Monkeybot Report

Sebastian Martinez and Andrew Wilson
6/8/2011

Contents

1. Introduction and Motivation

The monkeybot is a two-link brachiating robot designed to climb along vertical walls. It attaches to the wall surface using electromagnets at the end of each link. A DC motor at the pivot provides torque to actuate the robot during its dynamic swinging motion. The energy pumped into the system by this torque should allow the robot to overcome friction and swing side to side or climb. Northwestern University's laboratory for intelligent mechanical systems can use the monkeybot for research in dynamic locomotion of hybrid mechanical systems. Figure 1 below shows the monkeybot in its original state with its existing hardware.

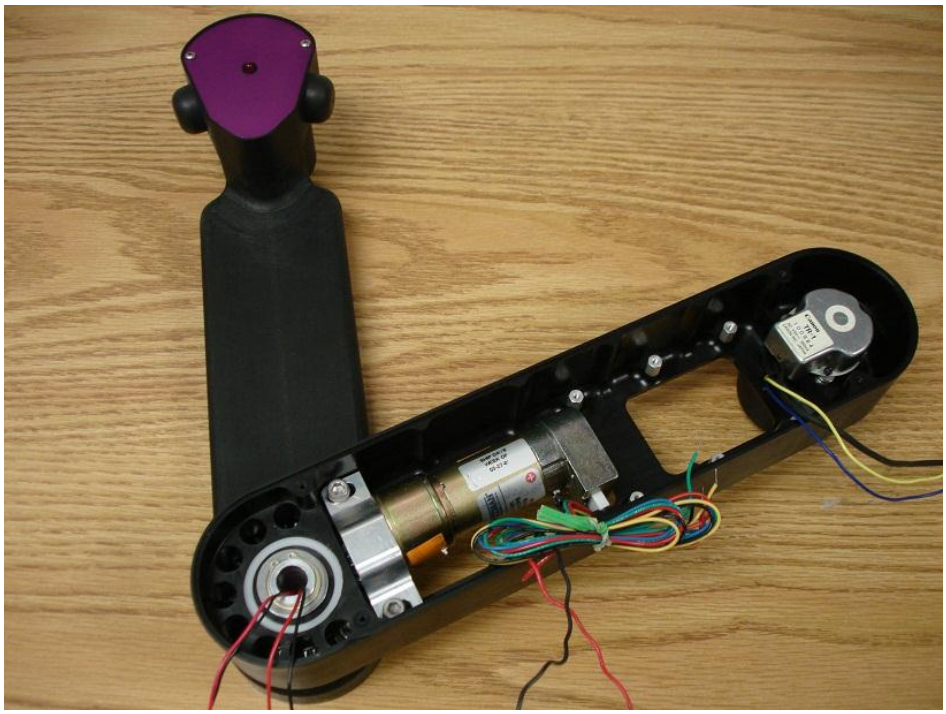


Figure 1: Existing monkeybot hardware

The monkeybot has completed open-loop, timed swinging motions successfully before the start of this project. However, in order to further investigate and integrate a more robust closed-loop control strategy, more sensors were required.

2. Sensor and Hardware Requirements

Existing Monkeybot Hardware

The mechanical portion of the monkeybot was designed by Kinea Design. The following section describes the existing mechanical components and properties of the hardware.

Pittman GM8224 Motor

The motor used in the monkeybot runs nominally on 24V with 4.33 ohm coil resistance. The rated stall current of the motor is 5.54A and the maximum torque is 2 Nm. Although the maximum amperage is rated at over 5A, during testing, we never experienced current draws over 2A. The maximum speed of the motor is about 500 rpm and it is geared 19:5:1. The motor is attached to a 500 line encoder with A, B, and index signal pins. The motor encoder runs on 5V. Figure 2 to the right shows the motor on the monkeybot.

Figure 2: Pittman GM8224 motor

Electromagnets

There are two electromagnets specially designed for the monkeybot to provide free rotation even when active. At 22 volts, we measured the current draw at between 0.3 and 0.5 A. Although freely moving, there is a nominal amount of friction in the bearings; however, we did not find this to be significant in the operation of the monkeybot.

Figure 3: Electromagnet

Laser Rotary Encoders

Each electromagnet has a shaft which rotates with the magnets on the backside of the assembly. This shaft is designed for the Canon TR-36 laser rotary encoder which uses an optical disc and case which measures the rotation of the disc. This encoder has 3,600 counts per revolution and runs on 5V.

Figure 4: Encoder Optical Disk and Casing

New Monkeybot Hardware

In order to provide more absolute values for feedback, inertial measurement units on each link were desired. Two-dimensional accelerometers could be used to provide feedback on the absolute orientation of each link as shown in Figure 5 below. Adding gyros also provides information on each link's angular velocity. The main advantage of using these sensors is that the data provided is absolute, whereas encoders only provide relative orientations, as shown on the right in Figure 5 below.

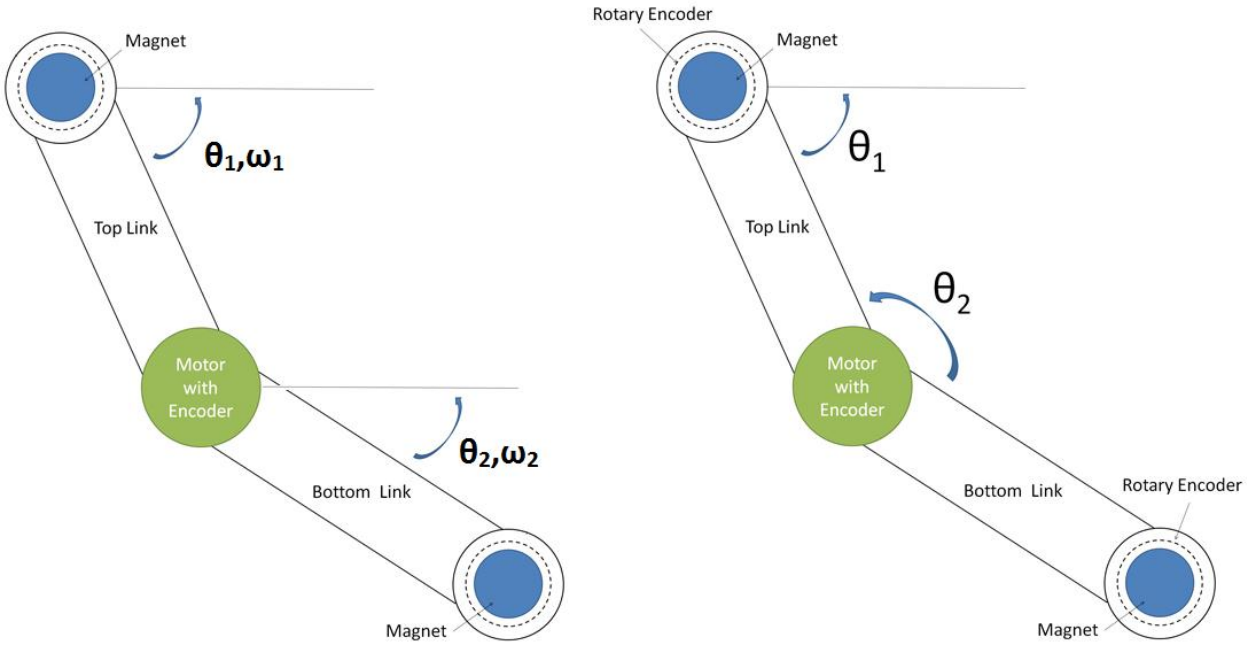


Figure 5: Data provided by IMU's (left) and relative positions provided by encoders (right)

Based on videos of the monkeybot in its swinging motion, the range for both accelerometers and gyros could be estimated. It was observed that the top link swings below 100°/s and the bottom link swings around 400°/s. These angular velocities provide accelerations no more than five times the acceleration of gravity. Thus a gyro with a ±400°/s range and an accelerometer with a ±5g range would suffice for each link.

To aid in the introduction of more robust control strategies, it was determined that a current controller was required. With a current controller, a feedback strategy as shown in Figure 6 below could be implemented.

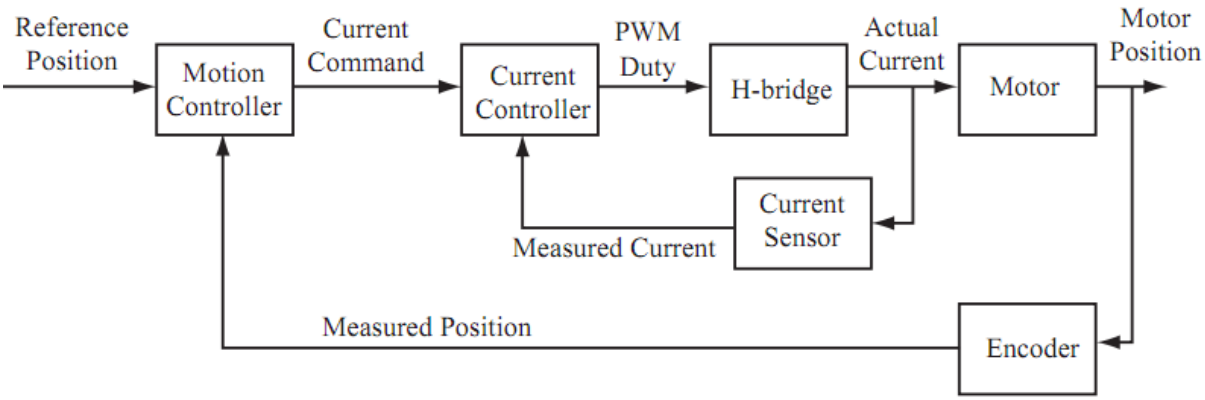


Figure 6: Motor control loop using current sensor

Thus, a current sensor that could handle the current through the motor was deemed advantageous. The maximum current the motor could handle was 5 Amps, so a current sensor that is able to handle this would be safe.

The electromagnets require 22 volts to operate. The microcontroller, however, uses a 3.3 V logic level. Hardware to manage the power and step the voltage down was necessary. A 5 volt line was also needed for certain components. Figure 7 below shows the overall relationships and hierarchy for: power management, control, sensors, and communication.

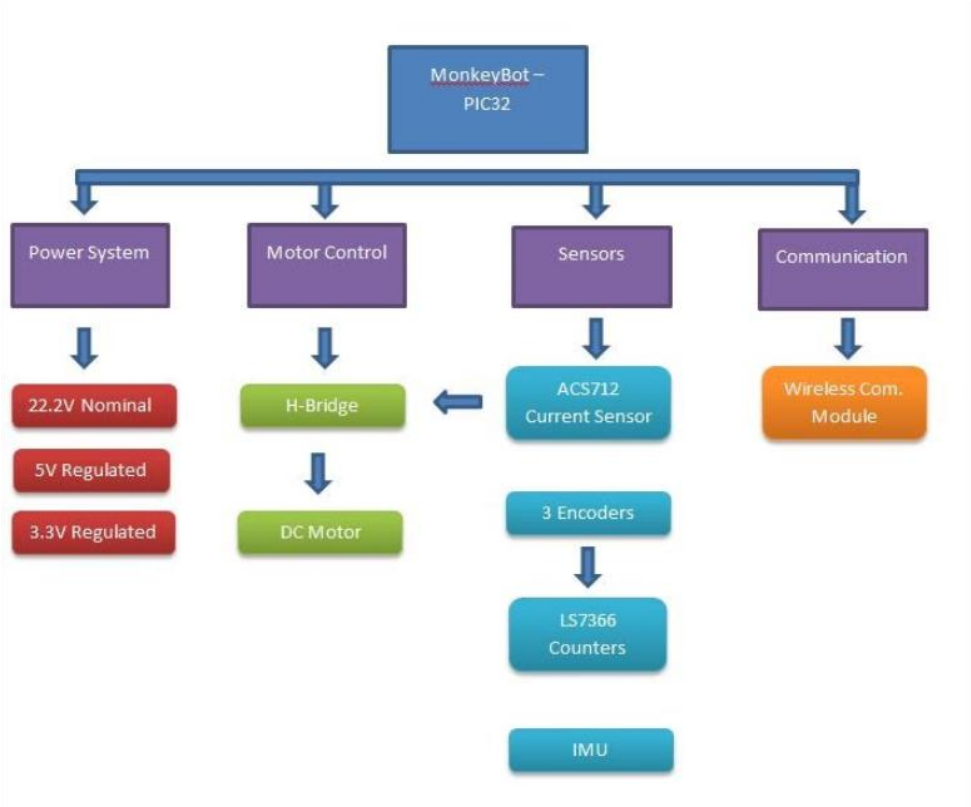


Figure 7: Overall System and Component Heirarchy

3. Electrical Component Selection

A challenge in the electrical design of the monkeybot was finding appropriate components to satisfy all of the hardware and sensor requirements for the monkeybot. We had a few basic groups components: motor control, orientation/speed sensing, and communication. This section will break down the exact components that were used in this design iteration of the monkeybot.

Motor Control

The crux of the motor control consisted of two basic components – the current sensor, and the motor driver. Initially, we planned to use an off the shelf motor driver controlled by a filtered pwm signal; however, the driver was burnt out during testing, so we then decided to use the L298N chip.

L298N Full-Bridge Driver

The L298N is a full-bridge driver which has two channels. Each channel can carry up to 2A, and the circuit can be parallelized resulting in a 4A driver. Although the stall current rating on the monkeybot motor is 5.5A, during testing, we didn't see current over 2A, and using the L298N in parallel didn't present any problems during testing. If the motor does require over 4A, a different motor driver would have to be used. We had two inputs to the L298N, a digital direction pin, and a PWM signal. A third digital pin could be useful to turn the L298N on or off, allowing the motor to coast instead of brake.

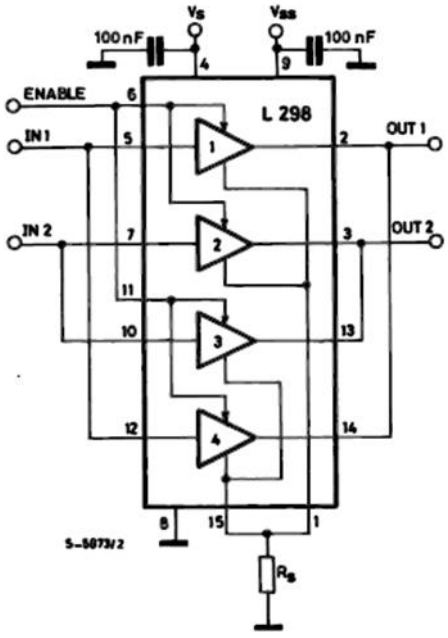


Figure 8: Parallel connection of the L298N

ACS711 Current Sensor

The current sensing chip we chose to use was the ACS711. There are two different chips with different sensitivities, +-12.5A and +-25A. Since the motor can't come close to the lower bound, we used the +-12.5A chip to get a sensitivity of 110mA from the IC. The chip runs on 3.3V and outputs between 0-3.3V with 0A at 1.65V. Since the motor that is being only uses a fraction of the current that the sensor can measure, the sensitivity of the sensor can be boosted using an op-amp to amplify the output of the current sensor before the analog input on the PIC measures the signal. The op-amp circuit is a standard non-inverting amplifier design. The schematic of the current sensor and op-amp is shown below in Figure 9.

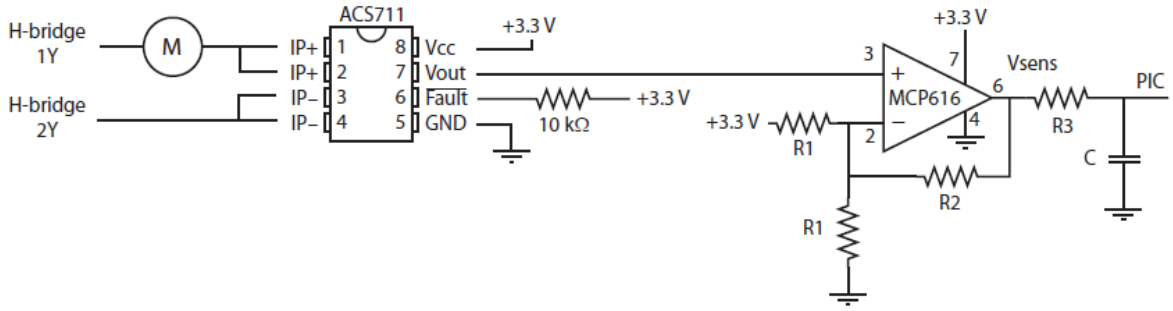


Figure 9: Current sensor and op-amp schematic

Sensing

Pololu Dual Axis Gyro

Although this gyro provides both pitch and yaw (x and z) rotational velocities, for the monkeybot, we only use the yaw axis of the gyro. The feature of this gyro is that is provides two different sensitivity levels: up to 100 degree/sec and up to 400 degrees/sec. There are two separate pins for the yaw rotation, one for each sensitivity. The sensor runs on 3.3V and the ouputs are 0-3.3V with the zero velocity at 1.65V. The gyro sensor is also gravity independent, so orientation does not affect the output.

Pololu 3-Axis Accelerometer

The Pololu accelerometer measures all three axis; however, again, we only need planar motion, so we have used the x and y outputs. This accelerometer also provides two different sensitivity levels, 1.5g and 6g. Unlike the gyros, the sensitivity is selected with a digital pin and there is one analog output pin for each axis. The sensor runs on 3.3V and the outputs are 0-3.3V with zero acceleration at 1.65V. Gravitational acceleration is captured by the sensor, so at zero velocity, the accelerometer measures the component of gravity along an axis.

Figure 10: MMA7361L 3-Axis Accelerometer ±1.5/6g

LS7366R Counter

The LS7366R counters are SPI 32-bit counters which can measure two counting pins and an index pin. The use of these chips frees up the PIC timers and allows the PIC to poll the counters over an SPI channel to get the current count.

Power

DC-DC Converter - EC4BW

Since the monkeybot motor and electromagnets need a high voltage source, we decided to use a DC-DC converter to step down to 5V. Since both the magnet and switching electromagnets are very noisy, the converter provides a better output signal with very low ripple despite any inconsistency in the input. It also allows the monkeybot to operate at different voltages if a different motor or battery combination is tested. The EC4BW converter accepts input voltages from 9-36V and can output up to 2A of current.

N-channel Power MOSFET

We decided to use a power MOSFET as the switch for the electromagents since they draw very little power in either on or off states and are easy to interface with the PIC. We first tried a SO-23 surface mount package; however, the transistors could not handle the current and did not have a large enough thermal rating. The MOSFETs that worked were the NTD5865NL transistors which have a 52W power rating. The special characteristic of these transistors is a low gate transition voltage, under 2V to allow direct connection to the PIC. In the future, it may be best to add an optoisolator in case of transistor failure; however, we have not had any issues with these transistors.

Communication

XBEE Wireless Module

The XBee module is a well-documented serial communication device which we decided to use as our primary means of data transfer and for programming our PIC onboard the monkeybot. The XBee chip simply acts as an invisible serial cable, so implementation is very straightforward. The baud rate had to be lowered to 38400, however, to get reliable communication. Higher transfer rates may be possible by optimizing the XBee internal parameters.

4. Circuit Design

The design of the monkeybot circuit was particularly challenging because of the space limitation based on the existing mechanical device. The amount of space that the main circuit board and electrical components could occupy is 2.4" W x 3.6" L x 1.5" H. The previous electrical design of the monkeybot used the first version of the NU32 PIC breakout board, and secondary electrical component board to interface the motor and encoders. In order to add the additional circuitry for the accelerometers and gyro components, we decided early in the design phase to create an integrated circuit board with the PIC included to maximize the amount of room we had to work with. The schematic and board layout for the board designed can be seen in Appendices A&B. Figure 11 below shows the designed board with the PIC32 on it.

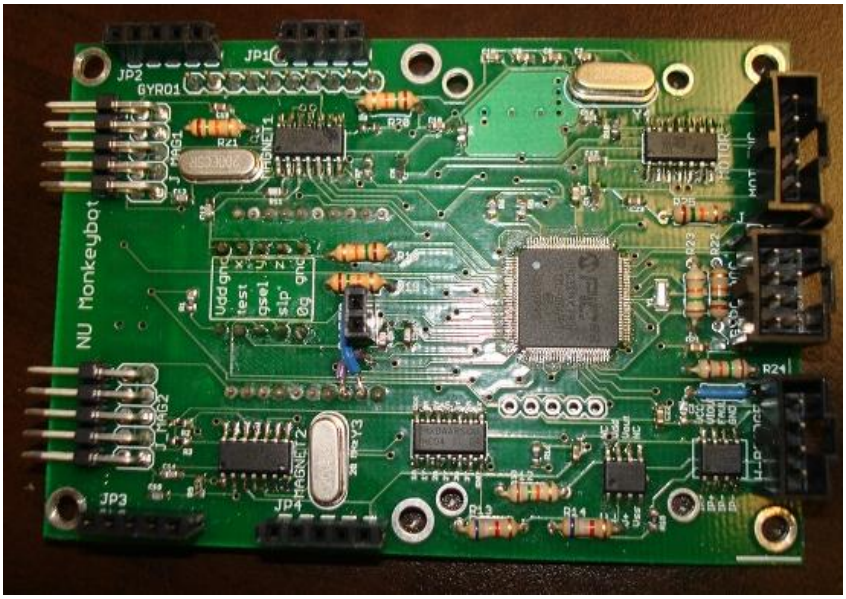


Figure 11: PCB with on-board PIC32

The second consideration in the circuit design was to try to isolate the high and low voltage circuitry as much as possible. We decided to create a top layer board which would contain the high voltage motor driver and electromagnet inputs as well as the DC-DC converter. The 5V

output from the converter is connected to the lower logic board which contains all of the sensors and the PIC. This attempt to separate the voltages seemed to protect the circuitry. The one exception that we overlooked was the transistor gates which are tied to both boards and not protected in case the transistor fails. Figure 12 below shows the power management board which mounts on top of the board with the PIC32 on it.

Figure 12: Power management board

The monkeybot hardware provides an opening on the back of the unit, so the backside of the circuit board is accessible without opening the monkeybot case. Therefore, we wanted to exploit this by providing the reset and programming button as well as the power switch and XBEE module on the backside of the logic board. In addition, the LED power and two indicator LEDs are on the back for troubleshooting. The robot can then be controlled and reprogrammed without reopening the case. The backside of the board with the switches and LED's can be seen through the opening in the figure below.

Figure 13: Backside of mounted PCB

5. Software

PIC32

The code on the monkeybot's PIC32 was written in C using the MPLAB IDE compiler. It was separated into three source files and two header files: one header and source file pair is for the decoder chips used (7366R), another header and source file pair is the NU32v2 code which enables UART and serial communication, and the last source file contains the main monkeybot code. The pseudo code in the main monkeybot source file is as follows:

```
Initialize all analog inputs, timers, interrupts, PWM;
Turn electromagnets off;
while(TRUE)
{
        If start command sent
        {
                Turn an electromagnet off to let go;
                Delay;
                Turn motor on;
                Delay;
                Turn motor off;
                Delay;
                Turn same electromagnet on to grab on;
        }
}
ISR to send data to Processing
{
        If not swinging (start==0)
        {
                Calculate orientation of each arm;
                Send orientation of each arm and the current delay times to processing
        }
        Else (swinging, start==1)
        {
                Send all sensor data to processing for plotting;
        }
}
ISR to receive data from Processing
{
        Change states of electromagnets;
        Start swinging motion program (start==1);
        Change delay times;
        Set direction to swing in;
}
```

Processing

A basic processing GUI was created to allow a user to quickly setup the monkeybot and set open loop timing values. A screenshot of this screen can be seen in Figure 14 below.

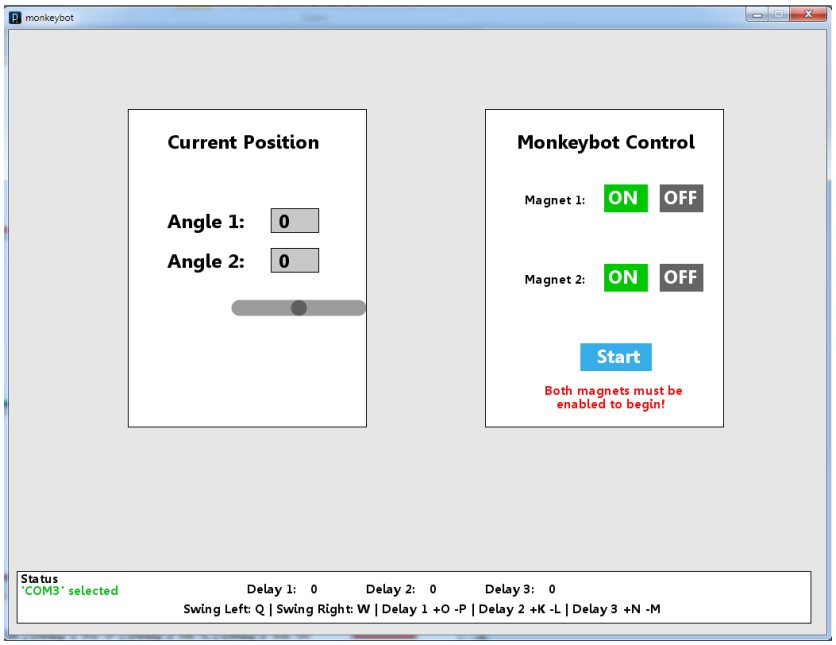


Figure 14: Initial GUI Screen

The GUI also displays a plot of several sensors as the monkeybot program executes. Figure 15 displays this screen.

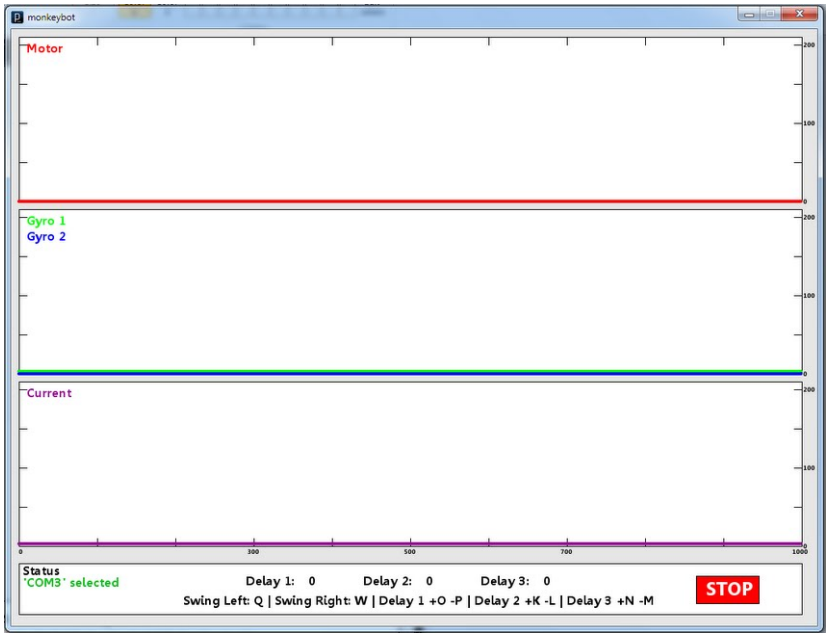


Figure 15: Plotting GUI Screen

6. Results

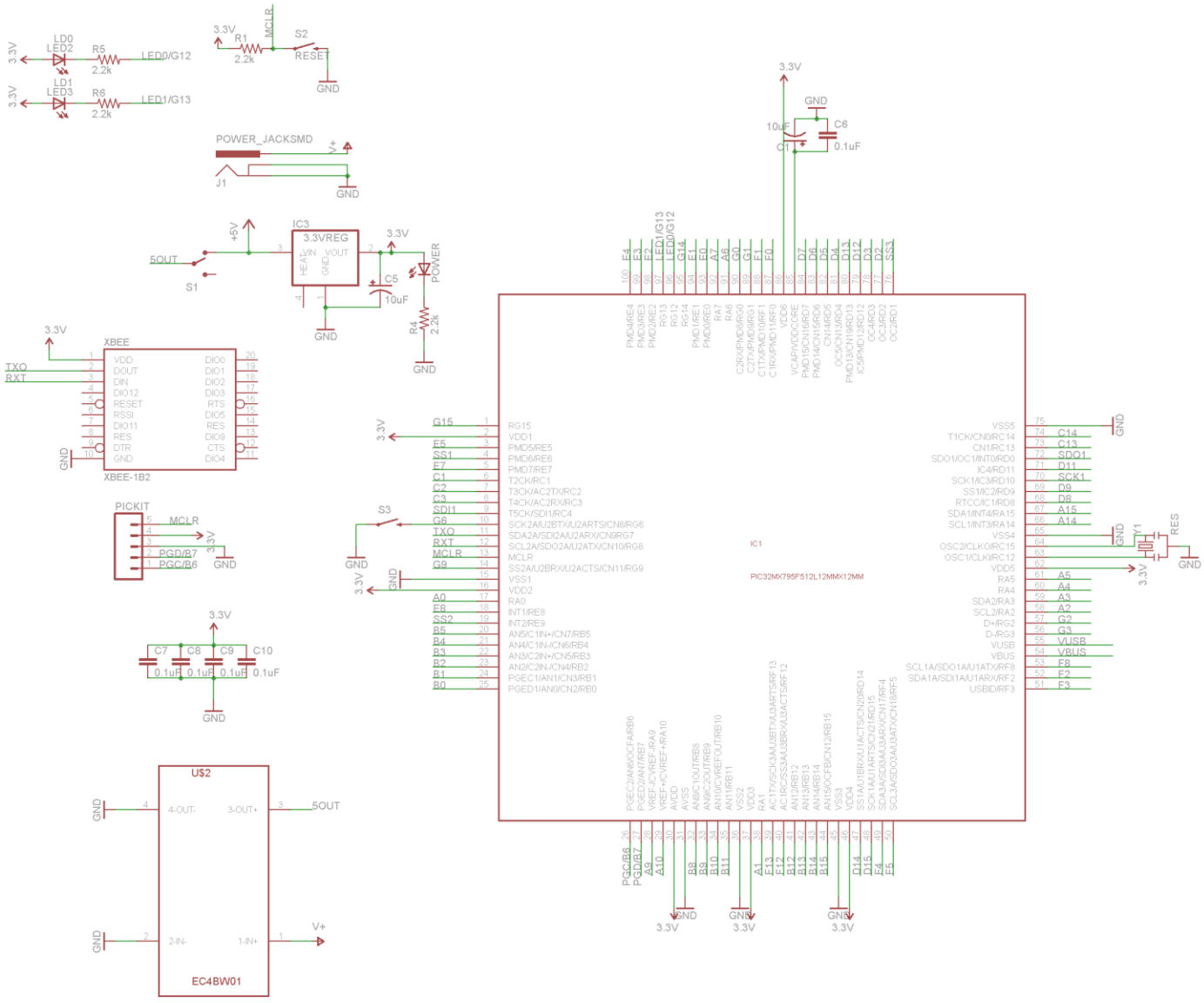
The monkeybot was able to swing on command in whichever direction desired. The delay times for the open loop swinging motion could also be modified to study different swings. Visual feedback of the sensors was also provided by the Processing GUI. This was all enabled with the addition of the XBee wireless communication module and extra sensors. The PIC32 can also be reprogrammed wirelessly.

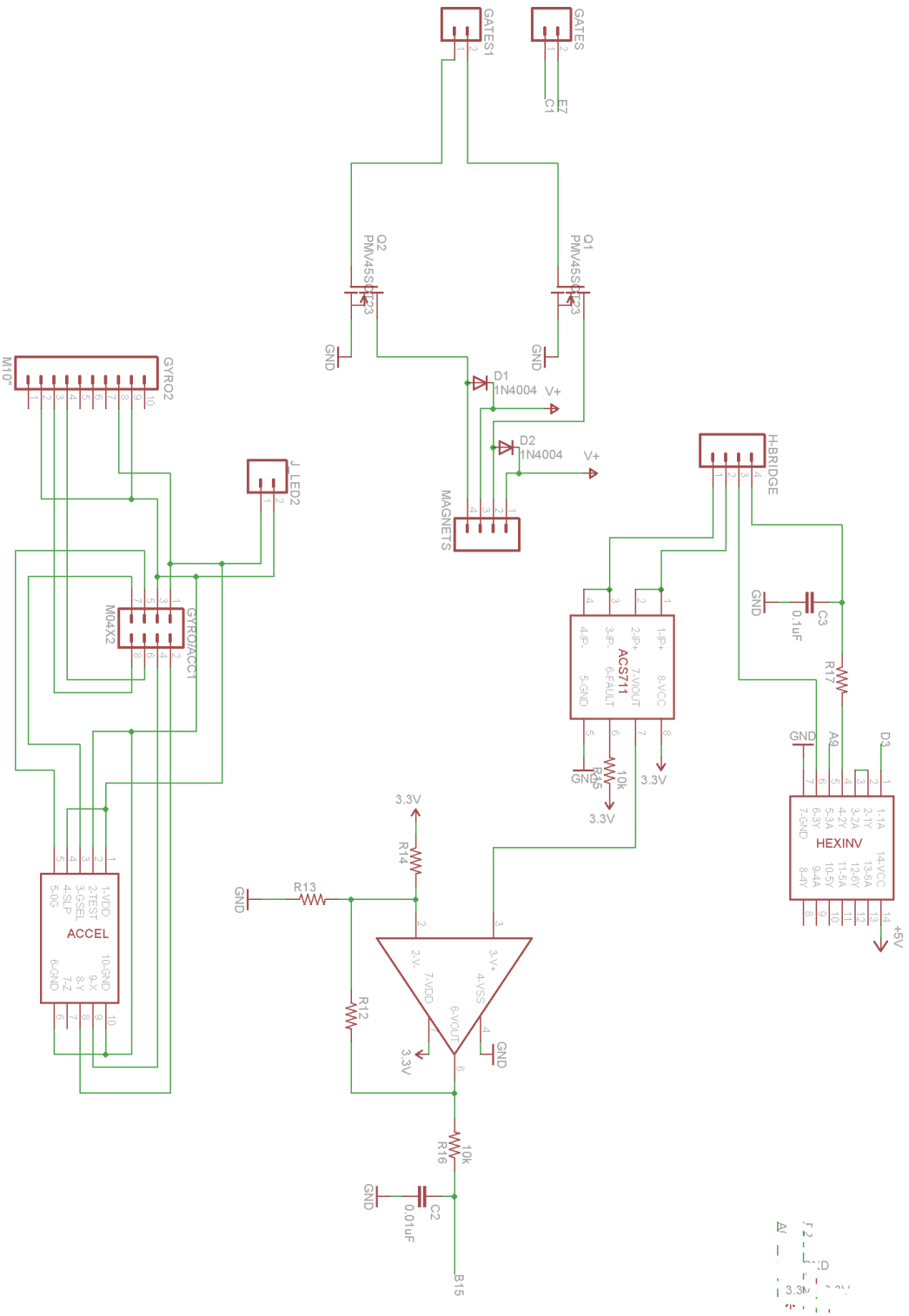
7. Conclusion

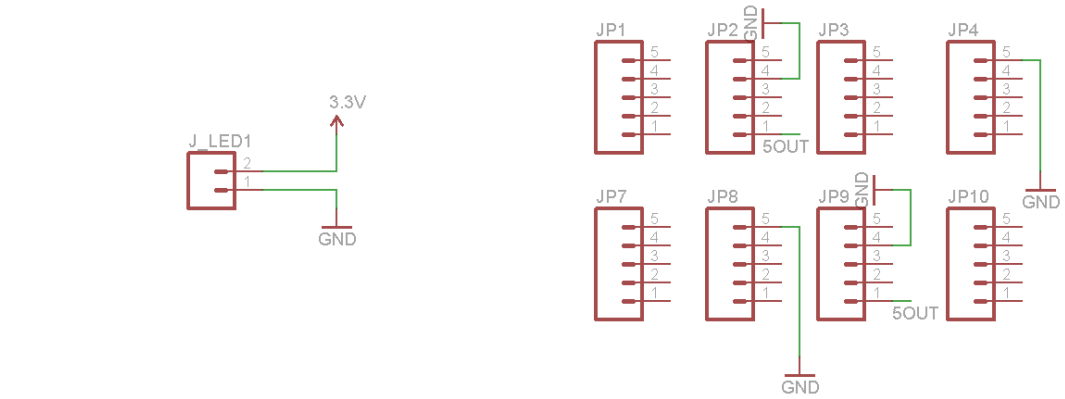
The monkeybot project provided a great means for hands-on electrical design and implementation. The biggest challenge was balancing the small footprint we had to work with and the power requirements of the monkeybot. The DC-DC converter was a great component which gave us reliable logic power. Our biggest problems were implementing the MOSFET transistors and h-bridge driver. The high voltage at some point burnt out each of these components; however, using the larger transistors and the L298N motor driver provided an adequate solution. After testing the monkeybot, there are still many open problems, such as dealing with the large friction on the wall through the swing. The speed of the motor may need to be increased, or a better control feedback algorithm could be developed to create a swing up motion. It appears to be impossible to correctly gauge the trajectory of the monkeybot purely through open loop timing tuning.

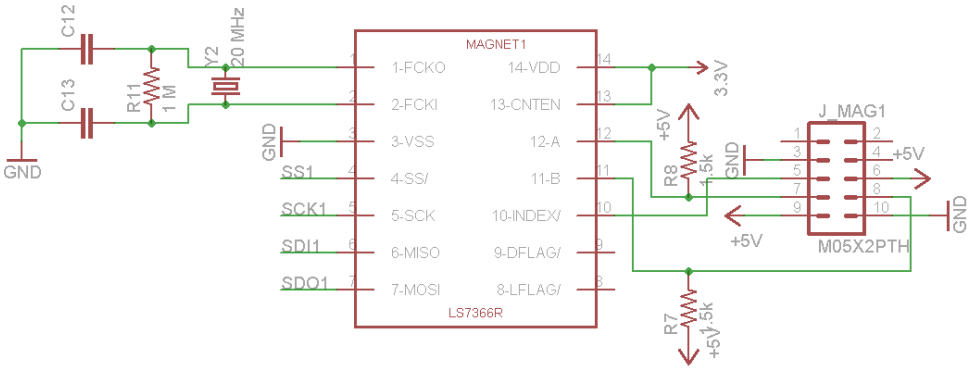
This circuit design and electrical testing of the monkeybot should provide the sensing capabilities necessary to test some basic feedback controllers and hopefully result in a better trajectory and further the research behind the monkeybot.

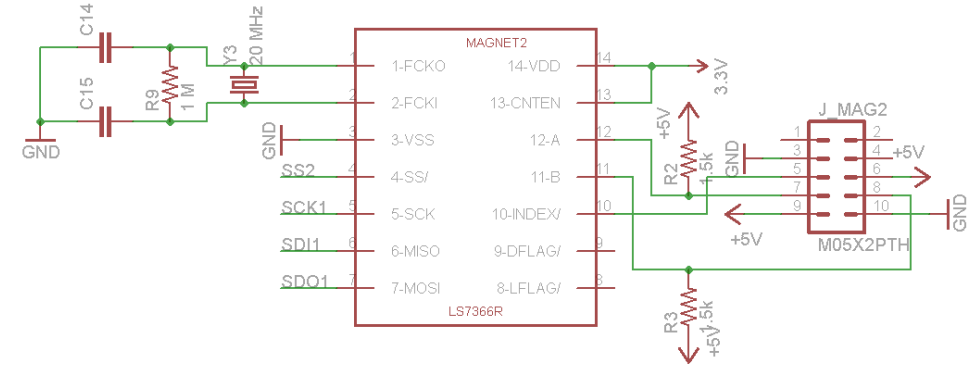
Appendix A: Eagle PCB Schematic

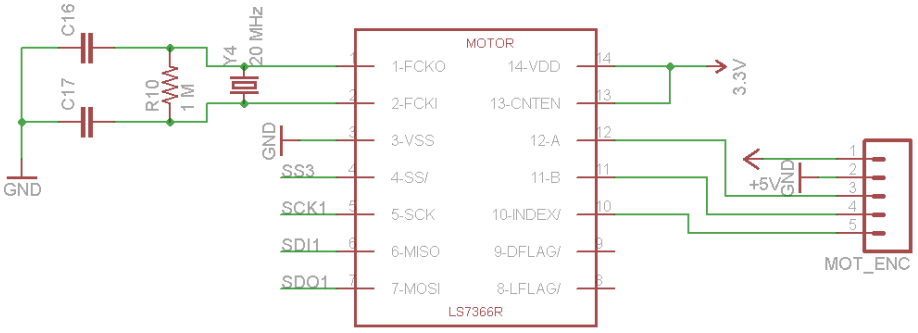


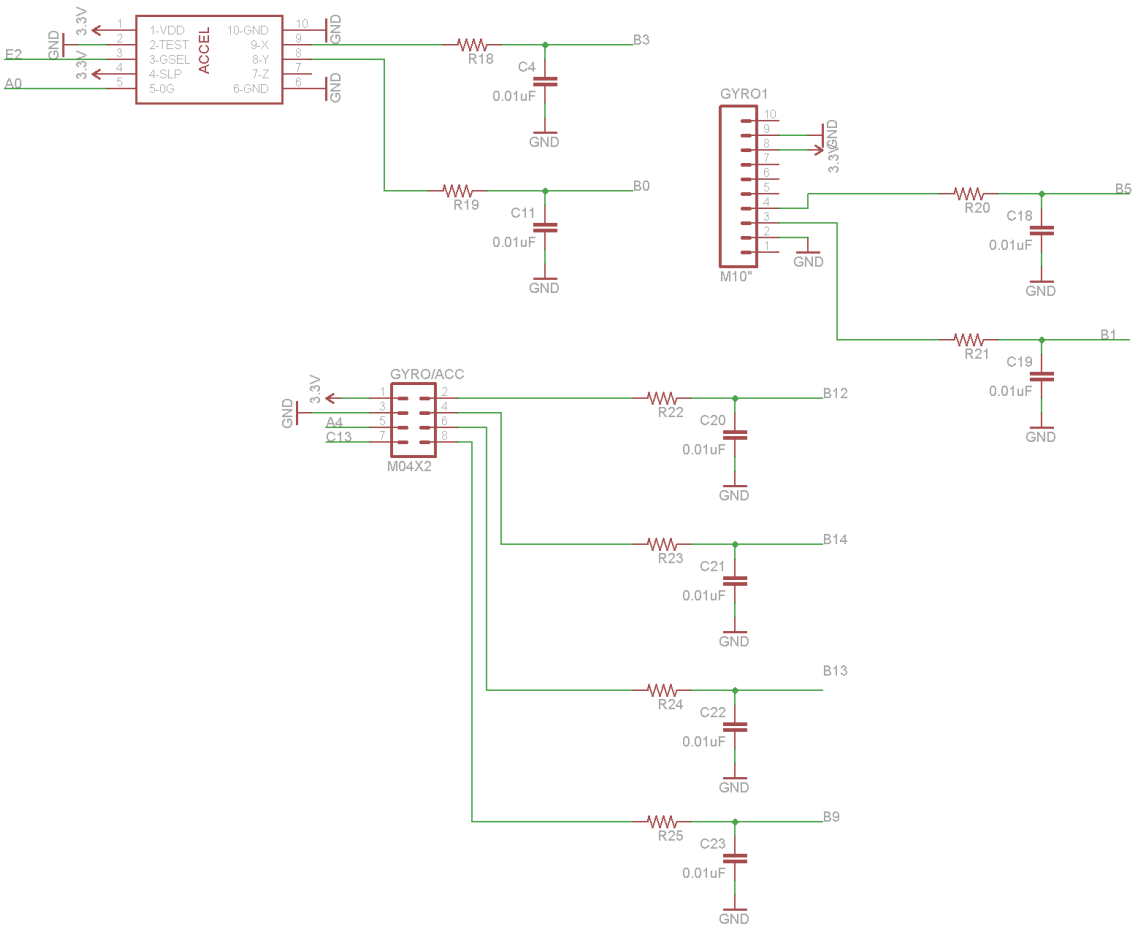

GATES1
GATES
E7
C1
Q2
PMV45SN
Q1
PMV45SN
GND
GND
D1
1N4004
V+
D2
1N4004
V+
MAGNETS
H-BRIDGE
GYRO2
M10
J_LED2
M04X2
GYRO/ACC1
ACS711
1-IP+
2-IP+
3-IP-
4-IP-
5-GND
6-FAULT
7-VIOUT
8-VCC
GND
C3
0.1uF
R17
GND
A9
D3
HEXINV
+5V
R15
10k
3.3V
3.3V
R14
R13
GND
3.3V
R12
3.3V
GND
R16
10k
GND
C2
0.01uF
B15
ACCEL
1-VDD
2-TEST
3-GSEL
4-SLP
5-0G
6-GND
10-GND
9-X
8-Y
7-Z


3.3V
J_LED1
GND
JP1
JP2
GND
5OUT
JP3
JP4
GND
JP7
JP8
JP9
GND
5OUT
JP10
GND

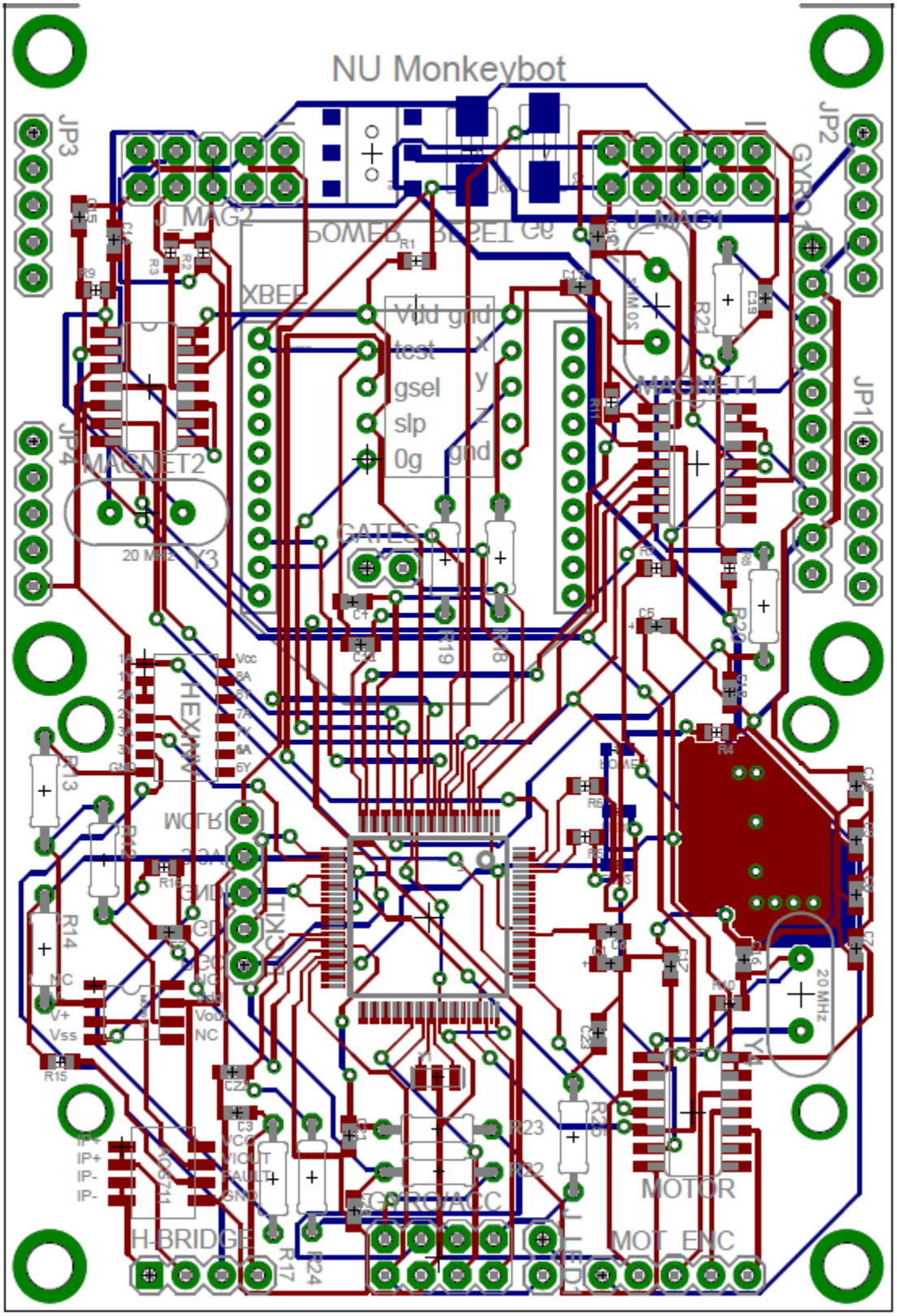

C12
C13
R11
1 M
Y2
20 MHz
GND
MAGNET1
1-FCKO
2-FCKI
3-VSS
4-SS/
5-SCK
6-MISO
7-MOSI
14-VDD
13-CNTEN
12-A
11-B
10-INDEX/
9-DFLAG/
8-LFLAG/
LS7366R
SS1
SCK1
SDI1
SDO1
3.3V
+5V
R8
1.5k
GND
J_MAG1
+5V
+5V
GND
M05X2PTH
R7
1.5k
+5V

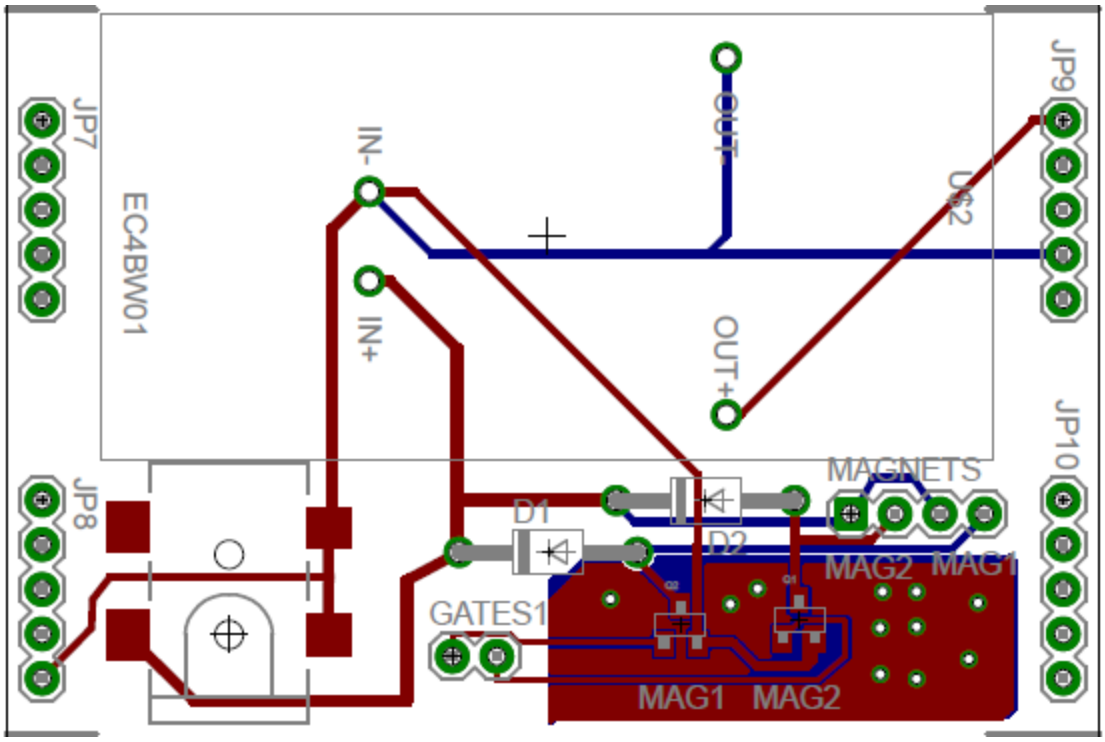

C14
C15
R9
1 M
Y3
20 MHz
GND
MAGNET2
1-FCKO
2-FCKI
3-VSS
4-SS/
5-SCK
6-MISO
7-MOSI
14-VDD
13-CNTEN
12-A
11-B
10-INDEX/
9-DFLAG/
8-LFLAG/
LS7366R
SS2
SCK1
SDI1
SDO1
3.3V
+5V
R2
1.5k
GND
J_MAG2
+5V
+5V
GND
M05X2PTH
R3
1.5k
+5V

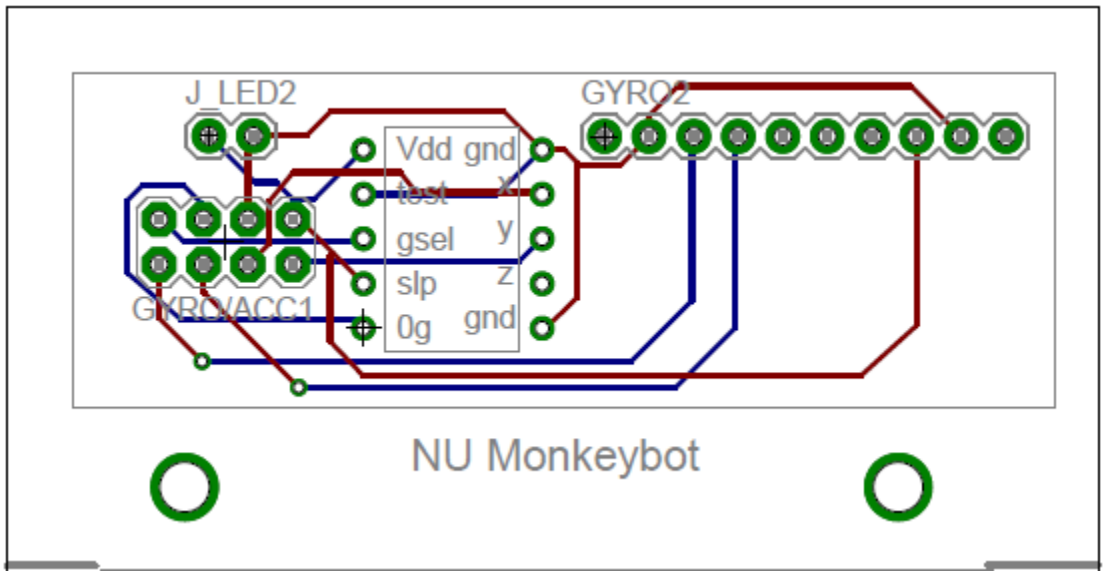

C16
C17
R10
1 M
Y4
20 MHz
GND
MOTOR
1-FCKO
2-FCKI
3-VSS
4-SS/
5-SCK
6-MISO
7-MOSI
14-VDD
13-CNTEN
12-A
11-B
10-INDEX/
9-DFLAG/
8-LFLAG/
LS7366R
SS3
SCK1
SDI1
SDO1
3.3V
+5V
GND
MOT_ENC


ACCEL
1-VDD
2-TEST
3-GSEL
4-SLP
5-0G
10-GND
9-X
8-Y
7-Z
6-GND
3.3V
GND
E2
A0
R18
R19
C4
C11
0.01uF
B3
B0
GYRO1
M10"
R20
R21
C18
C19
B5
B1
GYRO/ACC
M04X2
A4
C13
R22
R23
R24
R25
C20
C21
C22
C23
B12
B14
B13
B9

Appendix B: Circuit Board Layout




JP7
EC4BW01
IN-
IN+
OUT-
OUT+
JP9
JP10
JP8
MAGNETS
D1
D2
MAG2
MAG1
GATES1
MAG1
MAG2


J_LED2
GYRO2
Vdd
gnd
test
x
gsel
y
slp
z
0g
gnd
GYROACC1
NU Monkeybot

Appendix C: Source Code

```
// Monkeybot code

#include <plib.h>
#include <math.h>
#include "NU32v2.h"
#include "7366.h"

// global variables
int HBridgeDuty;
unsigned char readbuf[COUNTER_BYTES];
int start=0;
int delay1=12;
int delay2=25;
int delay3=7;
int direction=1;


int main(void) {
        timesec=0;
        // set PIC32 to max computing power
        SYSTEMConfig(SYS_FREQ, SYS_CFG_ALL);
        INTConfigureSystem(INT_SYSTEM_CONFIG_MULT_VECTOR);
        INTEnableSystemMultiVectoredInt();
  initLEDs();
  LED0 = 1;
  LED1 = 1;
  initSerialNU32v2();
        setup_counters();

        CloseADC10();
        #define PARAM1  ADC_MODULE_ON | ADC_FORMAT_INTG | ADC_CLK_AUTO |
ADC_AUTO_SAMPLING_ON
        #define PARAM2  ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE | ADC_SCAN_ON |
ADC_SAMPLES_PER_INT_16 | ADC_ALT_BUF_OFF | ADC_ALT_INPUT_OFF
        #define PARAM3  ADC_CONV_CLK_INTERNAL_RC | ADC_SAMPLE_TIME_31
        #define PARAM4  ENABLE_AN0_ANA | ENABLE_AN1_ANA | ENABLE_AN2_ANA |
ENABLE_AN3_ANA | ENABLE_AN5_ANA | ENABLE_AN15_ANA
        OpenADC10( PARAM1, PARAM2, PARAM3, PARAM4,0);
        EnableADC10();


  // Setup and turn off electromagnets
  EMAG1 = 0; EMAG2 = 0;
  TRISEbits.TRISE7 = 0; TRISCbits.TRISC1 = 0;
        //Direction Output
        DIR = 1;
```

```
        TRISAbits.TRISA9 = 0;
        //g-select Outputs
        GSEL1 = 0; GSEL2 = 0;
        TRISEbits.TRISE2 = 0; TRISCbits.TRISC13= 0;
        //0g Inputs
        TRISAbits.TRISA0 = 1;
        TRISAbits.TRISA4 = 1;

        // 20kHz PWM signal, duty from 0-1000, pin D3
        OpenTimer2(T2_ON | T2_PS_1_4, 1000);
        OpenOC4(OC_ON | OC_TIMER2_SRC | OC_PWM_FAULT_PIN_DISABLE, 0, 0);
        HBridgeDuty = 0;
        SetDCOC4PWM(HBridgeDuty);

// 50Hz ISR
OpenTimer3(T3_ON | T3_PS_1_64, 25000);
mT3SetIntPriority(1);
mT3ClearIntFlag();
mT3IntEnable(1);
        int state=1;
        LED1=!state;
        LED0=!direction;

        while(1) {
                LED1=!state;
                LED0=!direction;
                if(start){
                        if(state==1 && direction==1){
                                EMAG2=0;
                                SetDCOC4PWM(100);
                                DIR=1;
                                delaysec(delay1);
                                SetDCOC4PWM(1000);
                                delaysec(delay2);
                                SetDCOC4PWM(500);
                                delaysec(delay1);
                                EMAG2=1;
                                SetDCOC4PWM(0);
                                state=!state;
                        }
                        else if(state==0 && direction==1){
                                EMAG1=0;
                                SetDCOC4PWM(900);
                                DIR = 0;
                                delaysec(delay1);
                                SetDCOC4PWM(0);
                                delaysec(delay2);
                                SetDCOC4PWM(700);
```

```
                        delaysec(delay1);
                        SetDCOC4PWM(1000);
                        EMAG1=1;
                        state=!state;
                }
                else if(state==1 && direction==0){
                        EMAG1=0;
                        SetDCOC4PWM(100);
                        DIR = 1;
                        delaysec(delay1);
                        SetDCOC4PWM(1000);
                        delaysec(delay2);
                        SetDCOC4PWM(300);
                        delaysec(delay1);
                        SetDCOC4PWM(0);
                        EMAG1=1;
                        state=!state;
                }
                else{
                        EMAG2=0;
                        SetDCOC4PWM(900);
                        DIR = 0;
                        delaysec(delay1);
                        SetDCOC4PWM(0);
                        delaysec(delay2);
                        SetDCOC4PWM(700);
                        delaysec(delay1);
                        SetDCOC4PWM(1000);
                        EMAG2=1;
                        state=!state;
                }
                start=0;
        }
    }
}


// timer3 ISR
// Read the encoder, and send over the data desired by printMode
void __ISR( _TIMER_3_VECTOR, ipl1) T3Interrupt( void) {
    timesec++;

    int angle1;
  int angle2;
    int enc1, enc2, enc3;

    if(!start) {
```

```
        angle1 = (int)(atan2(((float)ACC1X-505.0),((float)ACC1Y-530.0))*180.0/3.1416);

        angle2 = (int)(atan2(((float)ACC2X-505.0),((float)ACC2Y-530.0))*180.0/3.1416);

        sprintf(RS232_Out_Buffer,"%i %i %i %i %i %i\r\n",start, angle1, angle2, delay1,delay2, delay3);
        WriteString(UART3, RS232_Out_Buffer);

        }

        else {

   read_7366(MOT, CNTR, readbuf);
        enc1 = (((int)(readbuf[0]))<<8) | ((int)(readbuf[1]));
        enc1 = (int)((float)enc1*1023.0/39000.0);


        sprintf(RS232_Out_Buffer,"%i %i %i %i %i\r\n", start,enc1,GYRO1HI,GYRO2HI,CURRENT);
        WriteString(UART3, RS232_Out_Buffer);

        }

        // clear interrupt flag and exit
        mT3ClearIntFlag();
}


// UART3 interrupt handler, priority level 2
void __ISR(_UART_3_VECTOR, ipl2) IntUart3Handler(void) {
  // Is this an RX interrupt?
  if(INTGetFlag(INT_SOURCE_UART_RX(UART3))){
    char data = UARTGetDataByte(UART3);


        if(data =='x') {
                //Stop program, both magnets ON
                start = 0;
                EMAG1 = EMAG2 = 1;
        }

    if(data =='e') {
      // Magnet 1 ON
      EMAG1 = 1;
    }

    if(data =='f') {
      // Magnet 1 OFF
      EMAG1 = 0;
    }
```

```
 if(data =='g') {
   // Magnet 2 ON
   EMAG2 = 1;
 }

 if(data =='h') {
   // Magnet 2 OFF
   EMAG2 = 0;
 }

        if(data =='s') {
                // start motion program
                start = 1;
        }
 if(data =='o') {
    delay1 = delay1+1;
        }
        if(data =='p') {
    delay1 = delay1-1;
        }
        if(data =='k') {
    delay2 = delay2+1;
        }
        if(data =='l') {
    delay2 = delay2-1;
        }
        if(data =='n') {
    delay3 = delay3+1;
        }
        if(data =='m') {
    delay3 = delay3-1;
        }
        if(data =='w') {
    direction=1;
                start=1;

        }
        if(data =='q') {
    direction=0;
                start=1;
        }


 // Clear the RX interrupt Flag
 INTClearFlag(INT_SOURCE_UART_RX(UART3));
}
```

```
  // We don't care about TX interrupt
  if(INTGetFlag(INT_SOURCE_UART_TX(UART3))) {
    INTClearFlag(INT_SOURCE_UART_TX(UART3));
  }
}
```

Appendix D: Datasheet Links

Laser Encoders:
https://docs.google.com/viewer?a=v&pid=explorer&chrome=true&srcid=0B_dSf22opav8MDMzMWEwNDgtNGU2Ni00OWY4LTkzOTMtOTI2ZGMwZDQwMGRj&hl=en

Motor:
http://hades.mech.northwestern.edu/images/5/50/Pittmangearmotor.pdf

Accelerometers:
http://www.freescale.com/files/sensors/doc/data_sheet/MMA7361L.pdf

Gyros:
http://www.pololu.com/catalog/product/1267

MOSFET:
http://www.onsemi.com/pub_link/Collateral/NTD5865NL-D.PDF

Hex Inverter:
http://focus.ti.com/lit/ds/symlink/sn7407.pdf

DC-DC Converter:
http://www.cincon.com/data/products/dcdc2_1/EC4BW.pdf

LS7366R Counter:
http://www.lsicsi.com/pdfs/Data_Sheets/LS7366R.pdf

ACS711 Current Sensor:
http://www.allegromicro.com/en/Products/Part_Numbers/0711/0711.pdf

MCP616 Opamp:
http://ww1.microchip.com/downloads/en/DeviceDoc/21613c.pdf

L298N Full-Bridge Driver
http://www.datasheetcatalog.org/datasheet/SGSThomsonMicroelectronics/mXxwur.pdf