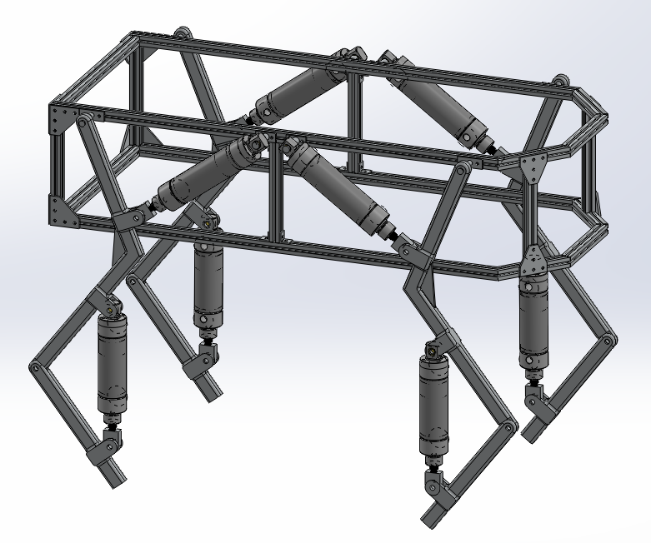
http://www.joyglobal.com/clientCSS/images/logo_JoyGlobal.png

**Implementation of an Agile Educational Robot**



**May 21, 2015**

Logan Beaver

Justin Campbell

Tyler Paddock

Ronald Shipman

***Advisor***: Dr. Luis A. Rodriguez

Contents

[Executive Summary 2](#_Toc419987880)

[Chassis Construction 3](#_Toc419987881)

[Pneumatic System 3](#_Toc419987882)

[Electrical System 3](#_Toc419987883)

[Control System 7](#_Toc419987884)

[Results and Future Work 8](#_Toc419987885)

[References 8](#_Toc419987886)

[Appendix I – XboxInterface.dll 8](#_Toc419987887)

[Appendix II – Java Application – Human Machine Interface 11](#_Toc419987888)

# Executive Summary

The Milwaukee School of Engineering (MSOE) participates in Science Technology Engineering and Mathematics (STEM) outreach events for prospective students. The school will benefit greatly from having a sophisticated robotic control system to build excitement about STEM as well as sparking interest in fluid power, automation, and the controls fields. An agile pneumatic robot is not only a complicated control system that can be used to get young people excited about STEM, but it will also increase the prestige of MSOE knowing that a group of seniors attending the school were able to design and build the system from the ground up. In addition it also provides an exciting opportunity for future groups to iterate on the design and integrate new and exciting features.

Implementation of the agile educational robot began. From the design quarter constraints were determined. The most important constraints are listed below:

* A maximum weight of 35 kg for portability
* Maximum size of 0.75 m x 0.75 m x 1.0 m box for portability
* Custom debug panel creation to facilitate troubleshooting
* MATLAB and Simulink model support to allow mechanical engineering students to update control algorithms without knowledge of C/C++
* Electronic fuses and shielding to protect the robot and operator during use and maintenance
* Mechanical protection to reduce the risk of pinching and self-collision damage to the robot
* An easy to access emergency stop to quickly depower the robot
* A pressure relief valve to reduce the risk of overloading and damaging pneumatic components

The design work done on this project was a continuation of the work done by Kevin Lee during the Research Experience for Undergraduates (REU) at MSOE. His work involved deriving a dynamic model for a simplified quadruped robot. Now with our team’s design work finished the implementation of this walking robot was conducted. All necessary components were ordered, tested, and assembled. Pneumatic components were mostly provided by the vendor Numatics. Numatics valves and cylinders are used in the implementation. From the previous design quarter a dog like robot was offered as the final design. The chassis of this dog-like robot was constructed out of T-slotted aluminum framing. The legs of the robot constructed out of solid aluminum. Electrical components were fabricated onto prototyping boards and tested. Software was written in Java, Windows C code, and MATLAB Simulink. The user interface was implemented in java, the controller driver was interfaced with c code dll, and the microcontroller code was written in Simulink block diagram logic.

# Chassis Construction

# Pneumatic System

# Electrical System

The electronics of the robot are broken up into two major subsystems, the motherboard and the debug panel. The motherboard was designed to contain the auxiliary electronics and signal conditioning components needed for the robot. The debug panel contains all necessary electronics to display battery levels and other statuses of the robot.

To power the electronics two different sets of batteries are used. A 9 volt battery is used to power the microcontroller. The valves are powered by multiple 12 volt batteries connected in series. The valves run on 24 volt power and the signal condition circuits use 12 volt power.

The debug panel subsystem contains a physical panel with light emitting diodes (LED) and connections for banana plug cables. The LEDs are used to show battery levels and the status of the robot. The banana plug connectors are used to interface to Milwaukee School of Engineering’s test equipment in the labs. Banana plugs are used because they are standard on test equipment. A USB slot is also included on the debug panel to assist in programming the microcontroller without removing it in the robot. The following figure shows a brief layout of the debug panel components.

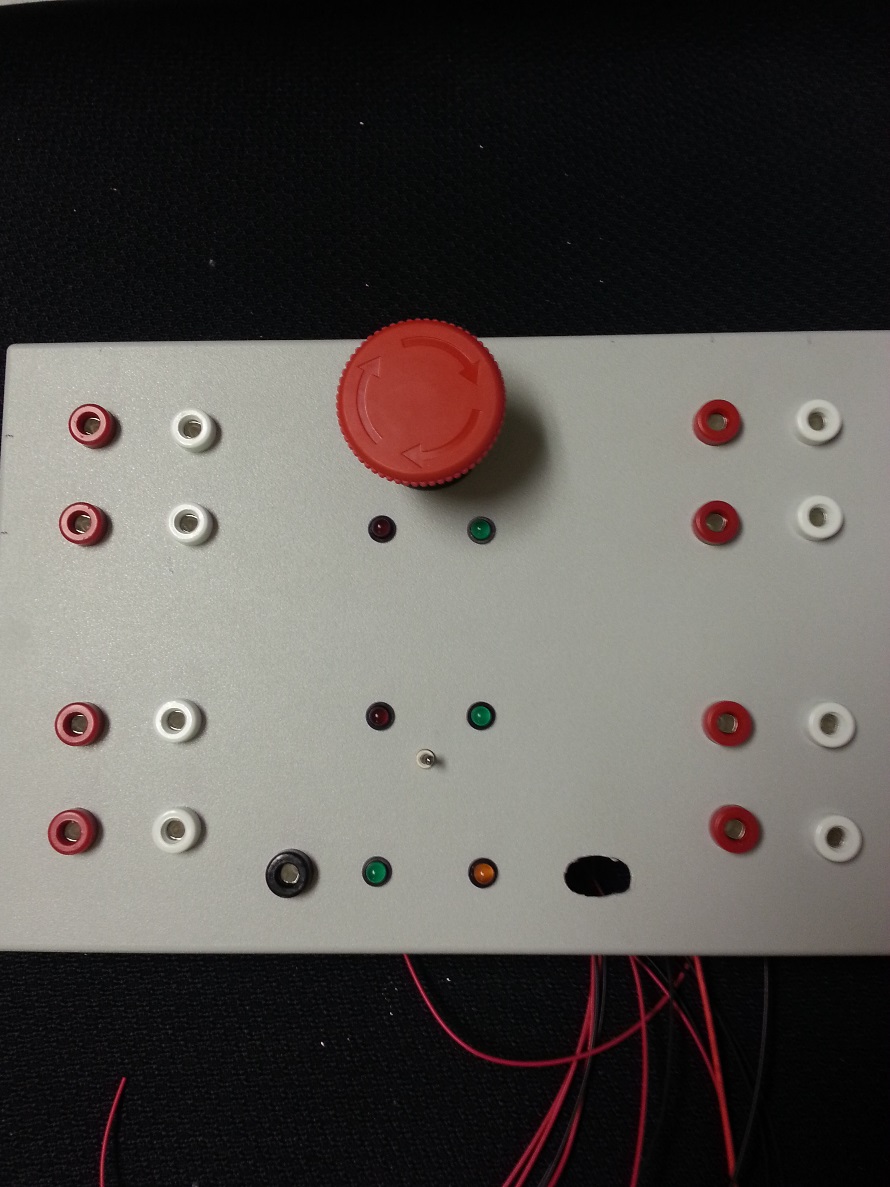


Figure 1: Debug Panel Component Layout

A nine volt battery level indicator was created using a 6.2 zener diode and two transistors. The circuit drives two different color LEDs to indicate if the voltage needs to be replaced. If the green LED is on then the battery is at a good level. Once the green LED goes out the operator should replace the nine volt battery. If the red LED goes out the microcontroller operation is not guaranteed to be successful based on the remaining batter voltage. At this condition the nine volt battery needs to be replaced.

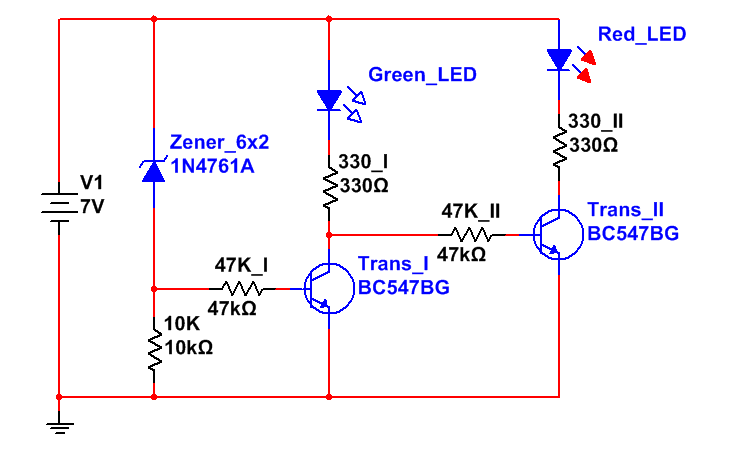


Figure 2: 9 Volt Battery Level Indicator

The battery level indicator is needed because of the discharge curve of a nine volt battery. The battery maintains close to its specified level and then drops quickly. The indicator allows the operators to change the battery at an appropriate time. The nine volt battery discharge curves are shown below in figure 3.

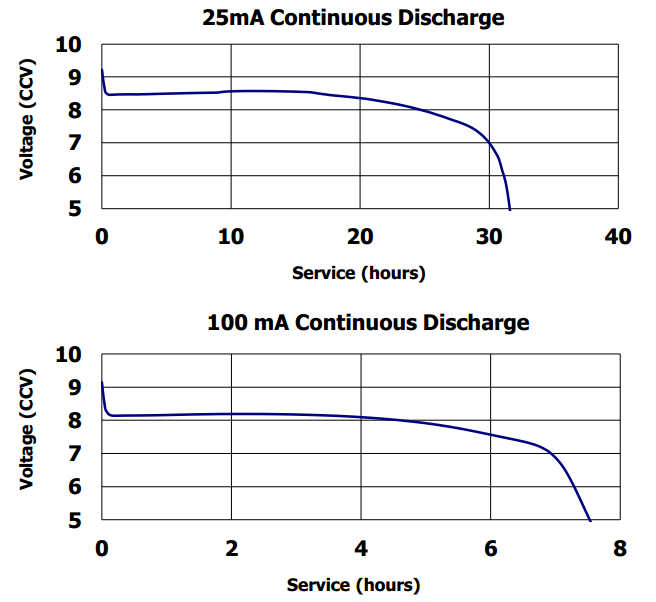


Figure 3: Energizer 9 volt battery discharge curve [1]

From the previous quarter a low pass filter was designed to convert the pulse-width modulated outputs of the Arduino microcontroller into analog signals. This quarter the filters were implemented on prototyping boards. Each board has two third order low pass filters and doubling amplifiers. Each board has seven connection points. The connections include two inputs and two outputs a positive rail and negative rail for the amplifiers and a ground connection. The layout of the prototype board is shown in figure 4.

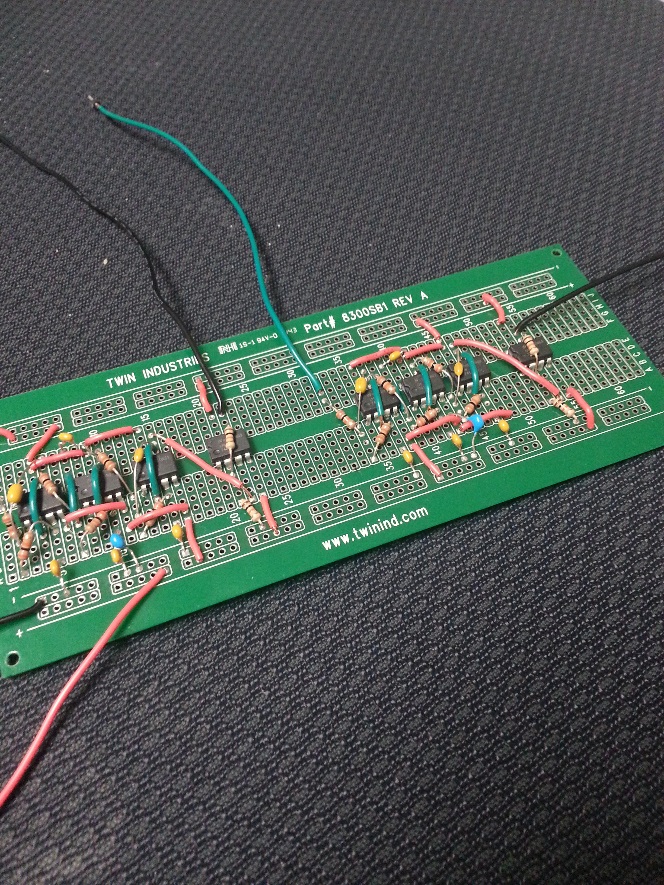


Figure 4: Prototyping Board with LPF and Amplifier

A few changes were made from the initial design. The capacitor values of the filter were changed to be standard capacitor values. The circuit diagram of a single filter and amplifier is shown in figure 5 and 6.

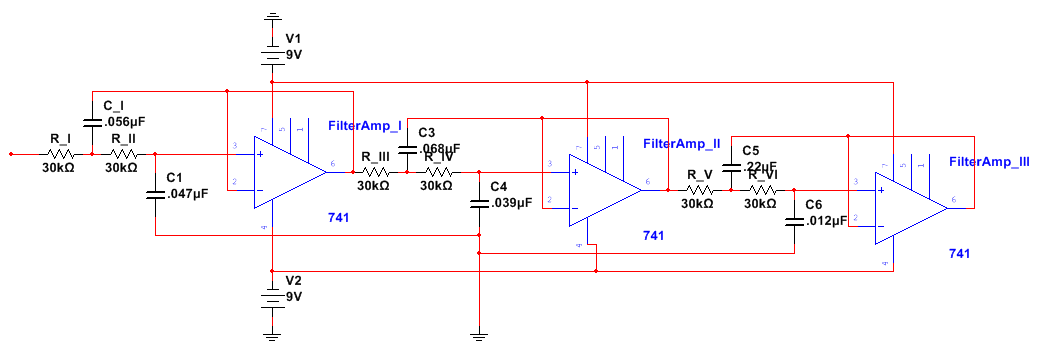


Figure 5: Low Pass Filter Layout

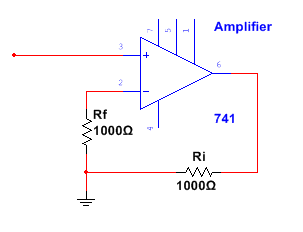


Figure 6: Amplifier Layout

The low pass filter converts the microcontroller PWM into an analog signal. The max value of this analog signal is 5 volts. The scaling amplifier afterwards increase the filter output by a gain of two.

In since the batteries are not truly 12 volts on each battery the voltage could be higher than needed. A voltage regulator circuit is used to limit the voltage to 24 volts. The circuit is taken directly from the voltage regulator datasheet. Figure 6 is the image taken from the datasheet.

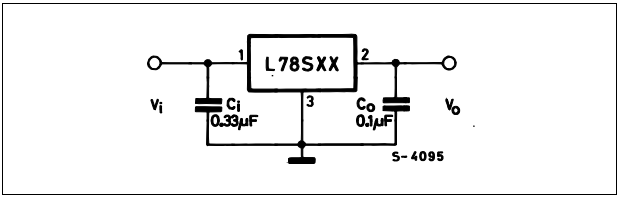


Figure 6: Voltage Regulator Circuit From Datasheet [2]

# Control System

The robot control system consists of two parts, a Simulink control architecture running on an Arduino Mega 2650 and a GUI developed in Java which runs on 64 bit windows. The Java application has two dependent libraries, Java Native Access (JNA) and RXTX. JNA is used to call native machine code to read from an xbox controller. The native code is the XboxInterface.dll which was a custom C code project completed by our team. The C code calls windows xbox driver code to read from the controller.

RXTX is a java library used to communicate with the serial ports on the PC. When the *connect* button is pressed in the GUI RXTX searches for available serial ports and connects to the first available to find the XBee radio transmitter/receiver. The java application calls the XboxInterface.dll to get a current controller conditions and identify a user request. The current user request is sent out the Xbee device to the robot’s receiver.

# Results and Future Work

The robot was constructed in subsystems. The electrical and control subsystems were tested throughout implementation and development. Upon completion of the first leg of the robot the cylinders were attached and driven using available servo valves. At this time the system was pressurized and connected into the filter and amplifier prototyping boards. The user interface was used to drive the leg forward and backward manually.

The next steps for this project include adding sensors to detect disturbances and tipping during walking motions. Using those sensors correction can be made to the gait of the robot. Additionally, advanced gaits and motions need to be programmed and tested.

One problem that occurred during the project was learning that the pneumatic valves are controlled by multiple solenoids after much of the design was completed assuming one solenoid. As a result the number of PWM outputs on the microcontroller need to be doubled to control the robot. Another result of this problem is that each leg needs twice as much signal conditioning circuitry.

# References

|  |  |
| --- | --- |
| [1] | Energizer, "www.energizer.com," [Online]. Available: www.energizer.com. [Accessed 26 2 2015]. |
| [2] | ST Microelectronics, "http://www.farnell.com," February 2003. [Online]. Available: http://www.farnell.com/datasheets/76145.pdf. [Accessed 20 May 2015]. |

# Appendix I – XboxInterface.dll

// dllmain.cpp : Defines the entry point for the DLL application.

// Author: Tyler Paddock

// Date: 4/01/2015

// Purpose: Interface with the XboxInput function on a windows machine

// Returns Button states from an xbox controller

#include "stdafx.h"

#include "XInput.h"

#include "Windows.h"

#include "math.h"

#include "XboxInterface.h"

BOOL APIENTRY DllMain( HMODULE hModule,

DWORD ul\_reason\_for\_call,

LPVOID lpReserved

)

{

return TRUE;

}

//Function Declarations

extern "C" \_\_declspec(dllexport) int getLeftJoyStick(int);

extern "C" \_\_declspec(dllexport) int getRightJoyStick(int);

extern "C" \_\_declspec(dllexport) int getButtonStates(int);

extern "C" \_\_declspec(dllexport) short getTriggerStates(int);

//The code implementation

#include "XboxInterface.cpp"

\_\_declspec(dllexport) int getLeftJoyStick(int controllerNumber){

DWORD dwResult;

XINPUT\_STATE state;

ZeroMemory(&state, sizeof(XINPUT\_STATE));

dwResult = XInputGetState(controllerNumber, &state);

//Declare retrun value

int retVal = 0;

if (dwResult == ERROR\_SUCCESS){

//Controller

retVal = (state.Gamepad.sThumbLX << SHORT\_SHIFT) | state.Gamepad.sThumbLY;

}

return retVal;

}

\_\_declspec(dllexport) int getRightJoyStick(int controllerNumber){

DWORD dwResult;

XINPUT\_STATE state;

ZeroMemory(&state, sizeof(XINPUT\_STATE));

dwResult = XInputGetState(controllerNumber, &state);

//Declare retrun value

int retVal = 0;

if (dwResult == ERROR\_SUCCESS){

//Controller

retVal = (state.Gamepad.sThumbRX << SHORT\_SHIFT) | state.Gamepad.sThumbRY;

}

return retVal;

}

\_\_declspec(dllexport) int getButtonStates(int controllerNumber){

DWORD dwResult;

XINPUT\_STATE state;

ZeroMemory(&state, sizeof(XINPUT\_STATE));

dwResult = XInputGetState(controllerNumber, &state);

//Declare retrun value

int retVal = 0;

if (dwResult == ERROR\_SUCCESS){

//Controller Connected

retVal = state.Gamepad.wButtons;

}

return retVal;

}

//short = byte(leftTrigger) + byte(rightTrigger)

\_\_declspec(dllexport) short getTriggerStates(int controllerNumber){

DWORD dwResult;

XINPUT\_STATE state;

ZeroMemory(&state, sizeof(XINPUT\_STATE));

dwResult = XInputGetState(controllerNumber, &state);

//Declare retrun value

short retVal = 0;

if (dwResult == ERROR\_SUCCESS){

//Controller Connected

retVal = ((state.Gamepad.bLeftTrigger) << BYTE\_SHIFT) | state.Gamepad.bRightTrigger;

}

return retVal;

}

#ifndef XboxInterface\_H

#define XboxInterace\_H

#include "stdafx.h"

#include "XInput.h"

#include "Windows.h"

#include "math.h"

#define SHORT\_SHIFT 16

#define BYTE\_SHIFT 8

//DWORD readControllerState(XINPUT\_STATE\*, int);

#endif

# Appendix II – Java Application – Human Machine Interface