A Development Summary and Post-
Mortem Consideration of:

Robotics Project 1
Sensing and Movement in Robotics

by

Group 6
Tao Zheng
Adam Barnett
Chris Madole

for

Dr. Dean Hougen
Introduction to Intelligent Robotics - CS 4023
School of Computer Science
University of Oklahoma

February 15, 2004

1) Introduction

Project 1 of "Introduction to Intelligent Robotics" entails the construction of a simple robot capable of correctly navigating a static course by periodically sensing the ground for changes in color. The following are the learning goals of this project (as taken from the CS 4023/5023 course website):

- To give you baseline experience with robotic sensing and acting on which you will build.
- To familiarize you with the particular hardware and the software environment that you will be using for some of the projects later in this course.
- To work out a team structure and method of operation that you will use (perhaps with modifications) throughout the semester. [1]

Teams were selected by the class instructor, and development on the project lasted approximately two weeks. The beginning of the project began with a planning meeting and generation of a proposal for development and a corresponding timeline with fall back plan. The project was concluded with an in class demonstration of the robot, a write-up of the development process, and an in class presentation of our development process. Excluding documentation, the project involves three main portions: robot design, robot code, and team organization.

2) Robot Design

Our robot was closely modeled after a tank or track vehicle, hence its given name "Tank!". Tank! employs a four wheel drive system via an extensive gearing configuration that interconnects all four wheels and all four motors. Initially Tank! was outfitted with rubber tracks, but due to the unreliability of the tension of these tracks, it was outfitted with very small wheels late in the development process. Figure 2.1 shows Tank! fully assembled as it operated during the demonstration.

[1] Dean Hougen, "Project 1 – Sensing and Movement," CS 4023/5023 – Intro to Intelligent Robotics – Spring 2004, Project 1. <http://www.cs.ou.edu/~hougen/classes/Spring-2004/Robotics/materials/project1.html> 15 Feb. 2004.

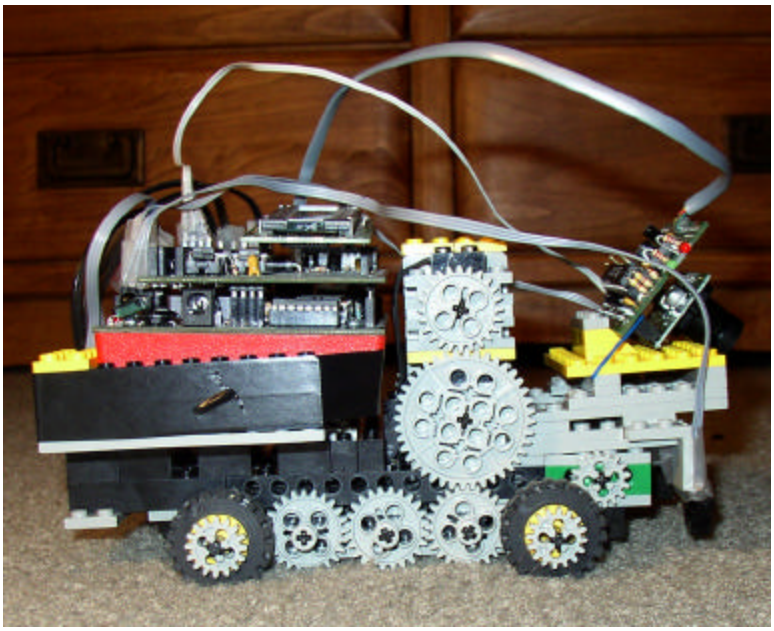

Figure 2.1 – Tank! Fully Assembled

Below is a detailed description and photographs of each part of Tank! including the body, suspension system, gearing, motors, and sensors.

2.1) Body

The body of Tank! is divided into four parts: a chassis, motor mount, Handyboard cradle, and a sensor mount.

The chassis is rectangular in shape and is built to support five axels; the first and last axel have wheels on each end, while the middle three axels are fitted on the end with gears. Each axel runs through two perpendicular side beams and two internal beams that allow for free rotation. On top of the beams are stabilizing boards that create a platform for the motor mount, Handyboard cradle, and sensor mount. Figure 2.2 below shows the chassis.

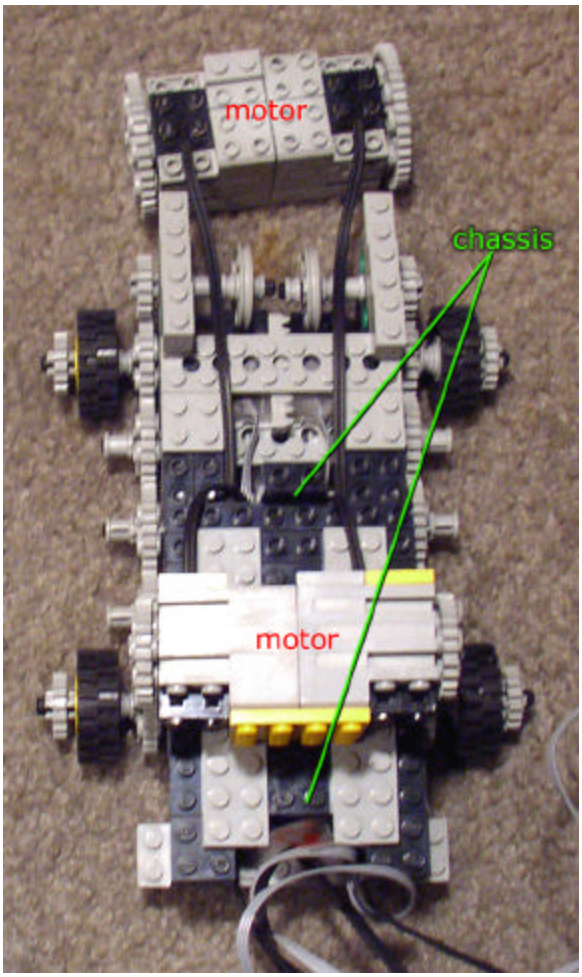

Figure 2.2 - Chasis

Connect directly to the chassis are the motor mount, sensor mount, and Handyboard. The Handyboard is supported by a cradle whose only purpose is to prevent the Handyboard from falling off Tank! while in motion. Figure 3.3 below shows the Handyboard cradle.

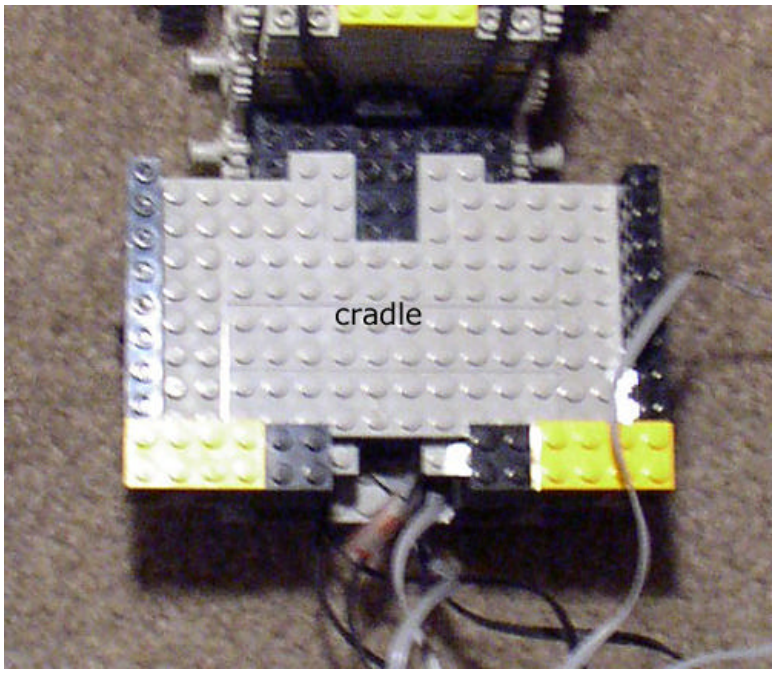

Figure 2.3 - Cradle

The motor mount and sensor mount will be discussed in greater detail below.

3.2) Suspension
Tank! did not employ an independent suspension system. The light weight of the robot did not facilitate a need for an independent system, so in order to simplify our design the chassis was connected directly to the axels in a static, non-moving fashion.

3.3) Gearing
Tank! employed a system of eight interconnected gears per side. Five z24 gears were used to connect the front and back wheels together in a 1-to-1 ratio, with the second and fourth gears moving in direction opposite of linear motion. This chain of gears assured that the front and back wheels would be moving at the same speed and that turns would be executed with a minimal amount of tire slipping. Figure 2.4 shows an outline diagram of the gear configuration.

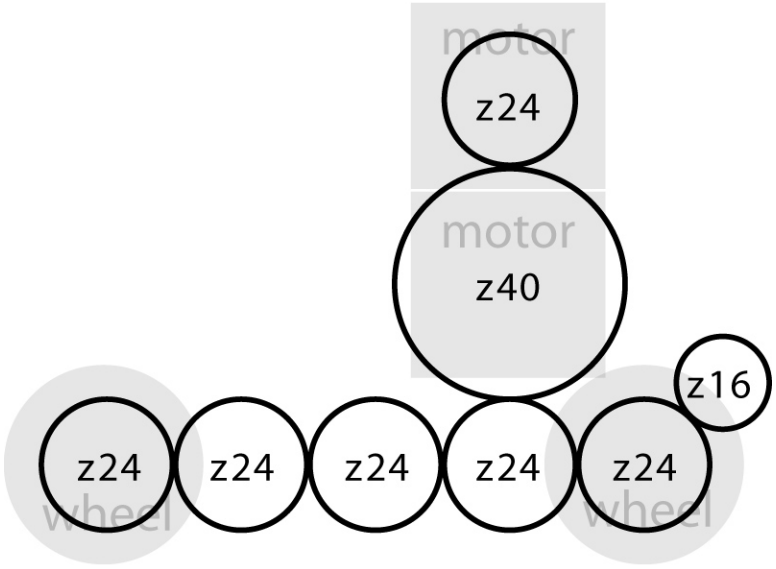

Figure 2.4 – Outline of Gear Configuration

Connected to the second axel back from the front of the vehicle is a z40 gear connected to the lower motor. The ratio between the z24 gear to the z40 gear is 10-to-6. Connected on top of the z40 gear is another z24 gear that is connected to a second motor. The z40 gear rotates in the direction of linear motion, while the upper z24 gear rotates opposite the direction of linear motion. Having two interconnected motors operating in parallel offered us a greater amount of power than just one motor. The original design did not call for this, but initial testing proved it to be necessary.

Finally, a small z16 gear is connected to the front of the first axel; the purpose of this gear is to turn a disc resting in between the encoder sensors that are used to count the number of ticks when executing turns. The benefit of using the small gears is improved precision in tick counting. For every two complete revolutions of the z24 gears, the z16 gears and thus the encoder turns 3 complete revolutions.

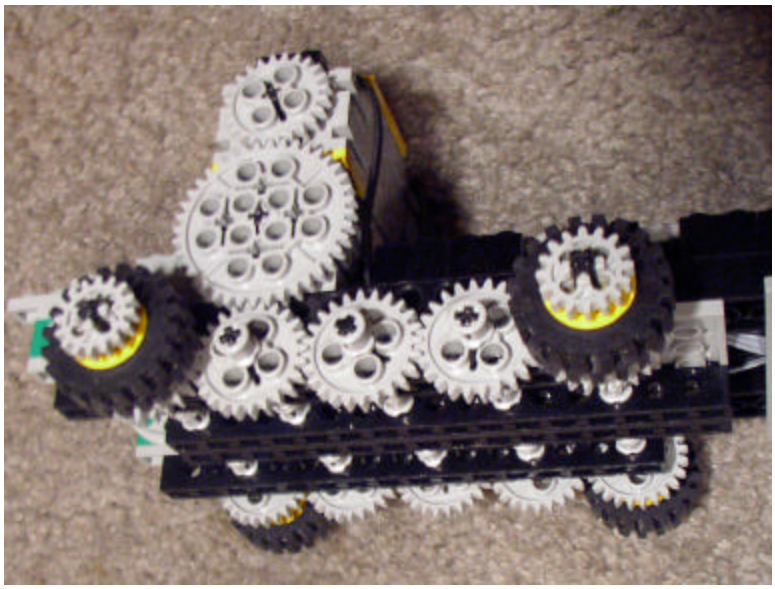

Figure 2.5 – Gear Configuration

Closely related to the gear configuration of Tank! are the four motors that power the robot.

2.5) Motor Mount

Our design employed four motors; two per side sitting directly on top of one another. The motors work in parallel, and are connected to the second axel of the chassis. Our original design called for only one motor per side, but after initial testing it became clear that one motor would not be enough to power Tank! consistently through turns. A second motor was added to each side to augment the power. In theory these motors operate in ratio from top to bottom of 5-to-3 revolutions. However, in practice this did not hold true. Inconsistencies between the motors prevented us from powering the motors in a direct relationship. The best relationship setting was found by testing; interestingly the final relationship used was 1-to-1 revolutions. Figure 2.6 below shows the motor mount.

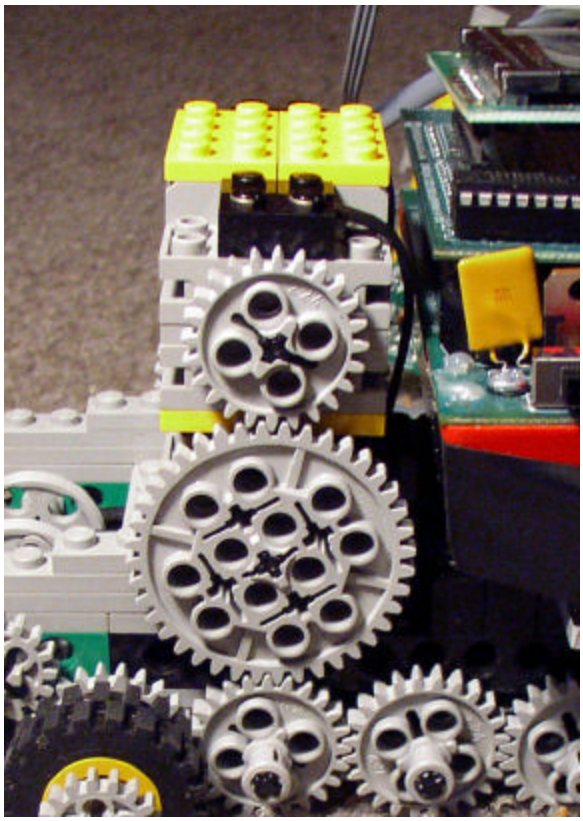

Figure 2.6 – Motor Mount

Positioned directly in front of the motors is the sensor mount.

2.6) Sensor Mount

Tank!'s sensor mount is a combination of three different types of sensors needed to complete the assigned tasks: a CMUcam Visions System, two Infrared "Top Hat" Reflectance Sensors, and two Digital Infrared "Break Beam" Sensors.

2.6.1) CMUcam Vision System

In order to sense color, the front portion of Tank! was equipped with a CMUcam Vision System mounted at a 45° angle towards the floor. Positioned in this manner, the tiny camera was able to scan the ground just ahead of the robot, while not blocking the light needed to sense the color accurately. The camera was held in place by a light rubber band. Figure 2.7 shows the CMUcam as part of the sensor mount.

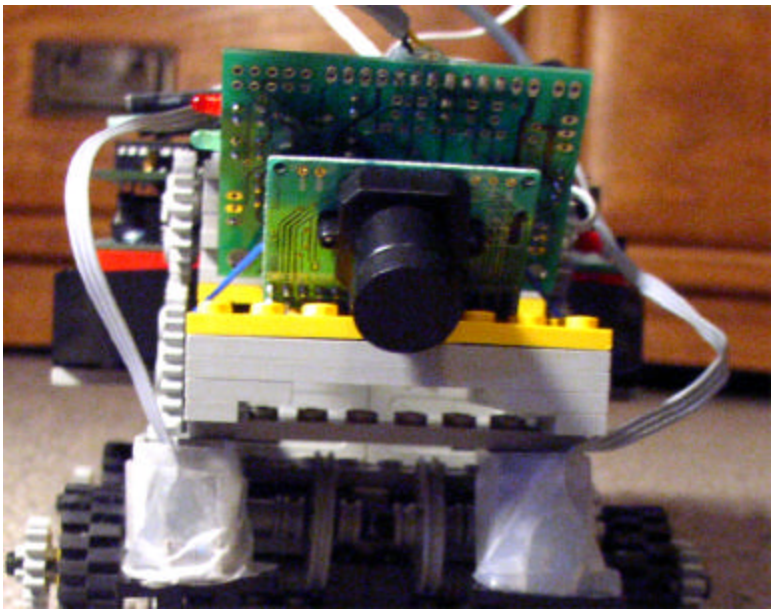

Figure 2.7 – CMUcam on Sensor Mount

Positioned below the CMUcam are the IR sensors.

2.6.2) Infrared "Top Hat" Reflectance Sensors

In order to find borders of squared and make alignment changes, Tank! used two top hat infrared reflectance sensors. These sensors can easily detect the difference between the color of the floor and black tape used to identify the squares. They were mounted on the front of the chassis at the bottom of the sensor mount, pointed down and affixed with tape, resting about 8mms off of the ground. Figure 2.8 shows the top hat reflectance sensors.

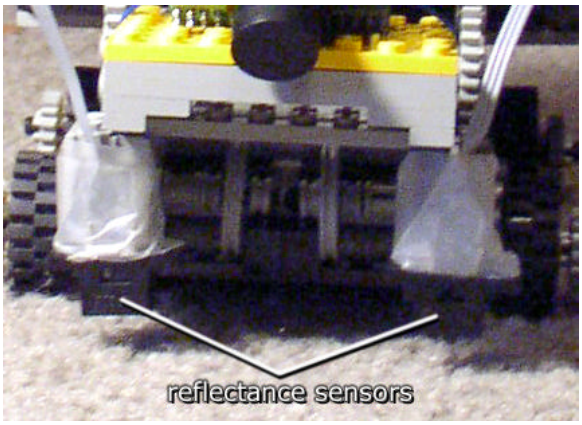

Figure 2.8 – "Top Hat" Reflectance Sensors

Underneate the camera and IR sensors are the "break beam" sensors.

2.6.3) Digital "Break Beam" Sensors

In order to track the revolution of the wheels for turning, it was necessary to employ break beam sensor with specially designed break beam discs. With each revolution of the disc, the IR beam between the sensor is broken four times by the revolution of the wheel. This allowed us to specify a certain number of desired ticks to turn the robot. In order to increase precision, we geared the break beam sensors to 3-to-2 revolutions against the actual wheels. Figure 2.9 shows the break beam sensors.

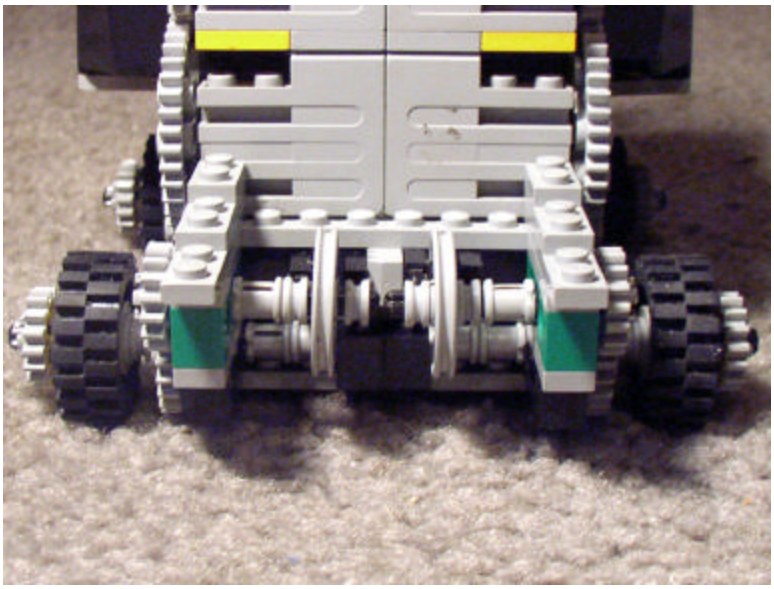

Figure 2.9 – Break Beam Sensors

Closely involved with the design of the robot was the production of software to run Tank!. A description of the code follows.

3) Robot Code

The code generated for the control and behavior of the robot is best described as a series of functional modules and the limited use of algorithms and data structures. The exact contents of the code can be seen as item 1.1 (group6_robot.ic – main controller code) and item 1.2 (track_color.ic color sensing code) in the Appendix.

3.1) Functional Modules

To meet the project requirements, the code must contain the following four functional modules: detect color, sense black tapes, go straight, and turn in a correct angle.

3.1.1) Detect color

In this project we needed to detect five different colors: green, pink, orange, blue, and yellow. We experimented with all sensors available in the robot kits, but we found that the CMUcam was the best device to detect and distinguish colors. To track these colors, we used the Interactive C Coding library function called "*trackRaw*"; it contains six parameters: (rmin, rmax, gmin, gmax, bmin, bmax). *trackRaw* has the ability to distinguish colors based on values of its parameters, when coupled with the CMUcam.

This camera can work in two different modes: RGB mode and YCrCb(YUV) mode. By default, the CMUcam works in the YUV mode, which translates to RGB as R for Cr, G for Y and B for Cb.

The difficulty in detecting color is identifying the correct ranges of values corresponding to different colors. If different ranges overlap, it becomes difficult to distinguish colors. During testing we discovered it was also difficult to distinguish between yellow and the off-white colored of the floor.

In trying to find the correct ranges, we experimented with many ways, including the CMUcamGUI program, a color picker, and YUV to RGB converter. Ultimately, we used the RGB converter in combination with testing to establish the correct color ranges for the five colors and the floor.

3.1.2) Sensing black tapes

We found two reasons that it was necessary to use sensors. Early on in the testing phase, it became clear that some alignment behavior would be necessary. In this project, one-foot squares formed by black tape are placed no more than six feet apart from one another and no closer than one foot apart from one another, we do not know how far the robot should travel to the next neighboring squares. So the robot has to keep sensing the black tape to know if it has reached a square or not. We have two IR sensors in the front to sense this black tape. If either one of the sensors reads a number greater than a preset threshold value, the robot will be told that a black tape has been sensed.

3.1.3) Going straight

Movement of the robot in a straight line is crucial to successful completion of this project. To accomplish this, we employed two schemes: one to give different speeds to the wheels on each side, and two to adjust the wheel speeds dynamically if the robot went off course.

During testing we discovered that Tank! tended to veer to the right if the motors were equally powered. To counteract this, we gave more power to the right motors.

To adjust the wheel speeds dynamically, we used encoders to detect the differences in wheel speeds. If the robot steers to the left, the right wheel is spinning faster than the left wheel, so we simply increase the power for the left motors and decrease power to the right motors. The opposite is applied if the robot veers right.

3.4) Turning

Tank! was required to make left and right turns. To achieve this, we determined experimentally the number of ticks needed to make a correct turn. Our design here was simplified because we knew that the robot had just been aligned at the last piece of black tape. This ensured that the robot was properly aligned before and after it made a turn. The pseudo code below describes the procedure for making right turns.

```
Turn_right( )
      Go straight and start tape sensing process;
      IF black tape found
            Stop and align;
      Start turning and start encoder checking process;
      IF encoder ticks greater than the preset value
            Stop and align;

Align( )
      WHILE(1)
            IF the reading of both sensors > threshold
                  Robot is aligned;
                  Break;
            ELSE IF only the reading of left sensor > threshold
                  Move the right wheels forward a little bit;
            ELSE IF only the reading of right sensor > threshold
                  Move the left right wheels forward a little
            bit;
            ELSE
                  Forward the robot a little bit.
```

3.2) Use of Algorithms

Tank! used no highly sophisticated algorithms to accomplish his tasks. The main looping process is described below in pseudo code:

```
Init_Camera;
WHILE   stop_button not pressed
   Track color and do actions;
   Go straight and start tape sensing process;
   If  black tape found
         Kill the tape sensing process;
         Stop and align;
```

3.3) Data Structures

Because this project is very simple in its composition, we did not find it necessary to use any advanced data structures. The program consists mostly of if statements and while loops. Global variables were to communicate among processes, but no other customized or pre-built structures were used.

4) Team Organization

Our team used a loose democratic style of organization. This worked well for our team, probably due to the diversity and maturity of the group. A brief description of our organization follows.

4.1) Overview

The team consisted of 3 people, Mr. Barnett, Mr. Zheng, and Mr. Madole.

4.1.1) Personnel Division and Task Assignment

During our initial team meeting we divided the work into three sections: Robot Build, Robot Code, and Documentation. By random assignment, Mr. Barnett was placed in

charge of building the robot, Mr. Zheng was committed to writing the code, and Mr. Madole was to handle the documentation process.

4.1.2) Leadership

No specific leader or manager was selected to head the group. At the initial meeting, it was clear that the people involved had experience working in teams, and the personalities and opinions blended well enough that no team leader was necessary to regulate differences.

4.2) Evaluation of Team Organization

4.4.1) Pros

- **Democratic Leadership**
  The democratic leadership prevented a dominating leader from emerging or dictating the development of the entire project. Consensus reaching before decision making and implementation
- **Synergistic / Teamwork Attitude**
  Use of age and maturity as an asset to allow a loose organization

4.4.2) Cons

- **Workload Balance**
  A balance of the workload was not wholly achieved.

- **Lack of Workload Balancing Mechanism**
  No mechanism for shifting the workload to keep it balanced

- **Limited Availability of Robotics Equipment**
  The basis of efficiency in the Project 1 organization relies on the ability for tasks to be completed in parallel. Unfortunately, this ability was limited because only one set of robotics equipment was available at any given time, but it was needed by all team members. Construction and design of the robot by Mr. Barnett obviously required the legos, sensors and handy board to be in his possession. This forced Mr. Zheng to wait until construction was partially completed to begin his development and testing of the camera to sense color. Mr. Madole was unable to begin the documentation of the robot design until after the demonstration because of the higher priority needs of Mr. Barnett and Mr. Zheng to have the robot.

- **Scheduling of Time**
  Due to underestimation of the scale and scope of the project, not enough time was scheduled for testing of the system. Extra meetings were held, but it was difficult to find times when all members could meet. On three occasions it was necessary for team members to work late in the night to complete their work. Table 4.1 gives a summary of Planned Development contrasted with Actual Development.

Table 4.1 - Summary of Development Timeline: Planned vs. Actual

Date	Planned Development		Actual Development
Jan 28	*Design:*	Discussion of proposal	Discussed proposal
	Code:	Discussion of proposal	Discussed proposal
	Doc:	Discussion of proposal	Discussed proposal
Feb 1	*Design:*	Begin robot build	Delayed; needed Handyboard and sensors
	Code:	Configure sensors	IR sensors identified not to work; camera had to be turned in to check for properly soldered connections
	Doc:	Prepare proposal	Proposal prepared, fallback plan left out by mistake
Feb 4	*Design:*	Finish Initial build and integrate with code	Began initial build
	Code:	Finish initial code and integrate with design	Began work on camera sensing
	Doc:	Document development process	Difficult to document process without having an active role. Schedules could not be coordinated to assist in design.
Feb 8	*Design:*	Finish testing cycle	Integration completed and testing cycle began
	Code:	Finish testing cycle	Integration completed and testing cycle began
	Doc:	Compile documentation	Unable to attend meeting; out of town on business commitment
Feb 11	*Design:*	Deliver build documentation	Testing of robot – two extra meetings required
	Code:	Deliver code documentation	Testing of robot – two extra meetings required
	Doc:	Compile documentation Prepare for post-mortem Rough draft of presentation	Helped with testing on Wednesday meeting, unable to attend other two meetings.
Feb 15	*Design:*	No deliverables	Unable to attend meeting due to other commitments
	Code:	No deliverables	Delivered code documentation
	Doc:	Post-mortem discussion Final compilation of documentation Mach delivery of presentation	Post-mortem discussion Began compilation of documentation

Taking this information into account, our team proposes the following plan of revision to our organization.

4.3) Revision Plan

Given the nature of the problems encountered during our robot development in Project 1, our team has agreed to make two major changes in the organization of our development process. We feel these changes will greatly improve our ability to complete

4.3.1) Division of Work

Instead of segmenting the workload into the 3 task areas of construction, code, and documentation, a new organization will be adopted that employs contribute from each team member to all areas of development. This will be accomplished by focusing on the cumulative completion of subtasks for each stage of development instead of having one team member wholly responsible for an entire stage.

Each member will have a small portion of work to complete for each task, as the tasks need completion. Using the coding stage as an example we will consider the contrast of the organization used in Project 1 versus the organization that will be used on the future. In Project 1, Mr. Zheng was responsible for coding each of the modules for sensing and mobilization. In the new team organization, all members will contribute a portion to the coding. For example, Mr. Zheng might write the code for IR sensing and alignment, Mr. Barnett might write the code for forward movement and turning, and Mr. Madole might write the code for color sensing.

4.3.1) Time Organization

In our original organization, time efficiency would come from accomplishing more than one task at once. Because of dependency on previous tasks and the limited availability of the robot kit, we found it difficult to accomplish more than one task at once. In the new organization, task completion will be scheduled in a more serialized fashion. Only one task will be completed at any given time, but the workload will be divided among three people. This should allow each task to be completed much faster and remove the necessity to spend long, consecutive hours working on the project.

Below is the Appendix where the code for Tank! can be found.

Appendix

1.1) group6_robot.ic

```
/* Project 1 -- Sensing and Movement
   Tao Zheng
   Adam Barnett
   Christopher Madole
**************************************************
In this project, we designed an autonomous robot moving
around a course of colored tiles. The movement the robot
making is based on the color it detects.
Green - go forward.
Pink - go reverse.
Blue - turn right.
Orange - turn left.
Yellow - stop.
*/
```

```
#use "track_color.ic"

#define GREEN 1
#define PINK 2
#define BLUE 3
#define ORANGE 4
#define YELLOW 5

//ports used by motors
#define LEFT_LOWER_MOTOR 0
#define RIGHT_LOWER_MOTOR 1
#define RIGHT_UPPER_MOTOR 2
#define LEFT_UPPER_MOTOR 3

//Encoder ticks for left turn
#define LEFT_TICKS 42
//Encoder ticks for right turn
#define RIGHT_TICKS 40
//Encoder ticks for reverse
#define REVERSE_TICKS 62

//digital port 12
#define RIGHT_ENCODER 2
//digital port 13
#define LEFT_ENCODER 3
//ports for right and left IR sensors
#define RIGHT_SENSOR 2
#define LEFT_SENSOR 3

//initial power to right motor
#define RIGHT_INIT_SPEED 11
//initial power to left motor
#define LEFT_INIT_SPEED 20

//Threshold value to detect black tape
#define BLACK_THRESHOLD 120
#define ALIGN_POWER 70
#define BACKOFF_TIME 100L

//global variables used for communication among processes
int right_encoder = 0; //right encoder value
int left_encoder = 0; //left encoder value
int black_tape_found = 0; //flag indicating black tape is found or not
int stop_flag = 0; //flag indicating yellow color is found or not

void main()
{
    init_camera(); //initialize the CMU camera
    //set the camera white balance for the current lighting conditions.
    clamp_camera_yuv();
    alloff(); //make sure all motors off at the beginning.
    while(!start_button());  //press start to start off the robot.

    //enable encoder channels before using them.
    enable_encoder(RIGHT_ENCODER);
    enable_encoder(LEFT_ENCODER);
```

```
    start_off();
    while(!stop_button())
      {
        //yellow color is not found.
        if (stop_flag == 1)
          break;
        track_color(); //detect color and do some actions
    }
    alloff(); //turn off all motors
}

//At the beginning, back off the robot a little bit
//such that the camera can see the color piece in the
//center of the square.
void start_off()
{
    int pid;
    go(-RIGHT_INIT_SPEED, -LEFT_INIT_SPEED);
    sleep(1.0);

    //detect black tape when the robot backs off.
    pid = start_process(check_tape());

    while(black_tape_found == 0)
      {
        msleep(1L);
    }

    //black tape found, stop detecting the black tape.
    kill_process(pid);
    //align the robot with the black tape.
    align();
}

//detect the color and do some action.
void track_color()
{
    int color;

    //keep detecting the color if nothing detected.
    while((color = check_color())==0);

    color = check_color();
    if (color == GREEN)
      forward();
    else if (color == PINK)
        reverse();
      else if (color == BLUE)
          turn_right();
        else if (color == ORANGE)
            turn_left();
          else if (color == YELLOW)
              stop();
}

//go in a straight line until a black tape is found. After that,
```

```
//align the robot with the black tape found.
void go_straight()
{
    int pid;
    go(RIGHT_INIT_SPEED, LEFT_INIT_SPEED);
    sleep(1.0);
    pid = start_process(check_tape());

    while(black_tape_found == 0)
      {
        msleep(1L);
    }
    //black tape is found. Reset the flag to be false.
    black_tape_found = 0;
    kill_process(pid);

    //align the robot with the black tape found.
    align();
}

//The robot forwards ahead.
void forward()
{
    go_straight(); //the robot travels throught the square.
    go_straight(); //the robot travels between two squares.
}

//The robot turns 180 degree.
void reverse()
{
    move(-70, 70, REVERSE_TICKS);
}

//The robot turns to the right 90 degree.
void turn_right()
{
    move(-70, 70, RIGHT_TICKS);
}

//The robot turns to the left 90 degree.
void turn_left()
{
    move(70, -70, LEFT_TICKS);
}

//The robot stops within a square.
void stop()
{
    go_straight();
    ao();
    stop_flag = 1;
    beep();
}

//The process to check the encoder ticks.
void check_encoder()
{
```

```
    reset_encoder(RIGHT_ENCODER);
    reset_encoder(LEFT_ENCODER);
    while(1)
      {
        right_encoder = read_encoder(RIGHT_ENCODER);
        left_encoder = read_encoder(LEFT_ENCODER);
        printf("%d,%d\n",left_encoder,right_encoder);
        msleep(10L);
    }
}

//The process to detect black tape.
void check_tape()
{
    int left_sensor = 0;
    int right_sensor = 0;
    black_tape_found = 0;
    while(1)
      {
        right_sensor = analog(RIGHT_SENSOR);
        left_sensor = analog(LEFT_SENSOR);

        //Black tape is found.
        if(left_sensor > BLACK_THRESHOLD || right_sensor > BLACK_THRESHOLD)
          {
            ao(); //stop the robot.
            printf("Black tape found\n");
            sleep(3.0);
            black_tape_found = 1; //set the flag.
            return;
        }
        else { //No black tape found.
            printf("left_sensor:%d\nright_sensor:%d\n", left_sensor,
right_sensor);
        }
    }
}

//align the robot with a black tape.
void align()
{
    int right_sensor, left_sensor;
    while(1)
      {
        right_sensor = analog(RIGHT_SENSOR);
        left_sensor = analog(LEFT_SENSOR);
        printf("%d,%d\n",left_sensor,right_sensor);
        // if both sensors detect black, the robot is already aligned
        if((right_sensor>=BLACK_THRESHOLD)&&(left_sensor>=BLACK_THRESHOLD))
          break;

        else if(right_sensor>=BLACK_THRESHOLD && left_sensor<BLACK_THRESHOLD)
            {
              //turn the robot to the right a little bit.
              go(-ALIGN_POWER,ALIGN_POWER);
          }
```

```
          else if(right_sensor<BLACK_THRESHOLD &&
left_sensor>=BLACK_THRESHOLD)
              {
                //turn the robot to the left a little bit.
                go(ALIGN_POWER,-ALIGN_POWER);
            }
            else if(right_sensor<BLACK_THRESHOLD &&
left_sensor<BLACK_THRESHOLD)
                {
                  //forward the robot a little bit.
                  go(RIGHT_INIT_SPEED,LEFT_INIT_SPEED);
              }
    }
    ao();
}

//move the robot forward at specified speeds.
void go(int right_power, int left_power)
{
    motor(RIGHT_UPPER_MOTOR, right_power);
    motor(RIGHT_LOWER_MOTOR, right_power);
    motor(LEFT_UPPER_MOTOR, left_power);
    motor(LEFT_LOWER_MOTOR, left_power);
}

//backoff the robot such that it can stay in the center of
//a square.
void backoff(long time)
{
    go(-RIGHT_INIT_SPEED, -LEFT_INIT_SPEED);
    msleep(time);
}

//move the robot at specified speeds for specified number of
//encoder ticks.
void move(int right_power, int left_power, int num_ticks)
{
    int pid;
    go_straight(); //get into the square the robot heads towards.
    //start checking the encoder ticks.
    pid = start_process(check_encoder());
    go(right_power, left_power);
    while(right_encoder < num_ticks)
      {
        msleep(10L);
    }

    printf("%d\n", right_encoder);
    //specified number of ticks has been reached. Stop checking encoders.
    kill_process(pid);
    ao();
    msleep(100L);
    backoff(BACKOFF_TIME);
    go_straight(); //continue to travel to the next square.
}
```

1.2 track_color.ic

```
/*Project 1 -- Sensing and Movement
   Tao Zheng
   Adam Barnett
   Christopher Madole
*******************************************************
This file contains codes detecting five different colors:
green, pink, blue, orange, and yellow.
*/

#use "cmucamlib.ic"

#define NONE 0
#define GREEN 1
#define PINK 2
#define BLUE 3
#define ORANGE 4
#define YELLOW 5
#define CONFIDENCE 80

int check_blue()
{
    if(trackRaw(0,120,40,120,100,240)>CONFIDENCE)
      return track_confidence;
    else return 0;
}

int check_green()
{
    if(trackRaw(0,100,20,150,16,18)>CONFIDENCE)
      return track_confidence;
    else return 0;
}

int check_pink()
{
    if(trackRaw(240,255,0,150,16,66)>CONFIDENCE)
      return track_confidence;
    else return 0;
}

int check_yellow()
{
    if(trackRaw(50,140,114,155,16,18)> CONFIDENCE)
      return track_confidence;
    else return 0;
}

int check_orange()
{
    if(trackRaw(180,230,20,150,16,18)>CONFIDENCE)
      return track_confidence;
    else return 0;
}

int check_color()
```

```
{
    if (check_blue() > CONFIDENCE)
      {
        printf("Blue: %d\n", track_confidence);
        return BLUE;
    }
    else if (check_orange() > CONFIDENCE)
        {
          printf("Orange: %d\n", track_confidence);
          return ORANGE;
      }
      else if (check_green() > CONFIDENCE)
          {
            printf("Green: %d\n", track_confidence);
            return GREEN;
        }
        else if (check_pink() > CONFIDENCE)
            {
              printf("Pink: %d\n", track_confidence);
              return PINK;
          }
          else if (check_yellow() > CONFIDENCE)
              {
                printf("Yellow: %d\n", track_confidence);
                return YELLOW;
            }
            else
              {
                printf("Nothing...\n");
                return 0;
            }
}
```