Robot Design

The robot that we built for the project#1 is a three-wheeled differential drive with two front-wheels driving and one rear free wheel. The robot consists of two major functional components: sensing unit and driving unit.

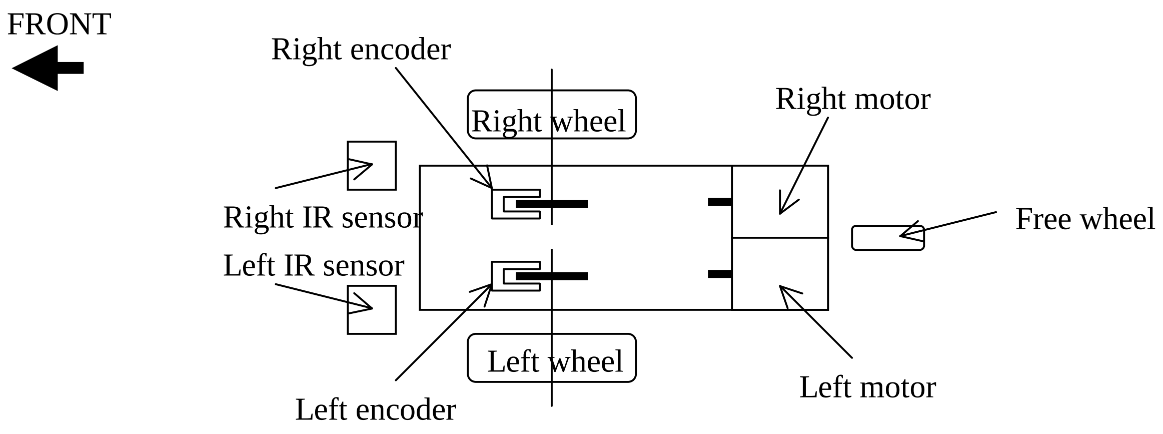


Figure 1: Robot design overview

Sensing unit

At the front of the robot, there are two IR reflection sensors. The sensors are located under the chassis of the robot, about 1.0cm above the floor to ensure the accurate reading of the color on the floor. They are fixed by several 1x4 beams and 1x4 plates, and attached to the chassis of the robot by two 2x4 plates.

There are two slot encoders located on the axles of two medium pulleys, each of which are on the same axle of the two front wheels. The encoders are used to calculate the distances that the two wheels traveled. See Figure 1.

Driving unit

There are two 81.6x15(mm) motorcycle wheels in the front and one free wheel that is a medium pulley with tire in the back. Two 9V motors are located at rear part of the robot. One in the left will drive the robot forward or backward and the other in the right leftward or rightward. When the left motor is on, one 8 tooth gear on the stud of the motor drives a 24 tooth crown gear, which transfers the torque through two 24 tooth gear to a 16 tooth gear. This 16 tooth gear is on the same axis with two differential gears. When driving forward or backward, the other ends of the two differential gears will not move. So, the torque from the 16 tooth gear will be transferred to the body of the differential gears, and then drive two front wheels forward or backward through a 40 tooth gear that is on the same axle with the wheel. See Figure 2 and 3.

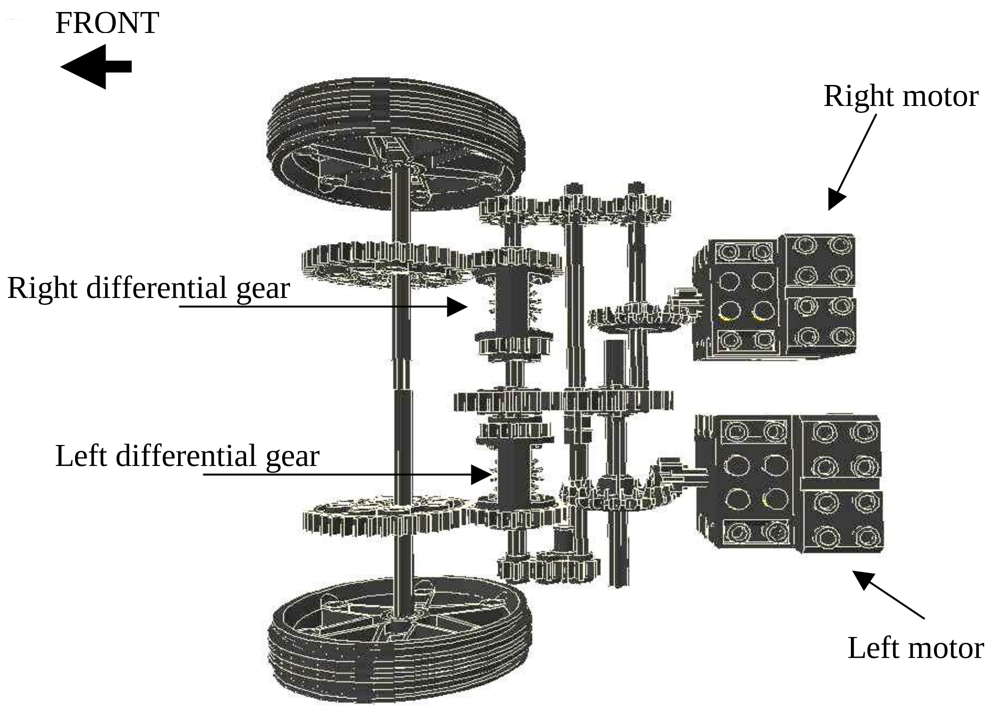


Figure 2: top view of driving unit

When the right motor is on, similarly, one 8 tooth gear on the stud of the motor will also drive a 24 tooth crown gear, which changes the direction of the torque. The crown gear is on the same axis with a 16 tooth gear, which can be seen from outside the robot. This 16 tooth gear is actually connected with other two 16 tooth gears, thus generating a three 16 tooth gear train. The third 16 tooth gear connects to a differential gear inside the robot. When turning, the other side of the differential gear will be fixed by the gear chain of the other motor and won't move. So, the 24 tooth gear on the body of the differential gear will drive the right front wheel forward or backward through the 40 tooth gear.

Go back to the three 16 tooth gear train outside the robot. Because the direction of the torque of the second gear is opposite to the others in a three 16 tooth gear train, the second gear in the gear train will transfer torque of opposite direction to the other side of the robot through two jointed studs. At the other end of the stud train, there is an 8 tooth gear, which drives two other 8 tooth gears. This forms a gear chain to transfer the torque to the left differential gear and then drive the left wheel backward or forward. The difference between left and right wheels is the driving direction. They are opposite to each other.

If we drive two motors at the same time, the output of the differential gears will be the sum or subtraction of the two torquees from the two motors, depending on the directions of the torquees. This mechanism makes possible for smooth turning and angle correction.

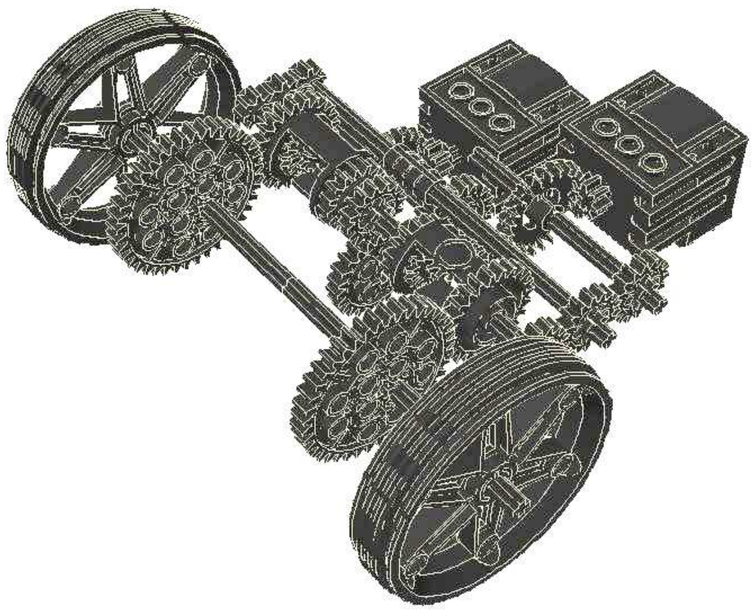


Figure 3: 3-D view of the driving unit

Problems and solutions for similar projects

The original initiative to build this dual differential drive robot instead of using one motor for each wheel is to keep the robot walking straight. Theoretically, it should go really straight because the torque offered to two front wheels are exactly the same. The testing results turned out that it was not really the case.

There are some factors making the robot not walking straight. One is the inaccurate and unreliable construction of the robot. LEGO parts have some limits on accuracy and stabilization. This reason also causes other problems such as the slippage of the gears.

In addition, building the robot's gear or wheel too tight on one side might cause the robot drifts to one side or the other. We should be able to avoid or less the problem by improving the gear trains or the construction.

The third reason might involve tires and the mass of the robot. The tires on the wheels are not inflated tires. They can't provide strong and reliable support for the whole "bulky" robot. Our robot is pretty heavy after putting the handy board on it, and looks bulky from outside because the driving unit occupy too much rooms of the robot. The mass of the robot causes the tires working worse.

Furthermore, because the touching areas of the two wheels are small, any changes by the wheels will be very sensitive to the heading of the robot. One of the possible improvements for this would be change the motorcycle wheels to thicker wheels. The robot should be more fault-tolerated.

Code Design

The overall pattern in the design of the code came from the Hierarchical Paradigm, where there are separate modules for sensing, planning and acting. Since the goal of the project was simple, make a robot go three times around a six by six foot square, it was difficult to enforce separating the different tasks into their own modules. It was decided from early to approach the project from a motor control (acting) perspective. Sensor techniques like odometry would play an important part in the design. Because the plan portion was given to us in the project handout, the brains of the robot were to sense and to react appropriately.

Sensing

As mentioned above the robot used four sensors: two IR reflection sensors and two encoders. The original plan called for just one sensor, but due to the finicky nature of the IR sensors in experiments, the plan was changed to incorporate an additional IR reflection sensor. With the original plan, one IR reflection sensor in cooperation with the odometry subsystem would be used to calculate the time/distance used to cross the tape in order to make an approximation of the angle that the robot crossed the tape with. The calculations necessary to do that proved too complex. An additional IR reflection sensor negated the need for odometry to be used in those calculations.

The odometry subsystem was envisioned to only control the path of the robot through dead reckoning. However we soon found more ways in which to use the two encoders; the 90 degree turn operation as well as the complete-miss check were some of the other ways that the odometry system was used.

Planning

As mentioned above the planning portion of the operation was made less complex due to the simple nature of the rules for the project. Since the robot was only to traverse a six-foot by six-foot square three times a separate module/function for this aspect of the code was not worth the effort. The planning code treats the traversal of a 4-sided polygon three times as a single operation that could be repeated twelve times. Each leg of the traversal involves discovering the tape marking the exit region of the current corner square, calibrating the robots heading via the exit edge, discovering the ‘entrance’ tape of the next corner square, calibrate against the ‘entrance’ edge of the square, move some distance into the square and, finally, turn ninety degrees to the right.

Contingencies for less than perfect behavior amounted to making some action functions there own set of SENSE, PLAN and ACT systems in order to achieve overall system stability. We chose not to implement our dead reckoning system because the late addition of a coaster wheel on the back of the robot allowed the robot to move forward in a straight path, negating the need for a guidance system.

Acting

As mentioned above the planning process dictates that a sequence of six actions needed to be performed in order to successfully drive the robot through the course. Some of these actions were operationally equivalent and were thus factored out and grouped together. Each of these action functions could be considered their own subprogram as each has SENSE, PLAN and ACT properties.

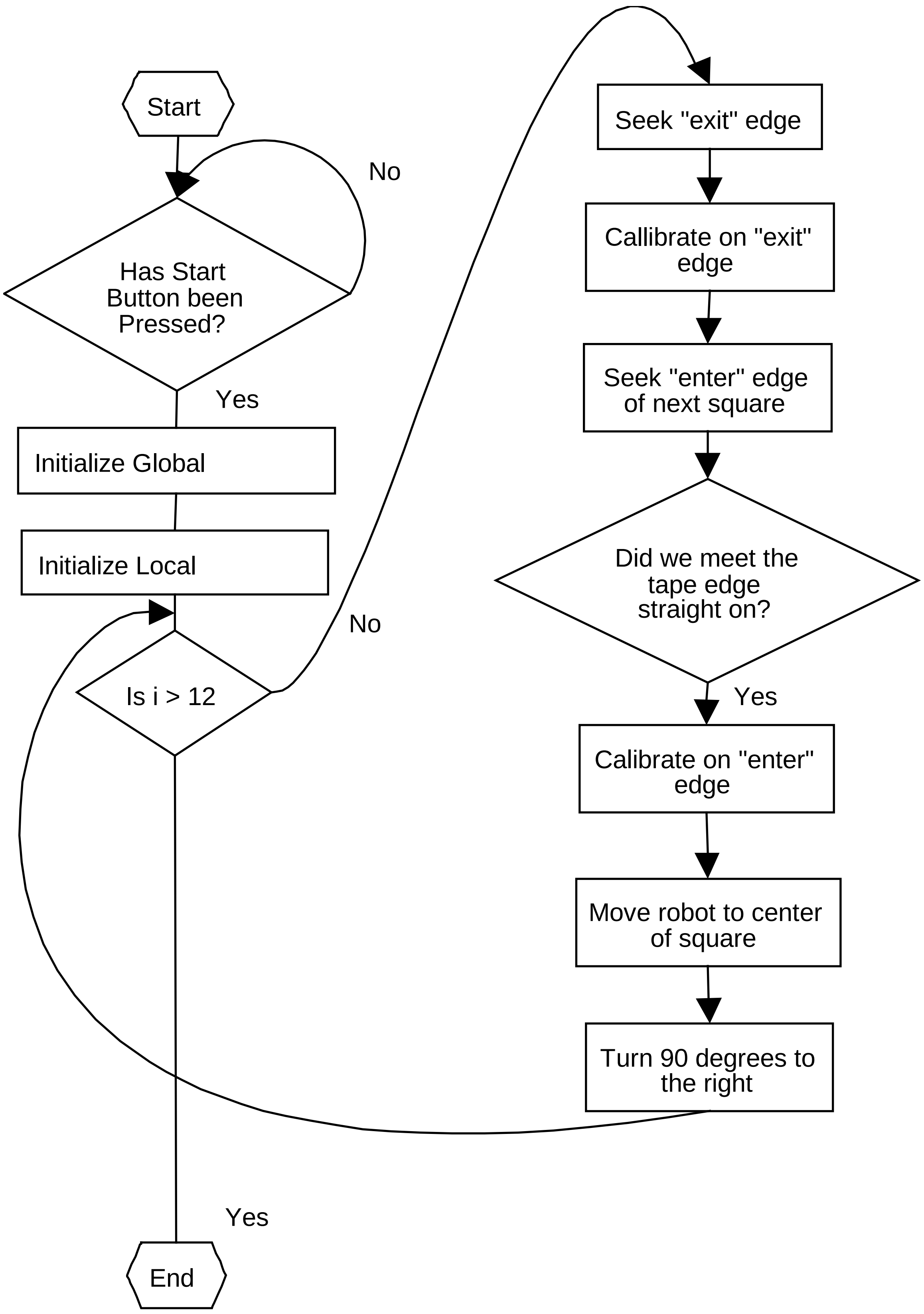
The first of these functions is the **seek_tape()**, function. This function makes the assumption that both the IR reflector sensors are reporting the absence of tape when the function is called. **seek_tape()** was designed to inform the caller through the return parameter whether or not the left or right IR reflector or neither of the reflectors were reported to have crossed the tape. Initially this method was designed so that it would make a decision immediately on the first detection of tape, but since it was nigh impossible for the robot to meet the tape straight on we had to increase the model of the tape to two dimensions. If the tape was picked up by both sensors in between the time that it took for the first sensor to pick up the tape and for both sensors to drop the tape then the function returned **0** for meeting the tape more or less straight on. The function returned either **–1** or **1**, as appropriate, if just one of the sensors reported that they hit the black tape during the same time period. Exceptions to the rule were if either of the sensors was continually picking up tape for more than two seconds than it was counted as one of the IR reflectors missing the tape entirely.

The second largest function was **CALIBRATE()** and its cousin functions **CALIBRATE_2()** and **CALIBRATE_3()**. **CALIBRATE()** assumes that the sensors are in front of the black tape and will progress slowly backwards until one of the IR sensors picks up the black tape. Then the function will turn the robot in the appropriate direction in order to ‘line up’ the sensors. Once the condition is met the function returns. **CALIBRATE_2()** and **CALIBRATE_3()** are designed to back up the robot for a constant amount of time and then orient the robot by a fixed rotation to where the taped square is.

Both the **recenter_robot()** and the **TURN_90()** functions move the robot forward for a constant amount of time determined via experimentation.

Data Organization

The data represents the link between the sensor functions **GET_STATUS()** and the action functions. These state variables are all stored as global variables and are updated very frequently via a separate process running parallel to the code that chooses between the action functions in the main method.



Start
No
Has Start Button been Pressed?
Yes
Initialize Global
Initialize Local
No
Is i > 12
Yes
End
Seek "exit" edge
Callibrate on "exit" edge
Seek "enter" edge of next square
Did we meet the tape edge straight on?
Yes
Calibrate on "enter" edge
Move robot to center of square
Turn 90 degrees to the right

Figure 4: Flowchart Diagram of Program

Code Listing

```
/*********************************************************
*********************************************************
            PROJECT1 : SENSING AND MOVEMENT
            GROUP 6  : Jianwei Zhuang
                       Joe Garfield
                       Manohar Darapureddi
                       Pedro Diaz
**********************************************************
*********************************************************/

/**********************************************************
                   NAMING THE CONSTANTS
*********************************************************/

#define CLICKS_L 16
#define CLICKS_R 12
#define DIAM_L 7.3
#define DIAM_R 7.3
#define B 15.5
#define SLEEP_TIME 0.2
#define CORR_POWER 50
#define LEFT_ENCODER 0
#define RIGHT_ENCODER 2
#define IR_SENSOR 2
#define FORWARD_MOTOR 0
#define TURN_MOTOR 2
#define PI 3.1415
#define THRESHOLD 0.1

/*********************************************************
                   GLOBAL VARIABLES
********************************************************/

int EDGE;
int do_correction;
int PREV_CLICKS[2];
int CURR_CLICKS[2];
int FD,BK,TR,TL;
int TAPE_VAL=225;
long PREV_ST;
long CURR_ST;
float diff;
float ERROR=0.0;
float PREV_POS[2];
float CURR_POS[2];
float CURR_VEL[2];
float CURR_ANGLE;
float time_t;
float ANGLE_CHANGED;
float PREV_ANGLE;

/**********************************************************
           INITIALIZING ALL THE VARIABLES
*********************************************************/
```

```
void INIT()
{
    FD=0;
    BK=0;
    TR=0;
    TL=0;
    PREV_CLICKS[0] = 0;
    PREV_CLICKS[1] = 0;
    CURR_CLICKS[0] = 0;
    CURR_CLICKS[1] = 0;
    reset_encoder(0);
    reset_encoder(2);
    PREV_POS[0] = 0.0;
    PREV_POS[1] = 0.0;
    CURR_POS[0] = 0.0;
    CURR_POS[1] = 0.0;
    CURR_ANGLE = 0.0;
    PREV_ANGLE = 0.0;
    ANGLE_CHANGED = 0.0;
    PREV_ST = mseconds();
    CURR_ST = PREV_ST;
    do_correction = 1;
}

/*************************************************************
       UPDATING THE POSITION OF ROBOT,NUMBER OF TICKS
        OF THE ENCODERS AND THE ANGLE IT HAD DRIFTED
*************************************************************/

void UPDATE_STATUS()
{
    PREV_ST = CURR_ST;
    PREV_POS[0] = CURR_POS[0];
    PREV_POS[1] = CURR_POS[1];
    PREV_CLICKS[0] = CURR_CLICKS[0];
    PREV_CLICKS[1] = CURR_CLICKS[1];
    PREV_ANGLE = CURR_ANGLE;

}

/************************************************************
      GETTING THE CURRENT STATUS OF THE ROBOT
************************************************************/

void GET_STATUS()
{
    float sl,sr;
    float dl,dr,di;
    float min;

    if(FD||TR) sl =1.0;
    else if(BK||TL) sl=-1.0;
      else;

    if(FD||TL) sr=1.0;
    else if(BK||TR) sr = -1.0;
```

```
      else;

    CURR_ST = mseconds();
    CURR_CLICKS[0] = read_encoder(LEFT_ENCODER);
    CURR_CLICKS[1] = read_encoder(RIGHT_ENCODER);

    //distance travelled by left wheel
    dl = ((float)(CURR_CLICKS[0] - PREV_CLICKS[0])/
      (float)CLICKS_L)*PI*DIAM_L;

    // distance travelled by right wheel
    dr = ((float)(CURR_CLICKS[1] - PREV_CLICKS[1])/
      (float)CLICKS_R)*PI*DIAM_R;

    // distance travelled by the robot
    di = (dl+dr)/2.0;

    //  angle drifted by the robot from its previous updated position
    CURR_ANGLE = (((float)CURR_CLICKS[0]*sl/(float)CLICKS_L)-
                (float)CURR_CLICKS[1]*sr/(float)CLICKS_R))*PI*DIAM_L/B;

     //absolute position of the robot from (0,0) along X-coordinate
    CURR_POS[0] = CURR_POS[0] + di*sin(CURR_ANGLE);

    //absolute position of the robot from (0,0) along Y-coordinate
    CURR_POS[1] = CURR_POS[1] + di*cos(CURR_ANGLE);

    UPDATE_STATUS();
    printf("y = %f\n", CURR_POS[1]);
}

/*****************************************************************
             ERROR CORRECTION FUNCTION
*****************************************************************/

void ERROR_CORR()
{
     if(diff<0.0)
      {
        motor(2,10);
        sleep(0.1);
        off(2);
      }
    else
      {
        motor(2,-10);
        sleep(0.1);
        off(2);
      }
    beep();
}

/********************************************************
    CALIBRATE FUNCTION MAKES SURE THAT WHEN BOTH THE COLOR
     SENSORS SENSE THE TAPE THE ROBOT IS PERPENDICULAR TO
                     THE BLACK TAPE
********************************************************/
```

```
void CALIBRATE()
{
    while (1) {
        if(analog(2) < TAPE_VAL || analog(4) < TAPE_VAL) {
            motor(0,-50);
            if(analog(2) > TAPE_VAL) {
                while (analog(4) < TAPE_VAL) {
                    motor(2,22);
                }
                off(2);
            }
            else if(analog(4) > TAPE_VAL) {
               while (analog(2) < TAPE_VAL) {
                    motor(2,-22);
               }
                  off(2);
            }
        }
        else
          break;
    }
    ao();
}

/*********************************************************
   THE CALIBRATE_2 FUNCTION MAKES SURE THAT WHEN ONLY LEFT
   SENSOR SENSES THE TAPE THE ROBOT MOVES BACK AND CHANGES
   ITS DIRECTION SO THAT BOTH THE SENSORS SEEK THE TAPE
**********************************************************/

void CALIBRATE_2()
{
    motor(FORWARD_MOTOR, -100);
    sleep(1.0);
    ao();
    motor(TURN_MOTOR, -50);
    sleep(0.4);
    ao();
}

/*********************************************************
   THE CALIBRATE_3 FUNCTION MAKES SURE THAT WHEN ONLY RIGHT
   SENSOR SENSES THE TAPE THE ROBOT MOVES BACK AND CHANGES
   ITS DIRECTION SO THAT BOTH THE SENSORS SEEK THE TAPE
**********************************************************/

void CALIBRATE_3()
{
    motor(FORWARD_MOTOR, -100);
    sleep(1.0);
    ao();
    motor(TURN_MOTOR, 50);
    sleep(0.4);
    ao();
}
```

```
/*********************************************************
        THIS FUNCTION MAKES SURE THAT THE ROBOT TAKES
       A 90 DEGREE TURN AFTER IT IS INSIDE THE SQUARE BOX
**********************************************************/

void TURN_90(float angle)
{
    int target_clicks, start_clicks;
    motor(TURN_MOTOR, 100);
    sleep(1.2);
    off(2);
    ao();
}

/*******************************************************
    THIS PROCESS KEEPS UPDATING THE STATUS OF THE ROBOT
********************************************************/

void update_status_process()
{
    while(1){
        GET_STATUS();
    }
}

/**********************************************************
    THIS FUNCTION MAKES SURE THAT THE ROBOT MOVES TILL IT
    SENSES THE TAPE AND STOPES A LITTLE BIT FURTHER AHEAD
    OF THE TAPE.THE FUNCTION RETURNS A VALUE OF "0" WHEN
    THE BOTH THE SENSORS SENSE THE TAPE.THE FUNCTION RETURNS
    "-1"WHEN THE LEFT SENSOR ONLY SENSES THE TAPE.THE FUNCTION
    RETURNS "1" ONLY WHEN RIGHT SENSOR SENSES THE TAPE
***********************************************************/

int seek_tape(int power)
{
    int left_hit, right_hit, adjusted;
    float time_stamp;
    float y;

    left_hit = 0;
    right_hit = 0;
    adjusted = 0;

    motor(FORWARD_MOTOR, power);
    while (analog(4) < TAPE_VAL && analog(2) < TAPE_VAL) {
        y = CURR_POS[1];
        if(!adjusted && y > 2.0 * 12.0 * 2.54) {
            adjusted = 1;
            motor(TURN_MOTOR, -10);
            sleep(0.3);
            motor(TURN_MOTOR, 0);
        }
        if(y > 8.0*( ((float)knob())/255.0) * 12.0 * 2.54) {
            return -1;
        }
     }
    time_stamp = -1.0;

    printf("%d %d\n",analog(2),analog(4));
    while((analog(4) > TAPE_VAL || analog(2) > TAPE_VAL) ) {
        if(analog(4) >= TAPE_VAL){
            right_hit = 1;
        }
        else if(analog(2) >= TAPE_VAL) {
              left_hit = 1;
        }
        //If an IR has been activated and becomes 'inactive'
        //we start the timer
        if((right_hit == 1 && analog(4) < TAPE_VAL) ||
            left_hit == 1 && analog(2) < TAPE_VAL)) {
              time_stamp = seconds();
        }

        if(time_stamp > 0.0 && seconds() - time_stamp >= 2.0) {
            if(analog(2) < TAPE_VAL && analog(4) >= TAPE_VAL) {
                ao();
                motor(FORWARD_MOTOR, -100);
                sleep(7.0);
                return 1;
            }
        }
        if(analog(4) < TAPE_VAL && analog(2) >= TAPE_VAL) {
                ao();
                motor(FORWARD_MOTOR, -100);
                sleep(7.0);
                return -1;
              }
        }

    }
    if(left_hit && !right_hit)
      return -1;
    else if(!left_hit && right_hit)
        return 1;
      return 0;
}

/*********************************************************
    THE ROBOT MOVES TO APPROXIMATELY THE MIDDLE OF
          THE SQUARE AFTER IT SENSES THE ROBOT
*********************************************************/

void recenter_robot()
{
    ao();
    motor(0, 30);
    while(CURR_POS[1] < 5.0 * 2.54);
    off(0);
}

/*********************************************************
                  MAIN  FUNCTION
*********************************************************/
```

```
void main()
{
    int update_status_pid;
    int error_correction_pid;
    float angle,y;
    int i, direction;

// This makes sure that the robot doesn’n start untill start button is
//pressed
    while(!start_button());
    sleep(0.5);
    enable_encoder(LEFT_ENCODER);
    enable_encoder(RIGHT_ENCODER);
    INIT();
    FD=1;
    update_status_pid = start_process(update_status_process());
    y=CURR_POS[1];

    for(i=0;i<12;i++) {
        INIT();
        FD=1;seek_tape(100);
        beep();
        sleep(0.50);
        ao();
        beep();
        CALIBRATE();
        motor(FORWARD_MOTOR, 100);
        sleep(0.5);
        ao();
        INIT();FD=1;
        beep();
        direction = seek_tape(100);
        ao();
        sleep(0.5);
        while(direction != 0){
            sleep(0.5);
            ao();
            if(direction < 0)
              CALIBRATE_2();
            else
              CALIBRATE_3();
            INIT();
            direction = seek_tape(40);
        }
        ao();
        sleep(0.5);
        CALIBRATE();
        beep();
        INIT();
        recenter_robot();
        ao();
        beep();
        sleep(1.5);
        INIT(); FD=0; TR = 1;
        TURN_90(angle);
        TR = 0;
    }

    kill_process(update_status_pid);
    ao();
}

/*************************************************************
                   END OF THE CODE
*************************************************************/
```