PROJECT I – Sensing and Movement
CS 5973 – Introduction to Intelligent Robots
Final Report

Group 4

- **Justin Fuller**
- **Matthew Lawrence**
- **Rahul Kotamaraju**

Contents

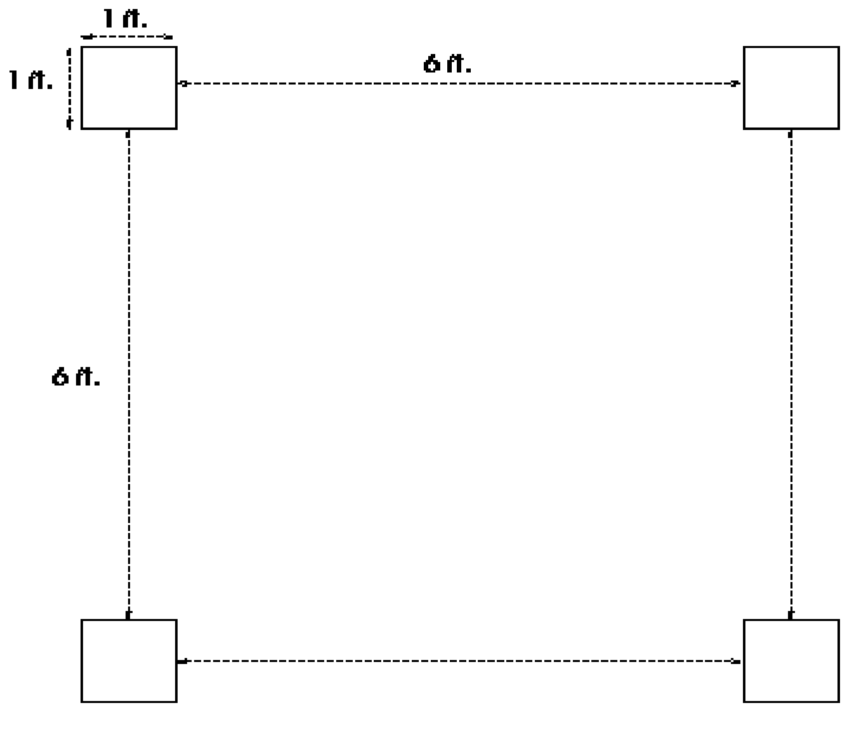
17th February 2003

Project 1 Final Report - Sensing and Movement (CS 5973)

Group 4:
Justin Fuller
Matthew Lawrence
Rahul Kotamaraju

1. Aim

Pieces of black electrical tape were attached to the floor in the front of the classroom to form one-foot squares. These squares were placed six feet apart from one another, to form the corners of a larger square as shown in the figure below. The robot must start completely within one of the tape squares and must move to within one inch of a second tape square, then a third, then the fourth, then back to the first, completing one circuit. The robot must then complete this circuit two more times. The robot must stop after completing the three circuits. The stipulated time for achieving this was five minutes or two-and-half minutes per attempt if the team desired another chance.



2. Robot Design

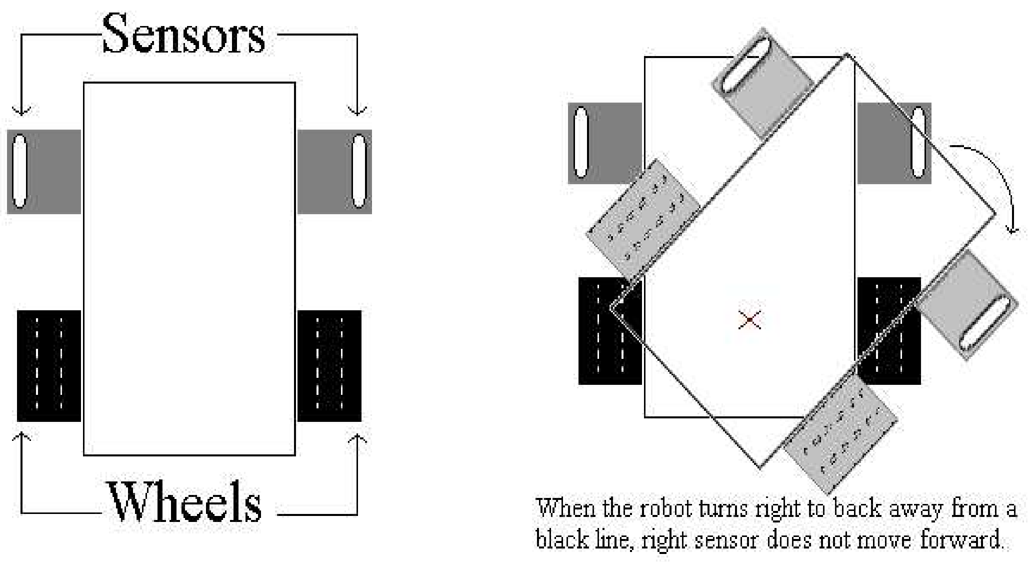
2.1. Initial Overview

The initial inspiration for our robot was the HandyBug 9719 model given in the Martin text. After construction, several small modifications were made, including the removal of the touch sensor apparatus. The front of the HandyBug rests on two smooth-bottomed legs. The friction was unacceptable, and the legs were promptly replaced by a caster inspired by one shown in The Art of Lego Building, also written by Martin. The only sensors, IR reflectance, hung just outside the wheel base along the driving wheels' axis.

2.2. Sensors

When the sensors were just outside the wheel base, readings were somewhat erratic. The group added electrical tape to "blind" the sensors and reduce possible interference. Also, the sensors took positions closer to the floor, as this aided in differentiating between white (the floor tile) and black (the electrical tape boundaries in this project's arena). The sensors hung from an interesting and somewhat unstable sensor mount which made the alignment process too difficult. The group then decided to move the sensors forward to avoid breakage and to ensure consistent readings. This worked to the group's satisfaction. It is noteworthy that all sensor placements took place after considering how the vehicle's rotation would affect the alignment process. As long as the sensors were outside the wheel base, rotation would never cause the sensor to unintentionally move too far forward and nullify alignment.

Figure 1: Sensor/Motor Arrangement


Sensors
Wheels
When the robot turns right to back away from a black line, right sensor does not move forward.

2.3. Handy Board Cradle

As various sensor mounts came and went, the Handy Board had to be moved to a higher location on the chassis. Its large size interfered especially with the original sensor arms. The structure of the cradle didn't change much, in spite of the large change in position. At later stages, this modification was never undone.

2.4. Gears and Wheels

Another item requiring change was the robot's gear ratio. The Handy Bug has a gear ratio of 1:5 with large wheels. This provides great speed, but low power. When low speeds were required of the robot, it was unable to propel itself. After some experimentation, slightly larger (24-tooth) gears replaced those on the driving motors (8-tooth), and smaller wheels took the place of the original ones with large diameter.

2.5. Forward Contacts

Many experimental setups vied for dominance regarding how the front of the robot should interact with the floor. The caster was favorite for most of the time, until it was determined that it introduced an unpredictable torque into the system. Various smooth legs had too much friction, and all caster setups proved to act as guide wheels. A compromise was reached when the group made a combination of the two: a leg that did not spin, bounded below by two large pulley wheels. This provides very low friction when moving forward, and a great deal of friction when turning. This added friction when turning is helpful because the robot can use large torques with less chance of overshooting the desired turn distance. Also, the wheels help guide the robot when moving forward, solving the problem with the initial caster.

2.6. Concluding Overview

The final product is a two-motor, differential drive robot. Each motor transfers power across a gear ratio of 3:5 to the driving wheels. Two other wheels exist in the form of free-turning guide wheels mounted near the front of the vehicle, similar to a caster but without the ability to change direction. Near the front of the robot, two IR reflectance sensors extend outside the wheel base. These indicate when the robot is over normal floor or a black line. This positioning for the sensors ensures that when sensing a line, corrective turning will always force the sensor to back up behind the line. Placed high on the robot's chassis, the Handy Board acts as the robot's brain, coordinating its movement with the information from its sensors.

3. Robot Code

```
/*  Author: M. Lawrence
    Course: CS5973, SP03
    Project: 1
    Last Modified: 2-12-03, M. Lawrence
    Simple code designed to satisfy the requirements
    of Project 1.  This program instructs the bot to
    move in a fixed pattern as follows:  align the
    bot’s front end with the tape square, cruise
    staright ahead until tape is found or bot times
    out, turn clockwise roughly 90 degrees, back up
    for a fixed period.  The bot will repeat the
    process for a given number of legs and laps.

*/

#define NUM_LAPS 3  /*number of laps to run*/
#define NUM_LEGS 4  /*number of legs to run*/
#define L_MOTOR 0  /*left motor*/
#define R_MOTOR 1  /*right motor*/
#define DRIVE_RATIO 80   /*ratio of motor power*/
#define INIT_DRIVE_RATIO 88   /*ratio of motor power at low speed*/
#define RUN_LENGTH 5500L  /*max runtime for fullAhead phase*/
#define TURN_LENGTH 280L  /*length of time for turning*/
#define BACK_LENGTH 500L  /*length of time for revers*/
#define CRUISE_SPEED 80  /*motor power for normal cruising*/
#define CRUISE_SPEED_LO 50  /*motor power for low-speed cruising*/
#define CRUISE_TIME_LO 1500L  /*amount of time for low-speed cruise*/
#define ALIGN_SPEED 7  /*motor power during align phase*/
#define ALIGN_SPEED_HI 28  /*alternate motor power during align*/
#define ALIGN_R_MOTOR 1
#define CROSS_LINE_LAG_TIME 100L  /*lag time after detecting tape*/
#define R_THRESHOLD 110  /*default threshold*/
#define L_THRESHOLD 110  /*default threshold*/
#define THRESH_DIFF 0  /*default threshold differential*/
#define MAX_TIME 700  /*max time to attempt alignment*/

int alignStatus[2];
int alignSpeed[2];
int threshold[2] = {L_THRESHOLD, R_THRESHOLD};
int overshoot;

int readSensor(int s) {
    //read light sensor and apply threshold
    int j = analog(s + 2);
    if(s)
      printf("%d\n", j);
    if(j > threshold[s])
      return 1;
    return 0;
}

void calibrate() {
    int i, l = 0, d = 0;
    //calbrate light condition
    while(!start_button()) {
        printf("Cal light, %d\n", analog(2 + L_MOTOR));
        msleep(100L);
    }
    //take 10 readings
    for(i = 0; i < 10; i++) {
        l += analog(2 + L_MOTOR);
        msleep(100L);
    }
    //calibrate dark condition
    while(!start_button()) {
        printf("Cal dark, %d\n", analog(2 + L_MOTOR));
        msleep(100L);
    }
    //take 10 readings
    for(i = 0; i < 10; i++) {
        d += analog(2 + L_MOTOR);
        msleep(100L);
    }
    //calculate and display thresholds
    threshold[0] = (5 * (d - l)) / 100 + l / 10;
    threshold[1] = threshold[0] + THRESH_DIFF;
    printf("%d, ", threshold[0]);
    printf("%d\n", threshold[1]);
    while(!start_button());
    msleep(500L);
}

void alignMotor(int m) {
    //compenstae for weaker right-side motor
    alignSpeed[m] = ALIGN_SPEED + ALIGN_R_MOTOR * m;
    while(1) {  //loop until thread is killed
        //drive motor while not over tape
        while(alignStatus[m] == 0) {
            motor(m, alignSpeed[m]);
            defer();  //thread sleeps for ~50ms
            alignStatus[m] = readSensor(m);
        }
        off(m);
        //burst motor backwards briefly
        motor(m, -ALIGN_SPEED_HI);
        msleep(500L);
        off(m);
        //set other motor to higher speed
        alignSpeed[1 - m] = ALIGN_SPEED_HI;
        //idle motor while sensor is aligned
        while(alignStatus[m] == 1) {
            defer();  //thread sleeps for ~50ms
            alignStatus[m] = readSensor(m);
        }
        //return other motor to normal speed
        alignSpeed[1 - m] = ALIGN_SPEED;
    }
}

int alignAxle() {
    int proc[2], i = 0;
    beep();
    printf("Align\n");
    alignStatus[L_MOTOR] = 0;
    alignStatus[R_MOTOR] = 0;
    //start left motor thread
    proc[0] = start_process(alignMotor(L_MOTOR));
    //start right motor thread
    proc[1] = start_process(alignMotor(R_MOTOR));
    //block while not aligned and not timed out
    while(i < MAX_TIME && (!alignStatus[L_MOTOR] ||
!alignStatus[R_MOTOR])) {
        i++;
        msleep(10L);
        defer();
    }
    kill_process(proc[0]);
    kill_process(proc[1]);
    alloff();
    beep();
    msleep(1000L);
    //return boolean indicating success of alignment
    if(i == MAX_TIME)
      return 0;
    return 1;
}

void reverse() {
    beep();
    printf("Reverse\n");
    //drive motors at given ratio
    motor(R_MOTOR, -100);
    motor(L_MOTOR, -(100 * DRIVE_RATIO) /100);
    msleep(BACK_LENGTH);
    alloff();
    msleep(500L);
}

void fullAhead() {
    beep();
    printf("Full ahead\n");
    overshoot = 1;  //indicate no tape found
    //drive motors at low speed
    motor(R_MOTOR, (CRUISE_SPEED_LO * INIT_DRIVE_RATIO) / 100);
    motor(L_MOTOR, CRUISE_SPEED_LO);
    //cruise while not detecting tape
    msleep(CRUISE_TIME_LO);
    //cruise at higher speed
    motor(R_MOTOR, 80);
    motor(L_MOTOR, 80);
    //run until tape is sensed
    while(!readSensor(L_MOTOR) && !readSensor(R_MOTOR));
    overshoot = 0;  //tape has been found
    msleep(CROSS_LINE_LAG_TIME);  //cruise into square
    alloff();
    beep();
    msleep(RUN_LENGTH);
}
```

```
void cruise() {
    //start fullAhead as process
    int p = start_process(fullAhead());
    msleep(RUN_LENGTH);
    kill_process(p);  //kill process after timeout
    alloff();
    beep();
    msleep(2000L);
}

void spin() {
    beep();
    printf("Spin\n");
    //drive motors in differential mode
    fd(L_MOTOR);
    bk(R_MOTOR);
    msleep(TURN_LENGTH);
    alloff();
    beep();
    msleep(1000L);
}

void burst() {
    motor(R_MOTOR, 100);
    motor(L_MOTOR, (100 * DRIVE_RATIO) / 100);
    msleep(40L);
    alloff();
}

void main() {
    int i, j;
    //calibrate();
    printf("DeathMonkey v2.3\n");
    while(!start_button());
    for(i = 0; i < NUM_LAPS; i++) {
        for(j = 0; j < NUM_LEGS; j++) {
            //repeatedly attempt to align bot
            while(!alignAxle())
              reverse();  //align on exit 0
            cruise();  //drive ahead ~7’
            if(overshoot)
              reverse();
            spin();  //spin ~90
            reverse();
        }
    }
    printf("Done\n");
}
```

4. Robot Code Documentation

Following is a table that describes the methods used in the software for Project 1.

Method	Accepts	Returns	Remarks
readSensor	<int> value indicating the left/right sensor.	<int> value specifying whether sensor reads black tape or otherwise	Reads the given light sensor and compares it to a threshold.
calibrate	<void>	<void>	Calibrates the thresholds for light and dark sensor readings.
alignMotor	<int> value indicating the left/right motor.	<void>	Positions the given motor over the tape square.
alignAxle	<void>	<void>	Aligns the robot with the tape square.
reverse	<void>	<void>	Drives the robot backwards for a fixed amount of time.
fullAhead	<void>	<void>	Drives the robot straight ahead until tape is detected.
cruise	<void>	<void>	Limits the time length of the fullAhead phase.
spin	<void>	<void>	Spins the robot for a given time.
Burst	<void>	<void>	Drives the robot ahead for a short burst of time.

Software consists of four phases:

- Align
- Cruise
- Turn
- Reverse

Alignment phase drives robot slowly until sensors are aligned over tape. If robot enters undesirable state, alignments will timeout. If alignment times out, robot will reverse, then reattempt alignment. After successful alignment, robot will be aligned squarely with tape and will face the target square.

Cruise phase drives the robot straight ahead until tape is detected or timeout occurs. Cruise begins at a slow speed to gain momentum, and then switches to a higher speed. During low speed, light sensors are ignored; this ensures that robot will fully exit the square before sensing begins. If robot detects tape, it will continue ahead briefly so that the robot will be fully inside the square. If the phase times out, then the robot has veered to one side and has overshot the square. In this case, robot will back up a fixed length; the target square should now be directly to either the left or the right.

The **Turn** phase spins the robot 90 degrees clockwise. After turn phase, if the robot is not in the target square, the square will be either behind or ahead of the robot.

The **Reverse** phase drives the robot backward for a fixed time. This ensures that the robot will be behind the square when alignment begins. After the reverse phase, the square will be in front of the robot regardless of whether the robot veered off-course during the cruise phase.

The software also includes a **Calibration** routine which was disabled during the demonstration. The routine instructs the user to place the robot over a clear floor space, and then takes an average light reading. The routine repeats over a "taped" spot on the floor. The routine now has a *light* reading and a *dark* reading and sets the threshold to the average of these two readings.

4.1. Ideal Robot Operation

- 1.1. Align
  - 1.1.1. Use left and right sensors to align with tape.
    - 1.1.1.1. Drive motors independently at low speed.
    - 1.1.1.2. Monitor sensors.
    - 1.1.1.3. Halt motor when corresponding sensor trips.
    - 1.1.1.4. Reverse corresponding motor briefly.
    - 1.1.1.5. Increase speed on opposite motor.
    - 1.1.1.6. Re-engage motor when sensor is no longer tripped.
  - 1.1.2. Repeat until both sensors are tripped simultaneously.
- 1.2. Cruise
  - 1.2.1. Engage both motors at low speed.
    - 1.2.1.1. Begin to build inertia without bucking.
    - 1.2.1.2. Drive straight ahead for fixed time.
    - 1.2.1.3. Ignore sensors.
    - 1.2.1.4. Ensure that robot travels outside of square.
  - 1.2.2. Increase motor speed.
    - 1.2.2.1. Drive straight ahead.
    - 1.2.2.2. Monitor sensors.
    - 1.2.2.3. Continue until tape is found.
  - 1.2.3. Continue cruise.
    - 1.2.3.1.Drive briefly after tape detection.
    - 1.2.3.2.Ensure robot fully enters square.
- 1.3. Turn
  - 1.3.1. Spin robot ~90 degrees clockwise.
  - 1.3.2. Drive motors differentially.
  - 1.3.3. Continue for brief, fixed time.

1.4. Reverse
    1.4.1. Drive both motors in reverse.
    1.4.2. Continue for fixed time.
    1.4.3. Ensure that tape square is in front of robot.

4.2. Fail-safe Operations

1.5. Align
    1.5.1. Introduce time limit to alignment phase.
    1.5.2. Fail alignment phase if timeout occurs.
        1.5.2.1. Drive robot backwards for brief time.
        1.5.2.2. Retry alignment phase.
    1.5.3. Repeat until alignment is successful.
1.6. Cruise
    1.6.1. Introduce time limit to cruise phase.
    1.6.2. Fail cruise phase if robot veers left or right and misses the tape.
        1.6.2.1. Reverse robot for fixed time.
        1.6.2.2. Ensure robot is in line with square.

NOTE: In all cases, the timings for fixed-time routines were determined experimentally.

5. Team Organization Evaluation and Plans

This is a consensus document of the evaluation of the team's performance in Project 1, Sensing and Movement.

5.1. Team Work

The team meetings and the brainstorming sessions worked out very well as each member had an equal and important contribution for the project. All the members agreed that the democratic approach worked very well for the team and for the completion of the project. The team, in a brief period of time, came to a good understanding. All the team meetings were productive with the melding of individual ideas to give the robot its ultimate form.

5.2. Software

Each one has one's own style of coding. The team agreed that one person will have the holistic view of the code, and Matthew does an excellent job coding by rendering it easily readable and modular. The team is very satisfied with the software design of the project. In future projects, the team shall have each member assigned to only one aspect of the project so that the project has one expert dealing with the task throughout the project.

5.3. Design

The team dedicated three days for design and realized that this was not sufficient as the robot's design underwent repeated alterations, even during the testing phase of the project. The team's fall back plan for the design phase was not specific, and the team agreed that in the future projects, a specific fall back plan will be outlined for each phase of the project to make the whole process more efficient. The team also decided that the experimentation so performed on the robot will be limited to not later than the design phase of the project.

5.4. Team Strategy

The team's approach of functioning as an autonomous one succeeded except for the part where no one took up the accountability for the team strategy, and checking for the adherence to timeline and fall back plan. The team wishes to change this aspect in its structure for the forthcoming group projects by one or more members of the team assuming this responsibility.