Ball High Technical Report

“PET”

Ball High School Underwater Robotics

Students

Stephen Schuler
Andrew Urbina
Chris Vineski
Evan Segal
Long Le
Jon Prater
Ronald Caraway

Instructors

Mr. Kim Page
Mr. Pete

June 14, 2004

Abstract:

Our goal this year was to build a fully functional underwater robot that could efficiently complete the tasks specified in the 3rd Annual MATE Underwater Robotics Competition. The team was fortunate in that they were able to use the previous year's 3rd place robot as the basis for their design. The team stripped the old ROV down to all of its bare parts, and redesigned the ROV with a thinner frame to hold a pneumatically-controlled arm, dual sonar, and non-blowable ballast tanks, making this year's ROV neutrally buoyant. Even though the ROV itself was completely redesigned, the surface computer stayed the same. A major problem was the team's complete lack of responsibility and organizational skills. However, members still agree that we have a chance to win this year.

Budget

PVC Pipe	$221.00
Water proof Cameras	$340.00
Plexi-glass	$98.00
Aluminum sheets	$23.00
Tool Chest	$76.00
Tie Wraps	$46.00
Duct tape rolls	$34.00
Cordless drill	$50.00
Southwest Airlines e-Ticket	$2283.20
Hotel Rooms	$430.00
Check Valves	$15.00
Connector Pins	$5.00
Quick Disconnect Adapters	$36.00
JB Weld	$16.00
Potting Compound	$53.00
Trolling Motors	$320.00
Pneumatic Cylinders	Previous Year
On board computer/parts	Last year
Surface Computer	Mr. Page’s
Flat Screen Monitor	Donated By Mr. Roe of GISD
Aluminum Housing for Computer Box	Donated By College of the Mainland
Total:	$4046.20

Design Rationale

Great detail and thought went into the design of this year's ROV. At first, our team concluded that we could use the previous year's ROV and make a few modifications. But there was too much interference with the preexisting computer box and manifold. First of all, the way that the original box was designed, no one could run anything new through the wall of the box. Second, the manifold inside the computer box didn't have enough room for anything new. Most of the current space was taken up by the large ballast system. So the team came to a unanimous decision that the ROV needed to be stripped and rebuilt.

Instead of using 1 ½ inch PVC, the team switched to ¾ inch pipe that offered maximum strength and minimum space. The frame was designed to be a roll cage around the motors and computer box. The motors are to be connected by an x-joint which then sends a shaft straight up into the bottom of the waterproof box for a direct link with the on board computer. This whole processes took about 3 months due to off and on team members and bad communication skills. The computer box was re measured and elongated and constructed out of welded Aluminum. The top of the box will consist of a rubber gasket, topped with a ½ inch plexi-glass face that is held down by a metal rim.

When the frame was completed, the team began to work on the arm, which would be our center piece of the ROV. The arm/claw was to be placed on the front of the ROV and to move in a 120 degree pivotal point, but due to some complications with the length of the pneumatic cylinders, the arm can only extend straight forward or point 90 degrees downward.

The water proof housing had to be remade and the team decided that an aluminum box would be the best for dissipating heat. This means we will still be able to have our plexi-glass top and allow other people to view the internal components. The argon lights were also to be kept from last year to be one of our "bells and whistles" on our ROV.

The ROV completes most of the task by the convenient "Swiss-Army-Arm". The Arm can move from a Parallel position with the ROV to a -90° degrees position from the ROV. On the arm are located two magnetically connected hooks that will clamp onto the tow fish and Captain's bell, which will then be towed to the surface. If not, the claw can pick up either item easily. Also attached is a suction nozzle which will allow us to abstract the fluid sample. Located on the right side of the ROV is a tape measure with a hose clamp attached for the measurement of the U-boat. The depth will be measured by a pressure sensor that measures a controlled amount of air pressure and a changing pressure. The final task was solved by a thermometer and a camera pointing at the read out.

Description of at Least One Challenge

The method of capture for the tow fish if fairly simple, located on the arm is a spring-locking hook that is magnetically released from the mid section of the arm. This hook is then placed in front of the ring on the tow fish and dragged backwards. It doesn’t take much force at all to trigger the spring lock on the hook. Once the tow fish is captured, the tether man then pulls the captured item to the surface.

If the first method does not work, we then try our secondary method which is position the tow fish parallel underneath the ROV and grab it with the arm and have the ROV bring it back to the surface.

Explanation of Troubleshooting Techniques

Our technique for trouble shooting problems was trial and error. We designed, built, and rebuilt about every item on the ROV many times. For example, the frame had to be built many times. In its very first stages of development, the frame easily fell apart during transportation, which led to the rebuilding of it many times. Other times, another team member would add something to the frame, causing the structural integrity to loose strength and the ROV would fall apart.

Description of at Least One Lesson Learned or Skill Gained

One lesson that the entire team has been exposed to and still is learning is communication and organizational skills. The instructors believe that a few individuals will never learn this important skill to life.

Discussion of Future Improvements

One improvement that could help next year’s team would be excellent leadership and a keen sense of which priorities should be present.

How ROV's Are Currently Being Used to Explore and Understand our National Marine Sanctuaries

Today, ROVs are not only used by the military, but also by oil companies and research scientists in every day work. The first tethered ROV came into existence in 1953, created by Dimitri Rebikoff. After that, the US Navy advances the technology to help recover lost ordnance or rescue people in sunken underwater vehicles. Now in the 21st Century, the technology in ROVs is very advance. ROVs can be either remote controlled or fully automated (no piloting by a human involved). One of the most well known ROVs is Alvin, a 4 man submersible which has explored some of the deepest sections of the ocean floor. Alvin has also been used in many research missions and aided in the discovery of the Titanic. Currently, Alvin has been deployed with two smaller, efficient ROVs, nicked name, Jake and Elwood. Both ROVs sit inside of a housing, located on the bottom of Alvin and tethered by a fiber optics cable.

Another useful example of ROVs in the real world is the submersibles deployed by the offshore oil companies. They send ROV's down to survey the ocean floor and help plan the proper location for an oil rig. The ROV then can help guide divers and the drilling equipment to the ocean floor with minimal risk involved.

ROVs have a wide and bright future ahead of them and with the advances of technologies; ROVs will soon be capable of completing more difficult task.

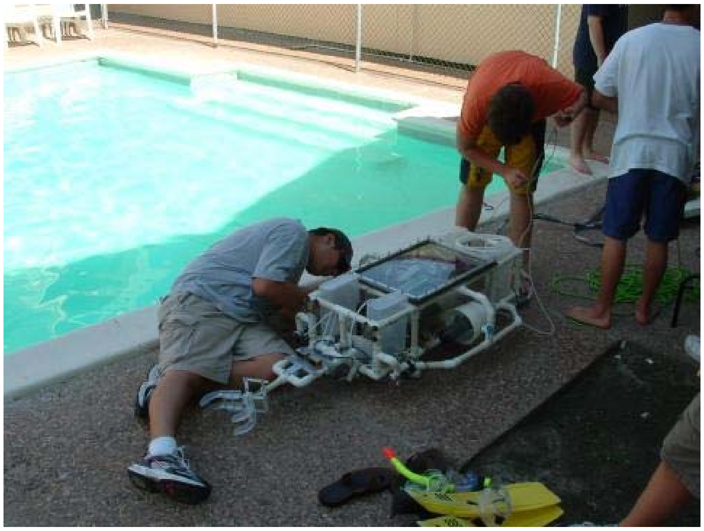
http://www.rov.org/educational/pages/history.html

http://www.marinetech.org/rov_competition/ROVs_in_Sanctuaries_Rev.pdf

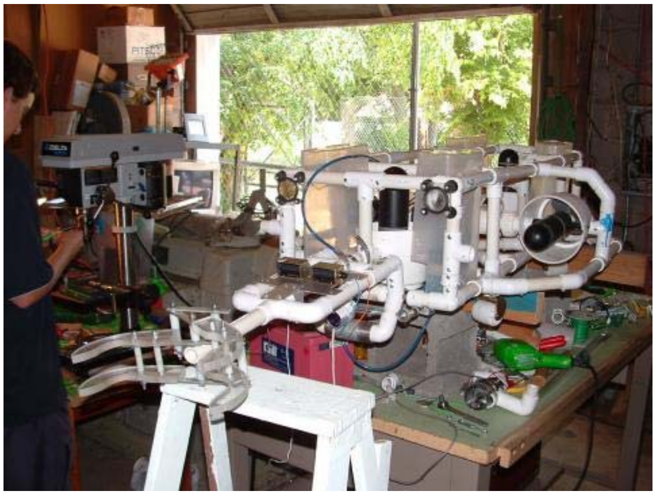
http://www.whoi.edu/marops/vehicles/alvin/

Acknowledgements

Many thanks go to Mr. Pete for helping us on all our electrical problems, and to Mr. Page for his financial support, organizational support and allowing us to work in his warehouse.



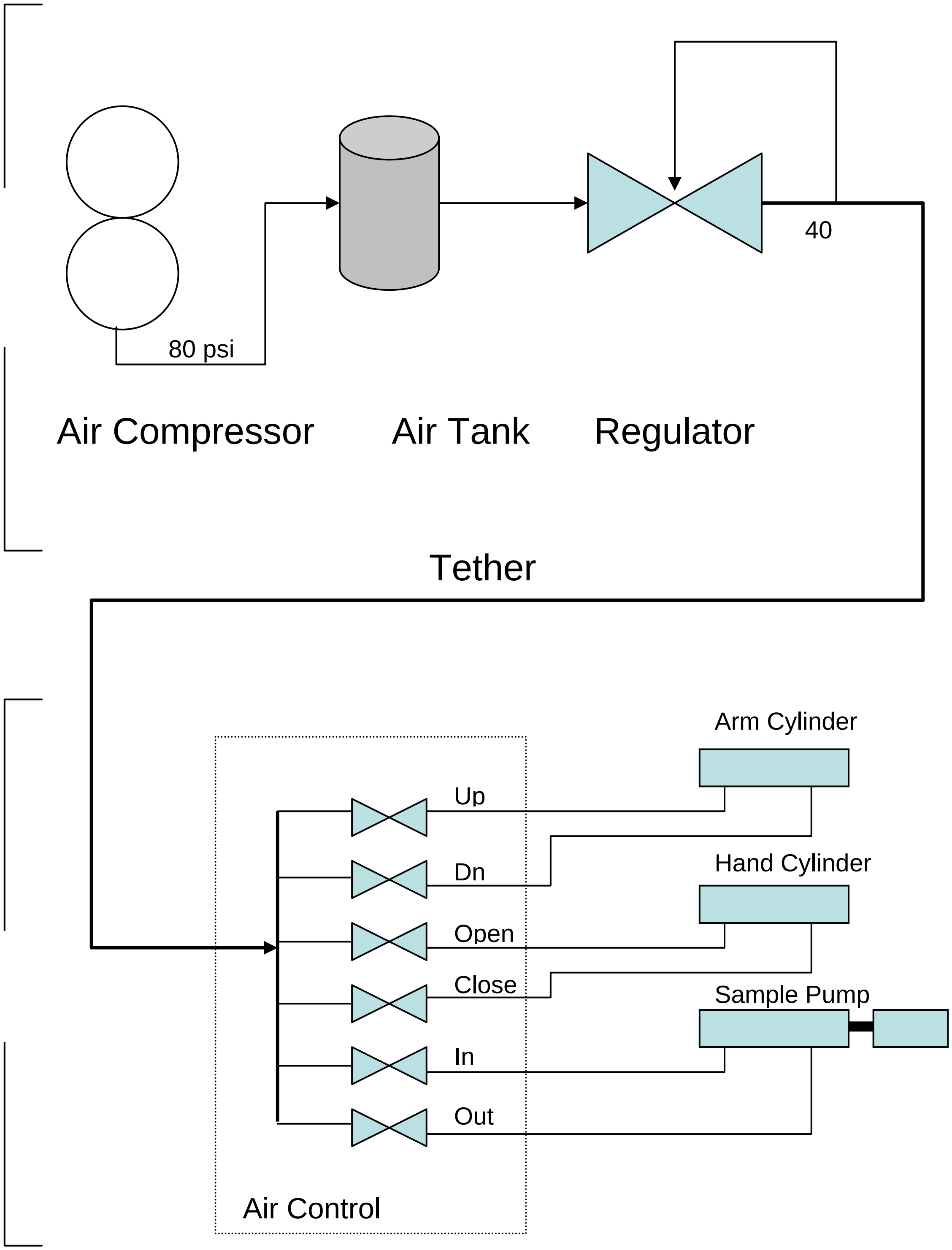
Prepping the ROV for a pool test.



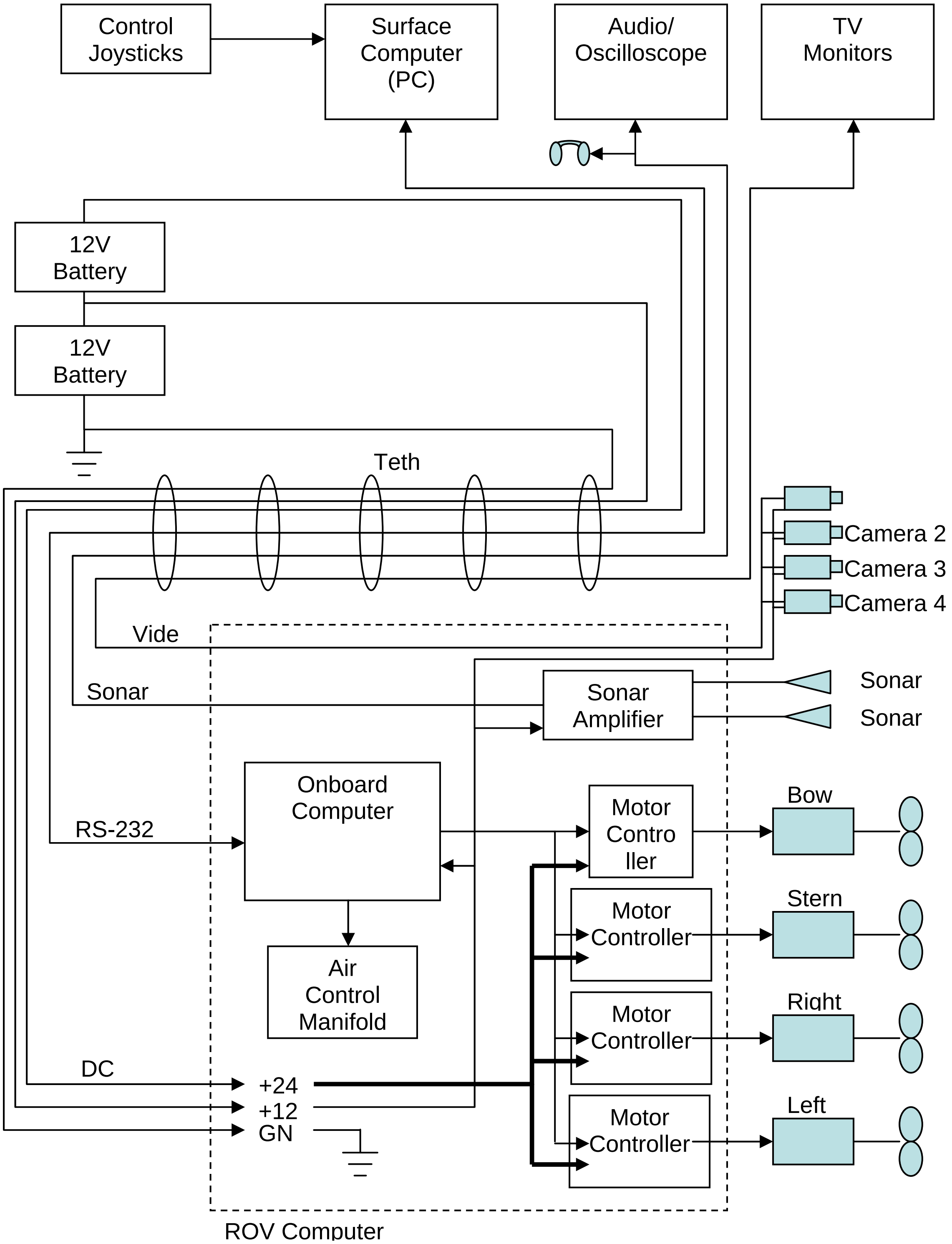
Completed arm

The team and Pet

ROV Pneumatic System



ROV Electrical System



This is an Addendum for the BHS technical report. This is the graphics program for the host computer.

```
/*********************************************************/
/* Master ROV host computer graphics display program       */
/* 0.1 - first functional                               */
/* Programmers:                                         */
/*                                                  */
/*********************************************************/

/**************************************/
/* Define include files               */
/**************************************/

        #include <conio.h>              /* Console specific functions  */
        #include <dos.h>                        /* Dos specific functions      */
        #include <stdio.h>              /* Standard I/O functions      */
        #include <stdlib.h>             /* Standard libraries          */
        #include <math.h>                       /* Standard math functions     */
        #include <graphics.h>           /* Standard graphics functions */
        #include <iostream.h>


/**************************************/
/* Define external variables          */
/**************************************/

extern int analog1;
extern int analog2;

extern float ROV_roll;
extern float ROV_pitch;

extern int graphics_init;

extern float ROV_pressure;
extern float ROV_temp;

extern float calibratedVal[4];

extern int p8020_cmd;

extern int switches;

extern int left_motor_cmd;
extern int right_motor_cmd;
extern int left_motor;
extern int right_motor;
extern int fwd_motor_cmd;
extern int aft_motor_cmd;
extern int fwd_motor;
extern int aft_motor;

extern int p8010_cmd;

extern int p8010_cmd;

extern int left_trans;
extern int right_trans;

extern float ROV_heading;

extern float ROV_motorVCC;
extern float ROV_V121;
extern float ROV_V122;

extern int ROV_frames_good;

extern long receive_error;
extern long good_frames;

/*************************************/
/* Define standard constants           */
/*************************************/

      #define FALSE                  0x00
      #define TRUE            0xFF

      #define AI_x            circx-130      /* x coordinate of upper left */
      #define AI_y            circy-90/* y coordinate of upper left */
      #define AI_width        20      /* width in pixels  */
      #define AI_height       180     /* height in pixels */

      #define RI_x            circx-125      /* x coordinate of upper left */
      #define RI_y            circy+120      /* y coordinate of upper left */
      #define RI_width        150     /* width in pixels  */
      #define RI_height       20      /* height in pixels */

      #define Cx              getmaxx()/2 // x coord of center of circle
      #define Cy              getmaxy()/2 // y coord of center of circle
      #define Cr              50
```

```
#define Rxt             Cx - 100  // x coord of joy top left
#define Ryt             Cy - 100  // y coord of joy top left
#define Rxb             Cx + 100  // x coord of joy bottom right
#define Ryb             Cy + 100  // y coord of joy bottom right
#define r               2         // radius of dot
#define COLOR                 8   // background color


#define Lx              60 // Light indicator X
#define Ly              340       // Light indicator Y

#define Motorx          280     //Motor indicator x
#define Motory          400      //Motor indicator y
#define Motorr          15      //Motor circle radius

#define Ballastx        280     //Top left Ballast x
#define Ballasty        70     //top left Ballast y
#define Ballastr        15      //ballast circle radius

#define R               90      // radius of attitude circle
#define circx           140 //X of attitude circle
#define circy           100 //Y of attitude circle

#define px              475
#define py              120     //PRESSURE
#define pr              60

#define tx              610
#define ty              200     //TEMPERATURE
#define tw              10
#define th              150
#define tr              tw + 5

#define cx              450     //CAPTURE
#define cy              235

#define rad             M_PI/180

#define fx              0     //FRAMAGE
#define fy              280
#define FALSE                 0x00
#define TRUE            0xFF

#define vx            0
#define vy            310
```

```
        #define fanx        0
        #define fany        340

void write_graphics_screen(void)
{
        static int last_roll;
        static int last_pitch[2];
        static int max_width;
        static int max_height;
        int roll;
        int pitch[2];
        int max_pitch;
        int max_roll;
        int magnitude;


        char text1[80];         /* create temporary text buffer */
        int ol[8];              /* create temporary poly fill buffer */

        static float x1, x2, y1, y2, m, lx1, lx2, ly1, ly2;
        static float p, lp, t;

        static float Angle;

        static float i;
        static float q;
        static float j, k, l, n;

        static float lasthead;


        /** BACKGROUND COLOR **/
        setbkcolor(8);


        /*** initialize graphics screen first time ***/

        if (graphics_init == TRUE)
        {

                max_width  = getmaxx();
                max_height = getmaxy();

                /*** print fixed text characters ***/

                outtextxy(270,1,"The Sea Ghost");
                /*** MOTORS ***/
                setcolor(GREEN);
                outtextxy(Motorx + 17, Motory - 1, "Motors");
                circle(Motorx, Motory, Motorr);
                circle(Motorx + 80, Motory, Motorr);

                circle(Motorx + 40, Motory - 35 , Motorr);
                circle(Motorx + 40, Motory + 35 , Motorr);

                //translation
                circle(Motorx - 27, Motory, Motorr - 8);
                circle(Motorx + 107, Motory, Motorr - 8);

                /*** BALLAST ***/
                outtextxy(Ballastx + 17, Ballasty + 30, "Ballast");
                circle(Ballastx, Ballasty , Ballastr);
                circle(Ballastx + 80, Ballasty , Ballastr);
                circle(Ballastx, Ballasty + 60, Ballastr);
                circle(Ballastx + 80 , Ballasty + 60, Ballastr);

                /*** NEONS BOX ***/
                setcolor(WHITE);
                setwritemode(COPY_PUT);
                rectangle(Lx, Ly, Lx + 40, Ly + 17);

                /*** ATTITUDE CIRCLE INDICATOR! ***/

                setcolor(WHITE);
                circle(circx, circy, R+2);
                circle(circx, circy, R+3);
                circle(circx, circy, R+4);
                circle(circx + R-5, circy - R+5, 9);
                ly1 = 0;
                ly2 = 0;
                lx1 = 0;
                lx2 = 0;

                /*** draw attitude indicator box ***/
                setcolor(GREEN);
                settextstyle(0,1,0);
                outtextxy(8,40,"Attitude");
                setcolor(WHITE);
                setwritemode(COPY_PUT);
                rectangle((AI_x-1),(AI_y-1),(AI_x + AI_width+1),(AI_y +
AI_height+1));
```

```
                /*** initialize roll indicator ***/

                last_roll = AI_y + (AI_height / 2);
                setcolor(YELLOW);
                setwritemode(COPY_PUT);
                line (AI_x,last_roll,(AI_x + AI_width),last_roll);

                /*** draw pitch indicator box ***/

                setcolor(WHITE);
                setwritemode(COPY_PUT);
                rectangle((RI_x-1),(RI_y-1),(RI_x + RI_width+1),(RI_y + RI_height+1));

                last_pitch[0] = RI_x + (RI_width / 2);         /* left side "y"  */
                last_pitch[1] = RI_y + (RI_height / 2);        /* right side "y" */

                setcolor(YELLOW);
                setwritemode(COPY_PUT);
                line (last_pitch[0],RI_y,last_pitch[0],RI_y + RI_height);

                /*** draw thermometer**/
                setcolor(WHITE);
                circle(tx, ty, tr);
                rectangle(tx-tw-1,ty-th-1,tx+tw+1,ty);
                setcolor(RED);
                setfillstyle(SOLID_FILL, RED);
                pieslice(tx, ty, 0, 360, tr-1);
                bar(tx-tw+1,ty-tw-5,tx+tw-1,ty);

                /*** PRESSURE ***/
                setcolor(WHITE);
                circle(px, py, pr);
                circle(px, py, 1);
                line(px - pr*cos(M_PI/4), py + pr*sin(M_PI/4), px + 15 - pr*cos(M_PI/4),
120 - 15 + pr*sin(M_PI/4));

                /*** HEADING ***/
                setcolor(WHITE);
                circle(Cx,Cy,Cr+2);
                circle(Cx,Cy,Cr+3);
                line(Cx,Cy+Cr,Cx,Cy+Cr+5);
                line(Cx,Cy-Cr,Cx,Cy-Cr-5);
                line(Cx+Cr,Cy,Cx+Cr+4,Cy);
                line(Cx-Cr,Cy,Cx-Cr-4,Cy);
                lasthead = 0;

                /*** CAPTURE ***/
                rectangle(cx-1,cy-1,cx+57,cy+17);
                rectangle(cx-5,cy-5,cx+62,cy+94);
                j=0;
                k=0;
                l= -1;

                /*** FRAMAGE ***/
                settextstyle(0,0,0);
                setcolor(GREEN);
                outtextxy(fx+75,fy,"Good: ");
                setcolor(RED);
                outtextxy(fx+75,fy+15,"Bad : ");
                setcolor(WHITE);
                outtextxy(fx+5,fy+5, "FRAMES < ");
                rectangle(fx,fy, fx+55,fy+17);

                /*** VOLTAGE ***/
                setcolor(WHITE);
                rectangle(vx,vy, vx+115,vy+17);
                rectangle(vx+120,vy, vx+235,vy+17);
                rectangle(vx+240,vy, vx+355,vy+17);

                outtextxy(vx+5,vy+5, "Motor: ");
                outtextxy(vx+125,vy+5, "8010: ");
                outtextxy(vx+245,vy+5, "8020: ");

                /*** CHECK INDICATORS ***/

/*FAN*/                 rectangle(fanx,fany, fanx+47,fany+17);
                outtextxy(fanx+12,fany+5, "FAN");

/*PRESSURE*/            rectangle(fanx,fany+30, fanx+47,fany+47);
                outtextxy(fanx+5,fany+35, "PRESS");

/*TEMPERATURE*/ rectangle(fanx,fany+60, fanx+47,fany+77);
                outtextxy(fanx+8,fany+65, "TEMP");

/*ANGLE*/       rectangle(fanx,fany+90, fanx+47,fany+107);
                outtextxy(fanx+5,fany+95, "ANGLE");


                /*** spec op ***/
                i=10;
                q=0;

                /*** reset initialization flag ***/

                graphics_init = FALSE;
                settextstyle(0,0,0);


        }

        setcolor(WHITE);
        setwritemode(COPY_PUT);
        if(ROV_frames_good == TRUE)
        {
          setfillstyle(SOLID_FILL,GREEN);
        }
        else
        {
          setfillstyle(SOLID_FILL,RED);
        }
        bar(fx+1,fy+1, fx+54,fy+16);
        outtextxy(fx+5,fy+5, "FRAMES");

        setfillstyle(SOLID_FILL,COLOR);
        bar(fx+115,fy,fx+200,fy+21);
        sprintf(text1, "%4i\0",receive_error);
        outtextxy(fx+130,fy+15,text1);
        sprintf(text1, "%4i\0",good_frames);
        outtextxy(fx+130,fy,text1);


 /*     if(ROV_frames_good == TRUE)  //BIG IF - if bad frames dont draw!
    {  */


        /*** draw line inside attitude indicator box ***/

        /*** erase current roll indicator line */

//      setwritemode(XOR_PUT);
        setcolor(COLOR);
        line (AI_x,last_roll,(AI_x + AI_width),last_roll);

        /*** draw new roll indicator line */

        roll = (int) (ROV_roll * -5.0);// amplify roll value
```

```
max_roll = (AI_height / 2) - 2;
if (roll >= max_roll) roll = max_roll;
if (roll <= -max_roll) roll = -max_roll;

setcolor(YELLOW);
setwritemode(COPY_PUT);
roll = roll + (AI_y + (AI_height / 2));
line (AI_x,roll,(AI_x + AI_width),roll);
last_roll = roll;


/*** erase current pitch indicator line */

setwritemode(XOR_PUT);
line (last_pitch[0],RI_y,last_pitch[0],RI_y+RI_height);

/*** draw new pitch indicator line */

magnitude = (int) (ROV_pitch * -2.0);

max_pitch = (RI_width / 2) - 4;
if (magnitude >= max_pitch) magnitude = max_pitch;
if (magnitude <= -max_pitch) magnitude = -max_pitch;

pitch[0] = RI_x + (RI_width / 2) + magnitude;
pitch[1] = RI_y + (RI_height / 2) - magnitude;

setcolor(YELLOW);
setwritemode(COPY_PUT);
line (pitch[0],RI_y,pitch[0],RI_y + RI_height);
last_pitch[0] = pitch[0];
last_pitch[1] = pitch[1];


/* ERASE ATTITUDE CIRCLE LINE 1*/

setwritemode(XOR_PUT);
line(circx + lx1, circy + ly1, circx + lx2, circy + ly2);

/* DRAW ATTITUDE CIRCLE LINE 1*/

Angle = -1 * ROV_pitch * M_PI / 180;
if(Angle>M_PI/3) Angle = M_PI/3;
if(Angle<-M_PI/3) Angle = -M_PI/3;

m=tan(Angle);
```

```
    if(abs(ROV_roll) > R)
    {
      ROV_roll = R;
      setcolor(RED);
      setfillstyle(SOLID_FILL, RED);
      pieslice(circx + R - 5, circy - R + 5, 0, 360, 8);
    }

    if(ROV_roll < R && ROV_roll > -R)
    {
      setcolor(BLACK);
      setfillstyle(SOLID_FILL, BLACK);
      pieslice(circx + R - 5, circy - R + 5, 0, 360, 8);
    }

    x1 = (-m*ROV_roll + sqrt(-ROV_roll*ROV_roll + R*R + m*m*R*R)) / (1 +
m*m);

    x2 = (-m*ROV_roll - sqrt(-ROV_roll*ROV_roll + R*R + m*m*R*R)) / (1 +
m*m);

    y1 = -(m*x1 + ROV_roll);
    y2 = -(m*x2 + ROV_roll);

    setwritemode(COPY_PUT);
    setcolor(YELLOW);
    line(circx + x1, circy + y1, circx + x2, circy + y2);

    ly1 = y1;
    ly2 = y2;
    lx1 = x1;
    lx2 = x2;


    /*** ERASE PRESSURE GAUGE ***/

    setcolor(COLOR);
    line(px - (pr-2)*cos(M_PI/180*(45-lp)), py + (pr-2)*sin(M_PI/180*(45-lp)), px,
py);

    setfillstyle(SOLID_FILL, COLOR);
    pieslice(px, py, 226-lp, 0, pr-2);
    pieslice(px, py, 224, 360, pr-2);

    /*** DRAW PRESSURE GAUGE ***/
```

```
        p = ROV_pressure * 5;

        if(ROV_pressure > 15.0 && ROV_pressure < 16.2)
        {
          setfillstyle(SOLID_FILL, GREEN);
          setcolor(GREEN);
        }
        else
        {
          setfillstyle(SOLID_FILL, RED);
          setcolor(RED);
        }
        pieslice(px, py, 223, 226-p, pr-2);

        setcolor(YELLOW);
        line(px - (pr-2)*cos(M_PI/180*(45-p)), py + (pr-2)*sin(M_PI/180*(45-p)), px,
py);
        setcolor(WHITE);
        line(px - pr*cos(M_PI/4), py + pr*sin(M_PI/4), px + 15 - pr*cos(M_PI/4), 120 -
15 + pr*sin(M_PI/4));

        lp = p;


                /*** CHECK INDICATORS ***/

/*FAN*/         if(p8010_cmd & 0x40)        setfillstyle(SOLID_FILL,GREEN);
                else setfillstyle(SOLID_FILL,RED);
                bar(fanx+1,fany+1, fanx+46,fany+16);
                outtextxy(fanx+12,fany+5, "FAN");

/*PRESSURE*/    if(ROV_pressure > 15.0 && ROV_pressure < 16.2)
        setfillstyle(SOLID_FILL,GREEN);
                else setfillstyle(SOLID_FILL,RED);
                bar(fanx+1,fany+31, fanx+46,fany+46);
                outtextxy(fanx+5,fany+35, "PRESS");

/*TEMPERATURE*/ if(ROV_temp < 99.9) setfillstyle(SOLID_FILL,GREEN);
                else setfillstyle(SOLID_FILL,RED);
                bar(fanx+1,fany+61, fanx+46,fany+76);
                outtextxy(fanx+8,fany+65, "TEMP");

/*ANGLE*/       if((abs(ROV_roll) < 45.0) && (abs(ROV_pitch) < 45.0))
        setfillstyle(SOLID_FILL,GREEN);
                else setfillstyle(SOLID_FILL,RED);
        bar(fanx+1,fany+91, fanx+46,fany+106);
        outtextxy(fanx+5,fany+95, "ANGLE");


/*** VOLTAGE ***/

if(ROV_motorVCC > 20.0)   setfillstyle(SOLID_FILL,GREEN);
else setfillstyle(SOLID_FILL,RED);
bar(vx+1,vy+1,vx+114,vy+16);
sprintf(text1, "%4.1f\0",ROV_motorVCC);
outtextxy(vx+70,vy+5,text1);
outtextxy(vx+5,vy+5, "Motor V: ");

if(ROV_V121 > 10.0)setfillstyle(SOLID_FILL,GREEN);
else setfillstyle(SOLID_FILL,RED);
bar(vx+121,vy+1,vx+234,vy+16);
sprintf(text1, "%4.1f\0",ROV_V121);
outtextxy(vx+190,vy+5,text1);
outtextxy(vx+125,vy+5, "8010 V: ");

if(ROV_V122 > 10.0)setfillstyle(SOLID_FILL,GREEN);
else setfillstyle(SOLID_FILL,RED);
bar(vx+241,vy+1,vx+354,vy+16);
sprintf(text1, "%4.1f\0",ROV_V122);
outtextxy(vx+310,vy+5,text1);
outtextxy(vx+245,vy+5, "8020 V: ");


/*** DRAW THERMOMETER ***/

t = ROV_temp * 1;

setcolor(COLOR);
setfillstyle(SOLID_FILL, COLOR);
bar(tx - tw, ty - th, tx + tw, ty - t+1);

if(ROV_temp < 95)
{
  setcolor(GREEN);
  setfillstyle(SOLID_FILL, GREEN);
}
else
{
  setcolor(RED);
  setfillstyle(SOLID_FILL, RED);
}
```

```
pieslice(tx, ty, 0, 360, tr-1);
bar(tx - tw, ty - t, tx + tw, ty);


/*** ERASE HEADING ***/

setcolor(COLOR);
setfillstyle(SOLID_FILL, COLOR);
pieslice(Cx,Cy,0,360,Cr);

/*** DRAW HEADING ***/
setwritemode(COPY_PUT);
setcolor(RED);
line(Cx,Cy,Cx+Cr*cos(ROV_heading*rad),Cy+Cr*sin(ROV_heading*rad));
setwritemode(COPY_PUT);
setcolor(WHITE);
line(Cx-Cr*cos(ROV_heading*rad),Cy-Cr*sin(ROV_heading*rad),Cx,Cy);

lasthead = ROV_heading*rad;


/*** LIGHTS ***/

if(p8010_cmd & 0x80)
{
   setfillstyle(SOLID_FILL, 9);
   bar(Lx+1, Ly+1, Lx + 39, Ly + 16);
   setcolor(COLOR);
   outtextxy(Lx+5, Ly+5, "NEON");
}
else
{
   setfillstyle(SOLID_FILL, COLOR);
   bar(Lx+1, Ly+1, Lx + 39, Ly + 16);
   setcolor(9);
   outtextxy(Lx+5, Ly+5, "NEON");
}


         /*** Motors ***/
         /*** Side   ***/

if((switches & 0x10) != 0x10)
{
 //PORT
 if(left_motor_cmd & 0x80)
 {
  setcolor(RED);
  setfillstyle(SOLID_FILL, RED);
  pieslice(Motorx, Motory, 0, 360, 14);
 }
 else
 {
  setcolor(GREEN);
  setfillstyle(SOLID_FILL, GREEN);
  pieslice(Motorx, Motory, 0, 360, 14);
 }

 if(left_motor_cmd == 0x00)
 {
  setcolor(BLACK);
  setfillstyle(SOLID_FILL, BLACK);
  pieslice(Motorx, Motory, 0, 360, 14);
 }


 //STARBOARD
 if(right_motor_cmd & 0x80)
 {
  setcolor(RED);
  setfillstyle(SOLID_FILL, RED);
  pieslice(Motorx + 80, Motory, 0, 360, 14);
 }
 else
 {
  setcolor(GREEN);
  setfillstyle(SOLID_FILL, GREEN);
  pieslice(Motorx + 80, Motory, 0, 360, 14);
 }

 if(right_motor_cmd == 0x00)
 {
  setcolor(BLACK);
  setfillstyle(SOLID_FILL, BLACK);
  pieslice(Motorx + 80, Motory, 0, 360, 14);
 }
}
else
{
  setcolor(BLACK);
  setfillstyle(SOLID_FILL, BLACK);
  pieslice(Motorx + 80, Motory, 0, 360, 14);
  pieslice(Motorx, Motory, 0, 360, 14);
}


        /*** Fwd/Aft ***/

if((switches & 0x40) != 0x40)
{
 //FWD
 if(fwd_motor_cmd & 0x80)
 {
  setcolor(RED);
  setfillstyle(SOLID_FILL, RED);
  pieslice(Motorx + 40 , Motory - 35, 0, 360, 14);
 }
 else
 {
  setcolor(GREEN);
  setfillstyle(SOLID_FILL, GREEN);
  pieslice(Motorx + 40, Motory - 35, 0, 360, 14);
 }

 if(fwd_motor_cmd == 0x00)
 {
  setcolor(BLACK);
  setfillstyle(SOLID_FILL, BLACK);
  pieslice(Motorx + 40, Motory - 35, 0, 360, 14);
 }


 //AFT
 if(aft_motor_cmd & 0x80)
 {
  setcolor(RED);
  setfillstyle(SOLID_FILL, RED);
  pieslice(Motorx + 40, Motory + 35, 0, 360, 14);
 }
 else
 {
  setcolor(GREEN);
  setfillstyle(SOLID_FILL, GREEN);
  pieslice(Motorx + 40, Motory + 35, 0, 360, 14);
 }

 if(aft_motor_cmd == 0x00)
 {
  setcolor(BLACK);
  setfillstyle(SOLID_FILL, BLACK);
  pieslice(Motorx + 40, Motory + 35, 0, 360, 14);
 }

}
else
{
  setcolor(BLACK);
  setfillstyle(SOLID_FILL, BLACK);
  pieslice(Motorx + 40, Motory - 35, 0, 360, 14);
  pieslice(Motorx + 40, Motory + 35, 0, 360, 14);
}

                        /***TRANSLATION***/
if(p8020_cmd & 0x40)
{
  setcolor(GREEN);
  setfillstyle(SOLID_FILL, GREEN);
  pieslice(Motorx - 27, Motory, 0, 360, Motorr - 9);
}
else
{
  setcolor(BLACK);
  setfillstyle(SOLID_FILL, BLACK);
  pieslice(Motorx - 27, Motory, 0, 360, Motorr - 9);
}

if(p8020_cmd & 0x80)
{
  setcolor(GREEN);
  setfillstyle(SOLID_FILL, GREEN);
  pieslice(Motorx + 107, Motory, 0, 360, Motorr - 9);
}
else
{
  setcolor(BLACK);
  setfillstyle(SOLID_FILL, BLACK);
  pieslice(Motorx + 107, Motory, 0, 360, Motorr - 9);
}


                        /***BALLAST***/


                        /*   FLOOD   */
```

```
//top left

if(p8020_cmd & 0x01)
{
setcolor(GREEN);
setfillstyle(SOLID_FILL, GREEN);
pieslice(Ballastx, Ballasty, 0, 360, Ballastr-1);
}
if(p8020_cmd & 0x02)
{
setcolor(RED);
setfillstyle(SOLID_FILL, RED);
pieslice(Ballastx, Ballasty, 0, 360, Ballastr-1);
}
if(!((p8020_cmd & 0x01) || (p8020_cmd & 0x02)))
{
 setcolor(BLACK);
 setfillstyle(SOLID_FILL, BLACK);
 pieslice(Ballastx, Ballasty, 0, 360, Ballastr-1);
}

if(p8020_cmd & 0x04)
{
setcolor(GREEN);
setfillstyle(SOLID_FILL, GREEN);
pieslice(Ballastx + 80, Ballasty , 0, 360, Ballastr-1);
}
if(p8020_cmd & 0x08)
{
setcolor(RED);
setfillstyle(SOLID_FILL, RED);
pieslice(Ballastx + 80, Ballasty , 0, 360, Ballastr-1);
}
if(!((p8020_cmd & 0x04) || (p8020_cmd & 0x08)))
{
 setcolor(BLACK);
 setfillstyle(SOLID_FILL, BLACK);
 pieslice(Ballastx + 80, Ballasty , 0, 360, Ballastr-1);
}

if(p8010_cmd & 0x01)
{
setcolor(GREEN);
setfillstyle(SOLID_FILL, GREEN);
pieslice(Ballastx, Ballasty + 60, 0, 360, Ballastr-1);
}
```

```
if(p8010_cmd & 0x02)
{
setcolor(RED);
setfillstyle(SOLID_FILL, RED);
pieslice(Ballastx, Ballasty + 60, 0, 360, Ballastr-1);
}
if(!((p8010_cmd & 0x01) || (p8010_cmd & 0x02)))
{
 setcolor(BLACK);
 setfillstyle(SOLID_FILL, BLACK);
 pieslice(Ballastx, Ballasty + 60, 0, 360, Ballastr-1);
}


if(p8010_cmd & 0x04)
{
setcolor(GREEN);
setfillstyle(SOLID_FILL, GREEN);
pieslice(Ballastx + 80 , Ballasty + 60 , 0, 360, Ballastr-1);
}
if(p8010_cmd & 0x08)
{
setcolor(RED);
setfillstyle(SOLID_FILL, RED);
pieslice(Ballastx + 80 , Ballasty + 60 , 0, 360, Ballastr-1);
}
if(!((p8010_cmd & 0x04) || (p8010_cmd & 0x08)))
{
 setcolor(BLACK);
 setfillstyle(SOLID_FILL, BLACK);
 pieslice(Ballastx + 80 , Ballasty + 60, 0, 360, Ballastr-1);
}

                                    /*** CAPTURE ***/
if(p8020_cmd & 0x20)
{
    setfillstyle(SOLID_FILL, RED);
    bar(cx, cy, cx + 56, cy + 16);
    setcolor(WHITE);
    outtextxy(cx+1, cy + 5, "CAPTURE");
}
if(p8020_cmd & 0x10)
{
    setfillstyle(SOLID_FILL, BLUE);
    bar(cx, cy, cx + 56, cy + 16);
    setcolor(GREEN);
```

```
            outtextxy(cx+1, cy + 5, "CAPTURE");
        }
        if(!(p8020_cmd & 0x10) && !(p8020_cmd & 0x20))
        {
            setfillstyle(SOLID_FILL, BLACK);
            bar(cx, cy, cx + 56, cy + 16);
            setcolor(RED);
            outtextxy(cx+1, cy + 5, "CAPTURE");
        }

        if((p8020_cmd == 0x20 && j < 61) || (p8020_cmd == 0x10 && j > 0))
        {
          if(p8020_cmd == 0x10) n = -1;
          else n = 1;

          j = j + n;
          j = 2;
          if(j<30)
          {
            k = k - n;
            l = -1;
          }
          else
          {
            k = k + n;
            l=1;
          }
        }

        setcolor(BLACK);
        line(cx+26,cy+60,cx+26+k-l,cy+29+j);
        line(cx+26,cy+60,cx+26+k,cy+30+j);
        line(cx+26,cy+60,cx+26+k+l,cy+31+j);

        setcolor(WHITE);
        line(cx+26,cy+60,cx+26+k,cy+30+j);

                                  /*** TEXT ***/

        setcolor(WHITE);
        setfillstyle(SOLID_FILL, BLUE);

        // write value of ROLL

        ol[0]=20; ol[01]=RI_y+RI_height+20; ol[2]=160; ol[3]=RI_y+RI_height+20;
ol[4]=160; ol[5]=RI_y+RI_height+10; ol[6]=20; ol[7]=RI_y+RI_height+10;
```

```
        fillpoly(4, ol);

        sprintf(text1, "ROV Roll = %+4.1f\0",ROV_pitch);
        outtextxy (22,RI_y+RI_height+12, text1);

        // write value of PITCH

        ol[0]=circx-120; ol[1]=circy+100; ol[2]=circx+20; ol[3]=circy+100;
ol[4]=circx+20; ol[5]=circy+110; ol[6]=circx-120; ol[7]=circy+110;
        fillpoly(4, ol);

        sprintf(text1, "ROV Pitch = %+4.1f\0",ROV_roll);
        outtextxy (22,circy+102, text1);

        // write value of PRESSURE

        ol[0]=px-70; ol[01]=py+73; ol[2]=px + 70; ol[3]=py+73; ol[4]=px + 70;
ol[5]=py+63; ol[6]=px-70; ol[7]=py+63;
        fillpoly(4, ol);

        sprintf(text1, "ROV Pres = %+4.1f\0",ROV_pressure);
        outtextxy (px-68,py+65, text1);

        // write value of TEMPERATURE

        ol[0]=tx-tr-20; ol[1]=ty+tr+20; ol[2]=tx+tr+10; ol[3]=ty+tr+20; ol[4]=tx+tr+10;
ol[5]=ty+tr+10; ol[6]=tx-tr-20; ol[7]=ty+tr+10;
        fillpoly(4, ol);

        sprintf(text1, "%+4.1f\0",ROV_temp);
        outtextxy (tx-tr-15,ty+tr+12, text1);

        //write value of MOTORS

        //left
        ol[0]=Motorx-Motorr-80; ol[1]=Motory+Motorr-10; ol[2]=Motorx+Motorr-60;
ol[3]=Motory+Motorr-10; ol[4]=Motorx+Motorr-60; ol[5]=Motory+Motorr-20;
ol[6]=Motorx-Motorr-80; ol[7]=Motory+Motorr-20;
        fillpoly(4, ol);

        sprintf(text1, "%3i\0", left_motor_cmd & 0x7F);
        outtextxy (Motorx-Motorr-75,Motory+Motorr-18, text1);

        //right
        ol[0]=Motorx-Motorr+140; ol[1]=Motory+Motorr-10;
ol[2]=Motorx+Motorr+160; ol[3]=Motory+Motorr-10; ol[4]=Motorx+Motorr+160;
ol[5]=Motory+Motorr-20; ol[6]=Motorx-Motorr+140; ol[7]=Motory+Motorr-20;
        fillpoly(4, ol);

        sprintf(text1, "%3i\0", right_motor_cmd & 0x7F);
        outtextxy (Motorx-Motorr+145,Motory+Motorr-18, text1);

                        /////////

        //fwd
        ol[0]=Motorx-Motorr+30; ol[1]=Motory+Motorr-70; ol[2]=Motorx+Motorr+50;
ol[3]=Motory+Motorr-70; ol[4]=Motorx+Motorr+50; ol[5]=Motory+Motorr-80;
ol[6]=Motorx-Motorr+30; ol[7]=Motory+Motorr-80;
        fillpoly(4, ol);

        sprintf(text1, "%3i\0", fwd_motor_cmd & 0x7F);
        outtextxy (Motorx-Motorr+35,Motory+Motorr-78, text1);

        //aft
        ol[0]=Motorx-Motorr+30; ol[1]=Motory+Motorr+40; ol[2]=Motorx+Motorr+50;
ol[3]=Motory+Motorr+40; ol[4]=Motorx+Motorr+50; ol[5]=Motory+Motorr+50;
ol[6]=Motorx-Motorr+30; ol[7]=Motory+Motorr+50;
        fillpoly(4, ol);

        sprintf(text1, "%3i\0", aft_motor_cmd & 0x7F);
        outtextxy (Motorx-Motorr+35,Motory+Motorr+42, text1);

//      }//end of BIG IF
                                /*** spec op ***/

        if(i==10) q = 0;
        if(i==0)  q = 1;

        if(q==1)
        i+=1;
        if(q==0)
        i-=1;

        setcolor(COLOR);
        circle(600, 390+i, 10);
        circle(605, 390+i, 1);
        circle(595, 390+i, 2);
        arc(600, 390+i, 225, 315, 7);
        line(600, 400+i, 600, 440+i);
        line(600, 410+i, 620, 410+i+i);
        line(600, 410+i, 585, 405+i+i);
        line(600, 440+i, 615, 470+i);
        line(600, 440+i, 585, 470+i);

        circle(600, 390+i+1, 10);
        circle(605, 390+i+1, 1);
        circle(595, 390+i+1, 2);
        arc(600, 390+i+1, 225, 315, 7);
        line(600, 400+i+1, 600, 440+i+1);
        line(600, 410+i+1, 620, 410+i+i+2);
        line(600, 410+i+1, 585, 405+i+i+2);
        line(600, 440+i+1, 615, 470+i+1);
        line(600, 440+i+1, 585, 470+i+1);

        circle(600, 390+i-1, 10);
        circle(605, 390+i-1, 1);
        circle(595, 390+i-1, 2);
        arc(600, 390+i-1, 225, 315, 7);
        line(600, 400+i-1, 600, 440+i-1);
        line(600, 410+i-1, 620, 410+i+i-2);
        line(600, 410+i-1, 585, 405+i+i-2);
        line(600, 440+i-1, 615, 470+i-1);
        line(600, 440+i-1, 585, 470+i-1);

        //////////////////////////

        setcolor(YELLOW);
        circle(600, 390+i, 10);
        circle(605, 390+i, 1);
        circle(595, 390+i, 2);
        arc(600, 390+i, 225, 315, 7);
        line(600, 400+i, 600, 440+i);
        line(600, 410+i, 620, 410+i+i);
        line(600, 410+i, 585, 405+i+i);
        line(600, 440+i, 615, 470+i);
        line(600, 440+i, 585, 470+i);


        setcolor(YELLOW);
    }
```

This is an Addendum for the BHS technical report. This is the main source code for the Host computer

```
/*******************************************************/
/* Master ROV host computer control program version 0.5    */
/* 0.1 - first functional                                  */
/* 0.2 - motor control; analog readback from ROV           */
/* 0.3 - dual byte commands to/from ROV                    */
/* 0.4 - single joystick read command + operational screen */
/* Programmers:                                            */
/*                                                         */
/*******************************************************/

/*************************************/
/* Define include files              */
/*************************************/

        #include <conio.h>                      /* Console specific functions  */
        #include <dos.h>                        /* Dos specific functions      */
        #include <stdio.h>                      /* Standard I/O functions      */
        #include <math.h>                       /* Standard math functions     */
        #include <graphics.h>                   /* Standard graphics functions */
        #include <bios.h>                /* Standard BIOS functions */

/*************************************/
/* Define external files             */
/*************************************/

extern void write_graphics_screen (void);
extern void write_operational_screen (void);


/*************************************/
/* Define standard constants         */
/*************************************/

        #define VERSION         0x10            /* current version number */
        #define FALSE           0x00
        #define TRUE        0xFF
        #define NOCHAR          0x100

        #define motor_off       0x00            /* minimum value for motor
command */
        #define motor_max       0x7F            /* max value for motor
command */
```

```
        #define debug_view   0x00                   /* debug view = "0" */
        #define op_view              0x01                   /* operational view = "1" */
        #define graphics_view        0x02                   /* graphics view = "2" */

/****************************************/
/* Define Global Variables & Locations */
/****************************************/

        unsigned int analog1;                      /* ROV analog channel 1 - roll sensor
*/
        unsigned int analog2;                      /* ROV analog channel 2 - pitch sensor
*/
        unsigned int analog3;                        /* ROV analog channel 3 - TBD
*/
        unsigned int analog4;                        /* ROV analog channel 4 - TBD
*/
        unsigned int analog5;                        /* ROV analog channel 5 - TBD
*/
        unsigned int analog6;                        /* ROV analog channel 6 - TBD
*/
        unsigned int analog7;                        /* ROV analog channel 7 - TBD
*/
        unsigned int analog8;                        /* ROV analog channel 8 - TBD
*/

        int   ROV_hours;
        int   ROV_minutes;                       /* ROV clock minutes in BCD */
        int   ROV_seconds;                       /* ROV clock seconds in BCD */
        float ROV_pitch;                        /* Calculated / filtered pitch angle   */
        float ROV_roll;                                       /* Calculated / filtered roll
angle   */
        float ROV_temp;
        float ROV_pressure;
        float ROV_heading;
        float ROV_motorVCC;
        float ROV_V121;
        float ROV_V122;
        int   ROV_idle;
        int   ROV_status;
        int   ROV_frames_good;

        unsigned long rawVal[4];                     /* Raw joystick input values */
        float inputVal[4];                           /* Filtered joystick input values */
        float centerVal[4];                          /* Joystick center values */
        float maxVal[4];                     /* Maximum joystick value */
        float minVal[4];                     /* Minimum joystick value */
        float negative_coefficient[4];              /* Multipler for joystick pushed
forward */
        float positive_coefficient[4];            /* Multiplier for joystick pulled back */
        float calibratedVal[4];                       /* Final calibrated values from
joysticks */
        int   switches;                               /* Joystick switches */
        char  switches2[4];                           /* Parallel Input switches */
        char  LEDs;                                   /* Parallel Output LEDs */

        float left_motor;                           /* calculated left motor speed */
        float right_motor;                            /* calculated right motor speed */
        float fwd_motor;                        /* calculated forward motor speed */
        float aft_motor;                       /* calculated aft motor speed */

        int left_motor_cmd;                           /* MSB = direction, 7LSB =
magnitude */
        int right_motor_cmd;                          /* same */
        int fwd_motor_cmd;                            /* same */
        int aft_motor_cmd;                            /* same */
        int p8010_cmd;                          /* two bits per ballast tank */
        int p8020_cmd;                          /* 0-7 TBD */
        int p8030_cmd;                                /* 0=Fwd, 1=aft, 2=cathode light, 3-
7 TBD */
        int misc_cmd;                          /* 0-7 TBD */
        int left_trans;
        int right_trans;

        unsigned char command_buffer[12];             /* set of commands to go to ROV */
        int  response_buffer[20];                 /* info back from ROV */
        long receive_error;                   /* counts reception errors */
        long good_frames;                      /* counts reception success */

        int   quadrant;                     /* joystick: 1=UR, 2=LR, 3=LL, 4=UL */
        int   view_mode;                                      /* 0 = debug, 1 = text, 2 =
graphics */
        int   keycode;                                /* latest keyboard key pressed */
        int   calibration_complete;                   /* calibration complete flag */
        int   error_number;                           /* generalized error flag */
        int   graphics_init;
        long  bios_time;
        long  next_time;
        long  idle_ctr;
        long  min_idle;

        int ballast_mode = 0;                         //ballast switches/keyboard mode
        int led;                                //LED value for write_parallel
/******************************************/
/* Function to set up serial port       */
/* Called with: no value               */
/* Returns: no value                  */
/******************************************/

        #define PORT1                   0x3F8           /* com1 = 3F8, com2 = 2F8, com3 =
3E8, com4 = 2E8 */
        #defineA9KBAUD                  0x0C            /* 9,600 bits per second */
        #defineA19KBAUD 0x06                    /* 19,200 bits per second */
        #defineA38KBAUD 0x03                    /* 38,400 bits per second */


void init_serial (void)
        {
        outportb(PORT1 + 1, 0);                         /* turn off interrupts */
        outportb(PORT1 + 3, 0x80);              /* set DLAB on */
        outportb(PORT1 + 0, A9KBAUD);                   /* set for serial baud rate */
        outportb(PORT1 + 1, 0);                         /* divisor latch hi byte */
        outportb(PORT1 + 3, 0x03);              /* set DLAB off, 8 bits, no parity, 1 stop bit
*/
        outportb(PORT1 + 2, 0x07);              /* set FIFO on, clear rx and tx FIFO */
        outportb(PORT1 + 4, 0x0B);              /* Turn on DTR, RTS, and OUT2 */
        }


/******************************************/
/* Function to write to serial port      */
/* Called with: value to write          */
/* Returns: no value                  */
/******************************************/

void write_serial (int ch)
        {
        outportb(PORT1, ch);
        }


/**********************************************/
/* Function to read character from serial port */
/* Called with: no value                  */
/* Returns: character or NULL                */
/**********************************************/

int read_serial (void)
```

```
        {
        int LSR;
        int ch;

        LSR = inportb(PORT1 + 5);
        if (LSR & 0x01) ch = (inportb(PORT1) & 0xFF);
                else ch = NOCHAR;
        return ch;
        }


/***********************************************/
/* Function to check for serial FIFO empty     */
/* Called with: no value                       */
/* Returns: no value                           */
/***********************************************/

void    check_buffer_empty (void)
        {
        int buf;

        do
        {
        buf = inportb(PORT1 + 5) & 0x20;
        } while (buf == 0);
        }



/***********************************************/
/* Function to clear serial FIFO               */
/* Called with: no value                       */
/* Returns: no value                           */
/***********************************************/

void    empty_receive_buffer (void)
        {
        outport(PORT1 + 2, 0x03);    /* clear receive FIFO */
        }




/*******************************************/
/* Function to write to parallel interface */
/* Called with: value to write             */
/* Returns: no value                    */
/****************************************/



        #define PIO_DATA            0x378           /* LPT1 data out */
        #definePIO_STATUS           0x379           /* status port   */
        #definePIO_CONTROL                  0x37A           /* control port  */


void    write_parallel (char led_out)
        {
        outportb(PIO_DATA, led_out);
        outportb(PIO_CONTROL, 0x01);
        outportb(PIO_CONTROL, 0x00);
        outportb(PIO_CONTROL, 0x01);
        }


/****************************************/
/* Function to write to parallel interface */
/* Called with: value to write            */
/* Returns: no value                    */
/****************************************/



void    read_parallel (void)
        {
        outportb(PIO_DATA, 0x00);
        switches2[0] = (inp(PIO_STATUS) >> 4) ^ 0x07;
        outportb(PIO_DATA, 0x01);
        switches2[1] = (inp(PIO_STATUS) >> 4) ^ 0x07;
        outportb(PIO_DATA, 0x02);
        switches2[2] = (inp(PIO_STATUS) >> 4) ^ 0x07;
        outportb(PIO_DATA, 0x03);
        switches2[3] = (inp(PIO_STATUS) >> 4) ^ 0x07;
        switches2[3] = (switches2[3] ^ 0x0E);
        }



/****************************************/
/* Function to read joystick port         */
/* Called with: no arguments              */
/* Returns: input values in rawVal[i]      */
```

```
/*****************************************/

int     read_joysticks (void)
        {

        int cap_status;
        int timeout;

        outp(0x201,0);                          /* Discharge capacitors */

        rawVal[0] = 0;
        rawVal[1] = 0;
        rawVal[2] = 0;
        rawVal[3] = 0;

        timeout = 0;

        disable();                                              /* No interrupts during
measurement */

        do {

                cap_status = (inp(0x201) & 0x0F);    /* read and mask capacitor status bits
*/

                if (cap_status & 0x01) rawVal[0]++;
                if (cap_status & 0x02) rawVal[1]++;
                if (cap_status & 0x04) rawVal[2]++;
                if (cap_status & 0x08) rawVal[3]++;

                timeout++;

        } while ((cap_status != 0x00) && (timeout < 1000));

        enable();

        return (timeout);
        }



/*****************************************/
/* Function to read joystick switches      */
/* Called with: no arguments               */
/* Returns: value switch byte              */
/*****************************************/
```

```
int read_joystick_switches(void)
   {
         int switch_state;

         switch_state = inp(0x201);              /* Get switch status bits */
         switch_state &= 0x00F0;                         /* Get rid of lower bits  */
         return switch_state;                    /* Return top 4 bits of switch port */
   }


/***********************************************************************
***************************/
/****** FILTERS
************************************************************************
***********/
/***********************************************************************
***************************/

/*******************************************/
/* Function to filter joystick value #1    */
/* Called with: latest instantaneous value */
/* Returns: filtered value                 */
/*******************************************/

float FilterX1(int new_value)
   {
         static int value_array[16];             /* local storage for "last 16 values" */
         static int ptr_val;                     /* pointer to array says where to put next
value */
         int array_sum;                          /* sum of array values */
         float result;                           /* floating point result */
         char i;                                 /* for "for loop" */

         ptr_val &= 0x0F;                                /* mask off all but lower 4 bits */

         value_array[ptr_val] = new_value;       /* save latest instantaneous value into array
*/
         ptr_val++;                                      /* point to next location to store in */

         array_sum = 0;
         for (i=0; i<16; ++i)
            array_sum += value_array[i];

         result = (float) array_sum / 16.0;
```

```
        return result;
   }


/*****************************************/
/* Function to filter joystick value #2    */
/* Called with: latest instantaneous value */
/* Returns: filtered value                 */
/*****************************************/

float FilterY1(unsigned long new_value)
        {
        static int value_array[16];             /* local storage for "last 16 values" */
        static int ptr_val;                     /* pointer to array says where to put next
value */
        int array_sum;                          /* sum of array values */
        float result;                           /* floating point result */
        char i;                                 /* for "for loop" */

        ptr_val &= 0x0F;                                /* mask off all but lower 4 bits */

        value_array[ptr_val] = new_value;       /* save latest instantaneous value into array
*/
        ptr_val++;                                      /* point to next location to store in */

        array_sum = 0;
        for (i=0; i<16; ++i)
           array_sum += value_array[i];

        result = (float) array_sum / 16.0;

        return result;
        }


/*****************************************/
/* Function to filter joystick value #3    */
/* Called with: latest instantaneous value */
/* Returns: filtered value                 */
/*****************************************/

float FilterX2(unsigned long new_value)
        {
        static int value_array[16];             /* local storage for "last 16 values" */
        static int ptr_val;                     /* pointer to array says where to put next
value */
```

```
        int array_sum;                          /* sum of array values */
        float result;                           /* floating point result */
        char i;                                 /* for "for loop" */

        ptr_val &= 0x0F;                                /* mask off all but lower 4 bits */

        value_array[ptr_val] = new_value;       /* save latest instantaneous value into array
*/
        ptr_val++;                                      /* point to next location to store in */

        array_sum = 0;
        for (i=0; i<16; ++i)
           array_sum += value_array[i];

        result = (float) array_sum / 16.0;

        return result;

   }


/*****************************************/
/* Function to filter joystick value #4    */
/* Called with: latest instantaneous value */
/* Returns: filtered value                 */
/*****************************************/

float FilterY2(unsigned long new_value)
        {
        static int value_array[16];             /* local storage for "last 16 values" */
        static int ptr_val;                     /* pointer to array says where to put next
value */
        int array_sum;                          /* sum of array values */
        float result;                           /* floating point result */
        char i;                                 /* for "for loop" */

        ptr_val &= 0x0F;                                /* mask off all but lower 4 bits */

        value_array[ptr_val] = new_value;       /* save latest instantaneous value into array
*/
        ptr_val++;                                      /* point to next location to store in */

        array_sum = 0;
        for (i=0; i<16; ++i)
           array_sum += value_array[i];
        result = (float) array_sum / 16.0;

        return result;
   }


/******************************************/
/* Function to filter pitch input from ROV */
/* Called with: latest instantaneous value */
/* Returns: filtered value              */
/******************************************/

float Filter_Pitch(int new_value)
        {
        static int value_array[8];              /* local storage for "last 16 values" */
        static int ptr_val;                     /* pointer to array says where to put next
value */
        int array_sum;                          /* sum of array values */
        float result;                           /* floating point result */
        char i;                                 /* for "for loop" */

        ptr_val &= 0x07;                                /* mask off all but lower 4 bits */

        value_array[ptr_val] = new_value;       /* save latest instantaneous value into array
*/
        ptr_val++;                                      /* point to next location to store in */

        array_sum = 0;
        for (i=0; i<8; ++i)
           array_sum += value_array[i];

        result = (float) array_sum / 8.0;

        return result;
   }


/******************************************/
/* Function to filter roll input from ROV  */
/* Called with: latest instantaneous value */
/* Returns: filtered value              */
/******************************************/

float Filter_Roll(int new_value)
        {
        static int value_array[8];              /* local storage for "last 16 values" */
        static int ptr_val;                     /* pointer to array says where to put next
value */
        int array_sum;                          /* sum of array values */
        float result;                           /* floating point result */
        char i;                                 /* for "for loop" */

        ptr_val &= 0x07;                                /* mask off all but lower 4 bits */

        value_array[ptr_val] = new_value;       /* save latest instantaneous value into array
*/
        ptr_val++;                                      /* point to next location to store in */

        array_sum = 0;
        for (i=0; i<8; ++i)
           array_sum += value_array[i];

        result = (float) array_sum / 8.0;

        return result;
   }



/******************************************/
/* Function to filter temp input from ROV  */
/* Called with: latest instantaneous value */
/* Returns: filtered value                 */
/******************************************/

float Filter_Temp (int new_value)
        {
        static int value_array[128];            /* local storage for "last 16 values" */
        static int ptr_val;                     /* pointer to array says where to put next
value */
        int array_sum;                          /* sum of array values */
        float result;                           /* floating point result */
        int i;                                  /* for "for loop" */

        ptr_val &= 0x7F;                        /* mask off all but lower 4 bits */

        value_array[ptr_val] = new_value;       /* save latest instantaneous value into array
*/
        ptr_val++;                              /* point to next location to store in */

        array_sum = 0;
        for (i=0; i<128; ++i)
```

```
                array_sum += value_array[i];

            result = (float) array_sum / 128.0;

            return result;
    }

/**********************************************/
/* Function to filter pressure input from ROV  */
/* Called with: latest instantaneous value     */
/* Returns: filtered value                     */
/**********************************************/

float Filter_Pressure (int new_value)
            {
            static int value_array[64];           /* local storage for "last 16 values" */
            static int ptr_val;                   /* pointer to array says where to put next
value */
            int array_sum;                        /* sum of array values */
            float result;                         /* floating point result */
            int i;                                /* for "for loop" */

            ptr_val &= 0x3F;                      /* mask off all but lower 4 bits */

            value_array[ptr_val] = new_value;     /* save latest instantaneous value into array
*/
            ptr_val++;                            /* point to next location to store in */

            array_sum = 0;
            for (i=0; i<64; ++i)
                array_sum += value_array[i];

            result = (float) array_sum / 64.0;

            return result;
    }




/**********************************************/
/* Function to filter heading input from ROV   */
/* Called with: latest instantaneous value     */
/* Returns: filtered value                     */
/**********************************************/
```

```
float Filter_Heading (int new_value)
          {
          static int value_array[8];                /* local storage for "last 16 values" */
          static int ptr_val;                       /* pointer to array says where to put next
value */
          int array_sum;                            /* sum of array values */
          float result;                             /* floating point result */
          int i;                                    /* for "for loop" */

          ptr_val &= 0x7;                           /* mask off all but lower 4 bits */

          value_array[ptr_val] = new_value;    /* save latest instantaneous value into array
*/
          ptr_val++;                                /* point to next location to store in */

          array_sum = 0;
          for (i=0; i<8; ++i)
             array_sum += value_array[i];

          result = (float) array_sum / 8.0;

          return result;
   }




/*************************************************/
/* Function to filter motorvcc input from ROV  */
/* Called with: latest instantaneous value     */
/* Returns: filtered value                     */
/*************************************************/

float Filter_MotorVCC (int new_value)
          {
          static int value_array[16];               /* local storage for "last 16 values" */
          static int ptr_val;                       /* pointer to array says where to put next
value */
          int array_sum;                            /* sum of array values */
          float result;                             /* floating point result */
          int i;                                    /* for "for loop" */

          ptr_val &= 0x0F;                          /* mask off all but lower 4 bits */

          value_array[ptr_val] = new_value;    /* save latest instantaneous value into array
*/
```

```
        ptr_val++;                              /* point to next location to store in */

        array_sum = 0;
        for (i=0; i<16; ++i)
           array_sum += value_array[i];

        result = (float) array_sum / 16.0;

        return result;
   }



/*************************************************************************
****************************/
/****** Graphics
**************************************************************************
**********/
/*************************************************************************
****************************/

/*************************************************/
/* Function to set screen settings for debug    */
/*************************************************/

void set_view_debug (void)
{
        if (view_mode == graphics_view) closegraph();
        view_mode = debug_view;
        clrscr();
}

/************************************************/
/* Function to write debug settings screen     */
/************************************************/

void write_debug_screen (void)
{
        textcolor(WHITE);

        gotoxy(1,1);
        cprintf("Raw Yaw   = %4u", rawVal[0]);
        gotoxy(1,2);
        cprintf("Raw F/R   = %4u", rawVal[1]);
        gotoxy(1,3);
```

```
cprintf("Raw Up/Dn = %4u", rawVal[2]);
gotoxy(1,4);
cprintf("Raw Pitch = %4u", rawVal[3]);

gotoxy(1, 6);
cprintf("Yaw L/R = %6.1f", calibratedVal[0]);
gotoxy(1, 7);
cprintf("Fwd/Rev = %6.1f", calibratedVal[1]);
gotoxy(1, 8);
cprintf("Up/Dn   = %6.1f", calibratedVal[2]);
gotoxy(1, 9);
cprintf("Pitch   = %6.1f", calibratedVal[3]);


gotoxy(20,1);
cprintf("Filt Yaw   = %6.1f", inputVal[0]);
gotoxy(20,2);
cprintf("Filt F/R   = %6.1f", inputVal[1]);
gotoxy(20,3);
cprintf("Filt Up/Dn = %6.1f", inputVal[2]);
gotoxy(20,4);
cprintf("Filt Pitch = %6.1f", inputVal[3]);

gotoxy(20,6);
cprintf("Left Motor = %6.1f", left_motor);
gotoxy(20,7);
cprintf("Rght Motor = %6.1f", right_motor);
gotoxy(20,8);
cprintf("Fwd Motor  = %6.1f", fwd_motor);
gotoxy(20,9);
cprintf("Aft Motor  = %6.1f", aft_motor);

textcolor(LIGHTGREEN);

gotoxy(20,11);
cprintf("Roll  = %+6.1f", ROV_pitch);
gotoxy(20,12);
cprintf("Pitch = %+6.1f", ROV_roll);
gotoxy(20,13);
cprintf("Temp  = %6.1f", ROV_temp);
gotoxy(20,14);
cprintf("Pres  = %6.1f", ROV_pressure);
gotoxy(20,15);
cprintf("Hdg   = %6.0f", ROV_heading);
gotoxy(20,16);
cprintf("24V   = %6.1f", ROV_motorVCC);
```

```
        gotoxy(20,17);
        cprintf("12V#1 = %6.1f", ROV_V121);
        gotoxy(20,18);
        cprintf("12V#2 = %6.1f", ROV_V122);
        gotoxy(20,19);
        cprintf("Idle  = %6.1d", ROV_idle);
        gotoxy(20,20);
        cprintf("Time  = %2x", ROV_hours);
        if (ROV_hours < 10) {gotoxy(28,20); cprintf("0");}

        gotoxy(30,20);
        cprintf(":%2x", ROV_minutes);
        if (ROV_minutes < 10) {gotoxy(31,20); cprintf("0");}

        gotoxy(33,20);
        cprintf(":%2x", ROV_seconds);
        if (ROV_seconds < 10) {gotoxy(34,20); cprintf("0");}

        textcolor(WHITE);

        gotoxy(1,11);
        cprintf("Switches = %3x", switches);
        gotoxy(1,12);
        cprintf("Cal Done = %3x", calibration_complete);
        gotoxy(1,13);
        cprintf("Keycode  = %3x",keycode);
        gotoxy(1,14);
        cprintf("Quadrant = %3x",quadrant);
        gotoxy(1,15);
        cprintf("Err Num  = %3x",error_number);

        textcolor(LIGHTRED);

        gotoxy(1,16);
        cprintf("Bad  = %7u",receive_error);

        textcolor(YELLOW);

        gotoxy(1,17);
        cprintf("Good = %7u",good_frames);
        gotoxy(1,18);
        cprintf("Qual = %7.4f", ((float) good_frames / (float) (good_frames +
receive_error)));
        gotoxy(1,19);
        cprintf("Idle = %7u",min_idle);
```

```
textcolor(LIGHTGRAY);

gotoxy(1,21);
cprintf("F5 joystick cal");
gotoxy(1,22);
cprintf("ESC joystick zero");
gotoxy(1,23);
cprintf("F6 attitude zero");
gotoxy(1,24);
cprintf("Shift-C hdg cal");

gotoxy(20,22);
cprintf("F1 Text Gfx");
gotoxy(20,23);
cprintf("F2 Debug Display");
gotoxy(20,24);
cprintf("F3 Graphics Display");
gotoxy(43,24);
cprintf("Press Capital Q to exit");

textcolor(YELLOW);

gotoxy (43,1);
cprintf("CMDs To ROV");
gotoxy (43,2);
cprintf("Head1   %4X", command_buffer[0]);
gotoxy (43,3);
cprintf("Head2   %4X", command_buffer[1]);
gotoxy (43,4);
cprintf("Head3   %4X", command_buffer[2]);
gotoxy (43,5);
cprintf("L Mot   %4x", command_buffer[3]);
gotoxy (43,6);
cprintf("R Mot   %4x", command_buffer[4]);
gotoxy (43,7);
cprintf("F Mot   %4x", command_buffer[5]);
gotoxy (43,8);
cprintf("A Mot   %4x", command_buffer[6]);
gotoxy (43,9);
cprintf("P8010   %4x", command_buffer[7]);
gotoxy (43,10);
cprintf("P8020   %4x", command_buffer[8]);
gotoxy (43,11);
cprintf("P8030   %4x", command_buffer[9]);
gotoxy (43,12);
```

```
cprintf("Misc    %4x", command_buffer[10]);
gotoxy (43,13);
cprintf("Cksum   %4X", command_buffer[11]);

textcolor(GREEN);

gotoxy (43, 15);
cprintf("PIO1    %4X", switches2[0]);
gotoxy (43, 16);
cprintf("PIO2    %4X", switches2[1]);
gotoxy (43, 17);
cprintf("PIO3    %4X", switches2[2]);
gotoxy (43, 18);
cprintf("PIO4    %4X", switches2[3]);

gotoxy (43, 19);
cprintf("B Mode  %4X", ballast_mode);


textcolor(LIGHTGREEN);

gotoxy (60,1);
cprintf("Rcv From ROV");
gotoxy (60,2);
cprintf("Head1   %4X", response_buffer[0]);
gotoxy (60,3);
cprintf("Head2   %4X", response_buffer[1]);
gotoxy (60,4);
cprintf("Head3   %4X", response_buffer[2]);
gotoxy (60,5);
cprintf("Analog1 %4x", response_buffer[3]);
gotoxy (60,6);
cprintf("Analog2 %4x", response_buffer[4]);
gotoxy (60,7);
cprintf("Analog3 %4x", response_buffer[5]);
gotoxy (60,8);
cprintf("Analog4 %4x", response_buffer[6]);
gotoxy (60,9);
cprintf("Analog5 %4x", response_buffer[7]);
gotoxy (60,10);
cprintf("Analog6 %4x", response_buffer[8]);
gotoxy (60,11);
cprintf("Analog7 %4x", response_buffer[9]);
gotoxy (60,12);
cprintf("Analog8 %4x", response_buffer[10]);
gotoxy (60,13);
```

```
        cprintf("Tilt1   %4x", response_buffer[11]);
        gotoxy (60,14);
        cprintf("Tilt2   %4x", response_buffer[12]);
        gotoxy (60,15);
        cprintf("Roll1   %4x", response_buffer[13]);
        gotoxy (60,16);
        cprintf("Roll2   %4x", response_buffer[14]);
        gotoxy (60,17);
        cprintf("Slo 1   %4X", response_buffer[15]);
        gotoxy (60,18);
        cprintf("Slo 2   %4x", response_buffer[16]);
        gotoxy (60,19);
        cprintf("Dat Sel %4x", response_buffer[17]);
        gotoxy (60,20);
        cprintf("Status  %4x", response_buffer[18]);
        gotoxy (60,21);
        cprintf("EOF     %4X", response_buffer[19]);

        textcolor(WHITE);

}


/************************************************/
/* Function to set screen settings to operate  */
/************************************************/

void set_view_operate (void)
{
        if (view_mode == graphics_view) closegraph();
        view_mode = op_view;
        clrscr();

        textcolor(WHITE);
        gotoxy (10,1);
        cprintf("Galveston College / Ball High Underwater Robotics Team");
        graphics_init = TRUE;
        receive_error = 0;

        min_idle = 60000;

}



/**************************************************/
```

```
/* Function to write grapical info to screen     */
/************************************************/

void set_view_graphics (void)
{
        view_mode = graphics_view;
        graphics_init = TRUE;

        /* initialize the graphics system */

        int graphdriver = DETECT, graphmode, errorcode;
        initgraph(&graphdriver, &graphmode, "..\\bgi");


        /* read result of initialization */
        errorcode = graphresult();

        if (errorcode != grOk)
        {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        getch();
        view_mode = debug_view;
        }

        /** BACKGROUND COLOR **/
        setbkcolor(BLACK);

        min_idle = 60000;
}

/*************************************************************************
***************************/
/****** Subfunctions
**************************************************************************
******/
/*************************************************************************
***************************/

/************************************************/
/* Function to calibrate joysticks             */
/************************************************/

void calibrate_joysticks(void)
   {
        int temp;                                         /* temporary integer */
        float tempf,tempf1,tempf2;                /* temporary floating point */
```

```
unsigned long max[4];                     /* local max value array */
unsigned long min[4];              /* local min value array */
int i;                             /* temporary loop integer */
int timeout;
int cancel_exit;


clrscr();                                 /* clear screen */

centerVal[0] = inputVal[0];    /* store current values as center */
centerVal[1] = inputVal[1];
centerVal[2] = inputVal[2];
centerVal[3] = inputVal[3];

/* get joystick max and min values */

gotoxy(10,9);
cputs("Move each joystick in a complete circle several times");
gotoxy(10,10);
cputs("and press the 1 key on the keyboard");
gotoxy(10,12);
cputs("Press 2 to Cancel");

max[0] = max[1] = max[2] = max[3] = 0;
min[0] = min[1] = min[2] = min[3] = 65535;

/* loop here to collect min and max values; stop when "1" key pressed */

cancel_exit = FALSE;

do
        {

        timeout = read_joysticks();
        if (timeout == 4000) error_number = 2;

        for (i=0; i<4; i++)
                {
                if (rawVal[i] > max[i]) max[i] = rawVal[i];
                if (rawVal[i] < min[i]) min[i] = rawVal[i];
                }

        delay(50);

        if (kbhit()) keycode = getch(); else keycode = 0;
```

```
                if (keycode == 0x32) cancel_exit = TRUE;

                }while ((keycode != 0x31) && (cancel_exit == FALSE));


        /* calculate coefficients necessary to calibrate the joystick */

        if (cancel_exit == FALSE) {

        for (i=0; i<4; i++)
                {
                maxVal[i] = (float) max[i];     /* convert max integers to floating point
numbers */
                minVal[i] = (float) min[i];     /* convert min integers to floating point
numbers */

                /* solve (max - center) * coef = range */
                tempf = maxVal[i] - centerVal[i];
                positive_coefficient[i] = 100.0 / tempf;

                /* solve (center - min) * coef = range */
                tempf = centerVal[i] - minVal[i];
                negative_coefficient[i] = 100.0 / tempf;
                }

        calibration_complete = TRUE;

        }

        set_view_debug();

   }

/*********************************************/
/* Function to zero joysticks                */
/*********************************************/

void zero_joysticks(void)
   {
   float tempf;                 /* temporary floating point */
   int i;

   if (calibration_complete == TRUE)
        {

        centerVal[0] = inputVal[0];    /* store current values as center */
        centerVal[1] = inputVal[1];
        centerVal[2] = inputVal[2];
        centerVal[3] = inputVal[3];

        for (i=0; i<4; i++)                /* calculate new coefficients */
           {

           /* solve (max - center) * coef = range */
           tempf = maxVal[i] - centerVal[i];
           positive_coefficient[i] = 100.0 / tempf;

           /* solve (center - min) * coef = range */
           tempf = centerVal[i] - minVal[i];
           negative_coefficient[i] = 100.0 / tempf;
           }
        }

   }



/************************************************************************
***************************/
/****** MAIN
*************************************************************************
**************/
/************************************************************************
***************************/


void main ()
{
        int i;                                          /* loop counter */
        int ch;                                                 /* read-from-keyboard
character */
        float tempf;                                            /* temporary floating point */
        unsigned int timeout;                                           /* loop timer */
        int checksum;                                           /* for calculating checksum
of command buffer */
        int error;
        int x_tilt;
        int y_tilt;
        static float raw_pitch;
        static float raw_roll;
        static float pitch_zero;
        static float roll_zero;
        float raw_temp;
        float raw_pres;
        float raw_heading;
        float raw_volts;

        int raw_status;
        int heading;
        int data_good;
        int slow_data1;
        int slow_data2;
        int data_sel;

        double magnitude;                                               /* temp value */
        double alpha;                                           /* temp value */
        double yaw;                                                     /* yaw left / right
value */
        double translate_FR;                                    /* translate fwd/back */
        double pitch;                                           /* pitch up /down */
        double translate_UD;                                    /* translate up/dn */


        calibration_complete = FALSE;
        init_serial();                                          /* Configure serial comm
port for ROV */
        clrscr();                                               /* Clear the screen */
        error_number = 0;                                       /* No known errors */
        receive_error, good_frames = 0;
        bios_time = biostime(0,0);
        next_time = bios_time + 1;
        min_idle  = 60000;
        set_view_graphics();

        /***************** start main program loop here
*****************************/

        do
        {

                /* get and filter latest joystick position */

                timeout = read_joysticks();
                if (timeout == 4000) error_number = 1;

                inputVal[0] = FilterX1(rawVal[0]);   /* Read and Filter X1 */
                inputVal[1] = FilterY1(rawVal[1]);   /* Read and Filter Y1 */
                inputVal[2] = FilterX2(rawVal[2]);   /* Read and Filter X2 */
                inputVal[3] = FilterY2(rawVal[3]);   /* Read and Filter Y2 */


                /**** get latest joystick switch positions ****/

                switches = read_joystick_switches();            /* Get switch settings */


                /**** convert filtered joystick input into calibrated joystick position ****/

                for (i=0; i<4; i++)
                {
                        tempf = inputVal[i] - centerVal[i];
                        if (tempf >= 0.0) calibratedVal[i] = tempf * positive_coefficient[i];
                                else calibratedVal[i] = tempf * negative_coefficient[i];
                }

                /**** convert calibrated joystick#1 position into commands for left and
right motors ****/

                yaw = (double) calibratedVal[0];
                translate_FR = (double) (calibratedVal[1] * -1.0);

                /*** set deadband for joystick center */

                if (fabs(yaw) < 5.0) yaw = 0.0;
                if (fabs(translate_FR) < 5.0) translate_FR = 0.0;

                magnitude = sqrt ((yaw * yaw) + (translate_FR * translate_FR));

                if ((yaw >= 0.0) && (translate_FR >= 0.0))            /* quadrant 1 */
                {
                        quadrant = 1;
                        alpha = asin(yaw / magnitude);

                        left_motor = magnitude;
                        right_motor = magnitude * cos(2.0 * alpha);
                }

                else if ((yaw >= 0.0) && (translate_FR < 0.0))                  /* quadrant 2
*/
                {
                        quadrant = 2;
                        alpha = asin((translate_FR * -1.0) / magnitude);

                        left_motor = magnitude * cos(2.0 * alpha);
                        right_motor = -1.0 * magnitude;
                }

                else if ((yaw < 0.0) && (translate_FR < 0.0))                   /* quadrant 3
*/
                {
                        quadrant = 3;
                        alpha = asin((yaw * -1.0) / magnitude);

                        left_motor = -1.0 * magnitude;
                        right_motor = -1.0 * magnitude * cos(2.0 * alpha);
                }

                else if ((yaw < 0.0) && (translate_FR >= 0.0))                  /* quadrant 4
*/
                {
                        quadrant = 4;
                        alpha = asin((yaw * -1.0) / magnitude);

                        left_motor = magnitude * cos(2.0 * alpha);
                        right_motor = magnitude;
                }

                /**** set 'deadband' motor tracking ****/

                if (fabs(right_motor - left_motor) < 4.0)
                        right_motor = left_motor;

                /**** convert motor values into digital commands for ROV ****/

                right_motor_cmd = (int) fabs(right_motor * 1.2);
                if (right_motor_cmd > motor_max) right_motor_cmd = motor_max;
                if (right_motor < 0.0) right_motor_cmd = (right_motor_cmd | 0x80);

                left_motor_cmd  = (int) fabs(left_motor * 1.2);
                if (left_motor_cmd > motor_max) left_motor_cmd = motor_max;
                if (left_motor < 0.0) left_motor_cmd = (left_motor_cmd | 0x80);


                /* force motor commands OFF when needed */

                if ((switches & 0x10) == 0x10)   /* off if joystick trigger not pressed */
                {
                right_motor_cmd = motor_off;
                left_motor_cmd  = motor_off;
                }
                if (calibration_complete == FALSE)  /* off when joystick not calibrated */
                {
                right_motor_cmd = motor_off;
                left_motor_cmd  = motor_off;
                }


                /**** convert calibrated joystick#2 position into commands for forward
and aft motors ****/

                fwd_motor = (-1.0 * calibratedVal[2]) + calibratedVal[3];
                aft_motor = (-1.0 * calibratedVal[2]) - calibratedVal[3];


                /**** set zero and deadband for pitch / tranlate ****/

                if ((fabs(fwd_motor) < 10.0) || (fabs(aft_motor) < 10.0))
                        fwd_motor = aft_motor = 0.0;
                if (fabs(calibratedVal[3]) < 10.0) fwd_motor = aft_motor;


                /**** convert motor values into digital commands for ROV ****/

                fwd_motor_cmd = (int) fabs(fwd_motor);
                if (fwd_motor_cmd > motor_max) fwd_motor_cmd = motor_max;
                if (fwd_motor < 0.0) fwd_motor_cmd = (fwd_motor_cmd | 0x80);

                aft_motor_cmd  = (int) fabs(aft_motor);
                if (aft_motor_cmd > motor_max) aft_motor_cmd = motor_max;
                if (aft_motor < 0.0) aft_motor_cmd = (aft_motor_cmd | 0x80);


                /* force motor commands OFF when needed */

                if ((switches & 0x40) == 0x40)    /* off if joystick trigger not pressed */
                {
                fwd_motor_cmd = motor_off;
                aft_motor_cmd = motor_off;
                }

                if (calibration_complete == FALSE)  /* off when joystick not calibrated */
                {
                fwd_motor_cmd = motor_off;
                aft_motor_cmd = motor_off;
                }
                /**** read keyboard and other computer switch inputs inputs ****/

                if (kbhit()) keycode = getch(); else keycode = 0;

                misc_cmd = 0;

                /**** read parallel input board ****/

                read_parallel();


                // press "F5" to calibrate joysticks
                if ((keycode == 0x3F) && (view_mode == debug_view))
calibrate_joysticks();

                // press "esc" to center joysticks
                if (keycode == 0x1B) zero_joysticks();

                // press "F6" to zero pitch and roll
                if (keycode == 0x40) {pitch_zero = raw_pitch; roll_zero = raw_roll;}

                // press "F1" to view text-based graphics screen
                if (keycode == 0x3B) set_view_operate();

                // press "F2" to view debug screen
                if (keycode == 0x3C) set_view_debug();

                // press "F3" to view line-based graphics
                if (keycode == 0x3D) set_view_graphics();

                //press "F7" to set keyboard/switches ballast mode
                if (keycode == 0x41) ballast_mode++;

                // press "c" to send calibrate compass command to ROV
                if (keycode == 0x43) misc_cmd = (misc_cmd | 0x80);

                // press "i" to reset minimum host idle calculation
                if (keycode == 0x69) min_idle = 0xFFFF;

                // press "Shift-i" to reset host good and bad receive frame count
                if (keycode == 0x49) {good_frames = 0; receive_error = 0;}

                // press "t" to reset clock in ROV
                if (keycode == 0x74) misc_cmd = (misc_cmd | 0x40);
```

```
//use keyboard/switch mode for ballast tanks

if(ballast_mode%2 == 1) //keyboard mode
{

// press "l" to toggle neon light on and off
if (keycode == 0x6c) p8010_cmd = (p8010_cmd ^ 0x80);

// press "s" to toggle capture down
if (keycode == 0x73)
{
  if(p8020_cmd == 0x10) p8020_cmd = (p8020_cmd ^ 0x10);
  p8020_cmd = (p8020_cmd ^ 0x20);
}
// press "S" to toggle capture up
if (keycode == 0x53)
{
  if(p8020_cmd == 0x20) p8020_cmd = (p8020_cmd ^ 0x20);
  p8020_cmd = (p8020_cmd ^ 0x10);
}

// press "t" for LF ballast tank Blow
if (keycode == 0x74) p8020_cmd = (p8020_cmd ^ 0x01);

// press "T" for LF ballast tank Flood
if (keycode == 0x54) p8020_cmd = (p8020_cmd ^ 0x02);

// press "u" for RF ballast tank Blow
if (keycode == 0x75) p8020_cmd = (p8020_cmd ^ 0x04);

// press "U" for RF ballast tank Flood
if (keycode == 0x55) p8020_cmd = (p8020_cmd ^ 0x08);

// press "b" for LR ballast tank Blow
if (keycode == 0x62) p8010_cmd = (p8010_cmd ^ 0x01);

// press "B" for LR ballast tank Flood
if (keycode == 0x42) p8010_cmd = (p8010_cmd ^ 0x02);

// press "m" for RF ballast tank Blow
if (keycode == 0x6d) p8010_cmd = (p8010_cmd ^ 0x04);
```

```
// press "M" for RR ballast tank Flood
if (keycode == 0x4d) p8010_cmd = (p8010_cmd ^ 0x08);

// "y"
if (keycode == 0x79)
{
  p8020_cmd = (p8020_cmd ^ 0x01);
  p8020_cmd = (p8020_cmd ^ 0x04);
}

// "Y"
if (keycode == 0x59)
{
  p8020_cmd = (p8020_cmd ^ 0x02);
  p8020_cmd = (p8020_cmd ^ 0x08);
}

// "g"
if (keycode == 0x67)
{
  p8010_cmd = (p8010_cmd ^ 0x01);
  p8020_cmd = (p8020_cmd ^ 0x01);
}

// "G"
if (keycode == 0x47)
{
  p8010_cmd = (p8010_cmd ^ 0x02);
  p8020_cmd = (p8020_cmd ^ 0x02);
}

// "h"
if (keycode == 0x68)
{
  p8010_cmd = (p8010_cmd ^ 0x01);
  p8010_cmd = (p8010_cmd ^ 0x04);
  p8020_cmd = (p8020_cmd ^ 0x01);
  p8020_cmd = (p8020_cmd ^ 0x04);
}

// "H"
if (keycode == 0x48)
{
  p8010_cmd = (p8010_cmd ^ 0x02);
  p8010_cmd = (p8010_cmd ^ 0x08);
  p8020_cmd = (p8020_cmd ^ 0x02);
  p8020_cmd = (p8020_cmd ^ 0x08);
}

// "j"
if (keycode == 0x6a)
{
  p8010_cmd = (p8010_cmd ^ 0x04);
  p8020_cmd = (p8020_cmd ^ 0x04);
}

// "J"
if (keycode == 0x4a)
{
  p8010_cmd = (p8010_cmd ^ 0x08);
  p8020_cmd = (p8020_cmd ^ 0x08);
}

// "n"
if (keycode == 0x6e)
{
  p8010_cmd = (p8010_cmd ^ 0x01);
  p8010_cmd = (p8010_cmd ^ 0x04);
}

// "N"
if (keycode == 0x4e)
{
  p8010_cmd = (p8010_cmd ^ 0x02);
  p8010_cmd = (p8010_cmd ^ 0x08);
}

//"4" and "6" to left and right translation

if(keycode == 0x34)
{
  p8020_cmd = (p8020_cmd | 0x40);
  p8020_cmd = p8020_cmd & 0xBF;
}
if(keycode == 0x36)
{
  p8020_cmd = (p8020_cmd | 0x80);
  p8020_cmd = p8020_cmd & 0x7F;
}
if(keycode == 0x35)
{
  p8020_cmd = p8020_cmd & 0x3F;
```

```
}

}




if(ballast_mode%2 == 0)                    //switches mode
{
  //Left TRANSLATION
  if (switches2[2] & 0x04) p8020_cmd = p8020_cmd | 0x80;
  else p8020_cmd = p8020_cmd & 0x7F;
  //Right TRANSLATION
  if (switches2[2] & 0x08) p8020_cmd = p8020_cmd | 0x40;
  else p8020_cmd = p8020_cmd & 0xBF;

  // toggle NEON LIGHT on and off
  if (switches2[3] & 0x02) p8010_cmd = p8010_cmd | 0x80;
  else p8010_cmd = p8010_cmd & 0x7F;

  // toggle MISC 1 - COMP BOX PRESSURIZER
  if (switches2[3] & 0x04) p8010_cmd = p8010_cmd | 0x20;
  else p8010_cmd = p8010_cmd & 0xDF;

  // toggle MISC 2 - FAN
  if (switches2[3] & 0x08) p8010_cmd = p8010_cmd | 0x40;
  else p8010_cmd = p8010_cmd & 0xBF;

  // toggle CAPTURE DOWN
  if (switches2[2] & 0x02) p8020_cmd = p8020_cmd | 0x20;
  else p8020_cmd = p8020_cmd & 0xDF;
  // toggle CAPTURE UP
  if (switches2[2] & 0x01) p8020_cmd = p8020_cmd | 0x10;
  else p8020_cmd = p8020_cmd & 0xEF;


  //LF Blow
  if (switches2[0] & 0x01) p8020_cmd = p8020_cmd | 0x01;
  else    p8020_cmd = p8020_cmd & 0xFE;

  //LF ballast tank Flood
  if (switches2[0] & 0x02) p8020_cmd = p8020_cmd | 0x02;
  else    p8020_cmd = p8020_cmd & 0xFD;

  //RF ballast tank Blow
  if (switches2[0] & 0x04) p8020_cmd = p8020_cmd | 0x04;
```

```
                  else    p8020_cmd = p8020_cmd & 0xFB;

                  //RF ballast tank Flood
                  if (switches2[0] & 0x08) p8020_cmd = p8020_cmd | 0x08;
                  else    p8020_cmd = p8020_cmd & 0xF7;

                  //LR ballast tank Blow
                  if (switches2[1] & 0x01) p8010_cmd = p8010_cmd | 0x01;
                  else    p8010_cmd = p8010_cmd & 0xFE;

                  //LR ballast tank Flood
                  if (switches2[1] & 0x02) p8010_cmd = p8010_cmd | 0x02;
                  else    p8010_cmd = p8010_cmd & 0xFD;

                  //RF ballast tank Blow
                  if (switches2[1] & 0x04) p8010_cmd = p8010_cmd | 0x04;
                  else    p8010_cmd = p8010_cmd & 0xFB;

                  //RR ballast tank Flood
                  if (switches2[1] & 0x08) p8010_cmd = p8010_cmd | 0x08;
                  else    p8010_cmd = p8010_cmd & 0xF7;


                  //BLOW ALL
                  if (switches2[3] & 0x01)
                  {
                    p8020_cmd = p8020_cmd | 0x05;
                    p8010_cmd = p8010_cmd | 0x05;
                  }

                }

                // press "f" or "F" for all ballast off
                if ((keycode == 0x66 || keycode == 0x46) || (switches2[0] == 0x00 &&
switches2[1] == 0x00 && !(switches2[3] & 0x01)))
                {
                  p8010_cmd = p8010_cmd & 0xF0;
                  p8020_cmd = p8020_cmd & 0xF0;
                  p8010_cmd = p8010_cmd | 0x10;
                //  j=0;
                }
                /*if(j<20)
                {
                  p8010_cmd = p8010_cmd | 0x10;
                  j++;
                } */
```

```
        else
        {
          p8010_cmd = p8010_cmd & 0xEF;
        }


        /**** setup list of commands to send to ROV ****/

        command_buffer[0]  = 0xAA;                  /* setup header */
        command_buffer[1]  = 0x55;
        command_buffer[2]  = 0xFF;

        command_buffer[3]  = left_motor_cmd;
        command_buffer[4]  = right_motor_cmd;
        command_buffer[5]  = fwd_motor_cmd;
        command_buffer[6]  = aft_motor_cmd;

        command_buffer[7]  = p8010_cmd;
        command_buffer[8]  = p8020_cmd;
        command_buffer[9]  = p8030_cmd;
        command_buffer[10] = misc_cmd; misc_cmd = 0;

        checksum = 0;                          /* calculate checksum for
error detection */
        for (i=3; i<11; i++)
            {
            checksum = checksum + command_buffer[i];
            }

        command_buffer[11] = (checksum | 0x80);


        /**** send commands to ROV ****/


        for (i=0; i<12; i++)
            {
            write_serial ((char) command_buffer[i]);
            }


        /*** computer screen & wait for ROV to receive commands and reply
***/


        if (view_mode == op_view)
```

```
{
  delay(20);
  write_operational_screen();
}
if (view_mode == debug_view)
{
  delay(20);
  write_debug_screen();
}
if (view_mode == graphics_view) write_graphics_screen();


/*** clear response buffer ***/

for (i=0; i<20; i++)
            {
            response_buffer[i] = 0;
            }


/**** retrieve response buffer from ROV ****/

timeout = 0;
do {
            ch = read_serial();
            timeout++;
} while ((ch != 0xAA) && (timeout < 600));

if (timeout == 600) response_buffer[0] = 0;
            else response_buffer[0] = ch;

for (i=1; i<20; i++) {
            timeout = 0;
            do {
                        ch = read_serial();
                        timeout++;
            } while ((ch == NOCHAR) && (timeout < 1600));

            if (timeout < 1600) response_buffer[i] = ch;
                        else response_buffer[i] = 0;
}


/*** check receive buffer format for errors and save it ***/

if ((response_buffer[0] == 0xAA) &&
            (response_buffer[1] == 0x55) &&
                (response_buffer[2] == 0xFF))
                    //(response_buffer[19] == 0xFF))

            {
            good_frames++;
            data_good = TRUE;
            ROV_frames_good = TRUE;

            analog1 = response_buffer[3];
            analog2 = response_buffer[4];
            analog3 = response_buffer[5];
            analog4 = response_buffer[6];
            analog5 = response_buffer[7];
            analog6 = response_buffer[8];
            analog7 = response_buffer[9];
            analog8 = response_buffer[10];

            x_tilt  = ((256 * response_buffer[11]) + response_buffer[12]);
            y_tilt  = ((256 * response_buffer[13]) + response_buffer[14]);

            slow_data1 = response_buffer[15];
            slow_data2 = response_buffer[16];
            data_sel  = response_buffer[17];

            raw_status= response_buffer[18];

            }

            else {receive_error++; data_good = FALSE; ROV_frames_good =
FALSE;}

        /*** process incoming information ***/

        if (data_good == TRUE) {


        // process data received every frame

            raw_pres     = Filter_Pressure(analog1);
            ROV_pressure = raw_pres * .1324;

            raw_temp  = Filter_Temp(analog3);
            ROV_temp  = (3.75 * raw_temp) - 158.75;
            if (ROV_temp < 60.0) ROV_temp = 60.0;
        raw_pitch = Filter_Pitch(x_tilt);
        ROV_pitch = ((raw_pitch - pitch_zero) / 40.0);

        raw_roll  = Filter_Roll(y_tilt);
        ROV_roll  = ((raw_roll - roll_zero) / 40.0);

        raw_volts = Filter_MotorVCC(analog6);
        ROV_motorVCC = (raw_volts * 0.098);

        ROV_V121  = ((float) analog7 * 0.057);
        ROV_V122  = ((float) analog8 * 0.057);

        ROV_status = raw_status;


// process slow data 0 - ROV Heading

        if (data_sel == 0) {

        heading = ((256 * (slow_data1 & 0x01)) + slow_data2);
        ROV_heading = Filter_Heading(heading);
        }


// process slow data 1 - ROV Time

        if (data_sel == 1) {
        ROV_minutes = slow_data1;
        ROV_seconds = slow_data2;
        }


// process slow data 2 - ROV idle time

        if (data_sel == 2) {
        ROV_idle = ((256 * slow_data1) + slow_data2);
        }


// process slow data 3 - ROV hours

        if (data_sel == 3) {
        ROV_hours = slow_data2;
        }

}
```

```
        /**** LEDs ****/

        led = 0x00;
        if((switches & 0x40) != 0x40) led = led + 0x0C;
        if((switches & 0x10) != 0x10) led = led + 0x03;
        if((abs(ROV_roll) > 45.0) || (abs(ROV_pitch) > 45.0)) led = led + 0x10;
        if(ROV_pressure > 16.0) led = led + 0x20;
        if(ROV_temp >= 95.0) led = led + 0x40;
        if(ROV_status == 0x80) led = led + 0x80;
        write_parallel(~led);




        /**** synchronize delay to match PC hardware ***/

        idle_ctr = 0;
        do
        {
        bios_time = biostime(0,0);
        idle_ctr++;
        } while (bios_time < next_time);
        next_time = bios_time + 1;
        if (idle_ctr < min_idle) min_idle = idle_ctr;


    } while(keycode != 0x51);
}
```