# Final Report
# for Lunar-cy

Brad Mouring
Lunar-cy
EEL 5666
Intelligent Machine Design Lab
Dr. Arroyo
Dr. Schwartz
Adam Barnett
Kevin Claycomb

# Table of Contents

## Abstract

The main thrust of this robot is to explore possible approaches to the 2008 IEEE SECon hardware competition robot. This robot must simulate a Lunar Mining game in which various "ores" must be collected and deposited in the "home base." This will require a solid platform (needed for traversing the rough terrain) as well as some form of item identification. Identification is the area I explored with this design: locating blocks before leaving "home base" so as to try to minimize the time required to get a valuable block.

## Executive Summary

The primary thrust for this project will be exploring the feasibility of creating a vision system that allows more flexibility than currently available systems, including the (in)famous CMUcam. This goal is a means to an end, namely examining a possibility for an improved system for visual location of objects, a common goal of robotic competitions.

## Introduction

The problems presented by this project range from the standard issues (such as obstacle avoidance) to more complex (such as discerning a block of the same color as the background it sits in). To overcome the issues presented by the problem, various sensors will be exploited to gain knowledge of the surrounding environment and, thereafter, modify the behavior based on the observed environment.

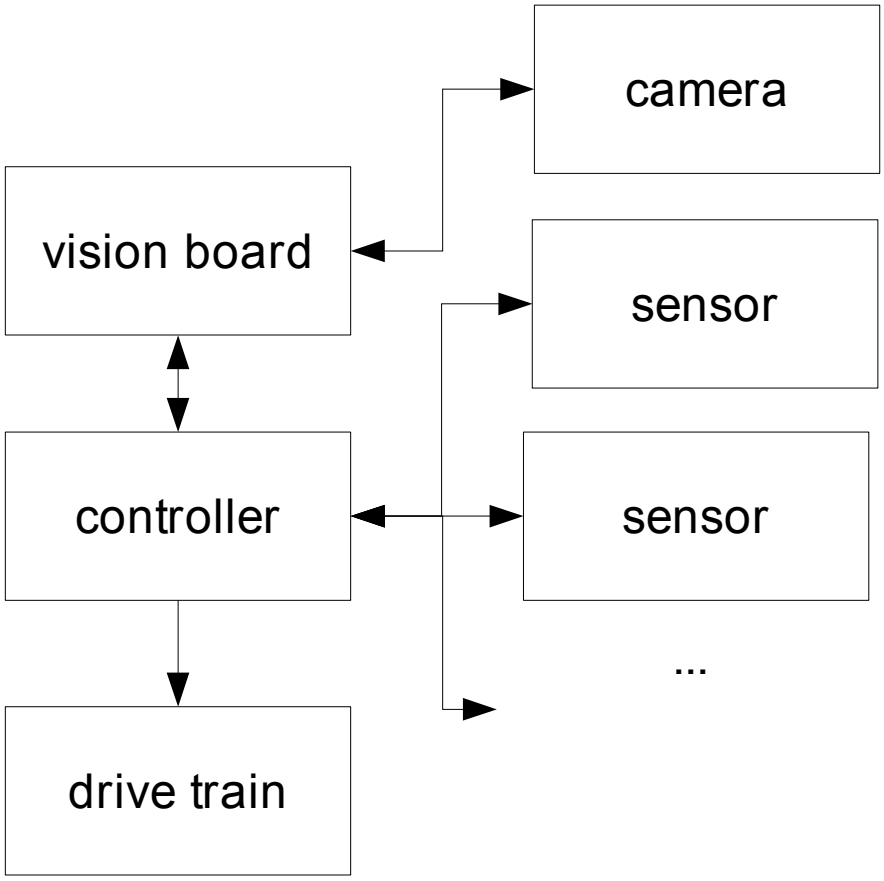
The reason for approaching this problem personally is the interest in advancing my personal knowledge of computer vision and basic artificial intelligence as well as mechanical skills and the tools that accompany them. It would have been quite easy to simply work towards personal strengths alone while planning the robot, however it is much more rewarding to expand new skill sets rather than fall back on existing ones.

The scope of such a project, as alluded to above, is many "locate items, find an efficient method to collect them" problems. The framework for identifying the goals will need to be changed, however the remainder of the system can remain relatively untouched and perform the desired new task.

This report will discuss proposed ideas and work already performed in the overall system construction, the platform design, motion or actuation systems and ideas, sensor selection and use, system behaviors, and will discuss experiments performed in the process of discovering the use and characteristics of the various subsystems.

## Integrated System

The system will consist of a primary drive controller, the drive train itself, the sensors it will use to determine how to proceed, and a vision system that will inform the drive controller where to go. A simple diagram laying out the overall hierarchy of that system is provided below.




This modular approach to separating platform control from vision systems allows for simplified integration between different revisions of the base, allowing for improvement and exploration in terms of testing out experimental base ideas.

From the flowchart, the vision board can communicate to the controller board as to the heading of the next block to acquire, while the controller board, interfaced with the pertinent sensors, can best decide how to get to the location dictated by the vision board while avoiding obstacles. The vision board simply indicates to the drive/collision avoidance board a heading and a distance to the desired object, and once within a reasonable range, will relinquish control to the sensors on the drive board to locate and acquire the blocks, waiting to hear from the driver board when that job is complete to begin looking for the next block.

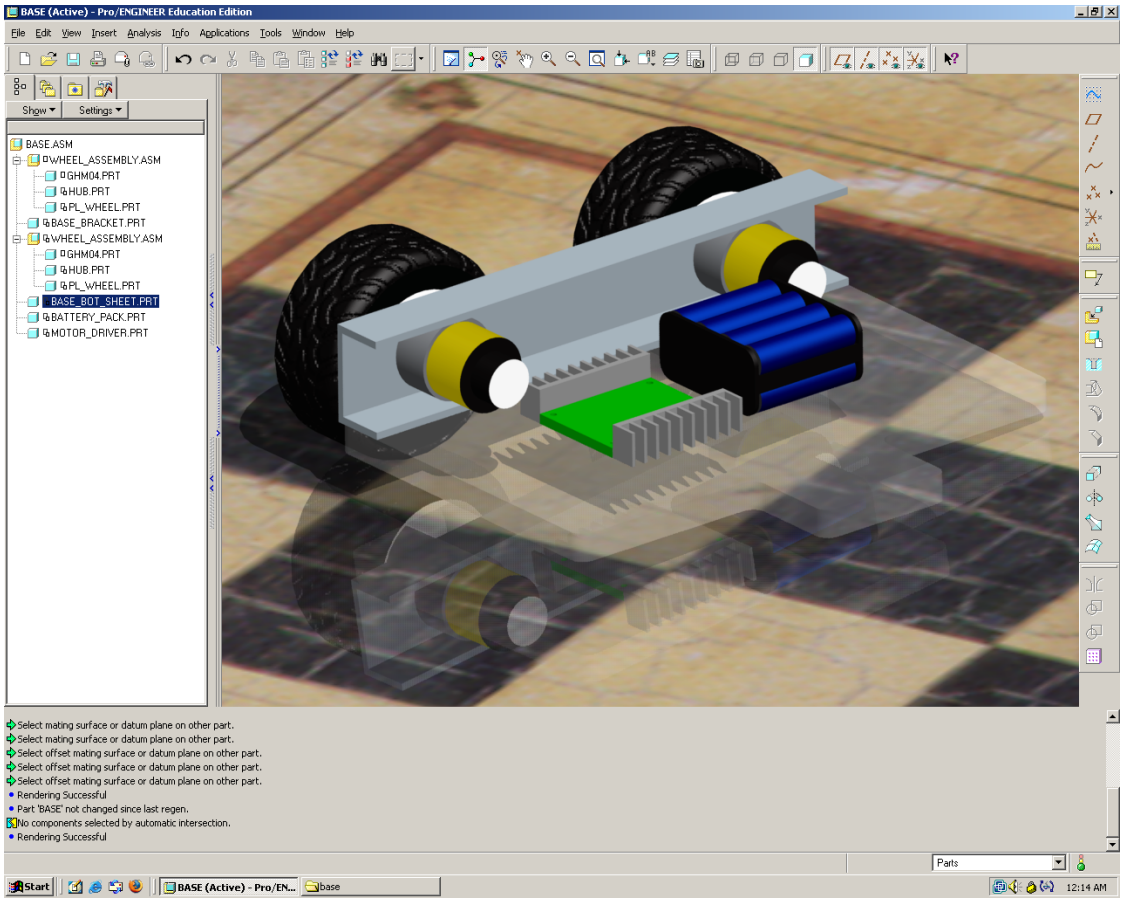
## Mobile Platform

The base for the project must be able to cope with semi-rugged terrain (pea gravel and sand adhered to the playing field, an easier goal than the previous rules indicating lava rock and marble chips). Additionally, if design decisions can be made to assist the project as a whole in the picking up of blocks, this would be a desirable feature.

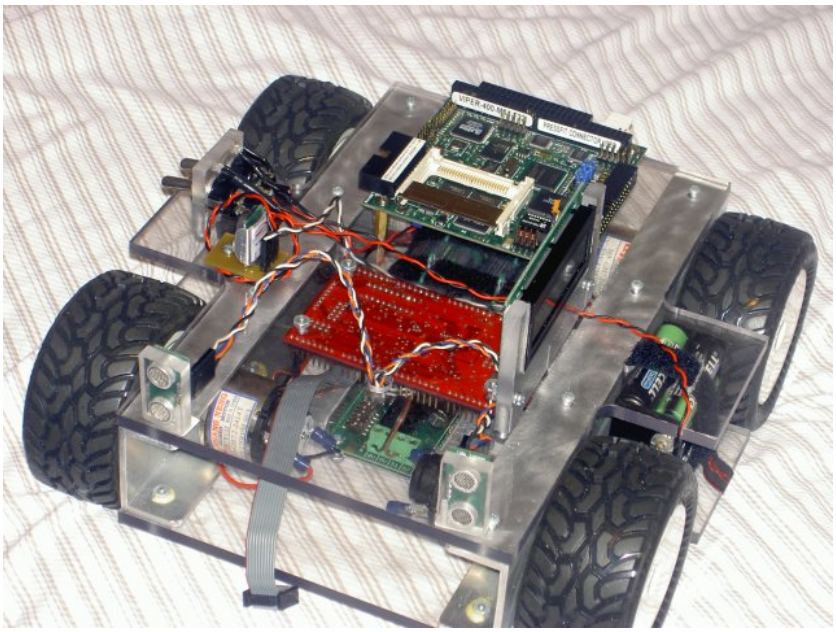
The terrain the platform must deal with demands a rugged, well-constructed base if it is to withstand a round in competition, let alone tens or hundreds of rounds of testing. In order to facilitate this demand, a planned, mechanically sound design will be implemented in a CAD program. A tank-drive, four-wheel drive base was chosen for the design. This provides a

simple, rugged, powerful base on which to work from.

The figure below shows a partial construction of the CAD-designed base as has been constructed




Below is a photograph of the base as it was actually constructed.



A hybrid roller/launcher was planned and constructed. Upon completion, it became obvious that the motors chosen for the design were simply too weak to pick up the blocks, and as such the idea was put on hold while a simple arm was constructed to pick up the blocks (in the interest of finishing the robot on-time since the correct motors for the roller would require ordering online).

A difficult part of building the actual base has been finding a shop willing to assist you in making the parts needed for your machine. Also, a very real pitfall is spending too much time on details of the base which results in erosion of the little time we have to build these robots in the first place. This has been without a doubt my biggest mistake this term (however it did result in a fantastically strong and workable base).

## Actuation

Actuation is required to “bring the robot to life,” so to speak. The actuation must move parts of the robot in order to accomplish the goals specified by the solution to the problem to be solved.

As a result of the planning of this robot, relatively little is needed in the way of actuation. The wheel system will be gearhead brushless motors to ensure they can traverse the terrain powered through an isolated power system to prevent spike-related system resets.

I chose the Lynxmotion GHM-04 50:1 7.2 volt gearhead motors for their voltage rating, speed, and torque specifications. As there will be 4 of them in a tank-drive configuration, this provided me with ample power to overcome the original obstacles let alone the simplified terrain in the latest rules. These will be controlled through a Dimension Engineering Sabertooth 2x10 dual channel motor driver (10 amps continuous per channel). Testing proved that anything that was too large for my base to push would be attempted to climbed instead.

As noted in the **Platform** section, I ended up having to make a concession and use a servo-based gripper due to poor motor choice. These servos, when combined with polycarbonate pieces, work to pick up the blocks when encountered.

Additional actuation is planned for future work to move the camera to “scan” the environment. This will simplify location of the blocks as currently the entire base must rotate in order to locate the blocks.

## Sensors

See sensor report.

## Behaviors

The behaviors ingrained in the code running on the robot serve to analyze the data received from the sensors and adjust the operation of the robot to fit the perceived situation. It is this facet that gives robots an anthropomorphic appearance. The goal with the behaviors is to create as failsafe a behavior scheme for all aspects of the robot as possible to ensure predictable responses in unpredictable environments.

The base can, without input from the camera, operate in a constantly moving obstacle avoidance mode. This behavior will examine the data from the sonars. When it detects a block (as evidenced by the lower sonar plane pinging close and the upper plane failing to ping or pinging far), the robot will attempt to collect the block. If a wall or another robot is detected (both planes detect a close object), the robot will avoid the object smoothly by adjusting the motor drive values depending on the perception of objects rather than halting current locomotion to force a new direction.

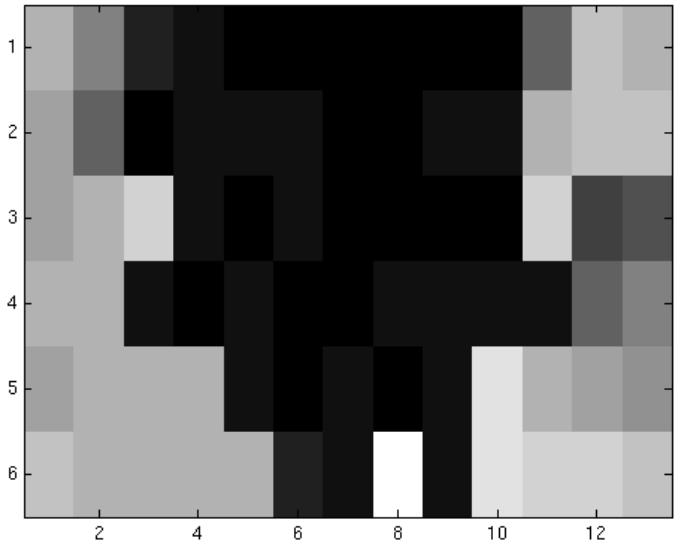
The vision board works to find the blocks in the initial scan phase at the start of the competition. Eventually, a system to catalog locations and values of blocks in a “best path” order will be implemented. While moving, the vision system will be tracking the current target block, updating the controller board with new heading information with a frequency of 8 Hz. Future plans also include coupling the vision system and the encoders to reacquire the next block visually. Simply put, I did not have the time to implement this system in the current robot.

## Experimental Layout and Results

Experiments are performed to better understand the characteristics and capabilities of various sensors and systems in the robot's design. Thus, they play a vital role in determining just how successful a theoretical application translates to real-world application.

In order to understand the characteristics of the sonar modules purchased, I performed a number of experiments to understand how it works as well as possible limitations. The first and simplest test was to figure out the sonar's effective range in various environments. It became evident quickly that edges in the environment would result in readings even though they were not objects, per se. In as sterile an environment as possible (outdoors, pointed upwards a bit) the sensor had a range of approximately 12 feet, a bit short of the advertised 6 meters (approximately 19 feet). However, considering the target environment, these results are sufficiently satisfactory.

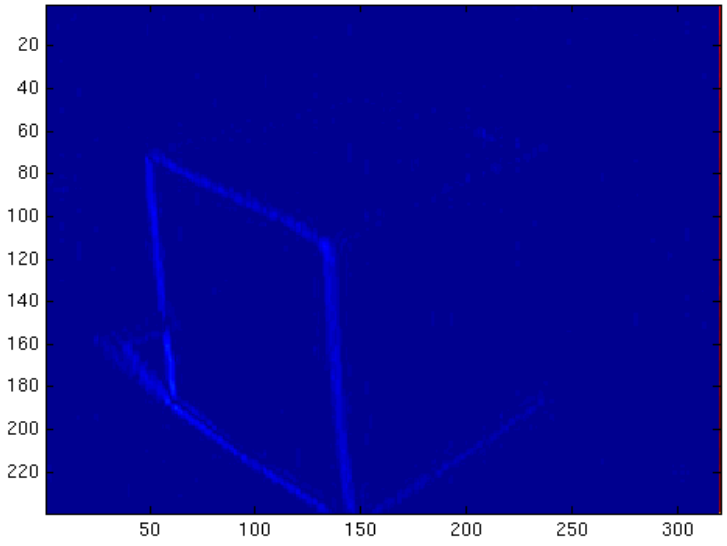
The next experiment was to determine the "sensitivity map" of the sensor. A ½" grid was created and a neutral target (a ½" diameter cylinder) was moved through the grid, while being measured. Measurements were then compared to the calculated target measurement. The pattern of experimentally-derived sensitivity (the difference from the calculated and the observed) is shown below. The dark sections are areas where the sensor is near the correct measurement (perceived distance versus actual object distance) while the lighter areas are where the sensitivity is low, indicating that the sensor cannot "see" the object.



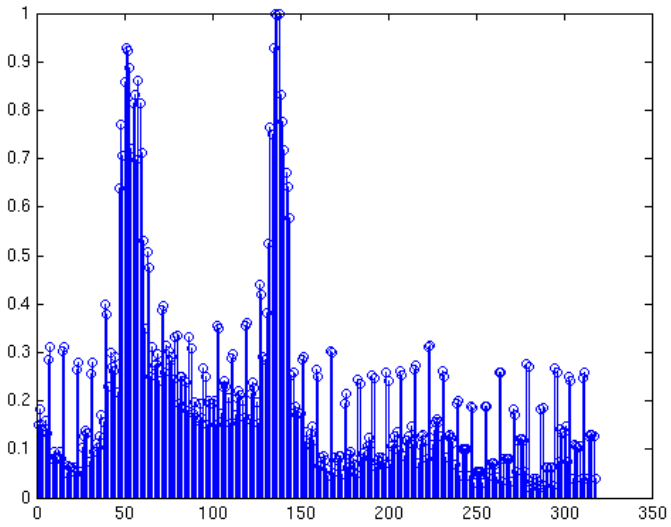

Notice the "blind spot" very close to the sensor and to the right. This is likely due to the object being missed by the ping while also obscuring the receiver.

Initial test code (done in MATLAB to allow quick prototyping of ideas) can take a difficult image, as shown below, and gather useful information about the image.

Clearly, color information cannot be used to determine the location of the block. As such, attempting to accentuate high-frequency components of the image can serve to assist in locating the block in the image. Below is the result of a simple vertical-accentuating Sobel filter.




With these results, a simple horizontal-running histogram can be used to locate the peaks that correspond to detected edges. At this point, locating the X values for the edges (or possibly just locating the weighted center of the detected edges) is extremely simple.




These results are quite promising, however this code did not make it into my final design due to time constraints.

A simple experiment was to verify that my encoder counter board was functioning correctly. I simply created a test circuit, hooked the encoder and motor to the board, and used the motor controller to spin the motor slowly in one direction, then the other. The output of the counter was sent to s series of bar LEDs where it was shown that the counter was working correctly, including holding at either maxima (0x80 or 0x7F) to prevent erroneous readings due to overflow.

# Conclusion

The actual work accomplished thus far consists of modeling of the base followed with actual construction of the base, familiarization with the micro controller chosen, familiarization and characterization of the sensors picked, construction of a vision system (structural, hardware and software), and created code to locate and retrieve a block.

One limitation would be the level of image processing possible on an embedded XScale system that is without a floating-point unit. Also, due to time constraints, my initial goals for this robot during this semester had to be reduced. Instead of completing a test run of the competition, I simply took the first steps towards a vision-based system to try to better our chances of success in the competition.

Future work involves continued work on a filtering/detection system that can locate blocks on like-colored backgrounds, integration of the encoder counter into the behavior system to indicate distance traveled, and further work on a roller pickup mechanism.

# Documentation

"8-Bit AVR Microcontroller with 128K Bytes in-System Programmable Flash." Atmel Corporation. 2006. 14 June 2007 <http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf>.

Cassia, Fernando. "D-Link DCS-900 IP Camera is a Flipping Bargain." The Inquirer. 30 Dec. 2005. 09 Apr. 2006 <http://www.theinquirer.net/default.aspx?article=28632>.

Lane, Thomas G. "Using the IJG JPEG Library." 1998. Independent JPEG Group. 14 Nov. 2006 <http://ou800doc.caldera.com/en/jpeg/libjpeg.txt>.

"MAVRIC-IIB Mega AVR Integrated Controller II Revision B Technical Manual." BDMicro. 30 May 2006. 14 June 2007 <http://www.bdmicro.com/mavric-iib/mavric-iib.pdf>.

# Appendix A

## *SRF10 Header*

```
#ifndef AVR_TWI_H_
#define AVR_TWI_H_

#include<util/twi.h>
#include<inttypes.h>

#define TWI_INIT (_BV(TWINT)|_BV(TWEN)|_BV(TWSTA))
#define TWI_TX (_BV(TWINT)|_BV(TWEN))
#define RANGE_IN 0x50
#define RANGE_CM 0x51
#define RANGE_US 0x52
#define INIT_SUCCESS 0x08
```

```
#define REINIT_SUCCESS 0x10
#define MASTER_TO_SLAVE 0
#define SLAVE_TO_MASTER 1
#define TWI_TIMEOUT 100
#define SRF_1 0xE2
#define SRF_0 0xE0

extern void delay_ms(int);

/*
 * Blindly signal the end of transmission
 * (must check status before attempting
 * further transmission)
 */
void twi_stop(void)
{
      TWCR = (_BV(TWINT)|_BV(TWEN)|_BV(TWSTO));
      return;
}

/*
 * Initialize master communication to the slave devices
 */
int8_t twi_init(uint8_t estatus)
{
      uint8_t status,count;
      count = 0;

      TWCR = TWI_INIT;
      while(!(TWCR & _BV(TWINT)) && (count<TWI_TIMEOUT))
      {
            delay_ms(1);
            count++;
      }
      if(count>=TWI_TIMEOUT)
            return -1;
      status = TWSR;
      if(status != estatus)
            return -2;
      return 0;
}

/*
 * Communicate with the slave device at address addr and prep
 * them for a read from master (0) or a write to master (1)
 */
int8_t twi_rw_init(uint8_t addr, uint8_t op, uint8_t estatus)
{
      uint8_t tosend,status,count;
      count = 0;
      tosend = addr | (op & 0x01);
      TWDR = tosend;
      TWCR = TWI_TX;
      while(!(TWCR & _BV(TWINT)) && (count < TWI_TIMEOUT))
      {
            delay_ms(1);
            count++;
      }
      if(count>=TWI_TIMEOUT)
```

```
            return -1;
      status = TWSR;
      if((status & 0xF8)!=estatus)
            return -2;
      return 0;
}

/*
 * As the name indicates, transmit data from master
 * to slave device
 */

int8_t twi_data_tx(uint8_t data, uint8_t estatus)
{
      uint8_t status,count;
      count = 0;
      TWDR = data;
      TWCR = TWI_TX;
      while(!(TWCR & _BV(TWINT)) && (count < TWI_TIMEOUT))
      {
            delay_ms(1);
            count++;
      }
      if(count>=TWI_TIMEOUT)
            return -1;
      status = TWSR;
      if((status & 0xF8)!=estatus)
            return -2;
      return 0;
}

/*
 * As the name indicates, receive data from slave device
 */

int8_t twi_data_rx(uint8_t *data,uint8_t ack, uint8_t estatus)
{
      uint8_t status,count;
      count = 0;
      if(ack)
            TWCR = (_BV(TWINT)|_BV(TWEN)|_BV(TWEA));
      else
            TWCR = (_BV(TWINT)|_BV(TWEN));
      while(!(TWCR & _BV(TWINT)) && (count < TWI_TIMEOUT))
      {
            delay_ms(1);
            count++;
      }
      if(count>=TWI_TIMEOUT)
            return -1;
      status = TWSR;
      if((status & 0xF8)!= estatus)
            return -2;
      *data = TWDR;
      return 0;
}

/*
 * A function to perform a ping-request
```

```
 * operation to the SRF10 module
 */

int8_t srf10_ping(uint8_t addr,uint8_t cmd)
{
      if(twi_init(INIT_SUCCESS))
            return -1;
      if(twi_rw_init(addr,MASTER_TO_SLAVE,TW_MT_SLA_ACK))
            return -2;
      if(twi_data_tx(0,TW_MT_DATA_ACK))
            return -3;
      if(twi_data_tx(cmd,TW_MT_DATA_ACK))
            return -4;
      twi_stop();
      return 0;
}

/*
 * A function to receive the ranging
 * info from the SRF10 after a ping
 */

int8_t srf10_range(uint8_t addr, uint16_t *range)
{
      uint8_t range_h, range_l;
      if(twi_init(INIT_SUCCESS))
            return -1;
      if(twi_rw_init(addr,MASTER_TO_SLAVE,TW_MT_SLA_ACK))
            return -2;
      if(twi_data_tx(2,TW_MT_DATA_ACK))
            return -3;
      if(twi_init(REINIT_SUCCESS))
            return -4;
      if(twi_rw_init(addr,SLAVE_TO_MASTER,TW_MR_SLA_ACK))
            return -5;
      if(twi_data_rx(&range_h,1,TW_MR_DATA_ACK))
            return -6;
      if(twi_data_rx(&range_l,0,TW_MR_DATA_NACK))
            return -7;
      twi_stop();

      *range = (range_h<<8)|range_l;

      return 0;
}

void srf10_init()
{
      TWSR &= 0xFC;
      TWBR = 28;
      TWCR |= _BV(TWEN);
}

#endif /*AVR_TWI_H_*/
```

## *MJPEG Decompression code*

```
#include<stdlib.h>
```

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<pthread.h>
#include"net_get_stream.h"

typedef struct{
        struct jpeg_source_mgr stdmgr;

        /* Other pertinent, network-oriented vars here */

        JOCTET *buffer;
        int bytes_left;
} jpeg_mem_src;

char global_buff[BUF_SIZE]; /* 64k */
int gb_size;
int reenter = 0;

pthread_mutex_t imbuff_lock = PTHREAD_MUTEX_INITIALIZER;
int imbuff_stat = IMBUFF_IDLE;

/* This is used to make a request to load a new image */
void imbuff_make_request()
{
        printf("Making image request\n");
        pthread_mutex_lock(&imbuff_lock);
        imbuff_stat = IMBUFF_REQ;
        pthread_mutex_unlock(&imbuff_lock);
}

/* Local function for creating a TCP socket */
int open_tcp()
{
        struct sockaddr_in sa;
        int sock;

        /* Attempt to open a socket (no connection yet) */
        if((sock = socket(AF_INET,SOCK_STREAM,0))<0)
        {
                fprintf(stderr,"ERR: Unable to open socket\n");
                return -2;
```

```
        }

        /* Setup the server socket info */
        bzero(&sa,sizeof(sa));
        sa.sin_family = AF_INET;
        sa.sin_port = htons(PORT);
        sa.sin_addr.s_addr = inet_addr(ADDR);

        /* Attempt a connect */
        if(connect(sock,(struct sockaddr*)&sa,sizeof(sa))<0)
        {
                fprintf(stderr,"ERR: Unable to connect to socket\n");
                close(sock);
                return -3;
        }

        /* If all goes well, return a handle that other functions may use for TCP comm */
        return sock;
}

/* Simple routine to try to locate the first instance of a bytestring */
int find_instance(char *buff,char *token,int bsize)
{
        int len = strlen(token),i,j;
        j = 0;
        for(i = 0;(i<(bsize-len)) && (j<len);i++)
        {
                if(buff[i] != token[j])
                {
                        j = 0;
                }
                else
                {
                        j++;
                }
        }
        if(j == len)
        {
                return i;
        }
        else
        {
                return -1;
```

```
        }
}

/* A simple adaptation of the above to locate the start of a JPEG */
int locate_jpeg(unsigned char *buff, int size)
{
        int i,j = -1;
        for(i = 0;i<(size-4) && j<0;i++)
        {
                if((buff[i]==0xFF) && (buff[i+1]==0xD8) && (buff[i+2]==0xFF) && (buff[i+3]==0xE2))
                {
                        j = i;
                }
        }
        return j;
}

/* This routine will fill the global buffer when a request
 * has been made via the imbuff_stat mutex'd variable */
void load_gbuffer(char *buff,int size, int offset)
{
        int i;
        printf("In load_gbuffer\n");
        pthread_mutex_lock(&imbuff_lock);
        printf("Lock obtained");
        if(imbuff_stat == IMBUFF_IDLE)
        {
                pthread_mutex_unlock(&imbuff_lock);
                printf("idle, so buffer not filled\n");
                return;
        }
        memset(global_buff,0,BUF_SIZE);
        for(i = 0;i<size;i++)
        {
                global_buff[i] = buff[i+offset];
        }
        gb_size = size;
        imbuff_stat = IMBUFF_FULL;
        pthread_mutex_unlock(&imbuff_lock);
        printf("buffer loaded\n");
}

/* This routine takes raw data from the HTTP stream and locates a full JPEG
```

```
 * in the MJPEG stream */
int strip_jpeg(char *buff,int size)
{
	int offset,jsize;
	offset = find_instance(buff, "Content-Length:",size);
	if(offset!=-1)
	{
		printf("Found instance at %d\n",offset);
		jsize = atoi(&(buff[offset]));
		offset += locate_jpeg((unsigned char*)&(buff[offset]),(size - offset));
		printf("%d\n",jsize);
		load_gbuffer(buff,jsize,offset);
		return 0;
	}
	else
	{
		printf("Unable to find instance in passed buffer\n");
		return -1;
	}
}

/* Used when part of a JPEG is discovered after a full JPEG has been
 * loaded into the global buffer */
void shift_buffer(char *buffer,int offset,int size)
{
	int i;
	for(i = 0;i<size;i++)
	{
		buffer[i] = buffer[i+offset];
	}
	buffer[size] = 0;
}

/* This is the worker thread that continuously grabs data from the camera */
void *get_stream(void *sockp)
{
	char tempbuff[BUF_SIZE*2];
	int ret, rtot,size,jsize,offset,sock;
	unsigned char state;
	char *msg;

	sock = (int)sockp;
```

```
printf("In get_stream(): %lu, %d\n",(unsigned long)sockp,sock);

size = BUF_SIZE*2;

msg = (char *)malloc(strlen(MSG_TO_SEND)*(sizeof(char)));
if(!msg)
{
        printf("Err: Unable to allocate mem for sending msg\n");
        return (void*)-1;
}

sprintf(msg,"%s",MSG_TO_SEND);

ret = (int)(send(sock,msg,(int)(strlen(msg)),0));
if(ret != strlen(msg))
{
        printf("Err: Message not completely sent!\n");
        free(msg);
        ret = -2;
        return (void*)-2;
}

free(msg);

printf("Message sent, waiting for reply\n");

memset(tempbuff,0,size);
rtot = 0;
state = WAIT;
/* Later change this to have a re-starting outter loop in case of disaster */
while((ret = (int)(recv(sock,&(tempbuff[rtot]),(size-rtot),0)))>0)
{
        rtot += ret;
        /* if we are waiting to see the start of a new Jpeg image */
        if(state == WAIT)
        {
                ret = find_instance(tempbuff,"Content-length:",rtot);
                /* We've found a part of the header we care about */
                if(ret!=-1)
                {
                        offset = ret;
                        //printf("Found instance at %d\n",ret);
                        /* get the size of the image we care about */
```

```
                        jsize = atoi(&(tempbuff[ret]));
                        //printf("%d\n",jsize);
                        //while(1);

                        /* Attempt to find the index into the temp buffer that starts the JPEG */
                        if((ret = locate_jpeg((unsigned char*)&(tempbuff[offset]),rtot-offset))!=-
1)
                        {
                                offset += ret;
                                /* Do we have the entire JPEG yet? yes? */
                                if(jsize<(rtot - offset))
                                {
                                        pthread_mutex_lock(&imbuff_lock);
                                        if(imbuff_stat == IMBUFF_REQ)
                                        {
                                                pthread_mutex_unlock(&imbuff_lock);
                                                printf("(W)Loaded image: %d\n",jsize);
                                                load_gbuffer(tempbuff,jsize,offset);
                                                pthread_mutex_lock(&imbuff_lock);
                                        }
                                        pthread_mutex_unlock(&imbuff_lock);
                                        /* if we find the start of the next one, shift it over and
set the state */
                                        if((ret = find_instance(&(tempbuff[offset+jsize]),"Content-
length:",(rtot-(offset+jsize))))!=-1)
                                        {
                                                shift_buffer(tempbuff,(ret+offset+jsize),(rtot-
(ret+offset+jsize)));
                                                state = RECEIVING;
                                                rtot = rtot - (ret+offset+jsize);
                                                offset = ret;
                                                jsize = atoi(&(tempbuff[ret]));
                                        }
                                        else
                                        {
                                                memset(tempbuff,0,rtot);
                                                state = WAIT;
                                                rtot = offset = jsize = 0;
                                        }
                                }
                                else
                                {
                                        state = RECEIVING;
                                }
                                /* end IF have full JPEG */
```

```
                        }
                        /* end IF found JPEG */
                }
                /* end IF found "Content-length" */
        }
        /* end IF state == WAIT */

        /* We've already seen the start of a JPEG */
        if(state == RECEIVING)
        {
                /* We've just gotten a full JPEG */
                if(jsize < (rtot-offset))
                {
                        pthread_mutex_lock(&imbuff_lock);
                        if(imbuff_stat == IMBUFF_REQ)
                        {
                                pthread_mutex_unlock(&imbuff_lock);
                                printf("(R)Loaded image: %d\n",jsize);
                                load_gbuffer(tempbuff,jsize,offset);
                                pthread_mutex_lock(&imbuff_lock);
                        }
                        pthread_mutex_unlock(&imbuff_lock);
                        /* if we find the start of the next one, shift it over and set the state
*/
                        if((ret = find_instance(&(tempbuff[offset+jsize]),"Content-length:",(rtot-
(offset+jsize))))!=-1)
                        {
                                shift_buffer(tempbuff,(ret+offset+jsize),(rtot-
(ret+offset+jsize)));
                                state = RECEIVING;
                                rtot = rtot - (ret+offset+jsize);
                                offset = ret;
                                jsize = atoi(&(tempbuff[ret]));
                        }
                        else
                        {
                                memset(tempbuff,0,rtot);
                                state = WAIT;
                                rtot = offset = jsize = 0;
                        }
                }
                /* Still waiting on a full JPEG */
                else
                {
```

```
                                state = RECEIVING;
                        }
                }
                //printf("Read %d so far\n",rtot);
        }
        return NULL;
}

/* A now-defunct routine that used to grab a single frame */
int get_image(int sock, char *buff, int size)
{
        int ret, rtot;
        char *msg;

        msg = (char *)malloc(strlen(MSG_TO_SEND)*(sizeof(char)));
        if(!msg)
        {
                printf("Err: Unable to allocate mem for sending msg\n");
                return -1;
        }

        sprintf(msg,"%s",MSG_TO_SEND);

        ret = (int)(send(sock,msg,(int)(strlen(msg)),0));
        if(ret != strlen(msg))
        {
                printf("Err: Message not completely sent!\n");
                free(msg);
                return -1;
        }

        free(msg);

        memset(buff,0,size);
        rtot = 0;
        while(((ret = (int)(recv(sock,&(buff[rtot]),(size - rtot),0)))>0)&&((rtot + ret) < size))
        {
                rtot += ret;
        }
        return rtot;
}

/* Setting up the JPEG decompression source */
```

```
void stream_mgr()
{
        int ret,sock;
        pthread_t net_listener;

        sock = open_tcp();
        if(sock<2)
        {
                printf("Err: Socket not opened!\n");
                return;
        }
        /* Have we already started the listener thread? No? then do it! */
        if(!reenter)
        {
                reenter = 1;
                ret = pthread_create(&net_listener, NULL, get_stream, (void*)(long)sock);
        }
        /* If it's been started, just make a request to load the buffer */
        else
        {
                imbuff_make_request();
        }
}

/* Used by the JPEG decompression routine to get data for
 * decompression.  NOT TO BE CONFUSED WITH LOAD_GBUFFER */
static int load_buffer(char *buff)
{
        pthread_mutex_lock(&imbuff_lock);
        while(imbuff_stat != IMBUFF_FULL)
        {
                if(imbuff_stat == IMBUFF_IDLE)
                {
                        pthread_mutex_unlock(&imbuff_lock);
                        imbuff_make_request();
                        pthread_mutex_lock(&imbuff_lock);
                }
                pthread_mutex_unlock(&imbuff_lock);
                usleep(125000);
                pthread_mutex_lock(&imbuff_lock);
        }
        pthread_mutex_unlock(&imbuff_lock);
```

```
        memcpy(buff,global_buff,gb_size);

        return gb_size;
}

/* JPEG library source init routine */
METHODDEF(void) init_source(j_decompress_ptr cinfod)
{
        printf("In init_source\n");
        ((jpeg_mem_src*)(cinfod->src))->bytes_left = -1;
}

/* JPEG decompression data request routine */
METHODDEF(boolean) fill_input_buffer(j_decompress_ptr cinfod)
{
        jpeg_mem_src *src = (jpeg_mem_src *)cinfod->src;
        size_t nbytes;

        printf("In fill_input_buffer\n");

        nbytes = load_buffer((char *)(src->buffer));

        if(nbytes <= 0 )
        {
                if(src->bytes_left == -1)
                {
                        ERREXIT(cinfod,JERR_INPUT_EMPTY);
                }
                WARNMS(cinfod,JWRN_JPEG_EOF);
                /* Bogus end-of-image code */
                src->buffer[0] = 0xFF;
                src->buffer[1] = 0xD9;
                nbytes = 2;
        }

        src->stdmgr.next_input_byte = src->buffer;
        src->stdmgr.bytes_in_buffer = nbytes;
        src->bytes_left = nbytes;

        return TRUE;
}

/* JPEG decompression lib skip-data routine.
```

```
 * I have no idea when this would be used */
METHODDEF(void) skip_input_data(j_decompress_ptr cinfod, long nbytes)
{
        jpeg_mem_src *src = (jpeg_mem_src *)cinfod->src;
        if(nbytes > 0)
        {
                if(nbytes > (src->bytes_left))
                {
                        nbytes = src->bytes_left;
                }
                src->stdmgr.next_input_byte += (size_t) nbytes;
                src->stdmgr.bytes_in_buffer -= (size_t) nbytes;
        }
}

/* JPEG decompression finsh-up routine. Called on destruction
 * of decompression sequence */
METHODDEF(void) term_source(j_decompress_ptr cinfod)
{
        ((jpeg_mem_src*)(cinfod->src))->bytes_left = -1;
        imbuff_make_request();
}

/* Setup of JPEG source for MJPEG stream */
GLOBAL(void) jpeg_memory_src(j_decompress_ptr cinfod /* Eventually, add IP address here! */)
{
        jpeg_mem_src *src;

        if(!cinfod->src)
        {
                cinfod->src = (struct jpeg_source_mgr *)(*cinfod->mem-
>alloc_small)((j_common_ptr)cinfod,JPOOL_PERMANENT,sizeof(jpeg_mem_src));
                ((jpeg_mem_src *)(cinfod->src))->buffer = (JOCTET *)(*cinfod->mem-
>alloc_small)((j_common_ptr)cinfod,JPOOL_PERMANENT,BUF_SIZE*sizeof(JOCTET));
        }
        src = (jpeg_mem_src *) cinfod->src;
        src->stdmgr.init_source = init_source;
        src->stdmgr.fill_input_buffer = fill_input_buffer;
        src->stdmgr.skip_input_data = skip_input_data;
        src->stdmgr.resync_to_restart = jpeg_resync_to_restart; /* Look into re-implementing this */
        src->stdmgr.term_source = term_source;
        /* Put other useful data into the src_mgr here, get it as passed params, like IP address */
        src->stdmgr.bytes_in_buffer = 0;
        src->stdmgr.next_input_byte = NULL;
```

```
        stream_mgr();
}
```