Semi-Autonomous Pac-Man Entertainment System

Submitted To

Yeojoon Kim and Tracy Wuster
Michael Becker

Prepared By

Aaron Landy,
David Tian,
Gary Shi-Yin Leung,
Si Chen,
Yu-Chieh Lee

EE464 Senior Design Project
Electrical and Computer Engineering Department
University of Texas at Austin

Fall 2010
### CONTENTS

**TABLES**................................................................................................................................. v
**FIGURES**............................................................................................................................. iv
**EXECUTIVE SUMMARY** ..................................................................................................... ii

1.0 INTRODUCTION ............................................................................................................... 1
2.0 DESIGN PROBLEM STATEMENT .................................................................................. 2
3.0 DESIGN PROBLEM SOLUTION .................................................................................... 2
   3.1 GAME AI ................................................................................................................ 4
      3.1.1 Game Mechanic ............................................................................................. 4
      3.1.2 Ghost AI ....................................................................................................... 5
   3.2 ROBOT HARDWARE .............................................................................................. 6
   3.3 CONTROL SYSTEM .............................................................................................. 6
   3.4 SENSORS AND DIGITAL COMPASS ................................................................. 7
   3.5 MAZE PLAYING FIELD ....................................................................................... 8
   3.6 POSITION TRACKING ....................................................................................... 9
      3.6.1 Radio Frequency Identification ................................................................. 11
      3.6.2 Infrared Range Sensors ............................................................................. 11
      3.6.3 Image Recognition .................................................................................... 12
   3.7 WIRELESS COMMUNICATION ........................................................................... 13
4.0 DESIGN IMPLEMENTATION ......................................................................................... 16
   4.1 GHOST AI .......................................................................................................... 17
      4.1.1 Deterministic Model .................................................................................. 17
      4.1.2 Non-Deterministic Model ......................................................................... 18
      4.1.3 AI Integration ............................................................................................ 19
   4.2 ROBOT HARDWARE ............................................................................................ 20
      4.2.1 Motor And Gearbox .................................................................................. 20
      4.2.2 H-Bridge Motor Driver Circuit ................................................................ 21
      4.2.3 Voltage Regulator/DC-DC converter ..................................................... 21
      4.2.4 C328 & Servo ......................................................................................... 22
CONTENTS (Continued)

4.3 CONTROL SYSTEM .................................................................23
  4.3.1 Scheduler .........................................................................23
  4.3.2 I2C Bus Interfacing ...........................................................24
  4.3.3 Sensor and Compass Data Handling Procedure ..................25
  4.3.4 PWM Motor Control .........................................................26
  4.3.5 Image Transmission ..........................................................27
4.4 SENSORS AND DIGITAL COMPASS .......................................28
4.5 MAZE PLAYING FIELD ..........................................................29
4.6 POSITION TRACKING ...........................................................30
4.7 WIRELESS COMMUNICATION ..............................................32

5.0 TEST AND EVALUATION ........................................................33
  5.1 ARTIFICIAL INTELLIGENCE ..................................................34
  5.2 ROBOTIC HARDWARE ..........................................................34
  5.3 CONTROL SYSTEM .............................................................35
  5.4 IR SENSORS AND DIGITAL COMPASS .................................35
  5.5 MAZE PLAYING FIELD .........................................................36
  5.6 POSITION TRACKING ..........................................................37
  5.7 WIRELESS COMMUNICATION ..............................................38

6.0 TIME AND COST CONSIDERATIONS .....................................39

7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN .......................40

8.0 RECOMMENDATIONS ..........................................................40
  8.1 ARTIFICIAL INTELLIGENCE ..................................................40
  8.2 ROBOT HARDWARE .............................................................40
  8.3 ROBOT HARDWARE ............................................................41
  8.4 MAZE PLAYING FIELD ........................................................41
  8.5 POSITION TRACKING ........................................................41
  8.6 WIRELESS COMMUNICATION ..........................................42

9.0 CONCLUSIONS ........................................................................43

REFERENCES ..............................................................................43
## CONTENTS (Continued)

<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>MESSAGE PASSING PROTOCOL DEFINITION</td>
<td>A-1</td>
</tr>
<tr>
<td>B</td>
<td>API DOCUMENTATION</td>
<td>B-1</td>
</tr>
<tr>
<td>C</td>
<td>MAZE CIRCUIT DIAGRAM</td>
<td>C-1</td>
</tr>
<tr>
<td>D</td>
<td>IMAGE RECOGNITION MARKER SHAPES</td>
<td>D-1</td>
</tr>
<tr>
<td>E</td>
<td>GAME STATISTICS</td>
<td>E-1</td>
</tr>
<tr>
<td>F</td>
<td>PROJECTED COMPONENTS AND PROTOTYPE COSTS</td>
<td>F-1</td>
</tr>
</tbody>
</table>
TABLES
1  Game statistics .................................................................................................................. E-1
2  Projected Component and Prototype Costs ................................................................. F-1
FIGURES

1 System Level Design ................................................................. 3
2 Original Design Of Pac-Man Maze .......................................... 4
3 Pac-Man II Design Of Maze .................................................. 5
4 Block Diagram Of Control System .......................................... 6
5 The Primitive Design Of The Maze Playing Field .................... 8
6 Placement of RFID Tags On the Maze ................................... 11
7 Infrared Range Sensor Based Position Tracking Configuration .... 12
8 Layered Design Of The Wireless Communication System .......... 14
9 POSIX System Call and Serial Port Library Software Primitive .... 15
10 Message Protocol Structure .................................................. 16
11 Pac-Man II With Paths Shown .............................................. 18
12 Pac-Man (left) and Ghost (right) Robots ................................ 20
13 Low Voltage, High-Performance D.C. Motor & Tamiya Twin Gear Box 21
14 H-Bridge Motor Driver – L293 ............................................. 21
15 Efficiency vs. Output Current ............................................... 22
16 C328 Camera and XBee Module .......................................... 22
17 Control System Structure ..................................................... 23
18 TCB Switch Mechanism ........................................................ 24
19 The I2C Bus Structure .......................................................... 25
20 The Data Sequence of I2C Bus ............................................. 25
21 The Flow Chart of Handling Procedure .................................. 26
22 An Example of PWM Waveform .......................................... 27
23 The Actual Implementation Of The Maze Playing Field ............ 29
24 Image Recognition System Sample Output ............................ 32
25 Original (top) and Interrupt-based Background XBee Library (bottom) Calling Scheme ......................................................... 33
EXECUTIVE SUMMARY

The Pac-Man II project brings the classic game Pac-Man to life in real-world dimensions. Inspired by the original Pac-Man arcade game, we created our own rendition of a real-life Pac-Man game using modern robotics technology. The synergy of robots, wooden maze, Artificial Intelligence (AI), image recognition, and wireless control presents a modern twist to the game play. Despite the dramatic virtual-to-real-life change, most of the game mechanics are still maintained to facilitate the user learning curve. The major difference is that there is one ghost robot and one Pac-Man robot playing on a smaller maze. The user still controls the Pac-Man robot to collect food, represented by LED-lights, on a wooden maze while avoiding the AI controlled Ghost robot. A server conducts the entire game, guiding the ghost using AI and blinking LED lights as Pac-Man rushes to collect food. Altogether, our Pac-Man II offers a real-time system that provides an entertaining game experience.

We employed a granular system by dividing up the functionality to individual modules. The AI module determines the actions of the ghost robot. This AI algorithm, together with the maze design and robotic software, provides a non-deterministic approach to the game play in order to allow the user roughly 50% chance of winning. The ghost robot utilizes an arsenal of IR sensors mounted in all directions and a compass sensor to properly navigate through the maze. The Pac-Man robot is controlled by a robust full-duplex wireless system using ZigBee. Behind the scenes, a central server uses image recognition for position tracking of the Pac-Man. As Pac-Man passes by “food”, the server sends ZigBee signals to a control circuit on the maze, which turns off the LED lights.

Unfortunately, we faced many issues in this arduous implementation plan. The AI needed to adjust in order to provide a fair chance for Pac-Man and ghost to win. This is mostly caused by physical world issues. When the robot runs low on batteries, it is plagued by inaccurate motor movements and crashes into the wall nearly 15% of the time. In addition, there are sometimes erroneous sensor readings and calibration issues. The compass used to guide turning is occasionally over-shot due to the variation of the momentum of the robot. Due to the inaccurate IR sensor readings under 21 cm ranges, the maze was forced to be resized. This limitation cut the maze down to 4 lanes with dead-ends. On the server side, we had issues with position tracking. We depended on using IR sensors and RFID tags to determine location of the ghost and Pac-Man; however, our RFID tags were not shipped on time, and this delay caused us to resort to image processing for position tracking.

For future plans, there are several improvements that can be made. If there were more funding, we would first replace the old IR sensors with more accurate IR sensors that can detect down to 5 cm. In turn, this will reduce the maze size, thus allowing more paths to be available. With the advent of new paths comes the possibility of more robots and a better AI.

Overall, the requirements were met. The system was real-time and fun to play. Although there were many challenges, we are satisfied to have overcome each issue. Each module did its part to contribute to a complete and functional system. With this, we are proud to present the Pac-Man II entertainment system.
1.0 INTRODUCTION

This report gives a final and complete review of our team’s project to design and implement a semi-autonomous robotic Pac-Man game. Our team developed a game system consisting of two custom-built competing robots, an interactive maze playing field, and a central coordinating server. Our team consisted of five members with a wide variety of specialties in Electrical and Computer Engineering, Si Chen, Aaron Landy, Yu-Chihe Lee, Gary Leung, and David Tian. Teaching Assistant Yeojoon Kim and Professor Michael Becker supervised the design and implementation of our project.

This project has been an exploration in the design, implementation, and coordination of mobile robots, as well as an exercise in implementing a multi-agent computer vision system consisting of state-of-the-art sensors and image recognition technologies. While the primary goal of this project was to produce an entertainment system, the autonomous mobile sensor platform, together with the omniscient camera-equipped coordinator, has clear potential for applications such as human assistance, manual labor, construction, and other practical tasks. Teams of such robots could be used to perform work in environments unsafe, impractical, or undesirable for humans.

This paper will begin with a discussion of our design challenge and goals for our system. We will then present our ideal design solution, actual implementation, and testing and evaluation process in the context of each module of the system; these modules include Artificial Intelligence (AI), maze playing field, robotic control system, robotic hardware, position tracking, and wireless communication. The paper will review the final outcome of our project in terms of time and cost considerations as well as safety and ethical concerns. Finally, we provide our recommendations for future improvements to the system.
2.0 DESIGN PROBLEM STATEMENT

Our design is intended to present a classic video game to players in an entirely new way by setting the action in the physical world using modern technology. We will use state-of-the-art robotics, sensors, wireless communication, and artificial intelligence algorithms to create a system faithful to the original game-play while providing players a more immersive experience. The design of the robots to be used in this system presents a significant technical challenge; our team must combine a multi-sensor real-time data acquisition system with a powerful processor on a small mobile platform. These robots must also be able to wirelessly communicate processed sensor data to and receive commands from a central server in real-time. The system must also allow these computer-controlled robots to work collectively towards their common goal of hunting down the user-controlled robot Pac-Man.

A successful, complete system should provide the end user with an enjoyable game-play experience. The Pac-Man robot should respond quickly and accurately to user control and the enemy ghost robots should effectively pursue Pac-Man. All robots should move at a speed to traverse the maze in an appropriate amount of time and should be intuitive to control. The central server should allow the user to monitor the status of and control the entire system. The wireless network should allow commands and messages to be sent with high bandwidth and low latency sufficient to allow the system to operate properly. The message protocol should be sufficient to dictate message format and traffic while not placing unnecessary burden of processing time to pass messages. Lastly, as we are constructing a game, the system should also be fun to play.

Initially, our plan was to recreate the real-life Pac-Man using the same design – five robots, Pac-Man maze, and food pills in the maze. We had planned on using four ghost robots each with a camera for image recognition in order to discover the maze as well as track Pac-Man. We had also planned to design the shape of the maze to be the same as the original Pac-Man maze. The purpose of keeping the game mechanics the same is to reduce the learning curve of users. Unfortunately, we realized that the cost of a robot is too expensive for our budget, so we could only afford two – one Pac-Man robot versus one ghost robot. Furthermore, due to the size of the robot and the range of its sensors, we were forced to expand lane width. This restriction caused
us to consider a maze with less open lanes. Lastly, the camera for tracking is now mounted on the server in order to oversee the entire game. Despite these issues, the goal still remains the same.

3.0 DESIGN PROBLEM SOLUTION

Our approach to the design problem relies on employing modules that best substitute the modules of the original Pac-Man game. For example, instead of the virtual Pac-Man and ghosts, we used two sensor-guided robots to serve the purpose. The maze platform is built on wood and uses LED lights as food. In order to track the locations of both the ghost and Pac-Man, we used a central server with a camera to auto-detect positions of both. Furthermore, the server also contained the ghost AI and LED food control. A detailed system level solution is shown in the Figure 1 below.

![Figure 1: System Level Design With All Components](image)

The interaction of the server, robots, and maze defines the game. Each component passes important data and control commands. In our solution, the user interacts with the Pac-Man robot to collect all of the food in the maze. The server controls the ghost to track down Pac-Man using AI. The positions of both Pac-Man and ghost are stored on the server. This way, the server
detects the end of game for when all food is collected or Pac-Man is captured. The following subsections cover idea solutions for the game AI, robot hardware, robot software, maze, position tracking, and wireless communication.

3.1 GAME AI
In order to create an entertaining system, the ghost must move intelligently to capture Pac-Man. Since Pac-Man is controlled by the human, there is no AI needed for the Pac-Man. There must also be a balance between the ghost and the Pac-Man. This means that neither the ghost nor the Pac-Man can win 100% of the time. The next sections detail the ideal ghost AI algorithm.

3.1.1 Game Mechanics
Before explaining the AI, we first present the game mechanics of our solution. Initially, we had planned for the maze to appear in the same design as the original Pac-Man, shown in Figure 2 below.

![Figure 2: Original Design Of Pac-Man Maze](image_url)

The idea is that ghosts must capture Pac-Man by surrounding Pac-Man using chokepoints. For example, if Pac-Man is at the bottom left corner of the maze, two ghosts must come from both the north side chokepoint and the east side chokepoint to trap Pac-man. Unfortunately, due to maze size and lane size restrictions, our maze was forced to expand to only four paths, shown in Figure 3 on the next page.
Given the restricted maze, there are noticeably only four paths from where the Pac-Man is currently placed. Furthermore, each path leads to a dead-end. This dead-end allows the ghost to capture Pac-Man, should the ghost enter the same path as Pac-Man. This is because Pac-Man can only go straight or turn around once it is traversing a path. Should the ghost follow Pac-Man into the same path, Pac-Man is forced to continue forwards until the dead-end. In this case, the chase ends in victory for the ghost.

3.1.2 Ghost AI
Our ghost AI takes full advantage of the maze environment. Initially, our AI algorithm was for the original design of the Pac-Man maze. The algorithm took a classical AI approach where all outcomes were near deterministic. This means that for the same input commands of the ghost, the output is relatively the same each time. The initial AI algorithm used chokepoints to cut off Pac-Man. The ghosts will find four different chokepoints and wait there until Pac-Man has moved to a location where it is vulnerable for closing in.

Upon changing the maze structure, the AI also adapted. The AI solution for the smaller maze in Figure 3 now takes advantage of dead-ends. Also, there is only one ghost. The chokepoint is reduced to one in this maze. However, we added the restriction that the ghost must move in one of the four paths and cannot stand still. Furthermore, the ghost would choose a path randomly.
This randomness factor, when added to the algorithm, mutates the AI algorithm to a non-deterministic model. In other words, the outcome is no longer predictable.

### 3.2 ROBOT HARDWARE

The hardware portion of robots carries out actions performed on the field. A good hardware design could provide a real-life experience to human players. Hardware design will vary with the roles of robots. Power consumption and size play important roles in the design of our autonomous system. From the standpoint of Pac-Man players, robots must be able to run continuously without crashing or requiring battery recharges. This is especially important for hardware that has high mobility and functionality. The robots should be built in compact sizes to meet the design specification of our maze. As the number of modules increases, added weight to the robotic platform causes higher current draw from the motors. Design of the robots is greatly affected by the selection of hardware components, so these should be chosen carefully before prototyping.

### 3.3 CONTROL SYSTEM

The heart of each robot is a control system consisting of a microcontroller and a software scheduler. The control system software module contains five device drivers. A control system is essential to a robot in order to coordinate that interaction between the different hardware modules on the robot. A software scheduler balances the execution time allocated for each device; the execution time will depend on priority and runtime conditions. The control system coordinates devices on the robot and facilitates the performance of the robots. Figure 4 below shows a block diagram of the control system.

![Block Diagram Of Control Subsystem](image-url)
First, infrared sensor signal is a binary data range from 10 to 16 bits which is transmitted from infrared sensor equipped off chip. Sensor encodes physical distance between the object and the robot as 10 bits binary value. This data could be used to avoid potential collision and to calculate the position of object relative to the robot. After received this binary data sensor device driver will convert it back to physical distance and store it in memory location for kernel to use.

Second, the signal from digital compass is used to notify robot’s current direction. This is a 32 bits binary number which represents the angle relative to the north. Microprocessor will store that information into a temporary buffer and wait for handling procedure to process. What handling procedure will do is to compare the current angle with desirable angle. If there is any deviation, it will try to correct it by adjusting the PWM output on the wheel.

Third, real time image is captured by webcam and transmitted pixel by pixel to CPU subsystem. Each image frame demands larger memory space and transmission bandwidth than other signals. The JPEG color camera has 640 by 480 resolutions and each pixel contains 8 bits RGB value. Each frame will need 308 kilo-bytes or 2.5 megabits memory space. Since our microprocessor’s clock rate is 500M Hz and equipped with 256MB memory, it will be able to process image data. Then Pac-Man robot will call wireless subsystem to send those pixels back to server for displaying. Finally, the command signal is came from server and used to instruct PacMan what direction to go or any other actions needs to be performed on the field.

Besides, there are two kinds of output signals could be passed out by control system. Mobility control signal is an 8 bits binary pixel and one bit PWM signal. The 8 bit pixel is sent back to server for display the field image and PWM signal is used to adjust corresponding servo for direction control. However, those two outputs are not dependent on each other and processor can send only one at any given time. Therefore system software need to balance the CPU time allocated for them to prevent crash.

3.4 SENSOR AND DIGITAL COMPASS
As stated in our Requirements and Specifications report, the robot must have the ability to avoid obstacles and collision while cruising in the maze. Also, the robot needs to self-correct, if any
deviation occurs. Initially, we only planned on using five IR sensors as our obstacle-detection device, but we add one more digital compass to our robots during the implementation phase. While implementing IR sensor and testing its functionality we realized that IR sensor could provide information about the distance between robot and obstacles, but it could not tell us if the robot heading to the wrong direction. Without a correct heading the robot cannot stay on the correct path. Combining IR sensors and digital compass provides a better robotic performance to our project.

3.5 MAZE PLAYING FIELD

As stated in our implementation plan, we planned to build a wooden playing field that was four feet long by eight feet wide. The primitive design is shown in Figure 5 below. The channels of the maze are about one foot wide, and the walls are 3 inches tall. This playing field is designed for robots approximately the size of a compact disc (CD).

![Figure 5: The Primitive Design Of The Maze Playing Field](image)

The wooden platform is made of pegboard and plywood. LEDs, representing the food Pac-Man must eat, are placed in the holes of the pegboard with all wiring and control circuitry hidden and protected beneath the platform. The maze can be divided into a four by eight grid with one LED per grid; in total, there are 32 LEDs on the maze. These LEDs will turn off when Pac-Man passes over them, thereby “eating” that food. When all the LEDs are off, Pac-Man wins the game. The maze is controlled by a Freescale MC9S12DP512 microcontroller, which receives control signals from the server and controls the LEDs on the maze.
The maze LED driver circuit is a scanned LED interface, which organizes the LEDs in a matrix with rows and columns. Each row and each column of the circuit has a unique output pin. Si decided to use this circuit because it is simple and costs less. An alternate circuit option we considered was the direct LED interface. The direct interface requires a unique output pin for each LED. As a result the direct interface requires 32 output pins while the scanned interface requires only 12 (4 rows + 8 columns). The scanned interface also requires far fewer resistors and transistors than the direct interface. Therefore, Si chose the scanned interface circuit, which is shown in Figure X. Since these LEDs drive large current levels, Si separated the power of the controller and that of the LEDs. She also used high power Darlington transistors (TIP120 and TIP125), which has a high current gain and high collector current, to drive the circuit.

The maze is controlled by a Freescale MC9S12DP512 microcontroller. This microcontroller has two primary tasks, scanning the LED matrix to maintain the persistence-of-vision illusion and communicating with the server. The microcontroller communicates with the server through a Zigbee wireless module. The server sends the position of Pac-Man to the microcontroller, and the microcontroller will turn off the LED that Pac-Man passed over.

Our control program is based on persistence-of-vision, a phenomenon of the eye by which an afterimage is thought to persist for approximately one twenty-fifth of a second on the retina [1]. If the LEDs refresh fast enough, we believe that they are always on, even though each LED is only powered on for a fraction of a second. For example, to refresh each LED at a frequency of 100Hz, we can use a periodic interrupt at 700Hz. Then there will be seven interrupts per LED cycle, during each of which one row of LEDs will be updated. All LEDs would be updated in one 100 Hz cycle. Since the LEDs are updated approximately every one millisecond, human eyes cannot see the LEDs powering off.

3.6 POSITION TRACKING

A key component in our design is a robot position tracking system. In order for our system to function, the system must be aware of the location of each robot in the maze. The system must be able to track Pac-Man’s progress traveling through the maze and collecting food, and must be able to determine if the Ghost robot has caught Pac-Man. In our initial design process, we
considered three alternative design solutions for the position tracking system. These include Radio Frequency Identification (RFID), infrared (IR) range sensors, and image recognition.

The team evaluated each alternative based on the requirements of the position tracking system, including resolution, latency, and implementation simplicity. Resolution refers to the smallest measurable distance; our design required a minimum resolution of one maze grid square. One grid square represents one logical game position that each robot can occupy. Latency refers to the total time between stable measurement outputs and indicates the response time of the system. This latency must be sufficiently faster than the operating speed of the robots such that their movement can be accurately represented by the tracking output. We determined that the measurement latency should be less than 250 ms. Finally, the system design must be simple enough to allow the highest chances for a successful implementation.

The team analyzed each alternative based on the above requirements. We determined that RFID provided the greatest risk and functionality tradeoff and was the most sensible choice. Infrared ranging provided the simplest and most reliable solution but with incomplete functionality; this would be a satisfactory backup plan. While image processing offered the ideal results, it would also be the most complex to implement and the riskiest to fail. Because position tracking was so vital to the operation of the system, the team chose to pursue all three alternatives to ensure the highest chance of success with the greatest results. The following subsections discuss the design solution for each alternative.
3.6.1 Radio Frequency Identification

The basic operating principle of using RFID tags for position tracking is to place an RFID reader on the bottom of each robot, and space tags periodically around the maze at each grid position. When a robot passes over an RFID tag, the unique ID number of the tag would be transmitted to the robot from the reader through a serial port, and that ID could then be transmitted back to the server. A simple lookup table could then map tag numbers to grid spaces on the maze. Figure 6 below shows the arrangement of RFID tags throughout the maze. In the figure, an RFID tag is highlighted for detail, and each circle represents a robot.

![Figure 6: Placement of RFID Tags On the Maze](image)

RFID position tracking provides resolution equivalent to the frequency of tag placement on the maze. Tags would be placed at exactly one tag per grid space, allowing for the exact level of resolution needed by the system. The position calculation operation is very low latency and would occur at a rate nearly equal to the robot speed. Experimentally, total latency from the time the tag was read, then received by the robot controller and transmitted by Zigbee was less than 100 ms.

3.6.2 Infrared Range Sensors

To determine robot position using infrared range sensors, sensors were aimed down each of the four rows of the maze. If no robot was present in a given row, the sensor for that row would read a distance greater than the length of the maze. Any of the four sensors reading a value less than the length of the maze indicated that a robot was present in that row. This output range represented one dimension of the robot’s position; the other dimension was indicated based on
which of the four sensors read that value. Figure 7 below illustrates the physical operation of this system. In the figure, the arrows represent the infrared signals emitted and detected by the sensors and reflected by the Pac-Man robot; the circles in the image represent the robots.

![Figure 7: Infrared Range Sensor Based Position Tracking Configuration](image)

The measurements for this solution were taken using Sharp GP2Y0A02YK0F infrared range sensors, which have a range between 15 and 150 centimeters [3]. The sensors were interfaced to a microcontroller’s analog to digital converter, and the output was filtered in software to both ensure accuracy and convert the measurement from voltage to distance.

This solution offers a one-of-four resolution in one dimension, and a distance measurement accurate within 10 centimeters in the other dimension. This easily meets the system’s resolution requirements. Additionally, the simple analog voltage output was easy to interface and implement; and the filtered sample rate was stable at approximately 100 Hz, well below our latency requirement. However, the key disadvantage to this method is that it was only able to track one robot. The sensors had to be elevated from the maze walls, and one robot had to be fitted with a marker to be made taller than the other robot to ensure reliable positioning. This would allow us to track Pac-Man’s progress through the maze, but not the Ghost robot. Because this solution would provide minimal functionality for the game to be playable, we developed this solution as a reliable backup plan.

### 3.6.3 Image Recognition

Image recognition based position tracking involves capturing successive images of the maze and robots, applying image processing algorithms to those images to find features of interest in the
images, and extracting position data based on the relative positions of those features. A Logitech G200 USB Webcam would be used to capture images of the maze. The image processing software would be developed using OpenCV, a multi-platform open source computer vision library. OpenCV simplifies the process of camera interfacing, image capturing, and data management, and implements many low-level mathematical functions needed for image processing.

Using OpenCV to develop our image recognition software allowed us to focus on the high-level recognition algorithm, rather than low-level data manipulation. The recognition software works by using mathematical transforms to search the image for predetermined shapes and patterns. The captured frames were analyzed to recognize the walls of the maze in order to determine the maze configuration, as well as to recognize the shapes of each robot. To analyze the frames, each color channel was extracted successively. The resulting image was then passed through thresholding and edge detection filters to find shapes in the images. These shapes were compared with the target shapes, and the centers of these shapes were located. The software then compared the relative positions of each shape to determine positions.

Because image processing is complex and computation intensive, it did not meet our simplicity requirements; and the latency requirement was stretched to the limit of 250 ms per stable output. However, image processing did meet our resolution requirement extremely well and allowed us to track both robots with great accuracy. Image processing also offered the greatest flexibility and capability to extend to more features such as aiding the robots in navigating the maze.

3.7 WIRELESS COMMUNICATION

Our system’s design required wirelessly passing data and command signals back and forth between the distinct modules of the system. The Wireless network was designed in layers, as shown in Figure 8 on the next page, between high-level logical message passing and the physical IEEE 802.15.4 Zigbee layer. This section details the design of each layer of the wireless network, from the bottom up.
The requirements for the Wireless network included a low data rate, low overhead interface, and low power consumption. The technologies we considered to implement the network included IEEE 802.15.4 Zigbee, IEEE 802.11 Wi-Fi, and IEEE 802.15.1-2005 Bluetooth. We chose Zigbee as the best alternative due to its simple interface and low power consumption. While Wi-Fi and Bluetooth offered higher data rates, they require a more complicated interface and significantly higher power consumption. Additionally, the higher data rates would not benefit our design.

The network’s physical transport layer was encapsulated in the XBee module and presented as serial data through an API. That data then flows through the Serial Port driver to the XBee packet library where it is processed and organized into data packets. Those data packets flow in and out of the messaging API, which transforms logical messages from the control program into XBee packets for communication to another node.

The software begins at the serial data level. The Xbee module outputs UART serial data, which consists of an asynchronous serial stream of one start bit, 8 data bits, and finally a stop bit. To implement the interface, we developed a Linux serial port library based on the standard Unix POSIX library. The module uses Unix system calls to read and write to device files that control
the physical port. The library provides basic functions to initialize, read data from, and write data
to the port. Figure 9 below is a listing of the software primitives for the POSIX library system
calls and the serial port library. The three different \texttt{Serial\_Read} functions implement differing
blocking or non-blocking read procedures.

\begin{verbatim}
/****** POSIX System Call Functions *******/
int open(const char *pathname, int flags);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);

/****** Serial Library Functions *******/
SerialPort Serial\_Open(char * path, int baud);
int Serial\_Write(int fd, char *buf, int bytes);
int Serial\_Read(int fd, char *buf);
int Serial\_ReadnX(SerialPort port, char * buf, int bytes);
int Serial\_Readn(SerialPort port, char * buf, int bytes);
\end{verbatim}

\textbf{Figure 9: POSIX System Call and Serial Port Library Software Primitives}

The XBee API library translates raw serial byte data into logical packets that format the data
flow. The software library packet processor places the incoming serial bytes into packet
structures to allow them to be used easily by the main program. Likewise, the send functions take
as input a data buffer, places the necessary packet header information around the data, and then
transmits the data to the XBee via the serial port.

The XBee library must accurately interpret incoming packets, as well as properly format
outgoing packets to ensure that the network operates properly. Additionally, the library internally
queues packets for sending and receiving; queue sizes must be optimized to limit wasted space
but allow enough space for packets to be queued to meet the needs of the main program and
speed of the data transfer.

The purpose of the wireless network is to pass instructions and data between the server, robots
and maze. To ensure that messages are passed efficiently and reliably, a message passing
protocol was needed on top of the XBee API protocol. Messages are held inside the data payload
section of the XBee packet. The messaging protocol was designed to allow the system to pass
any kind of message that would be needed, to be easily extendible as we added functionality to
the system, to require low processing overhead, and to protect the system from errant broadcast messages. Figure 10 below shows the message protocol structure.

Figure 10: Message Protocol Structure

To ensure that a received packet was in fact a valid message, not an errant message from an outside source, the first byte of the message is a predetermined packet prefix. The next byte is an opcode that indicates the purpose and meaning of the message and directs the receiver to the proper action to take after receiving the message. A common set of opcodes are defined throughout the network. Appendix A shows the set of opcodes defined in the message.h header file and shared throughout the network. The third message byte gives the length $N$ in bytes of the remainder of the message. The next $N$ bytes contain the message data, and finally a $0$ as the last byte indicates the end of the message. Inside the control programs of each node, the messages are interpreted by switch operations on the opcodes.

4.0 DESIGN IMPLEMENTATION

In theory, our solution will satisfy the requirements made in the problem design statement section; however, we encountered many implications in our implementation. The most important change since the beginning is the design of the maze. The maze and AI are scaled down due to
the size restriction and lack of funding for more robots. The following sections cover the implementation of the ghost AI, robot hardware, robot software, maze, position tracking, and wireless communication.

4.1 GHOST AI
Due to the switch from the arcade Pac-Man maze to our implementation of the Pac-Man II maze, the AI adjusted to match the appropriate maze. The ghost AI changed from a deterministic model to a non-deterministic model.

4.1.1 Deterministic Model
Initially, the ghost AI was based on the arcade Pac-Man maze. The ghosts would recognize chokepoints and wait there until it is safe to pursue Pac-Man further. The main high-level algorithm is included below:

```ruby
def trap_pacman
  0.upto(iteration.value) do |j|
    paths = find_chokepoints(pacman.location)
    ghosts.each do |g|
      p = paths.pop 0
      g.go p
    end
    while is_pacman_safe?
      bots_wait
    end
  end
end
```

This implementation is deterministic in that the outcome of the game is predictable. Pac-Man will almost always be captured before collecting all the food. In our implementation, we realized that there are neither power-pills nor portals in the game. In the original arcade game, Pac-Man had the power to disable a ghost and return it in ethereal form to the starting position. Furthermore, Pac-Man can also teleport instantly through a portal to the other side of the maze. There two features are physically impossible to implement in our robotic version of the game.
Unfortunately, we later realized that our implementation favors the ghost without these two abilities of Pac-Man. In this implementation, the ghost AI won almost every game in simulation. The only games that ended in stalemate are when the Pac-Man moved into a location with five escape chokepoints and stayed there. This implementation faced the problem of the number of bots versus the number of chokepoints. If there was a ghost for every chokepoint, then the game will always end in ghost victory. Otherwise, if there are more chokepoints than ghosts, the game ends in stalemate. This implementation is unacceptable due to the predictable outcome of the game. Predictable outcomes detract from the fun element of the game. Thus, we modified the game play to a non-deterministic model, where the outcome is not predictable. This model is more enjoyable for the user because the victory is based on chance.

4.1.2 Non-Deterministic Model
The key to a successful ghost AI is using a non-deterministic algorithm based on probability. In our adjusted solution, the maze has four paths each with a dead-end. The paths are shown below in Figure 11 on the next page.

![Figure 11: Pac-Man II Maze With Paths Shown](image)

Given this property in the maze, the game is determined by whether or not the ghost chooses the same path as Pac-Man. In order to make the algorithm non-deterministic, we add a randomizer function. The adjusted ghost AI algorithm is included below:
```python
def ghost_ai
    path = get_random 1,4,current
    if path.eql? 1
        go_path_1
    elsif path.eql? 2
        go_path_2
    elsif path.eql? 3
        go_path_3
    else
        go_path_4
    end
end
```

In this implementation, we randomize the paths that the ghost chooses. This randomness significantly increases the chances of Pac-Man winning. Due to the randomness nature, the ghost will more likely miss the Pac-Man for each path that is chosen. The ghost AI also relies partly on real-life issues such as calibration of the sensors and positional data. Since the sensors are not always accurate, the ghost AI also takes this issue into consideration. Both of these factors account for a moderate but significant percentage of times that the ghost loses. With all of these non-deterministic factors combined, the chances of Pac-Man winning are now significant enough for the game to be enjoyable.

4.1.3 AI Integration

After determining the AI commands for the ghost, there needs integration with the robot software. In other words, the AI and robot software must communicate using the same interface. The commands “go straight”, “turn left”, and “turn right” from the AI must have a physical meaning in the real world that can be accomplished by the robot software. This interface is split into two phases. The first phase of the integration is the ability for AI to choose whether to turn or go straight. The second phase of the integration is providing the capability for the robot to physically turn or go straight. These two integrations are documented in Appendix B in the software API. With this interface, the AI can effectively communicate with the robot’s movements directly.
4.2 ROBOT HARDWARE
Thanks to the UT Pharos Lab, we were able to check out some of the hardware including basic electronics components, standoffs of robot chassis, IR sensors, digital compasses, as well as remote control robots. We were provided with sufficient hardware resources and technical support throughout the semester. Since it required careful consideration before constructing and assembling our robots from scratch, the iRobot Create, which operated with a powerful x86 processor and high speed Wi-Fi connection, allowed us to test and implement our AI algorithm on a physical platform. Our custom robots were constructed using components including a motor and gearbox assembly, H-Bridge motor driver circuit, and a voltage regulator and DC-DC converter. Figure 12 on the next page shows our constructed Pac-Man and Ghost robots.

![Figure 12: Pac-Man (left) and Ghost (right) Robots](image)

Even though we have been quite considerate regarding our robot design, further improvement could be made to prevent accidental damage of these electronics and mechanical parts. Since there are so many modules mounted onto the robots, wires could be tangled easily. PCB could be built to avoid these kinds of problems.

4.2.1 Motor & Gearbox
There are many types of motors that could have been used to construct the drive system for our robots, including stepper, brushless, and high performance motors. The low-voltage, high performance DC motors would fit onto the Tamiya gearbox that allows for a compact design of the robot. It is low cost and consumes only 2.5Watt of power, while speed and torque could be controlled easily with the pulse width modulation duty cycle. However, the motor requires high
current draw, and may overheat when stalling. Figure 13 below shows the motor and gearbox assembly.

![Figure 13: Low Voltage, High-Performance D.C. Motor & Tamiya Twin Gear Box](image)

4.2.2 H-Bridge Motor Driver Circuit

The H-Bridge motor driver circuit allows a microcontroller to use pulse width modulation (PWM) to vary the voltage applied to the D.C. motor and thereby control the speed. The L293 integrated circuit is designed to separate high and low voltage and prevent high current to reach the microcontroller. For each motor, two separate pins are connected to one brush motor in parallel with a diode. This configuration could prevent back electromotive force from the motors from damaging the microcontroller. Each chip would support up to a maximum of 1A current draw. Manufacturers suggest that more current could be supplied by stacking the ICs on top of each other. The overall payload would be the sum of each current limitation.

The design of the controller circuitry is quite time consuming. This design implementation allows affordable solutions to our robot; however, it is also one of the obstacles our team faced to further construct more robots. Figure 14 below shows the L293 motor driver.

![Figure 14: H-Bridge Motor Driver – L293](image)
4.2.3 Voltage Regulator/DC-DC converter

The low-voltage motors require large current draw of up to 2000 mA. To optimize our power consumption design, we decided to use the DC-DC converter to achieve better efficiency. Refer to the DC-DC converter specification in Figure 15 on next page for the relationship between efficiency and output current. We were able to select both input and output voltage to achieve high performance. Vout is chosen to be 5V to support the input voltage of our microcontroller development board. Step-down conversion was considered because the overall current drawn is greater than 250mA. Therefore, Vin is picked to be 7.2 V by cascading 6 AA battery in series (within the range of 5V to 9V).

![Figure 15: Efficiency vs. Output Current](image)

4.2.4 C328 & Servo

The C328 camera module is integrated with a JPEG compression engine operating at maximum baud rate of 96.1k bps. It is mounted on a separate circuit board with a dedicated wireless XBee and a LM1117 voltage regulator, shown in Figure 16 on the next page. The entire module is mounted onto a servo that was controlled by the user during game play. For each image taken, bytes of image data are written into the solid state disk. The read and write operations are almost instantaneous upon snapshot.
4.3 CONTROL SYSTEM

The control system module of robot has a scheduler on the top of four to five major device drivers depending on the role of robots. Scheduler will switch between device drives and properly divides workload among drivers. A good control module should have a well balance scheduler and minimized latency of each device driver. The following subsection will describe the implementation of scheduler and each device driver. Figure 17 shows the overall structure of our control system.

**System Software**

![Diagram of Control System Structure](image_url)

**Figure 17: Control System Structure.**

**4.3.1 Scheduler**

As we mentioned above, the task of scheduler is to switch between the threads. A thread is an execution of program and in our project a thread represents an execution of device driver. Each thread has its own register file and program pointer. To successfully switching between threads without losing important information we set up thread control block (TCB) for each thread. A
TCB contains registers, process id, and execution counter belong to certain device driver. TCB could improve the performance of software module because scheduler only needs to change its runtime TCB pointer when switching from one thread to the other. An alternate design includes priority level for each thread. Since our control module contains less thread than normal operation system, we decide to not implement this feature. This design eased the debugging process during implementation. Figure 18 on the next page shows the interactions between run time pointer and TCB.

![Diagram showing TCB switch mechanism]

Figure 18: TCB Switch Mechanism.

4.3.2 I2C Bus Interfacing

I2C Bus is a protocol which defines a master-slave relationship among I2C device. In our project we set up microprocessor as a master and other IO device as slaves. A master is the device in charge of the bus and this device controls the clock and initiates START and STOP signals. Slaves will listen to the bus and wait for the commands from the master device. Once slaves receive a certain command they will act on control and data. Figure 19 on the next page shows
the topology of normal I2C bus structure. The rule of I2C is that master can send data to the slaves or receive data from them, but slaves do not have right to transmit data among themselves.

![I2C Bus Structure Diagram](image)

**Figure 19: The I2C Bus Structure**

In our implementation microprocessor will first sent out the hand shake signal which contains a 7 bits address and wait for the slave device to response with an ACK on that address. Also microprocessor will then sent out a clock signal on SCL line and one bits Read/Write command on SDA line. This could ensure digital compass is a functional mode. If digital compass experience any hardware hazard, microprocessor will be notified. This hand shake signal could prevent I2C device driver from an infinite loop, which will eventually cause control module crash. After responding to the hand shake signal, slave device will place data on the next clock cycle. Fugure20 below shows the sequence of data for I2C bus.

![I2C Data Sequence](image)

**Figure 20: The Data Sequence of I2C Bus.**

4.3.3 Sensor and Compass Data Handling Procedure

The main purpose of sensor and compass is to prevent collision and correct the heading of robot. However, sensors and compass could only provide distance and current heading of robot. We need a handling procedure to make a judgment that if microprocessor needs to modify the current robot motion. To ensure receiving a valid data from sensors and digital compass, this
handling procedure will always be scheduled right after microprocessor request data from sensors and compass. This algorithm is straight forward and we provide a flow chart in Figure 21 on the next page to demonstrate the decision making process.

**Handling Procedure Call**

1. **Does compass reading go over the desirable direction?**
   - NO
   - YES
     - Robot heading is deviated and decrease the PWM duty cycle by calling the PWM device driver

2. **Is the input data a IR sensors?**
   - NO
   - YES
     - Is IR sensor reading below the minimal distance?
       - NO
       - YES
         - Robot heading is deviated and decrease the PWM duty cycle by calling the PWM device driver

3. **No immediate collision and the robot is heading to the correct direction.**

**Figure 21: The Flow Chart of Handling Procedure.**

### 4.3.4 PWM Motor Control

The speed of DC motor is control through the voltage level of its two input pins. To drive the motor spinning toward certain direction we need to apply a voltage difference on those two pins. We could also drive motor to spin toward the opposite direction by reversing applied voltage level difference on pins. In order to gain a better control and voltage precision we implement a PWM waveform. Figure 22 on the next page shows an example of PWM waveform for motor control. The corresponding DC level will be the average voltage value over the period. Hence, adjusting the output voltage through PWM duty cycle will change the voltage level of DC motor.
We implemented PWM by initiate four dedicated I/O channels to control waveform. There are three actions required to form a dedicated PWM channel. Firstly, we initiate the channel polarity by sending 0x0A signal to the PWMPOL pin on the processor. Secondly, we select a clock signal and desired frequency for PWM waveform through PWMCLK and PWMPER pins. Finally, setting up PWMCAE, PWMCTL and PWMSCL pins will complete the set up actions. An alternative design choice to implement PWM is to use a traditional output compare interrupt to generate PWM through a general I/O pins. This design is easier to do and does not require additional knowledge about processor chip. However, output compare interrupt design occupies more processor’s execution time which could enlarge the latency of PWM device driver and has a negative impact on overall system performance.

### 4.3.5 Image Transmission

This device driver is used to transmit the image pixel from the webcam to the server. This device is only for PacMan robot because ghost does not have to send field image back to user. All of image pixel is stored into a temporary buffer. While transmission driver being called, it will look inside the buffer to see if any pixel needs to be transferred. Then it will call Xbee communication software and pass the memory location of pixel to it for transmission. Once Xbee notifies transmission driver the completion of transfer, the whole operation reach the end.
4.4 SENSORS AND DIGITAL COMPASS

To avoid collision, we used five GP2Y0A02YK IR sensors mounted on both the top and bottom plane of the robot. The Pharos lab provided us with the IR sensors and digital compass. These infrared sensors send range data to DP512 microcontroller on the ghost robot through ADC channels. The on-chip ADC will convert input voltage level to 10 bits digital data. Yu-Chieh Lee calibrated these sensors by converting 10 binary reading to corresponding distance.

The major difficulty encountered with this module occurred when we tried to set up the IR sensor position on the robot. Since our project built most of robot by ourselves, we did not have existing sensor platform or other mechanical support. A properly positioned IR sensor should be able to cover as much detecting distance as possible. At the same time all of IR sensor needs to prevent from blocking each other and being blocked by other device on the robot. When obstacles between the sensors and targets interfered with the IR sensor signal, sensor readings were not correct. The basic approach we adopted is try and error with different positions on the robot. After a series of testing we decide to put two IR sensors with 45 degree angle across the front part of robot. Since robot set up its motion toward its front side, using a 45 degree could give robot more time to response when obstacle detected. On the other hand, two more IR sensor position on the rear of robot with 90 degree angle. Because the obstacles on the back of robot are usually closer to sides of robot, using 90 degree is more practical.

For the robot heading correction, we implemented a digital compass to provide current direction of robot. By comparing the current heading with the desirable heading, we could ensure our robot always moves to the right path. The compass represents the direction with 32 bits binary data range from 0 to 3600. Those data are transmitted through I2C bus to the microprocessor. Since the ENS is not perfectly pointing to north, it requires calibration of offset value in order to instruct robot our desirable heading.

The only difficulty of compass we encountered is it sensitivity. Most of digital electronic compass use a magneto-inductive technology. The magneto-inductive technology is able to electronically sense the difference in the earth's magnetic field from a disturbance caused by external elements such as Ferro-magnetic materials and the magnetic field generated by
automobile electrical systems. Therefore, any adjacent electro-magnetic noise will cause errors. We put the compass relatively higher than other devices especially DC motor to avoid this issue.

4.5 MAZE PLAYING FIELD

This initial maze design works properly with CD size robots equipped with short-range infrared (IR) sensors. Si completed constructing the four feet by eight feet wooden platform and began testing the robots on the field; however, the performance of the maze and the robots were unsatisfactory due to the long minimum range of the robots’ IR sensors. The team decided to redesign the maze to a size of eight feet by eight feet, with a channel width of approximately two feet and wall height of about six inches. The actual implementation of the maze is shown in Figure 23 below. In this case, the maze is divided into a four by four grid and has 16 LEDs in total. The maze circuit diagram can be found in Appendix C.

![Maze Playing Field Diagram](image)

**Figure 23: The Actual Implementation Of The Maze Playing Field.**

There are three major reasons for the changes to the design. First, the channels are too narrow to guarantee accurate output from the IR sensors. According to the data sheet of IR sensor, the minimum measuring distance range is about 20 cm to 150 cm; however in our experimental testing, the IR sensors minimum accurate range was approximately 30 cm. As a result, we had to widen the maze channels to ensure the functionality of the IR sensors. Second, the height of the
robot increased significantly during the semester. Due to the space limitation of the robot chassis, Lee and Gary moved the IR sensors to a higher position than they planned. The height of the maze had to be changed accordingly such that the IR sensors would not read the empty space above the walls. Third, David’s new AI algorithm required the maze to have dead ends and a close number of nodes at every path.

In addition to the changes to the physical layout, the circuit and control program were also modified. Because of the maze redesign, there were only half as many LEDs on the maze as originally planned. While the circuit became smaller and simpler, the concept and mechanism of the LED display circuit did not change. However, because the maze was expanded, the LEDs were not bright enough and became difficult to see. To improve their visibility, Si decided to make the LEDs blink at approximately 2 Hz, rather than remaining powered on to represent the presence of food in a square. The control program was modified to implement this change. Instead of refreshing the LEDs at every LED cycle, Si alternated refreshing and powering off. This alternating will also reinforce the correct on or off status of each LED, allowing for all LEDs and circuit elements to fully discharge between powered on periods. This discharging prevents LEDs that should be off from appearing to be slightly on.

4.6 POSITION TRACKING
Our team was successful in developing an image processing based position tracking system to track the location of the robots in the maze. A USB webcam was hung from the ceiling over the maze and connected to a laptop running our OpenCV based image processing software. In the ideal solution, this system would have been able to identify the image of each robot and the shape of the maze, then map the robots’ location accordingly; however, we weren’t able to fully implement this detailed level of image recognition. To aid the image processing software in recognizing the robots, we used marker shapes, a triangle on the ghost robot and a grid of 4 squares on Pac-Man, mounted above each robot’s body. Appendix D shows close-up images of the each robot with marker shapes mounted. The software easily, reliably, and efficiently detected these large, simple shapes. Not only did this design choice simplify the design of the detection algorithm, it also improved speed and reliability by reducing both the amount of computation involved and the possibility for error.
The detection algorithm worked by first extracting each color channel from the image, then using successively higher threshold levels and edge detection filters to identify lines in the image. These lines were then connected to find shapes, primarily triangles and rectangles. The corners of each feature on the maze were then used to map the maze, and the vertex coordinates of each triangle and square were averaged to find the real centers of each shape. To prevent the error introduced by visual noise, the results were filtered through a three sample median filter to remove outliers. The output of 10 frames was then averaged together to produce a final, stable output result.

After extensively testing and calibrating the threshold values and shape sizes used in the software, the system proved to be extremely accurate and reliable, able to pinpoint the center of each robot in nearly every frame. The median and average digital filters were able to stabilize the output against visual noise. Figure 24 on the next page shows a sample output of the recognition software. The green highlights on the triangles and squares in the image indicate the shapes that the software has detected, and the yellow markers indicate the filtered center of the object. The red points indicate corners of the maze. Four of these points represent each grid square, and a simple x and y comparison was used to determine the grid point of each robot.
Figure 24: Image Recognition System Sample Output

4.7 WIRELESS COMMUNICATION

The wireless network was successfully implemented nearly exactly as originally designed, except for the XBee packet processor for the MC9s12 microcontroller. The final software that was used on the Linux server worked exactly as designed, and while the same software worked properly on the microcontrollers, the processing was burdened by unbuffered serial input and output. While integrating the XBee API into the control program on the microcontroller, the group determined that the calling procedure would stall the robot’s control program for unacceptably long periods of up to 100 ms.

This problem was solved by redesigning the packet processor and sending functions. The original data flow required the main program to call the packet processor function, which called the Serial_Read function to read the data from the port, and finally for the main to call the receive packet function to dequeue a packet. This calling paradigm worked well on the multi-threaded and pre-buffered Linux server, but was extremely overhead intensive with unbuffered
data on the 9s12. Figure 25 below shows a flow chart of the original and redesigned calling schemes.

![Flow Chart]

Figure 25: Original (top) and Interrupt-based Background XBee Library (bottom) Calling Scheme

To solve this problem, we redesigned the packet processor to be non-atomic and interrupt based. Rather than waiting for all serial data to be available before processing the data, the data was fed into the packet processor one byte at a time and based on a serial port receive interrupt. The redesigned send functions were also buffered and interrupt based. On a send operation, a pointer to the packet buffer was enqueued to the send queue. When the transmitter was free, an interrupt would be generated and the next byte would be transmitted. These design changes allowed the network software to run entirely in the background and run much more efficiently.

5.0 TEST AND EVALUATION

After implementation of the modules, we tested the modules and then the entire system. The purpose of testing and evaluating each component first is to determine the quality, but also it is
used to determine the variation in behavior. For example, the robot’s motors may torque at slightly angular velocities. This error is evaluated so that the robot’s software may account for this variance. Furthermore, the sensors’ read variance is also tested for in order to properly write the software accommodation. The position tracking needs to test for accuracy. The wireless needs to be secure and efficient. This section covers the testing and evaluation for each of the modules: AI, control system, sensors, maze, wireless, and position tracking.

5.1 ARTIFICIAL INTELLIGENCE
The AI testing is performed last due to the dependencies on every other module. In short, the AI testing is the equivalent of system testing. The AI utilizes the robots, position tracking, maze, and wireless modules. The AI testing is performed by statistically measuring the results of the game outcomes. We played the game 30 times using the AI and recorded the win-loss ratio for Pac-Man. The reason we chose 30 is to have a sufficient sample size in order to perform statistical calculations shown in Appendix E. From the experiment, the ghost won 18 times and the Pac-Man won 12 times. Upon further analysis, we realized that in certain games the sensor values are off-alignment. When the IR sensor readings were wrong, the ghost crashed into the wall. When the compass sensors were erroneous, the ghost over-turned. The last factor is the ability for a random user to control Pac-Man successfully, which was a significant contribution to ghost victory.

Although the win ratio slightly favors the ghost, the system performs overall balanced. The criterion for a successful system is for one side to not win 100% of the time, and neither side did. From our defined requirements, the system met its standards and is a success.

5.2 ROBOTIC HARDWARE
To find the overall run time of each robot, we performed the worse case analysis. Individual hardware components on the robots were powered through the bench power supply. This would allow us to estimate maximum current drawn. We estimated the values to be as follow:

\[
C328 \text{ Imax} = 150 \text{ mA}, \text{ Servo Imax} = 100 \text{ mA}, \text{ XBee Imax} = 200 \text{ mA}
\]

Knowing the efficiency of the DC-DC converter and voltage regulator, we could calculate the sum of current drawn from the battery.
The overall estimated run time for Pacman is 0.93 hours whereas for ghost is 0.5 hours.

5.3 CONTROL SYSTEM

The nature of control system is a combination of several device drivers. Hence, the best way to test control system is to run a field trip in the maze to observe the behavior of robot. For the ghost robot we wrote a test program, which will drive ghost robot to cruise around the maze. During the initial testing phase our system was not well balance because the compass direction and PWM duty cycle unit needed to be re-calibrated. And the ghost ran into the wall quite often because the physical position of IR sensors had been changed. After days of re-calibration the ghost robot had finished the testing trip smoothly.

To test if the PacMan received each command correctly, Gary used a GUI written in Matlab and pressed the push button for each command. If the PacMan received the command, it would perform a correct action. All the expected controller functions were tested successfully, with the exception of mapping, as mentioned earlier. When the “Move Forward” or “Move Back” commands were pressed, the DC motor will start to drive robot forward or back. When the “Webcam” button was pressed, an image from the car’s webcam would correctly be displayed. The “Webcam” command required the most rigorous testing – continuously pressing the push button and verifying a correct image was displayed – due to the potential bandwidth and memory limitation. Finally, the car would move in the corresponding direction, when corresponding movement button was pressed

5.4 IR SENSORS AND DIGITAL COMPASS

IR sensors are the basic elements in our sensor network. To ensure each IR sensor does not have any defects, we placed an object in front of sensors and use a voltage meter to probe their output pin. The expected voltage level should be around 4.3 to 5.1 voltage when the object is 20 centimeter away from the sensors. In reality, the voltage reading of same distances will not be the same. The result of testing shows that sensors work properly from 27cm to 40cm which is shorter than the specification from manufacturer. For the purpose of this project this range is sufficient for our project. On the other hand, compass reading requires I2C bus. Therefore we
tested compass by sending hand shake signal through hyper-terminal channel on the PC. As soon as terminal monitor receives a response from compass, we could verify that device is usable. After testing all of components were functional.

A complete sensor/compass module has a physical sensor and a software interface. We need an interface to process analog data from the field. The common interface for a sensor is accomplished by an analog to digital converter and a software driver. Since every sensor will use the same interface driver and bus channel on their chip, our testing will emphasis on the bandwidth and signal integrity. There are four sensors on the sides of a robot and on in the front. We tested them by driving the robot in the maze and collect distance reading from both sides and front. When the robot is moving in the middle of a channel, the sensor reading should be close and this means the signal data retained its integrity throughout the process. Those data were stored into a text file for us to determine bandwidth. On the other hand, we tested compass module by loading a testing program on microprocessor and the testing program will output the compass reading data to the LCD. Then we changed the heading direction of robot and compared the reading. Since the reading difference between heading are continues and reasonable, we concluded that compass module is functional.

5.5 MAZE PLAYING FIELD
The testing and evaluation of the maze consists of two parts: reliability and functionality. To test the reliability of the maze, Si kept the LEDs blinking for approximately 5 hours. All the components worked correctly throughout and after these 5 hours. The success of this test implies the stability, reliability and durability of the maze circuit.

To test whether the maze is functional or not, Si performed three tests. First, Si tested if the maze received and reacted properly to the signals from the server with the help of Aaron. They implemented an LCD on the maze to display the signal the maze receives from the server. If the maze received the signal correctly, then the LCD will display the same messages that they sent. All messages were received correctly. Second, Si tested whether each LED can be controlled independently. Si ran a test program to turn off one LED at a time, as well as change the LEDs according to random patterns. The control program performed perfectly in these tests. Third, Si
tested the combine system by sending out commands from the server. If the combine system worked well, the LEDs on the maze should be changed according to those commands. In Si’s tests, the maze responded correctly to all the server commands with a delay approximately one half second.

5.6 POSITION TRACKING
Testing the implementation of our position tracking system involved first testing the functionality of the image processing software, then evaluating the accuracy of the output data and performance of the system. Because of the complexity of image processing, we tested the system step-by-step in parallel with development. The software was developed incrementally; we tested each new feature for functional correctness and reliability before the next feature was added. After we had completed implementing the overall algorithm, calibrating the optimal software thresholds and camera viewing-angle required many iterations of real-time trial-and-error testing. To calibrate the thresholds, we moved marker shapes around the maze while observing the program output to make sure that the shapes were recognized at every point in the maze and at any orientation.

To evaluate the implementation we had to ensure that the final output of the tracking system provided a continuous and accurate representation of the position of each robot. To test the accuracy of the system we moved the image markers extensively around the maze and compared the output to ensure that the correct grid square was found, making sure that we covered the entirety of the maze. We then mounted the markers on the robots and allowed them to navigate the entire maze for several long tests up to the robots’ maximum battery life. To easily determine the system output for these tests, the maze LEDs were turned on only for the grid space in which we placed the marker.

During development and testing of the position tracking system, complications including ceiling height and movement in the image periphery. Because of the limited ceiling height above the maze, the camera was suspended offset from the center of the maze, the ideal location, and angled toward the maze. This skewed the image, with object at the far end of the maze appearing
smaller. This skew complicated calibration of the software thresholds as different levels were optimal for different sections of the image; to overcome this difficulty, the tolerated range was widened. Additionally, people moving around the maze at the periphery of the image caused unpredictable visual noise that occasionally skewed results. To solve this noise problem, as well as noise introduced by wider threshold ranges, we used digital median and averaging filters to stabilize the results.

5.7 WIRELESS COMMUNICATION
To functionally test the wireless network, each level in the hierarchy was tested during the development process. The serial library was tested by connecting two computers together and transmitting several kilobytes of data in each direction. The input and output were compared to ensure that there were no communication errors.

The XBee API library was tested in the same fashion, communicating and comparing several kilobytes of data through 100 byte XBee packets. This communication was tested both between two computers, and between a computer and a 9s12 microcontroller. The message protocol was tested by sending messages of each type to the nodes in the system and ensuring that the proper action was taken or response received.

To evaluate the performance of the network, we carried out tests similar to the functional test. Kilobyte sized transfers were repeated for hour-long test periods to both stress test the system, and measure its latency and bandwidth capabilities. The latency of each packet was recorded and averaged, as well as the total transfer and return time for the entire stream. We found that the system was capable of sending and receiving continuously for hour-long periods without failure and that messages of size approximately 100 bytes in length took on average between 300 and 1000 microseconds to be acknowledged.

6.0 TIME AND COST CONSIDERATIONS

Our final robotic Pac-Man system was successfully implemented within the scheduled time constraint and exceeding the budget presented in our Design Implementation Plan by $239.
Although we modified our project and revised the Gantt chart several times, we were still able to complete the prototype on schedule. The robot hardware and calibration contributed most of the unexpected setbacks to our project schedule. This period took approximately three weeks longer than planned. Implementing the maze and position tracking system involved significant trial-and-error experimentation, which added more delay to our schedule. To compensate for these unexpected setbacks, each term member worked extra time on our project every week. We also reduce the number of robots to only two, as well as reduced the projected testing period in order to meet our time constraints.

Our project did go over budget by $239. Despite the many equipment loans from the Pharos Lab, such as infrared range and compass sensors, the trial and error of the maze and position tracking system wasted a lot of money. Besides this, we burned our camera by short circuiting the input voltage and ground wires. We also did not anticipate needing some components such as the XBee Explorer Serial boards. Appendix F includes a detailed final cost table listing all prototype components, including those ordered out-of-pocket and those donated.

7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN

Our project followed the ethical standards of professional engineering practice, such as ensuring human safety, and giving credit to ideas, designs, and code that we used. However, there were still some safety, ethical, and environmental issues that related to our project.

Our project has three major safety issues. First, the robots’ motors may overheat if the wheels become stuck. This may cause fire and burn the robots. Second, there is a chance that the power source will burn and cause fire because we didn’t add any fuse or overload protection for it. Third, there was a lot of open wiring and exposed high current circuitry, such as H-bridges and voltage regulators, which may increase the risk of getting shocked.

The main concern of the ethical aspect is the privacy issue of using a camera in a public place. We addressed this issue by keeping the camera facing the maze only and not recording anything from the camera. Since the camera is placed in the public area without recording, we believe the
issue was resolved. There is also some other concern on the environmental aspect, such as the disposal of batteries. We made sure the batteries will be recycled or disposed of in a safe way.

8.0 RECOMMENDATIONS

Our team has many suggestions for anyone looking to repeat or continue our work, including improvements to both the hardware and software implementations in our system. We believe our system level approach is appropriate, and recommend keeping the same module structure while making modifications inside those modules. In the following sections, we will discuss how we would approach each module differently.

8.1 ARTIFICIAL INTELLIGENCE
The Artificial Intelligence (AI) can be improved by adjusting the randomizer function to stabilize the probability of ghost victory to 50%. This can be done by collecting a larger sample size of game data and continuously training the randomizer function to stabilize at the equilibrium point. If a future AI developer wanted to create a more challenging algorithm to capture Pac-Man, he or she might consider adding at least one more ghost robot to the game. An AI that collaborates all ghost robots to hunt down Pac-Man will add a lot of fun to the game.

8.2 ROBOT HARDWARE
If a future engineer recreated our robot, he or she might consider not building everything completely from scratch. Though some components, such as voltage regulator, are simple and cost far less to build from scratch, some components require tremendous time and effort to build. The most obvious example is the H-bridge motor driver circuit, a very complicated and dangerous circuit. Gary spent a week to build the first H-bridge and get everything working. When he worked on the second one, he burned the 90% completed circuit in the fifth day and needed to start all over again. A complete H-bridge IC chip costs approximately $15, so we believe that buying an IC chip is worth the expense and saved time and effort.

Another recommendation for a better robot is to use high quality motors with precise feedback. We used DC brush motors in our robots; this is a cheap type of motor. Though they are easy to
interface, they are difficult to control and measure precisely. A higher quality motor with built in sensors may cost more than the DC motors, but it will save a lot of time for calibration.

Finally choose IR sensors with the appropriate range. Short range sensors are much better than long range when implementing this project. Because the IR sensors we obtained from the Pharos Lab were not short enough, we had to enlarge the channels and reconstruct the maze to ensure the walls are in the operating range of the IR sensors.

8.3 ROBOT SOFTWARE
The robot software could be improved by adding median filters to the sensor signals. In our current system, we did not filter the sensor output with any kind of sensors. We believed this is one reason why the robots malfunctioned in some situations. The reading of the IR sensors was unstable and periodic spikes caused erratic robot behavior. By adding a median filter, which takes three consecutive values and returns the median value, the IR sensor reading becomes very stable. Similarly for the compass sensors, the median filter would increase the reliability and robustness of our system.

8.4 MAZE PLAYING FIELD
To improve the maze, we recommend reducing the size and increasing the complexity. If the sensors and the position tracking are not issues, the more paths the maze has the more entertaining the game will be. By reducing the size, the maze would be more portable and more changeable. Moreover, instead of using only one small LED per grid, we recommend using bigger and brighter LEDs. This makes the food more visible and causes less confusion. We also recommend designing a maze without dead ends.

8.5 POSITION TRACKING
Future developers should extend the usage of image processing and position tracking. In our project, position tracking was only used to control the LEDs on the maze. However, we recommend extending the position tracking to track not only the location of the robots, but also the direction and the speed of the robots. By doing this, we can use the server to control the
robots and set up AI on the server instead of the robot. With the help of the large computational resource on the server, we can possibly implement a better AI. Moreover, we can also use the server to help the collaboration when we have two or more robots.

8.6 WIRELESS COMMUNICATION
One future improvement in the Zigbee module is the ability to block incoming buffer smash attack. The encryption and decryption algorithm can be improved by recognizing, deflecting, and preventing overflow injections.

9.0 CONCLUSIONS
Our team successfully completed implementation of a prototype of our system on schedule, delivering a functional and playable game system in time for the open house on November 23. The system met nearly all major design requirements, including a human controllable Pac-Man robot, an autonomous Ghost robot capable of navigating the maze and chasing Pac-Man, and an interactive maze allowing Pac-Man to “eat.” This prototype does have room for improvement, and the team learned many lessons through which the project can be improved on a second attempt. The project could be most improved through better selection of hardware and mechanical components and the application of image recognition to robotic control. Unfortunately, cost and time constraints prevented the construction of more than two robots, but the success of the two completed robots indicates that there is clear potential upon revisit to complete an entire set of robots. We hope that the project will be revisited in the future and look forward to our recommendations being implemented towards improving the project on a second attempt.
REFERENCES


APPENDIX A – MESSAGE PASSING PROTOCOL DEFINITION

message.h Header File

/* rev 1.4.1122.1645.A

message.h
Define message passing protocol

Created: October 22, 2010, 15:00 - AML
Modified: November 22, 2010 16:45 - AML

Created for EE 464 Project Pac-Man
*/

/*Message passing scheme:
{ PACKET_PREFIX } {opcode} {data_length N} {data0} {data1} ... {dataN-1} {0}
*/

#define PACKET_PREFIX 0xA5

//Network Control
#define OPCODE_NET_JOIN                           0x01
#define OPCODE_NET_REASSIGN_ADDRESS               0x02
#define NET_ID_SERVER   0xFA
#define NET_ID_PACMAN   0xFB
#define NET_ID_GHOST1   0xFC
#define NET_ID_GHOST2   0xFD
#define NET_ID_MAZE     0xFE

//Debugging 0x8{...}
#define OPCODE_CHARACTER_MESSSAGE 0xD0  //Should be treated as a printf

//Robot Control 0x1{...}
#define OPCODE_START                              0x10
#define OPCODE_STOP                               0x11
#define OPCODE_PAUSE                              0x12
#define OPCODE_DISABLE_MOTORS                     0x13
#define OPCODE_ENABLE_MOTORS                      0x14
#define OPCODE_MOVE                               0x15
#define OPCODE_SET_MOTOR_POWER                    0x16

//Robot Calibration
#define OPCODE_GET_FULL_MOTOR_POWER               0x17
#define OPCODE_SET_FULL_MOTOR_POWER               0x18
#define OPCODE_GET_CALIBRATION_COEFF              0x19
#define OPCODE_SET_CALIBRATION_COEFF              0x1A

//Data 0x5{...}
#define OPCODE_DATA_STATUS                        0x50
#define OPCODE_DATA_STATUS_LONG                   0x51
#define OPCODE_DATA_SENSOR_REQUEST_IR             0x55
#define OPCODE_DATA_SENSOR_REQUEST_HEADING        0x56
#define OPCODE_DATA_SET_MAZE_REF_HEADING          0x57
#define OPCODE_DATA_RFID_TAG                      0x58

//Maze Control 0xC{...}
#define OPCODE_MAZE_STATUS                        0xC0
#define OPCODE_MAZE_SET_LEDS                      0xC1
#define OPCODE_MAZE_SET_LEDS_SPEC                 0xC3
#define OPCODE_MAZE_SET_LEDS_SPEC                 0xC4
APPENDIX B – API DOCUMENTATION

Phase I – Interface between AI and robotic control software

---

**void face_north()**

**Functionality:** this function turns the robot north no matter which direction it is currently facing.

**Input Parameters:** None

**Return Values:** None

---

**void face_south()**

**Functionality:** this function turns the robot south no matter which direction it is currently facing.

**Input Parameters:** None

**Return Values:** None

---

**void face_east()**

**Functionality:** this function turns the robot east no matter which direction it is currently facing.

**Input Parameters:** None

**Return Values:** None

---

**void face_west()**

**Functionality:** this function turns the robot west no matter which direction it is currently facing.

**Input Parameters:** None

**Return Values:** None
void bot_rotate_left()

Functionality: This function turns the robot left 90 degrees based on the compass values. There is also a compass variation value taken into account here for purposes of calibration.

Input Parameters: None
Return Values: None

void bot_rotate_right()

Functionality: This function turns the robot right 90 degrees based on the compass values. There is also a compass variation value taken into account here for purposes of calibration.

Input Parameters: None
Return Values: None

void bot_go_straight()

Functionality: This function moves the robot forwards until the front IR sensors read the wall.

Input Parameters: None
Return Values: None
APPENDIX C – MAZE CIRCUIT DIAGRAM
APPENDIX C – MAZE CIRCUIT DIAGRAM
APPENDIX D – IMAGE RECOGNITION MARKER SHAPES
APPENDIX D – IMAGE RECOGNITION MARKER SHAPES

Pac-Man Robot: Square Grid

Ghost Robot: Triangle
APPENDIX E – GAME STATISTICS
## APPENDIX E – GAME STATISTICS

### Table E-1: Game Statistics

<table>
<thead>
<tr>
<th>Game Number</th>
<th>Ghost AI Win?</th>
<th>Pac-Man Hard to control</th>
<th>Ghost improper turn</th>
<th>Ghost crashed wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>18</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>
APPENDIX F – PROJECTED COMPONENTS AND PROTOTYPE COSTS
## APPENDIX F – PROJECTED COMPONENTS AND PROTOTYPE COSTS

Table F-1. Projected Component and Prototype Costs

<table>
<thead>
<tr>
<th>Name of Components</th>
<th>#</th>
<th>Cost per Unit</th>
<th>Out of pocket</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pegboard</td>
<td>2</td>
<td>$17.00</td>
<td>$24.00</td>
<td>$24.00</td>
</tr>
<tr>
<td>Wood blocks</td>
<td>20</td>
<td>$2.00</td>
<td>$40.00</td>
<td>$40.00</td>
</tr>
<tr>
<td>Nail gun</td>
<td>1</td>
<td>$79.00</td>
<td>$79.00</td>
<td>$79.00</td>
</tr>
<tr>
<td>Supper glue</td>
<td>1</td>
<td>$3.00</td>
<td>$3.00</td>
<td>$3.00</td>
</tr>
<tr>
<td>IR sensors</td>
<td>7</td>
<td>$20.00</td>
<td>$0.00</td>
<td>$140.00</td>
</tr>
<tr>
<td>Pinewood</td>
<td>4</td>
<td>$5.00</td>
<td>$20.00</td>
<td>$20.00</td>
</tr>
<tr>
<td>Compass sensor</td>
<td>2</td>
<td>$30.00</td>
<td>$0.00</td>
<td>$60.00</td>
</tr>
<tr>
<td>Zigbee</td>
<td>7</td>
<td>$45.00</td>
<td>$315.00</td>
<td>$315.00</td>
</tr>
<tr>
<td>RFID</td>
<td>100</td>
<td>$0.50</td>
<td>$50.00</td>
<td>$50.00</td>
</tr>
<tr>
<td>Motor</td>
<td>3</td>
<td>$5.00</td>
<td>$15.00</td>
<td>$15.00</td>
</tr>
<tr>
<td>Gearbox</td>
<td>3</td>
<td>$5.00</td>
<td>$15.00</td>
<td>$15.00</td>
</tr>
<tr>
<td>Resistance</td>
<td>200</td>
<td>$0.20</td>
<td>$40.00</td>
<td>$40.00</td>
</tr>
<tr>
<td>LED</td>
<td>100</td>
<td>$0.30</td>
<td>$30.00</td>
<td>$30.00</td>
</tr>
<tr>
<td>Wire</td>
<td>5</td>
<td>$9.99</td>
<td>$49.95</td>
<td>$49.95</td>
</tr>
<tr>
<td>TIP120</td>
<td>5</td>
<td>$2.00</td>
<td>$0.00</td>
<td>$10.00</td>
</tr>
<tr>
<td>TIP125</td>
<td>8</td>
<td>$2.00</td>
<td>$0.00</td>
<td>$16.00</td>
</tr>
<tr>
<td>CMOS Camera</td>
<td>1</td>
<td>$19.95</td>
<td>$19.95</td>
<td>$19.95</td>
</tr>
<tr>
<td>Monitor</td>
<td>1</td>
<td>$85.00</td>
<td>$0.00</td>
<td>$85.00</td>
</tr>
<tr>
<td>Dell PC</td>
<td>1</td>
<td>$400.00</td>
<td>$0.00</td>
<td>$400.00</td>
</tr>
<tr>
<td>Wheel</td>
<td>6</td>
<td>$3.00</td>
<td>$18.00</td>
<td>$18.00</td>
</tr>
<tr>
<td>Batteries</td>
<td>20</td>
<td>$3.00</td>
<td>$60.00</td>
<td>$60.00</td>
</tr>
<tr>
<td>Battery charger</td>
<td>2</td>
<td>$30.00</td>
<td>$60.00</td>
<td>$60.00</td>
</tr>
<tr>
<td>Construction paper</td>
<td>1</td>
<td>$12.00</td>
<td>$12.00</td>
<td>$12.00</td>
</tr>
<tr>
<td>Computer power supply</td>
<td>1</td>
<td>$50.00</td>
<td>$50.00</td>
<td>$50.00</td>
</tr>
<tr>
<td>Standout</td>
<td>40</td>
<td>$0.50</td>
<td>$0.00</td>
<td>$20.00</td>
</tr>
<tr>
<td>Freescale 9S12DP512</td>
<td>2</td>
<td>$130.00</td>
<td>$0.00</td>
<td>$260.00</td>
</tr>
<tr>
<td>ARM MCBSTM32</td>
<td>1</td>
<td>$81.00</td>
<td>$81.00</td>
<td>$81.00</td>
</tr>
<tr>
<td>XBee Explorer Serial board</td>
<td>3</td>
<td>$30.00</td>
<td>$90.00</td>
<td>$90.00</td>
</tr>
<tr>
<td>Webcam</td>
<td>1</td>
<td>$30.00</td>
<td>$0.00</td>
<td>$30.00</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
<td>$1,071.90</td>
<td>$2,092.90</td>
</tr>
</tbody>
</table>