Final Report: Project 1

Team 3:
Jonathan Siegel
Kumaresh Rajan
Prateek Duggal

February 16, 2004

1.0 Robot Design

Initial Design

This robot was initially designed with a two-wheel approach. This initial design consisted of two wheels roughly one inch wide and one and a half inches tall located on the center of the robot. These two wheels represented the robot's axis of symmetry with respect to the front and the rear of the robot. This wheel placement was proposed and used in order to minimize the amount of deviation from the axis of rotation in order to keep the robot from going off course.

Motors supplied in the robotics kit drove the wheels. The gear ratio used in this design to drive the wheels was a one to one ratio. This design was accompanied by stabilizers, both to the front and to the rear of the robot in order to keep the robot relatively level with the surface it was placed on. The stabilizers consisted of extremely thin wheels with the tires removed. This was chosen to try to minimize the amount of friction that the floor would create with the stabilizers. One unforeseen advantage for the wheels that were chosen for the stability was that the wheels chosen actually aided the robot in achieving a straight path of travel for a relatively long period of time.

The placement of the CMUcam on the initial design of the robot was in the center of the robot between the two driving wheels. The CMUcam was positioned so that the axis of rotation was located in the center of the viewing window. This camera position was desired to eliminate the need to make a course correction after the sensing of the color and before the turning of the robot. It was theorized that minimizing the amount of course correction required would simplify the implementation of the Interactive C code that would be required to drive the robot.

The rest of the body of the robot consisted of a very simple support structure for the wheels, camera, and Handy Board. The support of the robot did not need to be very extensive because the robot was designed to minimize the amount of shearing force that was to be placed on the support of the wheels and motors. The Handy Board was mounted on the top of the robot in a very simple rectangular support structure. The vertical compression of the robot by the Handy Board was sufficient to keep the pieces of the robot from separating while the robot was moving.

Modified Design

The initial design was very good except for the problem of the wheels slipping. The tires used did not have sufficient traction, and this resulted in inconsistent turning. This design was thus altered by replacing the two wheels with two treads.

The two-tread design used the same housing for the Handy Board, but completely replaced the rest of the robot. In this new design, the motors were

set more to the back of the robot and drove the treads using a one-to-one gear ratio. This ratio was used because the team wanted a moderate speed accompanied by enough torque to power the treads.

The treads spanned most of the body of the robot, and actually maintained the ability of the robot to turn without significant, if any, deviation from the axis of rotation. The treads did, however, create a much greater shearing force on the structure of the robot and would cause the pieces to pull apart from one another. This resulted in a large problem with gear slipping, so more support had to be added. The supports used were attached to the sides of the pieces that were being separated, and bound them securely to one another. This addition solved the gear slipping issue that had arisen.

Another significant addition to the robot was the addition of optical encoders to make the turning problem a digital problem. These sensors were used in conjunction with wheels supplied in the robotics kit that had holes of uniform distance from the center of the wheel as well as uniform spacing around the wheel. This ensured that the reading of the optical encoders was consistent. These sensor units were placed on each tread to ensure the consistent turning of the robot. In order to improve the accuracy of the sensors, the wheels were connected via a four-to-one gear ratio with the speed of the treads. Maximizing the amount of revolutions of the sensor wheel over the treads will allow the sensor to take many more readings thus ensuring that the treads are moving correctly with respect to one another. The addition of these sensors almost trivialized the problem of turning the robot at consistent and accurate angles.

The last major design alteration involved the placement of the CMUcam. The original design was proposed and constructed before the camera was properly tested, and it was later discovered that the camera required a significant amount of light in order to read the colors effectively. Many different positions and alignments were proposed and considered, and the decision came to mount the camera a couple inches out from the front of the robot and align the camera so that it points straight down. This will ensure a consistent viewing window as well as allow enough light for the camera to operate satisfactorily. The issue of proper support for the camera was solved with a rubber band and a large garbage tie. This ensured that the camera would not move while operating the robot, ensuring a consistent viewing angle.

The major problem with the robot design is not so much a design problem, but a problem with the motors. The motors were found to be inconsistent, and completely dependent on the amount of battery power left in the Handy Board. This resulted in many inconsistent tests resulting in significant amounts of extra testing.

2.0 Robot Software Design and Code

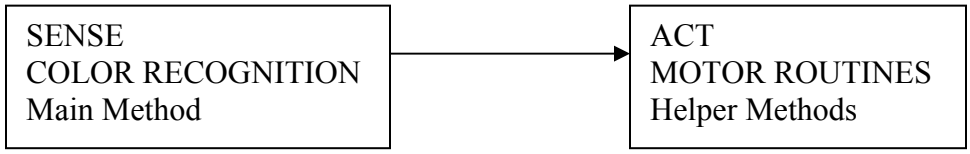
2.1 Introduction:

After testing the responsiveness and accuracy of the robot using Interactive C it became clear that simple algorithms were needed to operate the robot. Extra complexity in the code proved to be more of a hindrance than a benefit because of the unpredictability of the equipment in the robotics kit. Also, debugging the robot would have proven to be much more difficult with a complex design because it would be harder to tell if the robot was behaving incorrectly due to bad programming logic or if the problems were caused by inaccurate sensor readings. In keeping with this philosophy of simplicity it was decided that each module would only contain the bare essentials. For example, in doing the color recognition extra cmucam library functions like setwin() and variables such as track_x were not included in the code because these extra features caused the robot to behave unexpectedly.

2.2 Data Structures:

No data structures were used in the source code.

2.3 Functions:



main()
The main method in the code was responsible for sensing the environment (i.e. color recognition) and calling the appropriate helper function. The entire color recognition module was place in the main method and “if conditionals” were used to select the appropriate actions. Color was sensed using the trackRaw() method and if a high enough track_confidence was found then a helper method would be called to change the position of the robot, else the default go straight action would be preformed. The main method would terminate after the yellow color was found by setting the yellowfound variable to equal one.

forwardclicks()
Routine that is called after the robot has found a color. The routine moves the center of the robot over the colored square in order to align the robot with the maze.

reset_encode()
This method when called will reset both the left and right slot sensors to zero.

calc_turn()
Method returns the sum of axle rotations for the left and right wheel.

corr_err()
Note this code was taken from group 1 from the spring 2003 term of intelligent robotics. The code was modified to suit the needs of our robot.
Code calculated which motor of the robot rotated the fastest using the encoder values from the slot sensors. If the difference of the motor outputs for the robot is greater that the TURN_TOLERANCE (10 axle rotations) then the code would try to realign the orientation of the robot.
The code was never implemented in the final version of the robot because of the difference of power levels in the robot's motors (i.e. because the left motor was always stronger then the right motor the code would always indicate left drift even if no drift occurred)

forward()
Code tried to make the robot move in a straight line. The motors were allowed to pulse for .37 seconds and then went into idle for 1 second. The pulsing allowed the viewing window of the camera to examine every inch of the floor, and idling the motors gave enough time for the camera to sense the colored square before the motors began to move again. The "proper" power levels for the motors were found after intensive testing of the robot. In theory setting the power levels of each motor to the same value would allow the robot the to move in a straight line, but this was not the case for our robot due to the huge power difference between the left and right motors. In order to compensate for the power difference the left motor power level was made to be significantly less than the power level of the right motor.

reverse()
When routine is called the robot will perform a right turn for 180 degrees. Both slot sensors are reset and the motors are allowed to turn until the sum of the motor rotations equal TURN_TIME_REV (266 axle rotations). corr_err() was supposed to called to compensate for drift, but given the inaccurate readings of the motors the method was commented out.

left()
When routine is called the robot will perform a left turn for 90 degrees. Both slot sensors are reset and the motors are allowed to turn until the sum of the motor rotations equal TURN_TIME_L (120 axle rotations). corr_err() was supposed to called to compensate for drift, but given the inaccurate readings of the motors the method was commented out.

right()
When routine is called the robot will perform a right turn for 90 degrees. Both slot sensors are reset and the motors are allowed to turn until the sum of the motor

rotations equal TURN_TIME_R (134 axle rotations). corr_err() was supposed to called to compensate for drift, but given the inaccurate readings of the motors the method was commented out.

2.4 The final Code

```
#use "cmucamlib.ic"

/*
Motor Constants
POS_TURN                : MOTOR POWER LEVEL FOR A POSITIVE TURN
NEG_TURN                : MOTOR POWER LEVEL FOR A NEGATIVE TURN
TURN_ROTATIONS_REV : AMOUNT OF AXL ROTATIONS FOR A REVERSE
TURN_ROTATIONS_R    : AMOUNT OF AXL ROTATIONS FOR  RIGHT TURN
TURN_ROTATIONS_L    : AMOUNT OF AXL ROTATIONS FOR A LEFT TURN
LEFT_MOTOR              : INTEGER REP. OF THE LEFT_MOTOR
RIGHT_MOTOR             : INTEGER REP. OF THE RIGHT_MOTOR
FORWARD_CLICK       : NUMBER OF AXL ROTATIONS FOR A FORWARD CLICK (ie the small amount
of clicks needed to be on top of
colored square)
TURN_TOLERANCE      :USED TO COMPENSATE FOR ROBOT DRIFT DURING A TURN.  IF THE DIFFERNCE BETWEEN MOTOR ROTATIONS IS GREATER THAN TURN_TOLERANCE THEN THE ROBOT COMPENSATES FOR DRIFT
*/

#define POS_TURN 100
#define NEG_TURN -100
#define TURN_ROTATIONS_REV 266
#define TURN_ROTATIONS_R 134
#define TURN_ROTATIONS_L 120
#define LEFT_MOTOR 0
#define RIGHT_MOTOR 3
#define FORWARD_CLICK 30
#define TURN_TOLERANCE 10

/*
Max and Min RGB values of BLUE
MAX_BLUE_R: MAX R VALUE
MIN_BLUE_R: MIN R VALUE
MAX_BLUE_G: MAX G VALUE
MIN_BLUE_G: MIN G VALUE
MAX_BLUE_B: MAX B VALUE
MIN_BLUE_G: MIN B VALUE */
```

```
#define MAX_BLUE_R 62
#define MIN_BLUE_R 46
#define MAX_BLUE_G 170
#define MIN_BLUE_G 136
#define MAX_BLUE_B 130
#define MIN_BLUE_B 106

/*
Max and Min RGB values of GREEN
MAX_GREEN_R: MAX R VALUE
MIN_GREEN_R: MIN R VALUE
MAX_GREEN_G: MAX G VALUE
MIN_GREEN_G: MIN G VALUE
MAX_GREEN_B: MAX B VALUE
MIN_GREEN_G: MIN B VALUE
*/
#define MAX_GREEN_R 117
#define MIN_GREEN_R 93
#define MAX_GREEN_G 159
#define MIN_GREEN_G 127
#define MAX_GREEN_B 26
#define MIN_GREEN_B 20

/*
Max and Min RGB values of ORANGE
MAX_ORANGE_R: MAX R VALUE
MIN_ORANGE_R: MIN R VALUE
MAX_ORANGE_G: MAX G VALUE
MIN_ORANGE_G: MIN G VALUE
MAX_ORANGE_B: MAX B VALUE
MIN_ORANGE_G: MIN B VALUE
*/
#define MAX_ORANGE_R 180
#define MIN_ORANGE_R 144
#define MAX_ORANGE_G 69
#define MIN_ORANGE_G 55
#define MAX_ORANGE_B 16
#define MIN_ORANGE_B 16

/*
Max and Min RGB values of ORANGE
MAX_ORANGE_R: MAX R VALUE
MIN_ORANGE_R: MIN R VALUE
MAX_ORANGE_G: MAX G VALUE
MIN_ORANGE_G: MIN G VALUE
MAX_ORANGE_B: MAX B VALUE
```

```
MIN_ORANGE_G: MIN B VALUE
*/
#define MAX_YELLOW_R 180
#define MIN_YELLOW_R 144
#define MAX_YELLOW_G 166
#define MIN_YELLOW_G 134
#define MAX_YELLOW_B 18
#define MIN_YELLOW_B 16

/*
Max and Min RGB values of ORANGE
MAX_ORANGE_R: MAX R VALUE
MIN_ORANGE_R: MIN R VALUE
MAX_ORANGE_G: MAX G VALUE
MIN_ORANGE_G: MIN G VALUE
MAX_ORANGE_B: MAX B VALUE
MIN_ORANGE_G: MIN B VALUE
*/
#define MAX_PINK_R 176
#define MIN_PINK_R 136
#define MAX_PINK_G 18
#define MIN_PINK_G 16
#define MAX_PINK_B 16
#define MIN_PINK_B 16

/*
forwardclicks(); Is a robot routine that is called after the
robot has found a color.  The routine moves the center of the robot
over the colored square inorder to align the robot with the maze.
*/
void forwardclicks()
{
   reset_encode();
   printf("forward click \n");
   motor(RIGHT_MOTOR, 70);
   motor(LEFT_MOTOR, 30);
   sleep(0.6);
   ao();
}

/*
reset_encode();  This method when called will reset both the left and right
slot sensors to zero.
*/
void reset_encode()
{
```

```
    reset_encoder(0);
    reset_encoder(1);
}
/*
calc_turn(): Method returns the sum of axl rotations for the left and right
wheel.
*/
int calc_turn()
{
    return (read_encoder(0)+read_encoder(1));
}
/*
NOTE THIS CODE WAS TAKEN FORM GROUP 1 FROM THE SPRING 2003 TERM
OF INTELLIGENT ROBOTICS.  THE CODE WAS MODIFIED BY OUR GROUP.

CODE CALCULATED WHICH MOTOR OF THE ROBOT ROTATED THE FASTEST USING
THE ENCODER VALUES OF THE ROBOT.  IF THE DIFFERENCE OF THE MOTOR OUTPUTS FOR THE
ROBOT WAS GREATER THAN A TURN_TOLERANCE THEN THE CODE WOULD TRY TO REALIGN THE ORIENTATION
OF THE ROBOT.

THE CODE WAS NEVER IMPLEMENTED IN THE FINAL VERSION OF THE ROBOT BECAUSE
OF THE DIFFERENCE OF POWER LEVELS IN OUR ROBOT'S MOTORS (ie because the left motor was
always stronger then the right motor the code would always indicate left drift even if no drift occured)
*/

/*
void corr_err()
{
int diff = enc1-enc0;// positive is right drift
reset_encode();
   if(diff > TURN_TOLERANCE)
      {
        printf("right drift");
        while(calc_turn()<diff)
        {
        motor(right_motor, POS_TURN);
        motor(left_motor, NEG_TURN);
        }
    }
```

```
    if(diff < (-1* TURN_TOLERANCE)
      {
        printf("left drift");
        while(calc_turn()<(-1*diff))
        {
        motor(right_motor, NEG_TURN);
        motor(left_motor, POS_TURN);
        }
    }
}
*/

/*
```

forward(): Code tried to make the robot move in a straight line. The motors were allowed to pulse for .37 seconds and the went into idle for 1 second. The pulsing allowed the viewing window of the camera to examine every inch of the floor, and ideling the motors gave enough time for the camera to sense
the colored square before the motors began to move again. The power levels for the motors were used after intensive testing ot the robot. In theory setting the power levels of each motor to the same value would allow the robot the move in a straight line, but this was not the case for the robot due to the huge difference
power difference between the left and right motors. In order to compensate for the power difference the left motor power level was made to be significantly less than the power level of the right motor.

```
*/
void forward()
{
    reset_encode();
    while(calc_turn()<20)
      {
        motor(LEFT_MOTOR,25);
        motor(RIGHT_MOTOR,65);

    }
    sleep(0.37);
    ao();
    sleep(1.0);
}
/*
```

reverse(): When routine is called the robot will preform a right turn
for 180 degrees. Both slot sensors are reset and the motors are allowed
to turn until the sum of the motor rotations equal TURN_TIME_REV. corr_err()
was supposed to called to compensate for drift, but given the
inaccurate readings of the motors the method was commented out.

```
*/
void reverse()
{
   reset_encode();
   printf("reverse\n");
   while(calc_turn()<TURN_ROTATIONS_REV)
    {
      motor(LEFT_MOTOR, POS_TURN);
      motor(RIGHT_MOTOR, NEG_TURN);
   }
   ao();
   //corr_err();
}
/*
left(): When routine is called the robot will preform a left turn
for 90 degrees.  Both slot sensors are reset and the motors are allowed
to turn until the sum of the motor rotations equal TURN_TIME_L.  corr_err()
was supposed to called to compensate for drift, but given the
inaccurate readings of the motors the method was commented out.
*/
void left()
{
   reset_encode();
   printf("left \n");
   while(calc_turn()<TURN_ROTATIONS_L)
    {
      motor(LEFT_MOTOR, NEG_TURN);
      motor(RIGHT_MOTOR, POS_TURN);
   }
   ao();
   //corr_err();
}
/*
roght(): When routine is called the robot will preform a right turn
for 90 degrees.  Both slot sensors are reset and the motors are allowed
to turn until the sum of the motor rotations equal TURN_TIME_R.  corr_err()
was supposed to called to compensate for drift, but given the
inaccurate readings of the motors the method was commented out.

*/
void right()
{
   reset_encode();
   printf("right \n");
   while(calc_turn()<TURN_ROTATIONS_R)
    {
```

```
      motor(LEFT_MOTOR, POS_TURN);
      motor(RIGHT_MOTOR, NEG_TURN);
   }
   ao();
   //corr_err();
}

void main()
{
   int foundyellow=0;
   start_press();
   init_camera();
   enable_encoder(0);
   enable_encoder(1);
   //clamp_camera_yuv();
   while(foundyellow==0)
    {

      if(trackRaw(MIN_BLUE_R, MAX_BLUE_R, MIN_BLUE_G, MAX_BLUE_G,
              MIN_BLUE_B, MAX_BLUE_B)>45)
       {
         ao();
         forwardclicks();
         sleep(.5);
         right();
         printf("right=blue %d\n", track_confidence);
         ao();
         sleep(2.0);

      }

      else    if(trackRaw(MIN_GREEN_R,    MAX_GREEN_R,    MIN_GREEN_G,
MAX_GREEN_G,
                  MIN_GREEN_B, MAX_GREEN_B)>45)
        {

          forward();
       }
        else              if(trackRaw(MIN_ORANGE_R,              MAX_ORANGE_R,
MIN_ORANGE_G, MAX_ORANGE_G,
                   MIN_ORANGE_B, MAX_ORANGE_B)>45)
          {
            ao();
            forwardclicks();
            sleep(.5);
            left();
```

```
            printf("left=orange %d\n", track_confidence);
            ao();
            sleep(2.0);

          }
          else      if(trackRaw(MIN_PINK_R,      MAX_PINK_R,      MIN_PINK_G,
MAX_PINK_G,
                      MIN_PINK_B, MAX_PINK_B)>45)
            {
              ao();
              forwardclicks();
              sleep(.5);
              reverse();
              printf("reverse=pink %d\n", track_confidence);
              ao();
              sleep(2.0);
            }
            else            if(trackRaw(MIN_YELLOW_R,            MAX_YELLOW_R,
MIN_YELLOW_G, MAX_YELLOW_G,
                        MIN_YELLOW_B, MAX_YELLOW_B)>45)
              {
                forwardclicks();
                ao();
                beep();
                foundyellow=1;
              }
              else if(track_confidence<45)
                 {
                   forward();
                   printf("nothing %d\n", track_confidence);
                 }

    }
}
```

3.0 Team Organization Evaluation and Plans

3.1 Initial Plan & Time Line

As the initial plan stated, there were three major tasks being divided into separate groups working towards accomplishing the project goal and deadlines.

(a) The designer group consisting of the main design and secondary design group were to design the chassis / sensor locations and wheel / motor design.
(b) The coding was to be accomplished by dividing into three sub groups namely Color recognition, velocity control and turning control.
(c) The third and final group was to complete the testing and de-bugging.

We had defined the basic group structure to function democratically. We had decided not to delegate jobs to one specific person and decisions were to be made on majority basis.

3.2 Team Organization Evaluation:

We would like to divide the evaluation into four parts based on various sub-group

3.2.1 Team Division and Job Allocation
3.2.2. Communication between teams
3.2.2 Testing & Performance
3.2.4 Final Demonstration

3.2.1 Team Division and Job Allocation

Overall, our team organization worked quite well. Each team member completed his required individual tasks. However, with the impending time constraints and strenuous work schedule of each member, we faced quite some difficulties meeting our milestones and we re-adjusted most of the deadlines.

As described in the team organization plan, Prateek was to do the main design of the robot and Kumaresh & Jonathan were to be involved in the secondary design. The work on coding was shared as follows:

Color Recognition	: Jonathan
Velocity Control	: Prateek (Main) and Kumaresh
Turning Control	: Kumaresh

Finally , Testing / debugging was to be a combined effort .

Everyone contributed ideas which were incorporated into the robot design, which helped ensure that the conceptual design was correct. When one member hit a hurdle, other members contributed to the brainstorming process until a solution was found.
Most importantly the team members understood each other's time or potential constraints and helped each other tide away to finishing the designated work.

3.2.2 Communication between Teams

The best virtue of our group was open communication between all the members. We conducted lots of groups meetings lasting long and strenuous hours. All the members were very particular in attending all the group meetings.
Each member was very specific in expressing his view towards the issues and was patient to listen to other member's point of view. Also team was univocal in agreeing to decisions made.

3.2.3 Testing & Performance

The construction / design group had some setbacks initially as the two wheel design failed due to excessive slipping of the wheels. Because of this, we had to change the robot construction to a treaded design. Treaded design made us change almost the whole robot apart from the housing. This also made us re-design the original milestone planning.

Testing of the code on the final robot design took more time than we expected. Mostly because, although we thought that we had a perfect design and good code to back it up, the motor imbalance would not allow the robot to go straight. We tried for hours and hours, first using a error correction code and later on deleting the same, just to some how make the robot go straight. We changed the speed and turning code again and again. However, the camera program didn't take much time to grab all the colors.

The robot turned well and detected the colors properly, but only could not go straight.

3.2.4 Final Demonstration

As we could see , despite our tremendous effort, the final result was not satisfactory as we could not run the robot straight and we had to keep navigating the robot again and again to go in a straight line. Moreover, in the first run , the robot could not recognize the color , which while testing , it did for numerous number of time. However in the second run, the robot picked up all the colors (except for blue at one occasion).
But overall, it was not a good performance and we definitely should have done better.

3.3 Future plans for Team Organization and Planning

We think that we had planned our group activities and task allocation properly. Only thing we should concentrate more in future is on putting extra efforts on testing the robot on the demonstration room. Also we should rotate the group activities so as all the members can be exposed to minuteness of each work. But we will be retaining the democratic approach towards addressing issues as that we found is very effective.

P.S : “The idea of dividing the team organization evaluation was taken from submission by group 5 made for Project-1.”