Project 1 – Sensing and Movement

Group 1
Tim Hunt Manohar Pavuluri Adam Heck Romain Pradeau

Dr. Dean Hougen
CS 4970 / 5973
Introduction to Intelligent Robotics
Spring 2003

University of Oklahoma
Norman, Ok 73019

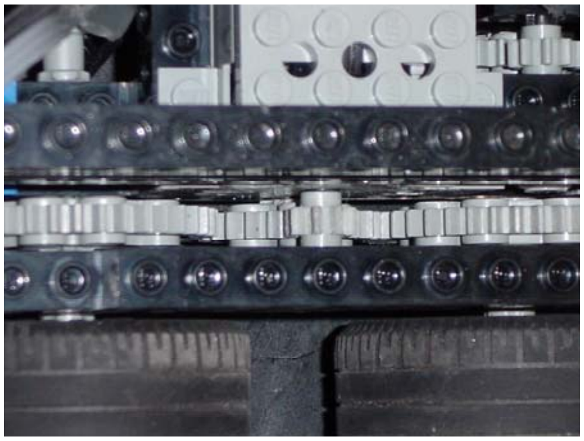
The project requirements served as the guidelines by which our design requirements were developed for the robot. The course presented seemed fairly simple. Thus, the team opted for a fairly simple approach to the construction and programming of the robot.

Robot Design

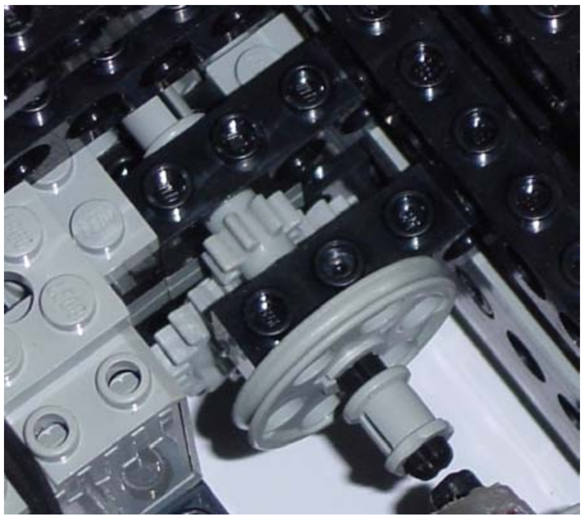
Three initial concepts lead to the development of the presented robot. Each of the prototypes seemed to give some insight to advantages and disadvantages of various robot bases and drive trains. These insights were ultimately used to construct a robot with the characteristics that we desired.

Chassis and Drive Train

The final robot design was a small, wheeled robot with a short and wide wheelbase. Two gray Lego motors powered the drive train of the robot. These motors were set up to control a particular pair of wheels on either side of the robot. The wheels were driven by a set of gears that extended from the motors. Each motor began by supplying power through an 8-tooth Lego gear. This gear immediately meshed up to an idler gear that had 24 teeth. The idler's purpose was to split the incoming power to two wheels on the same side of the robot. In addition, the idler also synced the two wheels to a have common rotational direction, rpm, and torque. Each wheel of the robot had a 40-tooth Lego gear. All together this gear system dropped the rpm of the wheels to a factor of 1/5th of the actual motor rpm. Likewise, the torque was increased by 5:1 to each side of the robot; splitting this between the two wheels is an increase in torque of 250% at each wheel. The gear system can be seen in picture to the right. The gears of the robot were placed between two black Lego beams. This method of mounting prevents play in the gears that could allow binding. Binding in the gears would create addition stress on the motor; therefore reducing that risk will release the burden of required compensation by the software. The additional black Lego beams also ease stress on the axles of the robot. By implementing a wider mount for the axles, some of the bending moment can be reduced. The reduction allows the axle to spin with less rotational friction.

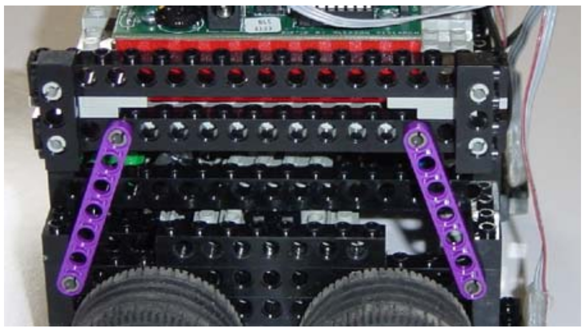


In a similar manner, a gear system was set up for the encoder wheels. On the interior of the robot, a 24-tooth Lego gear was mounted to the axle of a drive wheel, one per side of the robot. This gear meshed up to an 8-tooth Lego gear that drove an axle holding the encoder wheel. This two-gear system would spin the encoder at three times the rpm of the driven wheel. Using a small Lego pulley with six holes, an encoder will count 12 ticks for one complete revolution of a driven



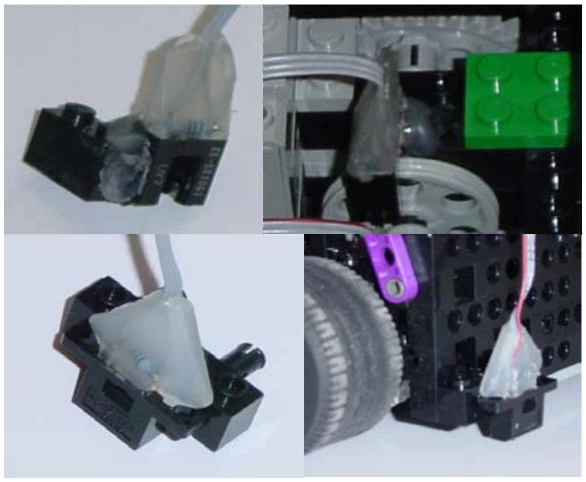
wheel. Together, the gears system and the encoder wheel generate 36 ticks for a single wheel revolution. This resolution breaks down to one tick being approximately 3/16th of an inch.

Two separate gear trains were set up on opposite sides of the robot. Additional black Lego beams spanned the front and rear of the robot. These cross beams combined with the housings of the gear systems made up the base of the final robot. Beams in the corner sections were staggered to mimic the corner of a brick wall. The beams were stacked to a height that rose just above the tops of the Lego motors. Only a cradle for the handy board arose from the robot base. To ensure a solid mount to the robot base, diagonal members reached up from the base and attached to the sides of the cradle.



Sensors

A total of four items required the use of hot glue. These four items were the sensors of the robot. Two slot sensors and two reflectance sensors were affixed to Lego bricks. The slot sensors were attached to the end of a short Lego brick such that when the brick was placed next to the encoder wheel, the sensor would straddle the encoder wheel. In a similar manner, the reflectance sensors were attached to the middle of a short Lego brick. Then using two black Lego pins, the beam was attached to the lower front edge of the robot base. The placement was in the outer most holes possible. This was done to allow the robot to have as wide of a field of view as possible.



Robot Code

The first big requirement for the robot was to possess the ability to drive in a straight line. If it could be guaranteed that the robot would drive in a straight line, then the only concern would be to point that line in the right direction and travel the correct distance. To aid the robot in driving straight, encoders were used on each of the two motors. When the computer detected a slip, both motor speeds would be adjusted with a correctional value. This value was half of the difference in the encoder values for each side of the robot. If the right encoder had 20 ticks more than the left encoder at the time of reading, then the right motor speed would be reduced by 10 while the left motor speed was increased by 10. As the encoders stabilized the signaled speed to the motors, the robot would establish a straight trajectory. An important note, if the resolution is too high on the encoders, a slight drift in the robot can cause high differences in the encoder values. If these high differences go unbalanced, then they can cause dramatic effects using this method of correction. Initially, to counter act this effect, the encoders were never allowed to grow greater than 300. However, the robot had to travel approximately 750 ticks to traverse the required six feet. To solve this problem the code implemented a counter to track the number of times the encoders were reset to zero. Then the value of 300 was experimentally trimmed down to a value that produced the correct distance when looped three times, approximately 233. A second function aided the robot in driving straight. A function called ramp-up slowly increased power to both wheels at the same time. This prevents a sudden surge of power to the wheels that caused the robot to rock backwards when it begins its straight run. After a calculated distance was traveled, the robot would execute a clockwise turn of 90 degrees. This turn was generated using the encoders. One motor was fired in the opposite direction of the second. This produced a turning motion that is near a zero turning radius. The encoders watched for both motors to reach a specified value. Once a motor reached the preset value, it was turned off and waited for the other motor to finish before progressing. While testing, it was seen that after the execution of a turn the front reflectance sensor were terrible close to the edge of the target square. This usually occurred if the robot had drifted slightly to the right during travel from one square to the next. Therefore, a short function calls the robot to back up slightly before continuing through the algorithm. The robot then travels to the first line it sees and squares up to the line. After each major movement of the robot a breaking function is fired. This prevents the robot from coasting to an undesired position. The breaking function cycles the motor commands between an arbitrary +/- values. When the robot is square with the line it is ready to cycle through the code again. A counter keeps track of the number of squares that the robot enters and exits the program at the proper time.

The plan behind the code called for the robot to NOT miss the box. This was seen as a simple assignment for group members to become acquainted with the kit contents. With this in mind great lengths were traveled to maintain a simple strategy. The robot did not have the ability to search for a missed box. Also, the robot did not watch for whether it was squaring up on the back line or front line of a square while preparing to travel to the next square. Instead, we took the time to build a solid base for the robot and then designed code around the mechanical reactions of the robot that were observed during testing.

Team Organization Evaluation and Plan

For project 1 our group adopted a democratic style for team organization. Tasks were documented for project success. Then these tasks were lumped together based upon there commonality with other supporting tasks. Overall, three main areas arose for the project: software, hardware, and documentation. At the start of the project, our team had three members. This seemed to match up to the three main areas that the project required. The group had a couple of brainstorming sessions early on concerning the allocation of the tasks. It was understood that we would cater to weakness on the team. Not all of the group members had great knowledge of programming intelligent systems. So, a stronger programmer took on the tasks of hardware design. This would allow them to take on more challenging software assignments as the semester progressed. Based on this allocation of talent, the next decision was based on preference to task sets. After settling on specific areas of responsibility, the group focused on discussing whether or not the current allocation of tasks was fair. To preserve the democracy, the final stages of debugging and testing were set a side as group responsibilities.

The method we used got the project finished. It’s hard say if it was the best method due to the small size of the group. However, some things can be pointed out. At times it seemed as though we might be getting too much cross pollination in the form of one member influencing another’s tasks, something which we were all guilty of. This ultimately served as a check on the robot development. Software and hardware designs had become a little elaborate. The cross pollination that occurred at the time of debugging resulted in the simplification of the robot.

Tasks were met in a timely fashion. The only missed time frame was the design freeze date. At the time when that milestone arrived, we had what we believed to be a solid base. However, we weren't actively testing the code and the robot together. Once we started testing, we found some inconsistencies in the performance. Seeing how the code was fairly simple, it was decided that mechanical issues had to be the cause. Trying some variations to the original design confirmed our suspicions and new platforms were sought. This led us to different prototypes that resulted in the formation of our final design. The decision to change bases was difficult to make considering the timetable. However, it greatly reduced the amount of time required to debug the code once we had a repeatable mechanical system.

The biggest lesson learned for project 1 was the ability to recognize when a task is pushing a project past its scope. Considering that all of us were present at the time that it happened, we have all developed a respect for how dangerous it can be. Each of us will be more aware if it happens again. This new awareness, our small size, and our success lead us to believe that our current set up in organization can be a successful plan in the future.

The plan for the next project is to maintain a similar organization. We will rotate the members to new responsibilities. Above all, focus on the allocated tasks and maintain the project within the scope presented without making it any more difficult.

Results of Demo

The robot preformed the required tasks on the bonus course during the in-class required demonstrations.

Appendix A: Robot Code

```
////////////////////////////////////////////////////////////////
//  Project 1
//   Team 1
//
//  Code to make our robot ("Kirby") go in a straight line
//  for six feet, turn 90 degrees, line up for the next box,
//  and then repeat until 3 circuits are made.
////////////////////////////////////////////////////////////////

#define thres 90          // set a threshold of reflectance to check against
#define left_enc 1
#define right_enc 0
#define left_motor 2
#define right_motor 0
#define ticks_2_turn 34    // number of encoder counts to turn for
#define front_line analog(2)

int count = 0;            // boxes traveled to
int leftspeed = 80;       // initial motor speeds
int rightspeed = 80;
int enc0, enc1;
int resets = 0;           // resets * ticks_2_dist is the distance traveled
int ticks_2_dist=198;

void main()
{
   int col;

   alloff();
   printf( "Project1!\n" );

   // Wait for the start button to be hit, and allow
   // time for the user to get away
   while( !start_button()){
      printf("\n%d", knob());
   }
   ticks_2_dist=knob();
   sleep(2.0);

   getSet();
   printf("I am starting off!!");
   rampup();
   sleep(0.1);
```

```
    // start counting distance
    enable_encoder(right_enc);
    enable_encoder(left_enc);
    while(!stop_button())
      {
        enc0=read_encoder(right_enc);
        enc1=read_encoder(left_enc);
        printf("\n%d, %d ", enc0, enc1);

        // see if we've gone 6 feet yet...
        if(resets>=2)
          {
            count++;
            break_all_motors();
            sleep(0.2);
            if(count>=12)  // if we've made three laps, end this
              return;

            //now we turn until our encoders reach a certain value
            motor(0, -70);
            motor(2, 70);
            reset_encoder(right_enc);
            reset_encoder(left_enc);

while(((enc0=read_encoder(right_enc))<ticks_2_turn)&&((enc1=read_encoder(left_enc))<ticks_
2_turn))
              {
               if(enc0>ticks_2_turn)
                 motor(right_motor,0);
               if(enc1>ticks_2_turn)
                 motor(left_motor,0);
            }

            // back up so the sensors are where the robot's center was
            break_all_motors();
            motor(0, -60);
            motor(2, -60);
            sleep(0.5);
            break_all_motors();

            // allign with the front stripe of the box
            sleep(0.25);
            getSet();
            sleep(0.2);
            // go again
            rampup();
            reset_encoder(right_enc);
            reset_encoder(left_enc);
            resets = 0;
        }

        // So we don't have a problem with rollover, break the
        // 6ft distance into chunks and periodically reset the encoders
        if((enc0>ticks_2_dist)||(enc1>ticks_2_dist))
          {
            reset_encoder(right_enc);
            reset_encoder(left_enc);
            resets++;
        }

        corr_err();

        // if we've made three laps, we're done
        if (count==12)
          {
            alloff();
            return();
        }
    }
    alloff();
}

////////////////////////////////////////////////////////////////
// void rampup()
//    slowly raise the motors to desired power to reduce jerking
////////////////////////////////////////////////////////////////
void rampup()
{
    int j;
    for(j=1; j<=8; j++)
      {
        sleep(0.1);
        motor(right_motor, (int)(rightspeed*j)/8);
        motor(left_motor, (int)(leftspeed*j)/8);
    }
}

////////////////////////////////////////////////////////////////
// void corr_err()
//    determine if motor speeds need to be adjusted
//    based on encoder counts and then adjust them
////////////////////////////////////////////////////////////////
void corr_err()
{

   int diff = enc1-enc0;// positive is right drift
   int half_diff=(int)(diff/2);
   // adjust motor speeds slightly if encoder counts are off
   if(diff > 0)
     {
      printf("right drift");
      rightspeed+=half_diff;
      leftspeed-=half_diff;
      motor(right_motor, rightspeed);
      motor(left_motor, leftspeed);
   }
   if(diff < 0)
     {
      printf("left drift");
      leftspeed-=half_diff;
      rightspeed+=half_diff;
      motor(right_motor, rightspeed);
      motor(left_motor, leftspeed);
   }
}

////////////////////////////////////////////////////////////////
// void break_all_motors()
//   oscillates commands to the motors, causing them to
//   lock up and halts movement quickly
////////////////////////////////////////////////////////////////
void break_all_motors()
{
   int j;
   for(j=1;j<30;j++)
     {
      motor(0,-50);
      motor(2,-50);
      motor(0,50);
      motor(2,50);
   }
   ao();
}

////////////////////////////////////////////////////////////////
// void getSet()
//   uses the refectance sesors to allign the robot
//   with the next black line it sees.
////////////////////////////////////////////////////////////////
void getSet()
{
   int ana0, ana1;
   while(1)
     {
      ana0=analog(2);
      ana1=analog(3);
      // if both sensors see black, call that o.k.
      if((ana0>thres)&&(ana1>thres))
        break;

      // now, for each motor, if it's corresponding
      //  sensor sees black, go backward.  if not,
      //  go forward.  The oscillation causes it to
      //  rapidly line up correctly.
      if(ana0>thres)
        motor(0,-40);
      else
        motor(0,40);

      if(ana1>thres)
        motor(2,-40);
      else
        motor(2,40);

   }
   break_all_motors();
}
```