Project 3 – Deliberating and Acting

Group 1
Tim Hunt, Manohar Pavuluri, Adam Heck, Romain Pradeau

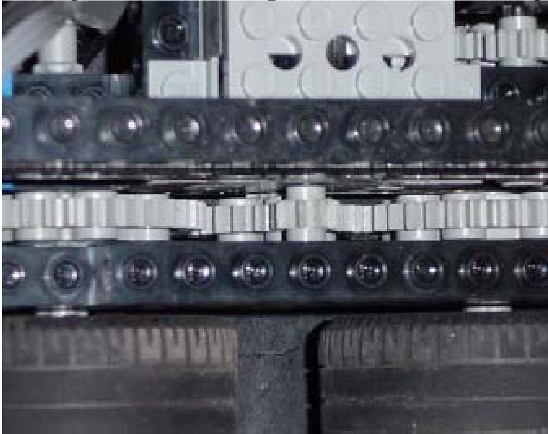
Dr. Dean Hougen
CS 4970 / 5973
Introduction to Intelligent Robotics
Spring 2003

University of Oklahoma
Norman, Ok

Robot design

As in the project 2, in this project also our robot from the first project was stripped down to the drive section and rebuilt with the new sensors required by our strategy. In this project shaft encoder was brought back to get the encoder count while making a turn and traveling a distance.

In this a total of six sensors are used- two slot encoders, one range sensor, two-reflectance sensor and CMU cam. 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. 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.

At front a claw like structure was attached to the robot. To operate this claw i.e. to open and close to grab and release the orange cube a cam was used. It is powered by a servo, which turns the cam to grab and release the target object. The Range sensor is mounted just above the claw on to the chassis, for finding the wall to realign itself incase it looses its orientation with the grid coordination. The servo was glued on to a Lego brick and placed inside the chassis. And the CMU cam was placed on the top front side of the robot. It was glued to a Lego brick at an angle facing down in such a way that it can have a good view of the target object in front of it.

Robot code

The code design was basically a hierarchical. Main functions calling three functions – *startup*, *collect known boxes* and *collect unknown boxes*. And these in turn calls other sub functions. *startup* function initializes all the things on the robot like opening the claw to get ready, enabling encoders, initializing its position and direction with respect to the grid.

Then comes the *collect known boxes* function which carries out the job of collecting the known boxes whose coordinates are known. It carries out different functions in sequence in order to complete the job like *approach target* (finding the

nearest coordinates for a target and approaching it), once its gets to the target if it finds the box it executes the *grab nearby box* function. Then it finds the goal that is closest to the target box by executing *target nearest goal* function then it executes *approach target* function to approach the goal in which the target box has to be dropped and this done by *drop box in goal* function. In case if it doesn't find any target box then it call the *realign* function assuming that it got distracted from its coordinates. In this it tries to locate closest wall in either south or north direction and goes for it. Then using the range sensor it sweeps from one corner to another to get itself realigned with the coordinates.

Now when it's done with the collecting the known target boxes it goes for the unknown target boxes by calling *collect unknown boxes* function. By calling this function it calls the *realign* function to go to the corners and make itself realigned with the grid coordinates. Then it orients itself at an angle with the axis of the grid and starts moving till it reaches the wall a then turns back and orients itself again at an angle and starts moving. This way it sweeps the complete arena in zigzag manner. And once say if it has swept along the y axis and reaches a corner then it orients itself along the x axis and starts sweeping in a similar way as before. And in case if it finds a target the it calls all the functions like *target nearest goal* function, *move to target* function, *drop box in goal* function and finally *realign* function to realign back with the grid.

Team Organization Evaluation and Plan

As we started off with the Project I, same kind of democratic strategy was implemented in the Project III also. Noticing the success of the team organization strategy for the past two projects, we split the whole project in the similar way of software, hardware and documentation instead of trying to spit in 4. We had one for hardware, one for documentation and two for software with one as senior and the other as the junior helping the senior. With the complex nature of the software we decided to have two members for it.

Now when we look back and at the success of the group at this project, some of the best things were a good software design, good hardware design that were met in a timely fashion. Hardware was finished way before the dead line giving the software people ample time for running and testing it as it progressed. And this thing worked out real well with the skipping of the actual testing phase of the project.

Result of the project

Good software, a good hardware and a good team organization lead to big success in the project. On the first day itself the team scored a highest score of ninety with dropping two cubes in the goal and knocking down the mobile robot two get an extra fifty points. On the second day the team scored a total of hundred points by dropping five cubes in the goals closest to them in an impressive way.

Robot code

```
// The_claw.ic
//
// Code for navigating a robot in an arena of orange boxes
// grabbing them, and depositing them one at a time in one
// of several black boxes

#use "cmucamlib.ic"

// DEFINES ////////////////////////////////////////////////
#define left_motor 3
#define right_motor 1

#define left_encoder 1
#define right_encoder 0

#define left_reflect analog(6)
#define right_reflect analog(5)
#define black_tape 160  // threshold for seeing the black tape

#define front_range_finder analog(17)
#define right_range_finder analog(16)
#define foot_range 100  // rangefinder value for a foot

#define closed_claw 2400  // claw servo values
#define open_claw 3700  // claw servo values
#define claw_servo servo0

#define foot_count 70  // necoder count for a foot drive
#define turn_count 32  // encoder count for a 90 degree turn

#define have_box 1  // constants for having a box or not
#define no_box 0

#define small_turn 11.125  // degrees used for a small turn
#define NORTH 1  // Enumeration of cardinal directions
#define EAST 2
#define SOUTH 3
#define WEST 4

#define start_direction 3 // Initial direction

float inputData[42] = {7.0, 3.0, 2.0, 3.5, 6.0, 4.5, 6.0, 6.0, 1.0, 7.0, 1.0, 9.5, 6.5, 10.0, 7.5,
10.0,
     -1.0, -1.0,-1.0, -1.0,-1.0, -1.0,-1.0, -1.0,-1.0, -1.0,-1.0, -1.0,-1.0, -1.0,-1.0, -1.0,
     1.0, 1.0, 6.0, 3.0, 2.0, 11.0,-1.0, -1.0,
```

```
    4.0, 5.5};

float target_x, target_y;     // target coords
float current_x, current_y;   // current coords
int facing;      // current facing
int targetBox;   // target box in inputData
int hasBox;      // are we dragging a box

// MAIN FUNCTION ////////////////////////////////////////////////

void main()
{
   startUp();
   collectKnownBoxes();
   collectUnknownBoxes();
}

// HIGH LEVEL BEHAVIORS //////////////////////////////////////////

// void startUp()
//    all the holistic robot initialization
void startUp()
{
   init_camera();
   init_expbd_servos(claw_servo);
   claw_servo = open_claw;
   enable_encoder(left_encoder);
   enable_encoder(right_encoder);

   current_x = (float)(int)inputData[40];
   current_y = (float)(int)inputData[41];
   facing = start_direction;
   hasBox = no_box;

   while(!start_button());

   start_process(statusDisplay());
}

// void collectKnownBoxes()
//    behavior to efficiently collect known boxes
void collectKnownBoxes()
{
   while(targetNearestBox() != -1.0)  // while there's more boxes listed
     {
      approachTarget();
```

```
        grabNearbyBox();
        if(hasBox == have_box) // if we've found a box
          {
           targetNearestGoal();
           approachTarget();
           dropBoxInGoal();
         }
        else  // if it didn't find a box
          reallign();
    }
}

// void collectUnknownBoxes()
//    behavior to sweep the arena looking for black boxes
//    Alligns in the corner, starts a search pattern,
//    occasionally alligning in the corner
void collectUnknownBoxes()
{
    float sweep_x, sweep_y;

    reallign();
    sweep_x = current_x;
    sweep_y = current_y;
    while(1)
      {
        sweep_x--;
        if(sweep_x <=0.0)
          sweep_x= 7.0;
        if(sweep_y == 1.0)
          sweep_y = 11.0;
        else
          sweep_y = 1.0;
        while((current_x != sweep_x)&&(current_y != sweep_y))
          {
           target_x = sweep_x;
           target_y = sweep_y;
           moveToTarget();
           grabNearbyBox();
           if(hasBox == have_box)
             {
              targetNearestGoal();
              approachTarget();
              dropBoxInGoal();
            }
         }
        reallign();
```

```
        target_x = sweep_x;
        target_y = sweep_y;
        moveToTarget();
    }
}

// MID LEVEL BEHAVIORS ////////////////////////////////////////

// float targetNearsetBox()
//    sets target_x, target_y to coordinates of the nearest box
float targetNearestBox()
{
    float thisDist;
    float bestDist = 21.0;
    int box;

    targetBox=1;

    for(box = 0; box <16; box++)
      {
        if(inputData[box*2]!= -1.0)
          {
           thisDist = abs(current_x - inputData[box*2]);
           thisDist += abs(current_y - inputData[box*2+1]);
           if (thisDist < bestDist)
             {
               targetBox = box;
               bestDist= thisDist;
             }
          }
      }
    target_x = inputData[targetBox*2];
    target_y = inputData[targetBox*2 + 1];

    if(bestDist == 21.0)
      return -1.0;
    return bestDist;
}

// float targetNearestGoal()
//    sets target_x, target_y to the coordinates of the nearest goal
float targetNearestGoal()
{
    float thisDist;
    float bestDist = 21.0;
    int box;
```

```
    int targetGoal=0;

    for(box = 16; box <20; box++)
      {
        if(inputData[box*2]!= -1.0)
          {
            thisDist = abs(current_x - inputData[box*2]);
            thisDist += abs(current_y - inputData[box*2+1]);
            if (thisDist < bestDist)
              {
                targetGoal = box;
                bestDist= thisDist;
              }
          }
      }
    target_x = inputData[targetGoal*2];
    target_y = inputData[targetGoal*2 +1];

    if(bestDist == 21.0)
      return -1.0;
    return bestDist;
}

// void grabNearbyBox()
//    behavior to reach out and grab a box, if one is there
void grabNearbyBox()
{
    float offset = 0.5;  // how far it goes off course

    smallMoveForward(offset);
    track_orange();
    if(track_confidence > 4){  // there is a box
        turnAndGrab();
        track_orange();

        while((track_confidence > 4)&&(track_y<0))
          {
            claw_servo = open_claw;
            turnAndGrab();
            track_orange();
          }
        hasBox = have_box;
    }
    else   // there is no box seen
      beep();
```

```
    smallMoveBackward(offset);

    //either way, assume there's no longer a box here
    if(targetBox != -1){
       inputData[targetBox*2]=-1.0;
       inputData[targetBox*2 + 1] = -1.0;
    }
}

// void turnAndGrab()
//    specific behavior to reach out a grab a seen box
void turnAndGrab()
{
    float degreesOff = 0.0;

    while(absI(track_x)>=10)
      {
       if(track_x>10)
         {
          smallTurnLeft(small_turn);
          degreesOff+=small_turn;
       }
       if(track_x< -10)
         {
          smallTurnRight(small_turn);
          degreesOff-=small_turn;
       }
       track_orange();
    }

    smallMoveForward(1.0);
    claw_servo = closed_claw;
    sleep(0.5);
    smallMoveBackward(1.0);

    //reorient
    if(degreesOff<0.0)
      smallTurnLeft(degreesOff);
    if(degreesOff>0.0)
      smallTurnRight(degreesOff);
}

// void dropBoxInGoal()
//    alligns to the tape, drops the box and celebrates
void dropBoxInGoal()
{
```

```
    // only one of next two lines!!!!
    allignToBlack();
    //smallMoveForward(.5);
    smallMoveForward(.5);
    claw_servo = open_claw;
    hasBox = no_box;
    sleep(.5);
    smallMoveBackward(1.0);
    //moveBackward(1);
    celebrate();
}

// void allignToBlack()
//    oscillate until both sensors see black simultaneously
//    and therefore allign with a black stripe
void allignToBlack()
{
    while((left_reflect < black_tape) || (right_reflect < black_tape))
      {
        if(left_reflect < black_tape)
          motor(left_motor, 50);
        else
          motor(left_motor, -50);
        if(right_reflect < black_tape)
          motor(right_motor, 50);
        else
          motor(right_motor, -50);
    }
}

// void moveToTarget()
//  issues commands to move to a target
void moveToTarget()
// WARNING - only considers one path
{
    if(current_x < target_x)
      {
        face(NORTH);
        moveForward((int)(target_x - current_x));
    }
    else if(current_x > target_x)
        {
          face(SOUTH);
          moveForward((int)(current_x - target_x));
      }
      if(current_y < target_y)
```

```
     {
      face(EAST);
      moveForward((int)(target_y - current_y));
    }
    else if(current_y > target_y)
       {
        face(WEST);
        moveForward((int)(current_y - target_y));
     }
}

// void approachTarget()
//    issues commands to move within one of a target
void approachTarget()
{
    if(current_x < target_x)
     {
      face(NORTH);
      if(current_y == target_y)
        moveForward((int)(target_x - current_x -1.0));
      else
        moveForward((int)(target_x - current_x));
    }
    else if(current_x > target_x)
       {
        face(SOUTH);
        if(current_y == target_y)
          moveForward((int)(current_x - target_x -1.0));
        else
          moveForward((int)(current_x - target_x));
     }
     if(current_y < target_y)
     {
      face(EAST);
      moveForward((int)(target_y - current_y - 1.0));
    }
    else if(current_y > target_y)
       {
        face(WEST);
        moveForward((int)(current_y - target_y - 1.0));
     }
}

// void reallign()
//    go to the nearest corner and allign by the walls
void reallign()
```

```
{
   // faces toward the nearest  N/S wall
   if(current_x<=4.0)
     {
      current_x = 1.0;
      face(SOUTH);
   }
   else
     {
      current_x = 7.0;
      face(NORTH);
   }
   // go to wall
   motor(left_motor, 100);
   motor(right_motor, 100);
   while(front_range_finder < foot_range);
   hitBreaks();

   rangeSweep();

   // run back from wall
   motor(left_motor, -100);
   motor(right_motor, -100);
   while(front_range_finder > foot_range);
   hitBreaks();

   if(current_y<=6.0)
     {
      current_y = 1.0;
      face(WEST);
   }
   else
     {
      current_y = 11.0;
      face(EAST);
   }
   motor(left_motor, 100);
   motor(right_motor, 100);
   while(front_range_finder < foot_range);
   hitBreaks();

   rangeSweep();
   rangeSweep();
}

// void rangeSweep()
```

```
//    turns from side to side, trying to find the nearest point
//    on the wall, and alligning there
void rangeSweep()
{
    int currentRange;
    int tempRange;
    currentRange = front_range_finder;

    while(1){
        smallTurnLeft(5.625);
        tempRange = averageRangeFront();
        if(tempRange > currentRange)
          currentRange = tempRange;
        else
          break;
    }
    while(1){
        smallTurnRight(5.625);
        tempRange = averageRangeFront();
        if(tempRange > currentRange)
          currentRange = tempRange;
        else
          break;
    }
    smallTurnLeft(5.625);
}

// LOW LEVEL BEHAVIOR ////////////////////////////////////////////////

// void face(int direction)
//    turn to the indicated direction
void face(int direction)
{
    if(facing == direction)
      return;
    if((facing == (direction - 1)) || ((facing == WEST) && (direction == NORTH)))
      {
        turnRight();
    }
    else if((facing == (direction + 1)) || ((facing == NORTH) && (direction == WEST)))
        {
          turnLeft();
      }
      else
        {
          turnRight();
```

```
      turnRight();
    }
}

// int averageRangeFront()
//    take an average of the front rangefinder sensor
int averageRangeFront()
{
  int loop;
  int sum=0;
  for(loop = 0; loop < 10; loop++)
    sum+=front_range_finder;
  return (sum/10);
}

// void moveForward(int dist)
//    move forward dist feet
void moveForward(int dist)
{
  int loop;
  for(loop = 0; loop<dist; loop++)
    {
     smallMoveForward((float)1.0);
     if(facing == NORTH)
       current_x += 1.0;
     if(facing == SOUTH)
       current_x -= 1.0;
     if(facing == WEST)
       current_y -= 1.0;
     if(facing == EAST)
       current_y += 1.0;
  }

  if(hasBox = no_box)
    {
     track_orange();
     if(track_confidence>4)
       {
        target_x = current_x;
        target_y = current_y;
        if(facing == NORTH)
          target_x += 1.0;
        if(facing == SOUTH)
          target_x -= 1.0;
        if(facing == WEST)
          target_y -= 1.0;
```

```
            if(facing == EAST)
              target_y += 1.0;

            targetBox = -1;
            return;
        }
    }
}

// void moveBackward(int dist)
//    move backward dist feet
void moveBackward(int dist)
{
    smallMoveBackward((float)dist);
    if(facing == NORTH)
      current_x -= (float)dist;
    if(facing == SOUTH)
      current_x += (float)dist;
    if(facing == WEST)
      current_y += (float)dist;
    if(facing == EAST)
      current_y -= (float)dist;
}

// void turnRight()
//    turn right 90 degrees
void turnRight()
{
    smallTurnRight(90.0);

    facing++;
    if(facing > WEST)
      facing = NORTH;
}

// void turnLeft()
//    turn left 90 degrees
void turnLeft()
{
    smallTurnLeft(90.0);

    facing--;
    if(facing < NORTH)
      facing = WEST;
}
```

```
// void moveForward(float dist)
//    move forward dist feet, for small distances
void smallMoveForward(float dist)
// WARNING - offset not kept
{
   int targetDist;
   reset_encoder(left_encoder);
   reset_encoder(right_encoder);
   motor(left_motor, 100);
   motor(right_motor, 100);

   targetDist = (int) ((float)foot_count * dist);
   while(read_encoder(left_encoder) < targetDist);
   hitBreaks();
   ao();
}

// void moveBackward(float dist)
//    move backward dist feet, for small distances
void smallMoveBackward(float dist)
// WARNING - offset not kept
{
   int targetDist;
   reset_encoder(left_encoder);
   reset_encoder(right_encoder);
   motor(left_motor, -100);
   motor(right_motor, -100);

   targetDist = (int) ((float)foot_count * dist);
   while(read_encoder(left_encoder) < targetDist);
   hitBreaks();
   ao();
}

// void turnRight(float deg)
//    turns right deg degrees, for small distances
void smallTurnRight(float deg)
// WARNING - offset not kept
{
   int targetDeg;
   reset_encoder(left_encoder);
   reset_encoder(right_encoder);
   motor(left_motor, 100);
   motor(right_motor, -100);

   targetDeg = (int) ((float)turn_count * deg /90.0);
```

```
    while(read_encoder(right_encoder) < targetDeg);
    hitBreaks();
    ao();
}

// void turnRight(float deg)
//    turns right deg degrees, for small distances
void smallTurnLeft(float deg)
// WARNING - offset not kept
{
    int targetDeg;
    reset_encoder(left_encoder);
    reset_encoder(right_encoder);
    motor(left_motor, -100);
    motor(right_motor, 100);

    targetDeg = (int) ((float)turn_count * deg /90.0);
    while(read_encoder(left_encoder) < targetDeg);
    hitBreaks();
    ao();

}

// void hitBreaks()
//    Attempts to stop on a dime
void hitBreaks()
{
    int loop;
    for(loop = 0; loop <2; loop ++)
      {
       motor(left_motor, -100);
       motor(right_motor, -100);
       motor(left_motor, 100);
       motor(right_motor, 100);
    }
    motor(left_motor, -60);
    motor(right_motor, -60);
    motor(left_motor, 30);
    motor(right_motor, 30);
    ao();
    sleep(0.1);
}

// void celebrate()
//    play the victory theme from Final Fantasy
void celebrate()
```

```
{
   tone(550.0,.1);
   tone(550.0,.1);
   tone(550.0,.1);
   tone(550.0,.3);
   tone(440.0,.3);
   tone(500.0,.3);
   tone(550.0,.2);
   tone(500.0,.1);
   tone(550.0,.3);
}

// int isBoxAt(float x, float y)
//   goes through the list to see if a known box exists at
//   the given coordinates
int isBoxAt(float x, float y)
{
   int loop;
   for(loop = 0; loop < 16; loop++)
     {
      if ((inputData[loop*2] ==x) &&(inputData[loop*2 + 1] == y))
        return 1;
   }
   return 0;
}

// HELPER FUNCTIONS ///////////////////////////////////////////////////
float abs(float n)
{
   if(n<0.0)
     return -n;
   return n;
}

int absI(int n)
{
   if(n<0)
     return -n;
   return n;
}

void statusDisplay()
{
   while(1)
     printf("%d, %d\n", current_x, current_y);
}
```