Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

PROJECT 1: FINAL REPORT
TEAM 10

Robot Design Synopsis

Design Phase:

Various ideas were contributed including one in which the robot travels in a circle to visit each square and one that has one set of wheels for North-South travel and one set for East-West travel. The initial design (see Fig 1) consisted of four independently powered wheels and two IR sensors placed at the front of the chassis to sense traversal of an edge of a square. We believed that sacrificing code simplicity for hardware simplicity would be an acceptable solution.

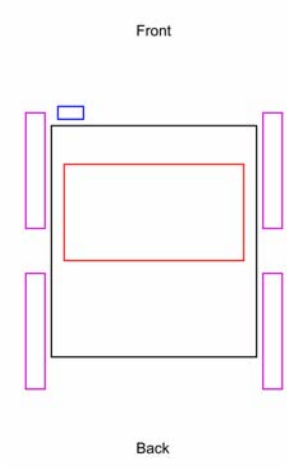

Fig 1. Initial Robot Design

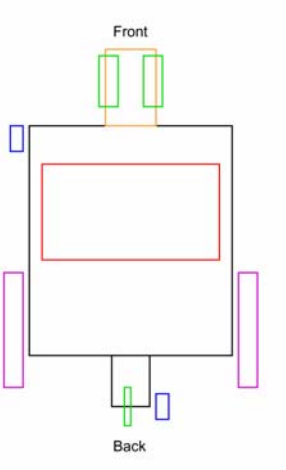

Fig 2. Final Robot Design

Hardware Phase:

During construction, the complexities of using four independently powered wheels forced us to reconsider our design. We removed the two front wheels and

replaced it with a third, passive wheel attached to the chassis using a caster mechanism. It became apparent that, in this configuration, the robot's ability to travel in straight lines was far too dependent on the position of this third wheel. Two options were considered. The first option was to change which end was considered the front and reposition the IR sensors accordingly. The second option involved replacing the caster mechanism with a servo that allowed us to specify specific positions for the wheel. During testing of this option, it became evident that the concept of traveling in a circle to visit each square could be realized by this design.

There would be a number of benefits to this approach that previous designs would not have. First, the robot would not be required to consistently perform precise 90° turns a difficult proposition with the supplied materials. It would need only to maintain its specified turn radius which is easily achievable using the servo. Second, the constant starting and stopping required by traveling in a square drains the battery at an appreciable rate. If the robot travels in a circle however, there is no need for it to start and stop. Thus, the load the motors have on the battery is significantly reduced. Finally, the stops and turns preformed during traveling in a square increases the travel time greatly and jeopardizes the possibility of performing more than one run in the allotted time. When traveling in a circle, not only are the stops and turns not required, but the robot maintains its speed throughout the circuit which drastically lowers the travel time. In testing, we were able to travel a complete circuit of the course in under fifteen seconds. It is important to note that each of these three areas of difficulty have non-trivial solutions. Rather than adding complexity to the robot's hardware and software design to implement

solutions to these problems, we have chosen a design that avoids the problems all together.

Testing Phase:

During testing, a caster wheel was attached to the rear of the robot along with an encoder to measure the distance the vehicle has traveled. This provides a secondary sensing device to determine when the robot has made three circuits of the course. The robot can be configured to stop after either the robot has traversed the requisite number of black lines, the specified distance has been traveled, or both.

Robot Design (Fig 2):

The robot vehicle is propelled by two powered wheels on the rear left and right of the vehicle. It is steered using a pair of small wheels positioned at the front of the vehicle and is controlled by a servo that specifies their rotation angle (see Fig 3). An infra-red sensor is attached to the front left of the vehicle and is used for sensing the traversal of the robot over a piece of black electrical tape (see Fig 4). To provide sensor redundancy, an optional wheel can be attached to the rear of the vehicle using a caster mechanism. An encoder is attached this wheel and is used to measure the distance traveled by the vehicle (see Fig 5).

Fig 3. Servo

Fig 4. IR Sensor

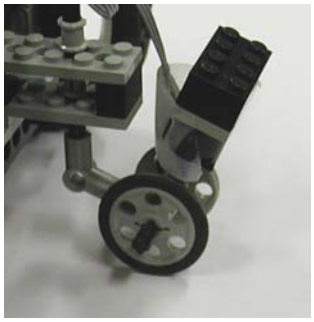

Fig 5. Encoder

Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

Robot Code

```
/* Project 1
 * Group 10
 * Eskridge, Lopez, Maole, & Utley
 */

/*
 * Global variables
 */

// Boolean globals
int TRUE = 1;
int FALSE = 0;

// Motor IDs
int MOTOR_R_ID = 0;
int MOTOR_L_ID = 2;

// Encoder ID
int ENCODER_ID = 0;

// IR ID
int IR_ID = 3;

// The servo value that turns us at the correct angle
// to navigate around the track in a circle.
int SERVO_VALUE = 2735;

// The power levels to send to the motors
int MOTOR_PWR_FWD = 20;
int MOTOR_PWR_BRAKE = -40;
float MOTOR_PWR_BRAKE_TIME = 0.2;

// Flag indicating that the robot is done
// performing its task(s)
int DONE = FALSE;

// The amount of time to sleep while waiting for the servo to position itself
float SERVO_POSITION_SLEEP = 0.5;

// The amount of time to sleep between progress checks
float MONITOR_SLEEP = 0.2;

// IR threshold value between low and high
```

Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

```
int IR_THRESHOLD_VALUE = 115;

// The default IR level
int DEFAULT_IR_VALUE = -1;

// The minimum number of lines we want to travers
int MIN_LINE_TRAVERSAL_COUNT = 24;

// The minimum number of encoder events in 3 circuits
int MIN_EVENT_COUNT = 3160 - 24;

// The amount of time to sleep between monitoring the distance
float MONITOR_DISTANCE_SLEEP = 0.2;

// Trigger for monitors to signal that they are finished
int CURRENT_DONE_TRIGGER_COUNT = 0;

// The threshold trigger value which signals that the monitors
// say we are done.  A value of '2' says both monitors must agree
// (logical AND).  A value of '1' says only one monitor must signal
// that it is finished (logical OR).
int DONE_TRIGGER_COUNT = 2;

// Current number of lines crossed
int CURRENT_LINE_COUNT = 0;

// Current encoder event count
int CURRENT_EVENT_COUNT = 0;

// PIDS
int CUTOFF_PID = 0;
int MONITOR_PID = 0;
int MONITOR_LINE_PID = 0;
int MONITOR_DISTANCE_PID = 0;

/*
 * Initialize the robot
 */
void init()
{
   printf( "\nInitializing..." );

   // Start our cutoff process
   CUTOFF_PID = start_process( cutoff() );
```

```
   // Initialize the servo
   init_expbd_servos( TRUE );

   // Position the servo
   servo0 = SERVO_VALUE;

   // Sleep so the servo has time to position itself
   sleep( SERVO_POSITION_SLEEP );

   // Start the monitor process
   MONITOR_PID = start_process( monitorProgress() );
}

/*
 * Monitors robot progress and signals completion of the tasks
 */
void monitorProgress()
{
   // Start the line traversal monitor
   MONITOR_LINE_PID = start_process( monitorLineTraversal() );

   // Start the distance monitor
   MONITOR_DISTANCE_PID = start_process( monitorDistanceTraveled() );

   // Loop until both are done
   while( CURRENT_DONE_TRIGGER_COUNT < DONE_TRIGGER_COUNT )
     {
      // Display our progress
      printf( "\nLines=[%d]      Events=[%d]",
            CURRENT_LINE_COUNT, CURRENT_EVENT_COUNT );

      // Sleep for a bit
      sleep( MONITOR_SLEEP );
   }

   // We are done
   MONITOR_PID = 0;
   DONE = TRUE;
}

/*
 * Monitors the number of black lines crossed
 */
void monitorLineTraversal()
{
```

Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

```
   // Set up our variables
   int currentValue = 0;
   int lastValue = DEFAULT_IR_VALUE;
   int finished = FALSE;

   // Loop until we are done

   while( finished == FALSE )
     {
      // Get the current value of the sensor
      currentValue = analog( IR_ID );

      if( (currentValue > IR_THRESHOLD_VALUE) && (lastValue <=
IR_THRESHOLD_VALUE ) )
        {
         // We have crossed a line
         CURRENT_LINE_COUNT++;

         // Have we crossed the desired number of lines?
         if( CURRENT_LINE_COUNT >= MIN_LINE_TRAVERSAL_COUNT )
           {
            // We are finished
            CURRENT_DONE_TRIGGER_COUNT++;
            finished = TRUE;
         }

         // Make sure the encoder is working
         if( CURRENT_LINE_COUNT == 1 )
           {
            // Is the encoder count at 0
            if( CURRENT_EVENT_COUNT == 0 )
              {
               // Something is wrong (maybe dying battery?)
               // Stop monitoring the encoder by triggering it's finished value
               CURRENT_EVENT_COUNT = MIN_EVENT_COUNT;
            }
         }
      }

      // Save the current value
      lastValue = currentValue;
   }

   // Clean up after ourselves
   MONITOR_LINE_PID = 0;
```

```
}

/*
 * Monitors the distance travelled
 */
void monitorDistanceTraveled()
{
   int finished = FALSE;

   // Only enable it if it isn't on
   if( read_encoder( ENCODER_ID ) == 0 )
     {
      // Not enabled
      enable_encoder( ENCODER_ID );
   }
   else
     {
      // Enabled
      reset_encoder( ENCODER_ID );
   }

   // Loop until we are finished
   while( finished == FALSE )
     {
      // Sleep for a bit
      sleep( MONITOR_DISTANCE_SLEEP );

      // Get the encoder event count
      CURRENT_EVENT_COUNT = read_encoder( ENCODER_ID );

      // Have we travelled the desired distance
      if( CURRENT_EVENT_COUNT >= MIN_EVENT_COUNT )
        {
         // We are done
         CURRENT_DONE_TRIGGER_COUNT++;
         finished = TRUE;
      }
   }

   // Clean up after ourselves
   MONITOR_DISTANCE_PID = 0;
}

/*
 * Turn on the motors and move forwards
```

```
 */
void startMotors()
{
   // Turn on both motors
   motor( MOTOR_R_ID, MOTOR_PWR_FWD );
   motor( MOTOR_L_ID, MOTOR_PWR_FWD );
}

/*
 * Turn off the motors and stop the vehicle
 */
void stopMotors()
{
   // Throw the motors in reverse so we don't coast
   motor( MOTOR_R_ID, MOTOR_PWR_BRAKE);
   motor( MOTOR_L_ID, MOTOR_PWR_BRAKE);

   // Sleep for a second so we stop
   sleep( MOTOR_PWR_BRAKE_TIME );

   // Turn all the motors off
   alloff();
}

/*
 * Cutoff method to shutdown robot
 */

void cutoff()
{
   // Loop forever until the stop button is pressed
   while( !stop_button() )
     {
       // Do nothing
   }

   // We are quiting so change our pid to 0
   CUTOFF_PID = 0;

   // Signal that we are done and quit
   DONE = TRUE;
}

/*
 * Displays a message and waits for the start button to be pressed
```

Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

```
 */
void waitForStart( char msg[] )
{
   printf( msg );
   start_press();
}

/*
 * Cleans up any lingering processes
 */
void cleanup()
{

   // Clean up the cutoff thread
   if( CUTOFF_PID > 0 )
     {
      kill_process( CUTOFF_PID );
   }

   // Clean up the  thread
   if( MONITOR_PID > 0 )
     {
      kill_process( MONITOR_PID );
   }

   // Clean up the  thread
   if( MONITOR_LINE_PID > 0 )
     {
      kill_process( MONITOR_LINE_PID );
   }

   // Clean up the  thread
   if( MONITOR_DISTANCE_PID > 0 )
     {
      kill_process( MONITOR_DISTANCE_PID );
   }

   // Disable the encoder
   disable_encoder( ENCODER_ID );

   // Disable the servo
   init_expbd_servos(0);
}

/*
```

Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

```
 * Main method
 */
void main()
{
   // Wait for the user to start everything
   waitForStart( "Press START..." );

   // Initialize
   init();

   // Start moving
   startMotors();

   // Idle processing until we are done
   while( DONE == FALSE )
     {
      // Do nothing
   }

   // Stop moving
   stopMotors();

   // Clean up after ourselves
   cleanup();

}
```

Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

Robot Code Documentation

The following global variables are used in the robot code:

int TRUE = 1;	Used as a Boolean value
int FALSE = 0;	Used as a Boolean value
int MOTOR_R_ID = 0;	Motor Port 0 is the Right Motor
int MOTOR_L_ID = 2;	Motor Port 2 is the Left Motor
int ENCODER_ID = 0;	Encoder Port 0 is the Encoder
int IR_ID = 3;	Analog Port 3 is the Infrared Sensor
int SERVO_VALUE = 2735;	Specifies the position for the Servo
int MOTOR_PWR_FWD = 20;	The motors use 20% power for forward
int MOTOR_PWR_BRAKE = -40;	The motors use 40% power for backward
float MOTOR_PWR_BRAKE_TIME = 0.2;	Wait for 0.2 seconds after stopping
int DONE = FALSE;	Flag to indicate when robot is done
float SERVO_POSITION_SLEEP = 0.5;	Waits for 0.5 seconds to position the servo
float MONITOR_SLEEP = 0.2;	Wait 0.2 seconds before checking progress
int IR_THRESHOLD_VALUE = 115;	Reading above 115 are black lines
int DEFAULT_IR_VALUE = -1;	The default IR level before starting
int MIN_LINE_TRAVERSAL_COUNT = 24;	The minimum number of lines to detect
int MIN_EVENT_COUNT = 3160 - 24;	The minimum number of encoder events to detect
float MONITOR_DISTANCE_SLEEP = 0.2;	Wait 0.2 seconds before checking distance
int CURRENT_DONE_TRIGGER_COUNT = 0;	Trigger for monitors to signal that they are finished
int DONE_TRIGGER_COUNT = 2;	The threshold trigger value which signals that the monitors say we are done. A value of '2' says both monitors must agree (logical AND). A value of '1' says only one monitor must signal that it is finished (logical OR).
int CURRENT_LINE_COUNT = 0;	Current number of lines crossed
int CURRENT_EVENT_COUNT = 0;	Current encoder event count
int CUTOFF_PID = 0;	PID for cutoff() process
int MONITOR_PID = 0;	PID for monitorProgress() process
int MONITOR_LINE_PID = 0;	PID for monitorLineTraversal() process
int MONITOR_DISTANCE_PID = 0;	PID for monitorDistanceTraveled() process

Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

The following processes are used:

cutoff() This process loops continuously waiting for the stop button to be pressed. When it is, the program terminates.

monitorProgress() The process continuously checks to see if CURRENT_DONE_TRIGGER_COUNT is less than DONE_TRIGGER_COUNT. If it is, the values of CURRENT_LINE_COUNT and CURRENT_EVENT_COUNT are displayed. If it is not, then the DONE flag is toggled and the program ends.

monitorLineTraversal() This process keeps track of how many lines have been detected. Lines are detected when the currentValue of the IR sensor is greater than the IR_THRESHOLD_VALUE and the lastValue was less than the IR_THRESHOLD_VALUE. This way, we only detect the line when we first go over it. Once the desired number of lines are detected, the process increments the CURRENT_DONE_TRIGGER_COUNT and exits.

monitorDistanceTraveled() This process keeps track of how far the robot has traveled. This is done by reading the encoder. Once the CURRENT_EVENT_COUNT is larger than the MIN_EVENT_COUNT, the process increments the CURRENT_DONE_TRIGGER_COUNT and exits.

The following functions are used:

waitForStart() This function prints out a message and waits for the start button to be pressed.

init() This function starts the cutoff() process, initializes the servos, sets the servo to the desired position, and starts the monitorProgress() process.

startMotors() This function turns on both motors at the desired power level.

stopMotors() This function puts both motors in reverse at double the power of forward. This is done for 0.2 seconds to prevent the robot from coasting. Then, all motors are turned off.

cleanup() This function is the last to execute. It goes through the processes and kills them all. It also disables the encoder and the servo.

Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

Testing and Demo Synopsis

The variables in the testing phase that received our attention were: the servo angle, the number of lines caught by the IR sensor, and the number of encoder events.

The servo setting, which dictated our turn radius, was empirically determined. We initially attempted to analytically determine the value, but found that to do so required technical specifications of the servo that we did not have. After a number of trials, the value of 2728 was found to steer the robot though the approximate center of each marked square. This value proved to give the desired path for the rest of testing that day. However, in testing the next day, an increased value of 2735, indicating a tighter turn radius, was required. On the day of the demo, the value again required modification and was increased to a value of 2741. The cause for this required change is unknown as no modifications were made to the robot.

The performance of the infra-red detector as a line traversal sensor was excellent. Once an appropriate position for the sensor and the threshold values were determined, it only failed to detect a line in a few instances. These failures were minimal and can be attributed to dirt on the tape which interfered with the sensing.

The encoder was used to determine the distance that the robot had traveled since the start. The total number of events was analytically determined and incorporated into the robot's software. The robot's ability to calculate the distance it had traveled performed quite well until the battery began to lose charge and shut down the sensor. As a result, the robot was given the ability to detect if the encoder was working and shut down the monitoring thread if necessary. During the final day of testing (after modifications to the servo value had been made), the calculated number of events proved

to be too small. Our calculations showed that one circuit of the course should generate 1053 events, but approximately 1200 events were being generated. The cause for this change is unknown as no modifications were made to the robot.

In conclusion, our first attempt at the demo failed as result of a low battery and the servo angle not being too small. Since the software required the encoders to signal completion, the robot did not shut down because the encoders were not functioning because the battery was low. This particular battery has charging problems from the start. The other problem was the servo angle being too small. For an unknown reason, the angles we tested and got to work were not sufficient to put the robot within an inch of the third box.

The first attempt at the demo failed as a result of a low battery and an incorrect servo value, which was previously discussed. The battery had proved to be a concern earlier in testing as we encountered difficulty in charging it and getting it to keep the charge. Since the turn radius was too large, the robot did not travel over the specified squares and therefore the line traversal sensor was unable to stop the robot. The second attempt failed as a result of an incorrect servo value and an incorrect calculation of the events required to complete the course. Since the servo value had proved to be incorrect in the previous attempt, the decision was made to change the two requirements for completion, namely crossing a required number of black lines and traveling a desired distance, be changed from an AND requirement to an OR requirement. Since the event count that would measure the distance traveled was too small, the robot halted before it had completed its third circuit of the course.

Amit Maole
Brent Eskridge
Klo Utley
Tony Lopez

Team Organization Evaluation and Plans

Organization Evaluation

Before attempting to complete Project 1, Team 10 decided to organize the team into six main divisions (Design, Hardware, Software, Testing, Management, and Presentation). Design and Presentation were to be team tasks, while the other four would be assigned to two group members. After completing Project 1, it has been decided that the organizational structure of the team is more than sufficient and works quite well. In instances where one team member was unavailable, the other member assigned to that task picked up for him. All tasks were accomplished on time except for hardware design which was delayed because of insufficient parts. The division of labor is one aspect of the organization that will be kept for Project 2.

One suggestion for change is to incorporate more milestones, with each milestone having a corresponding demonstrable goal (i.e., recognize a black stripe, make a 90 degree turn, etc.). This will allow the progress of the team to be tracked to a greater degree of accuracy. Another suggestion is to meet more frequently. During this first project, the team was adjusting to learning the member's schedules and it appeared that meeting more often would be beneficial. This would allow for greater communication between team members.

Lessons Learned

The first project always holds many surprises and this one was no different. Team 10 learned that a fully functional working battery is very important. A failing battery would cause the servos and encoders to fail. During coding, IC does not check

type mismatches in programs. It will crash if it encounters one during compiling. Additionally, servo positions are not 100% static from test to test. Testing is one of the most important phases of the project where many unexpected things could happen that could not be foreseen during the hardware or software phases. The initial design will be thrown away, but is useful for learning. In the future, group 10 will have a quick prototype phase to help get a good grasp of the problems in solving the task.