University of Florida

EEL5666

Intelligent Machine Design Lab

Dr. A. Arroyo

Dr. Eric M. Schwartz

balBot

A Self-Balancing Traveling Robot

Arturo G. Codina

9178-2130

Table of Contents

Abstract…..…………………………………………………………………. 3

Executive Summary...…………………………………………………………. 4

Introduction…..……………………………………………………………..…. 5

Integrated System………………………………………………………… 6

Mobile Platform……………………………………………………………….. 7

Actuation………………………………………………………………………. 8

Sensors…………………………………………………………………….….. 9

Behaviors..….……………………………………………………………… 10

Conclusion…………………………………………………………………….. 11

Documentation…………………………………………………………….... 12

Appendix...……………………………………………………………………. 14

Abstract

balBot is an autonomous robot that will balance on its own two wheels. balBot will try to keep itself upright by rotating its two wheels in the appropriate direction. The entire robot is completely autonomous and therefore has all of its systems onboard and powered through its own power supply. To accomplish the ability of balance, an accelerometer and a gyroscope will be used to determine the platform tilt to drive the wheels appropriately.

Executive Summary

balBot is a two-wheeled self balancing robot that will be capable of standing upright without any external aid. It has enough on board power to remain standing for at least an hour (and possibly longer). The robot is also capable of colliding with obstacles and not falling over while it attempts to balance itself.

balBot is driven by an Atmel based microcontroller unit running at 16MHz. The processor will be in charge of reading all of the tilt sensors and performing the necessary calculations required to balance the robot. It will also be responsible for sending the correct speed and direction for the motors.

The accelerometer and gyroscope values will be combined together in a complementary fashion to obtain a consistent tilt value. This complementary operation is required to avoid any errant data due to a drift in the gyroscope's output or errors introduced by the running integration on the output of the gyro.

Introduction

The inverted pendulum is the classic controls problem. The inverted pendulum is a system that is inherently unstable. Without active control, the system will become unstable and collapse (in this case, fall). In the case of balBot, the platform acts as the inverted pendulum itself. The motor axles will each act as the pivot point of the system. Using specialized sensors, the tilt position of the platform will be determined to produce a force that counteracts the falling platform. This opposing force will regain platform stability and prevent the platform from falling.

The following report will cover all the topics and aspects in designing a project as described above. Many factors must be considered such as platform design, actuation, sensor design and implementation, and behaviors. Each section will be discussed thoroughly below.

Integrated System

balBot's processing power will be based around the Atmel ATmega128 microcontroller unit (MCU). The MCU is part of a pre-built development board, the Mavric II (http://www.bdmicro.com). It has a built in clock frequency of 16 MHz which is far faster than will ever be needed to achieve balance in balBot. The MCU will be constantly monitoring the platform tilt sensors to determine its position. Using the information gathered from the sensors, the MCU will then decide how much torque to apply to the DC motors in order to achieve platform stability. Figure 1 shows a basic block diagram of the integrated system.

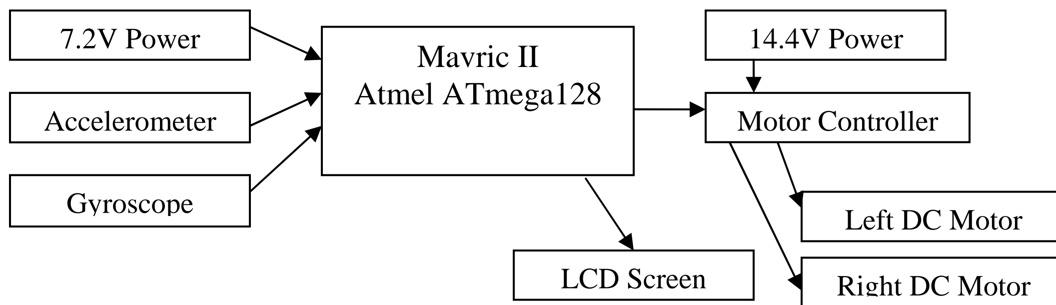

Figure 1: Overall Block Diagram

The MCU is powered by a 7.2V 3Ah NiMH rechargeable battery. The DC motors are powered by a separate and isolated 14.4V 3Ah NiMH battery in order to eliminate noise on the MCU power rail as well as provide enough power to the motors.

In addition, the robot platform will have a LCD screen for feedback purposes. This LCD will be useful in reporting back the actual tilt angle and motor speeds that are being sent to the motor controller. During the initial testing phases, the LCD screen provided debugging values in order to aid in the programming process.

Mobile Platform

The platform for balBot consists of three seven-inch diameter circular platforms which are supported by four ¼" threaded rods. Each circular disc was made out of five-ply balsa wood that was cut out using the T-Tec machine in the IMDL lab. The four support rods used to hold the platforms together are two feet long and represent the height of the robot. Each platform is placed on the top, middle, and bottom of the support rods and secured in place using nuts and washers. The tall nature of the robot and the heavy weight from the batteries on the top platform help move the center of gravity to a higher position on the robot. This helps increase the moment of inertia of the entire platform and simplifies the balancing process somewhat. One can think of this concept as the difference of trying to balance a baseball bat and a small pen by hand. The pen is inherently more difficult to balance compared to the baseball because its moment of inertia is much lower.

During the platform construction process, it was determined that the four support rods should be as close to the axis perpendicular to the rotational axis. If the robot were to fall over, the support rods would be able to absorb a good amount of the impact. Also, it may have been helpful to build the platform larger than two feet. This may have helped with the balancing process tremendously.

In order to cut the cost of the robot and improve on the looks, I decided to machine my own custom made wheels using some scrap acrylic. The two wheels were made to within a 0.001" tolerance and had diameter of a standard music CD. The hubs that attached the wheels to the motor shafts were also custom made out of some scrap aluminum rods. By machining these pieces, I was able to save approximately $30.

Actuation

The actuation of the robot platform is performed by two DC motors from Lynxmotion (http://www.lynxmotion.com). Each motor is a 12V DC gearhead motor capable of a maximum speed of 141 RPM. Although speed is important, the motor and gearhead must also be able to provide a large amount of torque. The specific torque rating for these motors is 231 on.- in. (~14.4 lb.- in.). To achieve this torque, the motor is geared down (through the gearhead) with a 50:1 ratio. The motors are controlled with a Nation Instruments (http://www.national.com) LMD18200 3A, 55V H-Bridge. The LMD18200 accepts a PWM input for which it then converts this PWM signal into a positive or negative current to the DC motors. The change in polarity is physically represented as a change in direction. Therefore, the motor can change direction very quickly and draw the current it needs from the H-Bridge configuration. The actual motor controller PCB was designed by our TA, Max Koessik. Using his PCB layout and the LMD18200s from National, a compact motor controller was created that allowed for easy interfacing of the motors. Once the board was assembled and all required signals and power was connected, I was unable to get my motors to respond to my PWM inputs. After much head scratching, I finally went and sought some help from Chris Taylor, a friend who had previously taken this course and inspired me to take on the task of a self-balancing robot. After looking at the robot for two minutes, he instantly found the problem. Since I had two separate power supplies, the ground lines were completely separate between the MCU and the motor controller. I completely forgot to connect a common ground between the two boards. Once this was done, the motors began to respond the PWM signals from the MCU.

Sensors

To maintain a sense of balance and platform position, two different sensors will be used in balBot. Each sensor is described below in detail:

Accelerometer

The accelerometer is a device that measures static and dynamic acceleration of gravity with respect to the Earth. Using the static acceleration measurements, the accelerometer can provide an excellent measurement of the platform tilt angle. Unfortunately, there is an undesired result with the dynamic acceleration measurements. If the platform were to be to accelerate towards the ground (e.g. falling), the increase in acceleration appears at the accelerometer output (after all, this is what it is supposed to do!). Because of this, using the accelerometer alone as a tilt sensor is only effective if the platform is not accelerating. In addition, any vibrations that the motors create within the platform are also picked up by the accelerometer. These vibrations tremendously decrease your output resolution since a few of the bits will be lost due to noise. To remedy this problem, another sensor is needed.

Gyroscope

A gyroscope is a device that measures angular rate/velocity. If the output of the gyro is integrated, the position of the platform can be determined. Ideally, the gyro can be used as a tilt sensor but there is an error introduced. Gyroscopes tend to drift over time and therefore report inaccurate information and the running integration of the output also introduces small errors. However, if the accelerometer and gyro were combined using a complementary filter, they would be able to help each other. The accelerometer would correct the drift of the gyro when the platform was not falling.

Behaviors

balBot's main behavior will consist of balancing on two wheels. All other behaviors will be based off of this primary and fundamental behavior of balancing. Once the main goal is achieved, basic behaviors such as obstacle *avoidance* and random traveling will be added. Collision detection is not that much of a necessity because if balBot runs into an object, he merely 'bounces' off of the object (due to his balancing algorithm) and continues traveling. One future goal (probably not in this semester) will be to add some sort of vision system and have the robot follow color blobs.

Ideally, the balancing behavior should be run as an interrupt driven process. As the robot begins to tilt, an interrupt can run the balancing algorithms and return to whatever it was doing in the first place. However, due to the difficult nature of balancing and lack of time, this idea was nor pursued. The current algorithm solely prevents the robot from tipping over.

Conclusion

Overall, balBot was almost a complete success. A crucial step in the balancing process is the calibration of the center point for the robot. If the center point is off by just a minute angle, the robot will begin to drift towards that direction and will not be able to recover.

Another problem that was discovered towards the end of the semester was the weight distribution created by the batteries. It was observed that just a slight change in the position (~ 1/8” movement) of the battery packs on top caused the robot to balance much better (or worse) when centered correctly. This is due to the fact that when the weight distribution is centered, the platform may in fact be tilting just slightly off-axis. This small tilt causes the constant downward force that the current algorithm in balBot cannot account for.

One of the more difficult portions of this type of project is the software. As this robot requires a simple yet sturdy platform, plenty of time should be dedicated to developing robust code for balancing. The solution to the balancing problem requires the use of a non-linear solution. In my case, more careful planning and taking advantage of lost time would have probably simplified my software headaches.

As a final note for those who may attempt this project, I cannot stress how important it is to start early. This project may seem easy enough (as did I when I first started it) but as you begin to try different algorithms, the frustration increasingly multiplies. The algorithm I used it very primitive and cannot do much in its present state. I will continue to work on my robot and hope to improve on its function over time. To be successful, one must fail many times!

Documentation

Special Thanks

I would like to give special thanks to the following people. If it was not for their help, I would not have been able to complete this project with the success that I've had.

- **Chris Taylor:** For inspiring me to take on a project like this and helping me throughout the way. For loaning me your oscilloscope, helping me with software issues (LCD & A/D code) and giving me endless tips that you learned when you attempted this project.
- **Steven Corbett:** For helping me with all the mechanical aspects of my robot as well as teaching me how to use some of the machines in the machine shop.
- **Ron Brown:** For being extremely patient with me on teaching me how to use the machines in his machine shop. He also took a lot of his own time explaining how to use certain tools to my advantage.
- **Max Koessik:** For sending out professionally made boards for my motor controller and breakout board for my accelerometer. He also cut out my robot platform on the T-Tec machine as well.
- **William Dubel:** For helping me in the beginning with a few questions I had about my robot design.
- **Dr. Arroyo:** For those wonderful, LOUD, and entertaining lectures early in the morning.
- **Dr. Schwartz:** For giving me the idea of how to mount my sensor board.

- **Mike Stapleton:** For taking time out of his extremely busy schedule to cut a board out for me on the T-Tec machines in the Senior Design lab before the summer break.
- **Larry Barello:** For corresponding with me through email about his balancing robot and for helping me with some hints as to how to integrate the accelerometer and gyroscope.
- **Ingrid Bobadilla:** For supporting me throughout the semester.

Parts

Parts	Price	Shipping	Used?
BDMicro Mavric II Board (ATmega128 @ 16 MHz)	$86.00	$5.00	Yes
Analog Devices ADXL203 Accelerometers	FREE	FREE	Yes
Analog Devices ADXRS150EB Gyroscope	$50.00	$11.99	Yes
Lynxmotion 12VDC Gearhead Motors - 50:1 - 231.5 oz.in.	$59.90	$7.33	Yes
Lynxmotion Motor Mounts	$7.95	$0.00	Yes
BatterySpace 7.2V 3000mAh NiMH Rechargeable Batters & Charger	$54.95	$9.50	Yes
Digikey AVRISP Programmer	$29.00	$6.07	Yes
Hamamatsu P5587 Photoreflectors	$12.00	$3.00	No
National LMD18200 H-Bridge 3A Motor Drivers	FREE	FREE	Yes
Ebay 24x2 LCD Screens	$17.00	$N/A	Yes
Custom-made Acrylic Wheels	FREE	FREE	Yes
Custom-made Aluminum Motor Hubs	FREE	FREE	Yes
Lowe's Hardware (nuts, bolts, washers, etc.)	~$15.00	N/A	Yes

Appendix

Code

→ lcd.h

```
/*
* lcd.h
*
* Author: Max Billingsley
* Adapted by: Greg Beckham
*/
#include <avr/io.h>
#include <avr/signal.h>
#include <inttypes.h>
#define LCD_PORT PORTC
#define LCD_DDR DDRC
#define ENABLE 0x08

/* function prototypes */
void lcd_set_ddr(void);
void lcd_init(void);
void lcd_delay(void);
void lcd_send_str(char *s);
void lcd_send_byte(uint8_t val);
void lcd_send_command(uint8_t val);
```

→ lcd.c

```
/*
* lcd.c
*
* Author: Max Billingsley
* Adapted by: Greg Beckham
*/
/*
* LCD_PORT1 = RS //
* LCD_PORT2 = R/W //
* LCD_PORT3 = EN //
* LCD_PORT4 = DB4 //
* LCD_PORT5 = DB5
* LCD_PORT6 = DB6
* LCD_PORT7 = DB7
*
* RS: Register Select:
*
* 0 - Command Register 15
* 1 - Data Register
*
*/
#include "lcd.h"
/* entry point */
void lcd_init(void)
{
        lcd_send_command(0x83);
        lcd_send_command(0x83);
```

```
		lcd_send_command(0x83);
		lcd_send_command(0x82);
		lcd_send_command(0x82);
		lcd_send_command(0x8c);
		lcd_send_command(0x80);
		lcd_send_command(0x0f);
		lcd_send_command(0x00);
		lcd_send_command(0x01);
}
void lcd_set_ddr(void)
{
		LCD_DDR = 0xff;
}
void lcd_delay(void)
{
		uint16_t i;
		for(i = 0;i<2000;i++){}
}
void lcd_send_str(char *s)
{
		while (*s) lcd_send_byte(*s++);
}
void lcd_send_byte(uint8_t val)
{
		uint8_t temp = val;
		val &= 0xf0;
		val |= 0x02;
		LCD_PORT = val;
		lcd_delay();
		LCD_PORT |= ENABLE;
		LCD_PORT &= ~ENABLE;
		temp <<= 4;
		temp |= 0x02;
		LCD_PORT = temp;
		lcd_delay();
		LCD_PORT |= ENABLE;
		LCD_PORT &= ~ENABLE;
}
void lcd_send_command(uint8_t val)
{
		uint8_t temp = val;
		val &= 0xf0;
		LCD_PORT = val;
		lcd_delay();
		LCD_PORT |= ENABLE;
		LCD_PORT &= ~ENABLE;
		temp <<= 4;
		LCD_PORT = temp;
		lcd_delay();
		LCD_PORT |= ENABLE;
		LCD_PORT &= ~ENABLE;
		lcd_delay();
}
```

→ adc.h

```
#ifndef __adc_h__
#define __adc_h__

void     adc_init(void);
void     adc_chsel(uint8_t channel);
void     adc_wait(void);
void     adc_start(void);
uint16_t adc_read(void);
uint16_t adc_readn(uint8_t channel, uint8_t n);
#endif
```

→ adc.c

```
/*
 * ATmega128 A/D Converter utility routines
 */

#include <avr/io.h>
#include <stdio.h>

/*
 * adc_init() - initialize A/D converter
 *
 * Initialize A/D converter to free running, start conversion, use
 * internal 5.0V reference, pre-scale ADC clock to 125 kHz (assuming
 * 16 MHz MCU clock)
 */
void adc_init(void)
{
  /* configure ADC port (PORTF) as input */
  DDRF  = 0x00;
  PORTF = 0x00;

  ADMUX = BV(REFS0);
  ADCSR = BV(ADEN)|BV(ADSC)|BV(ADFR) | BV(ADPS2)|BV(ADPS1)|BV(ADPS0);
}


/*
 * adc_chsel() - A/D Channel Select
 *
 * Select the specified A/D channel for the next conversion
 */
void adc_chsel(uint8_t channel)
{
  /* select channel */
  ADMUX = (ADMUX & 0xe0) | (channel & 0x07);
}


/*
 * adc_wait() - A/D Wait for conversion
 *
 * Wait for conversion complete.
 */
void adc_wait(void)
```

```
{
  /* wait for last conversion to complete */
  while ((ADCSR & BV(ADIF)) == 0)
    ;
}


/*
 * adc_start() - A/D start conversion
 *
 * Start an A/D conversion on the selected channel
 */
void adc_start(void)
{
  /* clear conversion, start another conversion */
  ADCSR |= BV(ADIF);
}


/*
 * adc_read() - A/D Converter - read channel
 *
 * Read the currently selected A/D Converter channel.
 */
uint16_t adc_read(void)
{
  return ADC;
}


/*
 * adc_readn() - A/D Converter, read multiple times and average
 *
 * Read the specified A/D channel 'n' times and return the average of
 * the samples
 */
uint16_t adc_readn(uint8_t channel, uint8_t n)
{
  uint16_t t;
  uint8_t i;

  adc_chsel(channel);
  adc_start();
  adc_wait();
  adc_start();

  /* sample selected channel n times, take the average */
  t = 0;
  for (i=0; i<n; i++) {
    adc_wait();
    t += adc_read();
    adc_start();
  }
  /* return the average of n samples */
  return t / n;
}
```

→ main.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include "lcd.h"
#include <stdio.h>
#include "adc.h"

#define CPU_FREQ 16000000L  /* set to clock frequency in Hz */

volatile uint16_t ms_count;

void ms_sleep(uint16_t ms) //delay for x milliseconds
{
  TCNT0  = 0;
  ms_count = 0;
  while (ms_count != ms);
}

SIGNAL(SIG_OUTPUT_COMPARE0) //millisecond counter interrupt vector
{
  ms_count++;
}

/*
 * initialize timer 0 to use the real time clock crystal connected to
 * TOSC1 and TOSC2 to generate a near 1 ms interrupt source
 */
void init_timer(void)
{
  /*
   * Initialize timer0 to use the 32.768 kHz real-time clock crystal
   * attached to TOSC1 & 2.  Enable output compare interrupt and set
   * the output compare register to 32 which will cause an interrupt
   * to be generated every 0.9765625 milliseconds - close enough to a
   * millisecond.
   */
  TIFR  |= BV(OCIE0)|BV(TOIE0);
  TIMSK |= BV(OCIE0);    /* enable output compare interrupt */
  TIMSK &= ~BV(TOIE0);   /* disable overflow interrupt */
  ASSR  |= BV(AS0);      /* use asynchronous clock source */
  TCNT0  = 0;
  OCR0   = 32;           /* match in 0.9765625 ms */
  TCCR0  = BV(WGM01) | BV(CS00); /* CTC, no prescale */
  while (ASSR & 0x07)    ;
  TIFR  |= BV(OCIE0)|BV(TOIE0);
}


int main(void)
{
	init_timer();
	lcd_set_ddr();
	lcd_init();
	fdevopen(lcd_send_byte,NULL,0); //override the printf function
```

```
	PORTB=0x00;
	DDRB=0x60;		//set PortB DDR for PWM outputs

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 62.500 kHz
// Mode: Fast PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// OC1C output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
	TCCR1A=0xA1;
	TCCR1B=0x09;
	TCNT1H=0x00;
	TCNT1L=0x00;
	ICR1H=0x00;
	ICR1L=0x00;
	OCR1AH=0x00;
	OCR1AL=0x80;//80 - dead stop
	OCR1BH=0x00;
	OCR1BL=0x80;//80 - dead stop
	OCR1CH=0x00;
	OCR1CL=0x00;

	adc_init(); //Init A/D
	sei();			//Enable interrupts

	float null_pt, new_pt, old_pt, offset, midpoint, accel;
	float accelDC = 0, gyroDC = 0;
	float torque2, position;
	int int_pos, int_torque, gyroDCint, accelDCint;

//Calibration for center point...
	for(int q=0;q<3;q++)
	{
		lcd_send_command(0x80);
		printf("Calibrating...");
		ms_sleep(1000);
		gyroDC += adc_readn(1,10);
		accelDC += adc_readn(0,10);
	}

	gyroDC /= 3;
	accelDC /=3;
	gyroDCint = gyroDC; //Making integers for LCD output
	accelDCint = accelDC;

	lcd_send_command(0x80);
	printf ("GyroDC = %4i", gyroDCint);
	lcd_send_command(0xC0);
	printf("AccelDC = %4i", accelDCint);
	ms_sleep(2000);
//End calibration block...

//Start initial point for running integral...
```

```
		null_pt = 0;
		new_pt = adc_readn(1,1) - gyroDC;
		offset = (new_pt - null_pt)/2;
		midpoint = null_pt + offset;
		position = midpoint * 0.1;
		old_pt = new_pt;
//End initial point integral block...

  while (1) //good values for acceldc = 491-493
  {
    ms_sleep(10); //Run loop every 1/100th of a second

//Running integral of Gyro output...
		new_pt = adc_readn(1,20) - gyroDC;
		offset = (new_pt - old_pt)/2;
		midpoint = old_pt + offset;
		position += (midpoint * 0.1);
		old_pt = new_pt;
//End integral block...

//Read the accelerometer and adjust the gyro drift...
		accel = adc_readn(0,30) - accelDC;
		position += (accel-position)*0.017; //Add 1.7% of the accelerometer reading
//End gyro correction block...

		torque2 = position * 2.56 + 128; //Set a motor speed according to tilt
		int_pos = position; //Outputting integers to LCD

//Adjusting the torque values...
		if (torque2 < 255 && torque2 > 0) //If within valid range of torque values...
		{
			if(torque2 < 140 && torque2 > 128) //Adjust small torque values since motors
			{					//don't start instantly. Also, give
				OCR1A=torque2*1.084;//1.0723	//the torque values an extra boost so
				OCR1B=torque2*1.084;//1.075	//platform can stabilize early.
			}

			else if(torque2 > 116 && torque2 < 128)
			{
				OCR1A=torque2/1.084;//1.081
				OCR1B=torque2/1.084;//1.084
			}
			else	//If not a small torque value, just use regular torque value.
			{
				OCR1A=torque2;
				OCR1B=torque2;
			}
		}
		else if (torque2 > 255) //If torque value is beyond MAX value, cap it.
		{
			OCR1A=255;
			OCR1B=255;
		}
		else if (torque2 < 0) //If torque value is beyond MIN value, cap it.
		{
			OCR1A=0;
```

```
                    OCR1B=0;
            }
            else {}

            int_torque = torque2;
            lcd_send_command(0x80);
            printf("Trying to balance...");
            lcd_send_command(0xC0);
            printf("Tilt = %4i | PWM = %4i",int_pos, int_torque);

   }
}
```