Boustrophedon Bandit™

Doug Densmore
Will Plishker
May 9, 2002
EE290o Final Demo

Outline

- Goals
- Coverage Path Planning Algorithms
- Design Issues
- System Structure
  - Grid System
  - Overall System Picture
  - Code Segments
- Conclusion

Physical Goals

- Sweep a predefined, obstacle free, environment for a predetermined target.
- Once the target is acquired, return to the point at which the search started with the target in tow.
- Perform the task with no human interaction.

Software Goals

- Potential to discuss time safety
- Task and driver separation
- Inner task communication
- EMachine structure

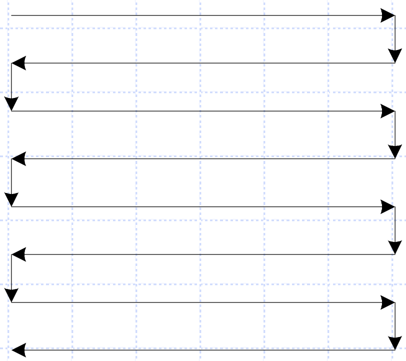
Coverage Path Planning Algorithms

- Emphasize the space swept out by the robots sensor.
- Requires integrating the robot’s footprint (detector range) along the coverage path.
- Similar to the traveling salesman problem but instead of just visiting neighborhoods, one must visit all points in the target environment

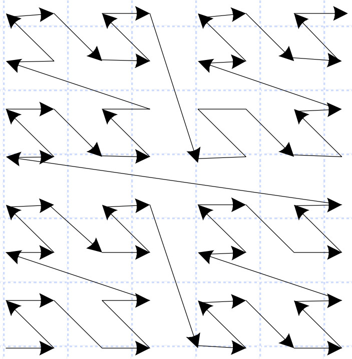
Coverage Path Planning Algorithms

- Four types
  - Heuristic (and random), approximate, partial-approximate, and exact cellular decomposition.
- Many variations within each to include obstacles and multiple searching parties.
- We focused on exact cellular decompositions. These are sets of non-intersecting regions and therefore planning is reduced to planning motions from one region to another.

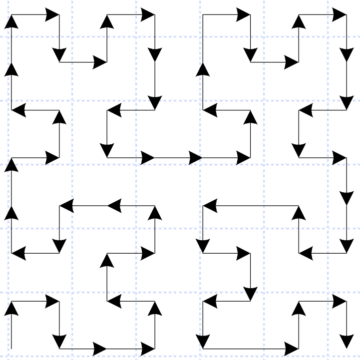
Exact Cellular Decompositions



Boustrophedon



Morton Order



Pi-Order

These patterns often used in raster scan

Boustrophedon means “way of the ox” in Greek.

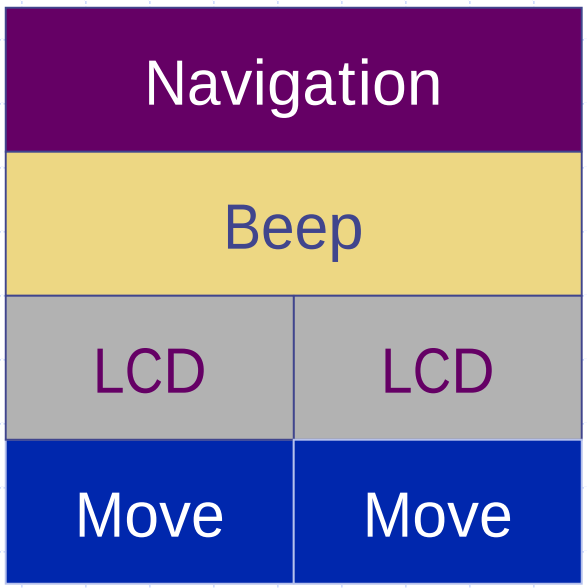
Physical Design Issues

- Environment
  - Lighting, search surface, search size, traction
- Vehicle
  - Weight, sensor input limitations, funneling mechanism limitations

Software Design Issues

- Trigger logic vs. Task Logic
  - Put effort into tasks and kept basic trigger system. Same behavior, potentially time safe.
- Granularity of Task
- Mode Switching

System Relationships

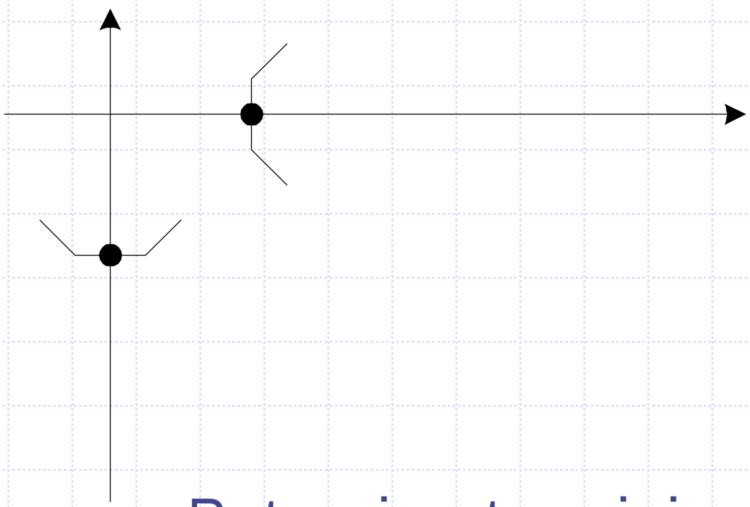


- Navigation
  - Tracks location, turning decisions
- Beep
  - Plays notes from “Mary Had a Little Lamb”
- LCD
  - Displays information such as position, rotation and light measurements.
- Move
  - Powers motor for one X or Y movement

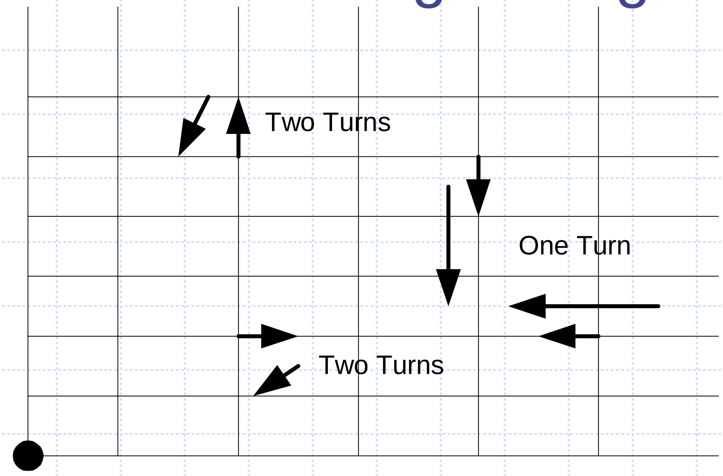
Grid System

- Based on simple x and y coordinates
- Theoretically the movement required to make a change in x is equal to the distance to change y.
- Starting point is (0,0)
- Location in reference to the starting point.

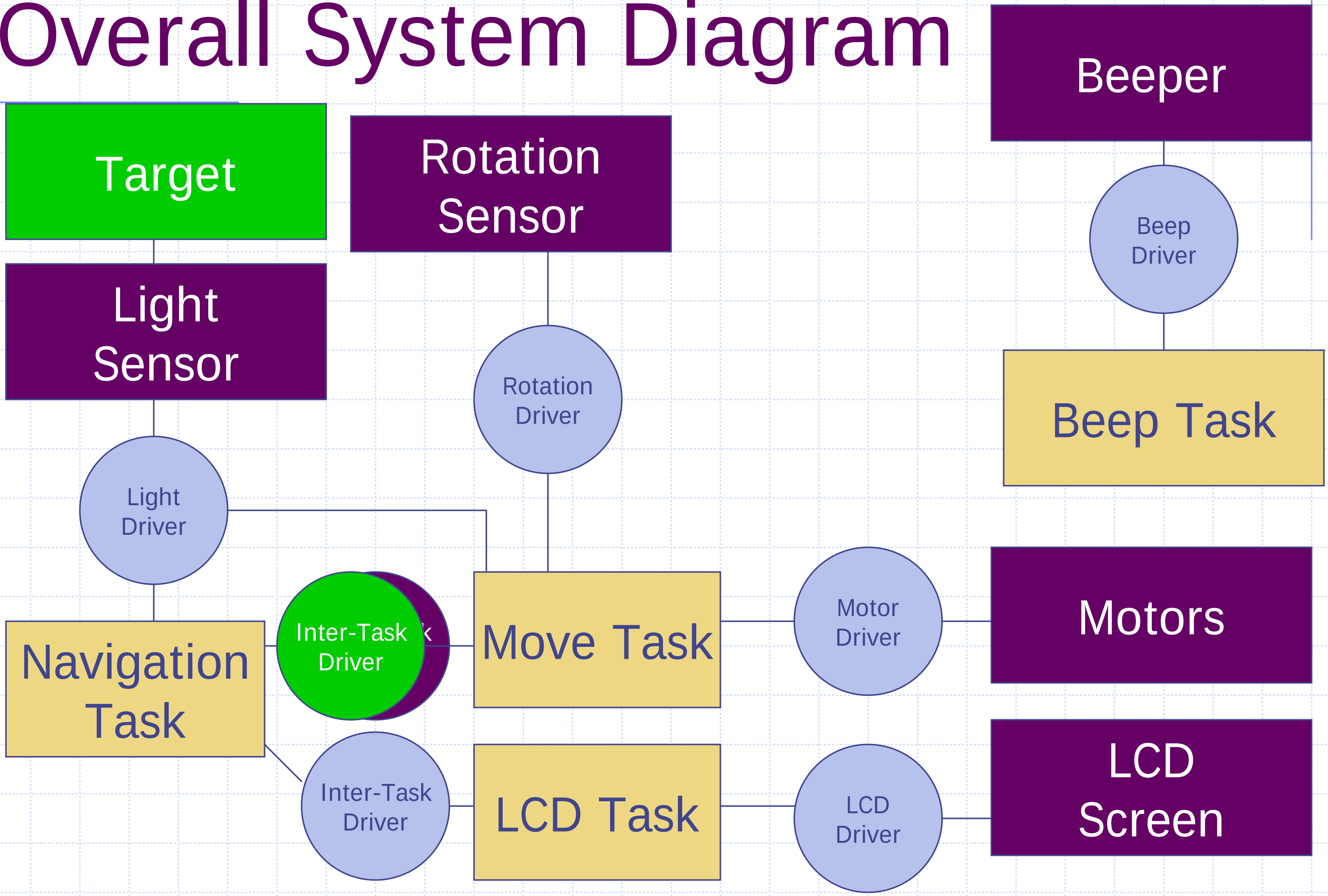
Change in (X,Y)



Returning to origin



Overall System Diagram



Code Segments

```
ecode[0].opcode = call;        ecode[0].driver  = beep_driver;
ecode[1].opcode = call;        ecode[1].driver  = lcd_driver;
ecode[2].opcode = call;        ecode[2].driver  = rotation_driver;
ecode[3].opcode = call;        ecode[3].driver  = motor_driver;
ecode[4].opcode = call;        ecode[4].driver  = intertask_driver;
ecode[5].opcode = call;        ecode[5].driver  = light_driver;
ecode[6].opcode = schedule;    ecode[6].task.fp = beep_task;       ecode[6].task.priority=2;
ecode[7].opcode = schedule;    ecode[7].task.fp = lcd_task;        ecode[7].task.priority=1;
ecode[8].opcode = schedule;    ecode[8].task.fp = navigation_task; ecode[8].task.priority=1;
ecode[9].opcode = schedule;    ecode[9].task.fp = move_task;       ecode[9].task.priority=1;
ecode[10].opcode = future;     ecode[10].index  = 20;

ecode[20].opcode = call;       ecode[20].driver  = rotation_driver;
ecode[21].opcode = call;       ecode[21].driver  = lcd_driver;
ecode[22].opcode = call;       ecode[22].driver  = motor_driver;
ecode[23].opcode = call;       ecode[23].driver  = intertask_driver;
ecode[24].opcode = call;       ecode[24].driver  = light_driver;
ecode[25].opcode = schedule;   ecode[25].task.fp = lcd_task;       ecode[25].task.priority=1;
ecode[26].opcode = schedule;   ecode[26].task.fp = move_task;      ecode[26].task.priority=1;
ecode[27].opcode = future;     ecode[27].index   = 0;
```

Driver Code

```
void intertask_driver(){
    moveInput.action_state = navigateOutput;
    navigateInput.done_moving = moveOutput;
}

void beep_driver(){
    dsound_play(beep_in);
    wait_event(dsound_finished,0);

}//end beep

void lcd_driver(){
    cputw(lcd_in);
}

void light_driver(){
    if((light->value()<BLOCK_FOUND))
        navigateInput.block_found = 1;
    else if(!navigateInput.block_found)
        navigateInput.block_found = 0;

    moveInput.block_found = navigateInput.block_found;

    if((moveInput.block_found)&&(light->value()>150))
        moveInput.block_up = 1;
}
```

```
void rotation_driver(){
    rot_sensor_axel = ROTATION_1;
    rot_sensor_dir  = ROTATION_2;

    if(reset_rot_sensor_axel)
        ds_rotation_set(&SENSOR_1, 0);

    reset_rot_sensor_axel = 0;
}
void motor_driver(){    //move forward
    motor_a_dir(axel_motor_dir);
    motor_a_speed(axel_motor_speed);

    //turn motor
    switch(turn_motor_dir){
    case left_turn:
        motor_b_dir(fwd);
        motor_b_speed(turn_motor_speed);
        break;

    case right_turn:
        motor_b_dir(rev);
        motor_b_speed(turn_motor_speed);
        break;

    case no_turn:
        motor_b_dir(off);
        motor_b_speed(turn_motor_speed);
        break;
    }

    motor_c_dir(arm_motor_dir);
    motor_c_speed(arm_motor_speed);
}
```

Code Tasks

```
void lcd_task(){
    lcd_in = (x<<8)|(y);
    //    lcd_in = rot_sensor_axel;
    //lcd_in = light->value();
}

void beep_task(){
    static note_t ourmusic[11];
    static int i=0;

//array here

    beep_in[0].pitch=ourmusic[i].pitch;
    i++;
    if(i>10) i=0;
}
```

```
void move_task() {
    static Turn_State turn_state = start;
    static int done_acked = 1;
    static int do_once=0;

      moveOutput = 0; //accept next command by saying
      not done
       switch(moveInput.action_state) {
       case right:
           switch(turn_state) {
           case start:
         case turning:
         case moving:
      case left:
         switch(turn_state) {
           case start:
          case turning:
           case moving:
      case forward:
       switch(turn_state) {
           case start:
          case turning:
           case moving:
case stopped:
default:
else {}
```

Navigation Task

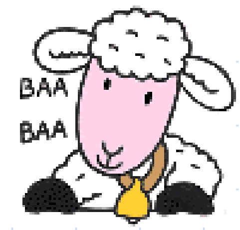
- void navigation_task () {

```
static Action_State action_state = stopped;

    if((action_state==stopped)) {
    switch(dir) {
        case forward:
        case right:
        case left:
        case backward:
    }
    } else if(!navigateInput.done_moving) {
        navigateOutput = action_state;
    } else { //else ack the end of move
        action_state = stopped;
        navigateOutput = action_state;
    }

    return;
}
```

Conclusion

- Difficult to balance task complexity with ecode complexity
- Hardware is imprecise ☹
- Ecode infrastructure limits the design of tasks
- Oxen are good

References



- H. Choset, Coverage for robotics – A survey of recent results, *Annals of Mathematics and Artificial Intelligence* (2001) pp113-126.
- L.Villa, LegOs HOWTO, http://legos.sourceforge.net/
- The NCGIA Core Curriculum in GIScience, *http://www.ncgia.ucsb.edu/giscc/*