Scrap The MxK – Assignment 2

Mechatronics 1 – Spring 2018
University of Technology, Sydney

BALANCING ROBOT

Dinh Tung “Tony” Le
13302245
CONTENTS

THE GOAL ..............................................................................................................................................2
   The main goal ......................................................................................................................................2
   Why build a self-balancing robot? ......................................................................................................2
OVERVIEW ..............................................................................................................................................3
   How the robot operates .........................................................................................................................3
   The components .....................................................................................................................................3
   The modules ........................................................................................................................................4
THE PROCESS ........................................................................................................................................5
   The overall process of making the robot .................................................................................................5
   The libraries .........................................................................................................................................5
   Testing individual modules ..................................................................................................................12
   Building the robot .................................................................................................................................13
   Dynamics and Control .........................................................................................................................14
   The main program flowchart ...............................................................................................................16
   The app ................................................................................................................................................17
PICTURES ..............................................................................................................................................18
APPENDIX .............................................................................................................................................19
REFLECTION .........................................................................................................................................20
THE GOAL

The main goal

Starting this project, I set out to learn how to use the Arduino board, with its microprocessor – the ATMega328P to the fullest. That means being able to write C codes and use its modules not relying on any prewritten libraries (especially the Arduino IDE). By doing so, I would gain a new skill of using microcontrollers by looking at their datasheets.

Why build a self-balancing robot?

There are three reasons why I chose to build a self-balancing robot to achieve that goal:

1. The self-balancing robot requires a lot of modules to operate. These include timers (timing and PWM), I/O, UART, and I2C. Therefore by building this robot, I will learn how to use these modules and create my own libraries to use in the future.

2. It has been my hobby project for the last two years. I have made two unsuccessful attempts of building such a robot in the past and I felt like this third time would be a charm.

3. At the time of making the robot, I was also studying the subject “Dynamics and Control” at UTS – which taught me how to analyze the dynamics of a system and control it using closed-loop controllers (like the PID controller).
How the robot operates

The robot must be able to do the following task:
1. Balance itself on 2 wheels without any assistance
2. Be controlled wirelessly with Bluetooth via a smartphone app

In order to achieve those tasks, I must use some external hardware components other than the microcontroller itself.

The components

1. The Arduino UNO R3 – with ATMega328P
2. Dual H-Bridge DC Motor Controller L298N (up to 12VDC)
3. HC-05 Wireless Bluetooth Module (both transmitting and receiving)
4. MPU-6050 3 Axis Accelerometer Gyroscope Sensor Module
5. 12V 350rpm DC Motors (with wheels and brackets to mount)
6. 11.1V 5500mAh 35C LiPo Battery
7. Buttons, resistors, and wires

To use these components, we need to use modules of the ATMega
The modules

1. The I/O PORTS:
   Using the Digital INPUT/OUTPUT pins on the Arduino to control external devices like the H-Bridge Motor Controller
2. The Timers:
   Using the timer modules to create PWM signals, as well as timing the actions taken by the Arduino.
3. UART:
   Since I wanted to control the robot with Bluetooth – which is a UART protocol at 9600 baudrate, I must use the UART module of the ATMega.
4. I2C (Two-Wire module)
   Using the Inter-Integrated Circuit (I2C) protocol or the Two-Wire Serial Interface (called by Microchip) is essential to the making of the robot. Because the MPU6050 sensor module uses I2C to communicate and send out values.
The overall process of making the robot

I. The libraries

I decided to start with writing my own libraries to replace the Arduino IDE. The only 2 AVR libraries that I used was <avr/io.h> to get the register names and <avr/interrupt.h> to handle interrupt using pre-allocated memory space (to get a faster response).

Digital Input/Output

```c
int checkpin(int pin, int state){ //check if the pin number or the state is valid
    if(pin < 0 || pin > 13 || state < 0 || state > 1) return LOW;
    return HIGH;
} //end of checkpin()

int pinMode(int pin, int state){ //set a digital pin as input or output
    if(checkpin(pin, state) == 0) return LOW;
    else if(pin < 8){
        switch(state){
            case OUTPUT: DORD |= (1<<pin); break;
            case INPUT: DORD &= ~(1<<pin); break;
        } //end of switch()
    } //end of else if()
    else{
        switch(state){
            case OUTPUT: DORB |= (1<<(pin - 8)); break;
            case INPUT: DORB &= ~(1<<(pin - 8)); break;
        } //end of switch()
    } //end of else()
    return HIGH;
} //end of pinMode()
```
I am replicating the Arduino IDE’s functions like “digitalWrite” and “digitalRead” by manipulating bits of the registers of the ATmega328P.

**Timers**

```c
//volatile uint8_t flagt0;
volatile unsigned long t0count = 0;
volatile unsigned long t2count = 0;
int millistimer = 0;

ISR(TIMER0_COMPA_vect){
    t0count++;
}

ISR(TIMER2_COMPA_vect){
    t2count++;
}

void enableGlobalInterrupt(){ //enable global interrupt
    SREG |= (1<<7);
}

void disableGlobalInterrupt(){
    SREG &= ~(1<<7);
}
```

Here I used ISR(), which is the Interrupt Service Routine function of AVR’s interrupt.h

Enabling and disabling the global interrupts
What this function does is to setup timer0 to compare mode, along with its prescalers and comparevalue. I also wrote the same function for timer0. This is useful because we are going to use these timers as “timers” to get the time in milliseconds, which is the code below (the function “millis()”).

Since the internal clock of the Arduino UNO is 16MHz, to get 1kHz (period of 1 millisecond), we need to divide it by 16,000; which is 64*250
**Pulse Width Modulator (PWM)**

I used timer 0 as a source of PWM; since I am only controlling two motors, using one timer is sufficient.

```c
void setupTimer0PWM(){
    TCCR0A |= (1<<WGM00) | (1<<WGM01) | (1<<COM0A1) | (1<<COM0B1); // Fast PWM mode - non-inverting
    TCCR0B |= (1<<CS00); // no prescaler - to get the maximum refresh rate (changing PWM)
    OCR0A = 0; // starts with 0% duty cycle
    OCR0B = 0; // same
}
```

After setting up the timer, changing the OCRxn values would equal to changing the duty cycle of the PWM.

Here, uncreatively, I name the PWM function “analogWrite”

```c
int analogWrite(int pin, int dutycycle){ // write PWM signal onto the PWM pins

    // first we check if the duty cycle is between 0 and 255 (2^8)
    if(dutycycle < 0) dutycycle = 0;
    if(dutycycle > 255) dutycycle = 255;

    switch(pin){
        case 3: OCR2B = dutycycle; break;
        case 5: OCR0B = dutycycle; break;
        case 6: OCR0A = dutycycle; break;
        case 9: OCR1A = dutycycle; break;
        case 10: OCR1B = dutycycle; break;
        case 11: OCR2A = dutycycle; break;
        default: return LOW;
    }

    return dutycycle;
}
```
Universal Asynchronous Receiver/Transmitter (UART)
The ATMega328P has a UART module that communicates via the Rx and Tx pins on the Arduino board.
First I wrote a function to setup the UART module using an input baudrate.

```c
void UARTSetup(unsigned int baudrate){
    unsigned int ubrr = 16000000/16/baudrate - 1;
    UCSRB |= (1<<RXEN0) | (1<<TXEN0);
    //set the UART baudrate registers
    UBRRL = (unsigned char)(ubrr); 
    UBRRH = (unsigned char)(ubrr>>8);
    //set frame format, here we use 8 bit for the data, because the HC-05 module sends a byte everytime
    //we also set the number of stop bits to be 2
    UCSRC |= (1<<UCSR0B) | (0b00000011); //UART control and status register C
}
```

```c
int UARTReadyToSend(){
    if((UCSR0A & (1<<UDRE0)) == (1<<UDRE0) ) return 1; return 0;
}
```

```c
int UARTreceived(){
    if((UCSR0A & (1<<RXC0)) == (1<<RXC0) ) return 1; else return 0;
}
```

```c
int UARTtransmitted(){
    if((UCSR0A & (1<<TXC0)) == (1<<TXC0) ) return 1; else return 0;
}
```

The UART module of the ATMega328P is actually really easy to use. It has a register called `UCSR0A` that we can use to check the current state of the UART module (is a new byte ready to be sent? Is a new data byte received?)

And when it comes to receiving/sending bytes of data, there is a hand `UDRO` register to use in both cases.

```c
unsigned char UARTGetdata(void){
    return UDRE;
}
```

```c
void UARTRTF(char data){
    while( !UARTReadyToSend() );
    UDRE = data;
}
```

```c
void UARTRTSC(char str[]){
    int i = 0;
    while(str[i] != 0x00){
        UARTRTF(str[i]);
        i++;
    }
    while(!UARTtransmitted());
}
```
Inter-Integrated Circuit (I2C)
I would say making the I2C protocol work is probably the hardest part of doing this project. It is because there is no physical way you can test individual functions in your I2C library without running a full send/receive data process.

The protocol (according to the ATmega328P’s datasheet) for receiving data in Master mode is as follow:
In short, to receive the data bytes from the MPU-6050, we must do this process:
Send a START condition -> Send the Slave (MPU) address in WRITE mode -> Send the address of the device/measurement data on the Slave -> Send another (repeated) START condition -> Send the Slave address in READ mode -> Get the data bytes with Acknowledge -> Get the last data byte with Not Acknowledge -> Send a STOP condition

Here is the function for doing that:

```c
uint8_t i2c_read(uint8_t slaAdr, uint8_t devAdr, uint8_t length, uint8_t *buffer) {
    if(length == 0) return 1; //check if length is zero
    if(i2c_sendStart()) return 1; //send a start condition
    if(i2c_sendChar( slaAdr<<1 | 0x00 )) return 1; //send the slave address as WRITE
    if(i2c_sendChar(devAdr)) return 1; //send the device address
    if(i2c_sendStart()) return 1; //send a repeated start condition
    if(i2c_sendChar( slaAdr<<1 | 0x01 )) return 1; //send the slave address as READ
    for(uint16_t i = 0; i < (length-1); i++) {
        buffer[i] = i2c_getChar_ack();
    }
    buffer[(length-1)] = i2c_getChar_nack();
    i2c_sendStop();
    return 0;
}
```

I will not go through the details of each functions and how they work because they are very long and complicated (and would add 3-4 pages to this document). So I will include my libraries, as well as any codes (fully commented) in the Appendix; as a GitHub repository.
II. Testing individual modules

To test the I/O Ports, timers and PWM, I used LEDs to flash, blink and change their brightness to verify.

To test the UART/Bluetooth, I used a Bluetooth terminal app downloaded from the internet to send/receive data (characters). And then I use a well-tested UART to test the I2C module (reading values from the MPU).

Here you see both in action.
III. Building the robot

The robot has 3 levels: the lowest one for the motors, the middle one for the Arduino, MPU and HC-05, and the highest one for the battery. It is made out of MDF and then laser-cut.
IV. Dynamics and Control

The robot is an inverted pendulum, with the mass of 1.036kg and its center of gravity is approximately 0.133m from the rotational axis (the axis of the motors).

Its equation of motion with regards to the torque Tau is: \( I \ddot{\theta} - mgh \sin \theta = \tau \)

And from that I derived the open-loop transfer function to be:

\[
\frac{1}{6.1 \times 10^{-3} s^2 - 1.35}
\]

Which means it has 2 poles at -14.88 and 14.88; this is very unstable system.

I designed a PID Controller to stabilize the robot, with the transfer functions block diagram below:
Implementing the PID Controller in Embedded C

```c
gyro = (double)(y_gyro-GYROOFFSET)*GYROCONST; //convert the gyroscope value into deg/sec
angle -= (gyro*interval)/1000; //sum (Integral) of deg/sec into deg
err = (angle - offset); err_sum += (err*interval)/1000; err_div = ((err - err_old)/interval)*1000;
err_old = err;
```

```c
dutycycle = (int) (Kp*err + Ki*err_sum + Kd*err_div); //PID CONTROLLER
if(dutycycle > 0) robotforward();
if(dutycycle < 0) robotbackward();
```

One important note from the code above: why am I using the gyroscope values (change in angle [deg/s]) to calculate the angle and not the accelerometer values? A: The accelerometer measurements will give wrong angle values when the robot is moving (adding to the acceleration).

After trying different methods of tuning the Kp, Ki and Kd values, I arrived at the final values of Kp = 170, Ki = 30 and Kd = 3; which keep the robot stable enough and can response to step inputs of reference value quickly (i.e. when being controlled via Bluetooth).
As I mentioned before, the GitHub link to the full program including libraries will be in the Appendix.
VI. The app

I used MIT’s App Inventor to write an Android app to control the robot via Bluetooth.
Balancing on different surfaces
Appendix A: GitHub link to my codes: https://github.com/dinhtunglhp/tonybalancebot
Appendix B: Project video: http://tiny.cc/TonyBotVideo
Appendix C: Rough calculations work
Through the process of working on this project, I have learned a lot. The most valuable skill that I have now I acquired, in my opinion, is the ability to read and grind through the datasheet. The world of microcontrollers was mystifying to me before I started learning Mechatronics 1 and did this project. Now I can understand how they work, and I have gained confidence that I will be able to learn how other microcontrollers work as well from now on.

The most challenging part while doing the project was testing out functions for modules that I wrote, especially the I2C one. Because ATMEL Studio does not have a “Serial Monitor” as the Arduino IDE does, it was tough to debug. However, once I have my UART and Bluetooth set up, I could use it as a way to debug my other codes.

This was a fun experience, and I cannot wait to study future subjects and doing even more challenging projects.

A few words on ScrapTheMxK
I think that studying Mechatronics in a studio-based program is an excellent opportunity to learn more than what a typical learning experience can offer. Being able to set out your own goals, doing what you enjoy and learn something along the way is a spectacular process.
The hardware and software that we use are also better than the MxK, Xilinx, and MPLAB in my opinion.
The only thing I think should be changed is the AVR part of the final exam. Everyone had their own goals, so the modules of the ATMega328P that they studied and used were also different. It would be better if in the final exam we had the option to choose the questions (question packs?) that we wanted to answer. For example, I spent most of my time looking at the UART and I2C module; but did not use the ADC.
Overall, ScrapTheMxK was a wonderful experience, and so I would like to thank everyone involved in this project.