Robot Code
&
Code Documentation

Group 9 – Project 3 – 30 April 2003
Joshua Page, Tim Stevens, Jangho Yoon

Introduction:

The goal of Project 3 was to implement some kind of path planning protocol in conjunction with the CMU Camera to locate orange blocks, and take the blocks to locations marked by black tape. Also, maneuvering within the arena was a dynamic object, a "blue robot" which could be disabled by our robot. The arena could also contain blocks and drop off locations that were not marked in the array of coordinates given at program initialization. Extra credit would be given to discovering and manipulating these hidden objects.

Initial Design:

The initial design called for 4 distinct procedures:

- *Hunter/Killer Procedure*: This procedure is tasked with identifying, tracking, closing with, and disabling the dynamic blue robot.
- *Navigate to Nearest Procedure*: This procedure is responsible for the path planning of project 3. It finds the nearest block or drop off location and maneuvers the robot to that destination point.
- *Disorient to Reorient Procedure*: In chasing the dynamic blue robot, or through accumulation of error through routine movement, it is very likely that after a certain amount of time, the robot will become disoriented. This procedure is tasked with properly reorienting the robot after it becomes lost.
- *Notice Hidden Locations & Blocks Procedure*: This procedure keeps an "eye" out for hidden or unmarked blocks and drop-off locations.

The appealing thing about this project is that it would require, at most, 2 processes. One process is the main thread, which all 4 of the distinct procedures would operate in. The other process is a simple read and update of the encoders and IR sensors. See Figure 1 for a graphical representation of this design.

The difficult thing about this project is its size and complexity. Due to the complexity of this project, with its inordinate amount of special cases, and the short amount of time allocated to this project, the only procedure that was able to be completely written and tested was the *"Navigate to Nearest"* Procedure. The other procedures are all important but it was felt that the primary mission was to secure and deliver the blocks.

Actual Implementation:

The actual implementation follows the initial design, but the only four modules able to be completed in time for the demonstration was the "*Navigate to Nearest*" procedure, the *"Command"* module, the "*Update Sensor Data"* module, and the *"Basic Motor Commands"* module, all of which are fairly self explanatory. See Figure 2 for a complete listing of the modules and all their methods.

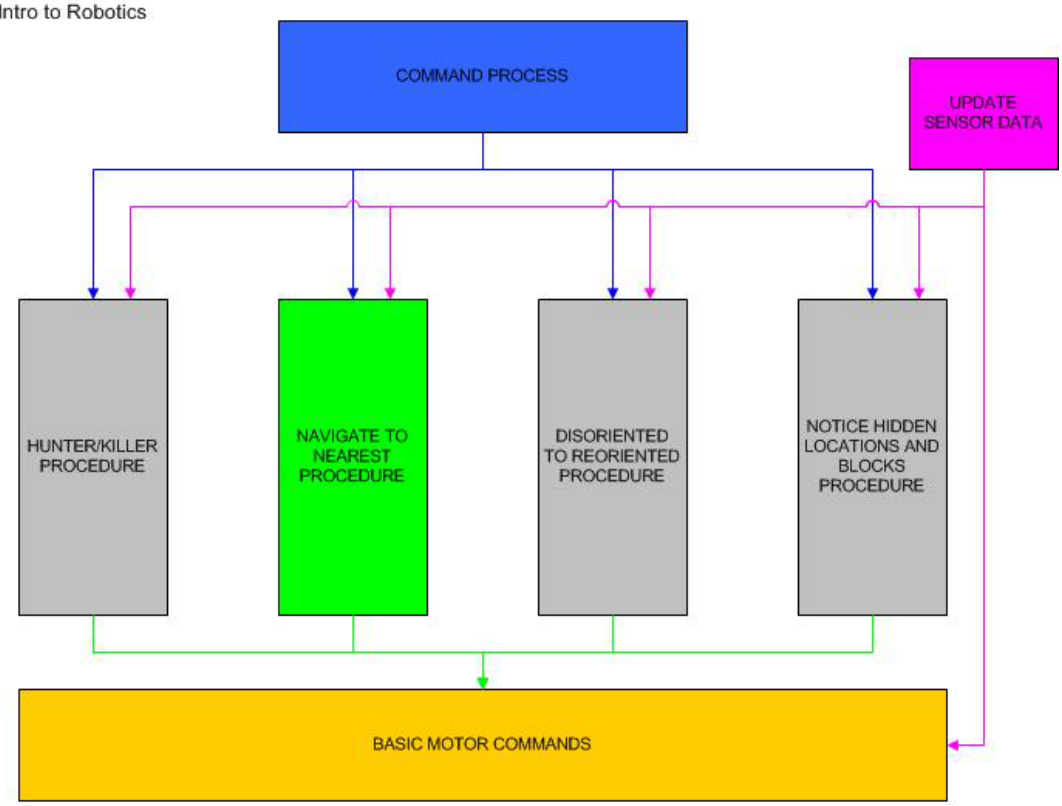
Operational Flow:

The program executes in the following manner. After initializing the camera, and other initialization functions, the main method calls "*Navigate_to_Nearest(int mode)*." If mode is set to "0," we are navigating to the nearest block, if "1" we are navigating to the nearest drop off location. Inside "*Navigate_to_Nearest(int mode)*," "*find_nearest(int mode)*" is called. This method finds the nearest block, or drop off location (depending on mode), and sets the destination equal to that coordinate. Then the distance to this coordinate from the current coordinates is computed. The method *Orient(float x_dist, float y_dist)* is called next. This method uses the current orientation and the location of the destination coordinate to determine if the robot is facing in the correct direction or needs to execute a 180 degree turn. After this method returns, the first leg of movement is completed by calling *movement(float x_dist, float y_dist)*. After the first movement is completed, the robot executes a turn (if needed) by calling *execute_turn(float x_dist, float y_dist)*. Then the second leg of movement is completed by another call to *movement(float x_dist, float y_dist)*. After this movement is completed, the robot is now in the vicinity of the destination coordinate, and can begin to look for the destination. This is the province of the *search(int mode)*. *search(int mode)* locates the block by means of the CMU camera, and implements the necessary motor commands to secure it. When looking for the drop off locations, *search(int mode)* calls *align_to_tape()* to locate and orient the robot to the drop off location. After securing the block, or dropping off the block at the drop off location, *Navigate_to_Nearest(int mode)* ends, and the main method calls *Navigate_to_Nearest(int mode)* again, with the opposite command to what was execute previously. So if the last call was to secure a block, this call will be to drop the block off at a drop off location, and vice versa. For a graphical description, please see Figure 3.

Conclusion:

During testing, the robot was able to accomplish this task fairly reliably. The problems arose if the block was not sufficiently close to its reported position, or if a location was falsely reported, contact with walls occurred, etc. In short, the software contained few provisions for special cases. This is because of the complexity of this project, and the limited time allocated to complete the project. The basic code described here was 14 pages by itself, without concern for special cases or the other 3 procedures to be implemented. But that does not mean this project was a failure. The code written here, while not complete, is a very stable and good base for a continuation to this project.

Original Software Design
Project 3
CS 4970 Intro to Robotics



This project originally had four main procedures/behaviors.

The "Hunter/Killer" Procedure enacted protocols where the robot sought out and attempted to disable the blue robot.
The "Navigate to Nearest" Procedure enacted protocols that planned a path to the nearest block or the nearest taped location.
The "Disoriented to Reoriented" Procedure enacted protocols that would guide the robot to a corner or known taped location so that the robot knew for certain its current position and/or orientation.
The "Notice Hidden Locations and Blocks" Procedure would attempt to discover and manipulate "hidden" blocks or taped locations.

Due to time constraints, the only Procedure fully implemented was the "Navigate to Nearest" Procedure.

FIGURE 1.

METHOD DIRECTORY

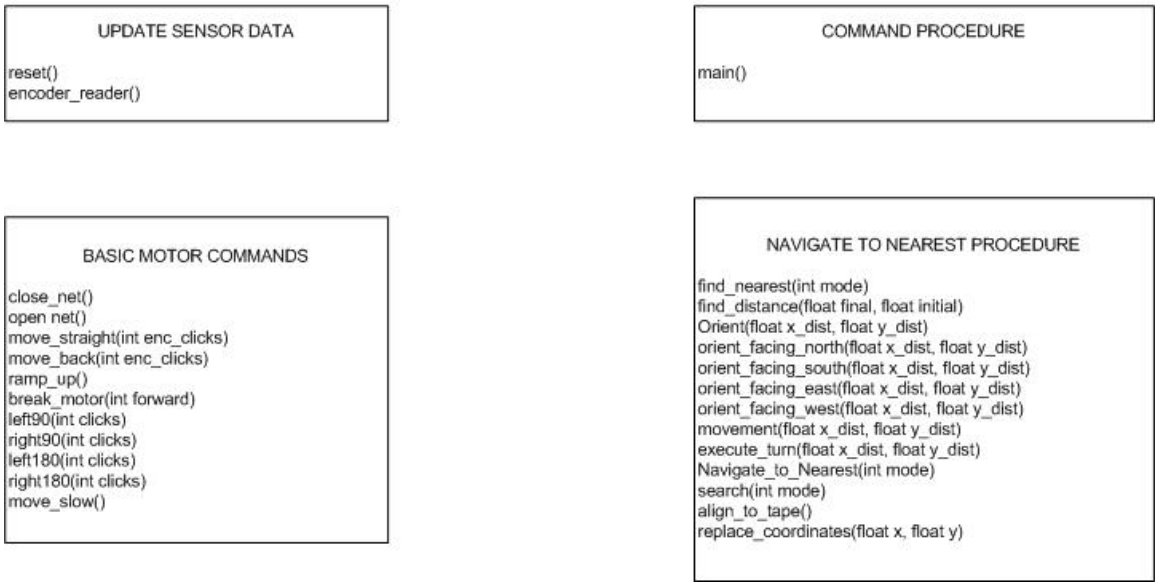


FIGURE 2

PROJECT 3: BASIC FLOW

Project 3 executes in a manner that is very similar to the chart depicted below.

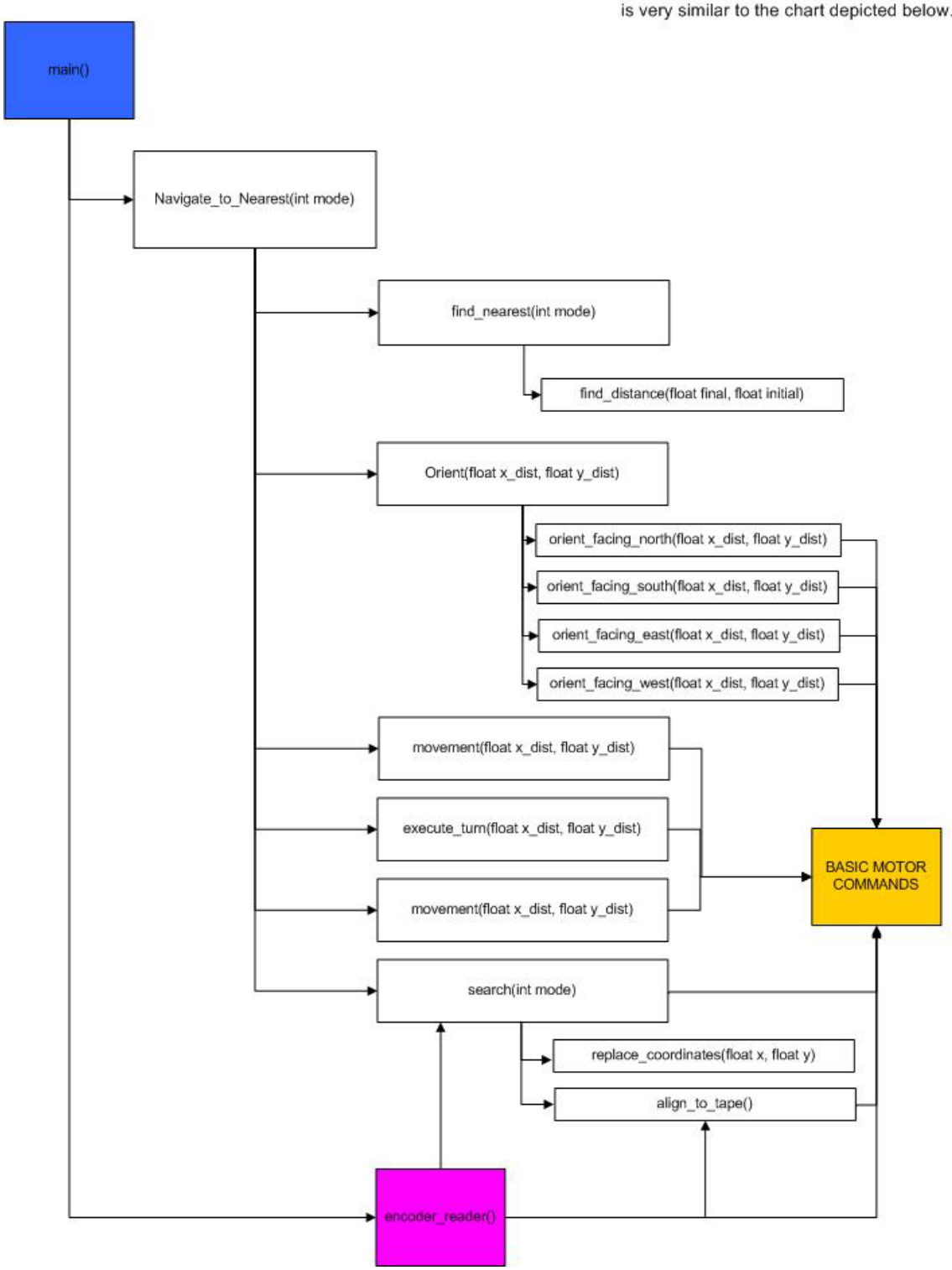


FIGURE 3

```
/*METHODS LEFT TO IMPLEMENT:

    24 APRIL:
        the stuff for finding the black location tape, and what to do if you don't find the location
        needs to be fleshed out.  right now, the robot will look for the black tape until it finds it.
        that's why the main is so simple right now.  if you succeed in capturing the block, take it to
        the nearest location, otherwise, go after the next location.  need to finesse this if possible.
    ***SOME KIND OF "SEARCH & DESTROY" PROCESS FOR THE BLUE ROBOT***

    ***A PROCESS FOR NOTICING AND REGISTERING UNLISTED ORANGE BLOCKS AND UNLISTED
       BLACK LOCATION AREAS***

    ***ATTACHING SENSORS AND DEVELOPING A PROCESS FOR "REORIENTING" THE ROBOT IF IT
       BECOMES LOST.  PROBABLY USING TOUCH AND/OR RANGE FINDERS AND THE CORNERS OF
       THE ARENA, OR ORIENTING TO THE TAPED LOCATIONS***
*/




#use "cmucamlib.ic"
/*RIGHT MOT0R, LEFT MOTOR, AND GRIPPER MOTOR*/
#define RIGHT_MOTOR 0
#define LEFT_MOTOR 1
#define NET_MOTOR 2

/*LEFT ENCODER, AND RIGHT ENCODER*/
#define LEFT_ENCODER 0
#define RIGHT_ENCODER 1

#define LEFT_IR 2
#define RIGHT_IR 3
#define IR_THRESHOLD 100

/*DRIVE POWER FOR THE MOTORS*/
#define POWER 40

/*SERVO_STR SETS THE POWERED CASTOR FORWARD, SERVO_LFT IS FOR LEFT TURNS, SERVO_RGT IS FOR RIGHT TURNS*/
#define SERVO_STR 2023
#define SERVO_LFT 3700
#define SERVO_RGT 300

/*350 encoder clicks for right turns, 345 encoder clicks for left turns,
  720 encoder clicks for a left 180, 720 encoder clicks for a right 180,
  540 clicks to travel 1 foot/1 coordinate.*/
#define RTURN_CLICKS 175
#define LTURN_CLICKS 173
#define L180_CLICKS 360
#define R180_CLICKS 360
#define CONV_CLICKS 580
#define POUNCE_CLICKS 810

int ORIENTATION;             //Which direction the robot is facing 0:N, 1:E, 2:S, 3:W
int left_encod, right_encod; //Global variables for the left and right encoders
int left_power, right_power; //Global variables for the left and right motor power
int left_ir, right_ir;

/** COORDINATES 1 - 16 INDICATE LOCATIONS OF ORANGE BLOCKS ** COORDINATES 17 - 20 ARE DESTINATIONS ** COORDINATE
21 IS THE STARTING LOCATION **/
/*|1      |2      |3      |4      |5        |6        |7        |8        |9        |10       |11       |12
|13       |14       |15       |16       |1      |2      |3      |4      |1      |*/
persistent float coordinate_array[42]={7.0,3.0,2.0,3.5,6.0,4.5,6.0,6.0,1.0,7.0,1.0,9.5,6.5,10.0,7.5,10.0,-1.0,-
1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,-1.0,1.0,1.0,6.0,3.0,2.0,11.0,-1.0,-
1.0,4.0,5.5};
//Current x position & Current y position
float curr_posX;
float curr_posY;

//Destination x position & Destination y position
float des_posX;
float des_posY;

/*****************************************************************/
/*This method searches the coordinate_array and finds the nearest box location
if mode is 0, or the nearest destination location if the mode is 1*/

void find_nearest(int mode)
```

```
{
    int i=0;
    float tempx = 0.0;
    float tempy = 0.0;
    float distance = 0.0;
    float x = 0.0;
    float y = 0.0;
    float sqrx = 0.0;
    float sqry= 0.0;
    float sum = 0.0;

    float min_distance = 100.0;
    float minx = 0.0;
    float miny = 0.0;

    //find block
    if(mode == 0)
      {
        while(i<32)
          {
            if(coordinate_array[i] != -1.0)
              {
                tempx = coordinate_array[i];
                tempy = coordinate_array[i+1];

                x = (tempx - curr_posX);
                y = (tempy - curr_posY);
                sqrx = x*x;
                sqry = y*y;
                sum = sqrx + sqry;
                distance = sqrt(sum);
                if(distance < min_distance)
                  {
                    min_distance = distance;
                    minx = coordinate_array[i];
                    miny = coordinate_array[i+1];
                }//end if
            }//end if
            i = i + 2;
        }//end while
    }//end if
    //find destination
    else
      {
        for(i=32; i<40; i=i+2)
          {
            if(coordinate_array[i] != -1.0)
              {
                tempx = coordinate_array[i];
                tempy = coordinate_array[i+1];
                x=tempx - curr_posX;
                y=tempy - curr_posY;
                x = x*x;
                y = y*y;
                sum = x + y;
                distance = sqrt(sum);
                if(distance < min_distance)
                  {
                    min_distance = distance;
                    minx = coordinate_array[i];
                    miny = coordinate_array[i+1];
                }//end if
            }//end if
        }//end for
    }//end else

    des_posX = minx;
    des_posY = miny;
}//end method find_nearest

/*****************************************************************/
/*find_distance returns the distance between two floats.*/

float find_distance(float final, float initial)
{
    return final - initial;
} //end method find_distance

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

```
/*The robot is facing north.  If the destination is south of the robot,
  execute a 180 degree turn.  Convert the distance to positive
  numbers if it is negative.*/

void orient_facing_north(float x_dist, float y_dist)
{
    if(x_dist < 0.0)
      {
        x_dist = x_dist*1.0;
        return;
    }//end if
    else
      right180(R180_CLICKS);
}//end method orient_facing_north

/*****************************************************************/
/*The robot is facing south.  If the destination is north of the robot,
  execute a 180 degree turn.  Convert the distance to positive
  numbers if it is negative.*/

void orient_facing_south(float x_dist, float y_dist)
{
    if(x_dist < 0.0)
      {
        x_dist = x_dist*1.0;
        right180(R180_CLICKS);
    }//end if
    else
      return;
}//end method orient_facing_south

/*****************************************************************/
/*The robot is facing east.  If the destination is west of the robot,
  execute a 180 degree turn.  Convert the distance to positive
  numbers if it is negative.*/

void orient_facing_east(float x_dist, float y_dist)
{
    if(y_dist < 0.0)
      {
        y_dist = y_dist*1.0;
        return;
    }//end if
    else
      left180(L180_CLICKS);
}//end method orient_facing_east

/*****************************************************************/
/*The robot is facing west.  If the destination is east of the robot,
  execute a 180 degree turn.  Convert the distance to positive
  numbers if it is negative.*/

void orient_facing_west(float x_dist, float y_dist)
{
    if(y_dist < 0.0)
      {
        y_dist = y_dist*1.0;
        left180(L180_CLICKS);
    }//end if
    else
      return;
}//end method orient_facing_west

/*****************************************************************/
/*Determine what direction the robot is facing, then decide if the robot
  needs to execute a 180 turn to head towards its destination */

void Orient(float x_dist, float y_dist)
{
    if(ORIENTATION == 0) //FACING NORTH
      orient_facing_north(x_dist, y_dist);
    else
      if(ORIENTATION == 1) //FACING EAST
        orient_facing_east(x_dist, y_dist);
      else
        if(ORIENTATION == 2) //FACING SOUTH
          orient_facing_south(x_dist, y_dist);
        else                 //FACING WEST
          orient_facing_west(x_dist, y_dist);
```

```
} //end method Orient

/*****************************************************************/
/*Execute the movement schemes.*/

void movement(float x_dist, float y_dist)
{
    float encoder_distance = 0.0;
    int enc_dist = 0;
    float encoder_conv = (float) CONV_CLICKS;

    if((ORIENTATION == 0) || (ORIENTATION == 2))
      {
        encoder_distance = encoder_conv * x_dist;
        enc_dist = (int) encoder_distance;
        printf("orient %d enc: %d\n", ORIENTATION, enc_dist);
        ramp_up();
        move_straight(enc_dist);
        break_motor(1);
        ao();
        curr_posX = curr_posX + x_dist;
        return;
    }//end if
    else
      {
        encoder_distance = encoder_conv * y_dist;
        enc_dist = (int) encoder_distance;
        printf("orient %d enc: %d\n", ORIENTATION, enc_dist);
        ramp_up();
        move_straight(enc_dist);
        break_motor(1);
        ao();
        curr_posY = curr_posY + y_dist;
        return;
    }//end else
}//end method first_movement

void execute_turn(float x_dist, float y_dist)
{
    /*if facing north, and the y_dist is negative,
      we need to turn right, otherwise turn left.*/
    if(ORIENTATION == 0)
      {
        if(y_dist < 0.0)
          {
            y_dist = y_dist * -1.0;
            right90(RTURN_CLICKS);
        }//end if
        else
          left90(LTURN_CLICKS);
    }//end if
    else
      /*if facing east, and the x_dist is negative,
      we need to turn left, otherwise turn right.*/
      if(ORIENTATION == 1)
        {
          if(x_dist < 0.0)
            {
              x_dist = x_dist * -1.0;
              left90(LTURN_CLICKS);
          }//end if
          else
            right90(RTURN_CLICKS);
      }//end if
      /*if facing south, and the y_dist is negative,
      we need to turn left, otherwise turn right.*/
      else
        if(ORIENTATION == 2)
          {
            if(y_dist < 0.0)
              {
                y_dist = y_dist * -1.0;
                left90(LTURN_CLICKS);
            }//end if
            else
              right90(RTURN_CLICKS);
        }//end if
        /*if facing west, and the x_dist is negative,
      we need to turn right, otherwise turn left.*/
```

```
        else
          {
            if(x_dist < 0.0)
              {
                x_dist = x_dist * -1.0;
                right90(RTURN_CLICKS);
            }//end if
            else
              left90(LTURN_CLICKS);
        }//end else
}//end method execute_turn

/*****************************************************************/
/*Determine the nearest box (if mode is 0) or nearest location
(if mode is 1).  Then compute the distance to that point, orient the robot,
and then proceed to that point.*/

int Navigate_to_Nearest(int mode)
{

    int success;
    float diff;
    float x_dist;
    float y_dist;

    find_nearest(mode);

    x_dist = find_distance(des_posX, curr_posX);
    y_dist = find_distance(des_posY, curr_posY);

    printf("O: %d dist: %f, %f \n",ORIENTATION, x_dist, y_dist);
    Orient(x_dist, y_dist);

    movement(x_dist, y_dist);

    if((ORIENTATION == 1)||(ORIENTATION == 3))
      {
        diff = curr_posX - des_posX;
        if((diff >= .5)||(diff <= -.5))
          {
            execute_turn(x_dist, y_dist);
        }//end if
      }//end if

    else
      {
        diff = curr_posY - des_posY;
        if((diff >= .5)||(diff <= -.5))
          {
            execute_turn(x_dist, y_dist);
        }//end if
    }//end else

    movement(x_dist, y_dist);
    success = search(mode);
    return success;
}//end method navigate_to_nearest

/*****************************************************************/
/*Close the gripper arms*/

void close_net()
{
    motor(NET_MOTOR, -200);
    sleep(1.2);
    ao();
}//end method close_net

/*****************************************************************/
/*Return the gripper arms to the open rear state.*/

void open_net()
{
    motor(NET_MOTOR, 200);
    sleep(1.2);
    ao();
}//end method open_net()

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

```
/*This method is called after any movement has been completed.
It resets the encoders and the corresponding global variables.*/

void reset()
{
    reset_encoder(0);
    reset_encoder(1);
    left_encod = 0;
    right_encod = 0;
    left_ir = 0;
    right_ir = 0;
}//end method reset

/*****************************************************************/
/*encoder_reader is a process that continually reads and updates the
  encoders.*/

void encoder_reader()
{
    int temp_ir=0;
    left_encod=0;
    right_encod=0;
    left_ir=0;
    right_ir=0;

    while(1)
      {
        left_encod = read_encoder(LEFT_ENCODER);
        right_encod = read_encoder(RIGHT_ENCODER);
        temp_ir = analog(LEFT_IR);
        if(temp_ir > left_ir)
          left_ir = temp_ir;
        temp_ir = analog(RIGHT_IR);
        if(temp_ir > right_ir)
          right_ir = temp_ir;
    }//end while
}//end process encoder_reader

/*****************************************************************/
/*move_straight moves the robot forward enc_clicks number of encoder
  clicks.*/

void move_straight(int enc_clicks)
{
    int l_speed, r_speed;
    int switch = 0;

    l_speed=r_speed=POWER;

    reset();

    servo0 = SERVO_STR;
    sleep(0.5);

    ramp_up();
    sleep(0.1);
    /*based off of dr. miller's straight line code, this one alternates
    both decreasing and increasing the motor power, alternately.*/
    while((left_encod < enc_clicks)&&(right_encod < enc_clicks))
      {
        if(left_encod<right_encod)
          {
            if (switch == 0)
              {
                r_speed=(8*POWER)/10;
                switch = 1;
            }//end if
            else
              {
                l_speed = (12*POWER)/10;
                switch = 0;
            }//end else
        }//end if
        else
          {
            if(left_encod>right_encod)
              {
                if (switch == 0)
                  {
```

```
                    l_speed=(8*POWER)/10;
                    switch = 1;
                }//end if
                else
                  {
                    r_speed = (12*POWER)/10;
                    switch = 0;
                }//end else
            }//end if
        }//end else
        motor(LEFT_MOTOR, l_speed);
        motor(RIGHT_MOTOR, r_speed);
    }//end while
    break_motor(1);
    ao();
}//end method move_straight

void move_back(int enc_clicks)
{
    int l_speed, r_speed;
    int switch = 0;

    l_speed=r_speed=POWER;

    reset();

    servo0 = SERVO_STR;
    sleep(0.5);

    ramp_up();
    sleep(0.1);
    /*based off of dr. miller's straight line code, this one alternates
    both decreasing and increasing the motor power, alternately.*/
    while((left_encod < enc_clicks)&&(right_encod < enc_clicks))
      {
        if(left_encod<right_encod)
          {
            if (switch == 0)
              {
                r_speed=(8*POWER)/10;
                switch = 1;
            }//end if
            else
              {
                l_speed = (12*POWER)/10;
                switch = 0;
            }//end else
        }//end if
        else
          {
            if(left_encod>right_encod)
              {
                if (switch == 0)
                  {
                    l_speed=(8*POWER)/10;
                    switch = 1;
                }//end if
                else
                  {
                    r_speed = (12*POWER)/10;
                    switch = 0;
                }//end else
            }//end if
        }//end else
        motor(LEFT_MOTOR, l_speed*-1);
        motor(RIGHT_MOTOR, r_speed*-1);
    }//end while
    break_motor(0);
    ao();
}//end method move_straight
/*****************************************************************/
/*ramp up is a precursor to forward movement, hopefully it gets the motors
  in a somewhat similary position before taking off.*/

void ramp_up()
{
    reset();
    motor(RIGHT_MOTOR, 10);
    motor(LEFT_MOTOR, 10);
```

```
    sleep(0.05);
}//end method ramp_up

/*****************************************************************/
/*break_motor is a method that breaks forward progress if forward is 1,
or backward progress if forward is 0.*/

void break_motor(int forward)
{
    reset();
    if (forward == 1)
      {
        motor(LEFT_MOTOR, -20);
        motor(RIGHT_MOTOR, -20);
    }//end if
    else
      {
        motor(LEFT_MOTOR, 20);
        motor(RIGHT_MOTOR, 20);
    }//end else
    sleep(0.1);
    ao();
}//end method break_motor

/*****************************************************************/
//90 degree turn to the left.

void left90(int clicks)
{
    int avg = 0;
    reset();
    servo0 = SERVO_LFT;
    sleep(0.05);
    while(avg < clicks)
      {
        motor(LEFT_MOTOR, -40);
        motor(RIGHT_MOTOR, 40);
        avg = (right_encod + left_encod) / 2;
    }//end while
    ao();
    servo0 = SERVO_STR;

    //update orientation, only if executing a left turn
    if(clicks == LTURN_CLICKS)
      {
        if(ORIENTATION == 0)
          ORIENTATION = 3;
        else
          if(ORIENTATION == 1)
            ORIENTATION = 0;
          else
            if(ORIENTATION == 2)
              ORIENTATION = 1;
            else
              ORIENTATION = 2;
    }
}//end left90

/*****************************************************************/
/*90 degree turn right*/

void right90(int clicks)
{
    int avg = 0;
    reset();
    servo0 = SERVO_RGT;
    sleep(0.05);
    while(avg < clicks)
      {
        motor(RIGHT_MOTOR, -40);
        motor(LEFT_MOTOR, 40);
        avg = (right_encod + left_encod) / 2;
    }//end while
    ao();
    servo0 = SERVO_STR;

    //update orientation
    if(clicks == RTURN_CLICKS)
      {
```

```
        if(ORIENTATION == 0)
          ORIENTATION = 1;
        else
          if(ORIENTATION == 1)
            ORIENTATION = 2;
          else
            if(ORIENTATION == 2)
              ORIENTATION = 3;
            else
              ORIENTATION = 0;
    }//end if

}//end method right90

/*****************************************************************/
/*180 degree turn to the left*/

void left180(int clicks)
{
    int avg=0;
    reset();
    servo0 = SERVO_LFT;
    sleep(0.05);
    while(avg < clicks)
      {
        motor(LEFT_MOTOR, -50);
        motor(RIGHT_MOTOR, 50);
        avg = (left_encod + right_encod) / 2;
    }//end while
    ao();
    servo0 = SERVO_STR;

    //update orientation
    if(clicks == L180_CLICKS)
      {
        if(ORIENTATION == 0)
          ORIENTATION = 2;
        else
          if(ORIENTATION == 1)
            ORIENTATION = 3;
          else
            if(ORIENTATION == 2)
              ORIENTATION = 0;
            else
              ORIENTATION = 1;
    }//end if

}//end method left180

/*****************************************************************/
/*180 degree turn to the right*/

void right180(int clicks)
{
    int avg = 0;
    reset();
    servo0 = SERVO_RGT;
    sleep(0.05);
    while(avg < clicks)
      {
        motor(RIGHT_MOTOR, -50);
        motor(LEFT_MOTOR, 50);
        avg = (left_encod + right_encod) / 2;
    }//end while
    ao();
    servo0 = SERVO_STR;

    //update orientation
    if(clicks == R180_CLICKS)
      {
        if(ORIENTATION == 0)
          ORIENTATION = 2;
        else
          if(ORIENTATION == 1)
            ORIENTATION = 3;
          else
            if(ORIENTATION == 2)
              ORIENTATION = 0;
            else
```

```
            ORIENTATION = 1;
    }//end if
}//end method right180

int search(int mode)
{
    int i=0;                        //for-loop variable
    int high_sample=0;              //the high orange value from camera sampling
    int sample=0;                   //the value for one orange sample value
    int turn_clicks=35;             //1/5 of a turn
    int turned_clicks=0;            //how many clicks have been turned, total
    int left_right=0;               //0-4, to the left, 5 back forward, 6-10, to the right
    int clicks_traveled=0;            //how many clicks moved forward in looking for the tape
    int forward_clicks = 200;       //how many clicks to move forward each time...
    int found=0;                    //found the black tape??? 1 for yes, 0 for no
    int orange_found = 0;           //found the orange box??? 1 for yes, 0 for no

    if(mode == 0)
      {

        while(orange_found == 0)
          {
            high_sample=0;
            sample=0;
            //POLL THE CAMERA TEN TIMES, TAKE THE HIGHEST VALUE.
            for(i=0; i<10; i++)
              {
                sample = track_orange();
                if(sample > high_sample)
                  high_sample = sample;
            }//end for
            //IF THIS HIGH VALUE IS GREATER THAN 15, GO GRAB IT.

            if (high_sample > 15)
              {
                printf("orange found:%d ts: %d \n", track_confidence, track_size);
                move_straight(POUNCE_CLICKS);
                close_net();
                move_back(POUNCE_CLICKS);

                //TURN THE ROBOT BACK IN THE DIRECTION IT WAS ORIGINALLY FACING!!!
                //TURN BACK TO THE RIGHT...
                if((left_right<5) && (left_right != 0))
                  {
                    right90(left_right * turn_clicks);
                    left_right = 0;
                }//end if
                //TURN BACK TO THE LEFT...
                if((left_right>4) && (left_right != 5))
                  {
                    left90(left_right * turn_clicks);
                    left_right = 0;
                }//end if
                //set found to "true"
                orange_found = 1;
                //REPLACE THE COORDINATES IN THE ARRAY WITH -1.0, -1.0
                replace_coordinates(des_posX, des_posY);
                return 1;
            }//end if

            else {
                beep();
                printf("nothing...\n");
                //TURN 35 CLICKS (1/5 OF A TURN) TO THE LEFT, THEN WE'LL LOOK AGAIN
                if(left_right < 5)
                  {
                    left90(turn_clicks);
                    left_right++;
                }//end if
                else
                  //AFTER 5 PARTIAL TURNS, MAKE A 90 RIGHT TURN BACK TO FORWARD.
                  if(left_right == 5)
                    {
                      right90(RTURN_CLICKS);
                      left_right++;
                  }//end if
                  else
                    //TURN 35 CLICKS (1/5 OF A TURN) TO THE RIGHT, THEN WE'LL LOOK AGAIN
                    if(left_right < 11)
```

```
                    {
                      right90(turn_clicks);
                      left_right++;
                  }//end if
                  //DIDN'T FIND SHIT IN 180 DEGREE SWEEP, MOVE FORWARD, AND TRY AGAIN???
                  else
                    {
                      left90(LTURN_CLICKS);
                      left_right=0;
                      move_straight(200);         //<----- CLEAN THIS UP LATER!!!!!!!
                      beep();
                      beep();
                      beep();
                  }//end else
          }//end else
        }//end while
        return 0;
    }//end if

    //WE ARE LOOKING FOR THE BLACK TAPED LOCATION!!!
    else
      {

        while(found == 0)
          {
            clicks_traveled = align_to_tape();
            open_net();
            sleep(3.0);
            beep();
            beep();
            beep();
            beep();
            found = 1;
            move_back(clicks_traveled);
            return 1;


        }//end while
        if(found == 1)
          return 1;
        else
          {
            return 0;
            //mark this location as missing, proceed to next location???
            //or call yourself lost and implement some task.
            //NEED TO FIGURE OUT WHAT WE WANT TO DO FOR THIS...
        }//end else
    }//end else
}//end method search


int align_to_tape()
{
    int traveled=0;
    int i, j;
    reset();
    while (analog(RIGHT_IR) < IR_THRESHOLD || analog(LEFT_IR) < IR_THRESHOLD) // keep looping ultin readings
from both light sensors
      {                                                     // are higher than threshold

        if (analog(RIGHT_IR) >= IR_THRESHOLD) // if right light sensor reading is higher than threshol
          {
            servo0 = 600;sleep(0.01);
            motor(RIGHT_MOTOR,-10); // set right motor power at 5 backward
            motor(LEFT_MOTOR,60);sleep(0.05);
            ao();
            // set left motor power at 20 forwar then wait 0.05 second
            servo0 = SERVO_STR;sleep(0.01);
            move_back(15);
        }
        else if (analog(LEFT_IR) >= IR_THRESHOLD) // if left light sensor reading is higher than threshold
            {
              servo0 = 3400;sleep(0.01);
              motor(LEFT_MOTOR,-10); // set left motor power at 5 backward
              motor(RIGHT_MOTOR,60);sleep(0.05);
              ao();
              // set right motor power at 20 forward then wait 0.05 second
              servo0 = SERVO_STR;sleep(0.01);
              move_back(15);
```

```
          }
          else
            {
              traveled = traveled + move_slow(); // set motor power low forward
          }
    }

    ao();
    servo0 = SERVO_STR;sleep(0.01);
    return traveled;
}


/*****************************************************************/
//replace this point in the coordinate array with -1,-1.
void replace_coordinates(float x, float y)
{
    int i=0;

    while(i<42)
      {
        if((x == coordinate_array[i])&&(y == coordinate_array[i+1]))
          {
            coordinate_array[i] = -1.0;
            coordinate_array[i+1] = -1.0;
            printf("Removed Coord: %d, %d\n", x, y);
        }//end if

        i=i+2;
    }//end while

}//end method replace_coordinates

int move_slow()
{
    int slow_power = 20;
    int l_speed, r_speed;
    int switch = 0;

    l_speed=r_speed=slow_power;

    servo0 = SERVO_STR;
    sleep(0.5);

    ramp_up();
    sleep(0.1);
    /*based off of dr. miller's straight line code, this one alternates
    both decreasing and increasing the motor power, alternately.*/
    while((analog(2) < IR_THRESHOLD)&&(analog(3) < IR_THRESHOLD))
      {
        if(left_encod<right_encod)
          {
            if (switch == 0)
              {
                r_speed=(8*slow_power)/10;
                switch = 1;
            }//end if
            else
              {
                l_speed = (12*slow_power)/10;
                switch = 0;
            }//end else
        }//end if
        else
          {
            if(left_encod>right_encod)
              {
                if (switch == 0)
                  {
                    l_speed=(8*slow_power)/10;
                    switch = 1;
                }//end if
                else
                  {
                    r_speed = (12*slow_power)/10;
                    switch = 0;
                }//end else
            }//end if
        }//end else
```

```
        motor(LEFT_MOTOR, l_speed);
        motor(RIGHT_MOTOR, r_speed);
    }//end while
 return (left_encod + right_encod)/2;
}//end method move_straight




/*****************************************************************/
/*main method, right now it's not used for much more than testing out
the auxiliary methods.*/

void main()
{
    int success = 0;
    int enc_clicks = 200;
    init_camera();
    init_expbd_servos(1);
    enable_encoder(LEFT_ENCODER);
    enable_encoder(RIGHT_ENCODER);

    ORIENTATION = 2;
    curr_posX = coordinate_array[40];
    curr_posY = coordinate_array[41];
    des_posX = 0.0;
    des_posY = 0.0;
    start_process(encoder_reader());
    servo0 = SERVO_STR;
    sleep(0.5);

    while(1)
      {
        //  printf("press start to initialize camera \n");
        while(start_button() != 1)
          {
        }//end while

        clamp_camera_yuv();
        printf("ready to go, press start again! \n");

        while(start_button() != 1)
          {
        }//end while

        while(1)
          {
            //WORK ON THIS AND FLESH IT OUT IF YOU CAN.
            success = Navigate_to_Nearest(0);
            printf("O: %d \n", ORIENTATION);
            //if(success == 1)
            Navigate_to_Nearest(1);
            //else
            //  Navigate_to_Nearest(0);
        }//end while
    }//end while
    init_expbd_servos(0);
}//end main

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