University of Florida

EEL5666 – Spring 2008

Intelligent Machines Design Lab

Butler

April 22, 2008

Matt Atwood

Table of Contents

Abstract

Butler will be an autonomous robot capable of retrieving a specified snack food or beverage based on an input by the user. A combination of food items and beverage items totaling two will be place on specific locations on a white area. Butler will follow a black line to the specified item, scan it with a barcode reader, and bring the item back to the starting location. Butler will also have to avoid any obstacle which may cross its path and avoid deviating from the line. This paper will describe the techniques and mechanisms I plan to use to accomplish this task.

Executive Summary

Butler was designed to be an autonomous robot that would act as, a butler. The main behaviors of butler were line following, obstacle avoidance, and object retrieval. However, due to a coding error, all three could not be integrated. Currently Butler can follow a line and avoid obstacles but cannot carryout its object retrieval routine at the same time. Its object retrieval routine was designed to receive an input via barcode, move to a successive barcode, deploy a platform to retrieve the item, and return back to the user. Future models of Butler will fix this error and improvements on the platform.

Introduction

Butler is a robot that was designed to retrieve a designated snack food or beverage item based on an input from a user. However, Butler does not have to be limited to just those items as the user can determine the items to be retrieved. The inspiration for Butler is to help people who are easily preoccupied with multiple tasks. When considering the hectic lives people have in this day, Butler's convenience is beneficial.

Integrated System

Butler is controlled by the Atmel ATmega128 microcontroller on a Mavric-IIB board ordered from BDMICRO. The Mavric board was chosen because of the recommendation from the instructors and TAs regarding its ease of use for mechanical and aerospace engineers. The board will be interfaced with the sensors and servos.

For the regular sensors a combination of IR and sonar is used. The sonar is used to avoid any obstacles which may come into Butler's path. The IR is used for line following.

Butler's special sensor is a barcode scanner mounted to its frame. The barcode scanner was designed to accept an input from a user at first. Then, as Butler progresses along the track it will encounter specific barcodes at specific locations that will keep track of its location and whether or not any activity needs to be performed.

Mobile Platform

Butler's mobile platform was designed following the axiom of "Keep It Simple Stupid". First and foremost, the design needed to allow easy access to all of Butler's parts Following this goal, a rectangular prism shape was decided upon. Butler has an upper and lower platform, which are held together by 4 3.5in bolts. Below the lower platform the servos and IR are mounted. The upper platform contains the microcontroller, battery packs and sonar.

This design is as simple as possible. This allows Butler to be assembled/disassembled with ease and speed. This also made the platform lightweight.

Actuation

For propulsion, a total of two servos are used. The forklift requires another servo. Due to the light weight involved in this design, servos were the most cost effective and efficient method of propulsion. The only downside was that the servos had to be hacked for continuous rotation, but this was not much of a problem.

The servos are driven by PWM signal from the microcontroller

Sensors:

IR:

These sensors are used for line following. Lynxmotion SinglLine IR sensors are used in Butler. These sensors provide an incredibly easy interface to use, all that is needed is +5V, a ground, and a signal connection. The IR emits light and measures the intensity of the reflected light returning either a ‘0’ or ‘1’.

Sonar:

The sonar is used as an obstacle avoidance system. Timing the delay of reflected sound waves allows accurate measurement of its distance. By utilizing the minimum distance allowable between the robot and any obstacle in its path provides the obstacle avoidance.

Barcode:

The barcode scanner provides a method of entering an input to the robot and also acts as a measure of the robots progress through its circuit. It is connected via RS-232 interface.

Conclusion

I started this class with the desire of building a robot and no prior knowledge of robotics. Although the robot failed to meet the goals I set for it at the beginning of the semester due to a coding error, I still learned a lot about robotics and wouldn't trade the experience for the world. I think with a little more time and effort, I will get Butler to work and I will have something to look back on with pride, because I know I gave my best effort.

APPENDIX A

CODE:

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

#include <stdlib.h>
#include <stdio.h>
#include <inttypes.h>
#include <avr/pgmspace.h>
#include <ctype.h>
#include <string.h>
#include <util/delay.h>



#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((14745600 / (USART_BAUDRATE * 16UL))) - 1)

// Declare your global variables here
char fromScanner;

void line_follow()
{
```

```
DDRB = 0b01100000;
int lineL = (PINE & 0b10000000);
int lineC = (PINE & 0b01000000);
int lineR = (PINE & 0b00100000);
    if (lineL == 1 && lineC == 0 && lineR ==1)
    {
    OCR1A = 1540;     //left servo
    OCR1B = 1460;     //right servo
    }
    else if (lineL == 0 && lineC == 0 && lineR == 1)
    {
    OCR1A = 1535;
    OCR1B = 1400;
    }
    else if (lineL == 0 && lineC == 1 && lineR == 1)
    {
    OCR1A = 1535;
    OCR1B = 1375;
    }
    else if (lineL == 1 && lineC == 0 && lineR == 0)
    {
    OCR1A = 1600;
    OCR1B = 1465;
    }
    else if (lineL == 1 && lineC == 1 && lineR == 0)
```

```
    {
    OCR1A = 1625;
    OCR1B = 1465;
    }
    else if (lineL == 1 && lineC == 1 && lineR == 1)
    {
    OCR1A = 1475;
    OCR1B = 1525;
    }
    else if (lineL == 0 && lineC == 0 && lineR == 0)
    {
    OCR1A = 1500;
    OCR1B = 1500;
    }
    else if (lineL == 0 && lineC == 1 && lineR == 0)
    {
    OCR1A = 1500;
    OCR1B = 1500;
    }
}
int get_Sonar1()
{
        int n = 0;

        PORTA = (PINA & 0x7F);    // these two lines create a rising edge
```

```
PORTA = (PINA | 0x80);       // on PortA pin 7


while (n < 40)
{
        //waste enough clock cycles for at least 10us to pass
        n += 1;
        n++;
}


PORTA = (PINA & 0x7F);    // force PortA pin 7 low to create a falling edge
                                                    // this sends out the trigger


while (!(PINA & 0x01))
{
        // do nothing as long as echo line is low
}


n = 0;   //re-use our dummy variable for counting


while (PINA & 0x01)
{
        n += 1;           // add 1 to n as long as PortA pin 0 is high
}


//when we get here, the falling edge has occured
```

```
	return n;
}
int get_Sonar2()
{
	int n = 0;

	PORTC = (PINC & 0x7F);	// these two lines create a rising edge
	PORTC = (PINC | 0x80);	// on PortC pin 7

	while (n < 60)
	{
		//waste enough clock cycles for at least 10us to pass
		n += 1;
		n++;
	}

	PORTC = (PINC & 0x7F);	// force PortC pin 7 low to create a falling edge
							// this sends out the trigger

	while (!(PINC & 0x01))
	{
		// do nothing as long as echo line is low
	}
```

```
	n = 0;   //re-use our dummy variable for counting

	while (PINC & 0x01)
	{
		n += 1;		// add 1 to n as long as PortA pin 0 is high
	}

	//when we get here, the falling edge has occured

	return n;
}
int get_Sonar3()
{

	int n = 0;

	PORTD = (PIND & 0x7F);    // these two lines create a rising edge
	PORTD = (PIND | 0x80);	  // on PortD pin 7


	while (n < 80)
	{
		//waste enough clock cycles for at least 10us to pass
		n += 1;
		n++;
```

```
        }

        PORTD = (PIND & 0x7F);    // force PortD pin 7 low to create a falling edge
                                                    // this sends out the trigger

        while (!(PIND & 0x01))
        {
                // do nothing as long as echo line is low
        }

        n = 0;   //re-use our dummy variable for counting

        while (PIND & 0x01)
        {
                n += 1;             // add 1 to n as long as PortD pin 0 is high
        }

        //when we get here, the falling edge has occured

        return n;
}

int main(void)
{
long SonarVal1 = 0;
```

```
long SonarVal2 = 0;

long SonarVal3 = 0;

int j;



// Declare your local variables here


   UCSR0B |= (1 << RXEN0) | (1 << TXEN0);   // Turn on the transmission and reception circuitry

   UCSR0C |=(1 << UCSZ0); // Use 8-bit character sizes


   UBRR0L = BAUD_PRESCALE; // Load lower 8-bits of the baud rate value into the low byte of the UBRR register

   UBRR0H = (BAUD_PRESCALE >> 8); // Load upper 8-bits of the baud rate value into the high byte of the UBRR register


   UCSR0B |= (1 << RXCIE0); // Enable the USART Recieve Complete interrupt (USART_RXC)

   sei(); // Enable the Global Interrupt Enable flag so that interrupts can be processed


// Input/Output Ports initialization

// Port A initialization

// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In

// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T

PORTA=0x00;

DDRA = 0x80;
```

```
// Port B initialization
// Func7=In Func6=Out Func5=Out Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=0 State5=0 State4=T State3=T State2=T State1=T State0=T
PORTB=0b11100000;
DDRB =0x00;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC = 0x80;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD= 0x00;
DDRD = 0x80;

// Port E initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTE=0b00001000;                       /* +5V on pin 3 GND on pin2*/
DDRE =0b00001100;               /* Output on pins 2 & 3*/
PINE =0b11100000;               /* Input on pins 5,6,7*/
```

```
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 1843.200 kHz
// Mode: Ph. & fr. cor. PWM top=ICR1
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR1A=0xA0;
TCCR1B=0x12;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1 = 20000;
OCR1A = 1500;
OCR1B = 1500;
OCR1C = 1500;

// Timer/Counter 2 initialization
```

```
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// Timer/Counter 3 initialization
// Clock source: System Clock
// Clock value: Timer 3 Stopped
// Mode: Normal top=FFFFh
// Noise Canceler: Off
// Input Capture on Falling Edge
// OC3A output: Discon.
// OC3B output: Discon.
// OC3C output: Discon.
// Timer 3 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
// Compare C Match Interrupt: Off
TCCR3A=0x00;
TCCR3B=0x00;
TCNT3H=0x00;
```

```
TCNT3L=0x00;
ICR3H=0x00;
ICR3L=0x00;
OCR3AH=0x00;
OCR3AL=0x00;
OCR3BH=0x00;
OCR3BL=0x00;
OCR3CH=0x00;
OCR3CL=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
// INT3: Off
// INT4: Off
// INT5: Off
// INT6: Off
// INT7: Off
EICRA=0x00;
EICRB=0x00;
EIMSK=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x00;
```

```
ETIMSK=0x00;

// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;



while (1)
    {

        fromScanner = UDR0;
        if (fromScanner == '1')
        {
        j = 1;
        SonarVal1 = get_Sonar1();
        SonarVal2 = get_Sonar2();
        SonarVal3 = get_Sonar3();
        while ((SonarVal1 > 500) | (SonarVal2 > 500))
        {
        line_follow();
        }
        while ((SonarVal1 < 500) | (SonarVal2 < 500))
        {
```

```
 DDRB = 0b01100000;
 OCR1A = 1440;
 OCR1B = 1560;
 }
}
else if (fromScanner == '2')
 {
 j = 2;
 SonarVal1 = get_Sonar1();
 SonarVal2 = get_Sonar2();
 SonarVal3 = get_Sonar3();
 while ((SonarVal1 > 500) | (SonarVal2 > 500))
 {
 line_follow();
 }
 while ((SonarVal1 < 500) | (SonarVal2 < 500))
 {
 DDRB = 0b01100000;
 OCR1A = 1440;
 OCR1B = 1560;
 }
}
else if (fromScanner == '3')
 {
 j = 3;
```

```
 SonarVal1 = get_Sonar1();
 SonarVal2 = get_Sonar2();
 SonarVal3 = get_Sonar3();
 while ((SonarVal1 > 500) | (SonarVal2 > 500))
 {
 line_follow();
 }
 while ((SonarVal1 < 500) | (SonarVal2 < 500))
 {
 DDRB = 0b01100000;
 OCR1A = 1440;
 OCR1B = 1560;
 }
}
else if (fromScanner == '4')
 {
 if (j == 1)
 {
 DDRB = 0b10000000;
 OCR1C = 1525;
 _delay_ms(250);
 OCR1C = 1475;
 _delay_ms(250);
 fromScanner = '1';
 }
```

```
else
{
fromScanner = '1';
}
}
else if (fromScanner == '5')
{
if (j == 2)
{
DDRB = 0b10000000;
OCR1C = 1525;
_delay_ms(250);
OCR1C = 1475;
_delay_ms(250);
fromScanner = '1';
}
else
{
fromScanner = '1';
}
}
else if (fromScanner == '6')
{
if (j == 3)
{
```

```
        DDRB = 0b10000000;
        OCR1C = 1525;
        _delay_ms(250);
        OCR1C = 1475;
        _delay_ms(250);
        fromScanner = '1';
        }
        else
        {
        fromScanner = '1';
        }
        }
        else if (fromScanner == '7')
        {
        DDRB = 0b01100000;
        OCR1A = 1540;
        OCR1B = 1460;
        _delay_ms_(250);
        DDRB = 0x00;
        }
}
        return 0;
}
```

```
SIGNAL(SIG_USART0_RECV)
{
  char ReceivedByte;
  ReceivedByte = UDR0; // Fetch the recieved byte value into the variable "ByteReceived"
  UDR0 = ReceivedByte; // Echo back the received byte back to the computer
}
```