Dec14-08 Powering the PUMA

Final Document

Team Members:
Matt Bogenschultz
Alex Grieve
Nhat Pham
Seth Taylor
Zeyu Zhang

Client/Advisor:
Dr. Greg Luecke

Revision History

Version	Description	Date
1.0	Initial design document written	March 11, 2014
1.1	Revisions for first semester version	April 16, 2014
2.0	Revisions for final version	November 28, 2014

Table of Contents

Project Introduction 1
Acronym Definitions 1
Background 1
Project Statement 1
Concept Diagram 2
System Requirements 2
Functional 2
Non-Functional 2
System Design 4
Primary Circuit Block Diagram 4
Linux Computer 5
MOTENC-Lite 4-Axis Motion Control & Data Acquisition PCI Board (MOTENC) 5
Pulse Width Modulated (PWM) Control 5
H-Bridges 5
Motor Power Supply 5
PUMA 5
Auxiliary Power & Secondary Circuit Block Diagram 6
Power Entry Module 6
Circuit Breakers 6
Control Relays 6
Auxiliary Circuit Power 6
Cooling Fans 7
Status Circuit 7
Electromagnetic Brake 7
Additional Specifications 8
User Interface 8
Linux Computer 8
Controller 8
I/O 8
Input 8
Output 8

Hardware 8
PUMA 560 (x2) 8
MOTENC-Lite (x2) 8
PWM Circuits (x6) 8
H-Bridges (x6) 8
Power 9
Circuit Breakers 9
Control Relays 9
Operator Switches/Buttons 10
Indicators 10
Cables 10
Terminal Blocks 11
Software 11
Primary functions 11
Helper functions 11
Design Documents 12
C Code Design 12
PWM Design 12
H-Bridge Design 14
Status Circuit Design 18
Electromagnetic Brake Release Design 21
Main Board Design 23
General PCB Design 23
Power Delivery Design 24
Enclosure Design 27
Testing Procedures and Results 28
MOTENC-Lite DAQ 28
Procedure 28
Results 28
C library 28
Procedure 28
Results 28

PWM Control 28
Procedure 28
Results 29
H-Bridge 29
Procedure 29
Results 29
Power Delivery 29
Procedure 29
Results 30
Status Circuit 30
Procedure 30
Results 30
System Integration 30
Procedure 30
Results 30
Appendix I: Operation Manual 32
Electromagnetic Brake Operation 32
Front Panel 32
Computer Interface 32
Motor Operation 32
Appendix II: Initial Design Versions 34
Old H-bridge Eagle CAD Drawing 34
Appendix III: Other Considerations 35
PWM Circuit 35
H-Bridge Circuit 35
Status Circuit 36
Common Reference Voltage 36
Appendix IV: Bill of Materials 37
PWM Control Circuits Parts List 37
H-Bridges Parts List 37
Status Circuits Parts List 37
Main Board Parts List 38

Power Delivery Parts List 38
Enclosure, Wire, Miscellaneous Parts List 39
Appendix V: C Code 41
Puma_config.h 41
Puma.h 42
Puma.c 45
Skel.c 57
Appendix VI: Enclosure Drawings 59
Front Panel 60
Rear Panel 61
Left Panel 62
Right Panel 63
DIN Rail View 64
Front Panel Hole Locations 65
Back Panel Hole Locations 66
Left Panel Hole Locations 67
Right Panel Hole Locations 68
Appendix VII: Electrical Schematics 69
Cover Sheet 70
BOM 71
Excelsys 72
Meanwell 73
Control Relays 74
Main Board 75
MOTENC 1 and 2 76
J1 and J2 PCBs 77
J3 and J4 PCBs 78
J5 and J6 PCBs 79
EM Brake 80
Harness Adapter 81
LED Display 82

Project Introduction

Acronym Definitions

CAD – Computer-Aided Design
EM – Electromagnetic or Electromechanical
EMI – Electromagnetic Interference
I/O – Input/Output
IEC – International Electrotechnical Commision
LED – Light Emitting Diode
MOSFET – Metal-Oxide-Semiconductor Field-Effect transistor
NEMA – National Electrical Manufacturers Association
PCB – Printed Circuit Board
PCI – Peripheral Component Interconnect
PEM – Power Entry Module
PMDC – Permanent Magnet DC Motor
PUMA – Programmable Universal Manipulation Arm
PWM – Pulse Width Modulation
SMPS – Switched-Mode Power Supply
VAL – Variable Assembly Language

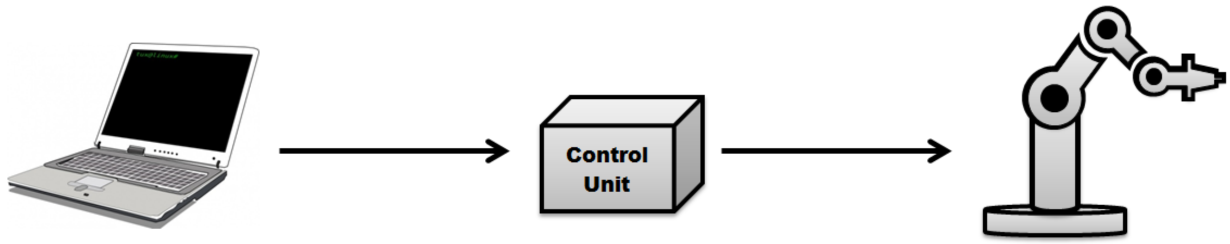
Background

The Unimation Programmable Universal Manipulation Arm (PUMA) is an industrial robotic arm with six degrees of freedom (i.e. six moveable joints). The PUMA system consists of a main control unit that sends commands to and samples feedback from sensors in the mechanical joints. The control unit also supports two mechanisms to program the PUMA. First, a teach pendant is utilized to manipulate the six joints and record their positions. Second, there is a terminal interface where the PUMA can be programmed in the VAL computer language. The control unit has a floppy disk drive for storing and loading programs.

Project Statement

Our client, Dr. Greg Luecke, has acquired two PUMAs. However, the original controllers are damaged beyond reasonable repair. The team objective is to develop a new control system for the PUMA that will replace the original controllers.

Concept Diagram



System Requirements

Functional

1. Six operational joints
   - All six joints of the PUMA will be operational, and movements will be defined by a user-specified torque value applied at each joint.

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

3. Use existing H-bridge design
   - The client, Dr. Luecke, has an existing H-Bridge design that is very robust. He would like it to be refined and utilized in the controller.

Non-Functional

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.

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

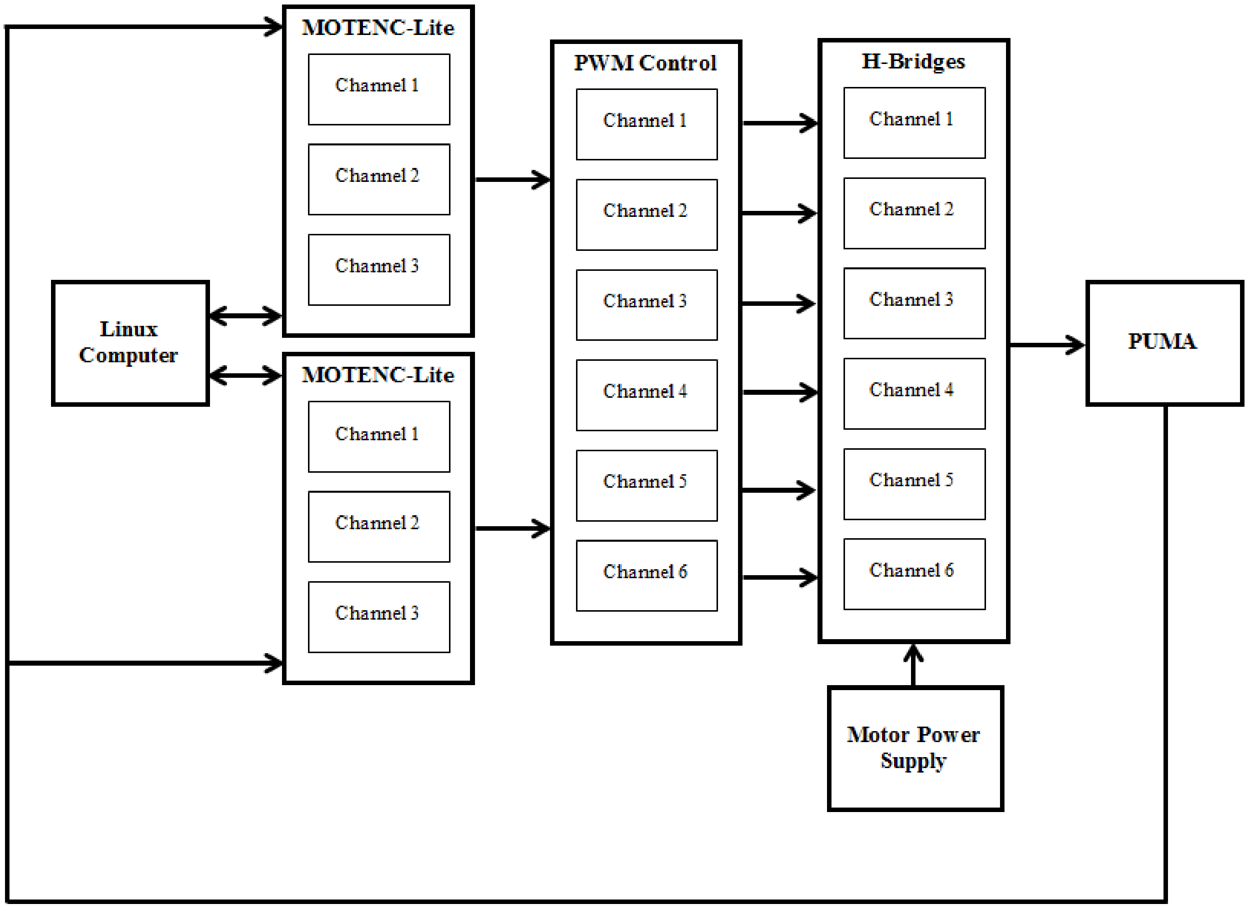
4. Status Circuit

- A separate circuit will monitor the control system and provide visual feedback. This will allow direct traceability of component failure in control system sub circuits.

5. Modular Design
    - The control system will be of a modular design. Separating aspects of the control system into sub circuits allows for easier repairs. Also, this allows for modifications in future sub circuit designs.

System Design

Primary Circuit Block Diagram



Linux Computer
The Linux computer utilizes a C library that allows for communication between the controller and custom application programs. The C library will serve as a basic API (Application Programming Interface) allowing application programs to easily interact with the PUMA.

MOTENC-Lite 4-Axis Motion Control & Data Acquisition PCI Board (MOTENC)
Two MOTENC boards are connected to the Linux computer via PCI slots (one PCI slot per board.) These boards are accessible through C code and will output desired positions for each PUMA joint as an analog voltage ranging from 0-5 volts. The MOTENCs' quadrature encoder support will be leveraged to get the relative position of each arm. *Note that a single MOTENC board only supports four degrees of freedom, while the PUMA robot has six degrees of freedom, hence the need for two boards.*

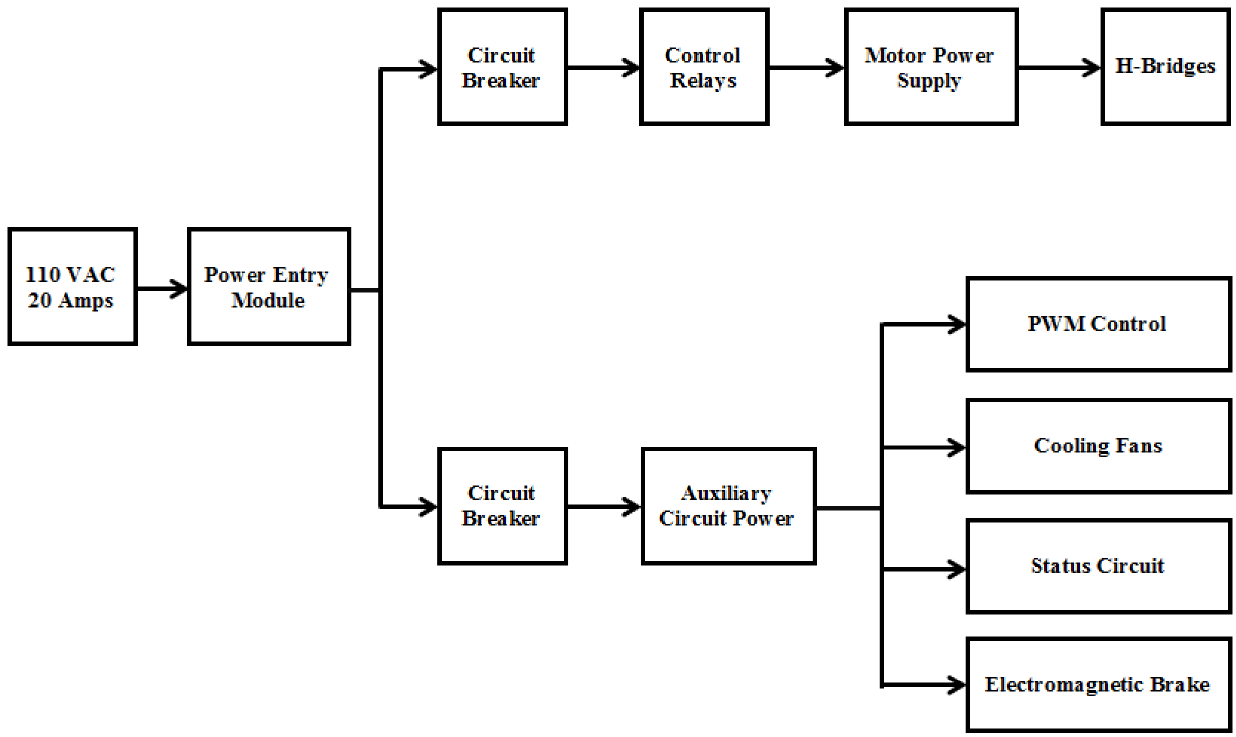
Pulse Width Modulated (PWM) Control
Each PWM circuit outputs a square wave signal whose duty cycle is controlled by an input voltage from the MOTENCs. This input voltage ranges from 0-5 volts and linearly scales the output square wave duty cycle.

H-Bridges
Each H-Bridge controls the rotational direction of the DC motor at a particular joint. There are two PWMs per H-Bridge, and each PWM changes the path that current from the power supply flows across the motor. This allows for bidirectional rotation of each motor.

Motor Power Supply
The power supply provides power to the six permanent magnet DC motors at 24 volts apiece. The robot is divided into two sets of three identical motors. Three motors are designed to operate at 2-6A, while the other three motors operate between 1-4A. The power supply is composed of six dedicated modules that can be treated as isolated channels of power for each joint.

PUMA
The PUMA has six movable joints, and 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.

Auxiliary Power & Secondary Circuit Block Diagram



Power Entry Module
The power entry module delivers ~110VAC (at 20A) wall power to the primary and secondary/auxiliary circuits. It has built in overcurrent and electromagnetic interference protection.

Circuit Breakers
The circuit breakers provide overcurrent protection to the power supplies. However, these breakers allow for excessive inrush currents over short periods of time.

Control Relays
The control relays include on/off and emergency shut-off capabilities of power to the motors. Additionally, a momentary on button is provided to release the PUMA electromagnetic safety brake.

Auxiliary Circuit Power
The auxiliary circuit power delivers the necessary voltage and current to the cooling fans, PWMs, Status Circuit, and PUMA electromagnetic safety brake.

Cooling Fans
The cooling fans provide increased air circulation to help keep components within their respective operational temperature range.

Status Circuit
The status circuit monitors the input commands from the MOTENC versus the output of the PWMs and H-Bridges at each joint. The circuit detects if the components are damaged or not receiving commands. A green LED indicates positive functionality, while a red LED shows where a particular failure occurred.

Electromagnetic Brake
Three of the six motors have electromagnetic brakes installed to lock the joints in position. The brake is switched on/off by either a low power H-Bridge or a momentary switch. The brake requires less power to function than the PUMA motors.

Additional Specifications

User Interface

Linux Computer

- Ubuntu 12.04 LTS operating system
- Library of functions written in the C programming language to control the PUMA (non-graphical)

Controller

- Motor power on/off switch
- Emergency stop button
- Momentary on button to release the PUMA brake

I/O

Input

- User specified torque to be applied to a specific PUMA joint in a particular direction

Output

- Applies corresponding power to PUMA motor, resulting in arm movement

Hardware

PUMA 560 (x2)

- Existing robot frame, motors, and wiring

MOTENC-Lite (x2)

- Version 7541
- 32-bit resolution quadrature encoder (x4)
- -10VDC to 10VDC programmable analog output (x8)
- 14-bit resolution analog inputs, 5VDC max (x8)

PWM Circuits (x6)

- Custom circuit
- 0-5VDC input
- 0-100% duty cycle (scales linearly with input voltage), 5VDC PWM output

H-Bridges (x6)

- Custom circuit
- 0-5VDC PWM input (x2, one per direction)
- Applies power to PUMA motor during PWM duty cycle

Power
Meanwell Dr-120-24

- Switched Mode Power Supply
- ~115VAC, 3.3A in / 24VDC, 0~5A shared output
- 120W rated power
- 20A inrush current

Excelsys UX6 (with 3 XgB modules & 3 XgE modules)

- Switched mode power supply
- ~120VAC, 11.5A input
- UX6 frame 1200W max output
- XgB 24VDC (nominal), 0~8.3A output
- XgE 24VDC (nominal), 0~5A output

Circuit Breakers
Power Entry Module

- 125/250VAC, 20A max
- IEC 320-C20 (Male Pins)
- 2 Pole, Quick Connect Terminals
- Shielded, EMI Line filter
- General Purpose, Panel Mount

Excelsys

- 480VAC, 5A max
- Lever Actuator
- 2 poles, Thermal Magnetic
- Screw Clamp Terminals
- DIN Rail mount

Meanwell

- 480VAC, 15A max
- Lever Actuator
- 2 poles, Thermal Magnetic
- Screw Clamp Terminals
- DIN Rail mount

Control Relays
Excelsys ON/OFF & Emergency Stop

- Non-latching, general purpose
- Contacts: 120VAC,15A max
- Coil: 24VDC, 37mA
- 1N.O.+ 1N.C.
- DIN Rail mount

EM Brake Momentary Button

- Non-latching, general purpose
- Contacts: 400VAC, 6A max
- Coil: 24VDC, 6mA
- 1N.O.+ 1N.C.
- DIN Rail mount

Operator Switches/Buttons

Motor Power Emergency Stop

- 22mm (dia.) cutout, 40mm (dia.) mushroom head
- Double Pole/Single Throw, On-Off/Off-On
- 1N.O.+ 1N.C.
- Coil: 24VDC, 10mA
- Panel Mount

Motor Power Selector Switch

- 22mm (dia.) cutout, 29.8mm (dia.) bezel
- Double Pole/Single Throw, Two Positioned, Maintained
- 1N.O.+ 1N.C.
- Coil: 24VDC, 10mA
- Panel Mount

EM BRake Momentary ON

- 22mm (dia.) cutout, 29.7mm (dia.) bezel
- Double Pole/Single Throw On-Mom (Off-Mom)
- 1N.O.+ 1N.C.
- Coil: 24VDC, 10mA
- Panel Mount

Indicators

Status Lights

- Kingbright RGB, through hole
- 1.9~3.3VDC (feedforward), 20mA (normally)
- LED
- Flush Mount

EM Brake Indicator

- Omron 16mm round yellow pilot light
- 24VDC, 8mA (normally)
- LED
- Panel Mount

Cables

Single Phase Wall Power

- 125VAC, 20A (rated)
- Male Pins (blades), Female Sockets (slots)
- NEMA 5-20P to IEC 320-C19
- 12 AWG, 3 conductor
- 10’ length

Single Phase Excelsys Power

- 250VAC, 15A (rated)
- Female Sockets (slots) to Leads
- IEC 320-C13, Right Angle
- 14 AWG, 3 conductor
- 3.28’ length

Terminal Blocks
AC (Black, White, Green/Yellow)
- 2 Position, feed through
- 1000V, 24A (rated)
- 12-26 AWG
- DIN Rail mount

DC (Blue, Grey)
- 2 Position, feed through
- 1000V, 24A (rated)
- 12-26 AWG
- DIN Rail mount

Software

Primary functions
- Drive MOTENC analog outputs for user specified torque values
- Read quadrature encoder counts for each PUMA motor
- Read analog inputs (PUMA motor potentiometer feedback)
- Turn PUMA electromagnetic brake on/off

Helper functions
- Search through all connected PCI devices, locate and memory map each MOTENC
- Initialize MOTENC quadrature encoders, analog outputs, and analog inputs for use
- Convert desired analog output voltage to the correct MOTENC output transfer function

Design Documents

C Code Design

The C library is the primary interface for controlling the PUMA, so many considerations were taken into account when designing the code. After talking with our client, it became clear that the end user of the code would be one with intermediate C programming knowledge and experience. Therefore, the C code was designed to hide the messy, lower level details of interfacing with the MOTENC-Lite boards (such as PCI interfacing, memory mapped I/O, bitwise operations, etc.) Instead, the C library provides a clean, easy to use interface to control the PUMA.

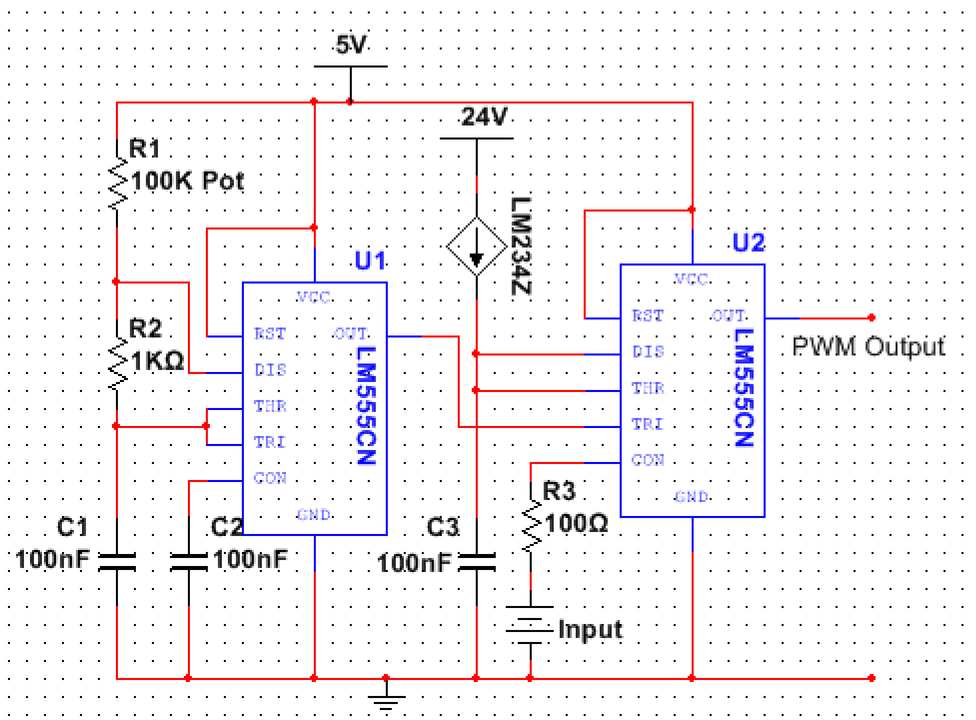
There were also a couple of safety considerations that went into the design of the C code. First, the code must never allow the user to activate both sides of an H-bridge at the same time. This can demagnetize the motor attached to the H-bridge, and ultimately destroy said motor. There are safeguards placed in the C code to prevent this situation from occurring. Second, in the event of a program crash or unexpected termination, the PUMA needs to be returned to a safe state (i.e. stop all motors and apply the electromagnetic brake.) This functionality can be easily achieved by inserting signal handlers into the C code. When termination or interrupt signals are unexpectedly sent to the user's program, a shutdown routine is run to place the PUMA in a safe state. These signal handlers, as well as suggested program flow, are located in a code skeleton file. The skeleton file should always be used as the starting point for any end user's PUMA application.

The C library consists of three files. The header file, `puma.h`, contains definitions of all the available functions the user can call to control the PUMA. The implementation file, `puma.c`, contains the implementation of all the functions defined in the header file, `puma.h`. It is strongly recommended that the end user only call the functions labeled as "Public" or "Primary" and never call functions labeled "Private" or "Helper" (see comments in the code.) The third file is a configuration header, `puma_config.h`, that contains all the constants for a particular PUMA. The constants defined in this header file can be tweaked by the user to better match the characteristics of a particular PUMA. It's recommended that these constants be changed in small intervals to prevent damage to the PUMA.

Please see Appendix V for the C library code and skeleton file.

PWM Design

There are many different ways to control the speed of a PUMA motor, but one simple and effective way is to use Pulse Width Modulation (PWM.) The PWM is the process of switching power ON and OFF to a device in pulses at a specific frequency. Driving a DC motor with a PWM signal is often used in conjunction with an H-bridge. In order to adjust the amount of current flowing through the H-bridge, a pulse width modulator circuit was designed. The main idea is that a change in the duty cycle of the PWM proportionally changes the average current flow through the H-bridge. The use of two 555 timers allows for a configurable duty cycle and generation at a desired frequency.



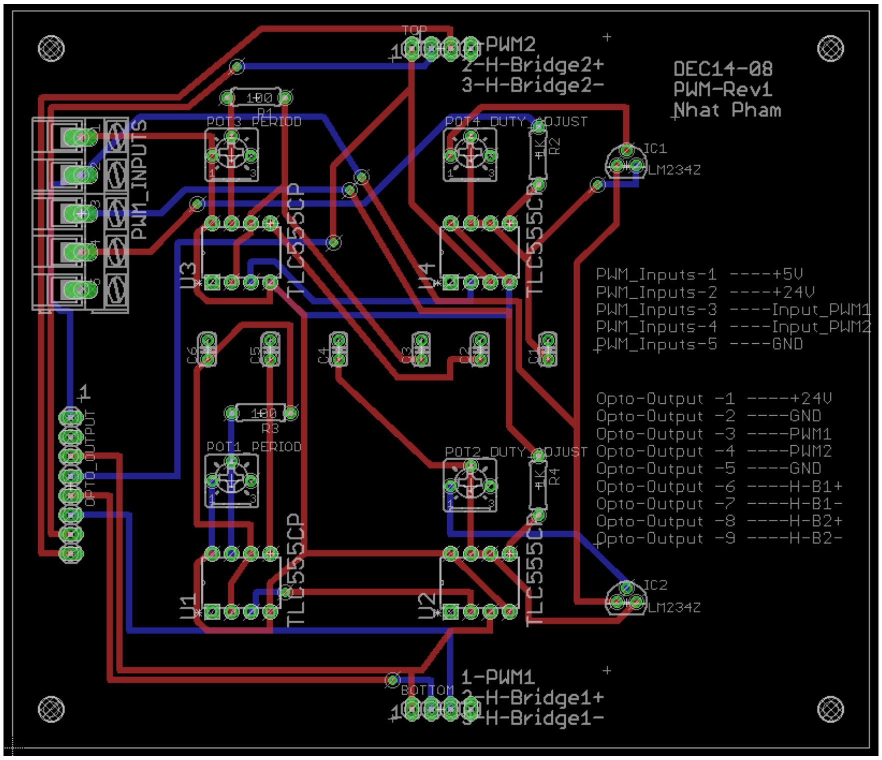
The PWM works by using the first 555 timer as a constant frequency square pulse generator that drives the trigger input of a second 555 used as a monostable output, with its delay timing varied by a voltage applied to its control pin. In other words, varying the internal set point of the second 555's control voltage pin will change the pulse width duty cycle. Every time the trigger pin pulses low to less than 1/3 of the supply voltage, the 555's output switches to high, and the discharge transistor is disabled. Disabling the discharge transistor allows the capacitor C3 to charge through an IC current source. The capacitor keeps charging until its voltage is above the control voltage at which the 555 changes state. Once it changes state, the output will go low and the discharge pin becomes active to discharge C3. The cycle will repeat once the trigger of the second 555 senses a falling edge from the first 555. Therefore, the duty cycle is determined by the control voltage and the frequency is dependent on the ratios between R1, R2, and C1. Some of the properties and guidelines that were taken into consideration for designing this PWM modulator are shown below.

1. Design tips for desired period/frequency on the first 555 timer
   Periodic time $T = 1/f$
   Charge time $t_C = 2/3T$ or $t_C = 0.7 \times (R1+R2) \times C1$
   Discharge time $t_D = 1/3T$ or $t_D = 0.7 \times R2 \times C1$

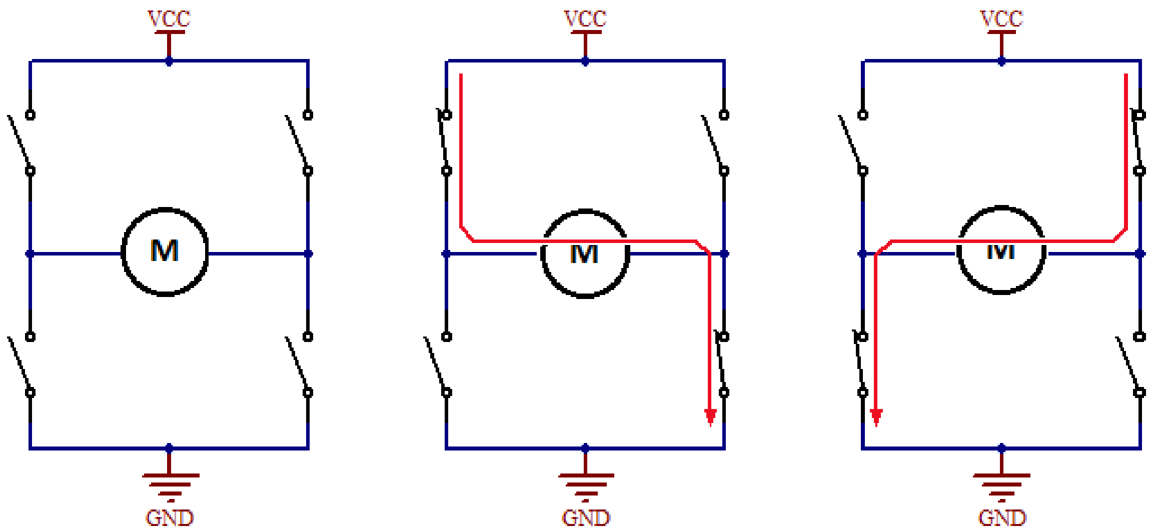
2. Current Source – the IC current source used in this design is a LM234Z. In order to obtain a linear duty cycle output, the threshold voltages have to increase linearly. If a resistor were to be used instead of an IC current source, the capacitor will charge in an exponential manner rather than linear manner. To drive the current source, a resistor is required; in this case a potentiometer is used. This potentiometer can also be used as a calibration tool.

3. To adjust the frequency in the first 555 timers, R1 could be replaced with a potentiometer.

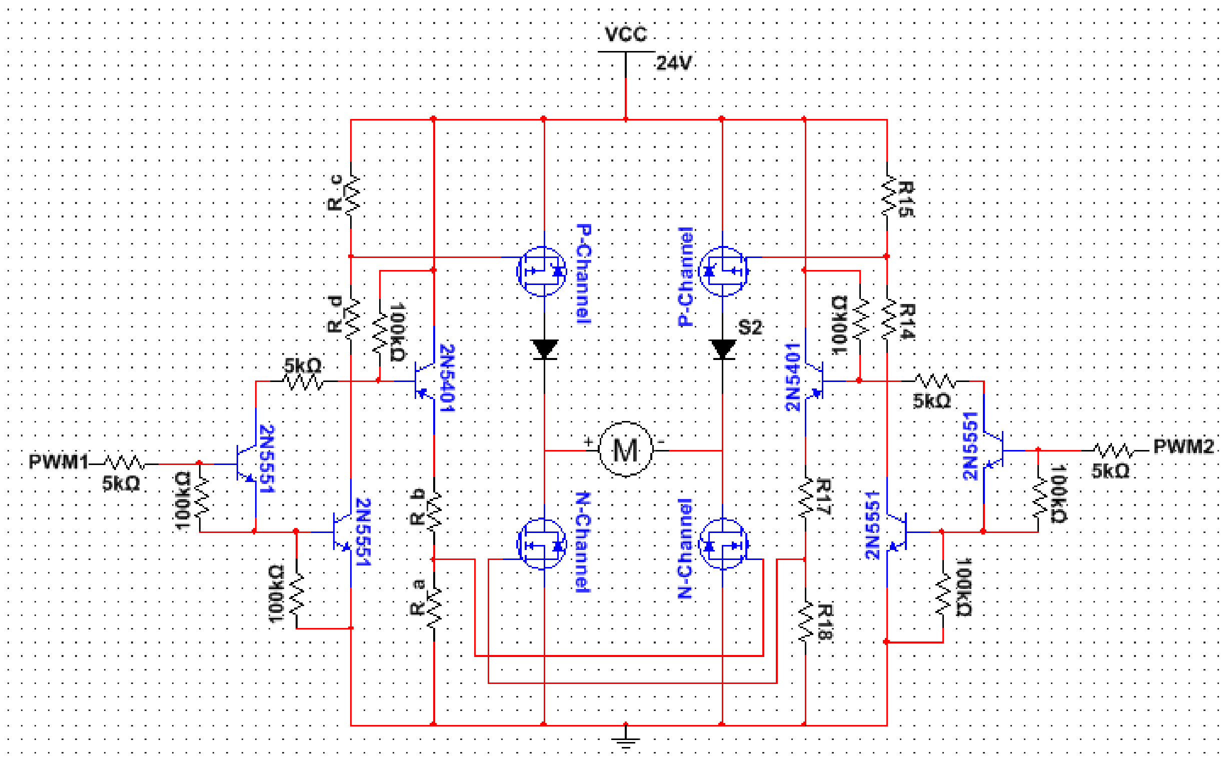
The PCB design can be found below.



H-Bridge Design



In general, an H-bridge is simply a set of switches used to alternate the polarity on a DC motor thus changing the direction of rotation, as shown above. Closing the correct switches creates a path for current to flow across the motor.



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. Capacitances of the same order of magnitude should be fine.

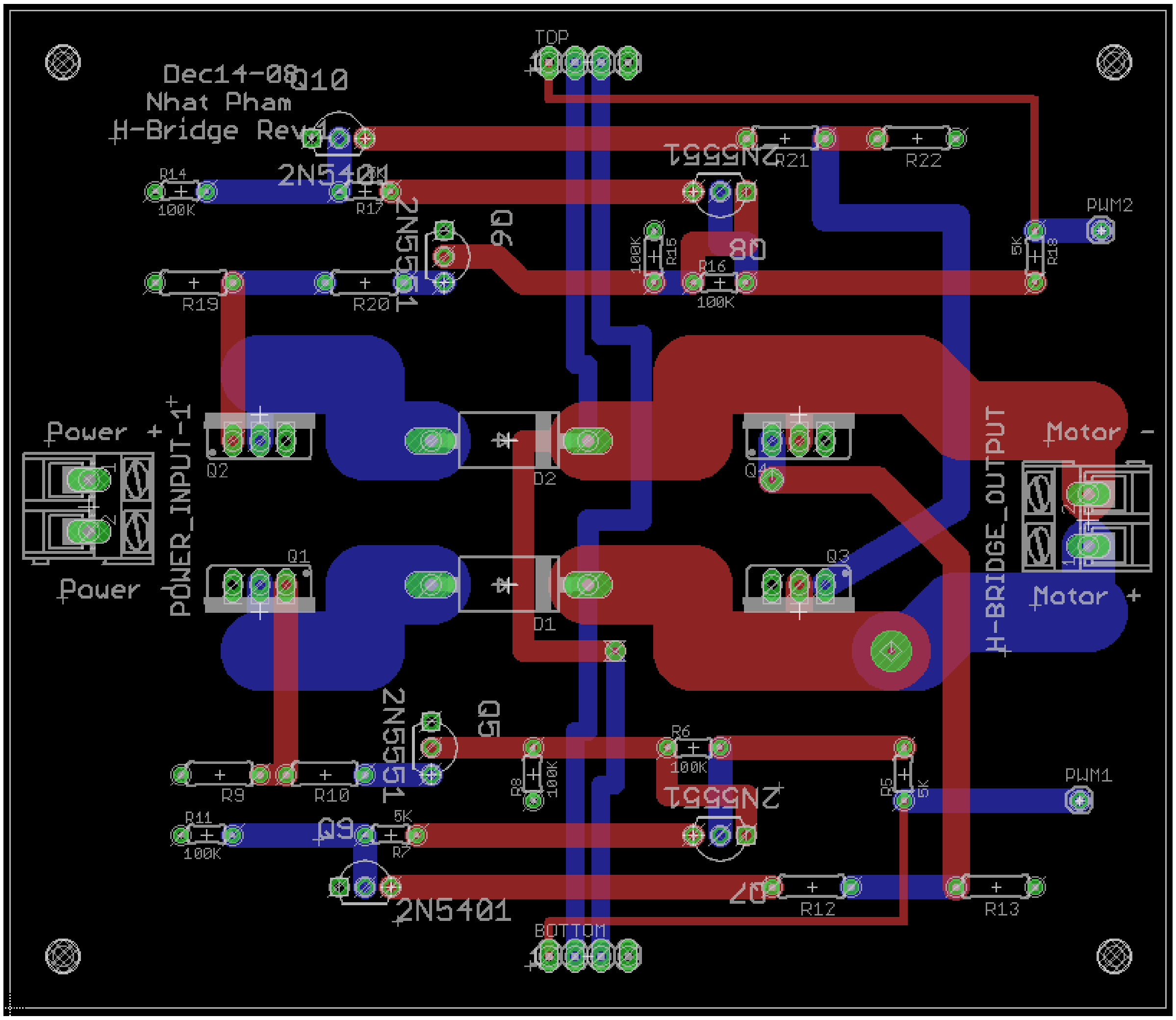
5. Absolute maximum Gate-to-Source voltage, and Gate ON threshold – Aiming for ±20V, with a 4V on threshold. Then, set R_a/R_b to drive the gate to ~9 volts at normal operation mode, and be able to allow for supply voltage to almost double or half without risking damaging the MOSFETs or killing performance.

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

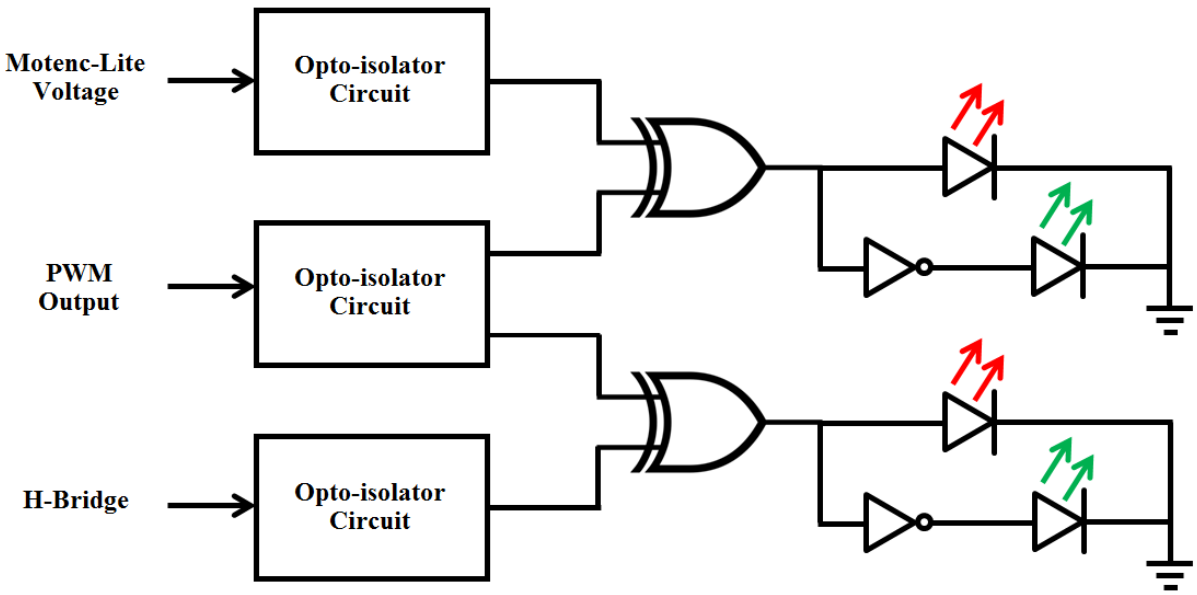
7. Isolated mounting area – A lot of the higher power dissipation MOSFETs will have a metal back, but a lot of them have that metal backing connected to the drain of the MOSFET. If all four of these are mounted to a heat sink for cooling without an electric isolating pad, the two MOSFETs will short circuit.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

In addition, when the DC motor is rotating, it creates a back electromotive force (EMF) that pushes against the current that is driving the motor. The back-EMF manifests itself as a voltage and is generated by the motion of the motor's armature relative to the magnetic field of the motor's magnets. To protect the circuit from the back-EMF, select the right diode based on the motor specification and application needs.

The PCB design can be found below. Note the large traces needed for the large amount of current that will be flowing to the motor.


Dec14-08
Nhat Pham
H-Bridge Rev1
TOP
BOTTOM
Q10
2N5401
2N5551
Q6
Q8
R14
100K
R17
R19
R20
R21
R22
R15
100K
R16
100K
R18
5K
PWM2
Power +
Power -
POWER_INPUT-1
Q2
Q1
D2
D1
Q4
Q3
Motor -
Motor +
H-BRIDGE_OUTPUT
Q5
R9
R10
R11
100K
Q9
R7
5K
R8
100K
R6
100K
R5
5K
PWM1
Q7
R12
R13

Status Circuit Design



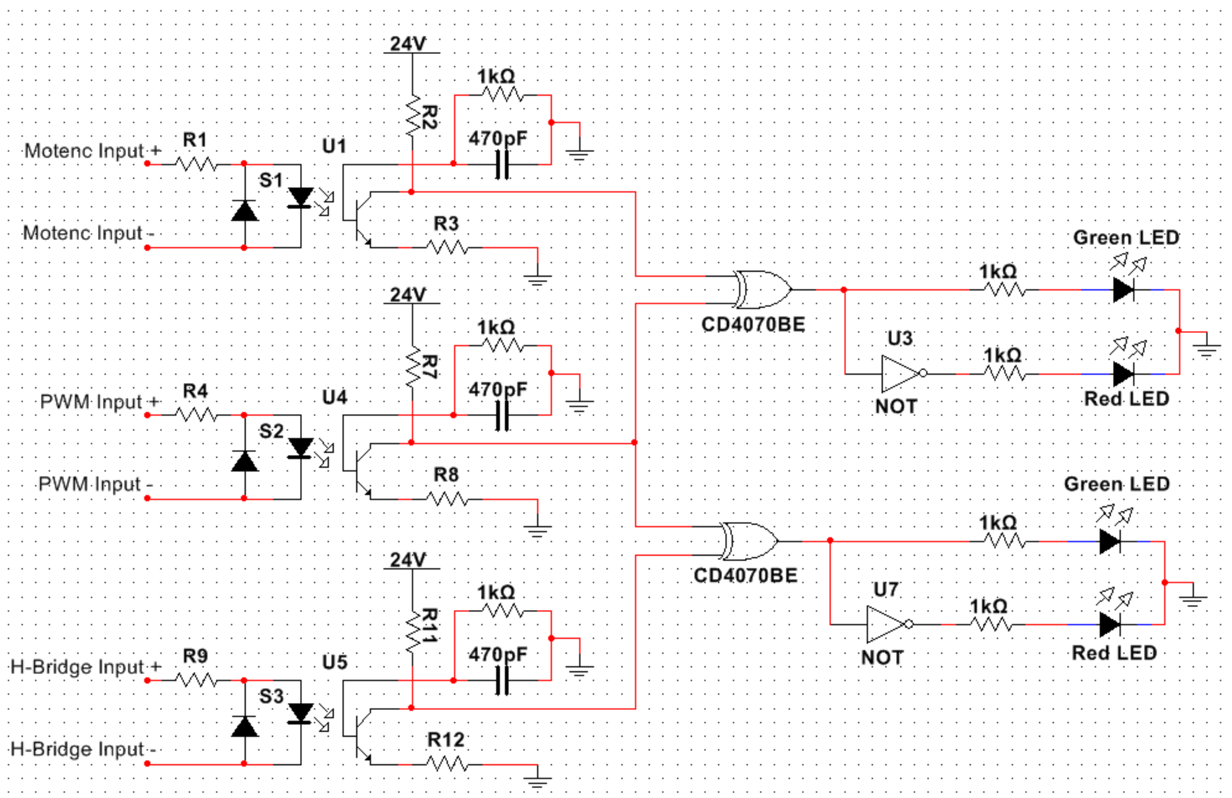
The status circuit is designed to show the current operating state of the controller. Red and green LEDs indicate whether the PWM and H-bridge PCBs are functioning correctly and can be utilized by the user for troubleshooting most problems. In this circuit, an illuminated red LED indicates a malfunctioning H-bridge or PWM circuit, while an illuminated green LED indicates proper functionality.

To determine the operating state of a specific joint's PWM circuit, the status circuit samples the corresponding MOTENC voltage input and compares it with the PWM output. The truth table below shows the desired output of the comparison. It should be easy to see that a single XOR gate and a single NOT gate suffice for the Boolean logic.

MOTENC Output	PWM Output	Error (Red LED)	Correct (Green LED)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

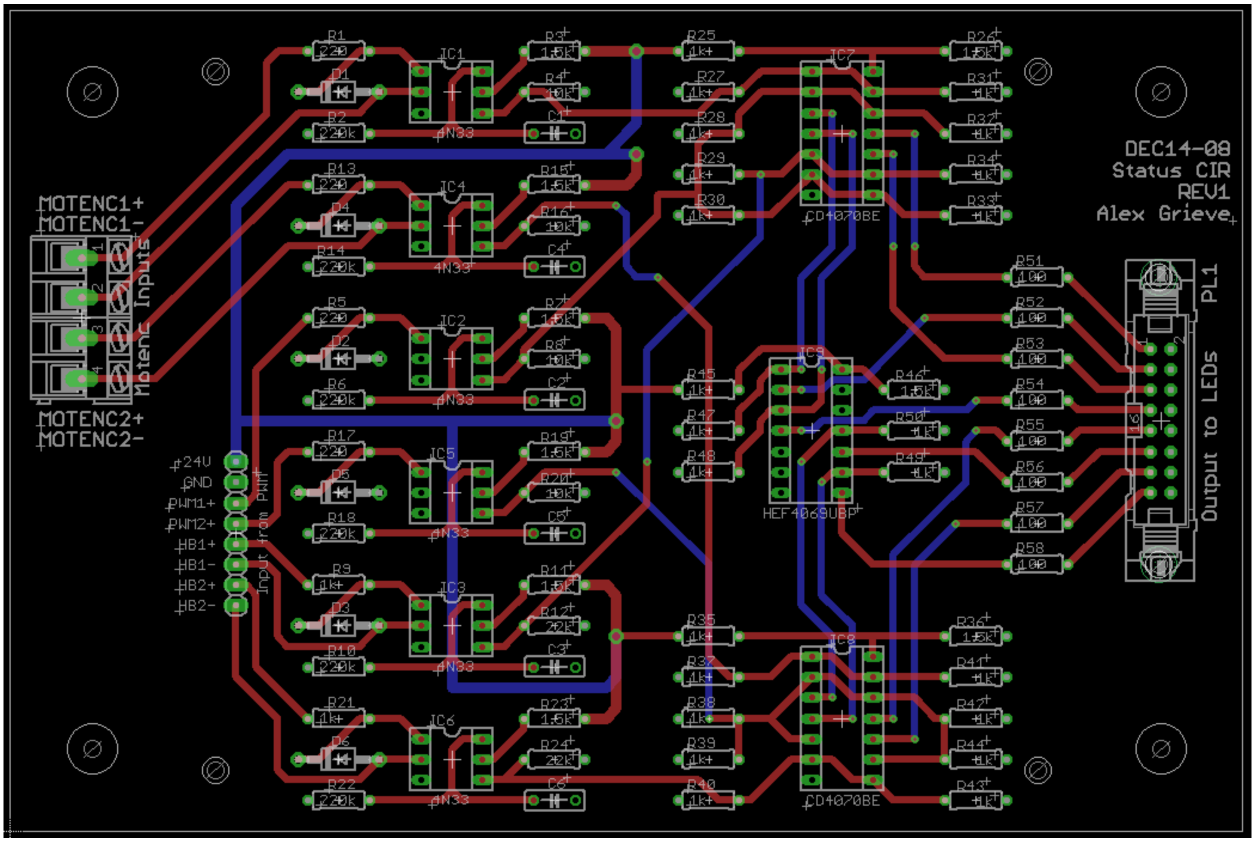
The similar approach can be used to find the operating state of a specific joint's H-bridge circuit. The status circuit samples the corresponding PWM output and compares it with the output of the respective H-bridge. The truth table below shows the correct output for the comparison. It is identical to the previous table, and can also be implemented with a single XOR gate and a NOT gate.

PWM Output	H-bridge Output	Error (Red LED)	Correct (Green LED)
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

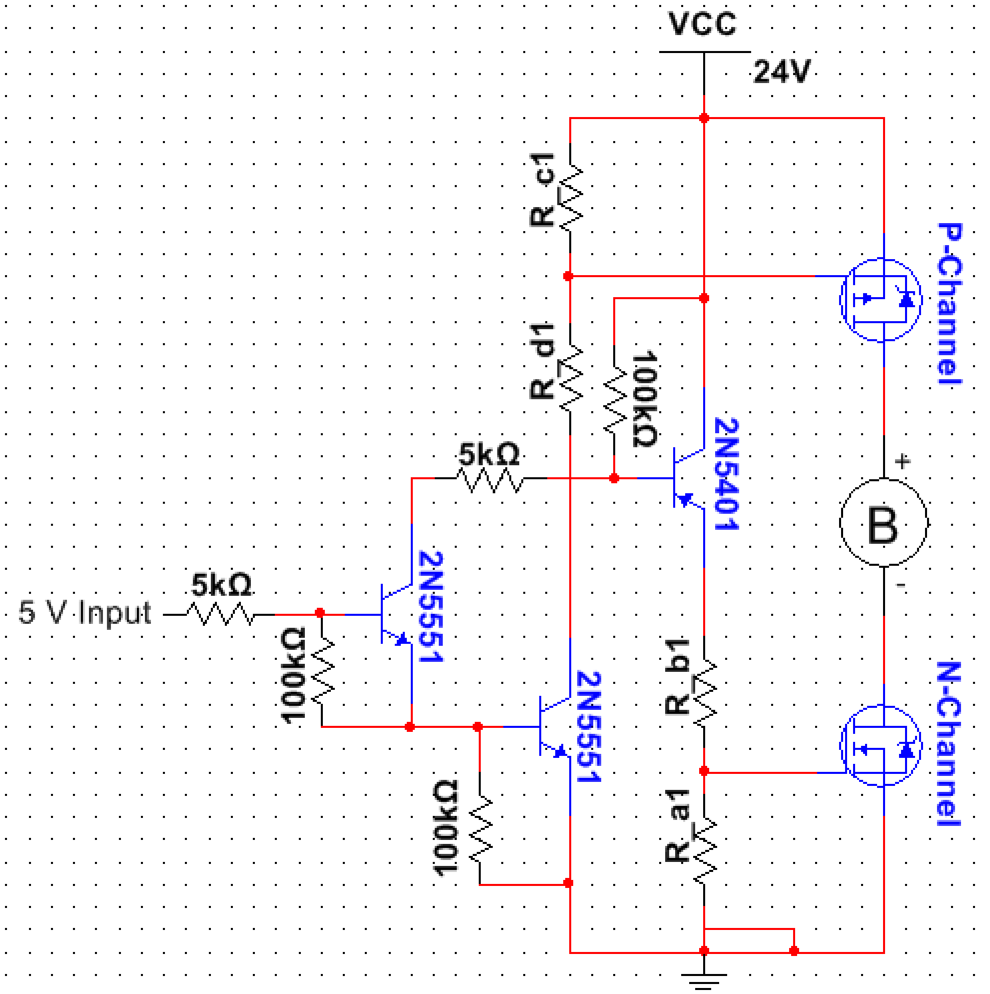


It is important that the status circuit be isolated from the other circuits. This prevents the performance of the PWM and H-bridge from being negatively affected. Opto-isolators were used to decouple the status circuit from the rest of the controller circuitry. Another design consideration is to use 1kΩ pull-down resistors on the inputs to the logic gates. This forces the input to the logic gate to be 0 when no input signal is applied. Additionally, current-limiting resistors need to be placed in series immediately before the red and green LEDs. The values of these resistors will need to be calculated based on the manufacturers recommended current for the LEDs.

The status circuit PCB design can be found below.


DEC14-08
Status CIR
REV1
Alex Grieve
MOTENC1+
MOTENC1-
MOTENC2+
MOTENC2-
Motenc Inputs
+24V
GND
PWM1+
PWM2+
HB1+
HB1-
HB2+
HB2-
Input from PWM
PL1
Output to LEDs
CD4070BE
HEF4069UBP

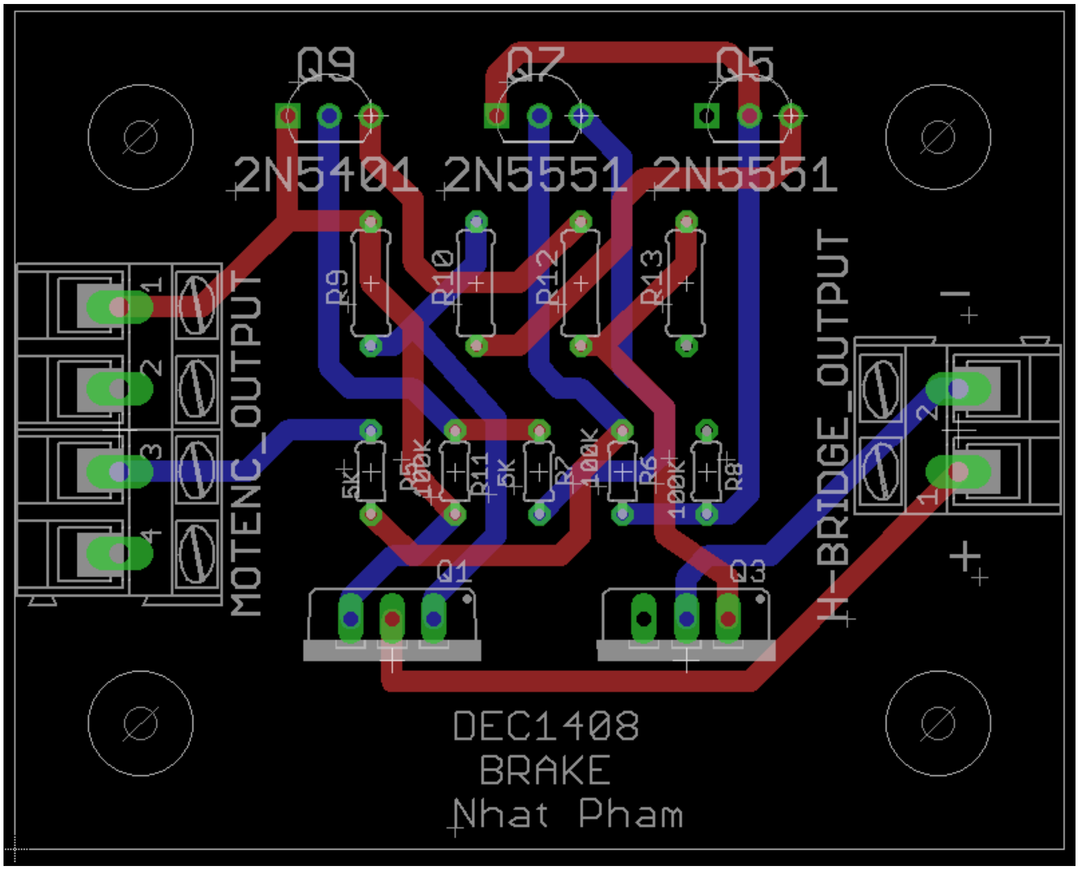
Electromagnetic Brake Release Design



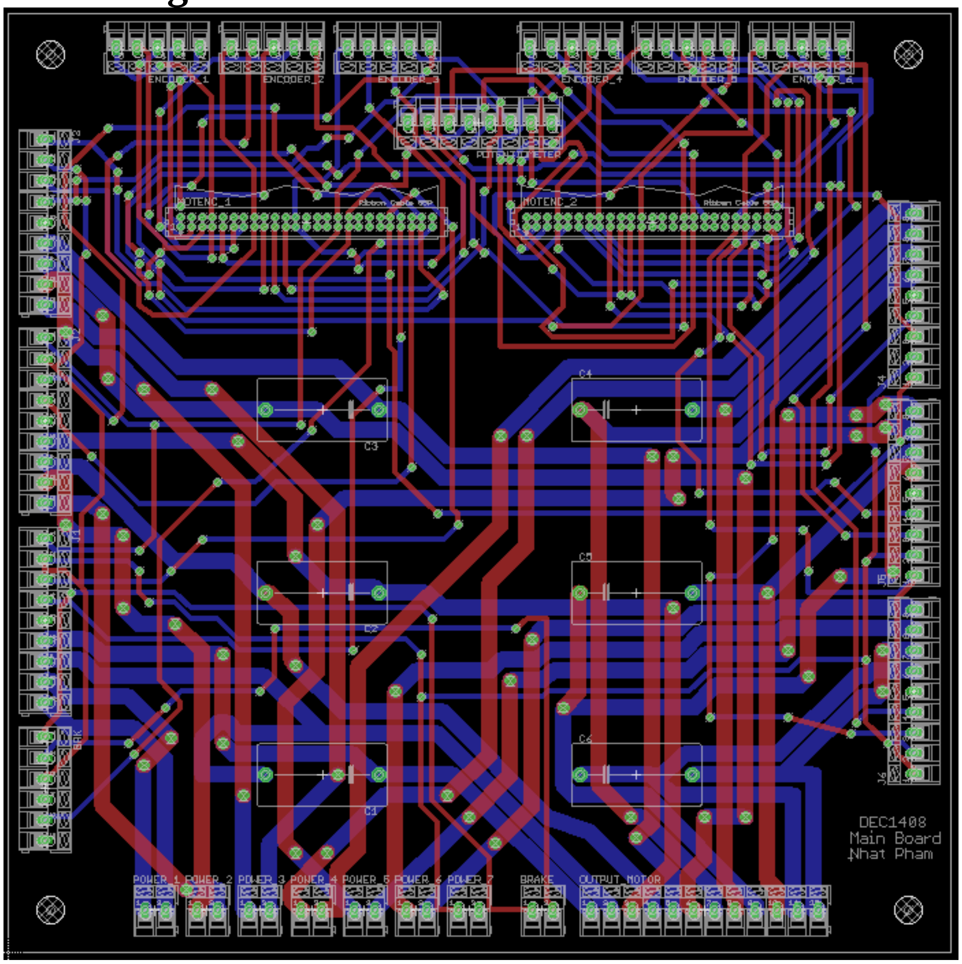
The original robot chassis is equipped with electromagnetic (EM) brakes for the three large motors. The source of power is 24VDC logic requiring approximately 1.5A to fully disengage the friction mechanism in each motor's housing. In order to control the brake via C-code, the PCB for the EM brake function receives signal directly from the MOTENC as well as power from the Meanwell.

Utilizing the idea of the H-bridge, the EM brake circuit contains a half H-bridge. Since the electromagnetic brake switches states at 24 VDC, the half bridge will act as a switch rather than a current controller like the H-bridge. Although the half bridge is a much simpler circuit, the BJTs can't be eliminated because the MOTENC board can only output up to 10 VDC and the MOSFETs require more than 12 VDC to be in the saturation region. Therefore, the BJTs must be used to help drive the gate of the MOSFETs. To ensure that the half bridge always outputs 24 VDC, the input PWMs have to be at 100% duty cycle. This means that instead of a PWM input, a constant 5 VDC signal can be used. See H-bridge design section for schematic and design detail.

The PCB design for the EM brake is below.



Main Board Design



The main board is meant to be the central signal routing point across the controller assemblies. On this board, power and signals handled by the MOTENCs are routed to the peripheral PCBs that manage each joint. The encoders, potentiometers, motors, and EM brake wires from the PUMA harness are also routed to their respective PCBs. The main board also contains six 2.2μF decoupling capacitors. Each capacitor protects one of the PUMA motors from voltage sag during operation and voltage spikes when the motor stops.

General PCB Design

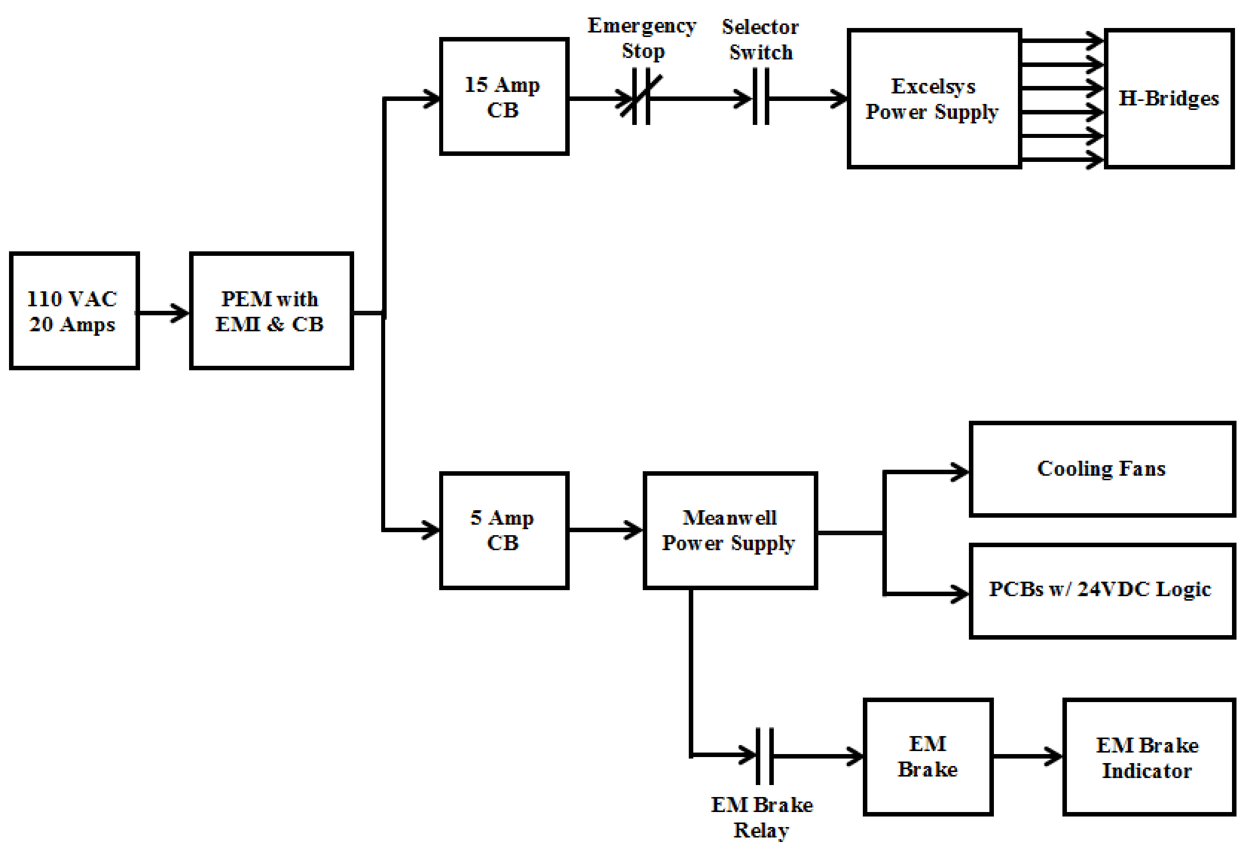
There were two main goals when designing all the PCBs. First, the PCBs were designed to be modular. This allows the circuit boards to be easily repairable, and easy to swap in better or alternative designs for the status, PWM, and/or H-bridge PCBs. Inspiration was drawn from Arduino shields that stack vertically on each other. The team's stack design consists of the status circuit PCB as the base, the PWM control circuit stacking on top of the status circuit, and the H-bridge PCB at the top of the stack.

The second design goal was for the PCBs to effectively dissipate heat. During initial testing, the team noticed that the power MOSFETs on the H-bridge generate a significant amount of heat. Additionally, the power supplies also created a sizeable volume of heat. To protect the electrical components from these heat sources, surface mount and through hole technologies needed to be carefully considered.

Surface mount components require an additional thermal plane on the PCB to dissipate heat, which ultimately drives up the cost of manufacturing the PCBs. This thermal plane also complicates the CAD schematics, increasing the chance of an error occurring in the electrical drawings. Additionally, surface mount components can be difficult to solder by hand. Due to the team's lack of electrical CAD and soldering experience, surface mount components were not a particularly attractive option.

Through hole components do not need a dedicated plane for heat dissipation. Instead, they dissipate heat radially from their solder joints. This keeps the cost of manufacturing the PCBs down, and keeps the CAD drawings simple. Additionally, through hole components are easier to solder than their surface mount brethren. With these considerations in mind, the team opted to use through-hole components instead of surface mount components for the PCBs.

Power Delivery Design



The main consideration for all power in the controller is the limitations of ~120VAC at 20A maximum delivered via wall power. As a general rule of thumb, it is necessary to keep power utilization at or below 80% of the available power supply rating. This limits the controller to a combined consumption of approximately 16A. While inrush currents and small system surges are to be expected, all attempts must be made to balance loads and mitigate transient noise. Such transients can arise from lack of in-line filters and improper wire gauging.

The Schurter power entry module was selected for meeting the 20A capacity of the service supply. It has an electromagnetic interference filter for any noise being transmitted via the AC wall supply. It also has an integrated circuit breaker that will allow the necessary inrush surges over short periods of time (e.g. it can deliver throughput at 132% rated current for nearly 3 minutes), but will cut power nearly instantly at surges above the tripping threshold. This means that a physical fuse is not necessary for this application as the breaker can disrupt continuity before catastrophic damage occurs. The Excelsys and Meanwell power supplies (discussed below) require steady current of 11.5A and 3.3A, respectively. This is 75% of the rated power throughput of the Schurter PEM.

The Excelsys and Meanwell power supplies convert AC into DC power via switching regulators. This regulation is important for efficiency. A side effect of higher efficiency is lighter weight and less heat dissipation (as compared to other power conversion methods.) These power supplies are also known for having internal circuitry to protect themselves and external components from over/under voltages/currents. In the event that damaging surges are sensed, each power supply will shut itself off for a certain length of time (this length of time is predetermined by the manufacturer and is based on factors such as temperature, residual energy, and type of over/under event.) An additional feature that both converters share is a potentiometer on the DC output to vary the voltage. This can allow for compensations due to sag or over voltage when the sub circuits are connected together.

The decision to run at 24VDC logic is highly influenced by the PUMA motors run currents. At 1~4A (depending on the joint), 24VDC delivers the appropriate amount of power within electrical/mechanical limitations. Specifically, the Excelsys contains six isolated modules delivering filtered DC power that can absorb electromagnetic forces from the rotating, inductive motors. The Meanwell supply was chosen due to its rugged performance and reliable output.

In order to protect the power supplies from current and/or voltage surges, circuit protection was added. There is one breaker each per power supply, and they follow the 80% utilization rule as outlined above. The two poles of a breaker manage the AC line and AC neutral conductors, while the AC physical earth (ground) conductor is allowed direct continuity throughout the single phase AC power network. Since the Excelsys and Meanwell have different inrush needs at startup, the tripping curves for the Schurter circuit breakers are dissimilar. The tripping mechanisms are twofold. First, thermal contacts slowly come apart as current passing through a bimetallic switch causes separation from heat. If the contacts heat up too far, the current will not be able to route through the breaker. Additionally, an electrical pulse can force the contacts of the breaker poles apart if too large of a current is instantaneously seen.

A modern convention is to control AC power delivery with lower power DC switches and relays. This makes it less likely for arc flashes and other harmful short circuits to exist at control interfaces. On the enclosure front, a button and a switch control AC throughput to the Excelsys power supply, while a single momentary button can be pressed to release the electromagnetic brake inside the PUMA chassis. These buttons and switch are connected to a 24VDC primary circuits that controls the solenoids of secondary ~120VAC circuits. For the EM brake, there is a 6A relay. This matches the 125% of the 5A output from the Meanwell for protection against over currents damaging this circuit path. The AC throughput for the Excelsys involves a 15A relay, approximately 125% of the 11.5A run current. The relays are also from the same manufacturer, TE Connectivity, though they are from different families within the relay category. This is due to the wiring convention being similar across the product line.

In order to conduct power from the AC source, the DC converters, and to the relays and circuit breakers, DIN rail mountable components were selected. This includes terminal blocks (and their associated subassembly parts). The terminal blocks are Phoenix Contact, and can handle wire from 12-26AWG (stranded recommended) with withstand parameters of 24A and 1000V. Screw terminals were chosen for ease of use; mating wire into spring clamp or other modern feed through terminal blocks can be troublesome for novice technicians. The blocks will be set in five groups of two: AC Line, AC Neutral, AC Ground, DC+ (24VDC), and DC return (0VDC). The jumper bars, end blocks, and end stops from this part family assist in either continuity or separation of signals.

The wire gauges, where applicable, follow the original standards of the PUMA. For example, 16AWG stranded wire was used for the three larger PMDC motors of the robot chassis. It is important to keep the wire as similar as possible when joining to the harness. Dissimilar wire can cause voltage sag and/or polarity shifts, and current parameters can be adversely changed when running upsizing/downsizing wire gauge. Since the harness is prebuilt, it was not possible to rebuild and therefore match the exact wire material in the number and quality of strands; this, in practice, is the best approach when running wire.

In addition to the factors affecting current/voltage for wiring, the need for wiring was utilized for its effect on modular design. Wherever the use of PCBs can be circumvented, wire is used to connect components and sources of power. This is also for cost savings. The general rules for ampacity regarding average current were used to decide upon gauges. Since the runs of most of the wires are less than 2 feet, voltage sag is not a major consideration across connections.

To address safety, finger safe connections were utilized when possible. This refers to having terminations that require a tool to touch any conducting parts, specifically that a human finger cannot get close enough to the conductor to cause shocks and/or arcs. The over/under current/voltage regulation via breakers and relays is another safety consideration. When tracing power dissipation *downstream*, all components are sized to follow the 80% consumption rule. This is to keep temperatures within optimal ranges. The cooling fans use low power (~15Watts apiece) and keep continual air movement. For elements operating near and/or above the limit where de-rating due to ambient temperature can cause undesirable behavior, constant cooling can offset or delay the effect multiplicative factor of heating.

Since the CPU does not directly monitor the power delivery from the Excelsys & Meanwell, it is necessary and sufficient to use relays to inhibit the on state of the motor power and EM brake power. Even if a C function is calling for movement, the motors cannot move unless the user explicitly has the selector switch and emergency stop in the on state. Since a higher class of safety was not a functional requirement, guided contacts and a monitored safety circuit were not built. The electromagnetic brake is directly connected to the Meanwell so that the robot can be moved out of an unsafe position (e.g. if the chassis is injuring a person or the frame itself.) Whenever the brake is released, a yellow indicator light on the front of the enclosure will be illuminated. The manual brake release is a momentary switch so that it has to be purposefully pressed and cannot become locked in a released position. The Meanwell does not have an emergency stop for this reason, and also so that the e-stop and selector switch can be operated on the Excelsys supply unless the Meanwell has suffered catastrophic shutdown.

The enclosure is both a mechanical structure and part of the electronic circuit. The chassis must be grounded to the AC source physical earth (PE) terminal, and all internal grounding traces back to the dissipation that a PE offers. This is to disrupt any ground loops that might injure a user and/or feedback to the CPU via the MOTENCs.

Please see Appendix VII for AutoCAD Electrical schematics.

Enclosure Design

The enclosure was selected for size and form. This allows room for the existing components, and a small percentage of space for future growth of the internal design. The low cost and standard form of server racks means replacement is easier, and the enclosure could possibly be mounted in a server cabinet. The material being aluminum allows for milling of mounting holes on a standard machine.

Please see Appendix VI for AutoCAD panel drawings.

Testing Procedures and Results

MOTENC-Lite DAQ

Procedure
Verify that the MOTENC-Lite boards are fully functional utilizing test code provided by the manufacturer. The test code allows the encoder values to be read & reset, values to be written to the DAC, write to output registers, and read from input registers.

Results
Both MOTENC-Lite boards were tested and fully functional within the manufacturer's specifications. The MOTENCs did not create noticeable lag between the software and hardware interfaces.

C library

Procedure
To verify that the library functions are operating correctly, a small test program calls each library function with different parameters. If the function call activates or deactivates a pin or pins on the MOTENC boards, the team verifies the behavior by connecting a multi-meter to the affected MOTENC board pin(s).

Results
All functions are working as designed. The C library correctly initializes the MOTENC boards, correctly reads quadrature encoder counts and potentiometer values, correctly changes the MOTENCs' analog outputs, and toggles the PUMA electromagnetic brake on and off.

PWM Control

Procedure
The first step is to construct the left half of the PWM circuit shown in the PWM design. The reason is to ensure that the first half will output any desire period/frequency. Then, apply 5 volt DC to power the first half of the PWM. Use an oscilloscope to check the output, and tune the R1 potentiometer to the desired period/frequency. Once the first half of the PWM works properly, construct the second half of the circuit. Apply at least 5 volts to the V+ of the current source, and then examine the threshold pin of the second 555 timer. Next slowly increase the 5 volts at the V+ of the current source to get a linear ramp from 0 to 5 volts. Then apply an input of 0 to 5 volts DC to the control voltage pin, 2.5 volts will correlate with a 50% duty cycle, as opposed to a continuous output (5 volts or 0 volt input, 100% or 0% duty cycle). Now connect the oscilloscope to examine the output pin of the second 555 timer. The signal might not be as clear as first; however, adjust the current source's potentiometer to get a clear signal (aim for 50% duty cycle.)

Results
After connecting 5 volts DC to the first half of the circuit and tuning the R1 potentiometer to desired period/frequency, the first half of the PWM circuit outputs a clear signal with the desired period. After constructing the second half, 24 volts DC was applied to the current source and the output was calibrated by adjusting the potentiometer of the current source. The circuit outputs a PWM proportional to the control voltage; 0 to 5 volts scales linearly to 15% to 95% duty cycle. Notice that there is limitation to this circuit; the range of operation only varies between 15% and 95% duty cycle. Therefore, to turn off the PWM (0% duty cycle), the voltage at the control voltage pin must be at -1 volts. In addition, the team also ran into problems when integrating multiple PWM circuits together. The output signal became really distorted and very noisy. The team would like to come up with a different design if time allows.

H-Bridge

Procedure
To ensure that the H-bridge works properly, first calculate the right resistor values so that the MOSFETs will be in saturation region with a 12 volt power supply. In addition use a different motor to avoid any damage to the original motor on the robot. Apply 12 volts to the power input on the H-bridge, and then connect the motor. A quick check before connecting the motor is to feel if the MOSFET overheats without any input. If the MOSFET feels hot, the circuit configuration is incorrect. Otherwise, connect the motor to the H-bridge. Insure that nothing is connected to the inputs, ensuring that the motor will not energize. To energize the motor, apply a 5 volt PWM to the input, and vary the PWM duty cycle to adjust the speed of the motor. Repeat the last step with the opposite input, to ensure that the motor rotates in the opposite direction.

After verifying that the H-bridge works as designed, repeat the steps above with the actual motor on the robot with extra caution. However, this time use a 24 volt 8 amp power supply. Be sure to unlock the EM brake with 24 volts DC before attempting to apply any inputs to the H-bridge.

Results
In connecting the 24 volts to power the H-bridge, no overheating issues were experienced with the MOSFET. However, the MOSTFETs generated a lot of heat during longer operation periods, which prompted the addition of heat sinks and cooling fans into the design. Once the motor was connected, without any input the motor was not energized. After applying a 5 volt PWM, the motor was energized. Alternating inputs changed the direction of motor, along with changes in speed when varying the PWM duty cycle.

Power Delivery

Procedure
Connect individual components to their respective power supply (Meanwell and Excelsys). The components will be connected in isolation and then together while monitoring current (amps)

and voltages (DC/AC) to ensure no under/over current/voltage situations occur. Once the proper power needs are witnessed, connect relays, switches, and circuit breakers to ensure no under/over current/voltage situations occur.

Results
In connecting the control and protection components, no over/under power issues were seen. The six PMDC motors developed voltage sag of 4~7VDC. This sag was corrected by the use of 2.2μFarad capacitors across the power terminals of each motor.

Status Circuit

Procedure
Connect MOTENC analog outputs, PWM outputs, and H-Bridge outputs to respective opto-isolator, and test that there is output from the opto-isolator. The opto-isolator output must be in the range of 10-15VDC for the logic gates to function correctly. Next, verify that the correct LED is illuminated for all possible inputs to the status circuit.

Results
TBD

System Integration

Procedure
Once all components and subassemblies are tested and working individually, they can be iteratively added to the system and tested for proper functionality. The following steps outline the process of integrating all components into the system:

1. C library and MOTENC boards are coupled and tested
2. PWM control circuits are added to the system, and tested for proper output from C function calls
3. H-Bridges are added to the system, and tested for proper output from C function calls
4. Power is coupled to the H-Bridges and the PUMA is connected to the system; Check for proper joint movement based on specific C function calls
5. Add auxiliary circuits (status circuit, relays, switches, fans, EM brake, and LED indicators) and verify they are working correctly

Results
First, the power delivery system was tested for proper functionality. The team verified that power was correctly being routed to the power supplies, and that the relays and switches functioned properly. A couple minor issues were found, mainly wiring the switches incorrectly (normally open/normally closed.) These were easily fixed. The output of the power supplies were then connected to the PWM circuits and H-bridges as each were tested.

Second, the MOTENCs were connected to the PWMs of the lower three PUMA joints. A test program on the computer changed the output voltages on the MOTENCs which propagated to the input of the PWM circuits. The outputs of the PWM circuits were monitored with an oscilloscope to verify the input voltage correctly changed the duty cycle of the output signal. No problems were found. The process was repeated for the PWMs of the upper three PUMA joints. Again, no issues were found.

Next, the EM brake was integrated into the system and was found to be functioning properly. The H-bridges for the lower three joints were then connected to their respective PWM circuits. A multimeter was used to check that the H-bridges were correctly operating when receiving input from the PWMs. No problems were found, so the process was repeated for the upper three joints. Again, no issues were found.

Finally, the H-bridges were connected to the motors. The C code test program first unlocked the EM brake, and then slowly moved the bottom three joints in both directions. The speed of joint movement was gradually increased, and no problems were encountered. Potentiometer values and encoder counts were also monitored and seemed to make sense. The process was then repeated for the upper three motors. A stray semi-colon in the C code initially prevented the upper three motors from moving, but once corrected, they functioned correctly.

Integration of the status circuits is still pending.

Appendix I: Operation Manual

Electromagnetic Brake Operation

The electromagnetic brake (EM brake) that locks/unlocks the motors for the bottom three joints can be controlled via computer interface or switches/buttons on the front panel of the controller. Whenever the EM brake is unlocked, a yellow (amber) light on the front of the enclosure will illuminate.

Front Panel

A momentary ON button is provided to unlock the EM brake in the event that the robot has failed/stopped in an undesirable position. In this manual operation, the brake will only be released while the button is pressed.

WARNING – Injury/damage will occur to the operator and the robot if the brake is released and the robot is allowed to drop to the ground or onto robot body parts.

Computer Interface

The C code provides an easy method for controlling the EM brake. After a successful call to `puma_init()`, the EM brake can be turned released by called `brake_off()` and applied by calling `brake_on()`. The user should check the return value of these functions, as they return 1 when the brake was successfully released or applied, or 0 if there was a problem.

When writing custom programs, the user should always start with the code template provided in the file `skel.c`. The template provides extra safety by applying the brakes when the program terminates unexpectedly or prematurely. It also shuts off all MOTENC outputs so that no joints will continue moving after the program ends.

Motor Operation

The six motors in the PUMA robot have a two-step process to induce movement of the robot joints. These two steps are allowing power to be distributed to the motors via the controller and coding torque/direction via the computer.

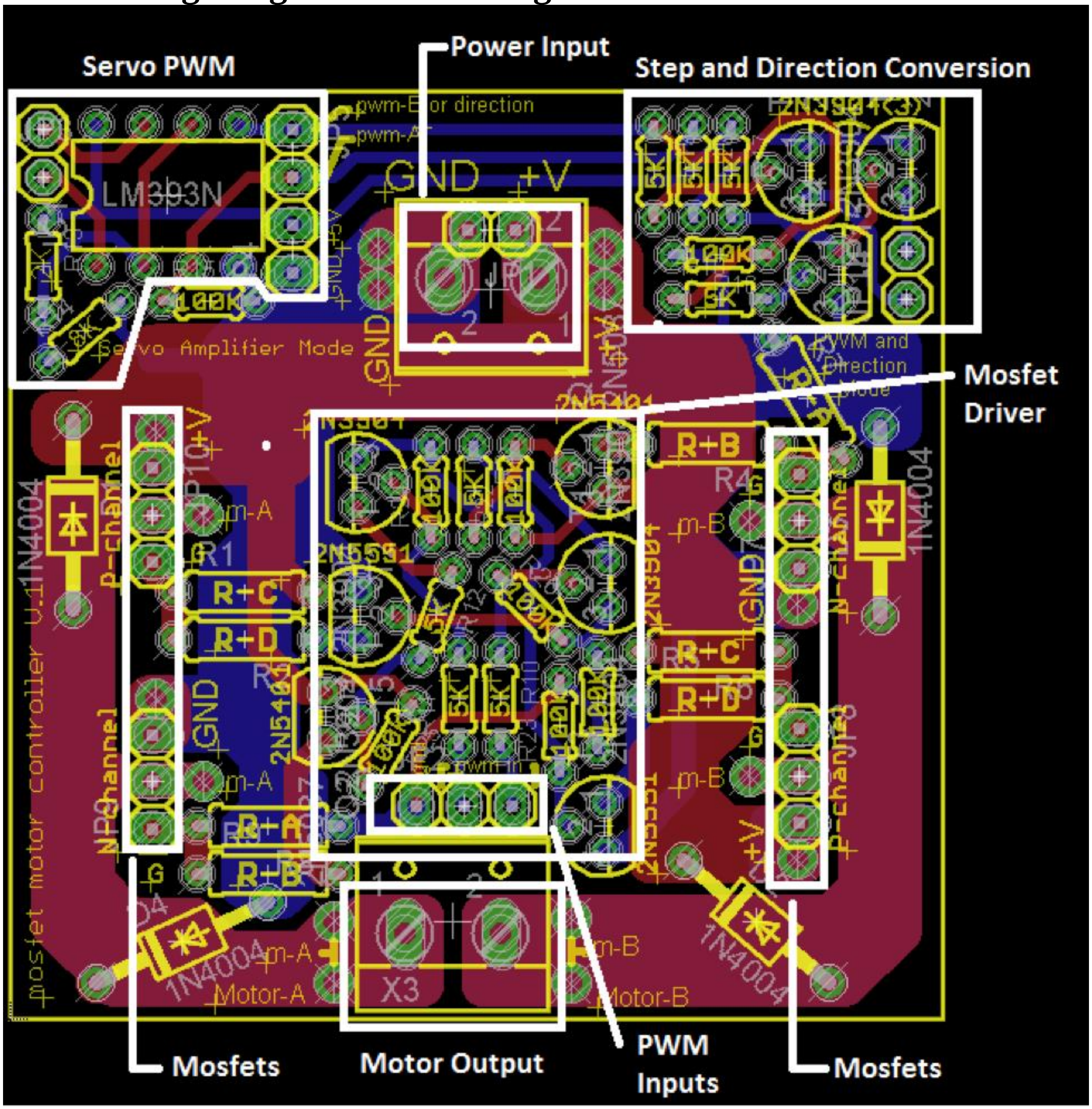
The front panel of the enclosure has a button and a switch tied directly to the internal power supply for the six motors. There is an emergency stop (E-stop) button to stop any movement of the robot when damage or injuries are possible, and a Selector Switch to allow power through the circuit. To allow powering of the motors, follow the steps below.

1) The E-stop stays engaged when fully pressed. To release, twist the button to the right.
2) The E-stop must be in the OFF position and the Selector Switch must be in the ON position to allow C-code to induce joint movement.
3) The PUMA can now be controlled through C-code. It is highly recommended that user programs use the file `skel.c` as a template for proper initialization and safe shutdown of the PUMA.
4) To move a motor, the user should:

a) Call the function `apply_torque()` or `write_dac()` with proper arguments. This will set the desired joint in motion.
b) Consecutively call `read_pot()` or `read_encoder()` to monitor the moving joint's position.
c) When the desired position has been reached, call the function `apply_torque()` with a torque value of 0 or `write_dac()` with the value `PWM_OFF`. This will stop the joint from moving.

Appendix II: Initial Design Versions

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. It was used as a starting point for creating a circuit design for the H-bridge. The Servo PWM and Step and Direction Conversion circuits in the drawing above do not work, so they were omitted from the final H-bridge circuit.

Appendix III: Other Considerations

PWM Circuit

Due to the instability of the PWM circuit, a new design should be taken into consideration. Problems with the current design are listed below.

1. The 555 timers are unreliable and imprecise. A new design needs to have a more stable way of creating oscillations.

2. The duty cycle of the current PWM is about 15-90%. This means that without any input to the PWM (i.e. 0 VDC), the PWM will generate a square wave with 15% duty cycle. This is a serious problem when connecting the PWM to the H-bridge. The H-bridge can only operate in one direction/side at a time. If a PWM is generating a signal with 15% duty cycle when it is technically off, both sides of the H-bridge would become active at the same time. This causes shoot through in the H-bridge and will ultimately burn the MOSFETs and the motors. The fix applied to the current PWM design is to set the PWM input to -1 VDC to completely turn them off. An improved design needs to have an operational range between 0-100% so the PWM can be turned off without needing to drive a negative voltage to the inputs.

3. One problem the current PWM circuit has is that it introduces noise into the system. When more than one PWM circuit is on (i.e. two joints moving at the same time), the noise generated by each PWM circuit negatively impacts the output of the others. This leads to jitter in each PWM signal, causing the duty cycle to be unstable. A new PWM design should not interfere with other PWM circuits.

H-Bridge Circuit

Although the current H-bridge works well, there are a couple of improvements that could be made. They are listed below.

1. The H-bridge is driven by a PWM signal with an adjustable frequency. The adjustable frequency may produce audible noise when sent to the motor. To eliminate the audible noise, the frequency of the PWM needs to be increased. However, the system becomes nonlinear as the frequency is increased because the transistors in the current H-bridge design aren’t capable of picking up such high frequency. New transistors should be selected that have faster switching characteristics.

2. Safety logic is currently implemented in software that prevents both sides of the H-bridge from being activated at once. However, a physical logic circuit should be constructed to protect the H-bridges against software errors.

Status Circuit

The current status circuit design does not output the correct state when the input voltage from the MOTENCs is less than 3 VDC. The opto-isolators correctly pick up the inputs from the MOTENC, PWM, and H-bridge at this voltage; however, the opto-isolators output a voltage that is considered “logic low” by the XOR and NOT gates. Although this problem does not directly affect the performance of the PUMA, it will cause the LEDs to turn red when the PUMA is actually operating in a valid state. This could potentially confuse the end users.

One solution to this problem would be to place Schmitt triggers after the opto-isolators. The Schmitt triggers could be tuned so that they always output a voltage that the XOR and NOT gates consider “logic high” whenever a positive voltage is applied to the Schmitt trigger inputs. This way, the output of the status circuit would be correct throughout the MOTENCs’ 0-5 VDC input range, effectively solving the current status circuit problem.

Common Reference Voltage

The current system shares the same common reference voltage throughout sub-circuits. Beside the noises in the PWM circuits, it had not created any issues within the system. However, sharing a same common reference voltage could damage the system with just a single wiring error. The user could accidently apply a 24 VDC to the common instead of an actual common. Therefore, it is recommended that future designs decouple common between sub-circuits to avoid damaging the system.

Appendix IV: Bill of Materials

PWM Control Circuits Parts List

Item	QTY	Manufacturer Part #	Unit Cost	Total Cost
1K-ohm 1/4W resistor	12	CF18JT100K	$ 0.10	$ 1.20
100K-ohm potentiometer	12	3362P-1-104LF	$ 1.02	$ 12.24
10K-ohm potentiometer	12	3386P-1-103LF	$ 1.60	$ 19.20
100nF capacitor	36	K104Z15Y5VE5TH5	$ 0.44	$ 15.84
Current Source	12	LM234Z-6/NOPB	$ 1.27	$ 15.24
555 Timer	24	LM555CN	$ 0.46	$ 11.04
Screw terminal, 5 pos, 5.00mm pitch	6	OSTTC050162	$ 0.84	$ 5.04
Headers, Male, 40 pin, 0.100" pitch	2	PREC040SAAN-RC	$ 0.56	$ 1.12
Headers, Female, 4 pos, 0.100" pitch	12	PPTC041LFBN-RC	$ 0.60	$ 7.20
			Total:	$ 88.12

H-Bridges Parts List

Item	QTY	Manufacturer Part #	Unit Cost	Total Cost
100K-ohm 1/8W resistor	36	CF18JT100K	$ 0.10	$ 3.60
5K-ohm 1/8W resistor	24	CF18JT5K10	$ 0.10	$ 2.40
1K-ohm 1/4W resistor	24	CF14JT1K00	$ 0.10	$ 2.40
2.2K-ohm 1/4W resistor	24	CF14JT2K20	$ 0.10	$ 2.40
NPN transistor	24	2N5551TA	$ 0.21	$ 5.04
PNP transistor	12	2N5401G	$ 0.40	$ 4.80
P-channel power MOSFET	12	FQPF47P06	$ 2.03	$ 24.36
N-channel power MOSFET	12	FQPF70N10	$ 2.07	$ 24.84
Heat Sink	24	581002B00000	$ 1.30	$ 31.20
Diode, 4A, 40V	12	SB540E-G	$ 0.44	$ 5.28
Screw terminal, 2 pos, 0.200" pitch	12	OSTTC022162	$ 0.41	$ 4.92
Headers, Male, 40 pin, 0.100" pitch	2	PREC040SAAN-RC	$ 0.56	$ 1.12
			Total:	$ 112.36

Status Circuits Parts List

Item	QTY	Manufacturer Part #	Unit Cost	Total Cost
100 ohm 1/4W resistor	48	CF14JT100R	$ 0.10	$ 4.80
220 ohm 1/4W resistor	24	CF14JT220R	$ 0.10	$ 2.40
1K-ohm 1/4W resistor	150	CF14JT1K00	$ 0.10	$ 15.00
1.5K-ohm 1/4W resistor	54	CF14JT1K50	$ 0.10	$ 5.40
10K-ohm 1/4W resistor	24	CF14JT10K0	$ 0.10	$ 2.40
22K-ohm 1/4W resistor	12	CF14JT22K0	$ 0.10	$ 1.20

220K-ohm 1/4W resistor	36	CF14JT220K	$ 0.10	$ 3.60
470pF capacitor	36	K471K10X7RF5UH5	$ 0.30	$ 10.80
1N4004 diode	36	1N4004-TP	$ 0.11	$ 3.96
RGB LEDs	28	WP154A4SUREQBFZGC	$ 1.18	$ 33.04
4N33 Opto-isolator	36	4N33	$ 0.65	$ 23.40
XOR Gates IC	12	CD4070BE	$ 0.52	$ 6.24
NOT Gates IC	6	HEF4069UBP,652	$ 0.98	$ 5.88
6 DIP IC Socket	36	A06-LC-TT	$ 0.24	$ 8.64
14 DIP IC Socket	18	ED14DT	$ 0.17	$ 3.06
16 Pin IDC Headers	6	AWH16G-0222-T-R	$ 1.33	$ 7.98
16 Pin IDC Connector	6	AWP16-7241-T-R	$ 0.68	$ 4.08
Ribbon Cable, 16 conductor, 100'	1	AWG28-16/G/300	$ 25.32	$ 25.32
Screw terminal, 4 pos, 0.200" pitch	8	OSTTC042162	$ 0.67	$ 5.36
Headers, Female, 8 pos, 0.100" pitch	6	PPTC081LFBN-RC	$ 0.86	$ 5.16
			Total:	$ 177.72

Main Board Parts List

Item	QTY	Manufacturer Part #	Unit Cost	Total Cost
50 Pin IDC Headers	2	AWH-50G-0232-T	$ 3.04	$ 6.08
50 pin IDC Headers, Panel mount	2	AWH50G-E232-IDC-R	$ 4.56	$ 9.12
Ribbon Cable, 50 conductor, 100'	1	AWG28-50/G/300	$ 79.33	$ 79.33
Screw terminal, 2 pos, 5.00mm pitch	20	OSTTC020162	$ 0.41	$ 8.20
Screw terminal, 5 pos, 5.00mm pitch	12	OSTTC050162	$ 0.84	$ 10.08
Screw terminal, 8 pos, 5.00mm pitch	1	OSTTC080162	$ 1.33	$ 1.33
Screw terminal, 12 pos, 5.00mm pitch	1	OSTTC120162	$ 1.86	$ 1.86
2.2uF capacitor	6	ECQ-U2A225ML	$ 1.70	$ 10.20
			Total:	$ 126.20

Power Delivery Parts List

Item	QTY	Manufacturer Part #	Unit Cost	Total Cost
Fan, 120mm, 24VDC, 0.7A	2	AFB1224SHE-CR00	$ 29.99	$ 59.98
Fan Guard, 120mm	2	8170	$ 0.85	$ 1.70
Circuit Breaker, 24VDC, 15A	1	K10P-11D15-24	$ 11.50	$ 11.50
Circuit Breaker socket	1	27E895	$ 7.56	$ 7.56
Circuit Breaker hold down clip	1	20C426	$ 0.82	$ 0.82
Circuit Breaker, 480VAC, 15A	1	AS168X-CB2F150	$ 55.61	$ 55.61
Circuit Breaker, 480VAC, 5A	1	AS168X-CB2G050	$ 76.61	$ 76.61
Power Entry Module, Wall power to IEC	1	EF12.0572.1110.01	$ 68.36	$ 68.36
Terminal Block, AC Line	2	3045088	$ 1.16	$ 2.32
Terminal Block, AC Neutral	2	3045075	$ 1.18	$ 2.36

Terminal Block, AC Ground	2	3044092	$ 4.05	$ 8.10
Terminal Block, DC Common	2	3044076	$ 1.11	$ 2.22
Terminal Block, DC +24V	2	3044089	$ 1.09	$ 2.18
Terminal Block, End stops	5	3047028	$ 0.56	$ 2.80
Terminal Block, End bracket	4	3022276	$ 1.03	$ 4.12
Terminal Block, Jumpers	5	3030161	$ 0.61	$ 3.05
Terminal Block, Marking Card	3	1050004	$ 1.19	$ 3.57
Hook-Up Wire, 22AWG, 25', Blue	1	3132-22-1-0500-005-1-TS	$ 19.35	$ 19.35
Relay, 24VDC, 6A	1	3-1416100-0	$ 22.66	$ 22.66
Switch, Momentary, 24VDC, 10A	1	A22-TR-11M	$ 27.30	$ 27.30
Switch, Selector, 2 pos, 120VAC, 10A	1	A22RS-2M-11	$ 17.06	$ 17.06
Switch, Emergency Stop, 120VAC, 10A	1	A22E-M-11	$ 37.80	$ 37.80
Pilot Light, 24V, Yellow	1	M165-TY-24D	$ 27.09	$ 27.09
Power Line, 15A, Excelsys to AC	1	800-1403-2-SJT0-BL-00100	$ 5.45	$ 5.45
Power Line, 20A, IEC-320-C19	1	P049-010	$ 34.72	$ 34.72
Power Supply, DIN Rail, 120W, 24V, 5A	1	DR-120-24	$ 41.61	$ 41.61
Power Supply, Chasis, 1200W, 6 Slot	1	UX6	$ 238.96	$ 238.96
Power Supply, Module, 5-28VDC, 5A	3	XGE	$ 73.33	$ 219.99
Power Supply, Module, 19.2-26.4VDC, 8.3A	3	XGB	$ 83.24	$ 249.72
			Total:	$ 1,254.57

Enclosure, Wire, Miscellaneous Parts List

Item	QTY	Manufacturer Part #	Unit Cost	Total Cost
Enclosure, Rack mount, Open chassis	1	RM-14225	$ 119.85	$ 119.85
Enclosure, Cover panel	2	TBC-14253	$ 33.60	$ 67.20
Enclosure, Handles	2	H-9113-B	$ 10.60	$ 21.20
Washer, flat, nylon	100	3365	$ 0.08	$ 7.74
T-Nut, 10 series, 1/4-20	50	3382	$ 0.12	$ 6.00
Screw, Button head, 1/4-20, 3/4"	50	Unknown	$ 0.10	$ 5.00
Wire, 12AWG, 25', Green	1	WH612-05-25	$ 16.84	$ 16.84
Wire, 12AWG, 25', White	1	WH612-09-25	$ 16.84	$ 16.84
Wire, 12AWG, 25', Black	1	WH612-00-25	$ 16.84	$ 16.84
Wire, 14AWG, 25', Green	1	WH614-05-25	$ 9.52	$ 9.52
Wire, 14AWG, 25', White	1	WH614-09-25	$ 9.52	$ 9.52
Wire, 14AWG, 25', Black	1	WH614-00-25	$ 9.52	$ 9.52
Wire, 16AWG, 25', Red	1	WH616-02-25	$ 6.26	$ 6.26
Wire, 16AWG, 25', Black	1	WH616-00-25	$ 6.26	$ 6.26
Wire, 18AWG, 25', Purple	1	WH18-07-25	$ 5.42	$ 5.42
Wire, 20AWG, 25', Red	1	WH20-02-25	$ 4.28	$ 4.28
Wire, 20AWG, 25', Black	1	WH20-00-25	$ 4.28	$ 4.28

Wire, 20AWG, 25', Blue	1	WH20-06-25	$ 4.28	$ 4.28
Wire, 22AWG, 25', Blue	1	WH22-06-25	$ 3.11	$ 3.11
Wire, 22AWG, 25', White	1	WH22-09-25	$ 3.11	$ 3.11
Wire, 22AWG, 25', Yellow	1	WH22-04-25	$ 3.11	$ 3.11
Wire, 22AWG, 25', Orange	1	WH22-03-25	$ 3.11	$ 3.11
Wire, 22AWG, 25', Red	1	WH22-02-25	$ 3.11	$ 3.11
Wire, 22AWG, 25', Black	1	WH22-00-25	$ 3.11	$ 3.11
MOTENC-Lite PCI card	2	7541	$ 595.00	$ 1,190.00
PWM PCB	6	-	$ 27.85	$ 167.10
H-Bridge PCB	6	-	$ 27.85	$ 167.10
Status Circuit PCB	6	-	$ 40.00	$ 240.00
Main board PCB	1	-	$ 135.02	$ 135.02
EM Brake Release PCB	1	-	$ 8.33	$ 8.33
			Total:	$ 2,119.71
			Grand Total:	$ 3,878.68

Appendix V: C Code

Puma_config.h

```c
/*
 * Configuration values for the PUMA robot. Be careful when changing
 * these values, as you can seriously damange the PUMA and/or the
 * Motenc-Lite cards.
 *
 * puma_config.h
 *
 * Alex Grieve, 2014
 */

#ifndef PUMA_CONFIG_H
#define PUMA_CONFIG_H


// This must be <= the actual amount of PCI memory
#define PCI_MEM_LEN 512

// Motenc-Lite identification values (DON'T CHANGE)
#define VENDOR 0x10b5
#define DEVICE 0x3001

// Hardware Board ID (Jumper J7 on Motenc-Lite Board)
#define LOWER_BOARD_ID 0x00 // J1-J3 controlled by this Motenc card
#define UPPER_BOARD_ID 0x01 // J4-J6 controlled by this Motenc card

// Joints
#define J1 0
#define J2 1
#define J3 2
#define J4 3
#define J5 4
#define J6 5

// Max/min DAC values
#define DAC_MAX 4.90
#define DAC_MIN -1.00

// Voltage to turn off the PWM
#define PWM_OFF -1.00

// Motor torque constants (Unimation's numbers)
#define J1_K .260
#define J2_K .260
#define J3_K .260
#define J4_K .090
#define J5_K .090
#define J6_K .090

// Max/min current that can be delivered to a joint
#define J1_I_MIN 0.25
#define J1_I_MAX 2.0
#define J2_I_MIN 0.0
```

```
#define J2_I_MAX 0.0
#define J3_I_MIN 0.0
#define J3_I_MAX 0.0
#define J4_I_MIN 0.0
#define J4_I_MAX 0.0
#define J5_I_MIN 0.0
#define J5_I_MAX 0.0
#define J6_I_MIN 0.0
#define J6_I_MAX 0.0

// Encoder resolution by joint (degrees/bit)
#define J1_ENC_RES .0057557
#define J2_ENC_RES .00417408
#define J3_ENC_RES .0067098
#define J4_ENC_RES .0047361
#define J5_ENC_RES .00500625
#define J6_ENC_RES .00469177

// Joint rotational limits (degrees)
#define J1_ROT_LIM_DEG 320
#define J2_ROT_LIM_DEG 250
#define J3_ROT_LIM_DEG 270
#define J4_ROT_LIM_DEG 300
#define J5_ROT_LIM_DEG 200
#define J6_ROT_LIM_DEG 532

// Joint rotational limits (encoder counts)
#define J1_ROT_LIM_CNT 55597
#define J2_ROT_LIM_CNT 59893
#define J3_ROT_LIM_CNT 40240
#define J4_ROT_LIM_CNT 63343
#define J5_ROT_LIM_CNT 39950
#define J6_ROT_LIM_CNT 113390

// Potentiometer ranges (volts)
#define J1_POT_VMIN 0.00
#define J1_POT_VMAX 5.00
#define J2_POT_VMIN 0.00
#define J2_POT_VMAX 5.00
#define J3_POT_VMIN 0.00
#define J3_POT_VMAX 5.00
#define J4_POT_VMIN 0.00
#define J4_POT_VMAX 5.00
#define J5_POT_VMIN 1.88
#define J5_POT_VMAX 3.76
#define J6_POT_VMIN 0.00
#define J6_POT_VMAX 5.00

#endif
```

Puma.h

```
/*
 * Primary functions to interact with the Motenc-Lite
 * cards and the PUMA controller.
```

```
 *
 * puma.h
 *
 * Alex Grieve, 2014
 */

#ifndef PUMA_H
#define PUMA_H

#include "puma_config.h"
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <pci/pci.h>
#include <sys/io.h>
#include <sys/mman.h>

/*********************************************************
 *                  Primary functions                    *
 *********************************************************/
/*
 * Finds and initializes Motenc-Lite cards for use.
 *
 * RETURN:  1 on successfull initialization, 0 otherwise
 */
uint8_t puma_init();

/*
 * Retrieves the unsigned encoder count from the Motenc-Lite
 * card for a specific joint.
 *
 * INPUT:       joint       Desired joint (J1 - J6)
 * RETURN:  Unsigned encoder count, 0 if error occurs
 */
uint32_t read_encoder(uint8_t joint);

/*
 * Retrieves the voltage across a specific joint's potentiometer
 * using the Motenc-Lite ADC.
 *
 * INPUT:       joint       Desired joint (J1 - J6)
 * RETURN:  Voltage across the potentiometer, 0 if error occurs
 */
float read_pot(uint8_t joint);

/*
 * Apply torque to a specific joint. This function uses the
 * following equation for torque:
 *
 *  torque = k * I
 *
 * where k is the motor torque constant and I is the current
 * flowing through the motor.
 *
```

```
 * INPUT:       joint           Desired joint (J1 - J6)
 *                      torque      Torque to apply
 *                      direction   Desired motor direction (0 or 1)
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t apply_torque(int joint, float torque, int direction);

/*
 * Writes a voltage value to the Motenc-Lite DAC for a specific
 * joint and motor direction.
 *
 * INPUT:       joint           Desired joint (J1 - J6)
 *                      voltage     Voltage to write
 *                      direction   Desired motor direction (0 or 1)
 * RETURN:  1 on successful write to DAC, 0 otherwise
 */
uint8_t write_dac(int joint, float voltage, int direction);

/*
 * Applies the brake to the PUMA joints.
 *
 * RETURN:  1 if successfully applied, 0 otherwise
 */
uint8_t brake_on();

/*
 * Releases the brake on the PUMA joints.
 *
 * RETURN:  1 if successfully released, 0 otherwise
 *
 */
uint8_t brake_off();

/*
 * Puts PUMA in a safe state by appying the brakes & setting all
 * DAC channels to 0.0 volts. Also clears encoder count registers
 * and powers down the ADC on the Motenc-Lite cards.
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t puma_close();


/************************************************************
 *                     Helper functions                     *
 *   DON'T CALL THESE UNLESS YOU KNOW WHAT YOU ARE DOING    *
 ************************************************************/
/*
 * Finds Motenc-Lite cards that are connected to the computer.
 * It specifically looks for TWO installed Motenc-Lite cards.
 *
 * RETURN:  1 if two Motenc-Lite cards are found, 0 otherwise
 */
uint8_t find_motenc_cards();

/*
 * Clears/resets the DAC by setting all channels to 0.0 volts.
```

```
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t clear_dac();

/*
 * Clears/resets the DAC. PWM circuits are switched off by writing
 * -1.0 volts to their DAC channels. The brake is applied, and
 * unused channels are written 0.0 volts.
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t clear_encoders();

/*
 * Initializes the Motenc-Lite ADC.
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t adc_on();

/*
 * Shuts off the ADC on the Motenc-Lite card.
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t adc_off();

/*
 * Converts a desired DAC voltage to the corresponding value of
 * the DAC transfer function. Transfer function is:
 *  0x0000 => +10 Volts
 *  0x1000 =>   0 Volts
 *  0x1FFF => -10 Volts
 *
 * INPUT:       voltage     Desired DAC output pin voltage
 * RETURN:  Proper value to write to the DAC
 */
uint32_t dac_conversion(float voltage);

#endif
```

Puma.c

```
/*
 * Primary functions to interact with the Motenc-Lite
 * cards and the PUMA controller.
 *
 * puma.h
 *
 * Alex Grieve, 2014
 */

#include "puma.h"
```

```c
// Mmapped memory access to Motenc cards
uint32_t *motenc_lower = NULL;
uint32_t *motenc_upper = NULL;

/*********************************************************
 *                    Public functions                    *
 *********************************************************/
/*
 * Finds and initializes Motenc-Lite cards for use.
 *
 * RETURN:  1 on successfull initialization, 0 otherwise
 */
uint8_t puma_init()
{
    // Mmap the Motenc cards, abort if there is an error
    if (!find_motenc_cards())
        return 0;

    int err = 1;

    // Clear DAC
    if(!clear_dac()) err = 0;;

    // Setup ADC
    if(!adc_on()) err = 0;

    // Clear encoder counters
    if(!clear_encoders()) err = 0;

    return err;
}

/*
 * Retrieves the unsigned encoder count from the Motenc-Lite
 * card for a specific joint.
 *
 * INPUT:       joint       Desired joint (J1 - J6)
 * RETURN:  Unsigned encoder count, 0 if error occurs
 */
uint32_t read_encoder(uint8_t joint)
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0;

    // Encoder registers are at offset 0, 1, 2
    if (joint == J1 || joint == J2 || joint == J3)
        return *(motenc_lower + joint);
    else if (joint == J4 || joint == J5 || joint == J6)
        return *(motenc_upper + joint - 3);
    else
        return 0.0;
}

/*
 * Retrieves the voltage across a specific joint's potentiometer
 * using the Motenc-Lite ADC.
```

```
 *
 * INPUT:       joint       Desired joint (J1 - J6)
 * RETURN:  Voltage across the potentiometer, 0 if error occurs
 */
float read_pot(uint8_t joint)
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0.0;

    uint32_t *mem;
    int16_t adc;
    float volts;
    int adc_joint;

    // Determine which card we need to use
    if (joint == J1 || joint == J2 || joint == J3) {
        mem = motenc_lower;
        adc_joint = joint;
    }
    else if (joint == J4 || joint == J5 || joint == J6) {
        adc_joint = joint - 3;
        mem = motenc_upper;
    }
    else
        return 0.0;

    // Write a 1 to register 40, starts a conversion
    *(mem+40) = 0x1;

    // Wait for conversion to finish
    while(((*(mem + 5)) >> 18) & 0x1) ;

    // Read the correct ADC value
    // ADC values are accessed by reading register 32 sequentially
    // All ADC values must be read (hence the temp variable)
    int i;
    uint16_t temp;
    for (i=0; i<3; ++i) {
        if (i == adc_joint)
            adc = *(mem+32);
        else
            temp = *(mem+32);
    }

    // Convert ADC to voltage
    volts = (float)adc * 10.0 / 16384.0;
    return volts;
}

/*
 * Apply torque to a specific joint. This function uses the
 * following equation for torque:
 *
 *  torque = k * I
 *
 * where k is the motor torque constant and I is the current
 * flowing through the motor.
```

```
 *
 * INPUT:       joint           Desired joint (J1 - J6)
 *                  torque      Torque to apply
 *                  direction   Desired motor direction (0 or 1)
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t apply_torque(int joint, float torque, int direction)
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0;

    uint8_t err = 1;
    float current;
    float voltage;

    switch (joint)
    {
        case J1:
            // Calculate the amount of current needed
            current = torque / J1_K;
            // Check current limitations
            if (current > J1_I_MAX) current = J1_I_MAX;
            else if (current < J1_I_MIN) current = J1_I_MIN;
            // Find the corresponding voltage value
            voltage = DAC_MAX / J1_I_MAX * current;
            // Write the voltage to the DAC
            err = write_dac(joint, voltage, direction);
            break;

        case J2:
            // Calculate the amount of current needed
            current = torque / J2_K;
            // Check current limitations
            if (current > J2_I_MAX) current = J2_I_MAX;
            else if (current < J2_I_MIN) current = J2_I_MIN;
            // Find the corresponding voltage value
            voltage = DAC_MAX / J2_I_MAX * current;
            // Write the voltage to the DAC
            err = write_dac(joint, voltage, direction);
            break;

        case J3:
            // Calculate the amount of current needed
            current = torque / J3_K;
            // Check current limitations
            if (current > J3_I_MAX) current = J3_I_MAX;
            else if (current < J3_I_MIN) current = J3_I_MIN;
            // Find the corresponding voltage value
            voltage = DAC_MAX / J3_I_MAX * current;
            // Write the voltage to the DAC
            err = write_dac(joint, voltage, direction);
            break;

        case J4:
            // Calculate the amount of current needed
            current = torque / J4_K;
            // Check current limitations
```

```
            if (current > J4_I_MAX) current = J4_I_MAX;
            else if (current < J4_I_MIN) current = J4_I_MIN;
            // Find the corresponding voltage value
            voltage = DAC_MAX / J4_I_MAX * current;
            // Write the voltage to the DAC
            err = write_dac(joint, voltage, direction);
            break;

        case J5:
            // Calculate the amount of current needed
            current = torque / J5_K;
            // Check current limitations
            if (current > J5_I_MAX) current = J5_I_MAX;
            else if (current < J5_I_MIN) current = J5_I_MIN;
            // Find the corresponding voltage value
            voltage = DAC_MAX / J5_I_MAX * current;
            // Write the voltage to the DAC
            err = write_dac(joint, voltage, direction);
            break;

        case J6:
            // Calculate the amount of current needed
            current = torque / J6_K;
            // Check current limitations
            if (current > J6_I_MAX) current = J6_I_MAX;
            else if (current < J6_I_MIN) current = J6_I_MIN;
            // Find the corresponding voltage value
            voltage = DAC_MAX / J6_I_MAX * current;
            // Write the voltage to the DAC
            err = write_dac(joint, voltage, direction);
            break;

        default:
            err = 0;
            break;
    }

    return err;
}

/*
 * Writes a voltage value to the Motenc-Lite DAC for a specific
 * joint and motor direction.
 *
 * INPUT:       joint           Desired joint (J1 - J6)
 *                  voltage     Voltage to write
 *                  direction   Desired motor direction (0 or 1)
 * RETURN:  1 on successful write to DAC, 0 otherwise
 */
uint8_t write_dac(int joint, float voltage, int direction)
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0;

    // Make sure value is within the acceptable range
    if (voltage > DAC_MAX)
        voltage = DAC_MAX;
```

```
else if (voltage < DAC_MIN)
    voltage = DAC_MIN;

// Write to the DAC
switch (joint)
{
    case J1:
        if (direction == 0) {
            *(motenc_lower + 24 + 1) = dac_conversion(PWM_OFF);
            *(motenc_lower + 24) = dac_conversion(voltage);
        }
        else {
            *(motenc_lower + 24) = dac_conversion(PWM_OFF);
            *(motenc_lower + 24 + 1) = dac_conversion(voltage);
        }
        break;

    case J2:
        if (direction == 0) {
            *(motenc_lower + 26 + 1) = dac_conversion(PWM_OFF);
            *(motenc_lower + 26) = dac_conversion(voltage);
        }
        else {
            *(motenc_lower + 26) = dac_conversion(PWM_OFF);
            *(motenc_lower + 26 + 1) = dac_conversion(voltage);
        }
        break;

    case J3:
        if (direction == 0) {
            *(motenc_lower + 28 + 1) = dac_conversion(PWM_OFF);
            *(motenc_lower + 28) = dac_conversion(voltage);
        }
        else {
            *(motenc_lower + 28) = dac_conversion(PWM_OFF);
            *(motenc_lower + 28 + 1) = dac_conversion(voltage);
        }
        break;

    case J4:
        if (direction == 0) {
            *(motenc_upper + 24 + 1) = dac_conversion(PWM_OFF);
            *(motenc_upper + 24) = dac_conversion(voltage);
        }
        else {
            *(motenc_upper + 24) = dac_conversion(PWM_OFF);
            *(motenc_upper + 24 + 1) = dac_conversion(voltage);
        }
        break;

    case J5:
        if (direction == 0) {
            *(motenc_upper + 26 + 1) = dac_conversion(PWM_OFF);
            *(motenc_upper + 26) = dac_conversion(voltage);
        }
        else {
            *(motenc_upper + 26) = dac_conversion(PWM_OFF);
```

```
                *(motenc_upper + 26 + 1) = dac_conversion(voltage);
            }
            break;

        case J6:
            if (direction == 0) {
                *(motenc_upper + 28 + 1) = dac_conversion(PWM_OFF);
                *(motenc_upper + 28) = dac_conversion(voltage);
            }
            else {
                *(motenc_upper + 28) = dac_conversion(PWM_OFF);
                *(motenc_upper + 28 + 1) = dac_conversion(voltage);
            }
            break;

        default:
            return 0;
    }

    return 1;
}

/*
 * Applies the brake to the PUMA joints.
 *
 * RETURN:  1 if successfully applied, 0 otherwise
 */
uint8_t brake_on()
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0;

    printf("Applying brake...");

    // Brake control is on DAC6 on card that controls lower joints
    if (motenc_lower != NULL) {
        *(motenc_lower + 24 + 6) = dac_conversion(0.0);
        printf("Done.\n\n");
        return 1;
    }
    else {
        printf("Failed.\n\n");
        return 0;
    }
}

/*
 * Releases the brake on the PUMA joints.
 *
 * RETURN:  1 if successfully released, 0 otherwise
 *
 */
uint8_t brake_off()
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0;
```

```c
    printf("Releasing brake...");

    // Brake control is on DAC6 on card that controls lower joints
    if (motenc_lower != NULL) {
        *(motenc_lower + 24 + 6) = dac_conversion(5.0);
        printf("Done.\n\n");
        return 1;
    }
    else {
        printf("Failed.\n\n");
        return 0;
    }
}

/*
 * Puts PUMA in a safe state by appying the brakes & setting all
 * DAC channels to 0.0 volts. Also clears encoder count registers
 * and powers down the ADC on the Motenc-Lite cards.
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t puma_close()
{
    uint8_t err = 1;

    // Put the brake on
    if(!brake_on()) err = 0;

    // Shut off all PWMs by clearing the DAC
    if(!clear_dac()) err = 0;

    // Clear encoder count registers
    if(!clear_encoders()) err = 0;

    // Power down the ADC
    if(!adc_off()) err = 0;

    return err;
}

/***********************************************************
 *                     Private functions                   *
 *   DON'T CALL THESE UNLESS YOU KNOW WHAT YOU ARE DOING   *
 ***********************************************************/
/*
 * Finds Motenc-Lite cards that are connected to the computer.
 * It specifically looks for TWO installed Motenc-Lite cards.
 *
 * RETURN:  1 if two Motenc-Lite cards are found, 0 otherwise
 */
uint8_t find_motenc_cards()
{
    printf("Searching for MOTENC-Lite cards...\n");

    // Check that we have access to /dev/mem
    if (iopl(3))
    {
```

```
        fprintf(stderr, "Cannot access /dev/mem. Try running as root.");
        return 0;
    }

    // File descriptor for /dev/mem
    int fd = open("/dev/mem", O_RDWR);
    if (fd == 0)
    {
        fprintf(stderr, "Could not open /dev/mem.");
        return 0;
    }

    // Struct to access PCI devices
    struct pci_access *pci_acc;
    // Struct for an individual PCI device
    struct pci_dev *pci_dev = NULL;
    // PCI region addresses
    uint32_t bar_lower[3];
    uint32_t bar_upper[3];
    // Base and offset for mmap
    uint32_t base_lower, offset_lower;
    uint32_t base_upper, offset_upper;


    // Initialize pci_access struct
    pci_acc = pci_alloc();
    // Initialize PCI library
    pci_init(pci_acc);
    // Build a list of devices on the PCI bus
    pci_scan_bus(pci_acc);

    // Search for Motenc-Lite cards
    pci_dev = pci_acc->devices;
    while (pci_dev != NULL)
    {
        if ( (pci_dev->vendor_id == VENDOR) && (pci_dev->device_id == DEVICE)
) {
            // Found a Motenc card
            if (motenc_lower == NULL)
            {
                // Do some PCI addressing magic
                bar_lower[0] = pci_dev->base_addr[0] &
PCI_BASE_ADDRESS_MEM_MASK;
                bar_lower[1] = pci_dev->base_addr[1] &
PCI_BASE_ADDRESS_IO_MASK;
                bar_lower[2] = pci_dev->base_addr[2] &
PCI_BASE_ADDRESS_MEM_MASK;
                base_lower = bar_lower[2] & 0xfffff000;
                offset_lower = bar_lower[2] & 0xfff;

                motenc_lower = mmap(0, PCI_MEM_LEN + offset_lower,
PROT_READ|PROT_WRITE, MAP_SHARED, fd, base_lower);
                if (motenc_lower == MAP_FAILED)
                {
                    fprintf(stderr, "mmap() failed.");
                    return 0;
                }
```

```
                motenc_lower = motenc_lower + (offset_lower/4);
                // Motenc card has been mapped
            }
            else if (motenc_upper == NULL)
            {
                // Do some PCI addressing magic
                bar_upper[0] = pci_dev->base_addr[0] &
PCI_BASE_ADDRESS_MEM_MASK;
                bar_upper[1] = pci_dev->base_addr[1] &
PCI_BASE_ADDRESS_IO_MASK;
                bar_upper[2] = pci_dev->base_addr[2] &
PCI_BASE_ADDRESS_MEM_MASK;
                base_upper = bar_upper[2] & 0xfffff000;
                offset_upper = bar_upper[2] & 0xfff;

                motenc_upper = mmap(0, PCI_MEM_LEN + offset_upper,
PROT_READ|PROT_WRITE, MAP_SHARED, fd, base_upper);
                if (motenc_upper == MAP_FAILED)
                {
                    fprintf(stderr, "mmap() failed.");
                    return 0;
                }
                motenc_upper = motenc_upper + (offset_upper/4);
                // Motenc card has been mapped
            }
            else
            {
                fprintf(stderr, "Found more than 2 Motenc-Lite cards,
aborting.");
                return 0;
            }
        } // end Motenc card assignment

        // Move onto the next PCI device
        pci_dev = pci_dev->next;

    } // end while

    // Make sure we found the 2 Motenc Cards
    if ( (motenc_lower == NULL) || (motenc_upper == NULL) )
    {
        fprintf(stderr, "Did not find 2 Motenc-Lite cards, aborting.");
        return 0;
    }

    // Make sure each pointer is referencing the right Motenc-Lite card
    // Hardware Board ID is at register 5, bits 16 and 17
    uint8_t lower_id = ((*(motenc_lower+5)) >> 16) & 0x3;
    uint8_t upper_id = ((*(motenc_upper+5)) >> 16) & 0x3;
    if ( lower_id != LOWER_BOARD_ID || upper_id != UPPER_BOARD_ID )
    {
        // Swap the pointers to the Motenc cards
        uint32_t *tmp = motenc_lower;
        motenc_lower = motenc_upper;
        motenc_upper = tmp;
    }
```

```
    printf("-->Found MOTENC-Lite w/ ID: %d\n", lower_id);
    printf("-->Found MOTENC-Lite w/ ID: %d\n", upper_id);
    printf("Done.\n\n");

    return 1;
}

/*
 * Clears/resets the DAC. PWM circuits are switched off by writing
 * -1.0 volts to their DAC channels. The brake is applied, and
 * unused channels are written 0.0 volts.
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t clear_dac()
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0;

    printf("Clearing DAC...");

    // Put the brake on for safety
    *(motenc_lower + 24 + 6) = dac_conversion(0.0); // DAC6 - Brake

    // Reset/turn off the lower joints
    *(motenc_lower+24) = dac_conversion(PWM_OFF);        // DAC0 - J1 PWM+
    *(motenc_lower+24+1) = dac_conversion(PWM_OFF); // DAC1 - J1 PWM-
    *(motenc_lower+24+2) = dac_conversion(PWM_OFF); // DAC2 - J2 PWM+
    *(motenc_lower+24+3) = dac_conversion(PWM_OFF); // DAC3 - J2 PWM-
    *(motenc_lower+24+4) = dac_conversion(PWM_OFF); // DAC4 - J3 PWM+
    *(motenc_lower+24+5) = dac_conversion(PWM_OFF); // DAC5 - J3 PWM-
    *(motenc_lower+24+7) = dac_conversion(0.0);     // DAC7 - Not used

    // Reset/turn off the upper joints
    *(motenc_upper+24) = dac_conversion(PWM_OFF);        // DAC0 - J4 PWM+
    *(motenc_upper+24+1) = dac_conversion(PWM_OFF); // DAC1 - J4 PWM-
    *(motenc_upper+24+2) = dac_conversion(PWM_OFF); // DAC2 - J5 PWM+
    *(motenc_upper+24+3) = dac_conversion(PWM_OFF); // DAC3 - J5 PWM-
    *(motenc_upper+24+4) = dac_conversion(PWM_OFF); // DAC4 - J6 PWM+
    *(motenc_upper+24+5) = dac_conversion(PWM_OFF); // DAC5 - J6 PWM-
    *(motenc_upper+24+6) = dac_conversion(0.0);     // DAC6 - Not used
    *(motenc_upper+24+7) = dac_conversion(0.0);     // DAC7 - Not used

    printf("Done.\n\n");

    return 1;
}

/*
 * Clears all encoder count registers on the Motenc-Lite card.
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t clear_encoders()
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0;
```

```
    printf("Clearing encoders...");

    *(motenc_lower+5) = 0xf;
    *(motenc_upper+5) = 0xf;

    printf("Done.\n\n");

    return 1;
}

/*
 * Initializes the Motenc-Lite ADC.
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t adc_on()
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0;

    printf("Initializing ADC...");

    // Set ADC to sample channels 0,1,2
    *(motenc_lower+32) = 2;
    *(motenc_upper+32) = 2;
    // Clear the ADC_DONE bit
    int temp;
    temp = ((*(motenc_lower + 5)) >> 18) & 0x1;
    temp = ((*(motenc_upper + 5)) >> 18) & 0x1;

    printf("Done.\n\n");
}

/*
 * Shuts off the ADC on the Motenc-Lite card.
 *
 * RETURN:  1 on success, 0 otherwise
 */
uint8_t adc_off()
{
    if (motenc_lower == NULL || motenc_upper == NULL)
        return 0;

    printf("Powering down ADC...");

    *(motenc_lower+32) = 8;
    *(motenc_upper+32) = 8;

    printf("Done.\n\n");

    return 1;
}

/*
 * Converts a desired DAC voltage to the corresponding value of
 * the DAC transfer function. Transfer function is:
```

```
 *  0x0000 => +10 Volts
 *  0x1000 =>   0 Volts
 *  0x1FFF => -10 Volts
 *
 * INPUT:       voltage     Desired DAC output pin voltage
 * RETURN:  Proper value to write to the DAC
 */
uint32_t dac_conversion(float voltage)
{
    long dac_value = (int)((10.0 - voltage) / 20.0 * 0x1FFF);
    return (uint32_t)dac_value;
}
```

Skel.c

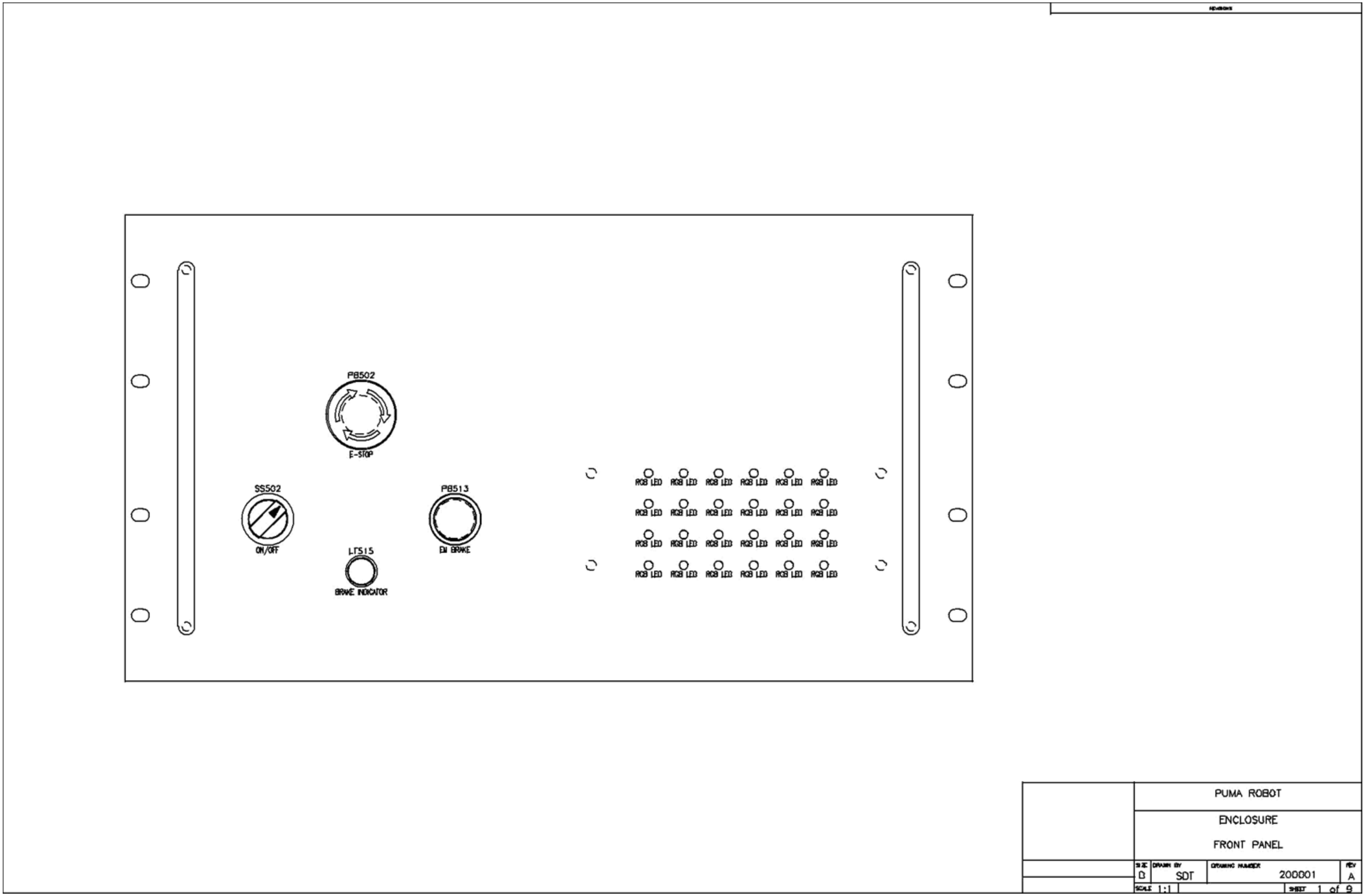
```
/*
 * Skeleton file for PUMA programs. User should always
 * use this template as a starting point.
 *
 * skel.c
 *
 * Alex Grieve, 2014
 */

#include "puma.h"
#include <signal.h>

void int_handler(int signal);

int main(void)
{
    // DON'T REMOVE THESE LINES - THIS ENSURES THE BRAKE
    // IS TURNED ON IF THE PROGRAM STOPS PREMATURELY
    signal(SIGQUIT, int_handler);
    signal(SIGINT, int_handler);

    // Initialize the PUMA and Motenc-Lite cards
    if(!puma_init()) {
        printf("PUMA initialization failed.\n");
        return 1;
    }



    /*****************************/
    /*     Program goes here     */
    /*****************************/




    // Shut down the PUMA and Motenc-Lite cards
    if(!puma_close()) {
```

```
        printf("WARNING: Could not properly shut down the PUMA - it may
remain in an unsafe state.\n");
        return 1;
    }

    return 0;
}

// DON"T REMOVE THIS FUNCTION
void int_handler(int signal)
{
    brake_on();
    puma_close();
    exit(0);
}
```

Appendix VI: Enclosure Drawings

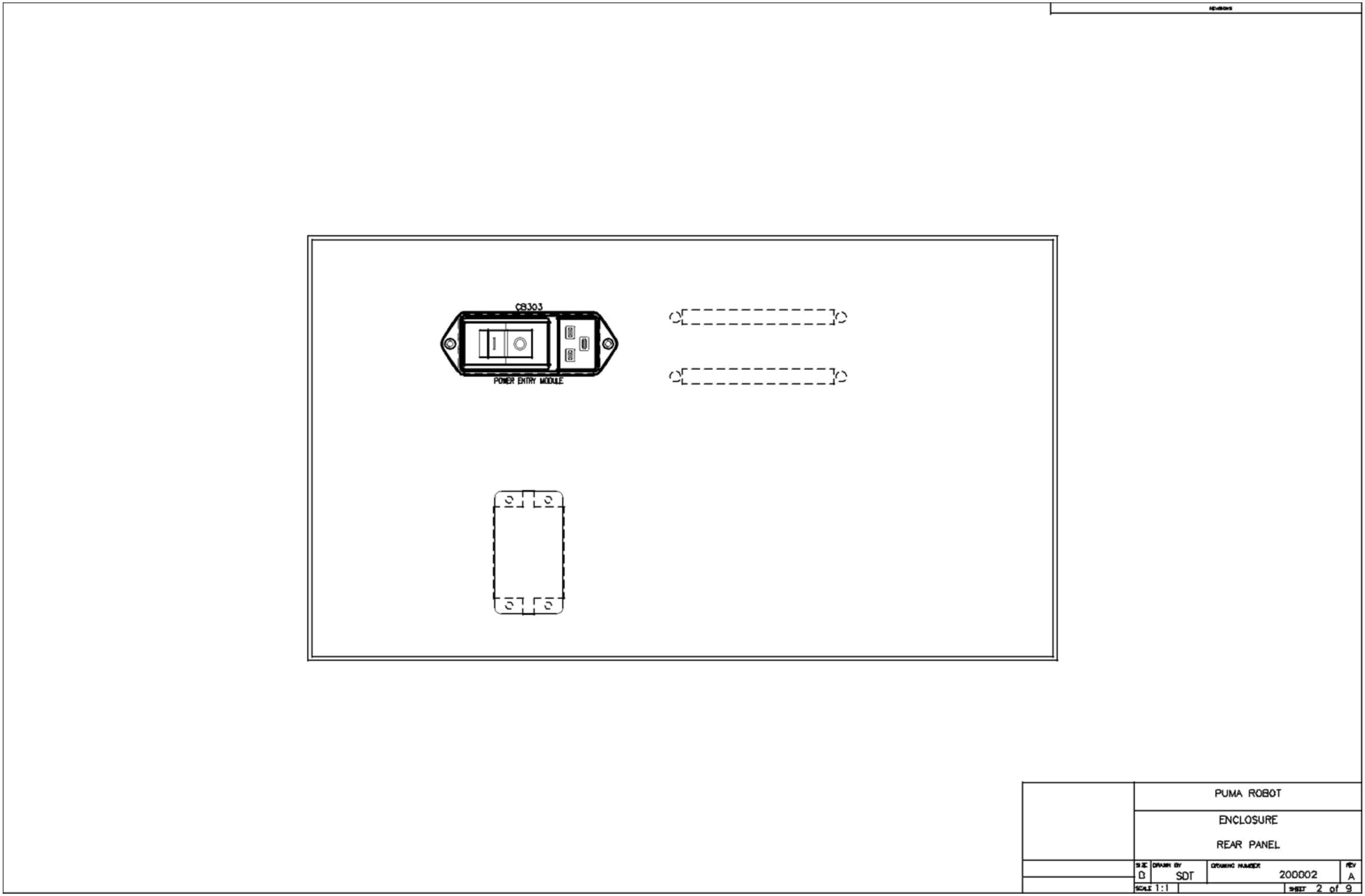
This page intentionally left blank.

Front Panel

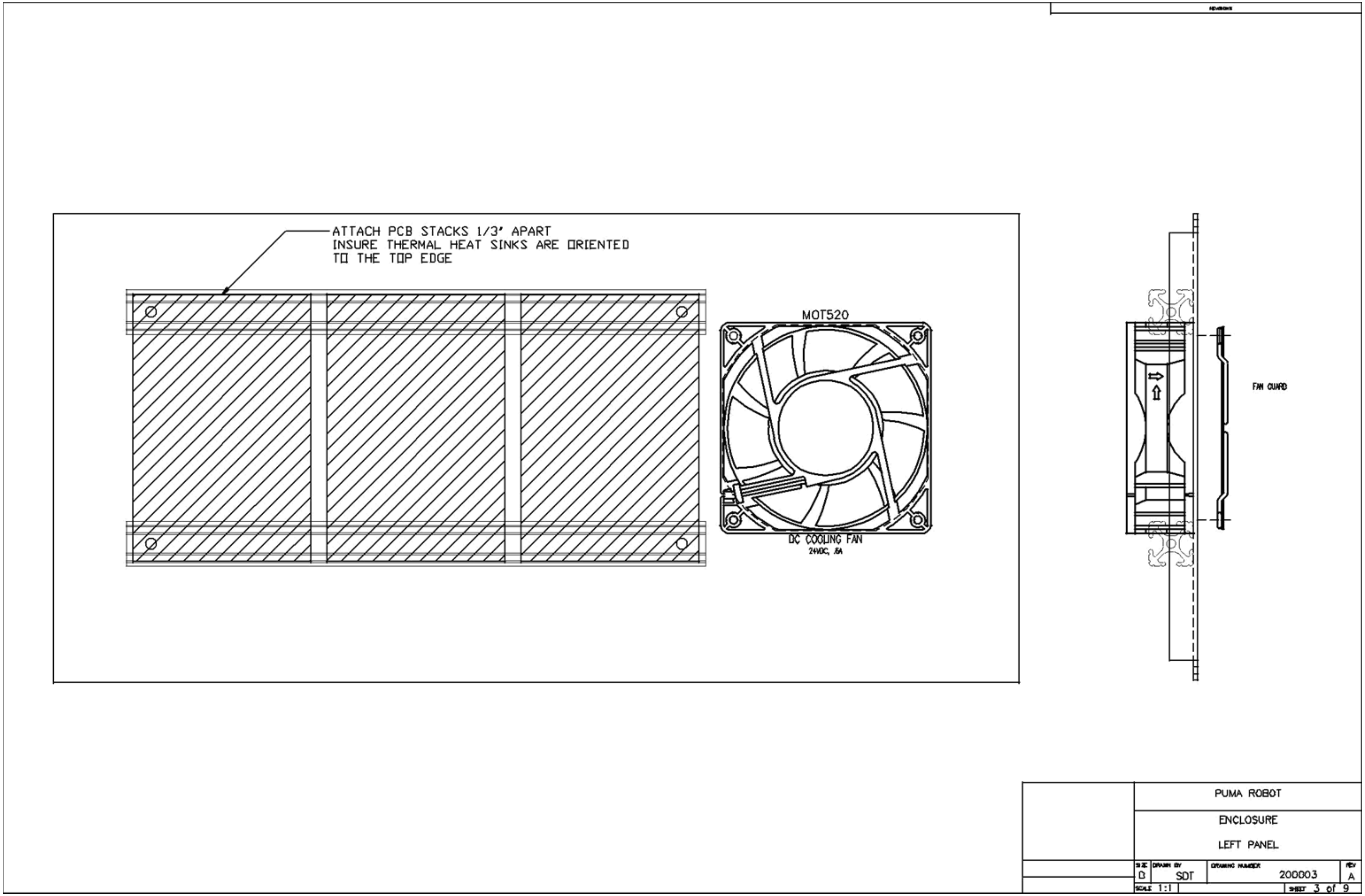


REVISIONS
PB502
E-STOP
SS502
ON/OFF
PB513
EM BRAKE
LT515
BRAKE INDICATOR
RGB LED
PUMA ROBOT
ENCLOSURE
FRONT PANEL
SIZE
B
DRAWN BY
SDT
DRAWING NUMBER
200001
REV
A
SCALE 1:1
SHEET 1 of 9

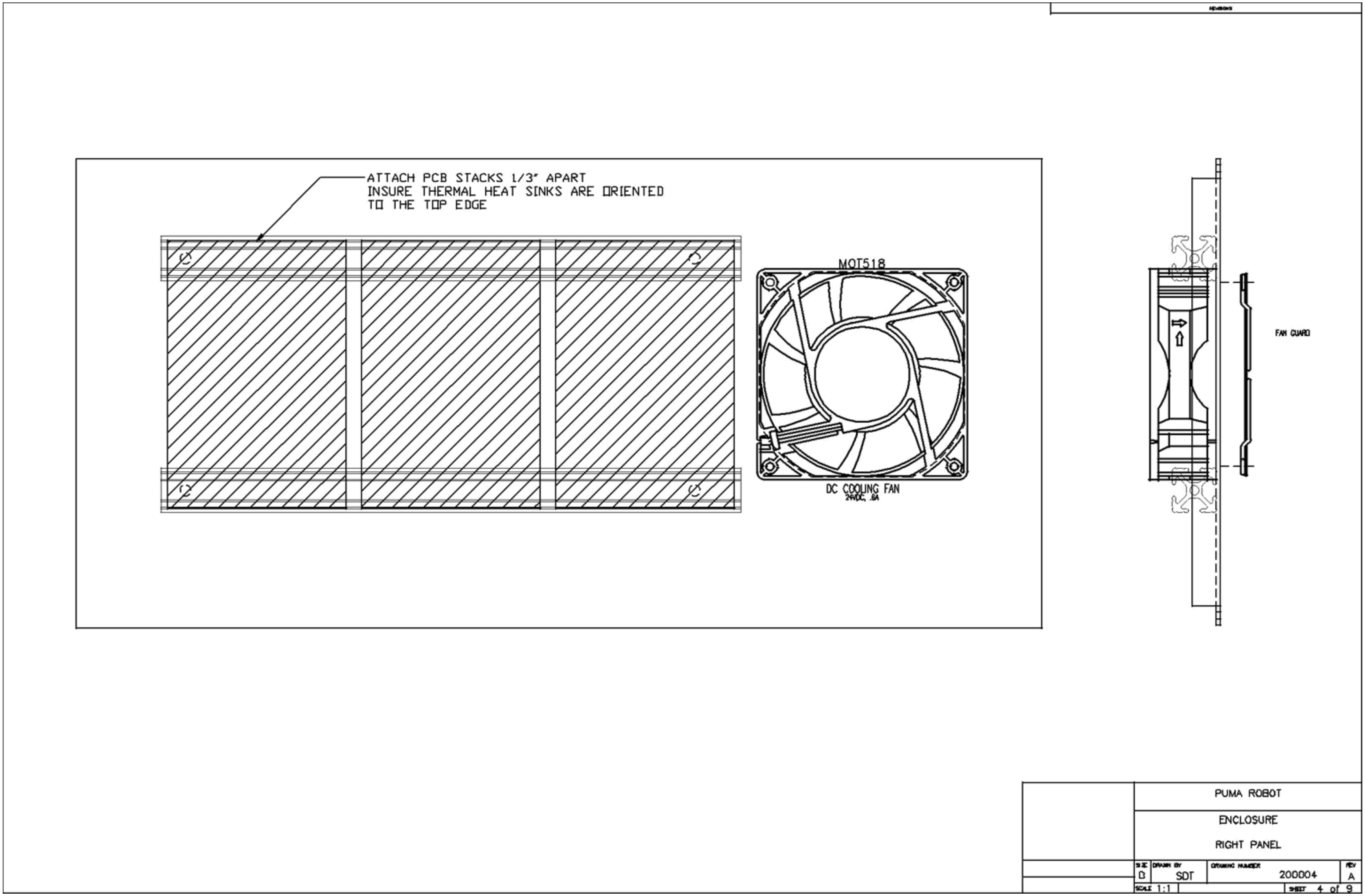
Rear Panel


CB303
POWER ENTRY MODULE
PUMA ROBOT
ENCLOSURE
REAR PANEL
B
SDT
200002
A
1:1
2 of 9

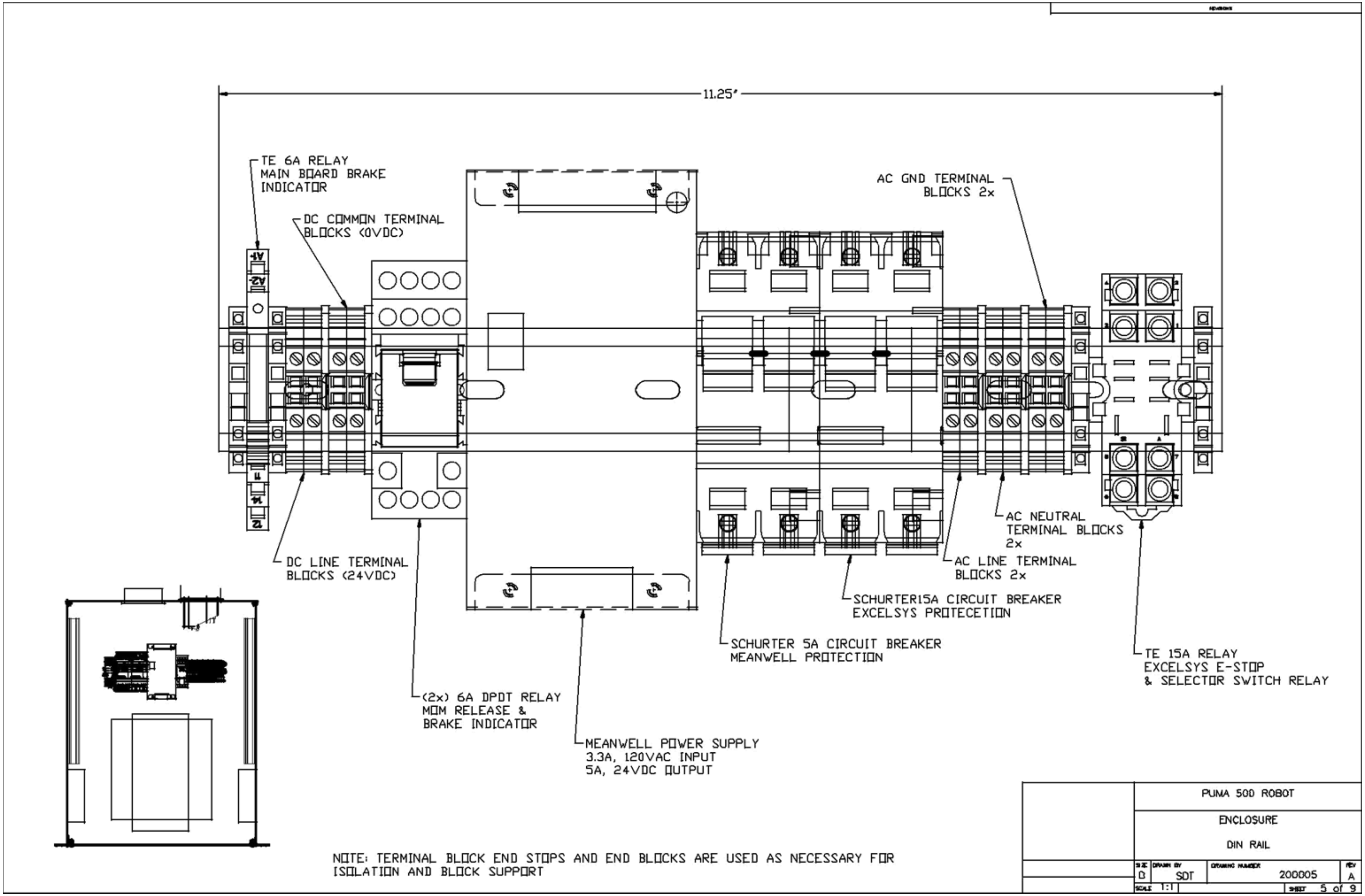
Left Panel


ATTACH PCB STACKS 1/3' APART
INSURE THERMAL HEAT SINKS ARE ORIENTED
TO THE TOP EDGE
MOT520
DC COOLING FAN
24VDC, .6A
FAN GUARD
PUMA ROBOT
ENCLOSURE
LEFT PANEL
SIZE B
DRAWN BY SDT
DRAWING NUMBER 200003
REV A
SCALE 1:1
SHEET 3 of 9

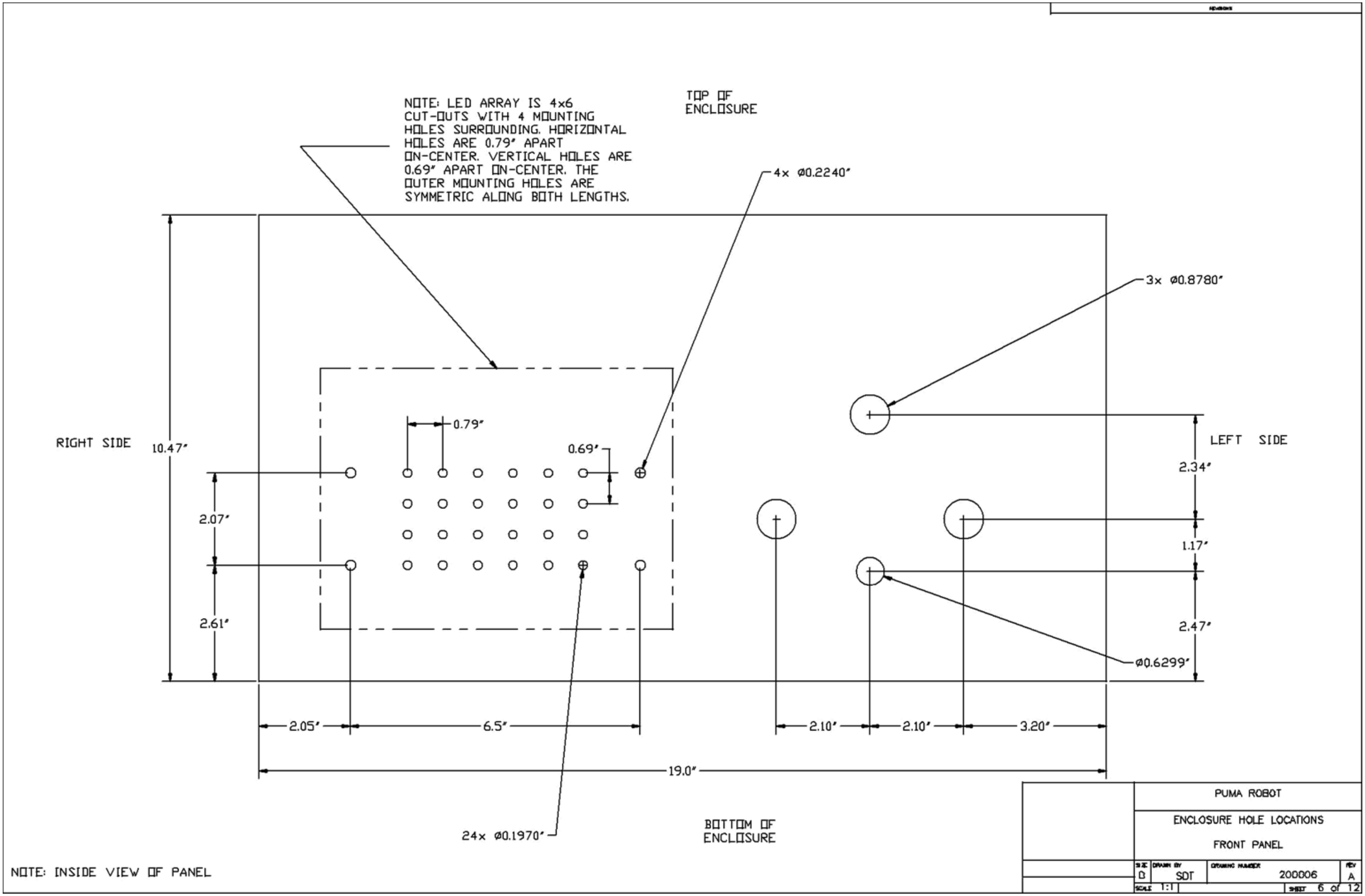
Right Panel


ATTACH PCB STACKS 1/3" APART
INSURE THERMAL HEAT SINKS ARE ORIENTED
TO THE TOP EDGE
MOT518
DC COOLING FAN
24VDC, .6A
FAN GUARD
PUMA ROBOT
ENCLOSURE
RIGHT PANEL
B
SDT
200004
A
1:1
4 of 9

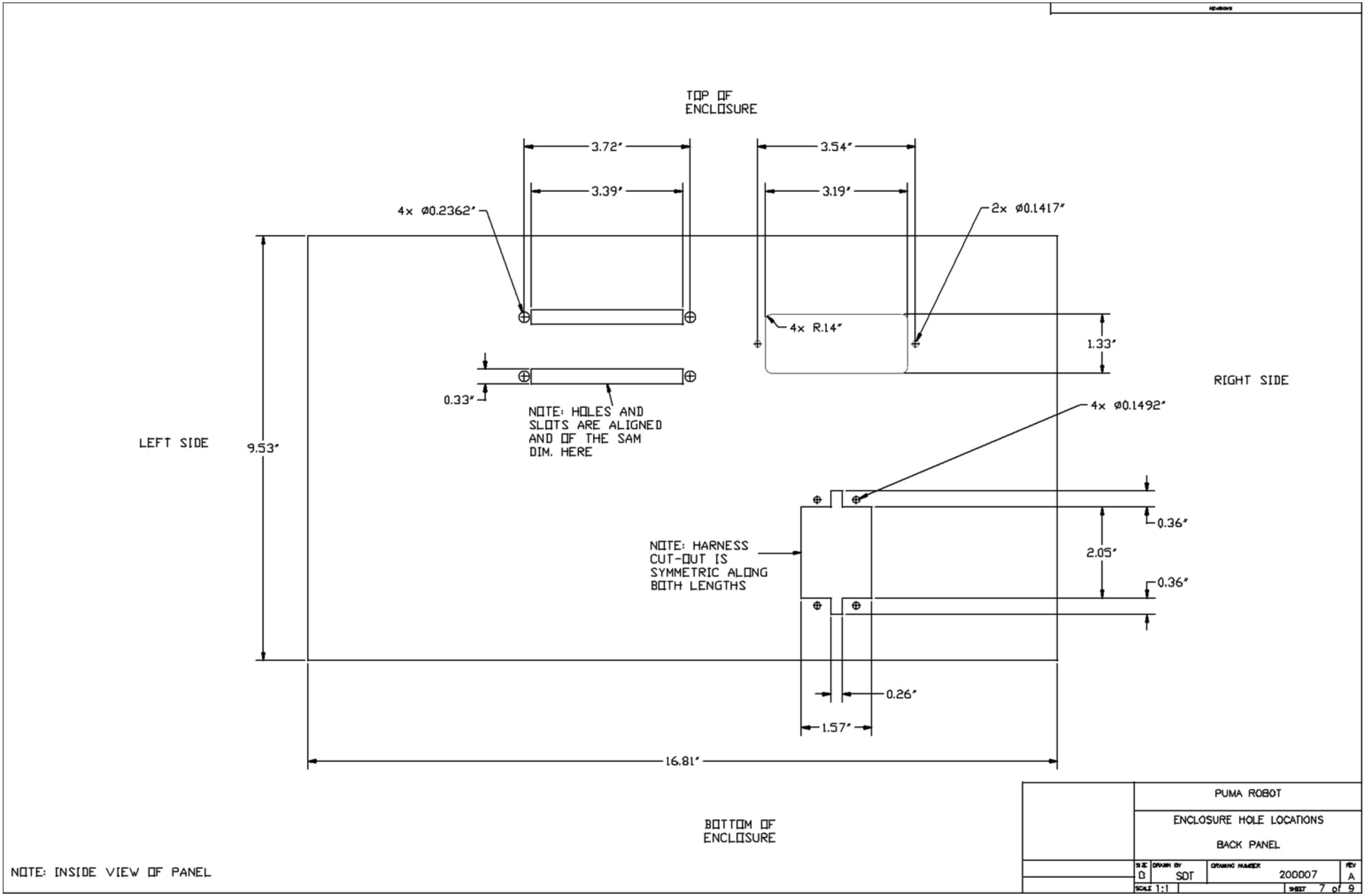
DIN Rail View



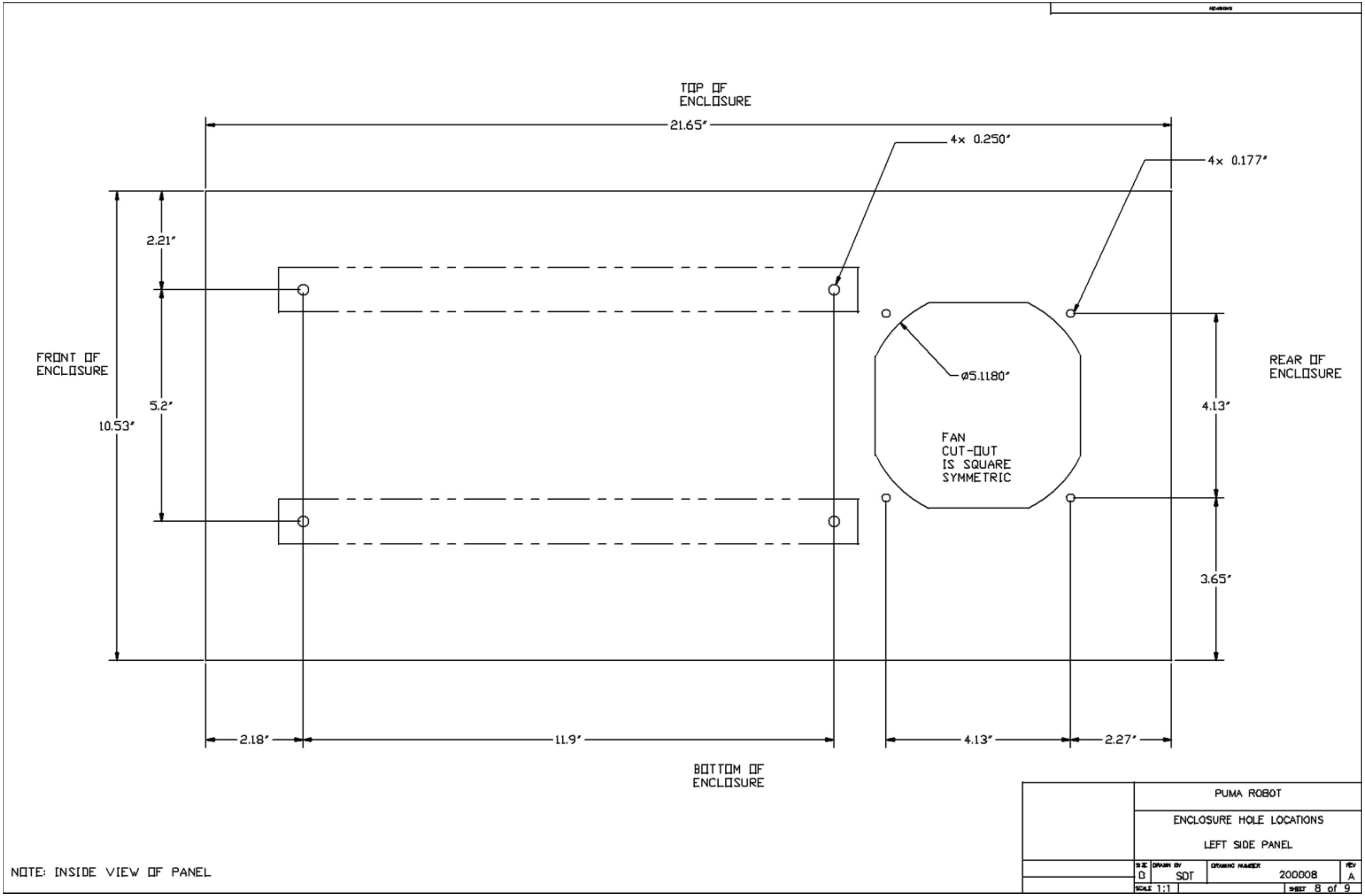
Front Panel Hole Locations



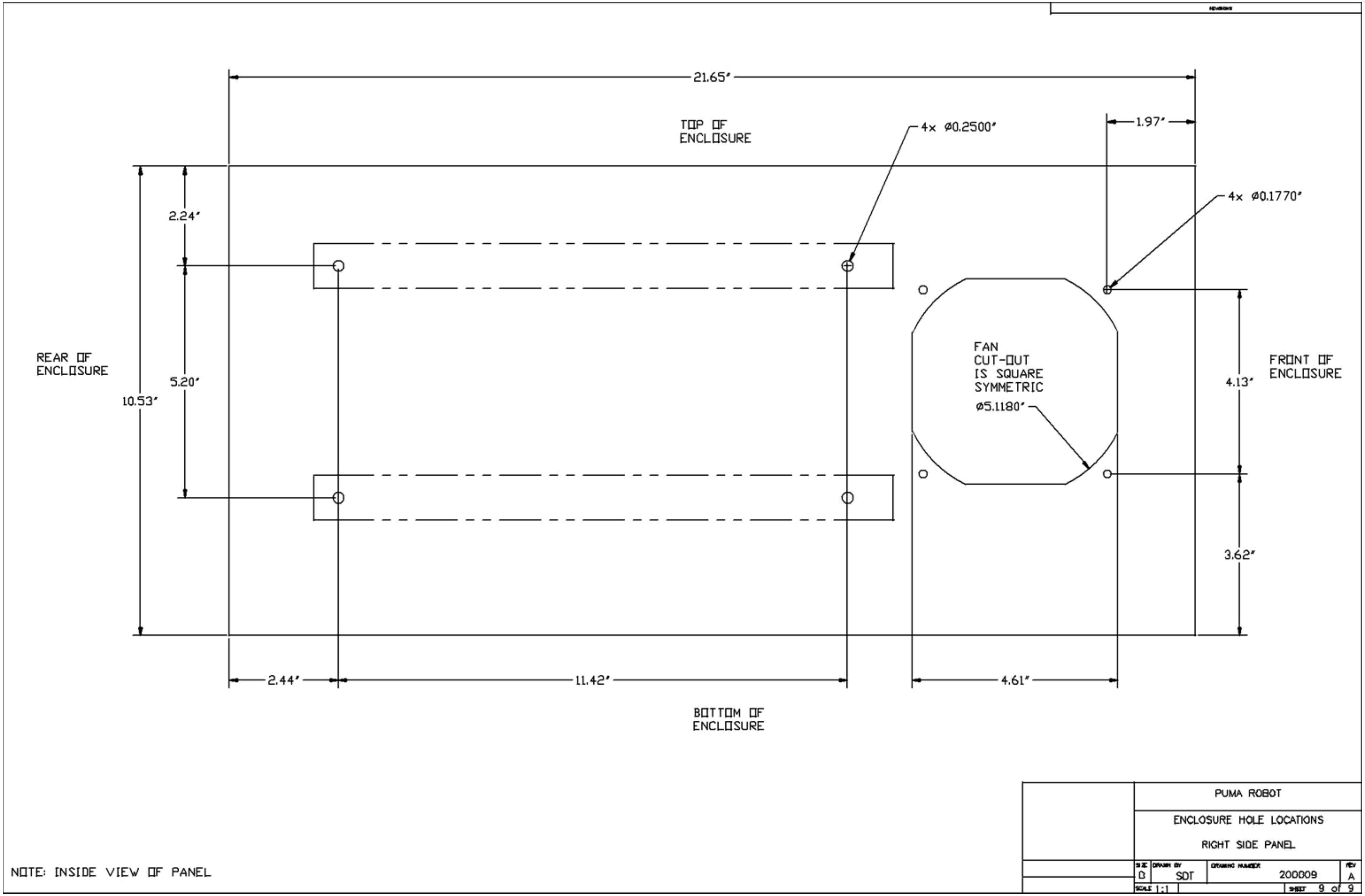
Back Panel Hole Locations



Left Panel Hole Locations



Right Panel Hole Locations



Appendix VII: Electrical Schematics

This page intentionally left blank.

Cover Sheet

WIRE COLOR FUNCTIONS:
12AWG GREEN: AC PE/GND
12AWGWHITE: AC COMMON
12AWGBLACK: AC LINE

14AWG GREEN: AC PE/GND
14AWG WHITE: AC COMMON
14AWG BLACK: AC LINE

16AWG RED: J1-J3 DC LINE (HIGHER DEMAG CURRENT)
16AWG BLACK: J1-J3 DC COMMON

18AWG PURPLE: 24VDC RELAYS/BUTTONS/SWITCHES/LOGIC

20AWG RED: J4-J6 DC LINE
20AWG BLACK: J4-J6 DC COMMON

22AWG BLUE: POTENTIOMETERS/ OTHER LOW CURRENT 24VDC/5VDC LOGIC
22AWG WHITE: ENCODER A
22AWG YELLOW: ENCODER Q
22AWG ORANGE: ENCODER I
22AWG RED: ENCODER 5VDC
22AWG BLACK: ENCODER GND

PUMA ROBOT

COVER SHEET

SIZE	DRAWN BY	DRAWING NUMBER	REV
B	SDT	100001	A

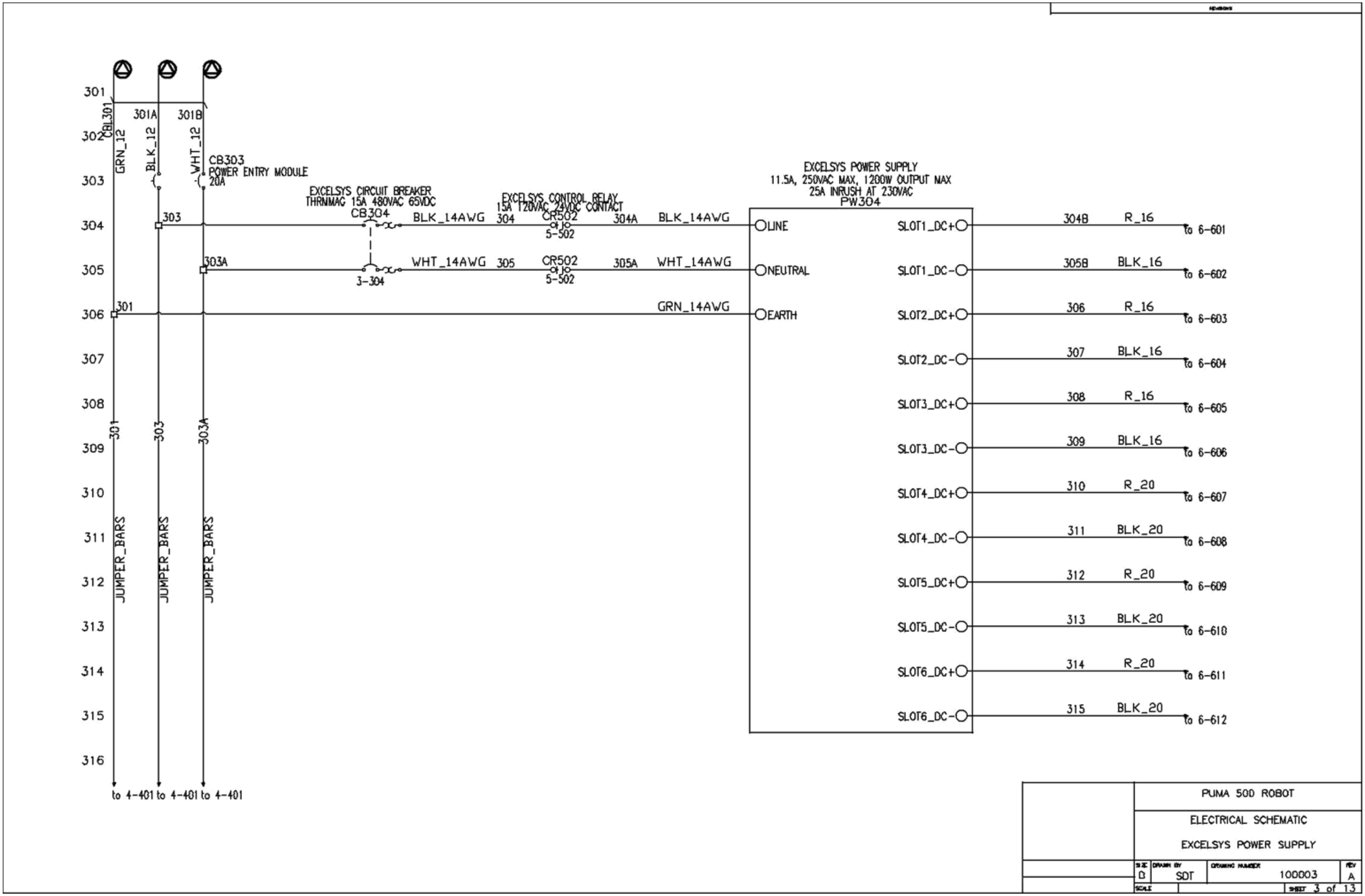
SCALE | SHEET 1 of 13

BOM

NOTE: SEE MASTER BILL OF MATERIALS FOR ALL COMPONENTS UTILIZED IN THIS DRAWING SET

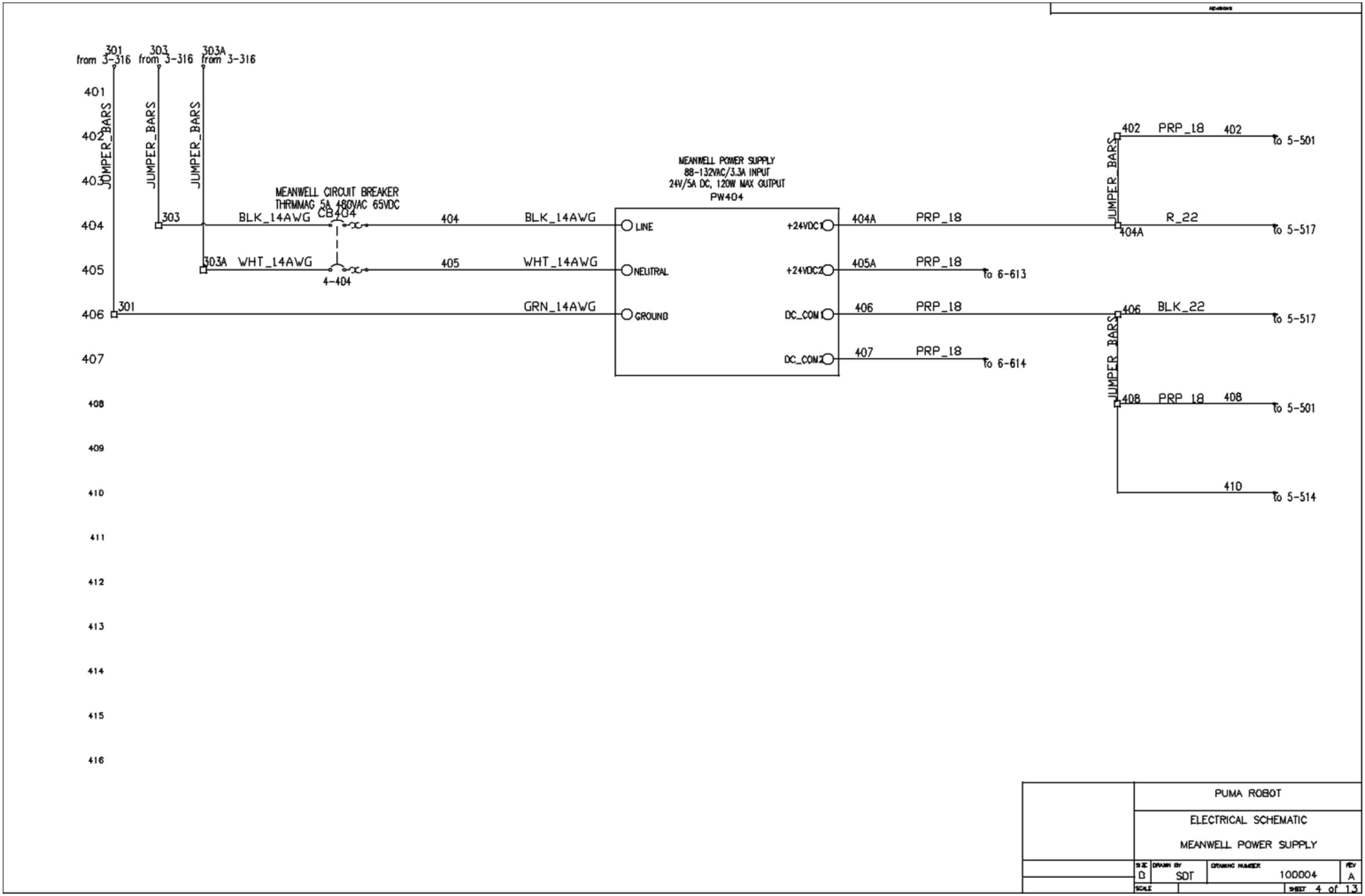
PUMA ROBOT			
BILL OF MATERIALS			
SIZE B	DRAWN BY SDT	DRAWING NUMBER 100002	REV A
SCALE			SHEET 2 of 13

Excelsys

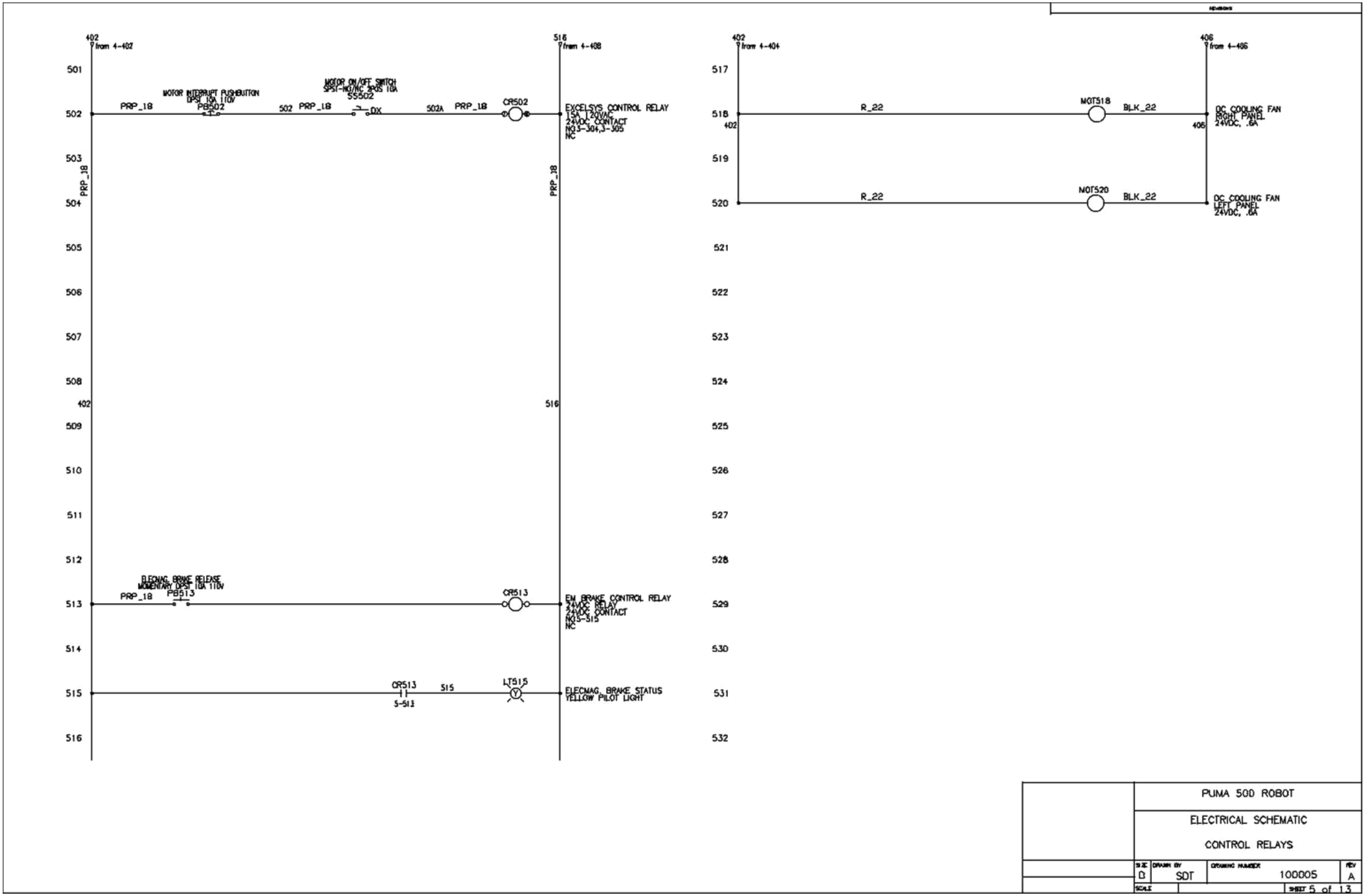


301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
CBL301
301A
301B
GRN_12
BLK_12
WHT_12
CB303
POWER ENTRY MODULE
20A
303
303A
301
EXCELSYS CIRCUIT BREAKER
THRMMAG 15A 480VAC 65VDC
CB304
3-304
BLK_14AWG
WHT_14AWG
GRN_14AWG
EXCELSYS CONTROL RELAY
15A 120VAC 24VDC CONTACT
CR502
5-502
304
304A
305
305A
EXCELSYS POWER SUPPLY
11.5A, 250VAC MAX, 1200W OUTPUT MAX
25A INRUSH AT 230VAC
PW304
LINE
NEUTRAL
EARTH
SLOT1_DC+
SLOT1_DC-
SLOT2_DC+
SLOT2_DC-
SLOT3_DC+
SLOT3_DC-
SLOT4_DC+
SLOT4_DC-
SLOT5_DC+
SLOT5_DC-
SLOT6_DC+
SLOT6_DC-
304B R_16 To 6-601
305B BLK_16 To 6-602
306 R_16 To 6-603
307 BLK_16 To 6-604
308 R_16 To 6-605
309 BLK_16 To 6-606
310 R_20 To 6-607
311 BLK_20 To 6-608
312 R_20 To 6-609
313 BLK_20 To 6-610
314 R_20 To 6-611
315 BLK_20 To 6-612
JUMPER_BARS
JUMPER_BARS
JUMPER_BARS
to 4-401 to 4-401 to 4-401
PUMA 500 ROBOT
ELECTRICAL SCHEMATIC
EXCELSYS POWER SUPPLY
B
SDT
100003
A
SHEET 3 of 13

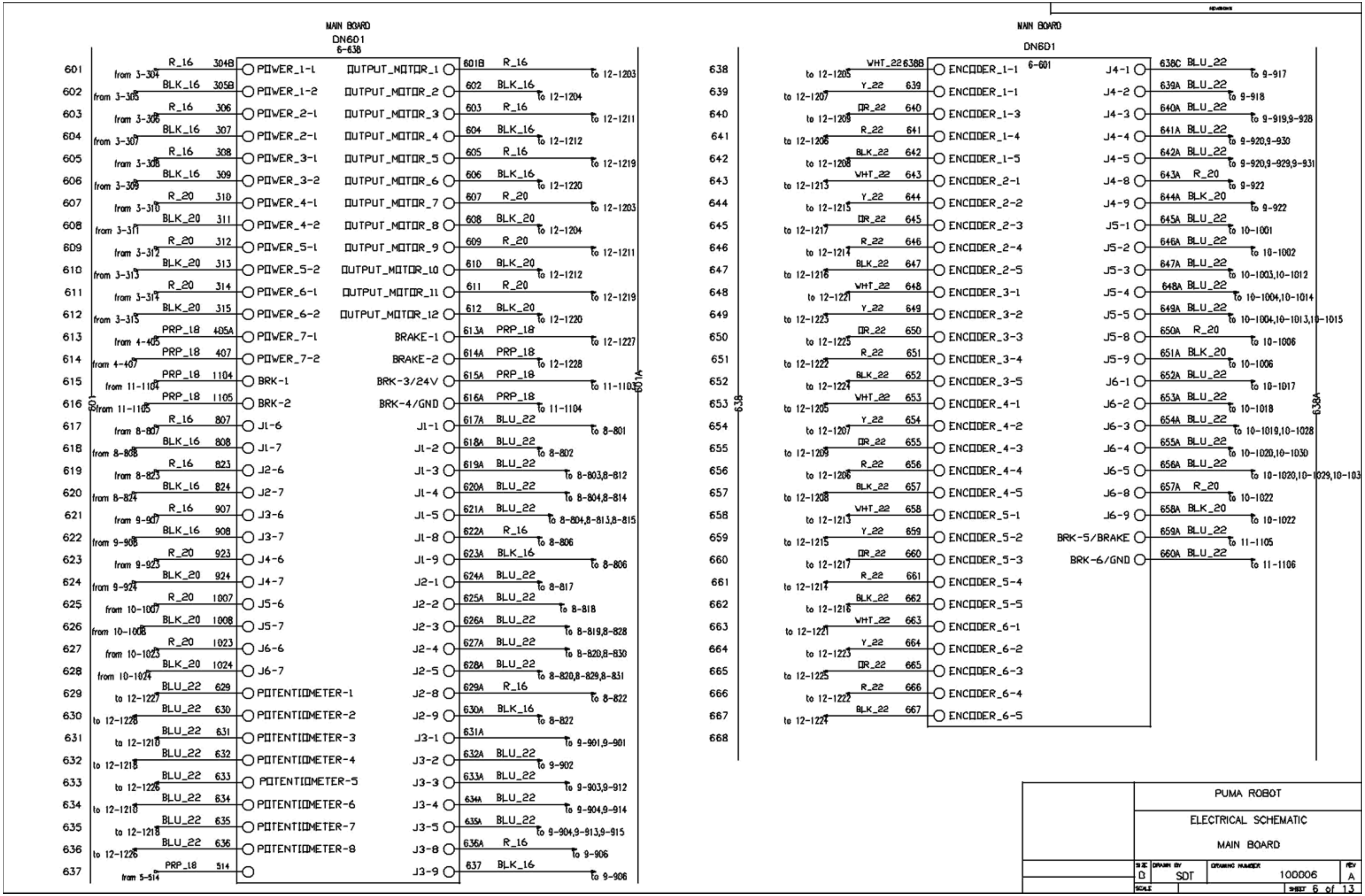
Meanwell


301 from 3-316
303 from 3-316
303A from 3-316
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
JUMPER_BARS
MEANWELL CIRCUIT BREAKER
THRMMAG 5A 480VAC 65VDC
CB404
4-404
BLK_14AWG
WHT_14AWG
GRN_14AWG
MEANWELL POWER SUPPLY
88-132VAC/3.3A INPUT
24V/5A DC, 120W MAX OUTPUT
PW404
LINE
NEUTRAL
GROUND
+24VDC1
+24VDC2
DC_COM1
DC_COM2
404A PRP_18
405A PRP_18 To 6-613
406 PRP_18
407 PRP_18 To 6-614
402 PRP_18 402 To 5-501
R_22 To 5-517
406 BLK_22 To 5-517
408 PRP 18 408 To 5-501
410 To 5-514
PUMA ROBOT
ELECTRICAL SCHEMATIC
MEANWELL POWER SUPPLY
B
SDT
100004
A
4 of 13

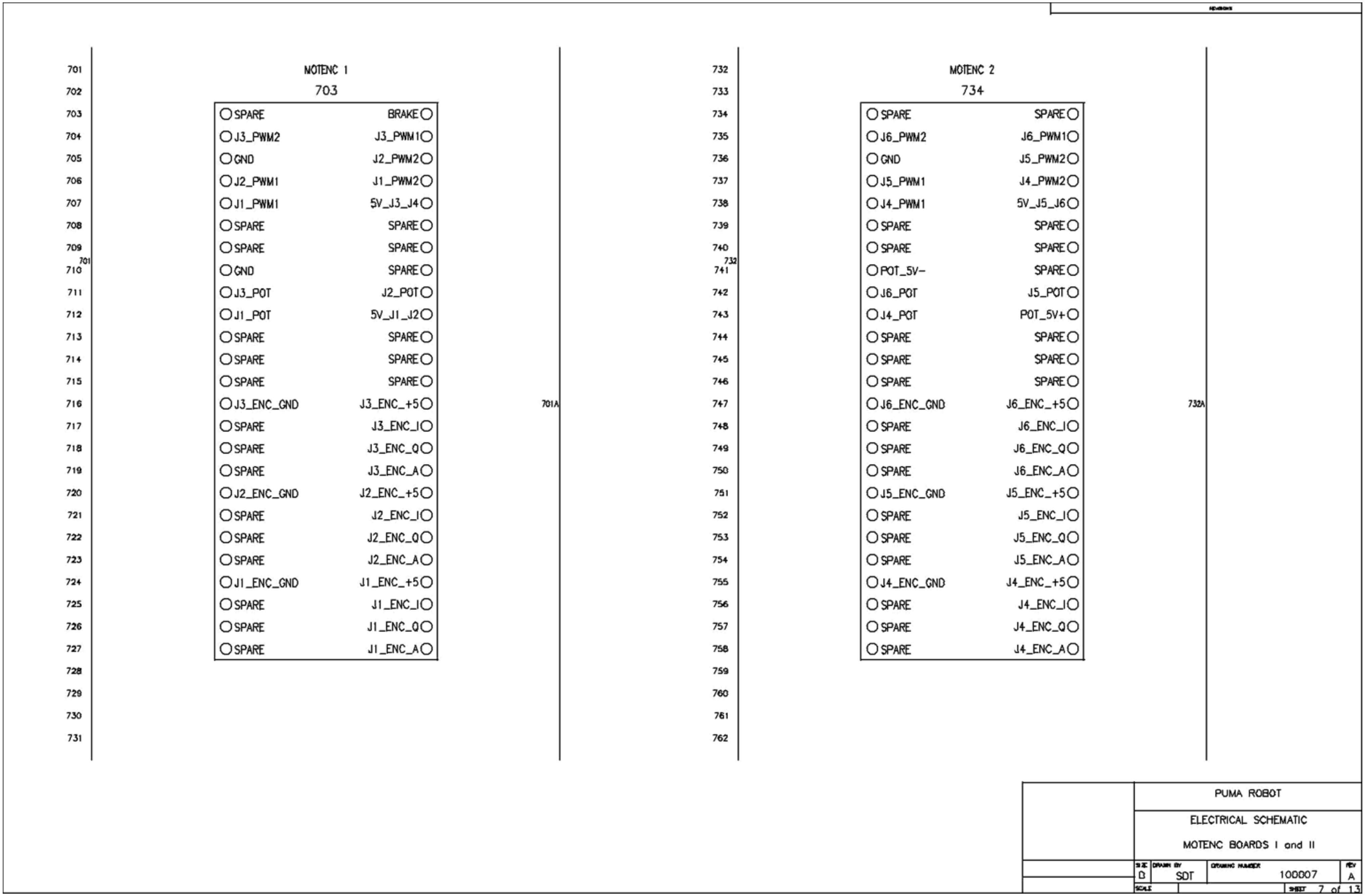
Control Relays



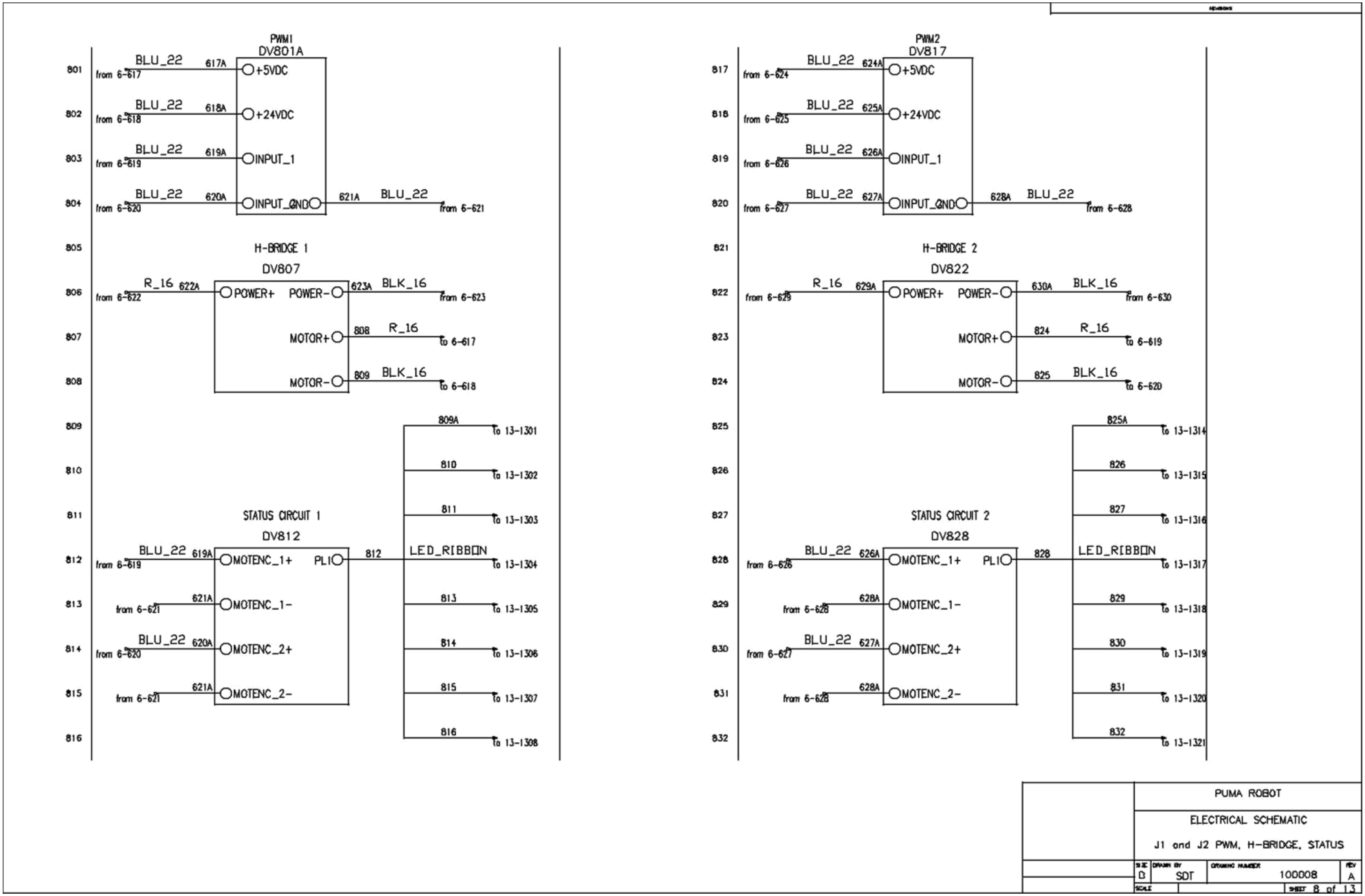
Main Board



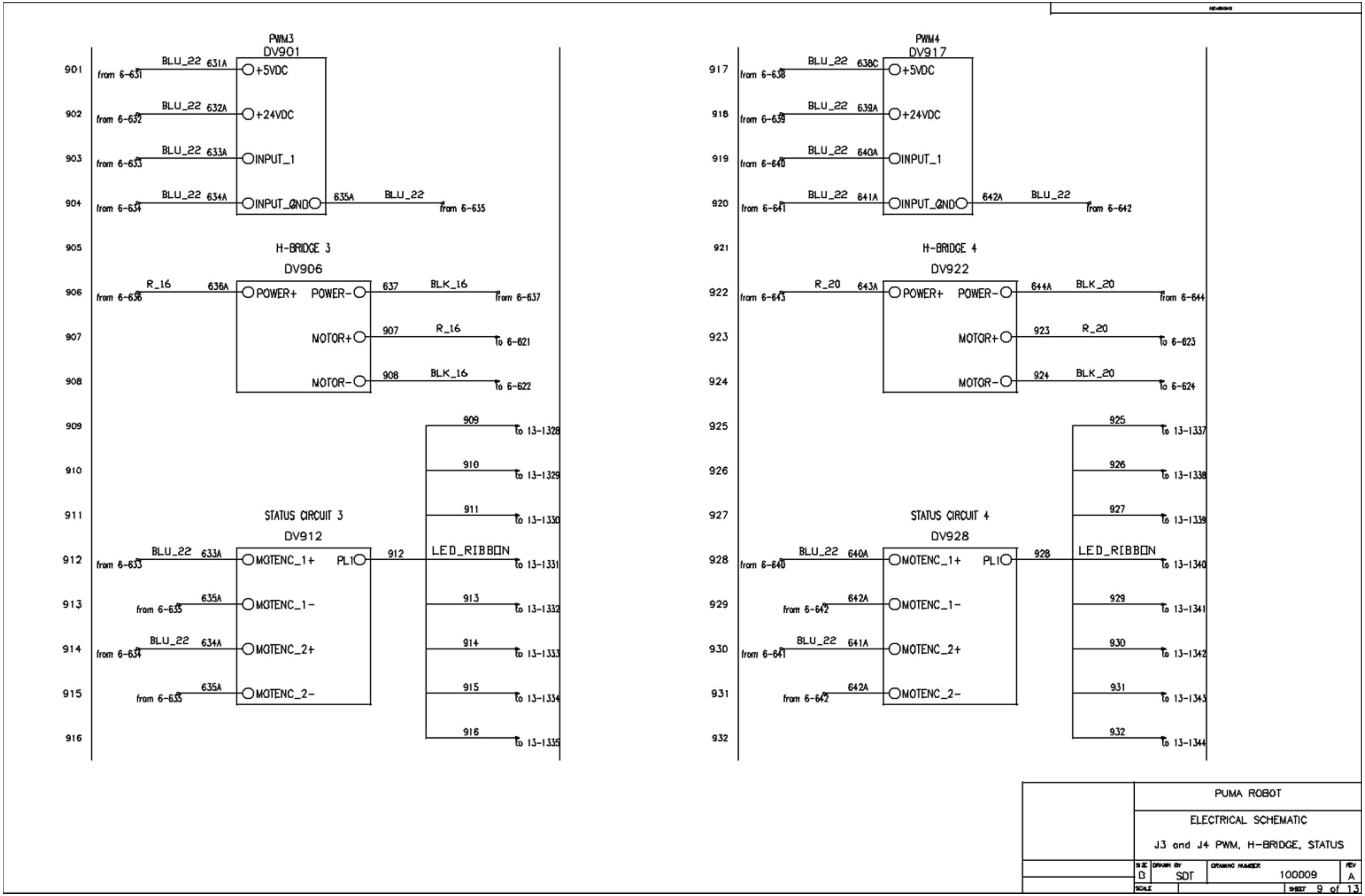
MOTENC 1 and 2



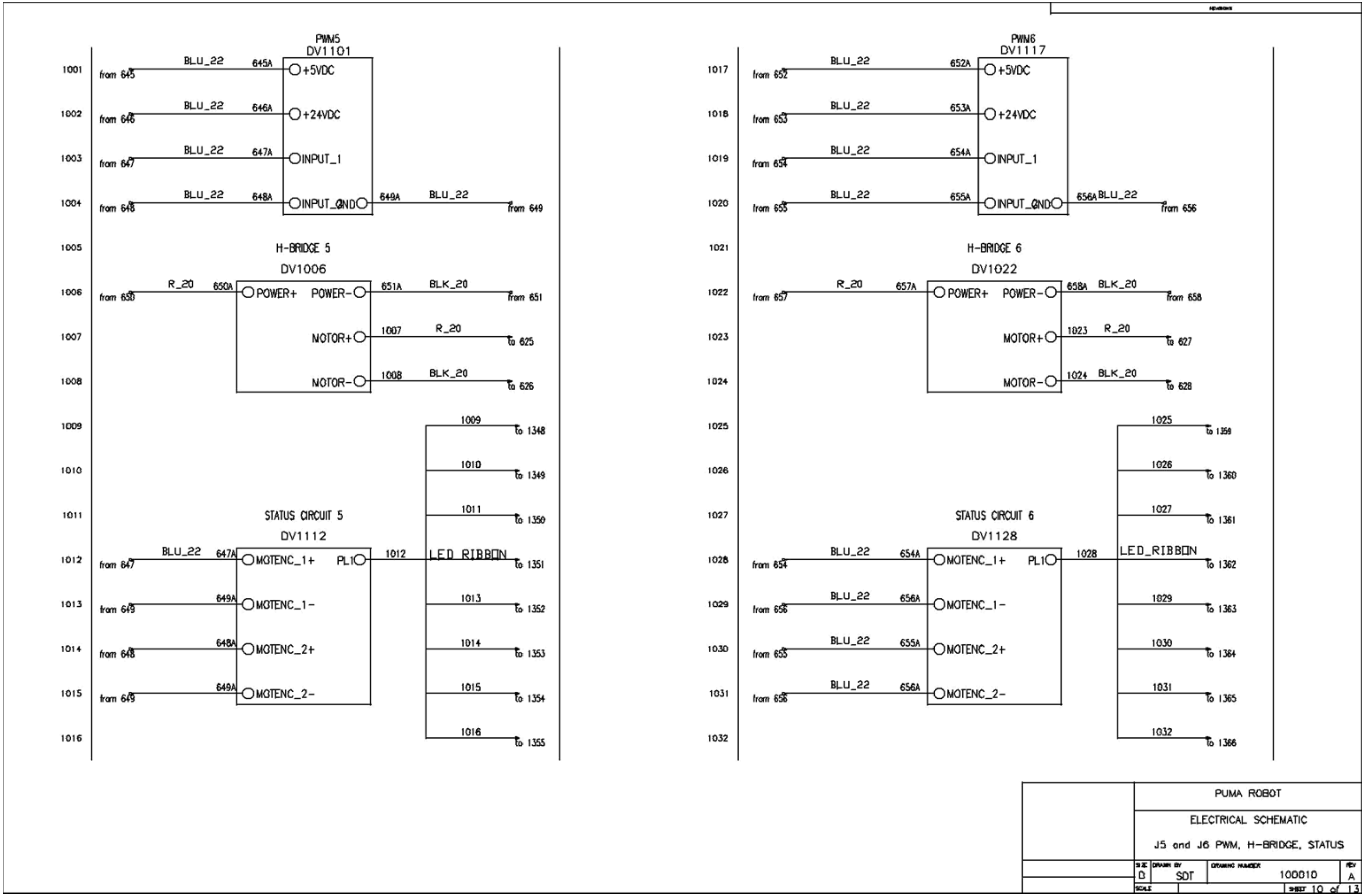
J1 and J2 PCBs



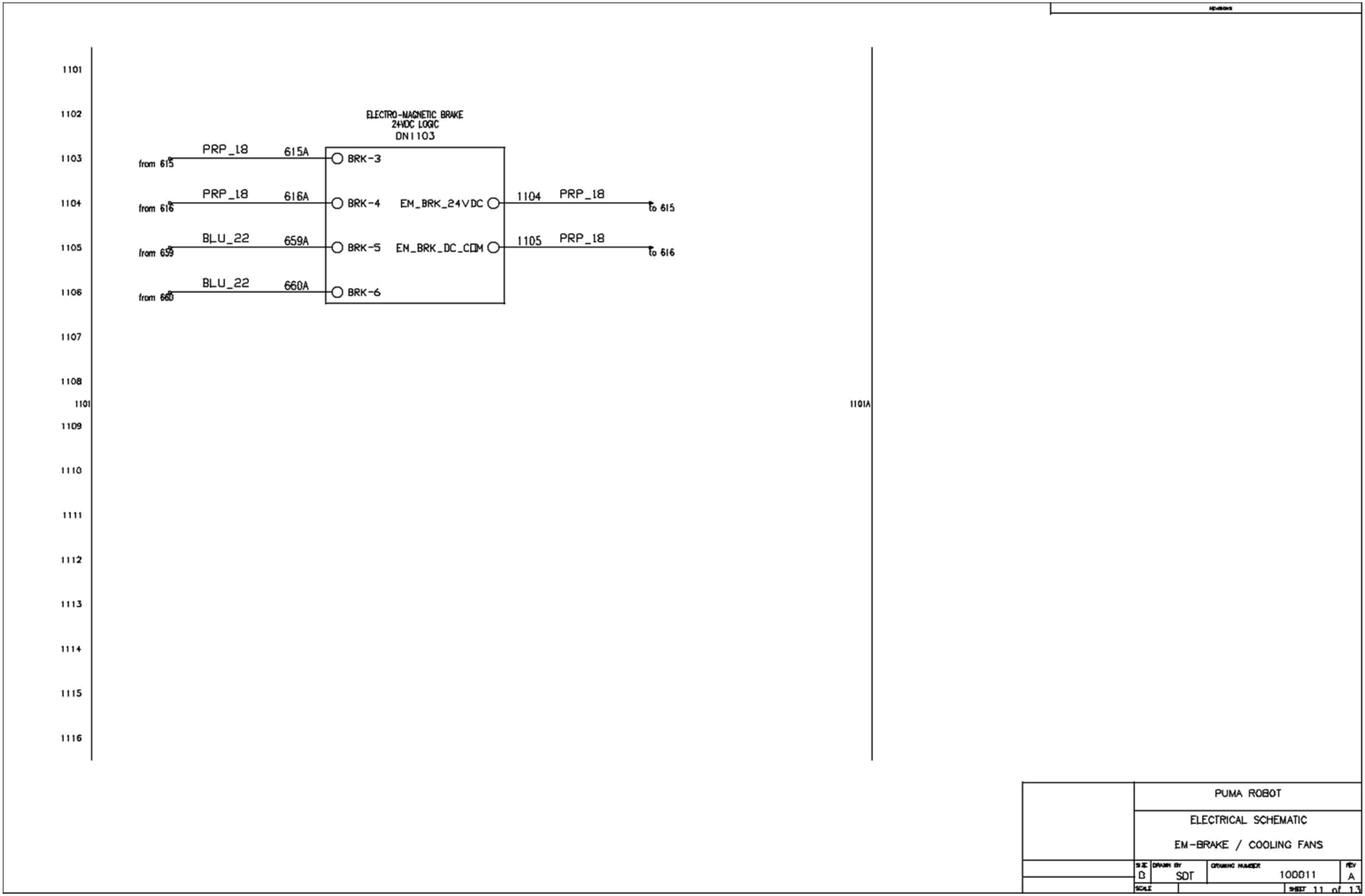
J3 and J4 PCBs



J5 and J6 PCBs

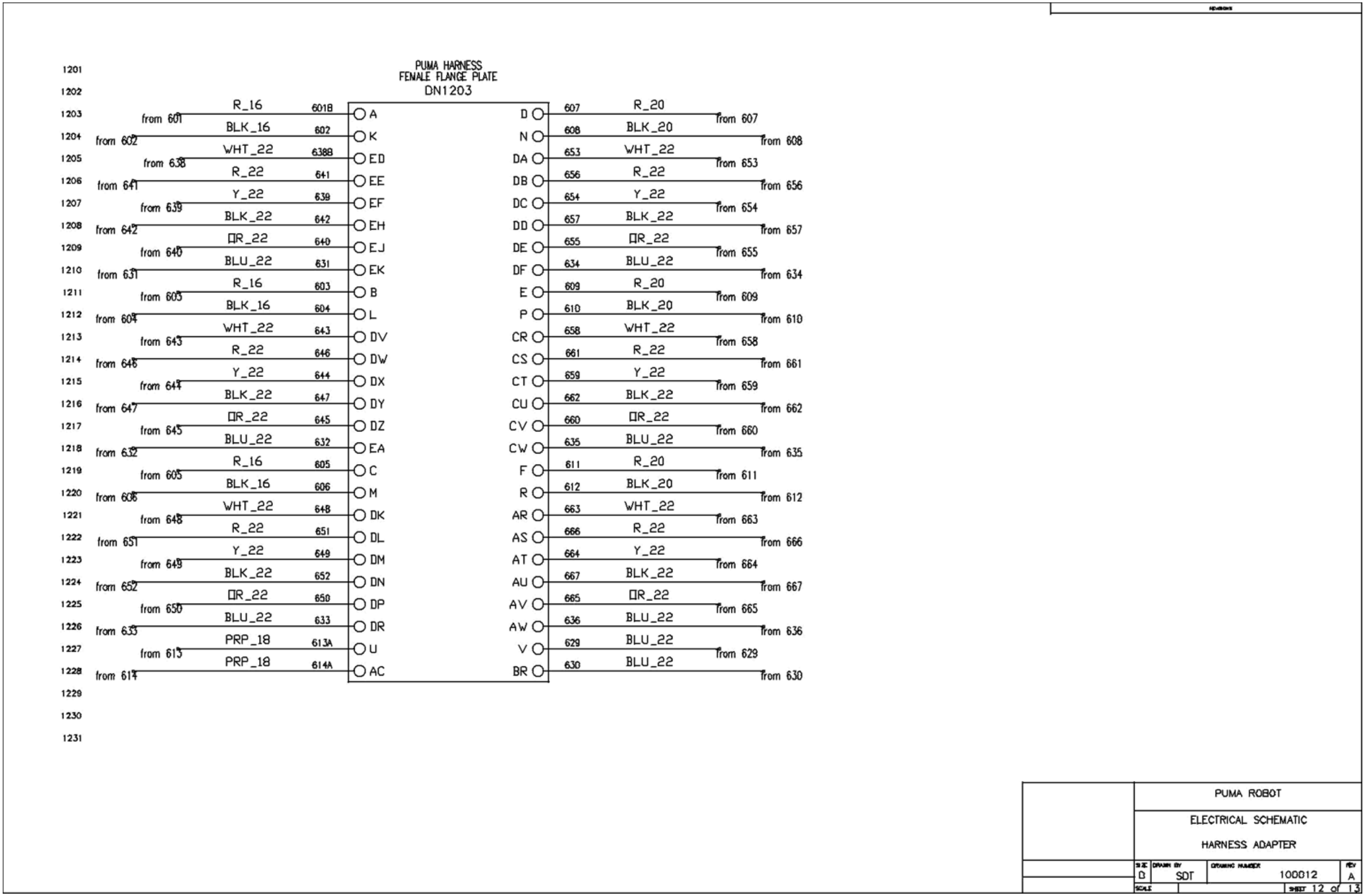


EM Brake

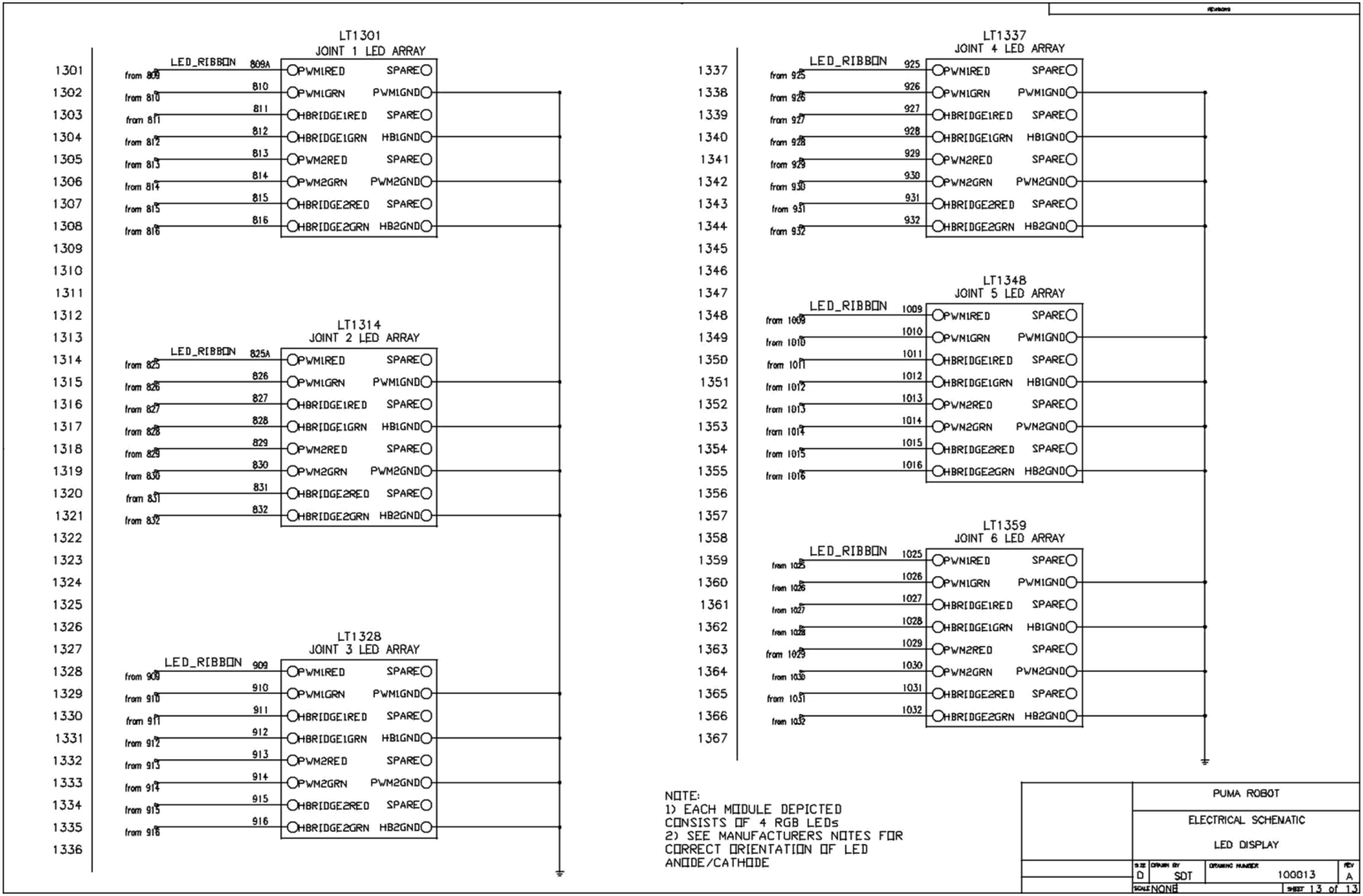


1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1101A
ELECTRO-MAGNETIC BRAKE
24VDC LOGIC
DN1103
from 615
PRP_18
615A
BRK-3
from 616
PRP_18
616A
BRK-4
EM_BRK_24VDC
1104
PRP_18
to 615
from 659
BLU_22
659A
BRK-5
EM_BRK_DC_COM
1105
PRP_18
to 616
from 660
BLU_22
660A
BRK-6
PUMA ROBOT
ELECTRICAL SCHEMATIC
EM-BRAKE / COOLING FANS
B
SDT
100011
A
11 of 13

Harness Adapter



LED Display


LT1301
JOINT 1 LED ARRAY
LED_RIBBON
LT1314
JOINT 2 LED ARRAY
LT1328
JOINT 3 LED ARRAY
LT1337
JOINT 4 LED ARRAY
LT1348
JOINT 5 LED ARRAY
LT1359
JOINT 6 LED ARRAY
PWM1RED SPARE
PWM1GRN PWM1GND
HBRIDGE1RED SPARE
HBRIDGE1GRN HB1GND
PWM2RED SPARE
PWM2GRN PWM2GND
HBRIDGE2RED SPARE
HBRIDGE2GRN HB2GND
NOTE:
1) EACH MODULE DEPICTED CONSISTS OF 4 RGB LEDs
2) SEE MANUFACTURERS NOTES FOR CORRECT ORIENTATION OF LED ANODE/CATHODE
PUMA ROBOT
ELECTRICAL SCHEMATIC
LED DISPLAY
D
SDT
100013
A
NONE
13 of 13