFFER=uart_getbyte();
	}

	while(i<10)
	{
		CAMERA_NDATA[i]=uart_getbyte();
		i++;
	}

	return CAMERA_NDATA[1];
}

int GetConfidence(void)
{
	int i=0;				// redundant loop variable initialization
	int BUFFER = 0;

	unsigned int CAMERA_NDATA[12];

	while(BUFFER!=255)
	{
		BUFFER=uart_getbyte();
	}

	while(i<10)
	{
		CAMERA_NDATA[i]=uart_getbyte();
		i++;
	}

	return CAMERA_NDATA[9];
}
```

APPENDIX E: MARV CODE

```
/********
 * MARV *
 ********/

 /************************************************************
 * MARV is an autonomous multi-terrain vehicle that can move *
 * in the water as well as on land.  These are his codes.    *
 ************************************************************/

/************
 * INCLUDES *
 ************/

#include <avr/io.h>             // standard AVR IO functions
#include "xmega.h"              // XMEGA, DELAY, SERVO, ADC
#include "uart.h"               // UART library
#include "cmucam.h"             // CMUcam Functions
#include "lcd.h"                // LCD output library
#include "Motor.h"              // Tank, Airsoft, Propeller Functions

/*************
 *  DEFINES  *
 *************/

 #define YES                    1
 #define NO                             0
 #define PWM_FORWARD            100
 #define PWML_NEUTRAL           -12
 #define PWMR_NEUTRAL           15
 #define CLR                    0x01
 #define HALF_SEC               500
 #define ONE_SEC                1000
 #define TWO_SEC                2000
 #define THREE_SEC              3000
 #define FOUR_SEC               4000
 #define FIVE_SEC               5000
 #define TWENTY_ms              20
 #define FIFTY_ms               50
 #define DEBUG_LIGHT            0x01
 #define DEBUG_TRUE                     1
 #define DEBUG_FALSE            0
 #define IR_R_CHECK_VALUE 1000
```

```
#define IR_L_CHECK_VALUE  1000
#define N                                       78
#define BUMP                                    PORTJ_IN;

/*************
 * MAIN code *
 *************/

int main(void)
{
        /***  Function Initializations ***/
                xmegaInit();            //initialize XMEGA
                delayInit();            //initialize delay capabilities
                lcdInit();                      //initialize LCD display
                tankInit();                     //initialize tank motor function

                                PORTJ_DIR = 0x00;
                                lcdData(CLR);
                                lcdString("MARV");
                                lcdGoto(1,0);
                                lcdString("  --Josh Phillips");
                                delay_ms(2000);

                                while (~PORTJ_IN & 0x01)
                                {
                                        //
                                }

                uart_init();            //initialize USARTE0
                CMUcamInit();           //initialize CMUcam functionality
                PropInit();                     //initialize propeller function
                GunInit();                      //initialize gun function
                ADCInit();                      //initialize PortA ADC functions

        /** PORT Declarations **/
                PORTQ_DIR = DEBUG_LIGHT;
                PORTQ_OUT = DEBUG_FALSE;
                PORTJ_DIR = 0x00;

        /** Variable Declarations **/

                int IR_Right=0;                 //ADC value for right IR
                int IR_Left=0;                  //ADC value for left IR
                int i;                                  //loop variable          1
```

```
            int LineFollow = 0;             //test variable for LineFollow procedure
            int TargetPractice = 0; //test variable for Target Practice procedure
            int H20 = 0;                    //test variable for water procedure
            int SERVO_POS;                          //Servo Position
            int CONFIDENCE;                         //Confidence
            int var1,var2,var3;             //temporary holders
            unsigned char ServoPosition[7] = {'S','1',32, 0, 0, 0, 13};  //initialize servo position
string
            int LOOP=1;

        /****************
         **  Pre-Demo  **
         ****************/

        lcdData(CLR);
        lcdString("Front Props");
        PropL_ForwardOn();
        PropR_ForwardOn();
        delay_ms(1500);
        PropL_ForwardOff();
        PropR_ForwardOff();

        lcdData(CLR);
        lcdString("Upward Props");
        PropL_UpwardOn();
        PropR_UpwardOn();
        delay_ms(1500);
        PropL_UpwardOff();
        PropR_UpwardOff();

        lcdData(CLR);
        lcdString("The Chomper");
        ClawClose();
        delay_ms(1000);
        ClawOpen();
        delay_ms(1000);
        ClawClose();
        delay_ms(1000);
        ClawOpen();
        delay_ms(100);
```

```
/*************************
 ** LineFollow Procedure **
 *************************/

delay_ms(TWO_SEC);
lcdData(CLR);
lcdGoto(0,0);
lcdString("Left/Right Motors");

while(LineFollow == 0)
{
        IR_Right = 0;           //re-initialize IR values for each loop
        IR_Left = 0;
        i = 0;

        for(i=0;i<10;i++)       //10th order running average filter for better accuracy
        {
                IR_Right += ADC6();
                IR_Left += ADC7();
        }

        IR_Right = IR_Right/10;
        IR_Left = IR_Left/10;

        if ( (IR_Right <= IR_R_CHECK_VALUE) & (IR_Left <= IR_L_CHECK_VALUE) )
        {

                PropR_UpwardOn();
                PropL_UpwardOn();
                lcdGoto(1,0);
                lcdString("FF");
        }

        else if ((IR_Right <= IR_R_CHECK_VALUE) & (IR_Left > IR_L_CHECK_VALUE) )
        {

                PropR_UpwardOn();
                PropL_UpwardOff();
                lcdGoto(1,0);
                lcdString("SF");
        }

        else if ((IR_Right > IR_R_CHECK_VALUE) & (IR_Left <= IR_L_CHECK_VALUE) )
```

```
        {

                PropR_UpwardOff();
                PropL_UpwardOn();
                lcdGoto(1,0);
                lcdString("FS");
        }

        else

        {

                PropR_UpwardOff();
                PropL_UpwardOff();
                LineFollow = 1;
                lcdGoto(1,0);
                lcdString("SS");
        }


        delay_ms(TWENTY_ms);

}


lcdData(CLR);
lcdGoto(0,0);
lcdString("Coast Line Aquired");
delay_ms(FIVE_SEC);

/**********************
 **  TARGET PRACTICE  **
 **********************/


while(TargetPractice == 0)              // until target taken out continue servo rotation
{

        lcdData(CLR);           //clear display and announce beginning of sequence
        lcdString("Target Practice");

        if(LOOP==1)
        {
                uart_sendstring("S1 20\r");             // starting position for Gun process
```

```
        delay_ms(FIFTY_ms);
        lcdGoto(1,0);
        lcdString("Rightwtard Track");
        CMUcamTC_active_red();                  // activate gun tracking
}

else if(LOOP==2)
{
        uart_sendstring("S1 64\r");    // Second (neutral) position for Gun process
        delay_ms(FIFTY_ms);
        lcdGoto(1,0);
        lcdString("Neutral Track");
        CMUcamTC_active_red();                          // active gun tracking
}

else if(LOOP==3)
{
        TargetPractice = 1;                     // Third position for Gun Process
        uart_sendstring("S1 115\r");  // final process, set to exit loop on end
        delay_ms(FIFTY_ms);
        lcdGoto(1,0);
        lcdString("Lefward Track");
        CMUcamTC_active_red();                          // active gun tracking
}

else
{
        TargetPractice=1;
}

delay_ms(1500);                         // delay for CMUcam to track value

LOOP++;                                 // increment loop counter

SERVO_POS = GetServoPos(); //Save Servo Position and Confidence Level
CONFIDENCE = GetConfidence();

lcdGoto(1,0);                   //Display Servo Position and Confidence Level
lcdString("                ");
lcdGoto(1,0);
lcdString("SERVO: ");
lcdInt(SERVO_POS);
lcdString(" CONF:");
lcdInt(CONFIDENCE);
```

```
delay_ms(ONE_SEC);

uart_sendstring('\r');
delay_ms(FIFTY_ms);

if(CONFIDENCE >=10) //if confident calculate angle correction and fire weapon
{
        SERVO_POS+=5;

        if(SERVO_POS>= 100)
        {
                var1=SERVO_POS/100;
                var2=- ( SERVO_POS - 100) / 10;
                var3=SERVO_POS-10*var2-100;
        }

        else if( (SERVO_POS >=10) & (SERVO_POS < 100) )
        {
                var1=0;
                var2=SERVO_POS/10;
                var3=SERVO_POS-(var2*10);
        }

        else if(SERVO_POS < 10)
        {
                var1=0;
                var2=0;
                var3=SERVO_POS;
        }

        else
        {
                var1=0;
                var2=0;
                var3=0;
        }

        var1+=48;
        var2+=48;
        var3+=48;

        ServoPosition[3] = var1;                //set up new servo string
        ServoPosition[4] = var2;
```

```
                        ServoPosition[5] = var3;

                        uart_sendstring(ServoPosition);//make angle correction to CMUcam
servo
                        delay_ms(50);
                        uart_sendstring("GM\r");

                        delay_ms(ONE_SEC);

                        uart_sendstring('\r');
                        delay_ms(FIFTY_ms);

                        lcdData(CLR);
                        lcdString("Target Locked");

                        delay_ms(ONE_SEC);

                        lcdData(CLR);
                        lcdString("BANG!!");
                        GunFire();                              // Fire weapon
                        delay_ms(ONE_SEC);
                }
        }
        lcdData(CLR);
        lcdString("The Coast is Clear");
        delay_ms(5000);


/*******************
 **  WATER ENTRY  **
 *******************/

        // insert test for water's presence goes here

/************************
 **  WATER FUNCTIONS  **
 ************************/
        lcdData(CLR);
        lcdString("Cargo Recovery");
        delay_ms(500);

        LOOP=1;

        while(H20 == 0)
```

```
{
        if(LOOP==1)
        {
                uart_sendstring("S1 20\r");
                delay_ms(50);

                lcdGoto(1,0);
                lcdString("Rightward         ");

                if(PORTJ_IN & 0x01)
                {
                        ClawClose();
                        H20=1;
                }
        }

        else if(LOOP==2)
        {
                uart_sendstring("S1 64\r");
                delay_ms(50);

                lcdGoto(1,0);
                lcdString("Neutral          ");

                if(PORTJ_IN & 0x01)
                {
                        ClawClose();
                        H20=1;
                }
        }

        else
        {
                uart_sendstring("S1 105\r");
                delay_ms(50);

                lcdGoto(1,0);
                lcdString("Leftward         ");

                if(PORTJ_IN & 0x01)
                {
                        ClawClose();
                        H20=1;
                }
```

```
		}

		LOOP++;

		if(LOOP>=4)
		{
			LOOP = 1;
		}

		CMUcamTC_active_red();
		delay_ms(1200);

		SERVO_POS = GetServoPos(); //Save Servo Position and Confidence Level
		CONFIDENCE = GetConfidence();

		lcdGoto(1,0);                    //Display Servo Position and
Confidence Level
		lcdString("Tracking          ");

		delay_ms(100);

		if(CONFIDENCE >=4)
		{
			while(CONFIDENCE >=4)
			{

				SERVO_POS = GetServoPos();

				if(SERVO_POS<54)
				{
					LOOP = 1;

					PropR_ForwardOn();
					PropL_ForwardOff();
				}
				else if(SERVO_POS>74)
				{
					LOOP = 3;
					PropR_ForwardOff();
					PropL_ForwardOn();
				}
				else
				{
```

```
                                LOOP = 2;
                                PropL_ForwardOn();
                                PropR_ForwardOn();
                        }

                        if(PORTJ_IN & 0x01)
                        {
                                ClawClose();
                                H20=1;
                        }

                        CONFIDENCE = GetConfidence();
                }

        }

        else
        {
                PropR_ForwardOff();
                PropL_ForwardOff();
        }

        uart_sendstring('\r');
        delay_ms(50);

        if(H20==1)
        {
                PropR_ForwardOff();
                PropL_ForwardOff();
        }
    }

    /**********************
     **  RETURN TO BASE  **
     **********************/

     lcdData(CLR);
     lcdString("On our way home");
     lcdGoto(1,0);
     lcdString("Good Job MARV");
     delay_ms(2000);
     ClawOpen();
     delay_ms(3000);
}
```