

shift

an exercise in robotics and game design

Jason Wishnov

University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligent Machines Design Laboratory

TA: Mike Pridgen
Adam Barnett
Sara Keen

Instructors: Dr. A. Antonio Arroyo
Dr. Eric M. Schwartz

Table of Contents

I. Abstract	3
II. Executive Summary	4
III. Introduction	5
IV. Game Overview	6
V. Platform Overview	7
VI. Mobile Platform	8
VII. Actuation	9
VIII. Sensors	10
IX. Behaviors	12
X. Experimental Layout	13
XI. Results	14
XII. Conclusion	15
XIII. Documentation	15
XIV. Appendix A (Program Code)	16
XV. Appendix B (Media)	27

I. Abstract

Shift is a two-player game involving tactical and strategic play. Players regularly draw cards and place them onto a grid-based board, which will issue specific directions to a simple robot. Each player will attempt to guide the robot into his or her goal, which will win the game. The robot will use line-tracking, barcode reading, and several other sensors to obtain necessary information about the field of play and autonomously navigate without human input. A simple ATmega 128 chip and MAVRIC IIB board are used for primary control of the robot, which features a very simple mechanical design.

II. Executive Summary

Shift's gameplay allows for strategic depth while maintaining a set of rules that is easy to understand. The game consists of two goals, one for each player; the object of the game is to guide the robot to your goal. The first player to achieve this is the winner. Each player maintains a hand of four cards; these may be turn, reverse, erase, or run cards, which may also include active cards, which may be played while the robot is running.

Each turn consists of two actions: either the playing of a card, or the discarding of one. Active cards may not be played during a regular turn, just as passive cards may not be played during the robot's operation.

The robot itself is quite simple, merely needing to traverse the board whilst following instructions on the cards. Line-following algorithms ensure that it remains on course, the simple, interrupt-driven UART communication between the microcontroller and barcode reader act as a simple yet effective method to communicate. A bump switch, for beginning and ending the runs, is also utilized.

Actuation is accomplished via two simple hacked servos, modified for continuous motion. Driver code was written for each servo to ensure ease of programming and use. Proper control of these servos is crucial, due to the precise motions necessary on the field of play.

Power is obtained via two battery packs of six, rechargeable double-A batteries each, bestowing a combined voltage of 8 volts to the microcontroller and 8 volts to the servos. With standard play, the robot may be run for approximately seven hours before requiring a recharge.

The robot is constructed primarily from acrylic, which standard-size bolts and nuts acting as structural support. Velcro, epoxy, superglue, angle brackets, and spacers were also used to ensure a stable and durable design.

In essence, Shift's gameplay and construction are simple enough to replicate on a wide scale without difficulty. The game is rewarding and not overly complicated, encouraging tactics and forward thinking to encourage victory. A small yet significant element of chance is also introduced via the cards drawn from the deck. Hopefully, the game is a success, and will be enjoyed by many people in the coming months.

III. Introduction

Shift is a project that merges the fields of robotics and strategic game design. The robot is used as a literal medium of gameplay, by which the objective is to lead the robot to a player's goal using a series of cards. The cards are placed by the player onto the game board, through which the robot will autonomously navigate. Game rules can be found in Section IV.

The robot features numerous sensors to obtain necessary information about the game environment, while featuring a sturdy and robust mechanical platform. Sensors include a line-tracking module, bump switches, and a barcode reader. Platform design is explored in Sections V-XII.

IV. Game Design

From a deck of thirty-two cards, each player draws an initial hand of four. Each turn, players may place two card on the game board in a legal position (both goals and adjacent squares are illegal), or if they have a “Run” card, they may play it to run the robot. The goal of the game is to lead the robot into your goal before the other player can do so.

After placing a card, each player must draw two to maintain a total of four cards. If all cards are depleted, the robot will be run one final time. If this final run does not produce a winner, the game will end in a draw. A “run” is over when the robot either hits a wall face-on, or an infinite loop is encountered.

Cards will include six “Run” cards, seven erase cards (two active), four reverse cards (one active), and fifteen generic turn cards (four active). All cards may be placed in any orientation desired by the player. Active cards are denoted by purple triangles around the edge of card, and may only be playing while the robot is running.

If the robot encounters a turn card on a side that does not directly tell it to turn, it will continue straight as if there were no card. This also applied to misaligned reverse cards.

If players touch the robot, it is considered an automatic forfeit. It is therefore risky to play active cards when the robot is getting too close, as an immediate loss is imminent.

V. Platform Overview

In Shift, the robot's task is to follow straight lines laid on the basic game board, as well as obey any instructions found on cards it encounters. A button on the top of the robot will act as a starting signal. From that point on, the robot will follow the lines laid out on the grid in a straight manner, and will continue until 1) A card is reached, 2) A wall is hit, or 3) An infinite loop is encountered. These three actions each require a different sensor.

Basic turn cards utilize both line following and barcode reading, to ensure that a turn is accurate and quick. Run cards and blank cards, of course, require no sensing. Reverse cards will be obeyed via the barcode system, described below.

Bump sensors will detect if a wall or obstacle has been reached. Here, behavior is simple: the turn ends.

Each card will have unique barcodes, one on each side relevant side. The robot will scan the barcodes as it rolls over the surface and maintain an internal database of which barcodes have been read. If a barcode is ever found twice, the robot has entered an infinite loop and the turn is over.

VI. Mobile Platform

The mobile platform consists of two acrylic layers, each holding various parts of the robot.

Top Layer, Top Side:

LCD Screen, for output

MAVRIC-II board with ATmega 128 microcontroller

Top Layer, Bottom Side:

Mounted RS-232 compatible Symbol LS-4004I Barcode Reader

Bottom Layer, Top Side:

Two servos used for actuation

Two battery packs with six rechargeable double-A batteries each

Bottom Layer, Bottom Side:

Super Slider™ plastic device for smooth sliding

LynxMotion Digital Line Follower Module

10-24' and M-4 metric bolts/nuts/spacers are used throughout the robot for structural support.

Epoxy, Velcro, Hot Glue, Superglue all used throughout the robot as adhesive and structural support.

The primary goal in building the mobile platform was a compact design, as every incremental size increase would be magnified by 49, as each square on the game board would have to be enlarged to compensate. This goal drove the two-layer design system. Though it was difficult to mount the unwieldy barcode circuitry (designed for the inside of a hand-grip gun), a hot glue gun eventually saved the day.

VII. Actuation

Actuation was accomplished with two regular servos purchases from Acroname.com, hacked to support continuous motion.

The hacking procedure consists of opening the servo. First, a small clip to prevent a full, 360 degree rotation had to be removed, as it prevented continuous motion. Then, upon feeding the servo a signal with a 20 ms (50 Hz) period and a 1.5 ms pulse, the internal potentiometer was adjusted to give the servo a “neutral” or “stop” command at this pulse width.

In software, output compares were initialized and setup to provide continuous Pulse Width Modulation (PWM) via interrupts. The specific value at the ATmega 128’s clock speed to generate the 1.5 ms pulse was 1382 for the left servo, and 1376 for the right servo. These values are defined as constants at the top of the program for easy changes.

The servo driver software quite simple; the function takes a value from -100 to 100, -100 representing full reverse and 100 representing full forward. The software multiplies these numbers by an appropriate scaling factor for equal drive power, inverts the signal around the appropriate servo (they are mounted in different directions), and places the new value in the corresponding output compare register. In this way, actuation is extremely easy and programmer-friendly.

VIII.Sensors

a. Bump Switch

The bump switch is a standard pushbutton switch mounted on the front of the robot. Wired active-high, the switch will send an interrupt to the main program and cause all function to cease. If the robot is already stopped, it will cause the robot to start running.

Part Number: Unknown

Vendor: IMDL Lab

b. Line Tracker

The line tracker is a digital module with sends three bits to the processor. If the pattern is 010 (or 111, or 000), no adjustment is required. If 100 or 110 is sent, the robot needs to move to the right. If 001 or 011 is sent, the robot needs to move to the left.

Due to manufacturing defects, each of the three IR receivers would register high at a different distance away from the ground.

	Outside	Inside (Halogen Lamp)
Left	2.63 cm	2.93 cm
Center	2.09 cm	2.34 cm
Right	1.87 cm	2.06 cm

Part Number: TRA-01 Ver. 5.0

Vendor: LynxMotion

c. Barcode Reader

The barcode reader constantly scans as the robot runs. If it successfully reads a barcode, that identification number is sent to the processor for storage and cross-checking with known barcode values.

The protocol used by the barcode reader (discovered after testing with both an oscilloscope and a logic state analyzer) is standard RS-232 operating at 9600 baud (Hz). It uses eight data bits, one stop bit, and no parity bits for data checking. The barcode reader will out as many 8-bit characters as it can detect on the barcode, one after the other, with almost no delay between the stop and start bits of consecutive characters.

As many things occur simultaneously, all communication is interrupt-driven through the onboard USART0 interrupt channel and vector.

Free 3-9, an open-source barcode font that does not require the pre-calculation of a checksum value, was used in this project. Requiring approximately 200 dpi at the 36-point font value used, it has support for all basic alphanumeric characters. The beginning and ends of the barcode must be delineated with a special character, which can be typed as an asterisk (*). These values are found on most cards in the game.

The values are cross-checked against an internal database of 39 possible barcodes: four reverses, 30 turns, 2 red goals, 2 blue goals, and one “start card” barcode to prevent a horribly damaging affair.

Part Number – LS-4004i (out of production, obtained via eBay)

Vendor – Symbol

IX. Behaviors

The robot's primary algorithm is simple.

1. If "run" button is pressed, proceed to number 2, else do nothing.
2. Follow straight line.
3. If barcode is found, check memory for repeat entry. If found, proceed to number 6. If not, store new entry in memory.
4. Obey barcode-specific instruction (turn, reverse).
5. If bump switch is pressed, proceed to number 7.
6. Go to number 2.
7. Stop moving. Go to number 1.

In this respect, the robot is quite simple. If all sensors are working properly, there is little chance of malfunction due to complex algorithmic code.

X. Experimental Layout

The game is composed of seven row and nine columns of squares. The northeast and southeast-most squares are the red and blue players' goals, respectively. The robot begins from the center square on the first row, proceeding in any direction the player so chooses. Lines of electrical tape run down the centers of all the rows and columns for the robot to follow.

At the intersection of four game squares, there is a small, plus-shaped wood cutout. This is to provide secure, discrete places for the cards to lay.

The game contains "dead zones", places where the player may not play cards, denoted by squares of diagonal lines. Purple means neither player can play cards there; red means only red can play cards there, and blue means only blue can play cards there. These zones' positions were determined through extensive playtesting.

Further materials used include white spray paint, nails, and wood adhesive.

XI. Results and Experiences

A game does not reflect a typical pattern of “results”, other than the result of increasing life quality and bringing a smile to people’s faces. The game, as seen thus far, has been a success and an absolute joy to create, and so hopefully, that should stand as a testament to this robot.

More important, perhaps, are the lessons learned along the way. Though rather knowledgeable in the theory of electrical engineering, this robot forced an understanding and practical application far beyond any class I have yet experienced as an undergraduate. Still, I essentially understood the electrical engineering principles in play from the onset.

Other disciplines were the crowning achievement. I had heretofore completely avoided anything even remotely resembling mechanical engineering, and I perhaps harbored even irrational fears regarding simple power tools and construction work. Throughout the semester, on the assembly of the platform, I became acquainted with dozens of tools and devices capable of assisting in mechanical design. I may not be fully competent, and the physical platform the least complex section of the project, but I am proud of my accomplishments and look forward to further honing these skills in the future.

Game design, a creative process absolutely unrelated to general engineering, is a personal pastime of mine and something I greatly enjoy. An interesting challenge was being constrained not by software, in which almost anything is possible, but by my physical ability to create the hardware necessary for the game’s operation. I might think of an idea, but if my ability to implement what was necessary for that idea, it had to be scrapped. Designing around these limitations was an interesting experience, and after nearly a dozen hours of playtesting both alone and with my roommates, I believe the current ruleset is an optimal balance between complexity and ease of understanding for the average person.

In the end, IMDL has been a fantastic experience, and I would recommend the class to any and all who come asking.

XII. Conclusion

Shift is an exercise melding completely separate disciplines from both the creative and technical worlds. The dual objectives of proper technical functionality and deep, rewarding gameplay were achieved, and I consider the project to be an absolute success.

XIII. Documentation

AVRFreaks – www.avrfreaks.net

ATMega 128 Documentation - www.atmel.com/atmel/acrobat/doc2467.pdf

Battlestar Galactica – www.scifi.com/battlestar/

XIV.Appendix A (Program Code)

```
/**
 * SHIFT Source Code - Version 1.4
 * Primary Author - Jason Wishnov
 * Secondary Authors - Adam Barnett,
 * ABCMiniUser (AVRFreaks.net)
 * EEL5666 - Intelligent Machine Design Laboratory
 */
```

```
/**
 * Pins - Inputs
 * D0 - Bump Switch
 * E0 - Receive Pin for RS-232
 * E7-E5 - Left, Center, Right Channels for
 * digital line following
 */
```

```
/**
 * Pins - Outputs
 * C7-C0 - LCD Output
 * B0 - LED Output
 * B5 - Left Servo PWM
 * B6 - Right Servo PWM
 * B7 - Trigger for Barcode Reader
 */
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
```

```
#define USART_BAUDRATE 9600
#define BAUD_PRESCALE 95
#define LEFT_CENTER_POINT 1374
#define RIGHT_CENTER_POINT 1380
```

```
void lcd_delay();
void lcd_init();
void lcd_cmd();
void lcd_disp();
```

```

void lcd_clear();
void lcd_row(int row);
void lcd_string(char *a);
void lcd_int(int value);
void irc_init();
void pwm_init();
void uart_init();
void left_motor(int value);
void right_motor(int value);
void turn_left();
void turn_right();
void reverse();
int check_repeats();
void cross_check();

// Global Variables
int stop = 1;
int wait = 0;
int byteCount = 0;
int scannedCount = 0;
unsigned char knownCards[39][2] = {{0x30, 0x30}, {0x30, 0x31}, {0x30, 0x32}, {0x30, 0x33}, //Reverse
Cards
                {0x30, 0x34}, {0x30, 0x36}, {0x30, 0x38}, {0x31, 0x30}, {0x31, 0x32},
                {0x31, 0x34}, {0x31, 0x36}, {0x31,
0x38}, {0x32, 0x30}, {0x32, 0x32},
                {0x32, 0x34}, {0x32, 0x36}, {0x32,
0x38}, {0x33, 0x30}, {0x33, 0x32},//Turn Left Cards
                {0x30, 0x35},{0x30, 0x37}, {0x30,
0x39}, {0x31, 0x31}, {0x31, 0x33},
                {0x31, 0x35},{0x31, 0x37}, {0x31,
0x39}, {0x32, 0x31}, {0x32, 0x33},
                {0x32, 0x35},{0x32, 0x37}, {0x32,
0x39}, {0x33, 0x31}, {0x33, 0x33}, //Turn Right Cards
                {0x33, 0x34}, {0x33, 0x35}, //Red Goal Cards
                {0x33, 0x36}, {0x33, 0x37}, //Blue Goal
Cards
                {0x33, 0x38}}; //Stop Card

unsigned char scannedCards[128][2];

void lcd_delay() {

```

```

long int ms_count = 0;
while (ms_count < 10000)
    ms_count = ms_count + 1;
}

void lcd_cmd(unsigned int myData) {
    unsigned int temp_data = 0;

    temp_data = ( myData | 0b00000100 );
    temp_data = ( temp_data & 0b11110100 );
    PORTC = temp_data;
    lcd_delay();
    PORTC = (temp_data & 0b11110000);

    temp_data = ( myData << 4);

    temp_data = (temp_data & 0b11110100);
    temp_data = (temp_data | 0b00000100);

    PORTC = temp_data;
    lcd_delay();
    PORTC = (temp_data & 0b11110000);
    lcd_delay();
}

void lcd_disp(unsigned int disp) {
    unsigned int temp_data = 0;

    temp_data = ( disp & 0b11110000 );
    temp_data = ( temp_data | 0b00000101 );
    PORTC = temp_data;
    lcd_delay();
    PORTC = (temp_data & 0b11110001);
    lcd_delay();

    temp_data = (disp << 4 );
    temp_data = ( temp_data & 0b11110000 );
    temp_data = ( temp_data | 0b00000101 );
    PORTC = temp_data;
    lcd_delay();
    PORTC = (temp_data & 0b11110001);
    lcd_delay();
}

```

```

}

void lcd_init() {
    lcd_cmd(0x33);
    lcd_cmd(0x32);
    lcd_cmd(0x28);
    lcd_cmd(0x0F);
    lcd_cmd(0x01);
}

void lcd_string(char *a) {
    while (*a != 0)
    {
        lcd_disp((unsigned int) *a);
        a++;
    }
    return;
}

void lcd_int(int value) {
    int temp_val;
    int x = 10000;

    while (value / x == 0)
        x/=10;

    if (value == 0) lcd_disp(0x30);

    else while (x >= 1)
    {
        temp_val = value / x;
        value -= temp_val * x;
        lcd_disp(temp_val+ 0x30);

        x /= 10;
    }
}

void lcd_clear() {
    lcd_cmd(0x01);
}

```

```

void lcd_row(int row) {
    switch(row)
    {
        case 0: lcd_cmd(0x02);
        case 1: lcd_cmd(0xC0);
    }
}

void irc_init() {
    EICRA = 0b00000011;
    EIMSK = 0b00000001;
}

void timer_init() {
    TCCR0 = 0x07;
    TIMSK = 0x01;
}

void pwm_init() {
    TCCR1A = 0b10100000; // .....00:P&FC PWM // 11.....,11.....:OC1A,OC1B inv
    TCCR1B = 0b00010010; // ...10.....:P&FC PWM // .....010:bclk/8 (~8kHz)
    ICR1 = 20000;
    TCNT1 = 0x0000;
    OCR1A = 0x0000; // 1/2 duty, ~2.5v dc level, motor 1 on PB5 (OC1A)
    OCR1B = 0x0000; // 0/256 duty, ~2.5v dc level, motor 2 on PB6 (OC1B)
}

void uart_init() {
    UCSRA=0x00;
    UCSRB=0x90;
    UCSRC=0x06; //Set up UART

    UBRR0L = BAUD_PRESCALE; // Load lower 8-bits of the baud rate value into the low byte of the UBRR0
register
    UBRR0H = (BAUD_PRESCALE >> 8); // Load upper 8-bits of the baud rate value into the high byte of the
UBRR0 register
}

void left_motor(int value) { //Values range from -100 to 100
    OCR1A = LEFT_CENTER_POINT - (value * 1.20);
}

```

```

void right_motor(int value) { // Values range from -100 to 100
  OCR1B = RIGHT_CENTER_POINT + (value * 1.02);
}

```

```

void start_stop() {
  unsigned char temp;
  int i, j;
  if (stop == 1) {
    left_motor(0);
    right_motor(0);
    temp = PINB;
    temp = temp & 0b01111111;
    PORTB = temp;
    for (i = 0; i < 128; i++)
      for (j = 0; j < 2; j++)
        scannedCards[i][j] = 0;
  }
  else {
    left_motor(100);
    right_motor(100);
    temp = PINB;
    temp = temp | 0b10000000;
    PORTB = temp;
  }
}

```

```

void turn_left() {
  unsigned char temp;
  int i;
  left_motor(100);
  right_motor(100);
  lcd_clear();
  lcd_string("Turn!");
  for (i = 0; i < 36; i++)
    lcd_delay();
  temp = PINB;
  temp = temp & 0b01111111;
  PORTB = temp;
  left_motor(60);
  right_motor(-60);
  for (i = 0; i < 48; i++)
    lcd_delay();
}

```

```

left_motor(-40);
right_motor(-50);
for (i = 0; i < 44; i++)
    lcd_delay();
temp = PINB;
temp = temp | 0b10000000;
PORTB = temp;
}

```

```

void turn_right() {
    unsigned char temp;
    int i;
    left_motor(100);
    right_motor(100);
    lcd_clear();
    lcd_string("Turn!");
    for (i = 0; i < 36; i++)
        lcd_delay();
    temp = PINB;
    temp = temp & 0b01111111;
    PORTB = temp;
    left_motor(-60);
    right_motor(60);
    for (i = 0; i < 48; i++)
        lcd_delay();
    left_motor(-40);
    right_motor(-50);
    for (i = 0; i < 44; i++)
        lcd_delay();
    temp = PINB;
    temp = temp | 0b10000000;
    PORTB = temp;
}

```

```

void reverse() {
    unsigned char temp;
    int i;
    left_motor(100);
    right_motor(100);
    lcd_clear();
    lcd_string("Reverse!");
    for (i = 0; i < 36; i++)

```

```

    lcd_delay();
temp = PINB;
temp = temp & 0b01111111;
PORTB = temp;
left_motor(60);
right_motor(-60);
for (i = 0; i < 96; i++)
    lcd_delay();
left_motor(-40);
right_motor(-50);
for (i = 0; i < 44; i++)
    lcd_delay();
temp = PINB;
temp = temp | 0b10000000;
PORTB = temp;
}

```

```

int check_repeats() {
    int i;
    for (i = 0; i < scannedCount; i++)
        if ((scannedCards[scannedCount][0] == scannedCards[i][0]) &&
            (scannedCards[scannedCount][1] == scannedCards[i][1]))
            return i;

    return -1;
}

```

```

void cross_check() {
    int i, j;
    unsigned char temp;
    for (i = 0; i < 39; i++)
        if ((scannedCards[scannedCount][0] == knownCards[i][0]) &&
            (scannedCards[scannedCount][1] == knownCards[i][1])) {
            if (i < 4)
                reverse();
            else
                if (i < 19)
                    turn_right();
            else
                if (i < 34)
                    turn_left();
            else {

```

```

        left_motor(100);
        right_motor(100);
        for (j = 0; j < 36; j++)
            lcd_delay();
    stop = 1;
        start_stop();
    lcd_clear();
    if ((i == 34) || (i == 35))
        lcd_string("Red Wins!");
    else
        if ((i == 36) || (i == 37))
            lcd_string("Blue Wins!");
        else
            lcd_string("Stopped");
    PINB = temp;
        temp = temp | 0b00000001;
    PORTB = temp;
    }
}

int main()
{
    unsigned char temp = 0;

    DDRB = 0xFF;
    DDRC = 0xFF;
    DDRD = 0x00;
    DDRE = 0x00;

    irc_init();
    lcd_init();
    timer_init();
    pwm_init();
    uart_init();
    sei(); // Enable the Global Interrupt Enable flag so that interrupts can be processed

    start_stop();

    lcd_string("SHiFT");
    lcd_row(1);
    lcd_string("By Jason Wishnov");

```

```

while(1) {

    if (stop == 0) {
        temp = PINE;
        temp = temp & 0b11100000;
        switch (temp) {
            case 0b11000000: left_motor(30); right_motor(100); break;
            case 0b10000000: left_motor(30); right_motor(100); break;
            case 0b01100000: left_motor(100); right_motor(30); break;
            case 0b00100000: left_motor(100); right_motor(30); break;
            default: left_motor(100); right_motor(100);
        }
    }
    else {
        left_motor(0);
        right_motor(0);
    }
}

return 0;

}

ISR(INT0_vect) {
    unsigned char temp;
    int i;
    cli();
    temp = PINB;
    temp = temp & 0b11111110;
    PORTB = temp;
    if (stop == 1)
        stop = 0;
    else
        stop = 1;
    if (stop == 0) {
        lcd_clear();
        lcd_string("Running");
        for (i = 0; i < 60; i++)
            lcd_delay();
    }
}

```

```

start_stop();
if (stop == 1) {
    lcd_clear();
    lcd_string("Stopped");
    for (i = 0; i < 255; i++)
        lcd_delay();
    for (i = 0; i < 100; i++)
        lcd_delay();
}
sei();
}

ISR(TIMERO_OVF_vect) {
    unsigned char temp;
    wait++;
    if (wait == 89)
        if (stop == 0) {
            temp = PINB;
            temp = temp & 0b01111111;
            PORTB = temp;
        }
    if (wait == 90) {
        if (stop == 0) {
            temp = PINB;
            temp = temp | 0b10000000;
            PORTB = temp;
        }
        wait = 0;
    }
}

ISR(USART0_RX_vect) {
    char ReceivedByte;
    ReceivedByte = UDR0; // Fetch the recieved byte value into the variable "ByteReceived"
    scannedCards[scannedCount][byteCount] = ReceivedByte;
    byteCount++;
    if (byteCount == 2) {
        if (check_repeats() != -1) {
            stop = 1;
            start_stop();
            lcd_clear();
            lcd_string("Infinite Loop");
        }
    }
}

```

```
}  
cross_check();  
byteCount = 0;  
scannedCount++;  
}  
}
```

XV. Appendix B (Media)

