Sam McCauley and Caitrin Eaton
Project 2: SumoBot
Final Report

PROBLEM

Our goal was to design an autonomous vehicle that could compete with a human-operated remote-controlled (RC) car in a sumo wrestling match. This requires a robotic vehicle capable of sensing impact on every side, distinguishing between frontal impact and impact to the side or rear, and detecting the sumo ring's boundary line, reacting to these stimuli in an intelligent manner.

The robot has to display intelligent behavior by attempting to push the RC vehicle out of the ring while being careful not to leave the ring itself. Balancing these goals is the key to the game. The robot must rely on limited sensory data (*very* limited compared to the human controlling the other vehicle), and must react as quickly as possible to changes in the environment (e.g. proximity to the boundary line and the decisions of the human opponent). A tight sensory-motor loop is required.

MOTIVATION

We believed it would be interesting to see how well our autonomous vehicle (SumoBot) could do against a human opponent with an RC car. The human possess superior planning and sensing abilities, but is hampered by a laggy control loop (human reaction time plus handicap for RC interface) and poor maneuverability (the robot has a dual differential drive, while the only RC car we could find in our price range turns like semi with 18 flat tires).

CHANGES MADE SINCE THE PROJECT PROPOSAL

The original proposal included a ring made of white poster board with black and grey circles (made of electrician's tape and duct tape) defining the "out of bounds" and "warning" boundaries. A vehicle could win by knocking its opponent on its side or pushing it onto (or over) the black outer boundary. A vehicle could also lose by falling or wandering outside the ring on its own.

The project was simplified by reducing the ring requirements to include on one boundary line (made of electrician's tape). This was made necessary by the LEGO light sensor's inability to distinguish consistently between white poster board and duct tape, which is reflective.

The ultrasonic (US) sensor was also removed from the design due to time constraints. The touch-sensor responses were still being debugged the day before the demo, so we decided it would be best to scale back the scope of the project. The purpose of the US

sensor was to allow the robot to locate its opponent when its touch sensors weren't active. Though information of this sort would make more competitive wrestling tactics available to the robot, it was decided that this was not completely necessary. The human can initiate contact. The main goal is to evade and fight back.

HARDWARE DESIGN

- Front bumper connected to two paired touch sensors tell the robot if it's been impacted from the front.
- "Back" bumper covers the left, right, and rear of the robot. It uses four stacked touch sensors to tell the robot if it has been hit from any side other than the front.
- Light sensor in active mode for detecting boundary lines.
- Differential drive.

See Figure 1 for a labeled picture of the vehicle.

The competition ring was made of white poster board, with an electrician's tape boundary line. Figure 2 shows the SumoBot in the competition ring.

Figure 1. The SumoBot!

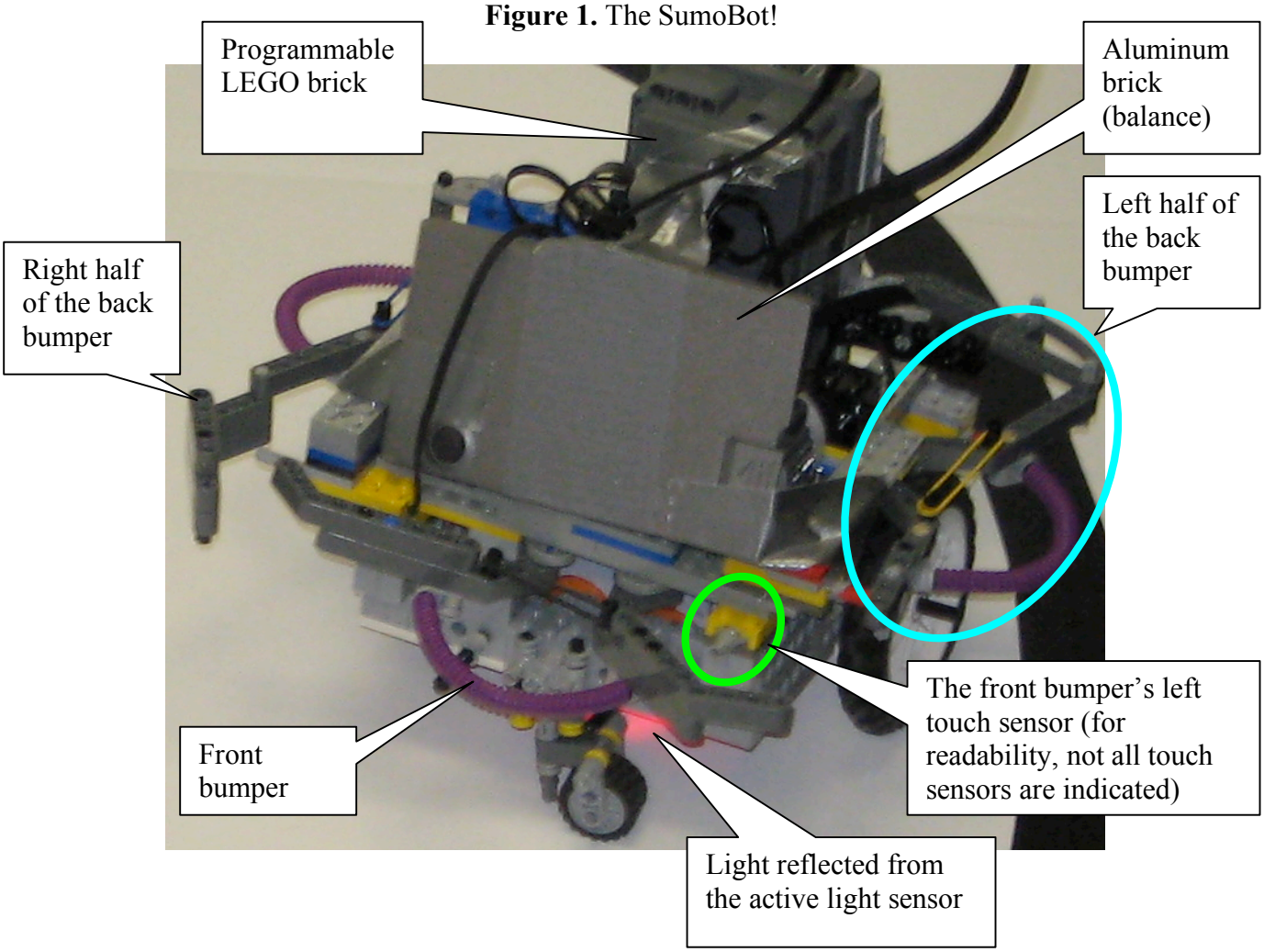
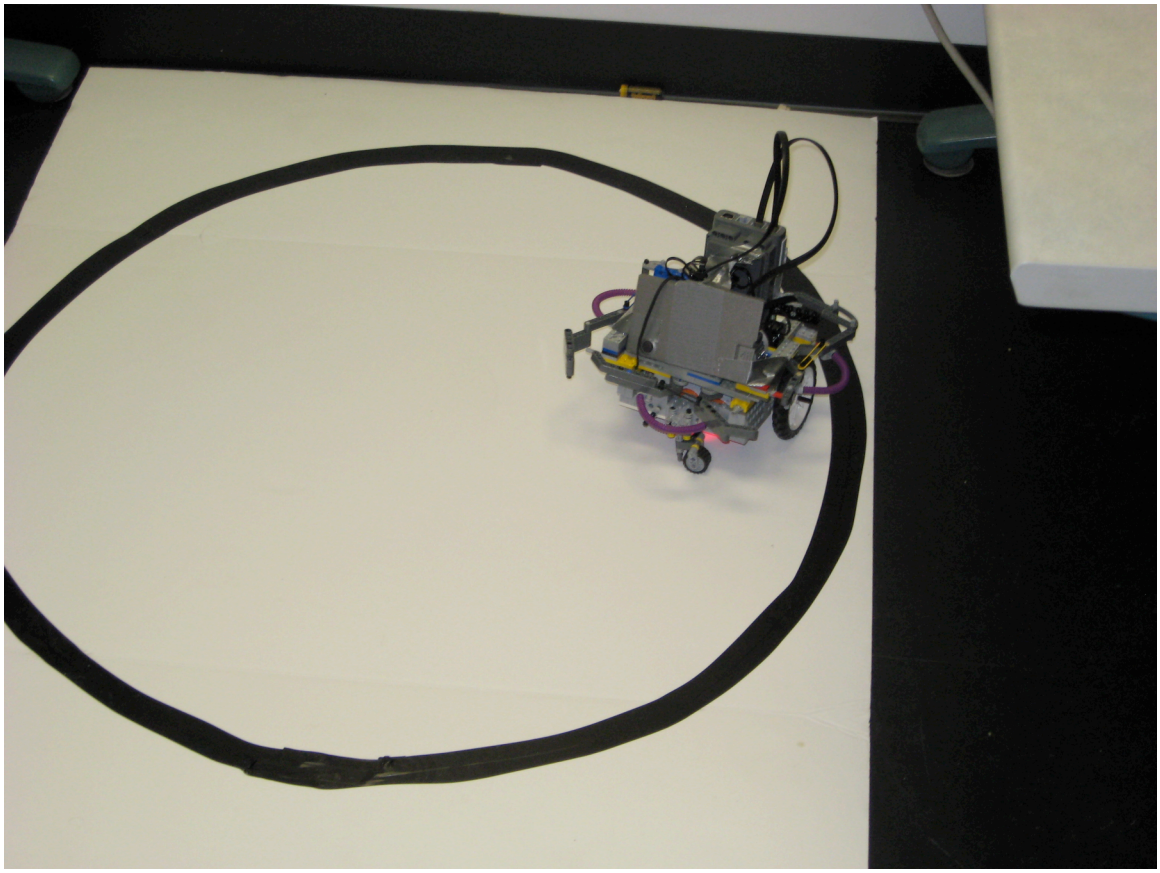


Figure 2. SumoBot in the competition ring



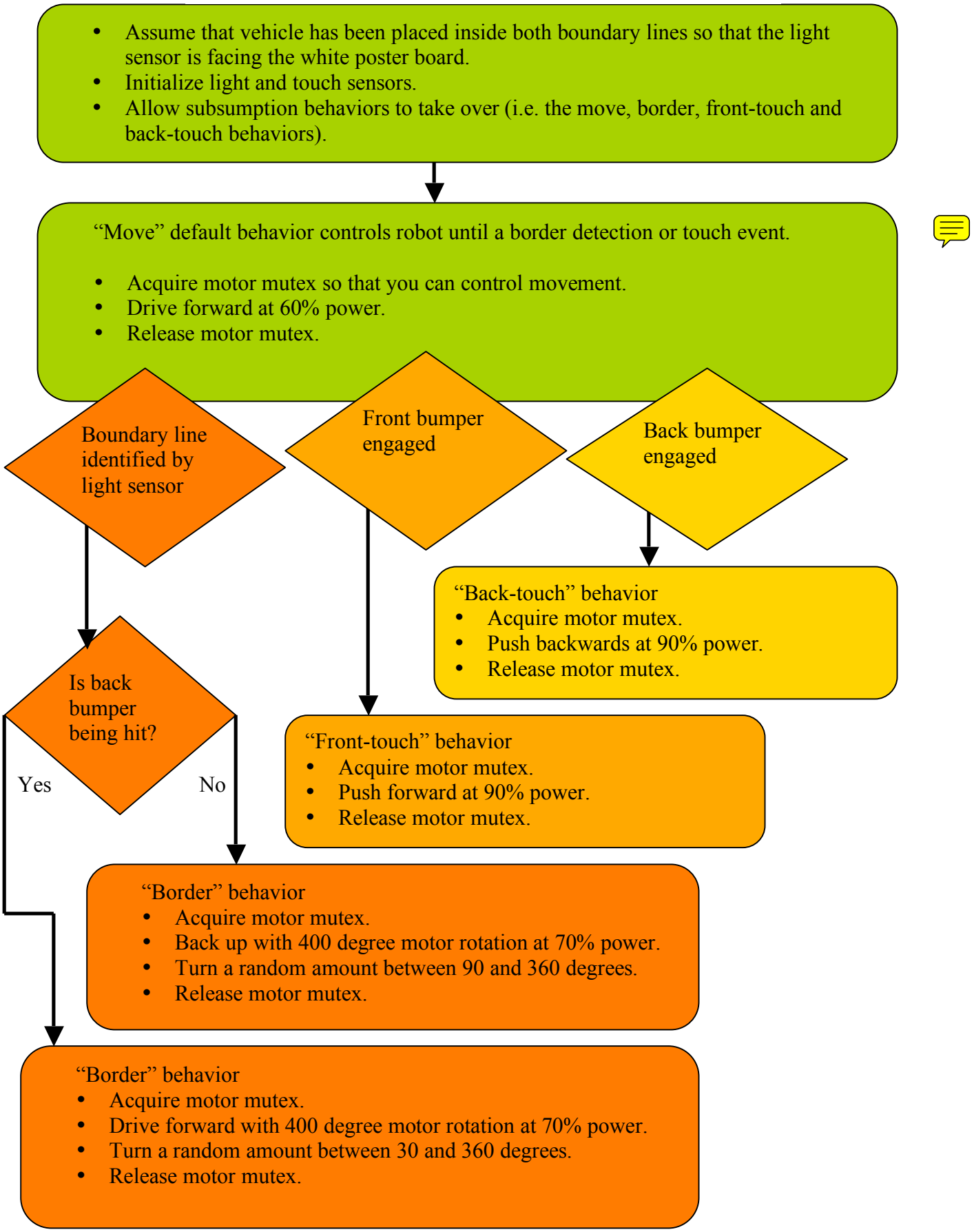
THE ALGORITHM

Subsumption architecture was used to integrate default behaviors with responses to interrupts like a touch-sensor activation or border detection.

The flow chart in Figure 2 describes the robot's behaviors and their priorities.

- Green boxes show the robots behaviors as executed without any interrupts.
- Orange diamonds describe the events that could interrupt those basic behaviors. Interrupts are prioritized from left (dark orange = highest priority) to right (light yellow = lowest priority).
- Orange boxes describe reactionary behaviors corresponding to each interrupt.

Figure 2. Flowchart of SumoBot's competition algorithm



TESTS AND ADJUSTMENTS

During testing of the original design, we came across a fundamental problem with the SumoBot's hardware. First, the placement of the motors and brick resulted in a center of mass well to the rear of the vehicle's geometric center. This caused dramatic tipping whenever something hit the vehicle's front bumper or whenever the motors experienced a sharp stop or start. This was a major design flaw both because the vehicle was prevented from maintaining stability while pushing against an opponent and because movement with the high motor power required for a battle of momentum caused the robot to tip over. Attaching an aluminum brick (approx. 5" x 2.5" x 0.5") to the front of the vehicle helped the issue a great deal, but the vehicle could still benefit from additional adjustments to weight distribution.

The bumpers were also an issue. While testing the touch-event responses, we found that the bumpers often became misaligned with the touch sensors. This kept the robot from reacting to every impact. We redesigned the bumpers to be twice as thick and added pegs to the touch sensors so that the bumpers wouldn't need to be pushed quite as hard to cause a positive sensor reading. This has helped a great deal.

We also had early problems identifying the boundary line due to our assumption that the light sensor would return consistent values for the same material. We set a constant threshold based on data collected by holding the light sensor over electrician's tape, but didn't take into account that differences in ambient light levels would effect readings even when the sensor was mounted under the robot's body. Whenever the robot started up, it would go immediately into it's boundary-response behavior, even when it was in the middle of the white poster board. In response, we altered the code so that now the vehicle takes an ambient light reading at startup and only enters the boundary behavior if it collects a reading that is significantly lower, specifically the ambient reading minus 30. This seems to have solved the problem completely. We have not had a border-detection error since.

Final code is included at the end of this report.

CONCLUSION

Success! And it's about time.

The robot responds "intelligently" to stimulus from the environment. It responds to frontal impact by pushing back against its opponent, attempts to dodge when it senses impact from the side or back, and above all else reacts to border detection events by trying to stay in the ring.

The vehicle still has trouble pushing hard enough against an opponent to actually move it. This could potentially be mitigated by further improvements to balance, allowing us to

use full power without causing the vehicle to tip (we currently go only as high as 90% motor power). We should also modify the gear ratio to favor torque over speed.

Also, SumoBot's bumpers only allow for opponents in a certain height range. Professor Varshavskaya suggested that our design could be improved upon by building taller, wall-like bumpers. This would allow for a wider array of opponent heights. Thanks, Professor!

Further improvements could be made by adding a more competitive evasive maneuver in response to side or rear impact. Differentiating between side and rear hits would be beneficial, and is possible now that the ultrasonic sensor has been removed (which freed up an input port on the Mindstorms brick). Alternatively, we could improve the robot's ability to compete by reintegrating the ultrasonic sensor. Identifying the opponent's position from a distance would give SumoBot the advantage of being able to orient itself towards the opponent rather than flying blind until impact.

In short, SumoBot performed adequately on demo day, but there are several improvements we would make before entering it in a second exhibition.

ACKNOWLEDGEMENTS

Thanks to Danny, Mark and Denise for all of their help with the bumper design!!! Also, thank you to Professor Varshavskaya for patience, brainstorming, constructive criticism, and a bazillion extra LEGOs.

FINAL CODE

```
//Sam McCauley
//Caitrin Eaton

#define THRESH1  10
#define THRESH2  180
#define WAITCONST 20
#define TURNTIME 1000
#define BACKUP 10006


mutex motormutex; //mutex for the motors
bool following;  //keeps track of whether or not the opponent was detected
int initialreading;


//if the front touch sensor is activated, push forwards
task fronttouch(){
 while(true)
 {
   if(Sensor(S2) > 0)
   {
       TextOut(10,10, "FRONTTOUCH", true);
       Acquire(motormutex);
       Off(OUT_AB);
       while(Sensor(S2) > 0){   //if you go back and something's still pushing,
       OnRev(OUT_A, 90);} //keep pushing instead of going back to move()
       Release(motormutex);
   }
   Wait(WAITCONST);  //this, at the end of each task, makes sure that each task only
                     //samples the sensors once every WAITCONST milliseconds.  Also, since
                     //there are many tasks, it's important that none hogs resources
 }

}

//if the back touch sensor is activated, push backwards.  Nearly identical to fronttouch()
task backtouch(){

 while(true)
 {
   if(Sensor(S3) > 0 && Sensor(S1) >= initialreading - 30)
   {
       TextOut(10,10, "BACKTOUCH", true);
```

```
        Acquire(motormutex);
        Off(OUT_AB);
        while(Sensor(S3) > 0 && Sensor(S1) >= initialreading - 30){
        OnFwd(OUT_A, 90); }
        Release(motormutex);
    }
    Wait(WAITCONST);
 }

}

//if the border is detected, go backwards
//(this is legitimate since the robot can only turn on a dime, so if
//it went off the border, it's from going forward.  With a second light sensor,
//a more accurate behavior could be described, but the touch and US sensors are
//more important.

//TODO:  if the back sensor is also being pushed, go backwards instead of forwards


task border(){

 while(true)
 {

    if(Sensor(S1) < initialreading - 10 && Sensor(S3) == 0)
    {
       TextOut(10,10, "BORDER", true);
       Acquire(motormutex);
       if(Sensor(S1) < initialreading - 30)
       {
        Off(OUT_AB);            //force motors to stop.
        RotateMotor(OUT_A, 70, 400);
        RotateMotor(OUT_B, 70, (Random(90) + 90) * Random(3)*2 - 3);
       }
       Release(motormutex);
    }
    else if( Sensor(S1) < initialreading - 10)
    {
       TextOut(10,10, "BORDER", true);
       Acquire(motormutex);
       if(Sensor(S1) < initialreading - 30)
       {
        Off(OUT_AB);            //force motors to stop.
        RotateMotor(OUT_A, 70, -400);
        RotateMotor(OUT_B, 70, 30+Random(330));
```

```
      }
      Release(motormutex);
    }
    Wait(WAITCONST);

 }
}

// NOTE: THIS FUNCTION IS NOT USED, SINCE THERE'S ONLY ONE
// BOUNDARY LINE. The original design called for a "warning" boundary line within a
// second boundary line that would indicate a loss. Detection of the outer boundary line
// would have triggered this function, but is not used in the new, single-boundary design.

// If, despite the robot's best efforts, it goes off the edge of the ring, then it
// should stop doing everything

task lose(){

while(true)
{
  if(Sensor(S1) < THRESH1)
  {
      Off(OUT_AB);
      TextOut(10,10, "LOSE", true);
      Wait(1000);
      StopAllTasks();    //Stops all tasks
      PlayTone(750, 30);  //Play a sound to signify that the game is over
  }
  Wait(WAITCONST);

}
}

//this is the default behavior, which exists only if there isn't more specific input
//from another sensor telling it where to move.  The robot turns for a bit, then moves
//forward for a bit.  This is randomized so that a) the human opponent has more difficulty
//determining the behaviors of the robot and b) the human opponent has more difficulty
predicting
//the actions of the robot if they determine the behaviors

task move(){

 while(true)
 {
    TextOut(10,10, "MOVE", true);
```

```
    Acquire(motormutex);
    Off(OUT_AB);
    OnRev(OUT_A, 60);
    Release(motormutex);
 }
}

// NOTE: THIS FUNCTION IS NOT USED, SINCE THE US SENSOR HAS BEEN
// REMOVED

// If the robot senses the other robot through its US sensor, it goes forward at a moderate
// pace

task follow(){

 while(true)
 {
     if(SensorUS(S4) < 90 && Sensor(S1) >= THRESH2)
     {
        TextOut(10,10, "FOLLOW", true);
        following = true;
        Acquire(motormutex);
        Off(OUT_AB);
        OnRev(OUT_A, 70);
        Release(motormutex);
     }
     Wait(WAITCONST);
 }
}

// NOTE: THIS FUNCTION IS NOT USED, SINCE THE US SENSOR HAS BEEN
// REMOVED

//if the robot was following the other robot due to a US reading, and the other robot has
since
//dissapeared from the US Sensor, then the other robot must have moved out of the cone.
Rather than
//reverting to the default behavior, which assumes nothing about the state of either robot,
the recover
//task provides a more intelligent reaction to the dissapearance of the other robot by
scanning for it

task recover(){

while(true)
{
```

```
  if(following && SensorUS(S4) > 130 && Sensor(S1) >= THRESH2)
  {
     TextOut(10,10, "RECOVER", true);
     Acquire(motormutex);
     Off(OUT_AB);
     OnRev(OUT_B, 60);
     Release(motormutex);
     Wait(TURNTIME);
     Acquire(motormutex);
     OnFwd(OUT_B, 60);
     Release(motormutex);
     Wait(TURNTIME);
     if(SensorUS(S1) > 90)
     {
       following = false;
     }
  }
  Wait(WAITCONST);
}
}

//the main task, in this program, simply initializes variables and calls the other tasks.
//more complicated behavior would interfere with the purely subsumption-based model

task main(){
 TextOut(10,10, "MAIN", true);
 //initialize sensors
 //SetSensorLowspeed(S4);
 SetSensorType(S3, SENSOR_TYPE_TOUCH);
 SetSensorMode(S3, SENSOR_MODE_BOOL);
 SetSensorType(S2, SENSOR_TYPE_TOUCH);
 SetSensorMode(S2, SENSOR_MODE_BOOL);
 SetSensor(S1, SENSOR_LIGHT);
 SetSensorType(S1, IN_TYPE_LIGHT_ACTIVE);
 SetSensorMode(S1, IN_MODE_RAW);
 following = false;
 initialreading = Sensor(S1); //doesn't do anything
 Wait(1000);
 initialreading = Sensor(S1);
 NumOut(20,20,initialreading, true);
 Wait(2000);
 Precedes(move, border, backtouch, fronttouch);//, recover);  //, follow);   //lose
}
```