Project 1 – Sensing and Movement

Team 1
Stephen Mckinney Jeremy Branecky Camilo Reyes

Dr. Dean Hougen
CS 4023 / 5023
Introduction to Intelligent Robotics
Spring 2003

The University of Oklahoma
Norman, OK 73019

Robot Design:

The team decided on a rotation of members assigned to work on the design of the robot; as a result each team member was given the opportunity to work on the design alone in order to give a more constructive approach to the final design.

Body Design:

Our final design was a big tractor like robot with large wheels on the back and a small skid platter centered at the front. The big wheels on the back provided overall stability to the alignment in order to keep the robot driving in a straight line. The skid platter on front was round and smooth enough to introduce minimal friction with the ground. With this design, the rear wheels controlled the movement of the robot; by contrast, the skid platter was intended to support the weight of the front of the robot while minimizing interaction with the ground. This configuration allowed for highly accurate turns and perfectly straight driving under test conditions.

Robot Gearing:

The gear system of our robot went through several stages of design. Initially, the robot made very little use of gears; the motors drove the wheels directly. However, it quickly became apparent that this approach would not be appropriate because such a design would cause the robot to be very fast but very inaccurate. Eventually, after several modifications, we settled on a design that used a 25 to 1 gear reduction (on top of the gear reduction that is present inside the motors). This design provided the robot with an acceptable level of torque and enabled it to drive slowly but accurately.

The geartrain began with an 8-tooth gear connected directly to the motor. This gear was horizontally connected to a 40-tooth gear that was coaxial with another 8-tooth gear. The second 8-tooth gear was horizontally connected to another 40-tooth gear that was coaxial with the wheel. This system, which of course was symmetrical for the right and left sides of the robot, formed the drive train of the robot. However, we used additional gears to achieve another effect. Attached diagonally to the first 40-tooth gear was a 24-tooth gear that was, in turn, attached horizontally to an 8-tooth gear. This 8-tooth gear was connected to an axle that held a pulley part that was used in our break-beam shaft encoder system. This configuration had the effect of negating the gear reduction on the shaft-encoded axle: the shaft-encoded axle spun at the same rate as the motor. Our design gave the robot a good amount of torque while allowing the shaft encoder system to be very accurate (because the encoded shaft spun at a much higher rate than the drive axle).

With our gear system design, we strove for stability and precision. However, the fact that we used Lego gears and pieces to accomplish this presented us with yet another challenge to accomplish the goal. Because of this, we ended up with a robot that naturally veered to the right or left (at different stages in our design). This problem had to be corrected in the software.

Sensors:

Two kinds of sensors were used in the robot: a digital color camera (the CMUcam) and a pair of break-beam shaft encoders. The shaft encoder sensors were placed in the back of the robot in special housings surrounding the axle-pulley system described above. The team decided to tape the shaft encoders on the back to add consistency to the design. The main task of these encoders is to get a tic count coming from the back wheels; this allowed for the team to keep the robot going straight and to make accurate turns by comparing the number of tics (and therefore the number of rotations) for each wheel. We relied very heavily upon the use of these encoders for the overall maneuverability of the robot. The CMUcam was used as a part of the project to acquire color readings in the course. The team placed the camera right on front of the robot, above the front platter; this allowed any surrounding light to help the camera read the colors laid out in the course. During the early stages of design our team planned to add an extra set of sensors: a pair of IR sensors to keep the robot aligned with black tape laid out in the course. However, due to time constraints and the fact that we achieved straight driving over distances of more than ten feet and extremely accurate turns during our testing, we did not add these sensors.

Overall Layout:

To give a more descriptive layout, the robot is touching the ground in three main parts of the design: the two big tractor like wheels on the back, and the small skid platter on front. The two shaft encoders have been placed in the back right behind the wheels; they are also attached to the gear train that is going into the wheels. The handy board is placed right above the back wheels in a comfortable housing: the team decided it was best to place most of the weight of the robot on the back because we did not want a lot of friction on the front part of the robot. The CMUcam is placed in front of the robot to give better color readings; at some point the team decided placing it at the front would give us better color readings than it would at its original position at the center of the robot because the camera would get more surrounding lighting in the front of the robot. The gear train was kept outside on the robot; it is natural to place the gears right next to the wheel because these two designs go together.

Robot Code:

```
/*************************************************************
 *  Project 1                                                *
 *  Team 1: Stephen Mckinney, Jeremy Branecky, Camilo Reyes  *
 *                                                           *
 *  Program to make robot follow a course with colored       *
 *  tiles on the floor that tell it where to go.             *
 *                                                           *
 *************************************************************/

#use "cmucamlib.ic"
#define L_MOTOR 0
#define R_MOTOR 3
```

```
#define L_ENCODER 0
#define R_ENCODER 1

#define STRAIGHT_SPEED 40
#define LEFT_TURN_TICS 365
#define RIGHT_TURN_TICS 370

#define LEFT 0
#define RIGHT 1

// global color values:
#define ORANGE 1
#define BLUE 2
#define YELLOW 3
#define PINK 4
#define GREEN 5
#define FLOOR 6



// global variable used for the confidence values in the colors.
int conf;


/*********************************************
 * void stop_wheels() - method used to brake   *
 * the motors.                                 *
 * Taken from dmiller-super-demo.ic            *
 *                                             *
 *********************************************/
void stop_wheels()
{
    motor(L_MOTOR,-20);
    motor(R_MOTOR,-20);
    sleep(0.05);
    ao();
}


/*********************************************
 * void straight(int tics) - method for driving*
 * the robot straight tics tics, where tics is *
 * the sum of the left and right encoder tics. *
 * The idea to scale the motor power by        *
 * (n*STRAIGHT_SPEED)/10 was taken from        *
 * the go_straight() method in                 *
 * dmiller-super-demo.ic                       *
 *********************************************/
// tics = total of left and right tics, 10000 ticks ~ 6ft.

void straight(int tics)
{
    int l_enc, r_enc;
    int totalTics = 0;
    reset_encoder(L_ENCODER);
    reset_encoder(R_ENCODER);
    while (totalTics < tics)
```

```
      {
        l_enc = read_encoder(L_ENCODER);
        r_enc = read_encoder(R_ENCODER);
        totalTics = l_enc + r_enc;
        if (l_enc > r_enc)
          {
            motor(R_MOTOR, STRAIGHT_SPEED);
            motor(L_MOTOR, (6*STRAIGHT_SPEED)/10);
        }
        else
          if (r_enc > l_enc)
            {
              motor(L_MOTOR, STRAIGHT_SPEED);
              motor(R_MOTOR, (7*STRAIGHT_SPEED)/10);
          }
          else
            {
              motor(R_MOTOR, STRAIGHT_SPEED);
              motor(L_MOTOR, STRAIGHT_SPEED);
          }
    }
    stop_wheels();
    ao();
}

/*****************************************
 * void turn_90(int dir) - method used to *
 * turn the robot to either right or left.*
 * Note: only makes 90 degree turns.      *
 * This was inspired by the turn_tics()   *
 * method in dmiller-super-demo.ic, but   *
 * several modifications have been made.  *
 *****************************************/
void turn_90(int dir)
{
    int tics, speed, tot_tics;
    int r_enc, l_enc;

    if (dir == RIGHT){
        speed = -30;
        tot_tics = RIGHT_TURN_TICS;
    }
    else{
        speed = 30;
        tot_tics = LEFT_TURN_TICS;
    }

    // reset encoders
    reset_encoder(L_ENCODER);
    reset_encoder(R_ENCODER);

    // measure encoders for turn
    while (tics < tot_tics) {
        r_enc = read_encoder(R_ENCODER);
        l_enc = read_encoder(L_ENCODER);
        tics = r_enc + l_enc;
        motor(R_MOTOR, speed);
```

```
        motor(L_MOTOR, -speed);
    }

    // stop the turn
    motor(R_MOTOR, -speed/2);
    motor(L_MOTOR, speed/2);
    sleep(.1);
    ao();
}

// make a right turn.
void right_turn()
{
    turn_90(RIGHT);
}

// make a left turn.
void left_turn()
{
    turn_90(LEFT);
}

// make a 180 degree turn.
void turn_180()
{
    left_turn();
    left_turn();
}

// track orange color.
int trackOrange()
{
    return trackRaw(150,200,40,100,10,35);
}

// track blue color.
int trackBlue()
{
    return trackRaw(65,90,40,80,80,150);
}

// tack yellow color.
int trackYellow()
{
    return trackRaw(120,150,16, 80,16,19);
}

// track pink color.
int trackPink()
{
    return trackRaw(179,230,61,120,30,80);
}

// track green color.
int trackGreen()
{
    return trackRaw(70,110,80,140,16,20);
```

```
}

// track floor.
int trackFloor()
{
    return trackRaw(120, 140, 60, 90, 20, 50);
}


/*********************************************
 * int getColor() - method used in cdetect for  *
 * detecting colors; dependent on conf value.   *
 * returns the color read by checking the       *
 * confidence value.                            *
 *                                              *
 *********************************************/
int getColor()
{
    int confidence;
    int confidence2;
    // checking for pink
    confidence = trackPink();
    if (confidence > 10)
      {
        conf = confidence;
        return PINK;
    }
    // checking for orange
    confidence = trackOrange();
    if (confidence > 30)
      {
        conf = confidence;
        return ORANGE;
    }
    // checking for blue
    confidence = trackBlue();
    if (confidence > 10)
      {
        conf = confidence;
        return BLUE;
    }
    // checking for green
    confidence = trackGreen();
    if (confidence > 30)
      {
        conf = confidence;
        return GREEN;
    }
    // check yellow make sure is the right color
    confidence2 = trackYellow();
    if (confidence2 > 30)
      {
        conf = confidence2;
        return YELLOW;
    }
    else return -1;
}
```

```
// very simple method used to get on top of a color if it seems
// to find one.
void getOnTop()
{
    straight(20);
}

// compensate will make the robot move properly to make a turn
// once it finds a color.
void compensate()
{
    straight(500);
}


/*************************************************************
 * main part of the code - our basic implementation is to run *
 * a process in order to make the robot move forward, then it *
 * checks for a color, it stops the motors and performs the   *
 * appropriate actions once it finds one. In case it never    *
 * finds any color the robot stops.                           *
 *                                                            *
 *************************************************************/
void main()
{
    int ticks=10000;  // 10000 ticks ~ 6 ft.
    int straight_pid; // process id for straight
    int color; // variable for checking color
    // initilize camera
    init_camera();
    clamp_camera_yuv();
    // enable encoders
    enable_encoder(L_ENCODER);
    enable_encoder(R_ENCODER);
    // wait for someone to press start button
    printf("Press start to let loose!\n");
    while(!start_button());
    // start the straight pid
    straight_pid = start_process(straight(ticks));  // start the robot!
    while(1)
    {
        // get the color
        color = getColor();
        // if color is not the floor then check the color
        if(color != -1)
        {
            kill_process(straight_pid);
            stop_wheels();
            sleep(1.0);
            color = getColor();
            if (color == GREEN)
            {
                // go straight
                printf("Green - %d\n", conf);
                straight_pid = start_process(straight(ticks));
            }
```

```
else if (color == PINK)
{
    // found pink make a 180 degree turn
    stop_wheels();
    printf("Pink - %d\n", conf);
    turn_180();
    straight_pid = start_process(straight(ticks));
}
else if (color == BLUE)
{
    // found blue make a right turn
    stop_wheels();
    printf("Blue - %d\n", conf);
    compensate();
    right_turn();
    straight_pid = start_process(straight(ticks));
}
else if (color == ORANGE)
{
    // found orange make a left turn
    printf("Orange - %d\n", conf);
    compensate();
    left_turn();
    straight_pid = start_process(straight(ticks));
}
else if ((color == YELLOW))
{
    // found yellow stop the robot
    stop_wheels();
    getOnTop();
    sleep(0.5);
    color = getColor();
    if(color == YELLOW)
    {
        printf("Course complete. - %d\n", conf);
        beep();
        beep();
        beep();
        beep();
        break;
    }
    else
    {
        straight_pid = start_process(straight(ticks));
    }
}
else if (color == FLOOR)
{
    // found the floor
    straight_pid = start_process(straight(ticks));
    printf("FLOOR... %d\n", conf);
}
else if (color == -1)
{
    straight_pid = start_process(straight(ticks));
    printf("ERROR!\n");
}
```

```
            else
            {
                straight_pid = start_process(straight(ticks));
                printf("BIG ERROR!\n");
            }
        }
        else
        {
            printf("It's the floor...\n");
        }
    }
    disable_encoder(L_ENCODER);
    disable_encoder(R_ENCODER);
}
```

Robot Code Documentation:

Our basic approach to the robot implementation is to have the robot running a continuous process that will make it go forward. The team figured that at any given point that the robot does not know what to do; it will just keep going forward. To give a brief overall overview of the project, the robot is to start out in any given foot by foot tile with a black tape around it. Then the robot will look for a 2 by 2 inch square and perform a function given by the color of this square, the color scheme is as follows: *Green-forward, Pink-reverse, Blue-Right, Orange-Left, Yellow-Stop.* Given the value of the color the robot is to do the respective action fully autonomously.

Data Structures:

Some of the data structures used in the code were simple definitions of constants and variables that are critical to the actual program. These included color code values, right and left motor values to communicate with the handy board, the straight speed, left and right turn ticks in order to make a 90 degree turn, and left and right encoder values. A global constant was also used to keep the value of confidence within each color that came from the camera. Most of the data for the code comes from single value variables stored in the program. The color code values in the program were derived from the *int getColor()* method, using the color tracking values for the camera the team put together the essential values that came from the *int getColor()* method. The right and left motor, alongside with the encoder left and right values came from the handy board itself; there is a predefined set of values in the handy board that are useful to talk to the various devices it is connected to. The straight speed was derived by the team; it is set to a value that gave us the optimum balance between speed and stability. The team also came up with the values in the right and left turn ticks, these were defined after days of testing the equipment to see which value gave us a precise 90 degree turn. Amongst other variables used in the main code are *ticks, and straight_pid,* these were used to keep control of the overall flow of the program.

Program Algorithms:

A very important and precise algorithm used in the implementation of the robot is the *straight(int ticks)* algorithm used to keep the robot moving in a straight line. The team had the help of shaft encoders to accomplish this goal; the overall design of the robot also gave us a critical advantage to do this. At first the *straight(int ticks)* algorithm resets the encoders to start counting the ticking values in each wheel; these values are essential since they give us how much each one of the two wheels is off compared to the other one. The algorithm then tries to compute the difference between the right and left wheel. Given a certain difference the algorithm will then compensate for any misalignments in the two back wheels by reducing the power supplied to either the left or right motor. During different stages of the design, we found that we needed to adjust the amount of power reduction on each side differently because certain unknown factors were causing the robot to veer to the left or to the right. This algorithm is implemented to go a certain number of ticks for a purpose: in case the robot misses a color or it gets lost, the algorithm will simply end the method and have the robot sit there idle. Our turning methods used the same principles as the straight algorithm, but the encoders were simply used to get a certain number of ticks count and no compensation was done whatsoever.

To obtain appropriate color readings from the CMUcam, we designed a simple algorithm. First, we created color tracking methods for each color that we were interested in: *trackOrange(), trackBlue(), trackYellow(), trackGreen(), trackPink(),* and *trackFloor()*. These methods simply made calls to the *trackRaw()* library function, supplying the YUV colorspace parameters that we determined experimentally for each color. Then, to get a color reading, we created a simple method, *int getColor()*. This method calls each color tracking method in succession and returns a corresponding constant if the confidence value for that color's tracking method is high enough. If no colors are recognized with enough confidence, a default value is returned. In this way, the color that currently predominantly fills the camera's window can be determined at any time.

The method called *compensate()* was used to force the robot to center its turns around the colored squares (i.e. after a color is recognized by the camera, the robot must move forward a certain amount before turning so that the robot will remain centered in the square). Another method called *getonTop()* was an algorithm used to get better color readings by placing the camera right on top of the color when it thinks it has found a color.

The main loop of our program starts a process to cause the robot to drive straight. It then continually senses the colors on the floor using *getColor()* and stops the robot when a color other then the floor is detected. After the robot is stopped, it senses the color again to obtain a more accurate reading and then instructs the robot to do one of several tasks depending on the color. These tasks include turning left, right, and around completely, driving straight, and stopping. If the robot sees a color it does not recognize, it simply goes straight again.

All other algorithms implemented in the program were designed to help aid the three main tasks of the robot, that is: *driving in a straight line, making turns, and getting*

color readings from the camera. These algorithms were successful at accomplishing the goal, however lots of time was spent finalizing the parameters used in such methods.

Team Organization Evaluation and Plans:

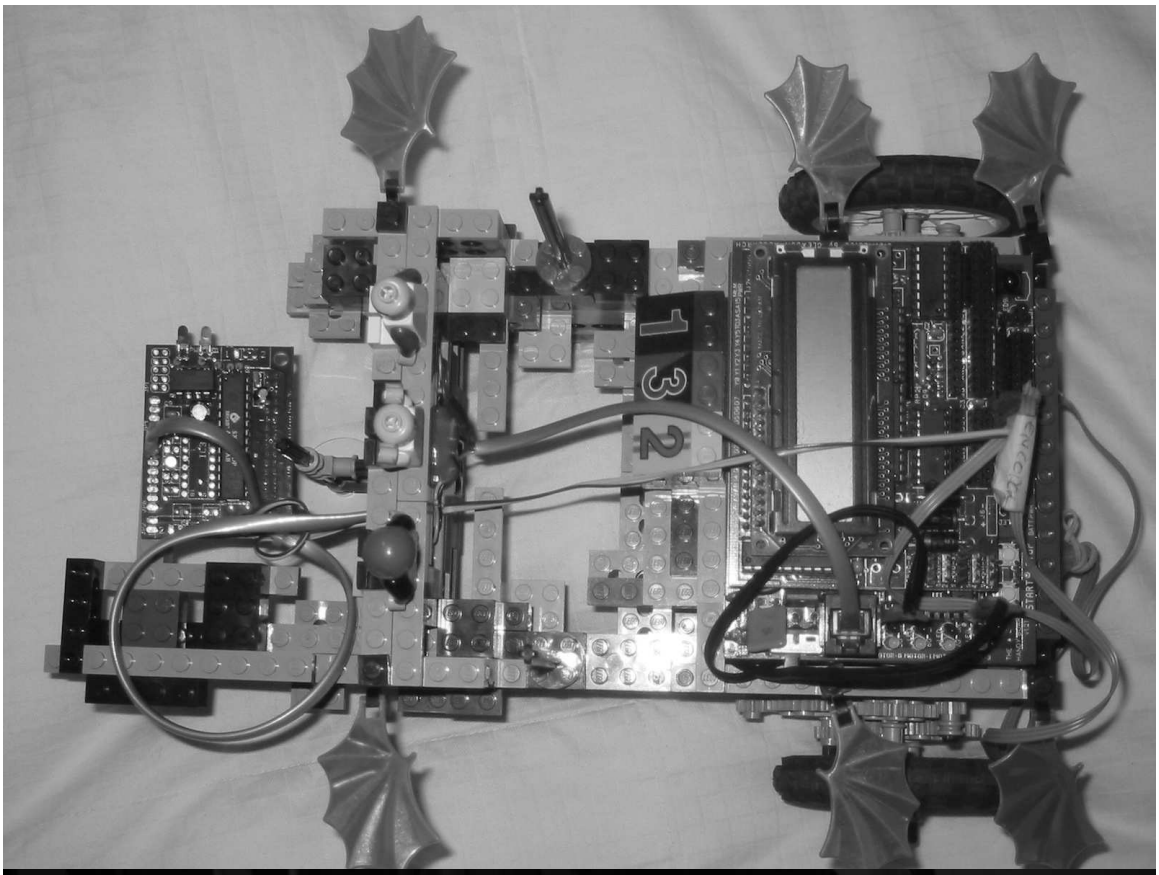
The team decided to use a concept of a rotating leader for the organization of the team. Camilo Reyes was assigned as the leader of this particular project. This style of organization will give each person on the team a chance to lead a project. One important job of the leader is to coordinate tasks and keep everyone informed on how things are going. The team also decided to allocate rotating tasks so that not one person in the group was given a general responsibility such as hardware, software, or paperwork; instead the team would assign one particular person to do one job, then another person would take on a different task as it seemed clear of what is to be done. This style of organization led us into working with different parts of the robot, and a general exposure to the project as a whole.

Given the small timeframe to complete this project the team was forced to arrange itself into a more democratic organization in which everyone was involved in every step of the decision making. As the project progressed the team went trough a decentralization phase where one person was given the robot to take home, and the next day the team would meet again to see what progress has been done and what changes have taken place on the robot. Lots of time was also spent during the fine tuning part of the project. For this last phase, the team as a whole went through a sleepless night of figuring out the proper parameters that would make the robot work properly. Our team organization suffered somewhat given the small timeframe, and small number of team members to complete the project; we believe this sort of project is meant more for one person, however having a group of members with different ideas working on the project helps a lot. The project was a success for the team; the team believes a successful demonstration was done at the end, but we wish that the robot would have been more accurate and that it had completed the course without “help” as it did in our test runs.

The team believes our overall organization looks good on paper, however the actual implementation of the project does not always seem to follow what has been written. Milestones were indeed completed in a timely fashion, every team member did their best to contribute to the project, and we believe the project was a success. The team organization worked out well, therefore a fall back plan was never needed; our team members stood together until the end and solved many number of challenges that came up as the project progressed.

Perhaps a primary lesson to be learned from this project is that building a robot is not as simple as it seems. When we read the description of the project, we assumed that it would be fairly easy. After all, our robot only needed to be able to perform three simple tasks: drive straight, turn accurately, and recognize colors. In reality, while all of those tasks proved to be more difficult than we anticipated, the true difficulty arose in the

combination of these tasks and in the appearance of unforeseen complications. For example, we initially achieved straight driving at distances of more than ten feet and accurate turns using a caster wheel in the front instead of the skid platter. But when we actually tested this design on a mock course, we found that the caster was catching on the electrical tape, rendering the robot useless. So we spent some time redesigning the front of the robot, only to find that this threw off our perfectly calibrated driving and turning routines. A similar setback occurred with our CMUcam code. We had routines that accurately recognized all of the colors in our test conditions, but when we went to the robotics lab to test our robot on a course we found that the different lighting conditions rendered our color recognition useless. Setbacks such as these seem to add up; because of these and many more, we were never as close to completing the project as we thought we were. Due to this unpredictable nature of robot design, we should plan to allow for more time to complete tasks for future projects.


1
3
2