CS 670 Autonomous Mobile Robots
Team 4
Final Report
12/15/2005

Justin Ellis
Tony Morelli
Amandeep Sohal

OBJECTIVE:

For the final contest our robot is taking part in the Robo Golf Contest. This contest consists of a white playing field with several golf balls placed throughout the field. In the center of the field is a large black circle which acts as a goal. The robots are paired two at a time and each competition lasts two minutes. The object of the contest is for the robot to collect golf balls, and leave them on the black circle in the center of the playing field.

STRATEGIC DESIGN:

The design of our robot consisted of three behaviors: obstacle avoidance, wandering, and ball release. Each of these are described in detail below.

Obstacle Avoidance:
The obstacle avoidance was designed to operate in the playing field without getting stuck. We wanted our robot to behave correctly when it encountered either the walls or an opponent. The obstacle avoidance utilized two touch sensors located in the front of the robot. When one of the sensors was tripped, the robot would stop, close the gate to protect the balls it held (if any), reverse, then turn to the side of the robot opposite of the sensor that was tripped. With this design our robot would safely wander the playing field without getting stuck.

Wandering:
The wandering was designed to cover the most of the playing field possible. Basically the robot would move forward until it hit the boundary of the playing field, then it would turn at a random rate as to not get stuck covering the same portion of the playing field over and over again. Obstacle avoidance will take over when obstacles are encountered, or when the goal is found, other than that the robot will wander continuously. The wander is not only designed to cover the entire playing field with hopes of collecting golf balls, but it is designed to eventually find the large black circle in the center of the field which will lead us to the ball release state.

Ball Release:
The ball release was designed to leave the golf balls held by the robot in the large black circle located in the center of the ring. The robot activates this behavior when the light sensors detect black. The sensors are located near the center of the robot on the sides. Once black is detected, the ball release behavior is activated and it performs the following tasks. First it must determine which sensor is detecting black, and turn appropriately such that both sensors are over black. At that point, the robot moves backwards with the gate open which will release the balls onto the black circle. Once the backup has completed, the robot should turn around and go back to wandering.

Error Detection:
This is not really a behavior, but or robot is equipped with error detection. Error detection was implemented as follows. All tasks were timed. When the task being performed took longer than the timeout for that task, the task was aborted, the robot would turn in a random direction, then continue wandering. This was implemented due to two circumstances. The first deals with the locating of the black circle. The cruising

speed of our robot was so great that once it detected the black circle, it could not decelerate, and stop quickly enough to remain on the circle. Once this occurs the robot will search for the circle as it knows it is near it. If there was no error detection, and the robot was searching for the circle and another robot crashed into the robot throwing it off course, we could end up in a state where we were searching infinitely. The error detection will cause our robot to give up if it cannot find the circle within 5 seconds. The other scenario deals with a hardware issue. It is possible (although not likely) that our robot could get wedged on the side of the arena. So if the robot thinks its going straight for more than 5 seconds, we determined that we are stuck, so turn randomly to get out of that state.

PROBLEMS ENCOUNTERED:

Most of our software was already tested in our other labs; we simply pieced it all together to create the functionality we needed. The majority of the problems we encountered were hardware related. We began with the robot we used in the harvesting contest. In this contest, the robot was to wander the arena and beep when it discovered a black puck. This was fairly similar to what the goal of this contest was, so we started from there.

The first hardware issue we had was that our robot was high enough off the ground to collect golf balls – our clearance was smaller than the height of a golf ball. To fix that we raised the base of our robot up about an inch. This allowed for the balls to be collected underneath the robot. Fortunately the balls would get caught under the robot, but unfortunately by raising the robot we also had to raise the touch sensors.

With the touch sensors located above the height of the arena wall, we needed to extend them down in such a way that the robot would correctly identify when it encountered a wall. Unfortunately our attempts at keeping our existing touch sensor design all failed. Although it looked like it should work, the plastic Lego pieces were too flexible. Our design had the two touch sensors attached to each other, and with the extensions on them, when one sensor needed to be triggered, it would flex with the other one which caused neither of them to trip. This resulted in our robot driving into a wall and then just sitting there as it did not know it was stuck. We redesigned the sensors to operate independently of each other and mounted them on the side of the robot and low enough to touch the arena wall without much additional materials. This allowed for our robot to accurately determine when it encountered an obstacle.

Another hardware problem we encountered was the gate. The gate needed to be closed when backing up to keep the collected golf balls under the robot. The first design was a swinging gate that would swing counter-clock wise to open and clock wise to close. Although this design looked impressive, it had issues. Mainly when the gate was closed, it was not strong enough to hold the balls under the robot. Reversing the robot caused the balls to put pressure on the swinging gate and the plastic lego pieces were too flexible and would bend, allowing some balls to escape!

So we thought of another design which was more robust and worked better than our first thought. We mounted a motor at the left hand side of the robot and mounted a wedge shaped gate on the motor shaft. So when the motor turned clockwise or anticlockwise it made the gate open or close respectively.

UNRESOLVED ISSUES:
One difference we have noticed between our robot and the others is that we don't have the ball holding capacity of the others. This should not be an issue as we seem to find the black circle once every 20 seconds. With this frequency and the ability to hold 4 balls at once, we felt this would make our robot very competitive.

APPENDIX A – Source Code

```
/***********************************************************************
****
  CS674 Team #4 Final Project
  Members:
       Justin Ellis
       Tony Morelli
       Amandeep Sohal
  12/15/2005
************************************************************************
***/

int avoiding = 0;  //global variable; 0 for not avoiding, 1 for avoiding
float lastTurn[4];
float bump_time = 3.0;
int range = 0;
int initValue[2];
float inc;
int RIGHT = 3;
int LEFT = 2;
float startTime;
int LEFTFORWARD = 70;
int RIGHTFORWARD = 70;
float waitForRelease = 0.0;
float STUCK_TIMEOUT = 5.0;
int timeSinceLastTurn = 0;

int idleTime = 0;

int invertTurn = 0;


/***********************************************************************
*
Function: main()
Parameters: None
Purpose: Main loop - Keep Robot Moving
************************************************************************
```

```
/
void main()
{
   initValue[0] = analog(5);
   initValue[1] = analog(6);
   while ( !start_button() ) { }


   motor(1,LEFTFORWARD);
   motor(3,RIGHTFORWARD);

   while ( !stop_button() )
   {
      check_sensors();
      motor(1,LEFTFORWARD);
      motor(3,RIGHTFORWARD);
      printf("%d\n", timeSinceLastTurn);
      if (timeSinceLastTurn > 300)
      {
         turn_random();
         timeSinceLastTurn = 0;

      }

   }
   off(1);
   off(3);

}

/*********************************************************************
*
Function: check_sensors()
Parameters: None
Purpose: Check sensor readings.  If we have tripped a touch sensor,
       behave accordingly to avoid the obstacle.  If either of the
       reflective sensors have reached the threshold, release the balls
*********************************************************************
/
void check_sensors () {

   int a = 0;
   int iDidSomething = 0;

   if (analog (RIGHT) > 150 || analog(LEFT) > 150)
     {

      tone(600.0,.25);
```

```
    ReleaseBalls();
    iDidSomething = 1;
  }


  /* test for touch on sensor 10 */
  if (digital(10))  {
    avoiding = 1;  //robot is avoiding

    if (a >= 4)
      a = 0;
    lastTurn[a] = seconds();
    a++;

    turn_left((float)random(50)/100. + .1);
    invertTurn = 1;
    avoiding = 0; //robot not avoiding
    idleTime = 0;
    iDidSomething = 1;

  }

  /*  test for touch on sensor 11 */
  else if (digital(11)) {

      avoiding = 1;  //robot is now avoiding
      if (a >= 4)
        a = 0;
      lastTurn[a] = seconds();
      a++;

      turn_right((float)random(50)/100. + .1);
      invertTurn = 1;


      avoiding = 0; //stop avoiding
      idleTime = 0;
      iDidSomething = 1;

}
if (iDidSomething == 0)
{
   timeSinceLastTurn ++;
}
else
{
    timeSinceLastTurn = 0;
}
```

```
}

/*******************************************************************
*
Function: ReleaseBalls()
Parameters: None
Purpose: Center robot over black circle, open gate and back up to leave
        balls in center of circle.
********************************************************************
/
void ReleaseBalls()
{
   int found = 0;

   motor(1,40);
   motor(3,40);

   CloseGate();

   waitForRelease = seconds();
   motor(1,-40);
   motor(3,-40);

   while (analog (RIGHT) < 150 && analog(LEFT) < 150 && ((seconds() -
waitForRelease) < STUCK_TIMEOUT))
   {
      printf("BACKUP\n");

   }
//   sleep(0.5);

   off(1);
   off(3);

   waitForRelease = seconds();

   if (analog(RIGHT) > 150 && analog(LEFT) > 150)
     {
      //dont do anything, just back up...
   }

   else if (analog(RIGHT) > 150)
      {
       printf("RIGHT\n", analog(LEFT));

       motor(3, RIGHTFORWARD);

       while((found == 0) && ((seconds() - waitForRelease) < STUCK_TIMEOUT))
```

```
            {
             if (analog(LEFT) > 150)
               {
                 printf("%d\n", analog(LEFT));
                 found = 1;
               }
            }
          off(3);
      }
      else if (analog(LEFT) > 150)
          {
           printf("LEFT\n", analog(RIGHT));


           motor(1,LEFTFORWARD);

           while((found == 0) && ((seconds() - waitForRelease) < STUCK_TIMEOUT))
             {
              if (analog(RIGHT) > 150)
                {
                  printf("%d\n", analog(RIGHT));
                  found = 1;
                }
             }
           off(1);
        }
        sleep(1.0);
    OpenGate();

    motor(1,-40);
    motor(3,-40);

    sleep(2.0);
    turn_random();

}
/**********************************************************************
*
Function: back_up()
Parameters: time
Purpose: Move the robot backwardds for passed in t
**********************************************************************
/
void back_up(float sleep_value)
{

    bk(1);
    bk(3);
```

```
   sleep(sleep_value);

}
/**********************************************************************
*
Function: turn_right()
Parameters: time
Purpose: Turn robot right for time passed in t
**********************************************************************
/
void turn_right(float sleep_value)
{
   CloseGate();
   off(1);
   off(3);
   motor(1,-70);
   motor(3,-70);

   sleep(.25);  //keep going backwards for sleep_value seconds

   motor(1,70);
   sleep(sleep_value); //turn for sleep_value seconds
   motor(1,LEFTFORWARD);
   motor(3,RIGHTFORWARD);
   sleep(0.1);
   OpenGate();
}
/**********************************************************************
*
Function: turn_left()
Parameters: time
Purpose: Turn the robot left for time passed in t
**********************************************************************
/

void turn_left(float sleep_value)
{
   CloseGate();
   off(1);
   off(3);

   motor(1,-70);
   motor(3,-70);
   sleep(.25); //keep going backwards for sleep_value seconds

   motor(3,70);

   sleep(sleep_value); //turn for sleep_value seconds
   motor(1,LEFTFORWARD);
   motor(3,RIGHTFORWARD);

   sleep(0.1);

   OpenGate();
}
/*********************************************************************
*
Function: turn_random()
Parameters: None
Purpose: Turn the robot in a random direction for a random amount of time
**********************************************************************
/
void turn_random() // turn a random direction for a random amount of time
{
   int i = 0;
   printf("Random Turn\n");

   if (random(2) == 0)
     turn_left((float)random(100)/100. + .5);
   else
     turn_right((float)random(100)/100. + .5);
   //initialize all lastTurn array values to zero
   for (i = 0; i < 4; i++)
     lastTurn[i] = 0.0;
   printf("\n");

}
/*********************************************************************
*
Function: OpenGate()
Parameters: None
Purpose: Open the collection gate.  With the gate open the robot can
        collect balls
**********************************************************************
/
void OpenGate()
{

   motor(2, -50);
   sleep(0.5);
   off(2);


}
/*********************************************************************
*
```

```
Function: CloseGate()
Parameters: None
Purpose: Close the collection gate.  With the gate closed the robot can
        hold balls
*********************************************************************
/
void CloseGate()
{
   motor(2, 50);
   sleep(.25);
}

/*********************************END****************************/
```