Homework 9: Software Design Considerations

Due: Friday, March 21, at NOON

Team Code Name: ____Two Wheel Deal___________________________ **Group No.** __12__

Team Member Completing This Homework: ____Eric Geier____________________________

e-mail Address of Team Member: __edgeier_____ **@ purdue.edu**

NOTE: This is the last in a series of four "design component" homework assignments, each of which is to be completed by one team member. The completed homework will count for 20% of the individual component of the team member's grade. The body of the report should be 3-5 pages, **not** including this cover sheet, references, attachments or appendices.

Evaluation:

SCORE	DESCRIPTION
10	***Excellent*** *– among the best papers submitted for this assignment. Very few corrections needed for version submitted in Final Report.*
9	***Very good*** *– all requirements aptly met. Minor additions/corrections needed for version submitted in Final Report.*
8	***Good*** *– all requirements considered and addressed. Several noteworthy additions/corrections needed for version submitted in Final Report.*
7	***Average*** *– all requirements basically met, but some revisions in content should be made for the version submitted in the Final Report.*
6	***Marginal*** *– all requirements met at a nominal level. Significant revisions in content should be made for the version submitted in the Final Report.*
*	***Below the passing threshold*** *– major revisions required to meet report requirements at a nominal level.* ***Revise and resubmit.***

* *Resubmissions are due within* **one week** *of the date of return, and will be awarded a score of "6" provided all report requirements have been met at a nominal level.*

Comments:

1.0 Introduction

The Two Wheel Deal is a personal transportation vehicle based on the control theory principle of the inverted pendulum [1]. The inverted pendulum is an inherently unstable system that consists of a mass atop a rod that is hinged to a cart. In order to stay upright a force is applied to the cart in a linear direction so that the cart is always under the system's center of gravity. This is usually accomplished using some sort of feedback control. With this principle in mind, there are a few major software design considerations.

The first and most important consideration is to ensure that there is a constant, correct, and efficient reading of analog outputs from the accelerometer, gyroscope, joystick, and battery. This data determines how the system needs to respond to keep the vehicle stable. Another major design consideration is to accurately determine the angle of tilt from vertical that the vehicle has sustained. This way the proper motor speed is obtained to keep the base of the vehicle under the center of gravity. One final major design consideration is to use a quick and efficient method of control that will not allow a lot of overshoot of the vertical angle yet will be fast enough to keep the vehicle upright. One other design consideration is to display important information to the rider such as battery life and other operating information.

2.0 Software Design Considerations

2.1 Memory Mapping

The microcontroller [2] that was chosen has 32 kB of self-programmable Flash memory. The Flash memory is addressed from $0000 to $3FFF. All static data which includes the Two Wheel Deal program as well as any constants will be stored in Flash memory. The current size of the program is about 8 kB and the final program size should be about 10 kB. This will leave plenty of space left to add additional features or functions. The microcontroller has 2 kB of SRAM and it is located from $0000 to $085F. The general purpose working registers make up the first 32 addresses in the SRAM space. The next 64 addresses are for the I/O registers and will be where the variables are kept. The stack will start at the bottom address $085F, and it will work its way up as items are added or back down as items are removed. No heap space will be used since all variables are declared when compiled.

2.2 Mapping of External Interfaces

There are a large number of components that are interfaced to the microcontroller with every port being used. Information such as the address [3], the function, and the peripheral, if applicable, of every port that is used can be found in Table 1 of Appendix C.

2.3 Utilization of Integrated Peripherals

The Two Wheel Deal utilizes the following peripherals: ADC, PWM, and TIM. The interrupt driven ADC is used to read the values from the sensors at a constant time interval. The 16-bit PWM is used to provide a signal to the motor controller that corresponds to the appropriate output torque of the each motor. The duty cycle of the PWM is based on the angle of the vehicle from vertical. The timer is used to create an accurate time base in which to run the main program.

The analog to digital register ADMUX is set so that the ADC uses AREF which is set equal to 5 V, the results are left adjusted so that the result can be read in one byte, and the ADC is initially set to start on channel 0. The register ADCSRA is set so that the ADC is enabled, there is no auto triggering, ADC interrupts are enabled, there is a clock prescalar of 128, and the ADC conversions are started. The PWM register TCCR1A is set so that on compare match OC1A/OC1B are cleared and force output compare on A/B disabled. The register TCCR1B is set so the input noise canceller is disabled, a falling edge is used for input capture, fast PWM mode is used with a top value of ICR1, and there is no clock prescaling. The register ICR1 is set so that the PWM frequency is 2 kHz. The timer register TCCR0 was set so that there was no force output compare, it used Clear Timer on Compare (CTC) mode, there was normal port operation, and a clock prescalar of 64 making the timer run at 250 kHz. The register OCR0 was set equal to 250 so that the timer interrupt service routine would run every 1 ms. Finally register TIMSK was set to allow output compare interrupts but not overflow interrupts. These register values can be viewed in Table 2 located in Appendix C.

2.4 Overall Organization of Application Code

The overall organization of the application code is a hybrid or flag driven program. This means that the main loop of the program is a simple “round robin” loop that checks to see if certain flags are set. Those flags are set by interrupt service routines that occur at a fixed and known rate. The rationale behind this is since the rider will most likely be in constant motion, it must be ensured that the controller will check the tilt angle and update the motor torque at a fast and constant rate. This is best performed using an interrupt that occurs at a known interval and increases a counter each time it is executed. Once enough interrupts have occurred a flag is then set in the "round robin" main loop. Since this main loop is small and only checking three flags, it can be assured that the balancing algorithm function will be run at a precise rate.

2.5 Debugging Provisions

There is nothing specific that has been included in the code that deals only with debugging. The microcontroller is programmed using the SPI and an inline programmer. The LCD has been used as a terminal to help with debugging code. Since it was the first interface that was programmed, it has been used to display register, variable, and port information to help locate sources of error in the software.

3.0 Software Design Narrative

The following sections include a detailed narrative of each module used throughout the Two Wheel Deal software. Each module is linked to allow the actual code to be viewed online. Also, a hierarchical arrangement of the code modules including the functions included in them can be viewed in Appendix B.

3.1 Main Module (main.c)

This module contains the "round robin" polling loop as well as the ADC and TIM interrupt service routines. The main loop serves as an endless polling loop that checks if the tens (LCD), huns (balance), and ones (battery) flags are set. If one of them is then it calls that function otherwise is continues looping. This module has been written, programmed, and verified to work correctly. The flowchart illustrating the activity of the main loop can be viewed in Figure 1 of Appendix A.

The first operation of the main loop is to call the function ioinit() which sets the data direction registers for Ports A, C, and D. It also calls varinit() which initializes the battery filter array so that the battery does not read 0 V when powered up. It then enables interrupts and calls the function PERinit() which initializes all the peripherals and interfaces. This includes the TIM, PWM, ADC and the LCD. In that function it calls lcdcch() which loads custom characters into the LCD driver. Next it prints an introduction greeting on the LCD for 2 seconds using print_greeting() and delay_ms() and then clears the display. Finally it enters the endless while loop where it checks the huns and tens flags repeatedly.

If the huns flag is set then 10 ms has passed and the balancing algorithm function is called. If the tens flag is set then 100 ms has passed and the LCD update function is called.

Finally if the ones flag is set then 1 second has passed and the battery filtering function is executed. Each of these functions is described in detail later.

The ADC ISR is used to continually check and update the outputs from the accelerometer in the X and Y directions, the gyroscope, the joystick, and the battery voltage divider. The flowchart describing the routine activity can be found in Figure 2 in Appendix A. It works by waiting until the ADC interrupt bit is set which means a conversion has been completed, and then it reads the ADC data register result from the completed conversion and stores it in the corresponding spot in a data array. It then increments the ADC channel counter, resets the ADMUX register for that channel, starts the ADC conversion, and then leaves the routine.

The TIM ISR is used to keep a continuous time base for the system and the flowchart can be viewed in Figure 3 of Appendix A. It works by waiting until the OCIF bit is set indicating that the timer counter equals the compare register. It then sets the 1 ms flag and increments the hundredths, tenths, and ones counters by one. Then if the hundredths counter equals 10 the huns flag is set and the hundredths counter reset. The same occurs with the tenths counter except it must equal 100, the tens flag is set, and the tenths counter reset. Finally if the ones counter equals 1000 then the ones flag is set and the counter is reset. The routine is then exited.

3.2 LCD Update Module (lcd.c)

This module contains the lcd_update() function which is called by the main function to update information displayed on the LCD every tenth of a second. This module has been written, programmed, and verified to work correctly. The flowchart for this function can be viewed in Figure 4 of Appendix A. The first operation performed by lcd_update() is to read the filtered battery level variable and determine how many full and empty blocks are necessary for updating the custom battery symbol.

Then the cursor is positioned at the first character of the first line using lcd_rc(rows,columns). This function uses the inputted rows and columns values and sets the corresponding cursor address using the lcd_addr() function which simply writes a command to the LCD driver. Next the strings “Angle:” and “Battery:” are written to the LCD using lcdstr(). This uses a character array and a pointer to display a string character by character on the LCD until a null character is reached. Then the cursor is repositioned on the second line and the angle is displayed using lcdnum(). This function is used to display an inputted float number to two significant digits. This is done by casting the original float to an int. The int is then displayed using lcdrite() which simply writes data be displayed on the LCD. The float and int are then multiplied by 100 and subtracted from each other. This remainder is the decimal portion that is then displayed on the LCD using lcdrite().

Next the cursor is repositioned on the third row and displays the left side of the custom battery symbol. Then using the number of full and empty blocks found earlier displays the filled and empty parts of the battery then displays the right side of the battery. This is done again on the fourth row to complete the symbol. Finally the function is exited to return to the main loop.

3.3 Battery Filter Module (bat.c)

This module contains the battery_alg() function which is called by the main function to average the values read by the ADC from the battery voltage divisor to determine the voltage left in the batteries. The flowchart is located in Figure 5 of Appendix A. It is called every second to read a new value from the ADC and determine the new average which is used when updating the LCD. The reason a filtered battery value is used to display the voltage left on the batteries is because if there are power spikes or dips due to increasing or decreasing the motor speed this will not be displayed to the rider. This module has been written, tested, and verified to work correctly. The first operation the function performs is to read value from the ADC and store it in the battery filter array. It then increments the array counter and uses a while loop to sum the array. It then averages the value and converts to volts to be used in the LCD update function.

3.4 Balance Update Module (bal.c)

This module contains the balance_alg() function which is called by the main function to determine the angle of tilt from vertical and update the motor controller PWM signals and therefore the motor torque accordingly. The function executes every hundredth of a second. This module has not been tested but is partially written and is laid out in pseudocode. The logic and activity of this function was based on open source pseudocode provided by Trevor Blackwell [4] as well as open source code provided by a group of MIT/Wayland High School students [5] while designing a similar transportation vehicle. The flowchart can be viewed in Figure 6 of Appendix A. The first operation performed is to read the ADC values for x axis accelerometer, y axis accelerometer, gyroscope, and joystick. Then a constant is subtracted from the x and y axis values to account for the positioning of the accelerometer. The angle in degrees is found by taking the arctangent of the y axis divided by the x axis then converted from radians.

The balancing torque is then found using a Proportional-Derivative Controller. The angle is multiplied by an appropriate constant and added to the rate multiplied by an appropriate constant. These two values represent the proportional and derivative values respectively. The balancing torque value is then set as the base motor value for the rest of the algorithm. Turning is then accounted for by first reading the value from the joystick and subtracting a constant from it to make a left turn a negative number and a right turn a positive number. This is then subtracted from the base motor value so that if a left hand turn is needed the left motor slows and vice versa for a right hand turn. Finally the motor value for the left and right side motors are written to the left and right PWM duty registers respectively, and the function is exited.

4.0 Summary

Those are the major design considerations for the Two Wheel Deal as well as the software design narrative describing the overall code functionality. One final note is that as of right now there are no safety functions or considerations taken into account. These are currently out of the scope of the project, but if there is time after the main project specific success criteria are completed, then they will be added as needed.

List of References

[1] Engineering at University of Michigan, “Modeling an Inverted Pendulum”. [Online]. Available: http://www.engin.umich.edu/group/ctm/examples/pend/invpen.html. [Accessed: Mar. 19, 2008].

[2] Atmel Corporation, “ATmega32/32L Microcontroller”. [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf. [Accessed: Mar. 19, 2008].

[3] Atmel Corporation, “AVR505: Migration between ATmega16/32 and Atmega164P/324P/644P”. [Online]. Available: http://www.atmel.com/dyn/resources/prod_documents/doc8001.pdf. [Accessed: Mar. 19, 2008].

[4] Trevor Blackwell, “Building a Balancing Scooter”. [Online]. Available: http://tlb.org/scooter.html. [Accessed: Mar. 19, 2008].

[5] Massachusetts Institute of Technology, “The DIY Segway”. [Online]. Available: http://web.mit.edu/first/segway/. [Accessed: Mar. 19, 2008].

Appendix A: Function Flowcharts

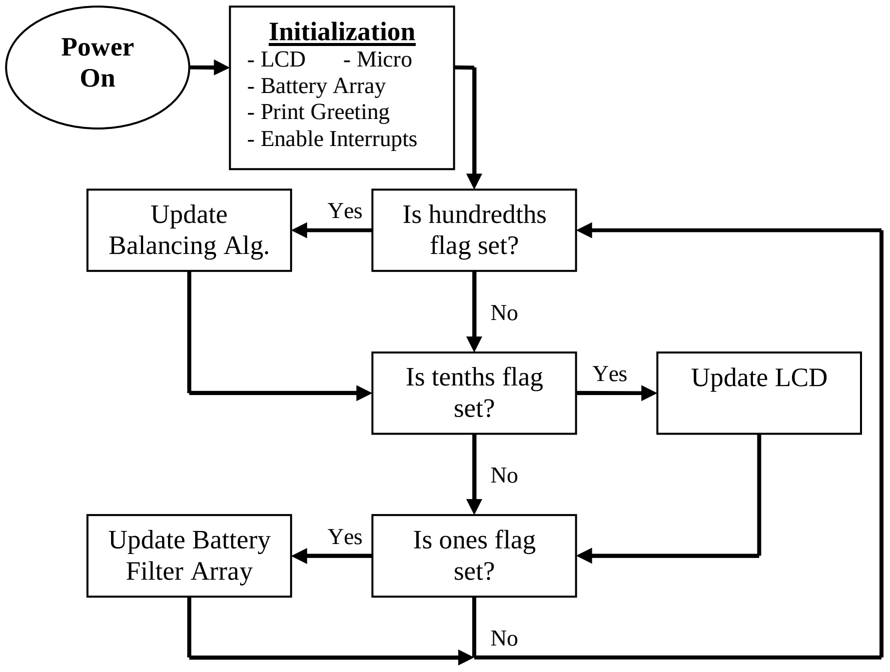
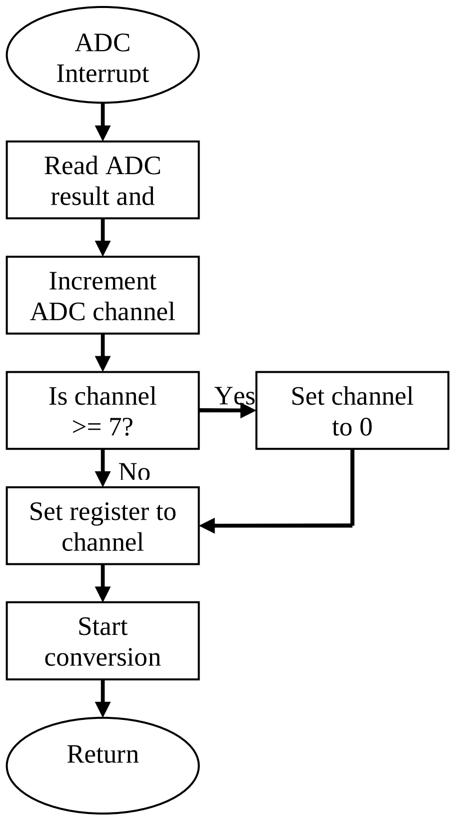


Figure 1: Main Loop Flowchart



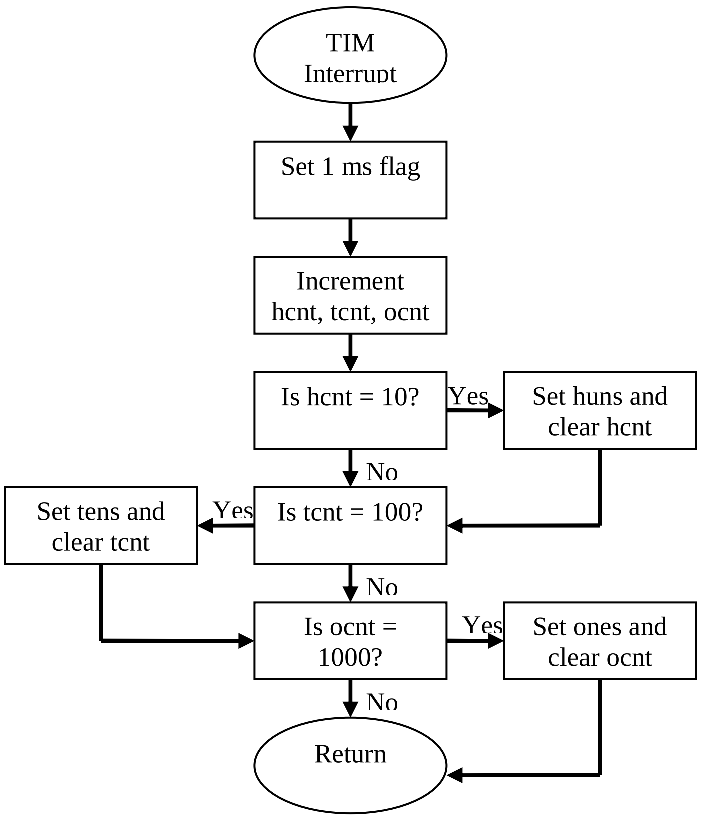


Figure 2: ADC_ISR Flowchart **Figure 3: TIM_ISR Flowchart**

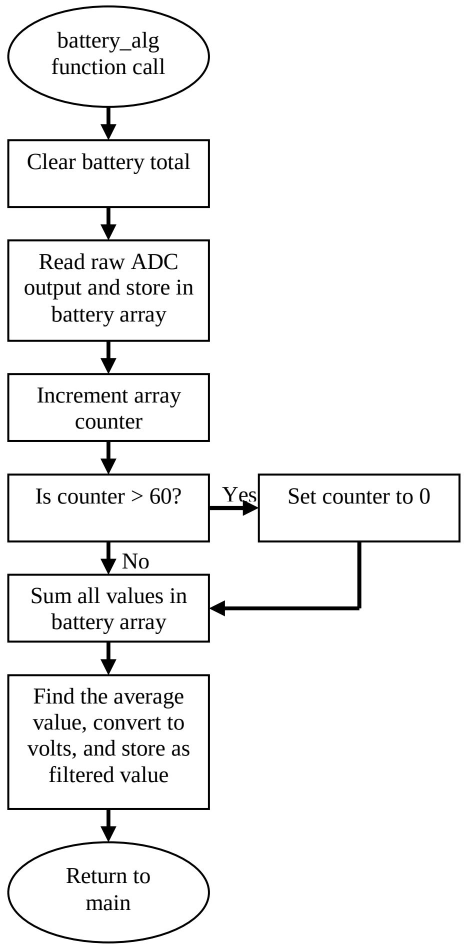

battery_alg function call
Clear battery total
Read raw ADC output and store in battery array
Increment array counter
Is counter > 60?
Yes
Set counter to 0
No
Sum all values in battery array
Find the average value, convert to volts, and store as filtered value
Return to main

Figure 4: Battery Function Flowchart

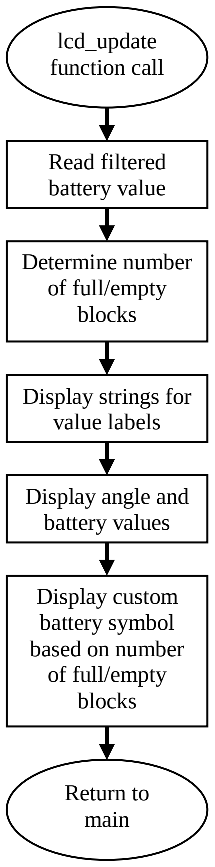

lcd_update function call
Read filtered battery value
Determine number of full/empty blocks
Display strings for value labels
Display angle and battery values
Display custom battery symbol based on number of full/empty blocks
Return to main

Figure 5: LCD Function Flowchart

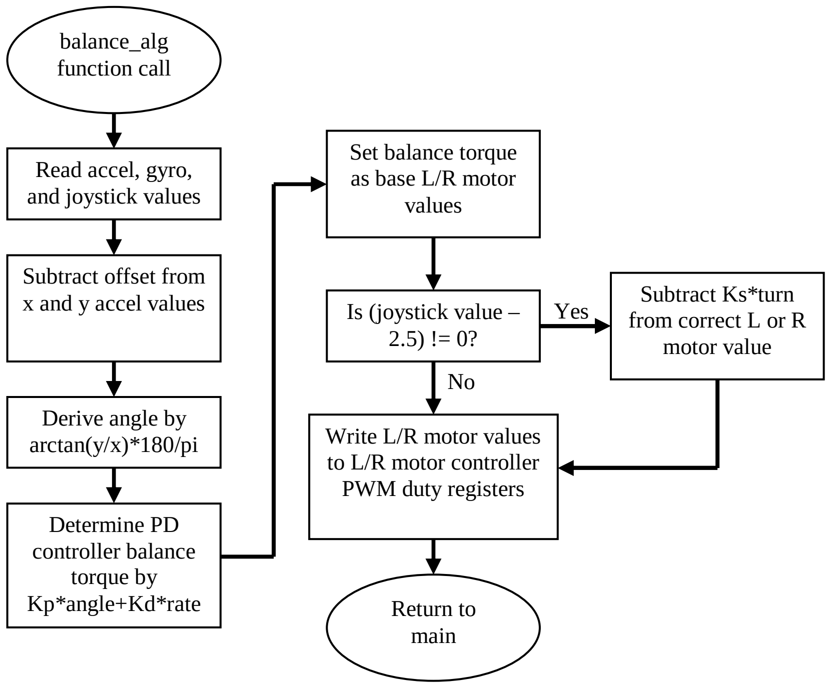
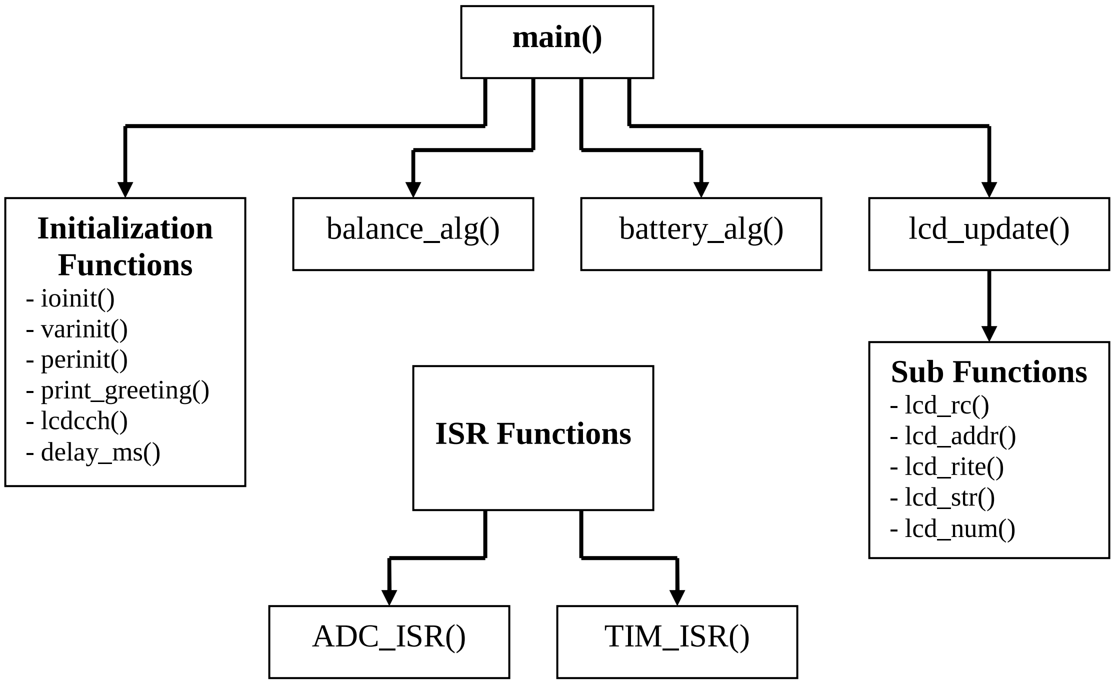

balance_alg function call
Read accel, gyro, and joystick values
Subtract offset from x and y accel values
Derive angle by arctan(y/x)*180/pi
Determine PD controller balance torque by Kp*angle+Kd*rate
Set balance torque as base L/R motor values
Is (joystick value – 2.5) != 0?
Yes
Subtract Ks*turn from correct L or R motor value
No
Write L/R motor values to L/R motor controller PWM duty registers
Return to main

Figure 6: Balance Function Flowchart

Appendix B: Hierarchical Block Diagram of Code Organization



Appendix C: Tables

Register Name	Address	Peripheral	Function
PORTA	$3B	ADC/GIO	PA0 – Battery Level PA1 – Accelerometer X Axis PA2 – Accelerometer Y Axis PA3 – Gyroscope Rate PA4 – GIO PA5 – Joystick Value PA6 – GIO PA7 – Not Used
PORTB	$38	SPI/GIO	PB0 – GIO PB1 – GIO PB2 – GIO PB3 – GIO PB4 – GIO PB5 – MOSI PB6 – MISO PB7 – SCK
PORTC	$35	LCD	PC0 – LCD Data Bit 0 PC1 – LCD Data Bit 1 PC2 – LCD Data Bit 2 PC3 – LCD Data Bit 3 PC4 – LCD Data Bit 4 PC5 – LCD Data Bit 5 PC6 – LCD Data Bit 6 PC7 – LCD Data Bit 7
PORTD	$32	LCD/PWM/GIO	PD0 – Right Motor Controller F/R Pin PD1 – Left Motor Controller F/R Pin PD2 – Not Used PD3 – Not Used PD4 – Right Motor Controller PWM PD5 – Left Motor Controller PWM PD6 – LCD RS (Command/Data) Pin PD7 – LCD E (Clock) Pin

Table 1: Port and Pin Descriptions

Peripheral	Register Values
ADC	ADMUX = 20h, ADCSRA = CFh
16-bit PWM	TCCR1A = A2h, TCCR1B = 19h, ICR1 = 01EAh
8-bit TIM	TCCR0 = 0Bh, OCR0 = FAh, TIMSK = xxxxxx10b

Table 2: Peripheral Register Initializations