Dec14-08 Powering the PUMA

Design Document

Team Members:
Matt Bogenschultz
Alex Grieve
Nhat Pham
Zeyu Zhang

Client:
Dr. Greg Luecke

Advisor:
Dr. Greg Luecke

Revision History

Version	Description	Date
1.0	Initial design document written	March 11, 2014
1.1	Revisions for final version	April 16, 2014

Table of Contents

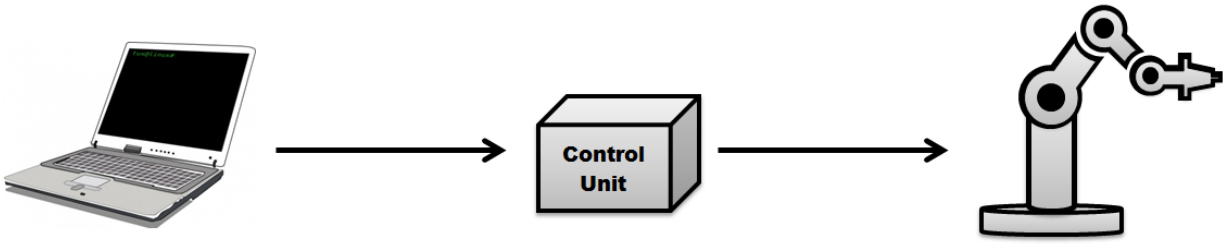
Background

The Unimation Programmable Universal Machine for Assembly (PUMA) is an industrial robot arm with six degrees of freedom. It has a main control unit that sends commands to the PUMA arm and samples feedback from the PUMA arm's sensors. The control unit also supports two mechanisms to program the PUMA arm. The first is a teach pendant that is used to manipulate the six joints and record their positions. The second is a terminal interface where the PUMA arm can be programmed in a language called VAL. The control unit has a floppy disk drive for storing and loading programs.

Project Statement

Our client, Dr. Greg Luecke, has acquired two PUMA 500 robot arms, but the controllers have been damaged and/or lost. Our team objective is to develop a control system that will interact with the PUMA robot arm, which will replace the original controllers.

Concept Diagram



System Requirements

Functional Requirements

1. Six operational joints
    - All six joints of the PUMA robot arm will be operational, and their movement will be controlled by the user.

2. User interface through C code
    - The PUMA robot arm will be controlled by making specific C function calls. This will allow the user to write custom programs that control the PUMA robot arm.

3. Use H-Bridge design
    - The client, Dr. Luecke, already has an existing H-Bridge design that is very robust. He would like it to be refined and utilized in the controller. We will add a logic circuit to the existing H-bridge design that will prevent short circuits.

4. PID Controller
    - We will need to design a PID controller that is operational for six joints. We will implement a proportional response to the current angle and desired angle of each joint. If we wish to have a faster or better loop performance, we can include integral and/or derivative gains.

Non-Functional Requirements

1. Professional Quality
    - Our client would like the controller to look professional. Its circuits should be fabricated on PCBs, and the controller's inputs and outputs should be clearly labeled.

2. Ease of Use
    - The C library of functions will be easy to use, allowing for rapid development of custom applications for the PUMA robot arm.

3. Performance
    - There will be no noticeable lag from the time a command is given to when the PUMA robot arm moves. The output degree of the controller should be accurate with respect to the input command given for each joint.

Detailed Description

User Interface Specification

The user interface will not have a GUI. Instead, we plan to provide a C library of functions to quickly build custom programs that control the PUMA robot for different research applications. The C library will be designed to be used in a Linux-based environment.

I/O Specification

The user will write code that utilizes our C library of functions. The C library will interface with the data acquisition board using its provided drivers. The data acquisition board will send a desired location to our controller alongside a feedback loop which will contain information of the current position of our arm. The error between these two positions will go into our PID controller, which will generate a certain torque value for our motor. As the error of the two

locations (desired-current) becomes smaller, our motor will generate less and less torque until coming to a complete stop at the desired location.

Hardware Specification

We will be using one H-bridge and one PID controller per joint on the PUMA robot (which has a total of 6 joints.) We also will be using a data acquisition board and a power amplifier.

Software Specification

The software drivers are already written and provided by the manufacturer of the MOTENC-Lite boards. The drivers are in raw form, so our C library will attempt to abstract away most of the complexity. The C library will provide functions to easily read encoder counts, read analog inputs, and drive analog outputs.

Simulations and Modeling

We plan to simulate and test our H-bridge design in a Cadence environment before we breadboard it. Then, we will breadboard the H-bridge circuit and test it by utilizing equipment provided in the lab. Once we have a successful H-bridge circuit built on a breadboard, we will design a circuit layout using Eagle CAD.

Implementation Issues/Challenges

Multiple joints to implement

We are combining many different systems together for this project, so implementation could be time intensive. We need to design a full chain for six different joints, for two different robots. Once we achieve successful control over one joint, the other five joints should theoretically be easier. If we choose to implement one of the hand joints first, we may run into trouble with the lower joints, because the required power and torque will be greater.

Component Utilization

The system of how each joint will be implemented will require specific I/O knowledge from each block to the next. Utilizing each component, be it our data acquisition boards, PID controller, or H-bridge, will be a challenge to make sure they all communicate with each other successfully. Specifically, the PID controller will be challenging as we must choose a set of controller gains that produce our system response within the desired specifications.

Testing Procedures

MOTENC-Lite DAQ

We will verify that the MOTENC-Lite boards are fully functional by utilizing test code provided by the manufacturer. The test code allows the encoder values to be read and reset, values to be written to the DAC, write to output registers, and read from input registers. This code will also serve as a starting point for building the C library.

C library
To test the C library, we will write small programs that call the C library functions we created. To verify that the library functions are operating correctly, we will use the gcc debugger to check proper variable values and function return values. If a library function call activates or deactivates a pin or pins on the MOTENC-Lite boards, we will verify this behavior by connecting an oscilloscope to the affected MOTENC-Lite board pin(s).

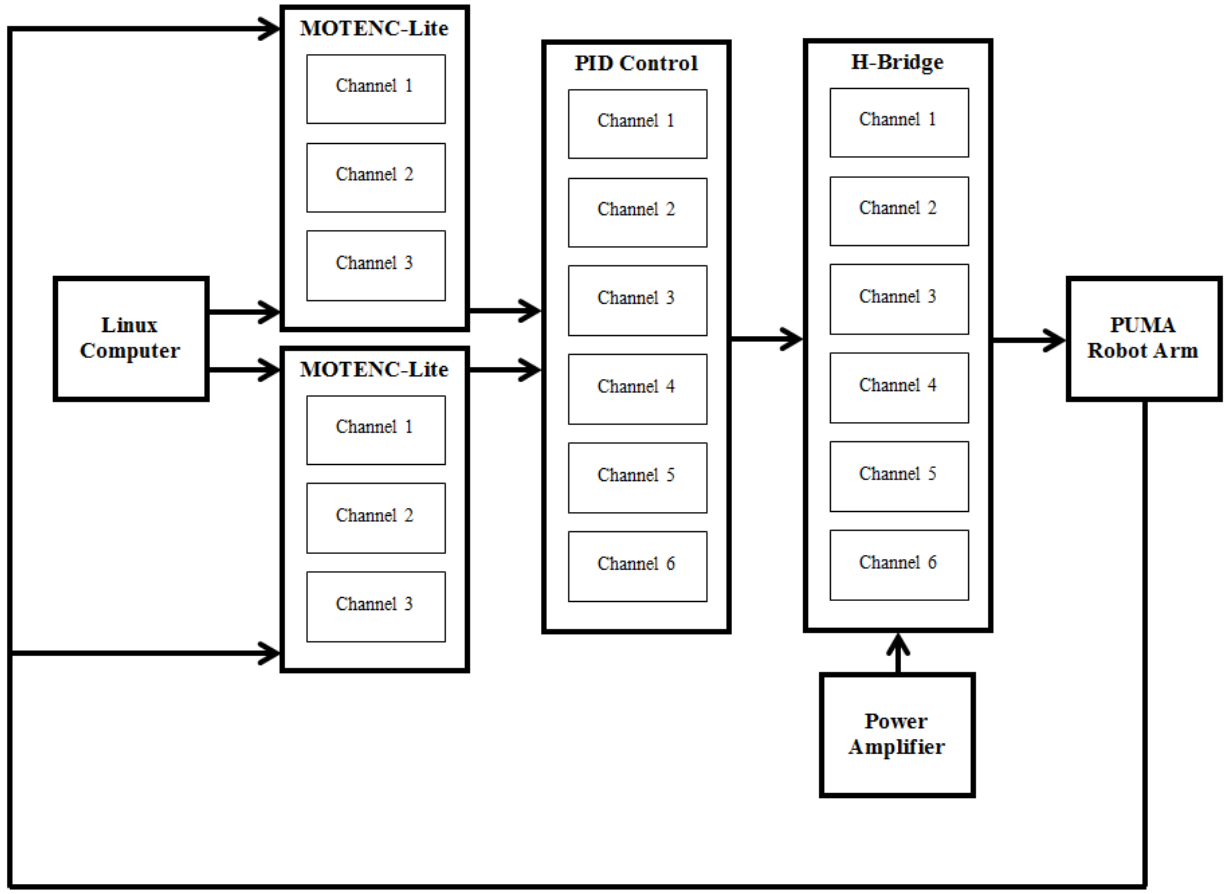
PID Controller
The PID controller will be implemented in software. Thus, testing the PID controller will involve issuing movement commands to the PUMA robot arm and observing the output movement. Each of the PID controllers will have the same general form, but will differ given that each joint has a different inertia. We will tune each controller by changing the proportional, differential, or integral gain until the PUMA arm reaches the desired position with minimal overshoot.

H-Bridge
We will add a logic circuit that prevents both sides of the H-bridge from being trigger at the same time, which could potentially burn out the MOSFETS and short the PUMA robot arm motors. To test the logic circuit, we will apply different input and enable voltages, and verify that only one side of the H-bridge gets activated. To test the H-bridge circuit, we will trigger one side of the bridge and check for continuity, heat dissipation, and polarity. Once the entire H-bridge circuit successfully controls the motor's direction without any error, we will begin the next step of fabrication.

System Integration
Once all components are tested and are working individually, we can begin hooking components together. We will start with the MOTENC-Lite boards and C library and iteratively add a PID controller and verify that it can be controlled correctly by the C code and MOTENC-Lite boards. When all the PID controllers have been integrated with the C library and MOTENC-Lite boards, we will iteratively add an H-bridge and verify that it can be controlled correctly by the C code, MOTENC-Lite boards, and PID controllers. When all H-bridges have been successfully integrated, we can then connect our controller to the PUMA robot and verify that we can correctly control the robot through C code. We will need to make sure that the desired angle input to the C code results in the affected joint moving to the desired angle. If the output angle doesn't match the input angle, we will need to tune the PID controllers to match the two angles.

Design Documents

System Block Diagram



Linux Computer
The Linux Computer will have a C library that will allow for communication between the controller and custom application programs. The C library will serve as a basic API, allowing application programs to easily interact with the PUMA robot arm.

MOTENC-Lite Data Acquisition Board (DAQ)
The MOTENC-Lite boards will be connected to the Linux computer via PCI slots (one PCI slot per board.) The MOTENC-Lite boards will be accessed through C code and will output desired positions for each PUMA robot arm. Additionally, we will leverage the MOTENC-Lite boards quadrature encoder support to get the current position of each arm and output it for the PID controller. Note that a single MOTENC-Lite board only supports four degrees of freedom, while the PUMA robot has six degrees of freedom, hence the need for two boards.

PID Controller
There will be a PID Controller for each joint of the PUMA arm and they will be implemented digitally through the Linux computer and MOTENC-Lite boards. By taking in two input values from the DAQ associated with a desired and current angle value of each joint, it will output a

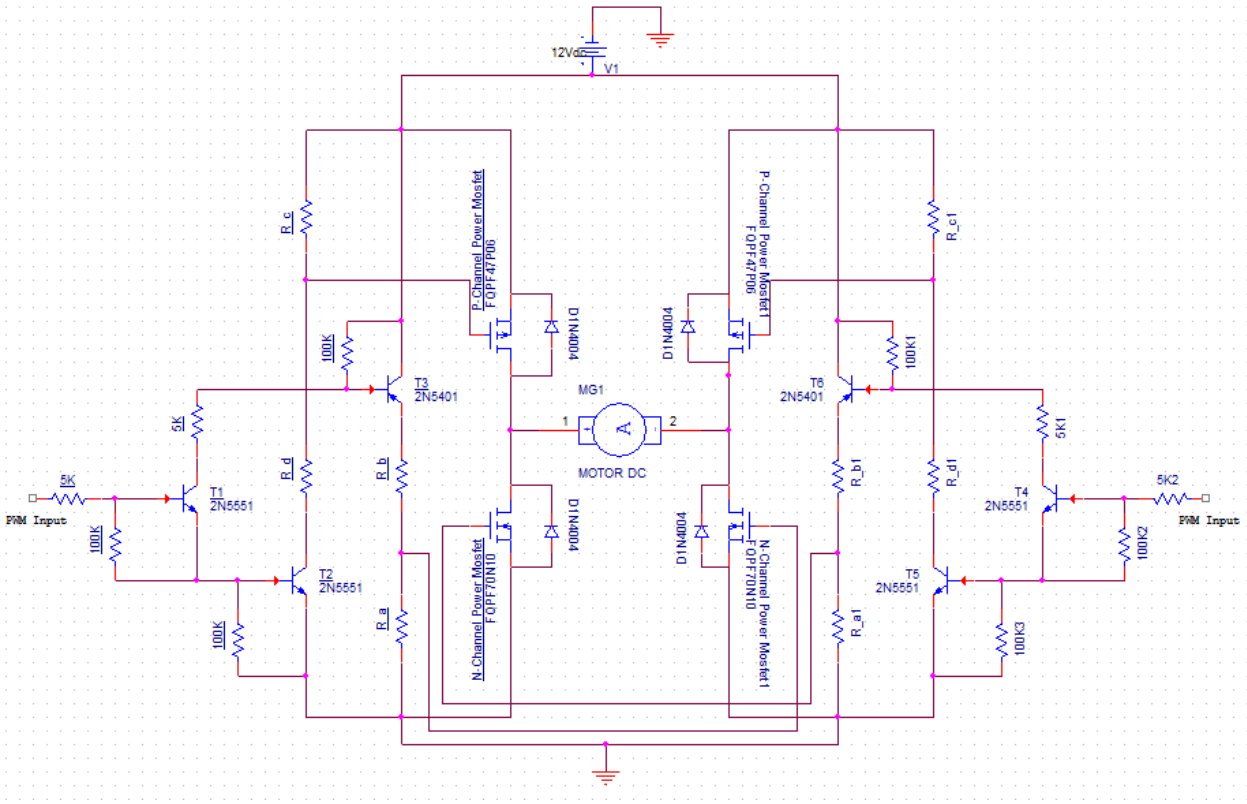
given torque value. These circuits may differ based on the power demands or the degrees of freedom each joint is able to move. Also, within this block we will need to know the certain inertial values of each joint, so that we can allot the correct torque value to its given motor.

H-Bridges
The H-Bridge controls the direction of the DC motor. They switch current from the power amplifier in one of two directions at a time, depending on the input from the PID control circuit. We will also implement a logic circuit to avoid a short circuit if both sides of the H-bridge are being trigger by code bugs, machine errors, etc.

Power Amplifier
The power amplifier provides power to the six DC motors. Three joints only require 12 volts, and the other three require 40 volts. The three 40 volt motors also have brakes that must be unlocked with 24 volt input. Current requirements will be calculated based on the needs for each H-bridge and arm motor, and will be verified with measurements.

PUMA Robot Arm
The PUMA robot arm has six movable joints. Each joint has a DC motor and a quadrature encoder attached to its shaft. As the DC motor is energized, the encoder rotates, creating pulse trains. These pulse trains are used to indicate the current position, speed, and direction of rotation of a specific joint. Each joint on the arm has some inertial value associated with how hard it will be to move that given joint. These values will need to be taken into account when deciding how much torque a given joint will need to move it.

H-Bridge Design



The H-bridge works by turning on a resistor voltage divider driving a voltage on each gate that exceeds the MOSFETs ON-gate voltage. The PWM input signal pulls the base of transistor T1 high which allows current to flow from the base of T3 to the base of T2, limited by a 5KΩ resistor. This turns T2 and T3 on, powering the voltage dividers comprised of R_a/R_b, and R_c/R_d.

This H-bridge design uses a mixture of P-channel and N-channel MOSFETs. Each motor path uses one P-channel and one N-channel MOSFET. Selecting the correct pair of MOSFETS is one of the most important parts of designing this circuit since so many factors affect its performance. It can get a little tricky at times to pair a P and N channel MOSFET together since different aspects of their construction cause functional differences between them. Perhaps this is why it's a common design practice to use N-channel MOSFETs for all four switches, and using a charge pump to drive the high side MOSFETs. This is a common practice, and actually exists in an IC called a bridge driver. Some of the properties that were taken into consideration for MOSFET selection are shown below:

1. Package dimensions – make sure that they are in a through hole TO-220-3 case, 0.1 inch spacing.

2. Continuous Drain Current – shooting for 30 amps, but could go higher or lower depending on needs.

3. Drain-Source On Resistance – The lower the better, but it's good to have these matched between P/N channel MOSFETs. The lower the on resistance, the lower the voltage drop across them in operation, resulting in lower heat buildup that needs to be dissipated in the MOSFETs. It's good to match these so they dissipate heat evenly.

4. Input capacitance – Obviously, the smaller this value is, the faster the H-bridges response time. As long as the capacitances are in the same order of magnitude we should be fine.

5. Absolute maximum Gate-to-Source voltage, and Gate On threshold – Aiming for ±20V, with a 4V on threshold. We can then set R_a/R_b to drive the gate to ~9 volts at normal operation mode, and be able to allow for supply voltage to almost double or half without risking damaging the MOSFETs or killing performance.

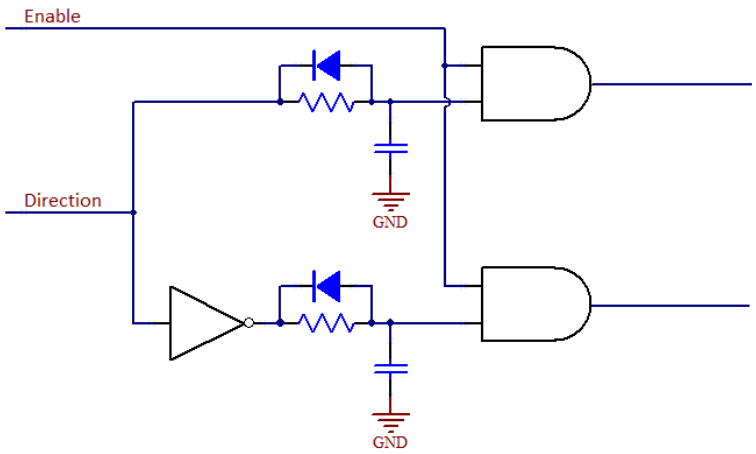
6. Power Dissipation – ensure the rated power dissipation on each MOSFET is high enough to handle our needs. This could be calculated by calculating the voltage drop across them in operation with our PID controller. For instance, if a 1Ω motor stalls at 12 volts, a MOSFET with 10mΩ will need to dissipate approximately 1.44 Watts.

7. Isolated mounting area – A lot of the higher power dissipation MOSFETs will have a metal back, but a lot of them have that metal backing connected to the drain of the MOSFET. If all four of these are mounted to a heat sink for cooling without an electric isolating pad, the two MOSFETs will short circuit. Usually the isolated ones have lower power dissipation, so that's something that will have to be decided based on our application.

H-Bridge Logic

The major challenge when designing an H-bridge system is avoiding shoot-through. Shoot-through is the condition where both PWM inputs, both left side switches, or both right side switches are closed at the same time. This will result in a short circuit which could potentially destroy the MOSFETs or other components. Shoot-through can occur through software bugs or overlap in gate drive signal when switching from one direction to another.

To overcome this challenge, we will implement a logic circuit that limits one side of the H-bridge being activated at a time using an enable input and a direction input:

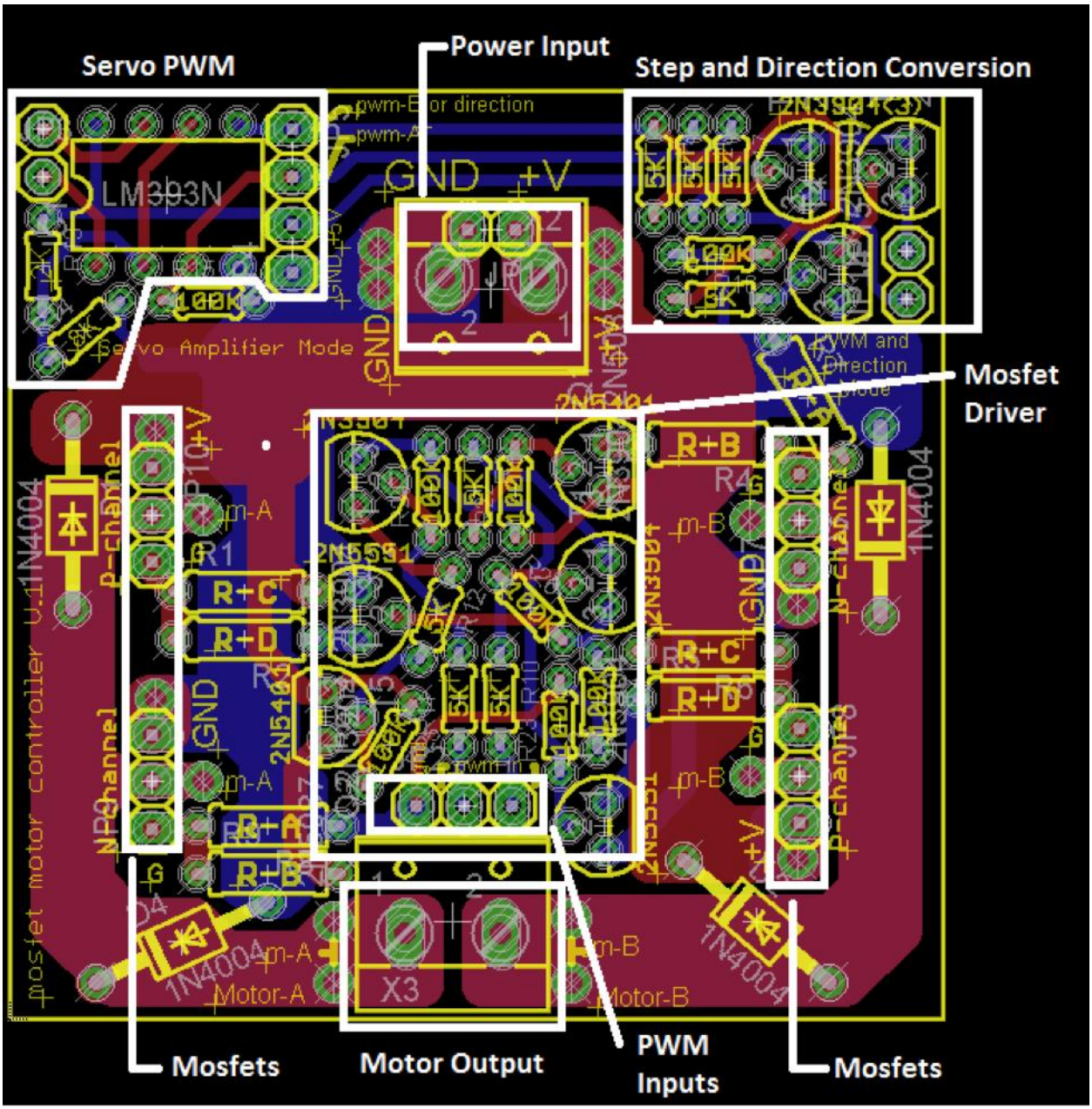


The enable input is a PWM signal and the direction input will decide which input of the H-bridge to trigger. The two AND gates allow the enable signal to disconnect the control signals from the base of the NPN. The diode/resistor combinations make the charging of the capacitors slower, thus it takes longer for the gate drive signals to reach the transistors when the transistors are about to turn on. This makes it possible to control the timing of the four transistors with just two delay circuits.

We are planning to use NAND gates instead of AND gates, and to have CMOS input levels which the Schmitt trigger inputs will switch at about half the supply voltage for stability. Also, we might have inverters to have CMOS outputs to drive the base of the NPN up to about 5V. The need for these extra components will become clearer when we integrate the H-bridge with the MOTENC-Lite boards.

CAD Electronic

Old H-Bridge Eagle CAD drawing



This is the Eagle CAD H-bridge schematic that we inherited from one of Dr. Luecke's previous students. We will use this as a starting point for creating a circuit design for our H-bridge. The Servo PWM and Step and Direction Conversion circuits in the drawing above do not work, so we will be omitting them from our circuit design.

PCB issues
There could possibly be PCB fabrication errors with our new H-Bridge design.

Conclusion
A basic overview of our system's functionality could be summarized as the ability to control the joints of a PUMA robotic arm from user input via C code. The code will set the user's desired location of each of the PUMA arm's joints. This information will be communicated through a data acquisition board into our PID controllers. The PID controllers will receive inputs of the joints' current locations, and will react according to the user's desired position. These PID controllers will output and effectively apply power to the joints through the DAQ outputs with the assistance of an H-bridge.

We are confident that our design is on the right track, and only need our advisor/client to help point us in the correct direction regarding certain technical aspects. We fully anticipate completing this project by the end of Senior Design II.