CHAPTER 1

INTRODUCTION

1.1 Background of study

In its early inception, robots were used as human replacements for hazardous work that requires repetition, precision and endurance. Most of them were initially confined to just manufacturing lines and automotive works. However, as time goes by, their reliability and stability in doing their job continuously again and again became recognised by a bigger audience, and today robots have integrated themselves more into the human society. They are no longer confined to just the manufacturing industry, as now they are increasingly being utilised in other fields such as agriculture, healthcare and surgery, physics, energy, research and even entertainment. In several places around the world, they are becoming a common sight, acting as caregivers, housekeepers and even educators. Their disposability and immunity against hazardous agents make them ideal tools in the military, law enforcement, and search and rescue.

1.2 Problem Statement

Maintenance and inspection is an essential process that needs to be done on a routine basis in every plant, factory and works. However, the task becomes more complicated when the plant in question deals with materials that are in general hazardous to human health, such as radioactive material in nuclear power plant, corrosive chemicals in a plastic factory or toxic fumes from sewage treatment facility. This problem is further compounded by the fact that many places that require inspection are often rather remote and too small for humans to even access. The personnel in charge of the maintenance tasks run the risk of being exposed to these harmful materials every time they need to inspect the place. Even if they are fully equipped with protective gear, there is a chance that something will go wrong, especially when there is an emergency situation due to leaks or fires occurring in critical areas.

As such, replacing humans with robots to do these hazardous work is an attractive prospect. However, before such a thing can be done, there are many factors that needs to be thought out and considered. Apart from the obvious financial and technological constraints, those who will be in charge of operating and taking care of the robot must also consider how the robot will be designed to operate in its intended environment. Several crucial deliberations include: how the robot will navigate around, whether it should be autonomous, manually operated or both, the sensors needed to allow the robot to receive input from its surroundings and whether the robot will be interacting with the environment passively or actively.

Therefore, it is imperative that the design of the robot caters specifically for its environment so as to ensure that it will be able to execute its intended tasks perfectly and without hassle. A reliable robot, in this case, is very crucial as its tasks are delicate, and any errors from inefficient design and implementation can result in huge losses to those who had depended on it.

1.3 Objectives and Scope of Study

The objective of this project is to design and create a robot that is capable of moving a pre-planned route and check for any fires through its IR sensor. The robot is intended to be lightweight and easy to manufacture, using commercially available and inexpensive materials as opposed to the current ones used in the industry.

The scope of the project study is limited to the study of the robot design, serial connection between the many systems of the robot, as well as the fire detection capabilities of the robot. At the end of this project a working prototype that showcases its abilities is expected.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

The idea of using robots as human replacements for hazardous inspection jobs is not an original one, as many others had also though out and implemented similar ideas. Most of the time, these ideas are more complex than the one proposed in this report, as they often involve multiple systems, with more manpower and monetary investments.

Yamamoto [1] proposed a basic idea, in that an inspection robot system is developed for use in nuclear plants. The robot can detect any type of abnormality using a T.V. camera, a microphone, and an infrared camera. It is an efficient approach, as it aims to increase inspection reliability through multiple sensors which preventing risk or radiation exposure. The robots are made to be compact and run along a guided rail, which minimises maintenance and simplifies navigation. On the other hand, Kawauchi, Shiotani, Kanazawa, Sasaki and Tsuji [2] opted for an actual humanoid robot, which can go around just like a normal human would by the help of RFID tags and autonomous tele-operation fusion control. In this approach, the robot replaces the human operator himself. Rather than confining itself to a single specialised job, the robot can be used for tasks of multitude nature. However, such a method would be difficult to do as the robot is very complex.

Most of the time, these robots are made to just carry out inspections in very specific areas. Several common designs are used in sewer pipes [3][4] and nuclear power plants[5]. The specialisation of tasks ensures that the robots are able to fully perform their intended usages with minimal drawbacks. Some of

these designs incorporate sophisticated sensors, for example to detect cracks[6] in order to further enhance its effectiveness.

However, the ability to function in hazardous conditions led robots to not just being adopted in plant inspection tasks but also in the fields of search and rescue. Murphy, Kravitz, Stover and Shoureshi [7] proposed using robots for search and rescue operations in collapsed mines. A number of people deal with not the creation of new systems, but rather new methods to implement already existing systems. There are several works devoted to controlling the behaviour of robots in hazardous conditions [8], or using a group of robots, instead of one, to get the job done[9]. Considering all factors, there are several trends that are common in today's inspection robots, which are listed below;

- Task specialisation – sewer, plant, fire extinguisher
- Autonomy; fully automatic, semi automatic or remotely controlled
- Use of different sensors – temperature, infrared, crack sensors
- Guidance system – line-tracking, robot vision, RFID tags

2.2 Task Specialisation

Specialisation defines what the robot design will be. For a hazard inspection robot, this mainly determines how the robot should move and what elements it should be capable of withstanding. As part of this project's job scope, the robot commutes around using wheels, as its environment comprises of flat surfaces that are easy to move on. The robot must also be robust enough to withstand minor elements present in the environment such as heat and dirt. Extreme heat and other applicable hazards must be avoided at all times whenever possible, as the robot is designed to just detect and report on the presence of such elements, not approach them and try to eliminate them.

2.3 Autonomous Control and Navigation

The robot should be designed with at least a semi autonomous capability in mind. This is necessary as the robot is meant to patrol select areas on a routine basis; it is more practical to have the robot automatically patrol its routes than to have an operator directly controlling the robot every time. The robot must also be able, upon detection, to alert the operator of any abnormalities. Being autonomous means that the robot must have some kind of navigation system to guide itself around. An ideal choice is to have a robust guidance system that is easy to implement and does not require sophisticated parts. As such, a line-tracking guidance system can be used to navigate the robot autonomously around its environment.

2.4 Sensor System

The sensors are integral to the robot in a sense that they allow the robot to receive input from the surroundings, whether from touch, vision, smell, sound or taste. In the context of this project, the only sensor used is the IR camera. The IR camera can detect IR signatures coming from flames or any objects emitting heat. In order to make the system more accurate and less prone to false positives, heat detectors should also be used. However, in this project the implementation of the sensor is kept simple by using only one type of sensor, and it is assumed that all the hot objects encountered by the robot are detectable via the IR camera and all IR light captured by the camera comes from the heat source. Other sensors used are line detectors for navigation guided by the lines on the floor.

CHAPTER 3

METHODOLOGY

3.1 Procedure Identification

In order to ensure that the project goes according to plan in an ordered fashion, a flowchart is used. The flowchart is used as a guide for the project, to determine what task should be done and in what order it should be done. The said flowchart is shown in Figure 1.

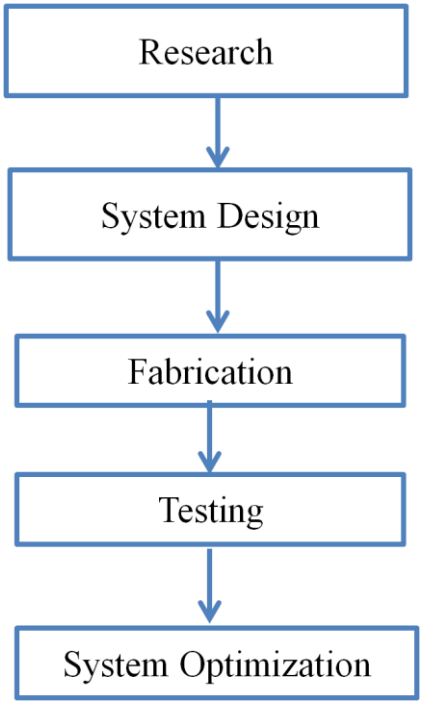


Figure 1: Flowchart of the project

3.1.1 Research

The research phase is the first phase of the project. The research conducted on this project involves study on previous and existing hazard inspection robot and its derivatives, how an autonomous robot works, I^2C connection and programming. Much time has been devoted to the understanding of the I^2C connection and its master-slave capabilities.

3.1.2 System Design

In this phase, all the aspects from the research phase are brought together to the project at hand. At this point the robot is designed according to the specifications. These specifications involve designing the circuits, sensors and the hardware necessary for the robot to operate. This is a crucial phase as the effectiveness of the proposed designs will determine whether the robot will be able to function as intended. The project is currently at this stage.

3.1.3 Fabrication

After the software and hardware parts of the robot have been devised, they are created using the chosen parts and materials. This is the phase which requires the most effort as in bring the robot from concept to reality. Much design modifications are expected at this point, where old designs are found wanting and new designs being necessary.

3.1.4 Testing

In this phase, the robot is tested to see whether it worked as intended. All the systems will be tested to see whether they are able to work with each other in unison. Several criteria are used to measure the effectiveness of the robot, such as its ability to detect IR and the robustness of the Wiimote - microcontroller connection.

3.1.5 System Optimization

This is the last phase of the project. After the robot has been tested to fulfil its basic requirements, further work are carried out on the robot to further enhance and improve the systems of the robot. New systems may even be considered to be put on the robot to complement the existing features. This is necessary in making the robot more efficient and robust for its expected task.

3.2 Tools and Equipment

The tools and equipment can be separated into software and hardware parts. Throughout the duration of this project, there are several systems that were used, but later abandoned as they were not working properly, and replaced by other methods. Nevertheless, these systems are still listed here for documentation purposes. The following is an explanation of all the main hardware and software tools and equipment, in no particular order of importance, used in the project.

3.2.1 Hardware

The Nintendo Wii Remote, or Wiimote, is the primary controller for the Wii game console. The Wiimote features wireless control, motion detection and gesture recognition by accelerometers, IR camera and a Bluetooth connection. Many hackers and homebrew enthusiasts had found new applications for the Wiimote outside its usage for Wii applications. The Wiimote used in this project is shown in Figure 2.

Figure 2: Wiimote.

In this project, the IR camera of the Wiimote is used to detect IR generated by open flames. The Bluetooth connects the Wiimote to the computer, where the images from the IR camera are sent to for processing.

The SK40C is a low cost PIC-based microcontroller development board made by Cytron. It is designed with basic elements for Microchip PIC users to begin project development. In this project, the SK40C serves as the initial main board for the mobile robot. The board will be connected to various subsections of the robot by means of wires. Figure 3 shows the SK40C board.

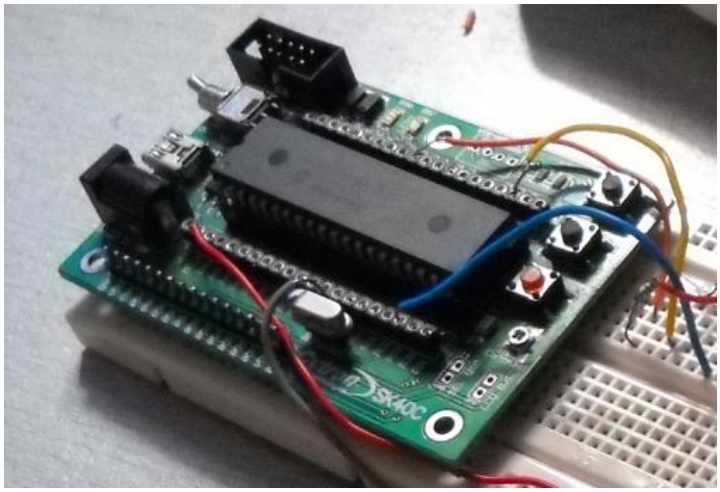


Figure 3: The SK40C board.

The Arduino Duemilanove is a development board similar to the SK40C, except it is based on the Atmel ATmega168 microcontroller. The use of the Duemilanove came later in the project, due to problems that occur during the use of the SK40C. A picture of the Arduino board used in this project is shown in Figure 4.

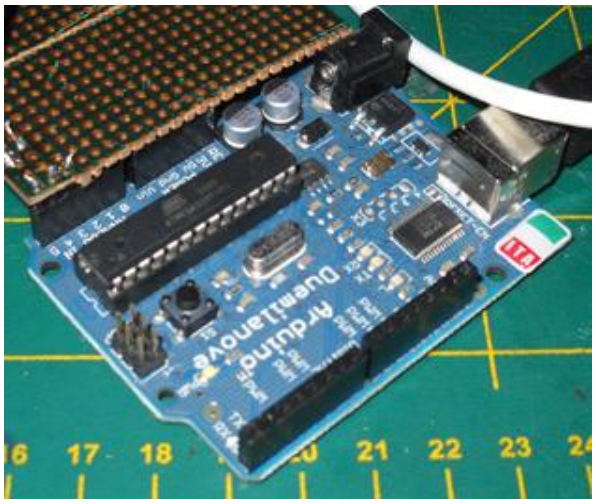


Figure 4: The Arduino Duemilanove.

K'Nex is a type of construction toys similar to Lego Technic. The toys consist of interconnecting plastic connectors and rods, that can be connected together to create different kinds of things. In this project, the K'Nex parts are used to construct the robot's main body.

Other physical tools and equipment, apart from those previously mentioned, are standard hardware used for circuit development such as breadboards, DC power supply, hand tools, soldering irons and various circuit components.

3.2.2 Software

The following list of software in Table 1 includes those that are used throughout the project as well as those that are only used partially, again for the purpose of documentation.

Table 1: Software used in this project.

No	Name	Purpose
1	MPLAB IDE	Used to write the codes for the PIC microcontroller algorithms.
2	CCS IDE	Another PIC-compatible application for writing algorithms.
3	PICKit 2	Used to program the PIC with the codes written previously by either MPLAB or CCS
4	Arduino IDE	The integrated development tools for the Arduino Duemilanove
5	Microsoft Visual Studio 2008	Used with the purpose of creating the application to display the IR dots on the PC from the Wiimote

3.3 Work Completed

The overall progress on the robot was broken down into smaller sections that measure the progress of various parts of the robot. Each of these parts was worked upon one by one. Later, the entire parts of the robot are combined together to see just how much has the project actually finished.

3.3.2 Robot base design and construction

After deliberating on the various aspects of the robot, the robot base was constructed. A suitable material is chosen as the basis of the robot. A set of tires are fastened onto the base and the base were tested to see if it would move smoothly.

3.3.2 Circuit testing and fabrication

Much thought had also been given to the line tracking system to ensure that the robot is able to patrol while not detecting any fires. Several circuit schematics on line sensors and motor drivers are studied, recreated and tested for its workability. After initial testing on the breadboard, the circuits are transferred onto the veroboard, where they are again tested to make sure they are properly functioning. Consideration is also given on the power supply used by the robot.

3.3.3 Wiimote connection

The Wiimote connection proved to be the most challenging part yet on this project. Initial attempts to establish the link between the Wiimote and the microcontroller using the SK40C were fruitless, even after changing the algorithm from using MPLAB to CCS. After the Duemilanove was put into use, other problems came up which hindered the Wiimote from being able to send and receive data from the microcontroller.

A more thorough explanation of each section, complete with results, is discussed in the following chapter.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Wiimote Test Results

The Wiimote is a main component of the robot of which the rest of the mechanisms rely on. Therefore, it is imperative that it works as planned. Initial testing of the Wiimote has yielded several results:

4.1.1 Bluetooth Connection

The Wiimote can easily connect with any computer so long a Bluetooth connection is supported on the computer. In order to connect the Wiimote to a desktop computer, a Bluetooth dongle is used. The maximum distance needed to maintain stable connection is not yet tested; however, a distance of 5 meters is considered within the acceptable limits for the connection.

4.1.2 Infrared Detection

The IR camera has several problems that hindered it from becoming a good, cheaper substitute for the normal IR camera:

- The Wiimote IR camera is originally meant to detect only a few IR points coming from the sensor bar, which is basically a bar with IR LEDS. It does not register IR as a bunch of blobs like a normal IR camera. As such, when presented with a flood of IR lights, it picks out four of the brightest IR points rather than entire blobs. An initial testing of the IR camera is done using the Multiple Wiimote Tester software [9]. The IR camera is pointed towards a constant IR source, in this case, a flame burning a paper. Multiple images of the IR dots are shown in Figure 5. Appendix A shows the full software.

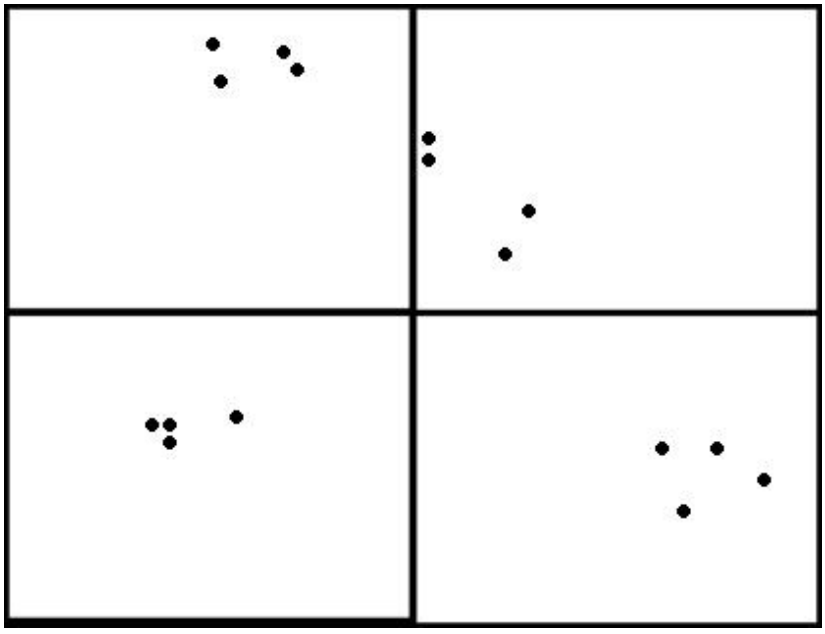


Figure 5: Four images of the IR dots

There are several random dots scattered around the four screens. These dots are the detected IR light from the IR source. The IR can only detect up to four dots at any one time. Obviously this does not make for a good IR image, as a single large blob is more convenient.

- IR can be reflected off shiny surfaces. When using the IR camera outdoors, false IR detection can come directly from the sun, or any shiny objects that reflect the sun's rays.

These issues rise from using the Wiimote in a different system not meant for its original intent. Nevertheless, there are solutions one can envision to solve these problems. The IR blob problem can be alleviated by using coding techniques that do not just take the detected IR points, but rather reconstruct them into blobs. In order to prevent interference, the robot can be limited to only be used in dark, cold places, where detection of IR guarantees fire breakout.

4.2 Robot Body

4.2.1 Body Material

In making the choice of the robot body, several factors need to be taken into account:

1. Adjustability

   Since the project is in its early stages, there are going to be lots of design changes implemented throughout the process. These changes may be temporary or stay until the final design. Therefore, it would be best if the material used to create the body can be easily and quickly changed and adjusted according to the current design of the robot.

2. Lightweight/Sturdiness

   Having a material that is both lightweight and robust is a added advantage as it allows the robot to move under minimal internal force while being tough enough to withstand normal locomotion conditions. There is no need to use a strong material as the robot is designed to only work on a smooth surface with minimal restrictions.

3. Cost

   As this project has limited budget, it is necessary that the material for the body to be as cheap as possible while still be commercially available. Most of the budget should go to other more important parts such as the Wiimote, electronic board and motor system, and as such minimal budget should be spent on the body.

Taking all factors into consideration, it is decided that the material for the body will be the construction toy system K'Nex. It is a system similar to Lego Technic and Meccano; however, it is chosen as it is deemed easier to work with as well as suited to the task at hand, although not as commercially available. The K'Nex body is intended for use as a prototype. Should this project become more serious, a better material will be used to replace it.

4.2.2 Construction

In order to see how the robot's entire body would look like in real life, an initial body was constructed using the available K'Nex parts with the Wiimote included. The early robot body is shown in Figure 6 below.

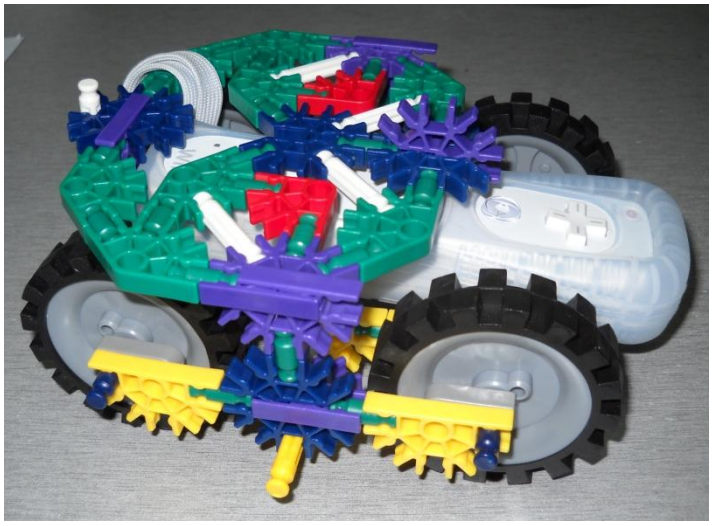


Figure 6: A side look of the first robot body

The robot body uses four wheels instead of two. A hollow compartment is made in the middle of the robot to accommodate the Wiimote. On top of the Wiimote is a flat platform intended for the microcontroller board and other circuits. There is no room for any power supply yet.

This body was found to be difficult to steer and turn around as it has four wheels. As such, the robot was redesigned. The second iteration of the robot, along with the attached line tracker sensor, is shown in Figure 7.

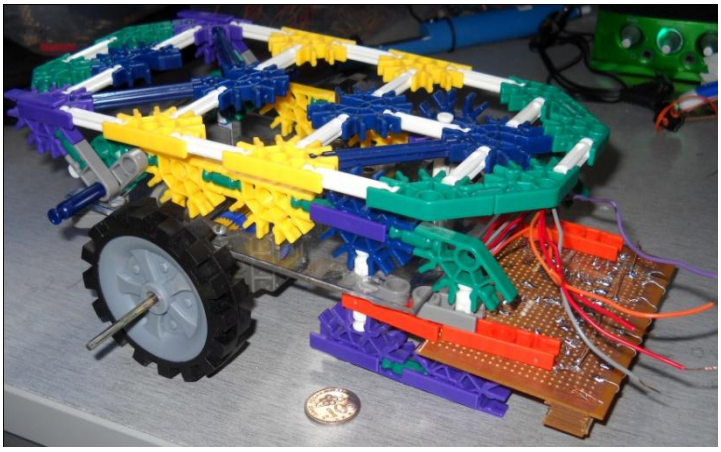


Figure 7: The second robot body with line tracker sensor attached

The K'Nex parts are built around a Perspex robot base on which the geared motors and wheels are attached. Two wheels are used, with two front casters for easy manoeuvrability. The robot line tracker sensor was put rather far away from the wheels for easier line tracking. A bigger platform is made to accommodate the circuit board, and hinges are added on the back end to allow the platform to open away and the Wiimote be placed in the middle of the robot, as show in Figure 8.

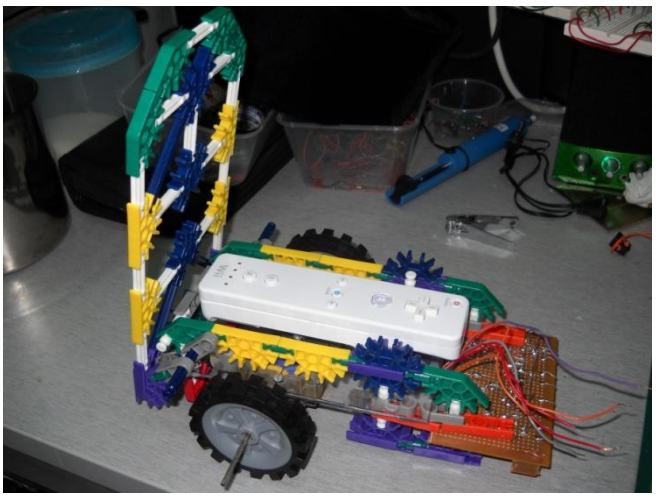


Figure 8: Placement of the Wiimote in the robot

A third iteration of the robot body was later made to accommodate the entire robotic system. The robot is made to be more compact, the power supply is placed at the back of the robot and the comparator and line tracker circuit are combined as one. A breadboard is placed on the platform as a base for the microcontroller board and the motor driver. Figure 9 showcases the new robot design along with the Arduino board. Different views of the robot are available on Appendix B and C.

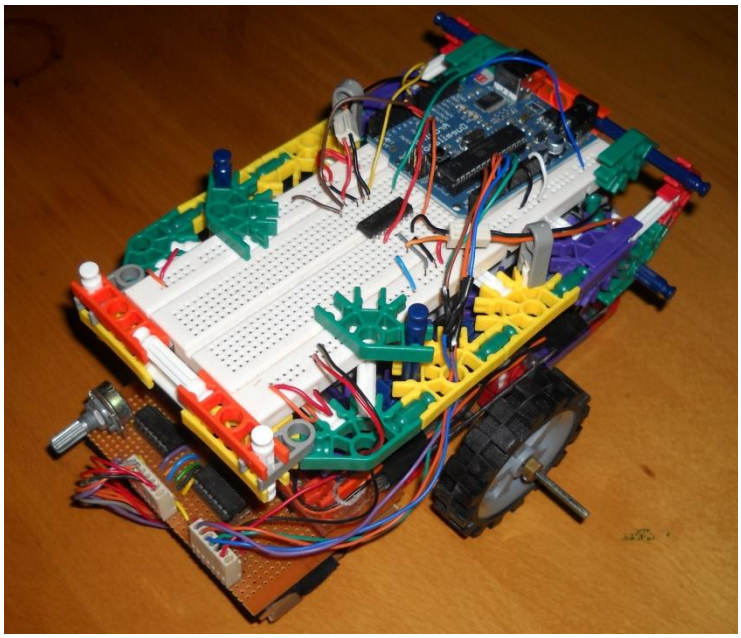


Figure 9: The final robot body with the Arduino attached.

Throughout the redesigning process of the robot body, the K'Nex parts proved its versatility and ease of usage. The robot was not only able to be disassembled and reassembled into different designs in little time, the natural shape of the individual K'Nex parts also aid in design. For example, the line tracker sensor circuit is able to be held firmly without much fastening, and the resulting boxed configuration behind the geared wheels under the robot is used to house the batteries.

4.3 Circuitry System

4.3.1 Line Tracker Sensor

Like the rest of the circuitry, the line tracker sensor is made to be as simple as possible. The sensor detects black line over a white background. There are four sensors used in the robot, and each sensor uses a combination of an LDR, a super bright LED and a comparator to detect lines. The schematic for a single sensor is shown in Figure 10 and a soldered line tracker sensor with the comparator circuit in Figure 11.

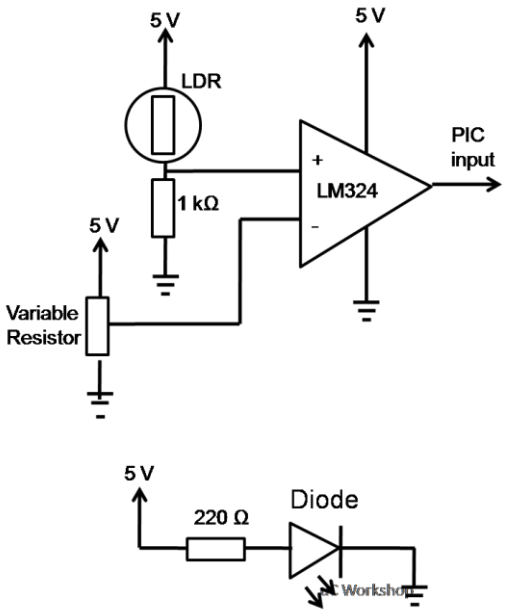
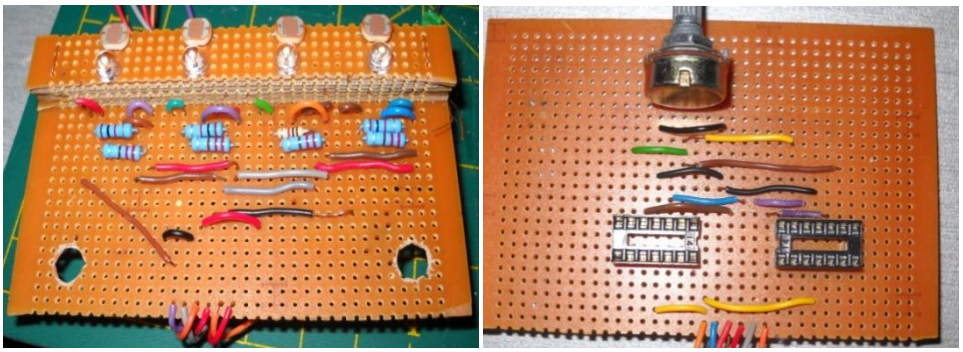


Figure 10: Circuit diagram of the line tracking sensor.



(a) (b)

Figure 11: Line tracking sensor (a) and comparator circuit (b)

4.3.2 Motor driver circuit

The motor driver takes the output from the microcontroller board and uses it to control the geared motors. The circuit uses an L298 motor driver due to its ease of use. The schematic for the motor system is shown in Figure 12.

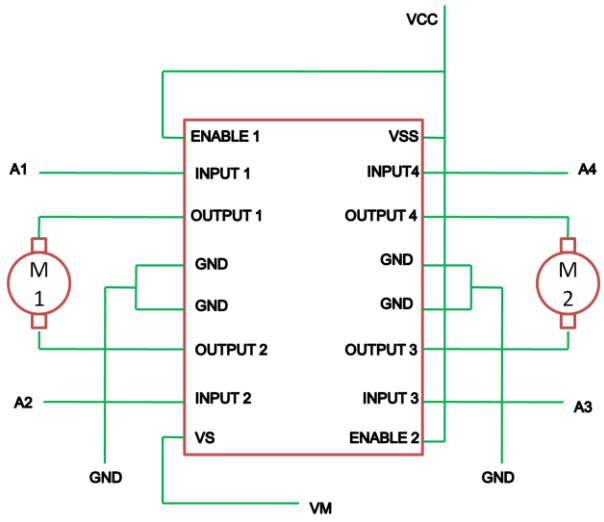


Figure 12: Schematic of the L293D motor driver system

This circuit was also soldered onto a veroboard like the line tracker sensor. However, after soldering, the circuit failed to properly work, even when all connections are checked and multiple boards are constructed. Eventually a breadboard is instead used and the circuit finally worked. The circuit is connected to the Arduino Duemilanove board along with the tracker circuit. The schematic for this whole setup is shown in Appendix D

4.3.3 Line tracking algorithm

The whole setup was first tested using a breadboard with switches and unused DC motors. An initial algorithm has been written using the MPLAB IDE software. After the motor driver was confirmed to be working, the code was expanded to include the input from the line tracker sensor. The flowchart for the code is shown in Figure 13.

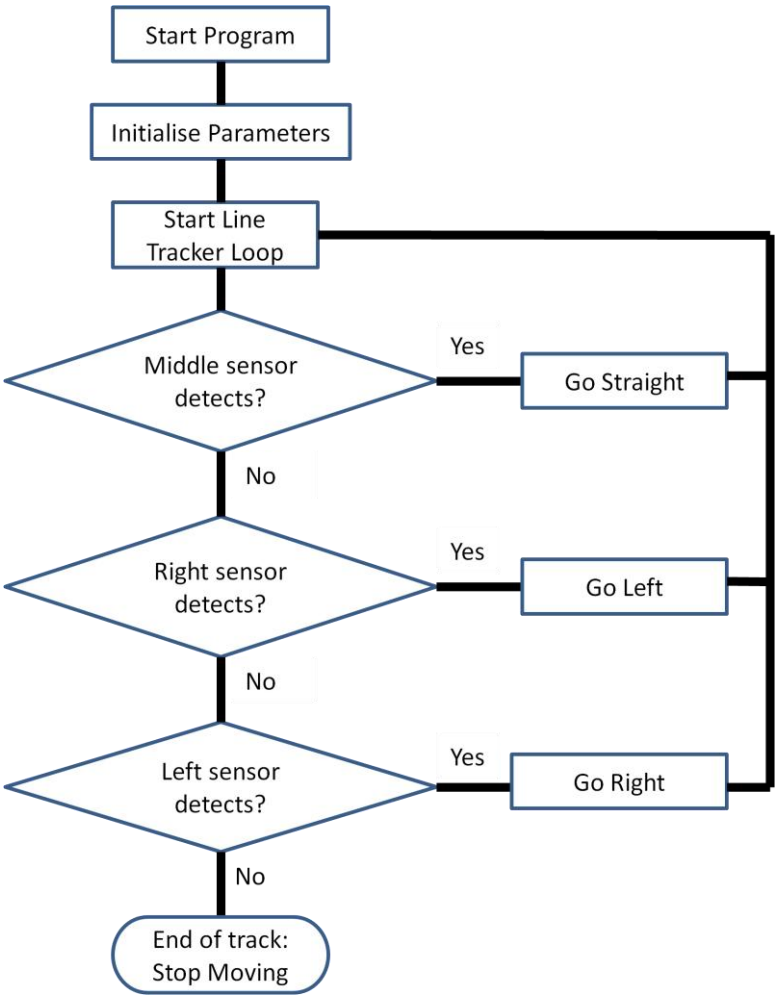


Figure 13: Flowchart of the motor driver algorithm

The entire code is later ported from the Microchip environment to the AVR microcontroller in the Arduino board. This is necessary as the final prototype will use Arduino as its control board while the SK40C is used mostly for testing. The tested line tracker code is shown in Appendix E.

4.3.4 Test Run

After the line tracker sensor, motor driver and Arduino board are ready, they are put together on the mobile robot for a test run. The trial run consists of a single white board with black tape on it to simulate a short line track course. Figure 14 shows this setup. The robot is switched on and made to run the test several times. Several issues are noted concerning the robot's performance.

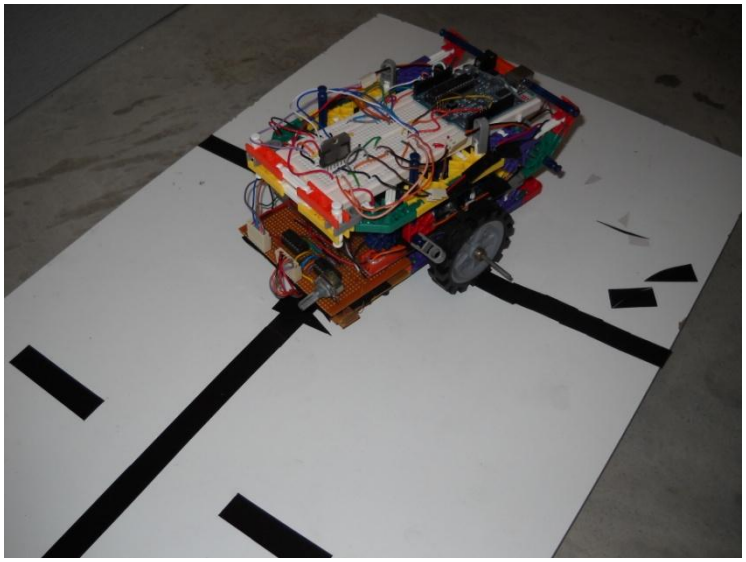


Figure 14: Testing the line tracking ability

The line tracker sensor responds to the black and white colour on the floor as expected, and the wheels both turn the correct way, corresponding to what the line tracker senses. One major problem is that the robot always veers to the right considerably when it is supposed to go straight. Although using two separate motors do result in both wheels not spinning at the same speed, the difference in speed should not be significant. On closer look, the right wheel is found to be not properly connected to the axle, resulting in a wobbly movement. After tightening the wheels to the axle, the robot moves straighter. Of course, given the line tracker's self-correcting ability, even with the wobbly wheel problem, the robot would still able to follow lines closely, albeit slower.

4.4 Wiimote I^{2}C Connection

Up until the moment this report is being made, the connection between the Wiimote and the PIC through the I^{2}C communication still cannot be established. Much effort had been put into trying to make the connection work; however, so far, the conversion of the Wiimote into a Bluetooth transceiver is yet to be achieved. Several attempts have been done which are further explained in the following paragraphs.

4.4.1 First Code using MPLAB

In the first attempt to establish the I^{2}C connection, the hardware used is the Microchip PIC 16F877A and Cytron SK40C development board, and the MPLAB IDE software with Hi-Tech C Toolsuite. The code is modified from the original code made by Michael Alon from the Microchip website [10]. The original code is chosen as it was recommended by other Microchip users. A sample of the main function is shown below in Figure 15.

```
void main()
{
ADCON1 = 0b00000011;//set RA3 as voltage reference

  i2c_init();    // init i2c

 init_nunchuk_nocode();    //initialise uncoded data
 init_nunchuk_encode();    //initialise encoded data

       while(1)
        {
                //write_six_bytes_nocode(0xa5);
               // write_six_bytes_nocode(0x52);
               write_six_bytes_encode(0xa5);
                //write_six_bytes_encode(0x52);
                DelayMs(255);
              if(read_six_bytes(0xa4))
              {
                     DelayMs(255);
              }
       }
}
```

Figure 15: The main function for the first I^{2}C code.

The initial test managed to get some data sent to the Wiimote and to the WiimoteTest program. However, all of the data are registered as IR instead of the intended Nunchuk data. This occurrence happened a few times, and then the Wiimote stopped receiving data from the PIC at all. It is assumed that the entire code might not be workable enough for the Wiimote, so after more attempts remain futile, a new I^2C code is used instead.

4.4.2 Second Code using MPLAB

The second code is taken from the samples provided by Microchip itself, as opposed from the first code which is adapted from CCS. The code is written differently from the first code. The main function is shown in Figure 16, while the data write and read functions are shown in Figure 17.

```
void main(void)
{
unsigned char count,val;

//setup ADC
ADCON1 = 0b00000011;       //set RA3 as voltage reference
TRISA = 0b00000111;        //configure PORTA I/O direction
TRISB=0;        /* use a led on RB0 - set as output */
PORTB=0;
RB0=0;

 init_nunchuk_nocode();    //initialise uncoded data
 init_nunchuk_encode();          //initialise encoded data

/* initialize i2c */
#ifdef I2C_MODULE
    SSPMode(MASTER_MODE);
    SSPEN = 1;
    CKP = 1;
#else
    SCL_DIR = I2C_OUTPUT;
    SDA_DIR = I2C_OUTPUT;
    SDA = 0;
    SCL = 0;
#endif
       while(1)
       {
              WriteByte(count,count); /* write to I2C EEPROM */
               val = ReadByte(count);   /* read back value */
              flashled3();
               DelayMs(200);                    /necessary
       }
}
```

Figure 16: The main function of the second I^2C code.

```
void WriteByte(unsigned char addr, unsigned char byte)
{
       //i2c_WriteTo(WRITE);
       if (i2c_PutByte(WRITE)==I2C_ERROR)
             flashled1();
       for (int i=0;i<6;i++)
       {                                3
              if (i2c_PutByte(outbufencode[i])==I2C_ERROR)
                    flashled1();
       }
}

int ReadByte(unsigned char addr)
{
       //i2c_WriteTo(READ);
       if (i2c_PutByte(READ)==I2C_ERROR)
             flashled2();
       for (int i=0;i<6;i++)
       {
        i2c_ReadFrom(READ);
       }
       return i2c_GetByte(I2C_LAST);
}
```

Figure 17: The write and read functions of the second code.

Unlike the first code, the second code failed to achieve any kind of data transmission with the Wiimote. When the I^2C bus is plugged into the Wiimote the WiimoteTest program registers no new events. Even when another Wiimote program is used (that uses the WiiYourself, another different wiimote emulation library), no data are shown being sent.

During the modification of both the first and second code, it is assumed that the PIC should emulate the I^2C connection as the master, and the Wiimote as the slave. However, closer inspection of the original Arduino codes reveals that the established I^2C link uses slave protocol instead of the master protocol. The code snippet is shown in Figure 18.

```
void
setup ()
{
  Wire.begin (0x52);            // join i2c bus with address 0x52
                                // this is the nunchuk address.
                                // all nunchuks use this address
  Wire.onReceive (receiveEvent);        // register event
  Wire.onRequest (requestEvent);        // register event
 }
```

Figure 18: I^2C connection initialisation of the Arduino code.

Going with this new knowledge, a new I^2C code for slave devices are searched for. However, no usable codes for I^2C slave devices are found. Therefore, the best option is to switch programming software from MPLAB to PIC C, which has plenty of master and slave example codes.

4.4.3 CCS PIC C Code

The PIC C code uses C language, simplifying the process of switching codes from MPLAB. Most importantly, PIC C has inbuilt options that provides the required I^2C specifications for the codes.

For the I^2C connection with Wiimote, the created code is made for a slave device. A sample project taken from the PIC C forums show a Nunchuk connected to a PIC with the PIC as the master. Technically, this would make the Nunchuk a slave device, and as such the PIC in this project should be made a slave device as well. The resulting slave codes are shown, with the main function in Figure 19 and the I^2C subroutine in Figure 20.

Currently, like its previous MPLAB counterparts, the code still fails to send data from the PIC to the Wiimote. The code is being continuously improved upon, and hopefully is able to connect the PIC to the Wiimote.

```
void main ()         //main program
{

   encrypt_init_encode();  //generate encoded data
   encrypt_init_nocode();  //generate simple data

   //series of on-off LEDS to signify program initialised and
working
       output_high(PIN_B7);
      output_low(PIN_B6);
      output_high(PIN_B5);
      output_low(PIN_B4);
      delay_ms(500);
      output_low(PIN_B7);
      output_high(PIN_B6);
      output_low(PIN_B5);
      output_high(PIN_B4);
      delay_ms(500);
      output_low(PIN_B7);
      output_low(PIN_B6);
      output_low(PIN_B5);
      output_low(PIN_B4);

   enable_interrupts(GLOBAL);
   enable_interrupts(INT_SSP);

   while (TRUE) {}
}
```

Figure 19: Main function of the PIC C code.

```
#INT_SSP
void ssp_interupt ()
{
   BYTE incoming, state;
   state = i2c_isr_state();

   if(state <= 0x80)              //Master is sending data
   {
      incoming = i2c_read();
      if(state == 1)      //First received byte is address
      {
         flashled1();
         address = incoming;
      }
      if(state == 2)            //Second received byte is data
      {
         buffer[address] = incoming;
         flashled2();
      }
   }
   if(state > 0x80)            //Master is requesting data
   {
      index = state & 7;  // Lower 3 bits of state = the index
      i2c_write(bufencode[index]);
//!      i2c_write(buffer[address]);
//!      i2c_write(bufencode[0]);
//!      i2c_write(bufencode[1]);
//!      i2c_write(bufencode[2]);
//!      i2c_write(bufencode[3]);
//!      i2c_write(bufencode[4]);
//!      i2c_write(bufencode[5]);
      flashled3();
   }
   //flashled4();
```

Figure 20: The PIC C I^2C subroutine code.

4.4.4 Original Arduino Code

Looking at the failures of the previous getups that use PIC microcontrollers and MPLAB and PIC C compilers, a contingency plan is drafted in which the project uses the Arduino, just like what the original project used [11]. The latest Arduino Duemilanove board with microcontroller Atmel ATMega 328 are used to replicate this project. The code for the Arduino is shown in Appendix F.

Even when using a direct code of the original program, the I^2C connection is still not established. No changes are made to the original code, and yet it still doesn't work. Figure 9 shows the Arduino setup.

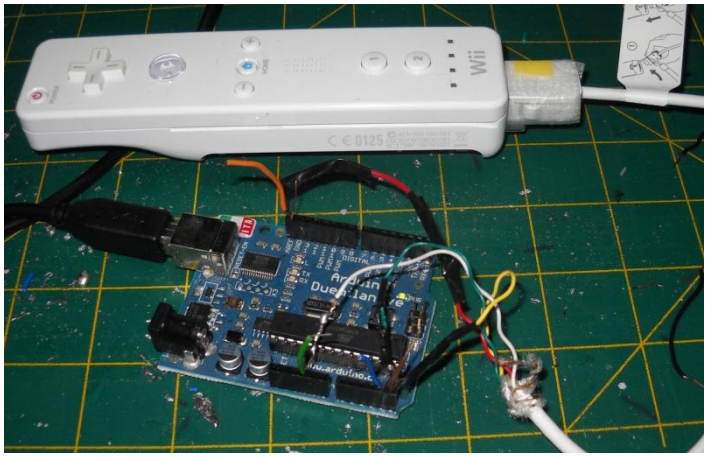


Figure 21: The Arduino setup.

The setup had been tested with both the WiimoteTest and the WiiYourself demo program with negative results. From Figure 21, it is shown that the four coloured input wires from the Wiimote are connected to its respective places. However, the connections are actually loose, since the square holes are bigger than the single-core wires. In order to have a tight fit, another short wire is inserted into each hole.

There is a possibility that such a configuration interfered with the programming; however, previous testing of the setup using just the four I^2C wires (in loose conditions) also yield the same results.

4.4.5 Wiimote Extension Library

Further research on the Internet yields an alternative method to the Arduino code; the Wiimote Extension Library (WEP) [12]. The WEP is a C library that gives an AVR microcontroller the ability to act as a Wiimote extension controller like the Nunchuk and the Guitar Hero controller. It addresses the problem with encryption by giving the microcontroller the encrypted handshaking and unique ID based on the actual extensions.

Problems arise when trying to program the Arduino using the code. The code itself is not the problem, but somehow the Arduino cannot be programmed with the code. While the WEP can use the Arduino due to its AVR microcontroller, programming the WEP into the AVR uses text-based software called AVRDUDE as C programming is incompatible with the Arduino IDE (which uses C++). The AVRDUDE lacks any GUI, and instead uses command line interfaces to access the various AVR microcontrollers.

Numerous attempts were done to try using the AVRDUDE to program the Arduino with no positive results. Several different commands that are possibly compatible with the Arduino are tried using the AVRDUDE to no avail. Some of these commands written into the command prompt are shown in Figure 22. The resulting error message from all these commands is shown in Figure 23.

The reason for this problem is still unclear. Searches at the Arduino and AVRDUDE forum yield several possible solutions, several which were tried but were not successful. Even when using a different OS, from Windows 7 to Windows XP, the problem still persists.

```
C:\WinAVR-20100110\Test Program avrdude -p m328p -c arduino -P com3 -U
flash:w:count.hex -v -v -v -v
```

```
C:\WinAVR-20100110\Test Program avrdude -p m328p -c stk500 -P com3 -U
flash:w:count.hex -v -v -v -v
```

```
C:\WinAVR-20100110\Test Program avrdude -p m328p -c stk500v1 -P com3 -U
flash:w:count.hex -v -v -v -v
```

```
C:\WinAVR-20100110\Test Program avrdude -p m328p -c stk500v2 -P com3 -U
flash:w:count.hex -v -v -v -v
```

Figure 22: The slightly different commands to program the Arduino using AVRDUDE.

```
avrdude: stk500_getsync(): not in sync: resp=0x00
avrdude done. Thank you
```

Figure 23: The resulting error message from previous commands.

4.4.6 Direct IR Camera Access

After many previous failures to turn the microcontroller into an emulated extension of the Wiimote, a final attempt is to access the IR camera directly. Rather than having the computer access the camera through Bluetooth and commands sent to the microcontroller, the microcontroller directly accesses the IR camera through the I^2C connection. This method is a bypass, as the microcontroller is not communicating with the Wiimote, but only its IR camera, as explained by Brian Dwyer[13] and Stephen Hobley[14]. The whole setup is as follows:

- Open the Wiimote cover and solder a wire to the port 7 of the Pixart IR camera.
- Connect the wire to a 24MHz crystal oscillator clock.
- Connect the red Vcc wire from the I^2C cable to a 3.3 V power supply and the white ground wire to the ground. Both connections must come from the Arduino board in use.

- Connect the SDA and SCL wires from the I^2C cable to the corresponding SDA and SCL pins on the Arduino

The Arduino is later programmed using the code provided by Bryan Dwyer, slightly modified to show any responses to the detection of the IR lights. The body of the code is shown in Figure 24. Appendix G shows the full code.

```
void loop()
{
  result = ircam.read();

  if (result & BLOB1)
  {

if (ircam.Blob1.X > 480 && ircam.Blob1.X < 520)
{
  myservo.write(STOP);
}

  else if (ircam.Blob1.X < 480)
{
    if (ircam.Blob1.X < 200)
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
   else if (ircam.Blob1.X < 400)
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
    else
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
}

else if (ircam.Blob1.X > 520)
{
    if (ircam.Blob1.X > 800)
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
    else if (ircam.Blob1.X > 600)
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
    else
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
}
  }
   else
  {
  digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
  delay(100);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(100);
  }
}
```

Figure 24: The main code for the direct IR camera access.

Even when using this setup, the Arduino failed to detect any IR lights from the Wiimote. The previous code will light up an LED if any IR is detected by the camera. As such, the LED did not light up even when there a flame is waved very close to the camera. However, when the code is changed so that the LED lights up when there are no IR points detected, the LED does light up – indicating that the code does work. The only problem is that the code somehow does not get the IR from the camera, even if the IR values are actually available. Several measures are tried out, with no success:

- Changing the value of the crystal oscillator to 20 MHz and 25 MHz.
- Using batteries instead of the 3.3 V to power the Wiimote.
- Using a separate code to determine if there is any I^2C data transfer going on
- Using 5 V instead of 3.3 V to power the Wiimote. This is not recommended as it can damage the circuit.

At this point, there is not much to be done as even the final attempt for the connection between the Wiimote and the microcontroller failed to produce any positive results. Searches on the internet yield some insight on why all the attempts so far have failed. The encryption and unique ID identified with each Wiimote peripheral renders the previous attempts using the Microchip boards futile. The original Arduino code failed probably due to the different versions of Wiimote used then and now. This is a crucial factor, as many people have posted their own results with the Wiimote online and some of them varied significantly from each other. A code that works with one Wiimote does not work with another Wiimote. Of course, there are also other possible reasons like improper implementation of the previous connection attempts.

4.4.7 Bluetooth Direct Connection

The last option available is to forgo the Wiimote as the Bluetooth transceiver and instead use a specialised Bluetooth transceiver to send data from the computer directly to the microcontroller. This means that the robot now has two Bluetooth connections, with the first Bluetooth used by the Wiimote to send data from its IR camera to the PC. This new system is illustrated by Figure 25.

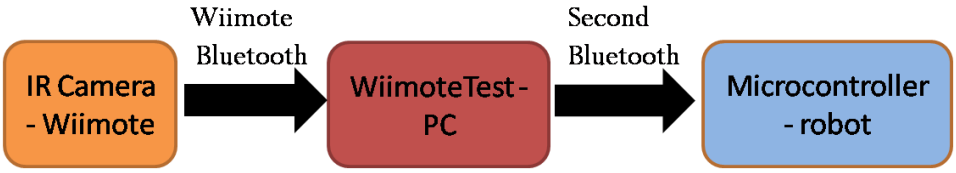


Figure 25: The new Bluetooth system.

A module, the KC Wirefree Bluetooth Transceiver SKKCA, is already acquired for this purpose. Figure 26 shows this Bluetooth module.

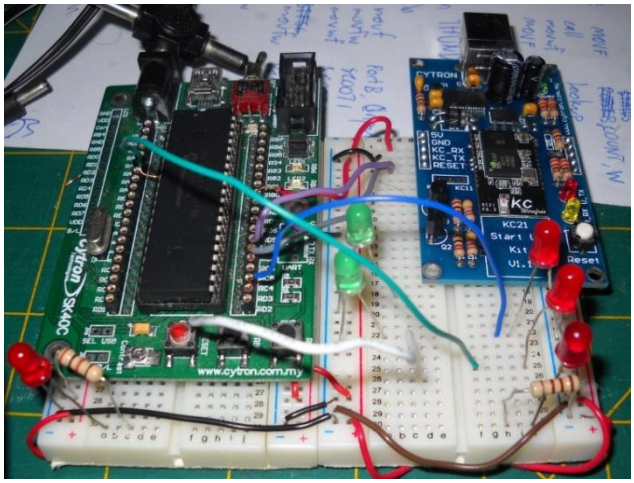


Figure 26: the KC Bluetooth module SKKCA, on the right.

With the new Bluetooth in place, the full Wiimote-PC-microcontroller link is finally established. This allows for the final robot design to be realised, as shown in the next section.

4.5 Final Working Design

With the various aspects of the robot finally put in order, the actual robot is finally able to be built. A software is created that can bring together the image from the IR camera and the commands needed to control the robot. Called the Wiimote Infrared Program (WIRP), it acts as the graphical user interface (GUI) for the entire project, displaying the IR images and their coordinates, and also the current mode of the robot. The WIRP software is shown in Appendix H.

The software is created using the Microsoft DirectX API. DirectX is a library specifically made for multimedia and graphics applications, and is used to show the IR images for the application. The WIRP use the application taken from the Codesampler website [15], as well as several other references [16][17]. The WIRP is divided into several sections that deal separately with the Wiimote, the serial Bluetooth port and the IR display and robot control.

4.5.1 Wiimote Connection

The WIRP is able to connect to the Wiimote by including the Wiimotelib library from the previous Multiple Wiimote Tester application as part of the project. The Wiimote is connected at the start of the program and disconnected when the program exits. The codes for the function for the processes are shown in Figure 27.

```
private void InitWiimote()
{
        Wyrm.Connect();
        Wyrm.SetReportType(InputReport.IRExtensionAccel, true);
        Wyrm.SetLEDs(false, false, true, false);
}

private void ExitWiimote()
{
        Wyrm.SetLEDs(false, false, false, false);
        Wyrm.SetRumble(false);
        Wyrm.Disconnect();
}
```

Figure 27: The functions to connect and disconnect the Wiimote.

4.5.2 Bluetooth Serial Port Connection

The computer uses the HyperTerminal, a built-in communications software for Windows to connect to the second Bluetooth. However, the HyperTerminal is a standalone program and cannot be integrated into the main WIRP application. Therefore, connection using the WIRP is done by calling the SerialPort class under the System.IO.Ports library. The class contains functions for opening, reading and writing to serial ports. Like the Wiimote function, the connection is divided into functions that define the port, open the port, write to and close the port. The port functions are shown in Figure 28.

```
SerialPort port = new SerialPort("COM9", 115200, Parity.None,
8, StopBits.One);

        private bool SerialOpen()
        {
            port.Open(); //
            return true;
        }
        private void SerialWrite()
        {
            if (input == 1)
            {
                port.Write("1"); //
            }
            else if (input == 2)
            {
                port.Write("2"); //
            }
            else if (input == 3)
            {
                port.Write("3"); //
            }
            else if (input == 4)
            {
                port.Write("4"); //
            }
            else port.Write("5"); //
        }
        private bool SerialClose()
        {
            port.Close();
            return true;
        }
```

Figure 28: The functions for the serial Bluetooth connection.

4.5.3 IR Display and Robot Control

The IR section of this code consists of the IR display and robot control functions. The IR display deals with showing the present IR values on the WIRP. Up to four values of the IR can be shown on the screen at any one time. The IR display is further divided into two parts: calculating the x and y coordinates of the values with respect to the IR camera's 768 x 1024 resolution, and drawing the squares on the screen to show the locations of the IR points. The calculation and drawing functions are shown in Figure 29 below.

```
IR1X = (Wyrm.WiimoteState.IRState.IRSensors[0].Position.X *
FormWidth) * 3 / 4;

IR1Y = (FormHeight -
(Wyrm.WiimoteState.IRState.IRSensors[0].Position.Y * 5 / 7) *
FormHeight) * 3 / 4;

IR2X = (Wyrm.WiimoteState.IRState.IRSensors[1].Position.X) *
FormWidth * 3 / 4;

IR2Y = (FormHeight -
(Wyrm.WiimoteState.IRState.IRSensors[1].Position.Y * 5 / 7) *
FormHeight) * 3 / 4;

IR3X = (Wyrm.WiimoteState.IRState.IRSensors[2].Position.X *
FormWidth) * 3 / 4;

IR3Y = (FormHeight -
(Wyrm.WiimoteState.IRState.IRSensors[2].Position.Y * 5 / 7) *
FormHeight) * 3 / 4;

IR4X = (Wyrm.WiimoteState.IRState.IRSensors[3].Position.X *
FormWidth) * 3 / 4;

IR4Y = (FormHeight -
(Wyrm.WiimoteState.IRState.IRSensors[3].Position.Y * 5 / 7) *
FormHeight) * 3 / 4;

            WiimoteDrawIR1(IR1X, IR1Y,0);
            WiimoteDrawIR2(IR2X, IR2Y,1);
            WiimoteDrawIR3(IR3X, IR3Y,2);
            WiimoteDrawIR4(IR4X, IR4Y,3);
```

Figure 29: The IR calculation and drawing functions.

The IR robot control gets the position of the IR point and sends commands to the robot as an output. This function is shown in Figure 30.

```
private void GetDirections()
{
if (manualcheck != 0)
input = manualcheck;
else
{
if (IR1X >= 599) //if IR is zero
{
input = 5;//go linetracker mode
}

else if ((IR1X >= 0) && (IR1X < 200)) //if IR less than 200
{
input = 2; //go left
}

else if ((IR1X >= 200) && (IR1X < 400)) //if less than 400
{
input = 1; //go straight
}

else if ((IR1X >= 400) && (IR1X < 599)) //less than 600
{
input = 3;// go right
}
else input = 5;
}
}
```

Figure 30: The IR robot control function.

The code works by dividing the IR camera screen into three regions: left, middle and right. For example, if a point is available on the left region, the code sends a “3” integer to the robot so the robot will move to the right to keep tracking the IR. The choice of the “3” integer is due to simplicity. Other characters or strings are also possible, with “3” chosen to reduce complexity and maintain a sense of numerical differentiation.

```
protected override void
OnKeyDown(System.Windows.Forms.KeyEventArgs e)
{
if(e.KeyCode == Keys.Space)
      {
       manualcheck = 0;//disables manual control
       arrow = "Automatic";
      }
     if (e.KeyCode == Keys.W)
      {
       manualcheck = 1;//goes straight
       arrow = "Straight";
      }
     if (e.KeyCode == Keys.S)
      {
       manualcheck = 4;//reverse
       arrow = "Stop";
      }
       if (e.KeyCode == Keys.A)
       {
       manualcheck = 2;// go left
       arrow = "Left";
       }
       if (e.KeyCode == Keys.D)
       {
       manualcheck = 3;//go right
       arrow = "Right";
       }
       if (e.KeyCode == Keys.Escape)
       {
        this.Dispose();
       }
}
```

Figure 31: The manual robot control function.

To control the robot manually, the user only needs to press the corresponding keys. For example, to make the robot go straight, the key ”W” is pressed. The robot can also be reverted to line tracker mode by pressing the spacebar. The current mode is also displayed on the WIRP, as determined by the strings “Automatic”, “Straight”, “Stop” and so on. This is shown in Figure 31.

The rest of the WIRP consists of functions necessary to setup the application. The full code of the entire WIRP application can be seen in Appendix I.

4.5.4 Final Robot Body

The final robot body is virtually the same with the third body design, with several differences. The SK40C is used instead of the Duemilanove due to its compatibility with the SKKCA. A new line tracker circuit that utilises five sensors is used. The SKKCA Bluetooth transceiver is put at the top front part of the robot. Five red LEDs are used to show what the line tracker circuit senses. Appendix J shows these changes in the robot body.

4.5.5 Microcontroller code

The 16F877A code used for the final design is similar to the previous Arduino line tracker code; apart from the addition of the serial data receive function. The function is shown in Figure 32.

```
void main(void)
{
       TRISB = 0b00011111; //port b output, input
       //TRISD = 0x00; //port d output
       init();
       level = 5;
       while(1)
       {a = receive();
              if (a == '1') // signal 1, go straight
                     {level = 1;
                     }
              else if (a == '2')  // signal 2, go lef
                     {level = 2;
                     }
              else if (a == '3')  // signal 3, go right
                     {level = 3;
                     }
              else if (a == '4')  //signal 4, stop
                     {level = 4;
                     }
              else if (a == '5')  //signal 5, stop
                     {level = 5;
                     }
              else if (a == '6')  //signal 5, stop
                     {level = 6;
                     }
              else   level = 5;
              serialcheck(level);
       }
}
```

Figure 32: The serial data receive function.

The function is similar to the one on the WIRP side. The code accepts the data and determines what the value is, and sets the variable ”level”, which is the mode for the robot. The function that checks the mode is shown below in Figure 33.

```
void serialcheck(int check)
{

              if (check == 1)     // signal 1, go straight
                     {
                            straight();
                     }
              else if (check == 2) // signal 2, go lef
                     {
                            left();
                     }
              else if (check == 3) // signal 3, go right
                     {
                            right();
                     }
              else if (check == 4) //signal 4, stop
                     {
                            stop();
                     }
              else if (check == 5) //signal 5, linetracker mode

                     {
                            linetrack(check);
                     }
              else if (check == 6) //signal 5, linetracker mod
                     {
                            reverse();
                     }
              else linetrack(check);
}
```

Figure 33: The mode checking function.

The serialcheck function determines what is the action of the robot based on the variable “level” provided. From the codes, the line tracker mode is called if the “level” variable is 5. Any other value and the robot goes on a manual control. The full microcontroller code is provided in Appendix K.

4.5.6 Final Test Run

Now that the robot is finally finished, it is ready for its final test run. The test run consists of putting the robot on a line track, letting it have its run and testing its fire detection ability.

The line track is made of a large aluminium sheet. The aluminium sheet is covered with white A4 papers where the black tape is present. Technically the reflective surface of the sheet will act as a good substitute to the A4 papers; however, this is not the case. Nevertheless, the track is good enough with the current setup. For the test run, a power supply cord is used to connect to the robot as rechargeable batteries are too heavy and does not last long. The cord is held high at all times to ensure that it does not become tangled up with the robot.

When the robot is turned on, the wheels started moving. However, it was not in a line tracker mode as testing the sensors does not yield expected results. This did not matter, as when the WIRP application is turned on, the robot went into line tracker mode. Results of the initial testing of the manual mode and fire detection mode made while the robot was immobile (wheels not touching the ground) were positive; the robot responded as expected. The robot also responded to IR coming from the window; this is also expected.

Putting the robot on the actual track yielded slightly different results. The robot followed the line, but often went off, especially when going round the bend. This was due to the robot's high speed and less calibrated line tracker sensor. After letting the motors run at 5 V and adjusting the sensitivity of the sensor, the robot performed a lot better. The robot can still run off course, but this occurrence was significantly reduced.

Fire detection is done using a candle. Due to the candle flame's flickering nature, its IR presence is not consistent and the robot might miss it. As such, two candles are used to maximise the IR source. The candles are cut in half to

cater for the IR camera's low position. The candles are put approximately 1 meter from the track, and latter tests involved putting the candles even further.

The robot again performed as expected. When running the track, the robot will break off and move towards the fire. Previously, when the robot was moving with high speed, the robot will swerve violently from left to right when moving towards the fire in an attempt to keep the IR in the middle of the camera. Often the robot ends up missing the IR and proceeds to move somewhere else. With the speed fixated to 5 V, the robot became more manageable and approached the fire source smoothly. Even when the candle is around 2-3 meters from the track, the robot still detects the IR source.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The hazard inspection robot is essential in high-risk maintenance jobs which involve dangerous working conditions and materials hazardous to human health. The average hazard inspection robot or equivalent costs a substantial amount of money, resources and technology sophistication. As such, the scope of this project is limited to using cheap and available materials as a proof of concept that such devices can be done in a more economical way.

The project managed to achieve its intended objective to produce a fully working prototype of a fire inspection mobile robot, although not without running into a few serious problems. Most notably is the failure to conceive the original idea of hacking the Wiimote to recognise the microcontroller as its extension controller. Although the current setup is able to do the job just as nicely, the fact remains that the robot would be a lot smaller than it is now, due to lack of the transceiver circuit.

The robot has been fully tested and was found to be performing its required tasks as expected. Although the robot will not work perfectly 100% of the time, nevertheless the results show that the current system definitely works, and any future work regarding the project may resume from here.

5.2 Recommendations

There is still much that can be done for this project should it be explored further in the future. For a start, the previous problems mentioned can be explored further to see what exactly is causing the Wiimote to not able to connect to the microcontroller. This study can be limited to just the AVR or Microchip microcontrollers and its development boards, or both if one choose to have a greater scope. There is a likely possibility that inappropriate use of hardware, as well as bad coding (as in the case with the AVRDUDE) are responsible for the whole rig to not work.

Other aspects of the robot can also be improved upon. The current patrolling capability of the robot can be further expanded from just using line tracking sensors to, for example, tracking underground cables, riding on rails or even utilising an odometer to track the distance travelled for navigation. Rather than relying on just the infrared camera, a temperature sensor can also be installed to ensure that the detected IR corresponds to an actual flame. A small fire extinguishing system can also be implemented to put out small fires.

Using a new material for the robot body is also a good consideration. The scope of this project requires it to be cheap and light, hence the use of K'Nex. However, K'Nex, being made of plastic, aren't good against extreme heat, and as such another more heat resistant material is preferable. If the next project is not limited by the budget, perhaps material like aluminium would be ideal.

REFERENCE

[1] Yamamoto, S., “Development of Inspection Robot for Nuclear Power Plant”, Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference, 12-14 May 1992, Pages: 1559 - 1566.

[2] Kawauchi, N.; Shiotani, S.; Kanazawa, H.; Sasaki, T.; Tsuji, H.; “Plant Maintenance Humanoid Robot System”, Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on 14-19 Sept. 2003, Pages: 2973 - 2978 vol. 3.

[3] Nassiraei, A.A.F.; Kawamura, Y.; Ahrary, A.; Mikuriya, Y.; Ishii, K.; “Concept and Design of A Fully Autonomous Sewer Pipe Inspection Mobile Robot “KANTARO””, Robotics and Automation, 2007 IEEE International Conference on 10-14 April 2007, Collaboration Center, Pages: 136 - 143.

[4] Yukawa, T.; Suzuki, M.; Satoh, Y.; Okano, H.; “Design of Magnetic Wheels in Pipe Inspection Robot”, Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on 8-11 Oct. 2006, Pages: 235 - 240.

[5] Briones, L.; Bustamante, P.; Serna, M.A.; “Wall-Climbing Robot for Inspection in Nuclear Power Plants”, Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on 8-13 May 1994, Pages: 1409 - 1414 vol. 2.

[6] Kobayashi, F.; Usami, T.; Kojima, F.; Nakatsuka, H.; “Crack Shape Recovery with ECT Sensor Robot for Remote Diagnosis System”, SICE-ICASE, 2006. International Joint Conference on 18-21 Oct. 2006, Pages: 964 - 969.

[7] Murphy, R.; Kravitz, J.; Stover, S.; Shoureshi, R.; "Mobile Robots in Mine Rescue and Recovery", Robotics & Automation Magazine, 2009, Volume: 16, Issue: 2, Page(s): 91 – 103.

[8] McCarty, K.; Manic, M.; "Adaptive Behavioural Control of Collaborative Robots in Hazardous Environments", Human System Interactions, 2009. HSI '09. 2nd Conference on 21-23 May 2009, Pages: 10 – 15.

[9] WiimoteLib, .NET Managed Library for the Nintendo Wii Remote, <http://www.brianpeek.com/blog/pages/wiimotelib.aspx>, 7 June 2008, [Accessed 3 March 2010].

[10] Michael Alon, C sample code for PIC micros and Hi-Tech C, http://www.microchipc.com/sourcecode/#i2c>, [Accessed 18 June 2010].

[11] Chad Phillips, How to make your own Wiimote peripheral, <http://www.windmeadow.com/node/37>, July 11 2006-07 [Accessed 1 January 2010].

[12] Frank Zhao, Wiimote Extension Library, Circle of Current, <http://frank.circleofcurrent.com/cache/wii_extension_lib.htm>,March 15 2009 [Accessed 2 September 2010].

[13] Brian Dwyer, Wiimote light follower with servo, <http://ohmwardbond.blogspot.com/2010/03/wiimote-light-follower-with-servo.html>,March 17, 2010 [Accessed 20 October 2010].

[14] Stephen Hobley, Pixart/Wiimote sensor library for Arduino, <http://www.stephenhobley.com/blog/2009/03/01/pixartwiimote-sensor-library-for-arduino>, March 1, 2009[Accessed 17 September 2010].

[15] Kevin R. Harris, Direct3D (DirectX 9.0) Code Samples, < http://www.codesampler.com/dx9src.htm>, June 15, 2005 [Accessed October 1, 2010].

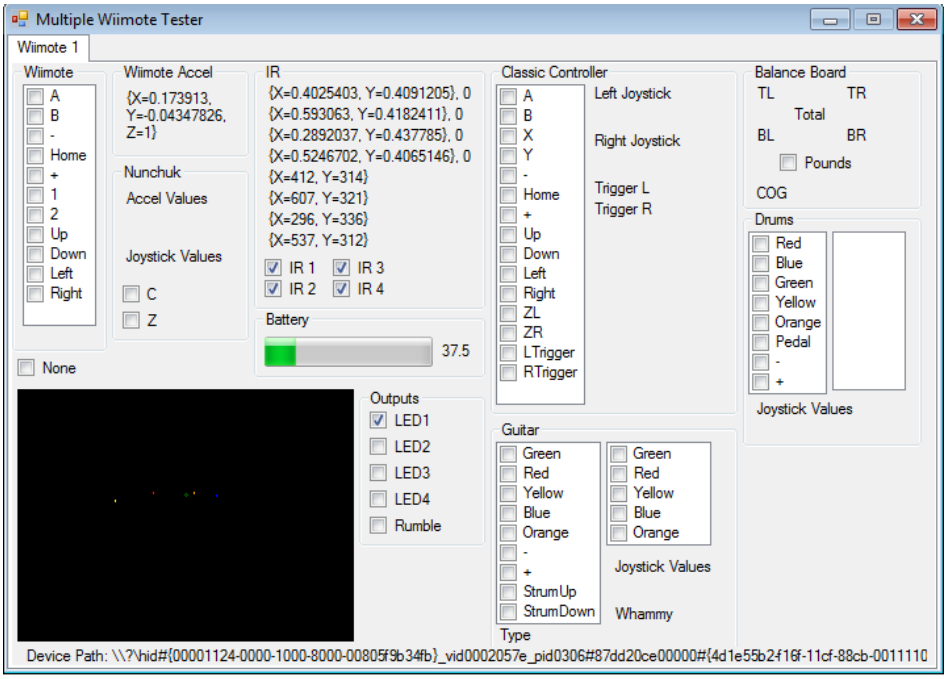
[16] Riemer Grootjans, Displaying text using DirectX, Riemer's XNA Tutorials, http://www.riemers.net/eng/Tutorials/DirectX/Csharp/Series2/tut18.php >, February 2008 [Accessed October 2, 2010].

[17] Craig Andera, Texture Basics Tutorial, Craig Andera's DirectX Wiki, http://alt.pluralsight.com/wiki/default.aspx/Craig.DirectX/TextureBasicsTutorial.html>, April 2005 [Accessed October 2, 2010].

APPENDICES

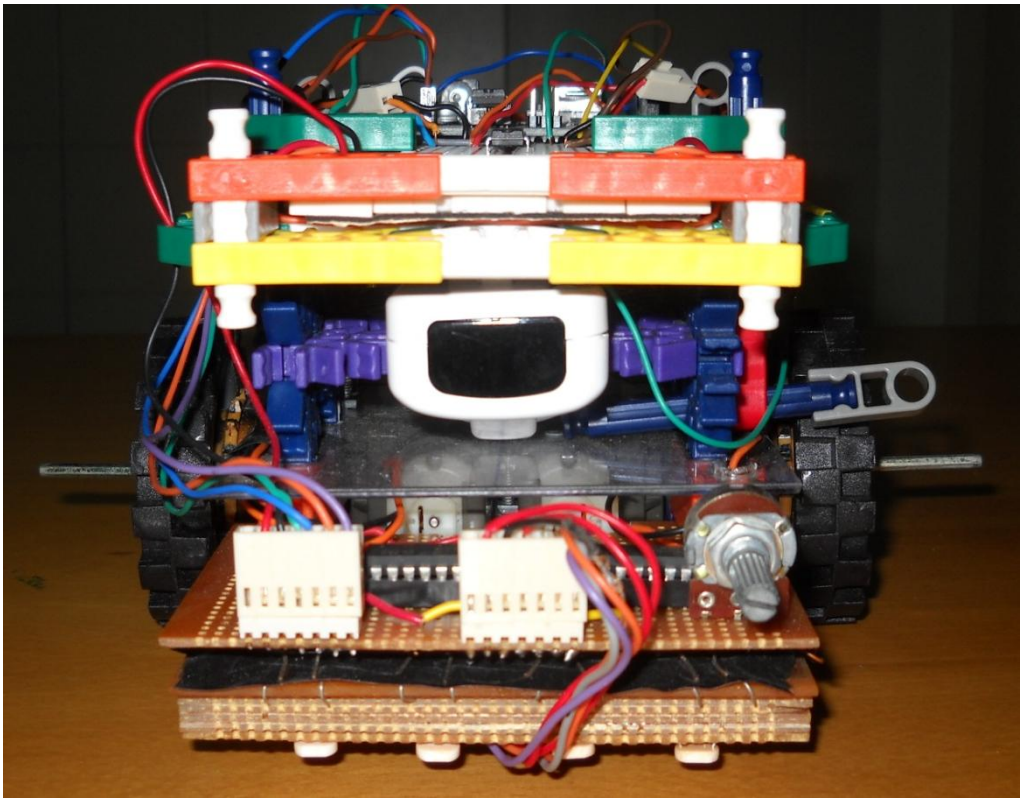
APPENDIX A

THE MULTIPLE WIIMOTE TESTER APPLICATION



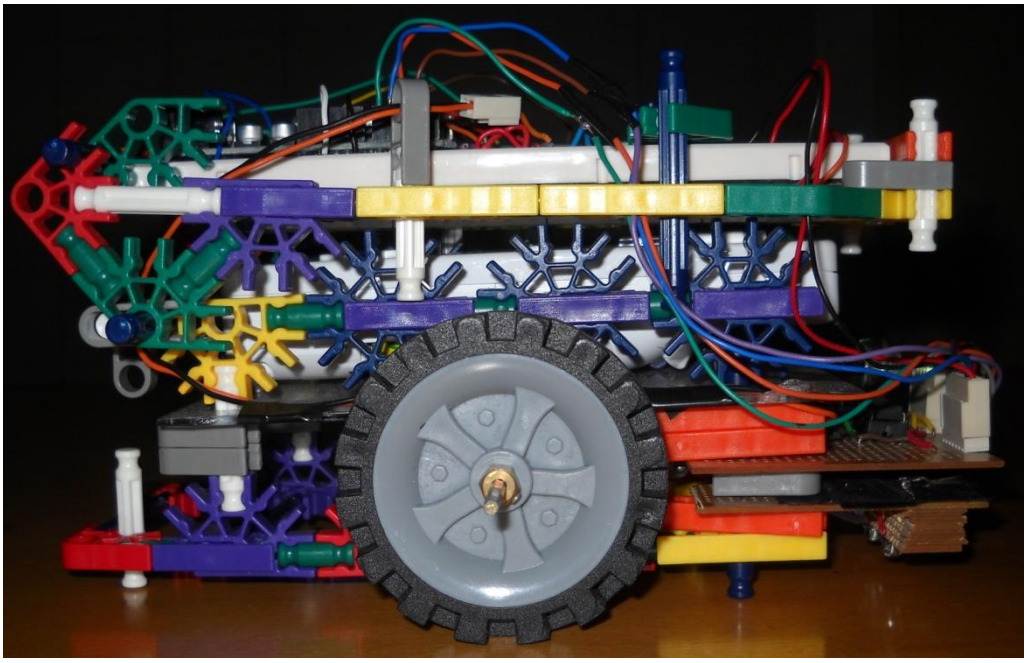
APPENDIX B

THE FRONT VIEW OF THE FINAL ROBOT BODY



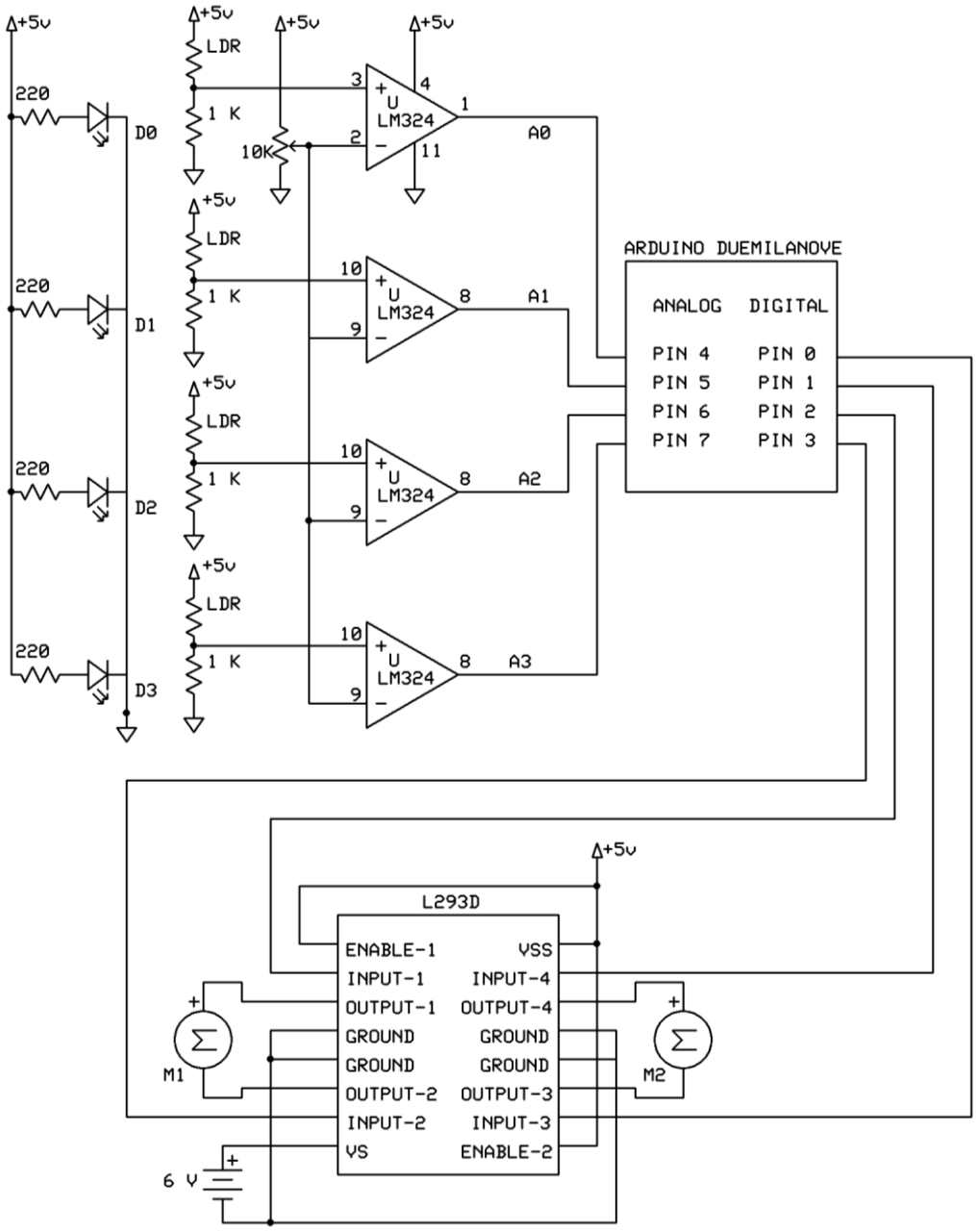
APPENDIX C

THE SIDE VIEW OF THE FINAL ROBOT BODY



APPENDIX D

THE SCHEMATIC FOR THE LINE TRACKER CIRCUIT



APPENDIX E

THE CODE FOR THE LINE TRACKER

```
int Input0 = A0;
int Input1 = A1;
int Input2 = A2;
int Input3 = A3;

int Output0 = 4;
int Output1 = 5;
int Output2 = 6;
int Output3 = 7;

void setup() {
  // initialize the digital pin as an output:
  pinMode(13, OUTPUT);
  //Inputs
pinMode(Output0, OUTPUT);
pinMode(Output1, OUTPUT);
pinMode(Output2, OUTPUT);
pinMode(Output3, OUTPUT);

pinMode(Input0, INPUT);
pinMode(Input1, INPUT);
pinMode(Input2, INPUT);
pinMode(Input3, INPUT);

digitalWrite(13, HIGH);
}

void loop()
{
   if((digitalRead(Input0)==1) && (digitalRead(Input1)==1)  &&
(digitalRead(Input2)==1)  && (digitalRead(Input3)==1)) //if everything is zero
 {
    digitalWrite(Output0, HIGH);
    digitalWrite(Output1, LOW);
    digitalWrite(Output2, LOW);
    digitalWrite(Output3, HIGH);
 }
 else if((digitalRead(Input0)==1) && (digitalRead(Input1)==0)  &&
(digitalRead(Input2)==0)  && (digitalRead(Input3)==1)) //0110, go straight
 {
    digitalWrite(Output0, HIGH);
    digitalWrite(Output1, LOW);
    digitalWrite(Output2, LOW);
    digitalWrite(Output3, HIGH);
 }
  else if((digitalRead(Input0)==1) && (digitalRead(Input1)==0)  &&
(digitalRead(Input2)==1)  && (digitalRead(Input3)==1)) //0100, go straight
 {
    digitalWrite(Output0, HIGH);
    digitalWrite(Output1, LOW);
    digitalWrite(Output2, LOW);
    digitalWrite(Output3, HIGH);
 }
  else if((digitalRead(Input0)==1) && (digitalRead(Input1)==1)  &&
(digitalRead(Input2)==0)  && (digitalRead(Input3)==1)) //0010, go straight
 {
    digitalWrite(Output0, HIGH);
    digitalWrite(Output1, LOW);
    digitalWrite(Output2, LOW);
    digitalWrite(Output3, HIGH);
```

```
 }
  else if((digitalRead(Input0)==0) && (digitalRead(Input1)==1)  &&
(digitalRead(Input2)==1)  && (digitalRead(Input3)==1)) //1000 go left
 {
    digitalWrite(Output0, HIGH);
    digitalWrite(Output1, LOW);
    digitalWrite(Output2, LOW);
    digitalWrite(Output3, LOW);
 }
  else if((digitalRead(Input0)==1) && (digitalRead(Input1)==1)  &&
(digitalRead(Input2)==1)  && (digitalRead(Input3)==0)) //0001 go right
 {
    digitalWrite(Output0, LOW);
    digitalWrite(Output1, LOW);
    digitalWrite(Output2, LOW);
    digitalWrite(Output3, HIGH);
 }
  else //default zero
 {
    digitalWrite(Output0, LOW);
    digitalWrite(Output1, LOW);
    digitalWrite(Output2, LOW);
    digitalWrite(Output3, LOW);
 }

}
```

APPENDIX F

THE ORIGINAL ARDUINO CODE FOR WIIMOTE EXTENSION

```
#include <Wire.h>

uint8_t outbuf[6];

void
receiveEvent (int howMany)
{
  while (Wire.available ())
    {
      char c = Wire.receive (); // receive byte as a character
    }
}
void
requestEvent ()
{
  outbuf[0] = nunchuk_encode_byte (125);  // joystick X
  outbuf[1] = nunchuk_encode_byte (126);  // joystick Y
  outbuf[2] = nunchuk_encode_byte (227);  // Axis X
  outbuf[3] = nunchuk_encode_byte (241);  // Axis Y
  outbuf[4] = nunchuk_encode_byte (140);  // Axis Z
  outbuf[5] = nunchuk_encode_byte (1);  // Press C button, byte[5] is
buttons
                                        //C,Z and accelaration data

  Wire.send (outbuf, 6);        // send data packet
}

void
setup ()
{
  Wire.begin (0x52);            // join i2c bus with address 0x52
                                // this is the nunchuk address.
                                // all nunchuks use this address
  Wire.onReceive (receiveEvent);        // register event
  Wire.onRequest (requestEvent);        // register event
}

void
loop ()
{
  delay (100);
}
char
nunchuk_encode_byte (char x)
{
  x = x - 0x17;
  x = (x ^ 0x17);
  return x;
}
```

APPENDIX G

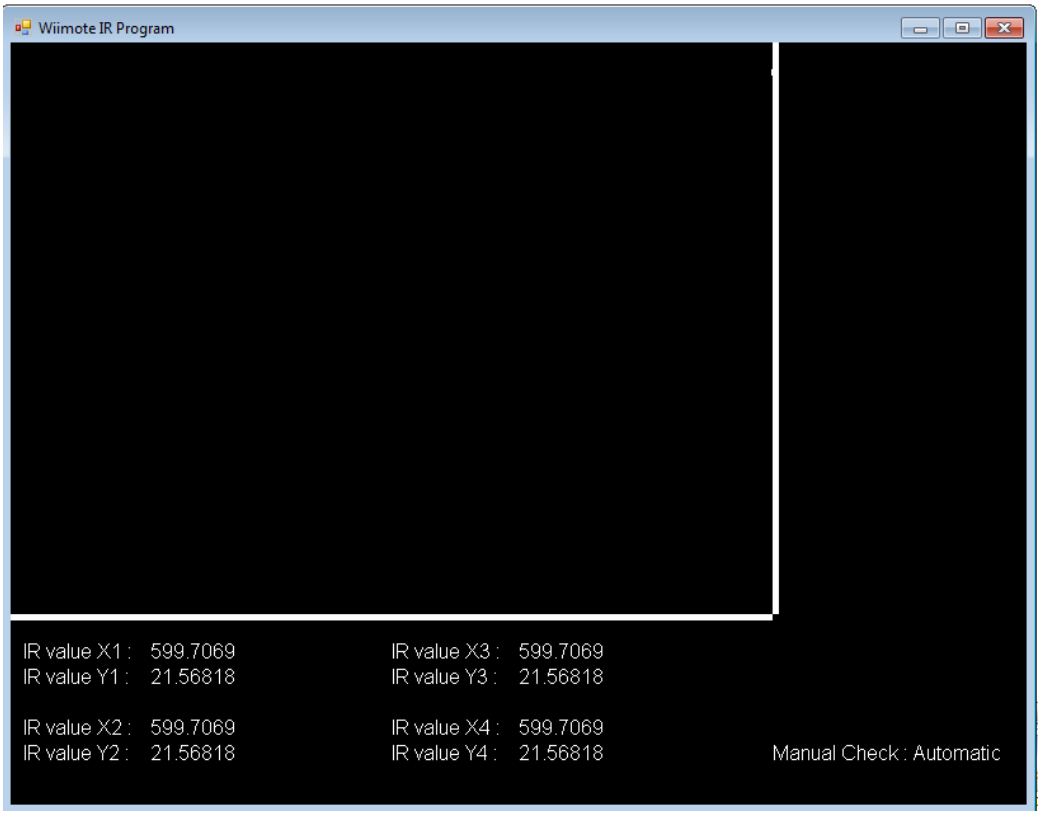
THE CODE FOR THE DIRECT IR CAMERA ACCESS METHOD

```
#include <Wire.h>
#include <PVision.h> // http://www.stephenhobley.com
#include <Servo.h>

Servo myservo;
PVision ircam;
byte result;

//Servo speeds
#define LEFT_SLOW 1510
#define LEFT_MEDIUM 1520
#define LEFT_FAST 1540
#define RIGHT_SLOW 1490
#define RIGHT_MEDIUM 1480
#define RIGHT_FAST 1460
#define STOP 1500

void setup()
{
  pinMode(13, OUTPUT);
  myservo.attach(9);
  ircam.init();
  digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);
}

void loop()
{
  result = ircam.read();

  if (result & BLOB1)
  {

if (ircam.Blob1.X > 480 && ircam.Blob1.X < 520)
{
  myservo.write(STOP);
}

  else if (ircam.Blob1.X < 480)
{
    if (ircam.Blob1.X < 200)
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
   else if (ircam.Blob1.X < 400)
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
    else
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
}

else if (ircam.Blob1.X > 520)
{
    if (ircam.Blob1.X > 800)
    {
```

```
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
    else if (ircam.Blob1.X > 600)
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
    else
    {
    digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
    }
}
  }
   else
  {
  digitalWrite(13, HIGH); //Turns Led on when ircam initiation is complete
  delay(100);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(100);
  }
}
```

APPENDIX H

THE WIIMOTE IR PROGRAM



APPENDIX I

THE WIIMOTE IR PROGRAM

```
using System;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
using Direct3D = Microsoft.DirectX.Direct3D;
using System.Data;
using System.Linq;
using System.Text;
using WiimoteLib;
using System.IO.Ports;

namespace DX9Sample
{
    public class DX9Form : System.Windows.Forms.Form
    {
        private Device d3dDevice = null;
        Direct3D.Font text;

        //serial connection variables
        int input = 0;
        SerialPort port = new SerialPort("COM9", 115200, Parity.None, 8,
StopBits.One);

        //Wiimote variables
        Wiimote Wyrm = new Wiimote();

        float IR1X, IR1Y;
        float IR2X, IR2Y;
        float IR3X, IR3Y;
        float IR4X, IR4Y;
        int FormWidth = 800;
        int FormHeight = 600;
        int manualcheck = 0;
        string arrow = "Uninitiated";
        private VertexBuffer vertices;
        private Texture texture;

        public DX9Form()
        {
            this.ClientSize = new System.Drawing.Size(FormWidth, FormHeight);
            this.Text = "Wiimote IR Program";
            this.SetStyle(ControlStyles.AllPaintingInWmPaint |
ControlStyles.Opaque, true);
        }

        protected override void OnPaint(System.Windows.Forms.PaintEventArgs e)
        {
            this.Render();
            this.Invalidate();
        }
```

```
        protected override void OnKeyDown(System.Windows.Forms.KeyEventArgs e)
        {

            if(e.KeyCode == Keys.Space)
            {
                manualcheck = 0;//disables manual control
                arrow = "Automatic";
            }
            if (e.KeyCode == Keys.W)
            {
                manualcheck = 1;//goes straight
                arrow = "Straight";
            }
            if (e.KeyCode == Keys.S)
            {
                manualcheck = 4;//reverse
                arrow = "Stop";
            }
            if (e.KeyCode == Keys.A)
            {
                manualcheck = 2;// go left
                arrow = "Left";
            }
            if (e.KeyCode == Keys.D)
            {
                manualcheck = 3;//go right
                arrow = "Right";
            }
            if (e.KeyCode == Keys.Escape)
            {
                this.Dispose();
                //break;
            }

        }

        private void InitWiimote()
        {
                Wyrm.Connect();
                Wyrm.SetReportType(InputReport.IRExtensionAccel, true);
                Wyrm.SetLEDs(false, false, true, false);
        }

        private void ExitWiimote()
        {
                Wyrm.SetLEDs(false, false, false, false);
                Wyrm.SetRumble(false);
                Wyrm.Disconnect();
        }

        private void WriteTexts()
        {
            text.DrawText(null, string.Format("IR value X1 :"), new
System.Drawing.Point(10, 470), Color.White);
            text.DrawText(null, string.Format("IR value Y1 :"), new
System.Drawing.Point(10, 490), Color.White);

            text.DrawText(null, string.Format("IR value X2 :"), new
System.Drawing.Point(10, 530), Color.White);
            text.DrawText(null, string.Format("IR value Y2 :"), new
System.Drawing.Point(10, 550), Color.White);
```

```
            text.DrawText(null, string.Format("IR value X3 :"), new
System.Drawing.Point(300, 470), Color.White);
            text.DrawText(null, string.Format("IR value Y3 :"), new
System.Drawing.Point(300, 490), Color.White);

            text.DrawText(null, string.Format("IR value X4 :"), new
System.Drawing.Point(300, 530), Color.White);
            text.DrawText(null, string.Format("IR value Y4 :"), new
System.Drawing.Point(300, 550), Color.White);


            text.DrawText(null, string.Format("Manual Check :"), new
System.Drawing.Point(600, 550), Color.White);

            text.DrawText(null, string.Format(arrow.ToString()), new
System.Drawing.Point(710, 550), Color.White);

            text.DrawText(null, string.Format(IR1X.ToString()), new
System.Drawing.Point(110, 470), Color.White);
            text.DrawText(null, string.Format(IR1Y.ToString()), new
System.Drawing.Point(110, 490), Color.White);
            text.DrawText(null, string.Format(IR2X.ToString()), new
System.Drawing.Point(110, 530), Color.White);
            text.DrawText(null, string.Format(IR2Y.ToString()), new
System.Drawing.Point(110, 550), Color.White);

            text.DrawText(null, string.Format(IR3X.ToString()), new
System.Drawing.Point(400, 470), Color.White);
            text.DrawText(null, string.Format(IR3Y.ToString()), new
System.Drawing.Point(400, 490), Color.White);
            text.DrawText(null, string.Format(IR4X.ToString()), new
System.Drawing.Point(400, 530), Color.White);
            text.DrawText(null, string.Format(IR4Y.ToString()), new
System.Drawing.Point(400, 550), Color.White);
        }
        private void GetIRLights()
        {
            IR1X = (Wyrm.WiimoteState.IRState.IRSensors[0].Position.X * FormWidth)
* 3 / 4;
            IR1Y = (FormHeight -
(Wyrm.WiimoteState.IRState.IRSensors[0].Position.Y * 5 / 7) * FormHeight) * 3 / 4;

            IR2X = (Wyrm.WiimoteState.IRState.IRSensors[1].Position.X) * FormWidth
* 3 / 4;
            IR2Y = (FormHeight -
(Wyrm.WiimoteState.IRState.IRSensors[1].Position.Y * 5 / 7) * FormHeight) * 3 / 4;

            IR3X = (Wyrm.WiimoteState.IRState.IRSensors[2].Position.X * FormWidth)
* 3 / 4;
            IR3Y = (FormHeight -
(Wyrm.WiimoteState.IRState.IRSensors[2].Position.Y * 5 / 7) * FormHeight) * 3 / 4;

            IR4X = (Wyrm.WiimoteState.IRState.IRSensors[3].Position.X * FormWidth)
* 3 / 4;
            IR4Y = (FormHeight -
(Wyrm.WiimoteState.IRState.IRSensors[3].Position.Y * 5 / 7) * FormHeight) * 3 / 4;

            WiimoteDrawIR1(IR1X, IR1Y,0);
            WiimoteDrawIR2(IR2X, IR2Y,1);
            WiimoteDrawIR3(IR3X, IR3Y,2);
            WiimoteDrawIR4(IR4X, IR4Y,3);

            DrawHoriBar();
            DrawVertiBar();
```

```
            text.DrawText(null, string.Format(IR1X.ToString()), new
System.Drawing.Point(110, 470), Color.White);
            text.DrawText(null, string.Format(IR1Y.ToString()), new
System.Drawing.Point(110, 490), Color.White);
            text.DrawText(null, string.Format(IR2X.ToString()), new
System.Drawing.Point(110, 530), Color.White);
            text.DrawText(null, string.Format(IR2Y.ToString()), new
System.Drawing.Point(110, 550), Color.White);

            text.DrawText(null, string.Format(IR3X.ToString()), new
System.Drawing.Point(400, 470), Color.White);
            text.DrawText(null, string.Format(IR3Y.ToString()), new
System.Drawing.Point(400, 490), Color.White);
            text.DrawText(null, string.Format(IR4X.ToString()), new
System.Drawing.Point(400, 530), Color.White);
            text.DrawText(null, string.Format(IR4Y.ToString()), new
System.Drawing.Point(400, 550), Color.White);

        }

        private void WiimoteDrawIR1(float IRX, float IRY, int IRno)
        {
            int size = 5;
            int xcoordinate = (int)IRX;
            int ycoordinate = (int)IRY;
            Texture t = TextureLoader.FromFile(d3dDevice, "V1.jpg");
            CustomVertex.TransformedTextured[] verts = new
CustomVertex.TransformedTextured[4];

            int i = 0;
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xcoordinate, ycoordinate, 0.5F, //
Vertex position
                                        1,                              // rhw
(advanced)
                                        0, 0);                          //
texture coordinates
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xcoordinate, ycoordinate+size, 0.5F,
                                        1,
                                        1, 0);
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xcoordinate + size, ycoordinate +
size, 0.5F,
                                        1,
                                        1, 1);
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xcoordinate+ size, ycoordinate, 0.5F,
                                        1,
                                        0, 1);
            VertexBuffer buf = new
VertexBuffer(typeof(CustomVertex.TransformedTextured), 4, d3dDevice,
Usage.WriteOnly, CustomVertex.TransformedTextured.Format, Pool.Default);

            GraphicsStream stm = buf.Lock(0, 0, 0);
            stm.Write(verts);
            buf.Unlock();

            //vertices.Unlock();
            d3dDevice.SetTexture(0, texture);


            d3dDevice.VertexFormat = CustomVertex.TransformedTextured.Format;
            d3dDevice.SetStreamSource(0, buf, 0);
            d3dDevice.DrawPrimitives(PrimitiveType.TriangleFan, 0, 2);
            t.Dispose();
        }
```

```
        private void WiimoteDrawIR2(float IRX, float IRY, int IRno)
        {
            int size = 5;
            int xcoordinate = (int)IRX;
            int ycoordinate = (int)IRY;

            Texture t = TextureLoader.FromFile(d3dDevice, "V2.jpg");
            CustomVertex.TransformedTextured[] verts = new
CustomVertex.TransformedTextured[4];

            int i = 0;
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xcoordinate, ycoordinate, 0.5F, //
Vertex position
                                        1,                              // rhw
(advanced)
                                        0, 0);                          //
texture coordinates
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xcoordinate, ycoordinate + size, 0.5F,
                                        1,
                                        1, 0);
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xcoordinate + size, ycoordinate +
size, 0.5F,
                                        1,
                                        1, 1);
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xcoordinate + size, ycoordinate, 0.5F,
                                        1,
                                        0, 1);
            VertexBuffer buf = new
VertexBuffer(typeof(CustomVertex.TransformedTextured), 4, d3dDevice,
Usage.WriteOnly, CustomVertex.TransformedTextured.Format, Pool.Default);

            GraphicsStream stm = buf.Lock(0, 0, 0);
            stm.Write(verts);
            buf.Unlock();

            //vertices.Unlock();
            d3dDevice.SetTexture(0, texture);


            d3dDevice.VertexFormat = CustomVertex.TransformedTextured.Format;
            d3dDevice.SetStreamSource(0, buf, 0);
            d3dDevice.DrawPrimitives(PrimitiveType.TriangleFan, 0, 2);
            t.Dispose();
        }
```

```
        private void WiimoteDrawIR3(float IRX, float IRY, int IRno)
        {
            int size = 5;
            int xcoordinate = (int)IRX;
            int ycoordinate = (int)IRY;

            Texture t = TextureLoader.FromFile(d3dDevice, "V3.jpg");
            CustomVertex.TransformedTextured[] verts = new
CustomVertex.TransformedTextured[4];

            int i = 0;
            verts[i++] = new CustomVertex.TransformedTextured(xcoordinate,
                                                              ycoordinate, 0.5F,
                                                              1,
                                                0, 0);
            verts[i++] = new CustomVertex.TransformedTextured(
                                                xcoordinate, ycoordinate + size, 0.5F,
                                                1,
                                                1, 0);
            verts[i++] = new CustomVertex.TransformedTextured(
                                                xcoordinate + size, ycoordinate +
size, 0.5F,
                                                1,
                                                1, 1);
            verts[i++] = new CustomVertex.TransformedTextured(
                                                xcoordinate + size, ycoordinate, 0.5F,
                                                1,
                                                0, 1);
            VertexBuffer buf = new
VertexBuffer(typeof(CustomVertex.TransformedTextured), 4, d3dDevice,
Usage.WriteOnly, CustomVertex.TransformedTextured.Format, Pool.Default);

            GraphicsStream stm = buf.Lock(0, 0, 0);
            stm.Write(verts);
            buf.Unlock();

            //vertices.Unlock();
            d3dDevice.SetTexture(0, texture);


            d3dDevice.VertexFormat = CustomVertex.TransformedTextured.Format;
            d3dDevice.SetStreamSource(0, buf, 0);
            d3dDevice.DrawPrimitives(PrimitiveType.TriangleFan, 0, 2);
            t.Dispose();
        }
```

```
private void WiimoteDrawIR4(float IRX, float IRY, int IRno)
        {
            int size = 5;
            int xcoordinate = (int)IRX;
            int ycoordinate = (int)IRY;

            Texture t = TextureLoader.FromFile(d3dDevice, "V4.jpg");
            CustomVertex.TransformedTextured[] verts = new
CustomVertex.TransformedTextured[4];

            int i = 0;
            verts[i++] = new CustomVertex.TransformedTextured(
                                            xcoordinate, ycoordinate, 0.5F, //
Vertex position
                                            1,                              // rhw
(advanced)
                                            0, 0);                          //
texture coordinates
            verts[i++] = new CustomVertex.TransformedTextured(
                                            xcoordinate, ycoordinate + size, 0.5F,
                                            1,
                                            1, 0);
            verts[i++] = new CustomVertex.TransformedTextured(
                                            xcoordinate + size, ycoordinate +
size, 0.5F,
                                            1,
                                            1, 1);
            verts[i++] = new CustomVertex.TransformedTextured(
                                            xcoordinate + size, ycoordinate, 0.5F,
                                            1,
                                            0, 1);
            VertexBuffer buf = new
VertexBuffer(typeof(CustomVertex.TransformedTextured), 4, d3dDevice,
Usage.WriteOnly, CustomVertex.TransformedTextured.Format, Pool.Default);

            GraphicsStream stm = buf.Lock(0, 0, 0);
            stm.Write(verts);
            buf.Unlock();

            //vertices.Unlock();
            d3dDevice.SetTexture(0, texture);


            d3dDevice.VertexFormat = CustomVertex.TransformedTextured.Format;
            d3dDevice.SetStreamSource(0, buf, 0);
            d3dDevice.DrawPrimitives(PrimitiveType.TriangleFan, 0, 2);
            t.Dispose();
        }
```

```
private void DrawHoriBar()
        {
            int size = 5;
            int xBar = FormWidth * 3 / 4;
            int yBar = FormHeight * 3 / 4;
            Texture t = TextureLoader.FromFile(d3dDevice, "output.jpg");

            CustomVertex.TransformedTextured[] verts = new
CustomVertex.TransformedTextured[4];

            int i = 0;
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xBar, 0, 0.5F, // Vertex position
                                        1,                              // rhw
(advanced)
                                        0, 0);                          //
texture coordinates
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xBar, yBar, 0.5F,
                                        1,
                                        1, 0);
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xBar+size, yBar, 0.5F,
                                        1,
                                        1, 1);
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xBar+size, 0, 0.5F,
                                        1,
                                        0, 1);
            VertexBuffer buf = new
VertexBuffer(typeof(CustomVertex.TransformedTextured), 4, d3dDevice,
Usage.WriteOnly, CustomVertex.TransformedTextured.Format, Pool.Default);

            GraphicsStream stm = buf.Lock(0, 0, 0);
            stm.Write(verts);
            buf.Unlock();

            //vertices.Unlock();
            d3dDevice.SetTexture(0, texture);


            d3dDevice.VertexFormat = CustomVertex.TransformedTextured.Format;
            d3dDevice.SetStreamSource(0, buf, 0);
            d3dDevice.DrawPrimitives(PrimitiveType.TriangleFan, 0, 2);
            t.Dispose();
        }
```

```
private void DrawVertiBar()
        {
            int size = 5;
            int xBar = FormWidth * 3 / 4;
            int yBar = FormHeight * 3 / 4;
            Texture t = TextureLoader.FromFile(d3dDevice, "output.jpg");
            //return t;

            CustomVertex.TransformedTextured[] verts = new
CustomVertex.TransformedTextured[4];



            int i = 0;
            verts[i++] = new CustomVertex.TransformedTextured(
                                        0, yBar, 0.5F, // Vertex position
                                        1,                              // rhw
(advanced)
                                        0, 0);                          //
texture coordinates
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xBar, yBar, 0.5F,
                                        1,
                                        1, 0);
            verts[i++] = new CustomVertex.TransformedTextured(
                                        xBar, yBar + size, 0.5F,
                                        1,
                                        1, 1);
            verts[i++] = new CustomVertex.TransformedTextured(
                                        0, yBar + size, 0.5F,
                                        1,
                                        0, 1);
            VertexBuffer buf = new
VertexBuffer(typeof(CustomVertex.TransformedTextured), 4, d3dDevice,
Usage.WriteOnly, CustomVertex.TransformedTextured.Format, Pool.Default);

            GraphicsStream stm = buf.Lock(0, 0, 0);
            stm.Write(verts);
            buf.Unlock();

            //vertices.Unlock();
            d3dDevice.SetTexture(0, texture);


            d3dDevice.VertexFormat = CustomVertex.TransformedTextured.Format;
            d3dDevice.SetStreamSource(0, buf, 0);
            d3dDevice.DrawPrimitives(PrimitiveType.TriangleFan, 0, 2);
            t.Dispose();
        }
```

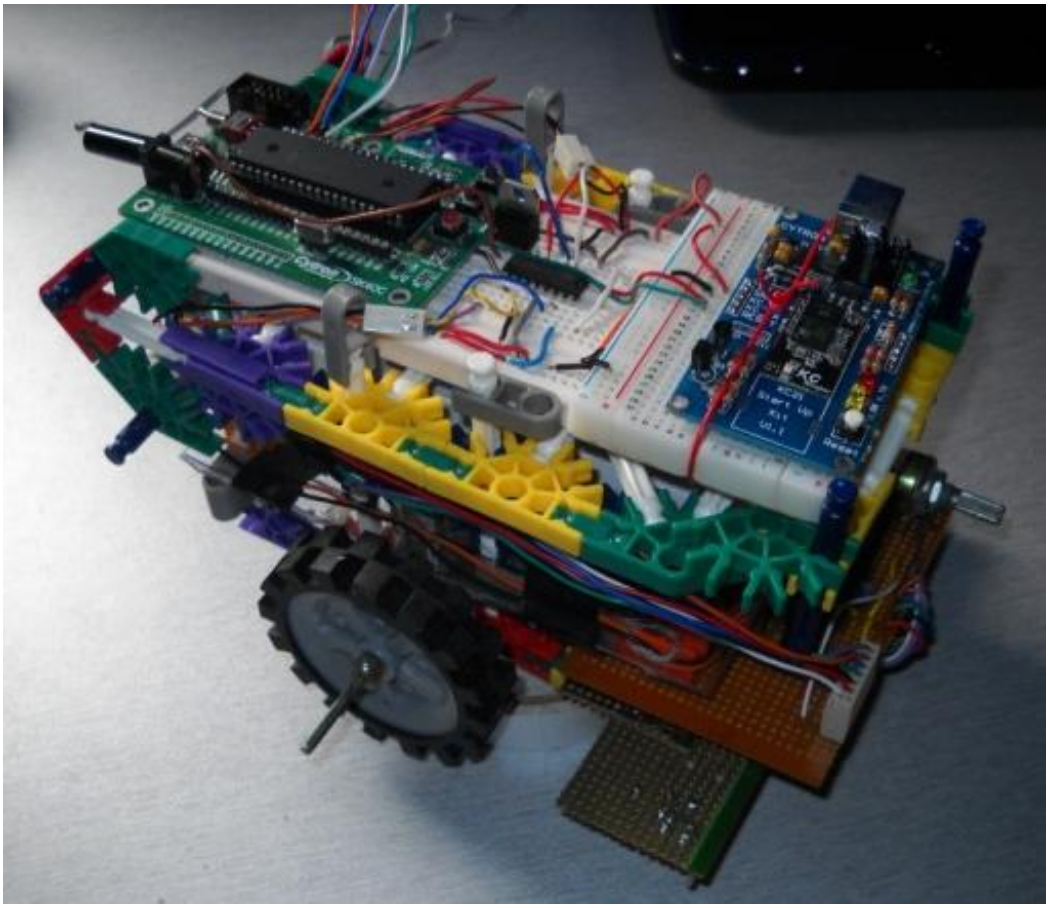
```
private void GetDirections()
{
    if (manualcheck != 0)
        input = manualcheck;
    else
    {
        if (IR1X >= 599) //if value registered is zero
        {
            input = 5;//go linetracker mode
        }

        else if ((IR1X >= 0) && (IR1X < 200)) //
        {
            input = 2; //go left
        }

        else if ((IR1X >= 200) && (IR1X < 400)) //
        {
            input = 1; //go straight
        }

        else if ((IR1X >= 400) && (IR1X < 599)) //
        {
            input = 3;// go right
        }
        else input = 5;
    }

}
private bool SerialOpen()
{
    port.Open(); // Write a string
    return true;

}

private void SerialWrite()
{
    if (input == 1)
    {
        port.Write("1"); //
    }
    else if (input == 2)
    {
        port.Write("2"); //
    }
    else if (input == 3)
    {
        port.Write("3"); //
    }
    else if (input == 4)
    {
        port.Write("4"); //
    }
    else port.Write("5"); //
}
private bool SerialClose()
{
    port.Close();
    return true;
}

protected override void Dispose(bool disposing)
{
    base.Dispose(disposing);
}
```

```
        static void Main()//this is the main program
        {
            using (DX9Form frm = new DX9Form())
            {

                frm.Show();
                frm.InitDirectX();
                frm.InitWiimote();
                frm.InitFont();
                frm.SerialOpen();
                Application.Run(frm);
                frm.SerialClose();
                frm.ExitWiimote();
            }

        }

        /// <summary>
        /// This method basically creates and initialize the Direct3D device and
        /// anything else that doens't need to be recreated after a device
        /// reset.
        /// </summary>
        private void InitDirectX()
        {
            // Does the hardware support a 16-bit z-buffer?
            if (!Manager.CheckDeviceFormat(Manager.Adapters.Default.Adapter,
                                           DeviceType.Hardware,

Manager.Adapters.Default.CurrentDisplayMode.Format,
                                           Usage.DepthStencil,
                                           ResourceType.Surface,
                                           DepthFormat.D16))
            {
                // POTENTIAL PROBLEM: We need at least a 16-bit z-buffer!
                return;
            }

            //
            // Do we support hardware vertex processing? if so, use it.
            // If not, downgrade to software.
            //

            Caps caps = Manager.GetDeviceCaps(Manager.Adapters.Default.Adapter,
                                              DeviceType.Hardware);
            CreateFlags flags;

            if (caps.DeviceCaps.SupportsHardwareTransformAndLight)
                flags = CreateFlags.HardwareVertexProcessing;
            else
                flags = CreateFlags.SoftwareVertexProcessing;

            //
            // Everything checks out - create a simple, windowed device.
            //
```

```
            PresentParameters d3dpp = new PresentParameters();

            d3dpp.BackBufferFormat = Format.Unknown;
            d3dpp.SwapEffect = SwapEffect.Discard;
            d3dpp.Windowed = true;
            d3dpp.EnableAutoDepthStencil = true;
            d3dpp.AutoDepthStencilFormat = DepthFormat.D16;
            d3dpp.PresentationInterval = PresentInterval.Immediate;

            d3dDevice = new Device(0, DeviceType.Hardware, this, flags, d3dpp);

            // Register an event-handler for DeviceReset and call it to continue
            // our setup.
            d3dDevice.DeviceReset += new System.EventHandler(this.OnResetDevice);
            OnResetDevice(d3dDevice, null);
        }
        private void InitFont()
        {
            System.Drawing.Font systemfont = new System.Drawing.Font("Arial", 12f,
FontStyle.Regular);
            text = new Direct3D.Font(d3dDevice, systemfont);
        }

        /// <summary>
        /// This event-handler is a good place to create and initialize any
        /// Direct3D related objects, which may become invalid during a
        /// device reset.
        /// </summary>
        public void OnResetDevice(object sender, EventArgs e)
        {
            // This sample doens't create anything that requires recreation
            // after the DeviceReset event.
            Device device = (Device)sender;

            device.RenderState.ZBufferEnable = true;
            device.RenderState.Lighting = false;


        }

        private void Render()//this is where D3D comes in
        {
            d3dDevice.RenderState.CullMode = Cull.None;

            // Determine if wireframe is needed
            d3dDevice.RenderState.FillMode = FillMode.Solid;

            d3dDevice.Clear(ClearFlags.Target | ClearFlags.ZBuffer,
                            Color.Black, 1.0f, 0);

            d3dDevice.BeginScene();

            // Render geometry here...
            WriteTexts();
            GetIRLights();
            GetDirections();

            d3dDevice.EndScene();
            SerialWrite();

            d3dDevice.Present();
        }
    }
}
```

APPENDIX J

THE FINAL ROBOT DESIGN



APPENDIX K

THE FINAL MICROCONTROLLER CODE

```
#include <pic.h>
//#include <delay.h>
void delay(unsigned int data);
void linetrack(int check);
void straight(void);
void left(void);
void right(void);
void stop(void);
void reverse(void);
void serialcheck(int check);

__CONFIG(0x3F32);
#define L1 RB6
#define L2 RB7
#define L3 RD4

//input
#define Input0 RB0
#define Input1 RB1
#define Input2 RB2
#define Input3 RB3
#define Input4 RB4

//output
#define Output0 RD2
#define Output1 RD3
#define Output2 RD4
#define Output3 RD5

unsigned char a;
int level;
```

```
void init(void)     // subroutine to initialize
{
       SPBRG=0x0A;  // set baud rate as 115200 baud
       BRGH=1;
       TXEN=1;
       CREN=1;
       SPEN=1;
       TRISD = 0b00000000;
       //seg   = 0b00000000;
       //TRISC = 0b00000000;
       //PORTC   = 0b00000000;
}
void display(unsigned char c
{
       while (TXIF == 0);
       TXREG = c;
}
unsigned char receive(void) // subrountine to receive text from PC
{
       while (RCIF == 0);
       a = RCREG;
       return a;
}

void main(void)
{
       TRISB = 0b00011111; //port b output, input
       init();
       level = 5;
       while(1)
{
a = receive();
              if (a == '1') //signal 1, go straight
                     {
                            level = 1;
                     }
              else if (a == '2')  // signal 2, go left
                     {
                            level = 2;
                     }
```

```
            else if (a == '3')  // signal 3, go right
                  {level = 3;
                  }
            else if (a == '4')  //signal 4, stop
                  {level = 4;
                  }
            else if (a == '5')  //signal 5, stop
                  {level = 5;
                  }
            else if (a == '6')  //signal 5, stop
                  {level = 6;
                  }
            else   level = 5;
            serialcheck(level);
      }
}

void serialcheck(int check)
{
            if (check == 1)     // signal 1, go straight
                  {
                        straight();
                  }
            else if (check == 2)       // signal 2, go left
                  {
                        left();
                  }
            else if (check == 3)       // signal 3, go right
                  {
                        right();
                  }
            else if (check == 4)       //signal 4, stop
                  {
                        stop();
                  }
            else if (check == 5)       //signal 5, linetracker mode
                  {
                        linetrack(check);
                  }
```

```
            else if (check == 6)       //signal 6
                  {
                        reverse();
                  }
            else linetrack(check);
}
void linetrack(int check)
{     L2 = 1;
if ((Input0==1)&&(Input1==1)&&(Input2==1)&&(Input3==1)&&(Input4==1))
//No input, no go
                  {
                        straight();
                  }
else if ((Input0==1)&&(Input1==1)&&(Input2==0)&&(Input3==1)&&(Input4==1))
//go straight
                  {
                        straight();
                  }
else if((Input0==1)&&(Input1==0)&&(Input2==1)&&(Input3==1)&&(Input4==1))
// go left
                  {
                        left();
                  }
else if ((Input0==0)&&(Input1==1)&&(Input2==1)&&(Input3==1)&&(Input4==1))
// go left
                  {
                        left();
                  }
else if ((Input0==0)&&(Input1==0)&&(Input2==1)&&(Input3==1)&&(Input4==1))
// go left
                  {
                        left();
                  }
else if ((Input0==1)&&(Input1==1)&&(Input2==1)&&(Input3==0)&&(Input4==1))
// go right
                  {
                        right();
                  }
```

```
else if ((Input0==1)&&(Input1==1)&&(Input2==1)&&(Input3==0)&&(Input4==0))
// go right
                    {
                           right();
                    }
else if ((Input0==1)&&(Input1==1)&&(Input2==1)&&(Input3==1)&&(Input4==0))
// go right
                    {
                           right();
                    }
             else          //no go
                    {
                           stop();
                    }
       L2 = 0;
}

void straight(void)
{
                           Output0 = 1;
                           Output1 = 0;
                           Output2 = 0;
                           Output3 = 1;
}
void left(void)
{
                           Output0 = 0;
                           Output1 = 0;
                           Output2 = 0;
                           Output3 = 1;
}
void right(void)
{
                           Output0 = 1;
                           Output1 = 0;
                           Output2 = 0;
                           Output3 = 0;
}
```

```
void stop(void)
{
                            Output0 = 0;
                            Output1 = 0;
                            Output2 = 0;
                            Output3 = 0;
}

void reverse(void)
{
                            Output0 = 0;
                            Output1 = 1;
                            Output2 = 1;
                            Output3 = 0;
}
```