Lab 3: Lab Testing Rig

ENGR2210: Principles of Engineering

Byron Wasti | Luke Morris

March 5, 2015

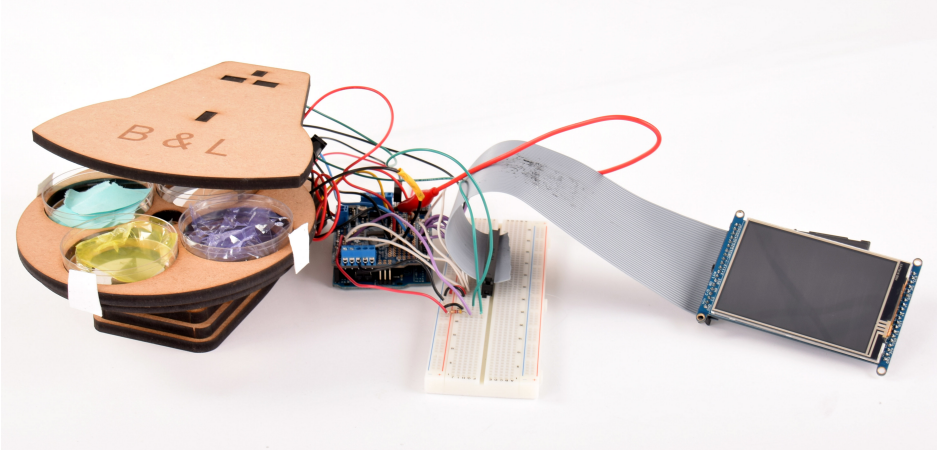

Figure 1: Lab Testing Rig

1 Introduction

In this lab we built a rig for testing the opacity of different samples in Petri dishes. To test the opacity we used a photo-resistor and an LED in pair. For controlling the position of the different Petri dish samples we drove a DC motor with an Adafruit Arduino motor shield. We used an analog reflective sensor to provide positional feedback. Our system makes use of a touch screen and LCD to manually choose which dish to sense or to enter an automatic mode which will take a reading at each Petri dish. No computer is needed to run the system, all it needs is a 12V supply line.

2 Video

Click here to view our lab testing rig in action.

3 Implementation

3.1 Mechanical Design

Figure 2: Design Render

The mechanical design was built using laser cut 1/4" MDF, a Mech.E. Stockroom motor, Petri Dishes and a 1/4" Dowel Pin. The Motor (with attached gearbox) is press fit into four layers of MDF with cut out sections for the additional output of the motor and wiring. These layers were aligned using the 1/4" dowel pin. A circular tray is press fit onto the output shaft of the motor. An additional circular plate was glued to this plate with holes to hold the Petri dishes. We glued white paper on the edge of these plate at the location of the dishes for position tracking. Embedded in the four layers of MDF is also a sensing arm. This arm has the reflective sensor used for tracking the position of the plate and a photo-resistor for taking measurements. The arm cantilevers over the plate to take an opacity measurement from the center of the Petri dish. Mounted above the tray is a plate to block ambient light from interfering with the sense measurements.

3.2 Circuit

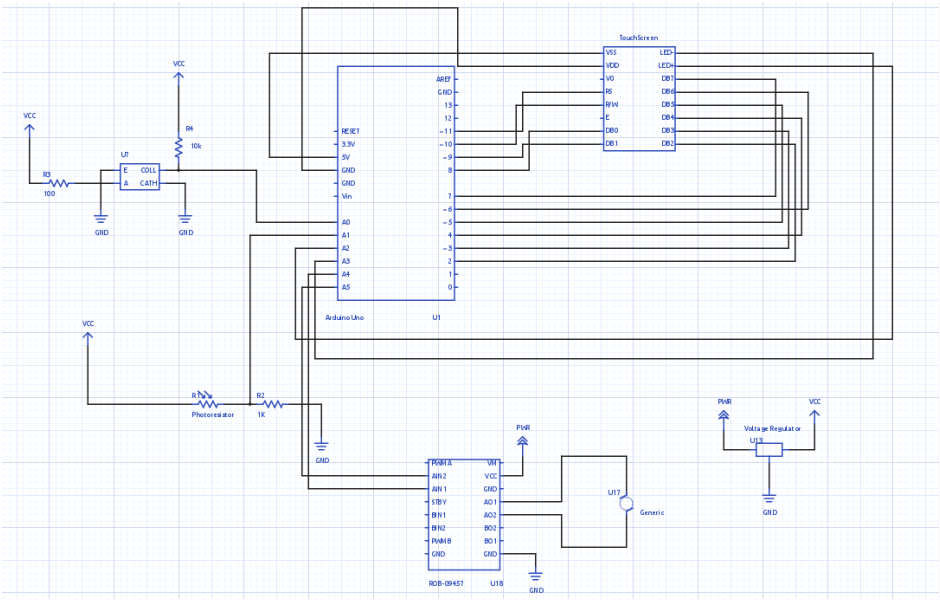

Figure 3: A glimpse of the circuitry used in this lab. The motor breakout board was substituted with a different motor breakout board, and the screen schematic shown does not have touchscreen inputs and outputs.

The circuit we used in this lab is fairly straightforward and self explanatory. Almost all of the Arduino pins were used, mostly by the touchscreen. One of the interesting things we had to deal with were the limitations in which pins we could use for different tasks. The motor breakout board uses A4 and A5 for it's functionality, and pins 0 and 1 are used during serial communication (which we needed during debugging). The touchscreen could also double up on pins by switching an arduino pin from output to input.

We also decided to use a linear regulator to get our 5V supply in order to eliminate the need for a computer. This allowed for one input power line of 12V which could run the motors and the arduino.

3.2.1 Light Sensor

One problem that we had with our original implementation was that the photoresistor was more sensitive to movement on the outside than it was from the contents of the petri dishes. To try to correct for this we added an LED underneath the petri dishes, which illuminates them. We also added a cover on top of the system to try and mitigate the amount of outside noise the photoresistor has to deal with.

3.3 Code

All of the code used is on the Arduino, at around 22Kb of memory. One of the goals we set for ourselves was to make the system completely computer independent and reliant on one source of 12V power. There were two main components to the code which were the touch screen and the movement control/position tracking.

3.3.1 Touch Screen

One of the fun additions we added to this lab is an LCD/touchscreen combo. The reasoning behind this addition is that we had one available and it seemed like fun to try and get it working. Actually programming the screen is not very difficult, thankfully, due to adafruit's libraries. We also wanted to keep everything as intuitive as possible and easy to use.

Figure 4: An image of what the screen displays. The larger numbers in the circles are the petri dish locations, and the smaller numbers underneath are the readings at those locations. Petri dish 2 is currently selected as seen by the little drop down.

3.3.2 Movement and Position Tracking

We controlled the motor using the Adafruit motor controller library. This library provides simple controls for forward and reverse as well as the speed of the motor. These were the only controls we needed to implement on the testing rig.

We tracked position using a binary system with the reflective sensor. There are white strips of paper at each petri dish location, which read a low value to the reflective sensor while all other locations read a high value. We assigned each petri dish with a position number from 0 to 3. When the reflective Sensor output falls below 250, the code increments a counter for the current position. This counter is modulated by 4 to ensure the value never goes out of range. When the current position matches the desired position, the motor stops moving.

3.3.3 How it comes together

The main loop of our code does one thing, which is checking to see if the touchscreen is pressed. After it detects something, it decides what was pressed and goes through a drawing animation to give visual feedback (as seen in the video) as well as moving the petri dishes to the correct position and taking a measurement.

Manual mode is incredibly intuitive to use, where selecting a petri dish will rotate it under the opacity reader and display a measurement on the display. Switching to another petri dish does the

same thing, however the previous measurement is not deleted and remains displayed. Going back to an already read petri dish rewrites the measurement to an updated value.

Automatic mode is really similar to manual mode, the only difference is that it starts at the first petri dish and then iterates through all of them one by one as if a person pressed the touchscreen. This mode is much faster than the normal mode, and only has a brief pause on each petri dish to make a measurement.

4 Appendix: Source Code

```
#include <Adafruit_GFX.h>
#include <Adafruit_TFTLCD.h>
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include <TouchScreen.h>

// Pin settings for Touchscreen
#define LCD_CS A3
#define LCD_CD A2
#define LCD_WR 10
#define LCD_RD 11
#define LCD_RESET A4
#define YP A3
#define XM A2
#define YM 9
#define XP 8

// Pin settings for sensors
#define REFLECT A1
#define OPACITY A0

// Touchscreen min/max x/y coordinates
#define TS_MINX 150
#define TS_MINY 120
#define TS_MAXX 920
#define TS_MAXY 940

// Colors! er...
#define BLACK         0x0000
#define WHITE         0xFFFF

// Defining lengths
#define BOX_SHORT  176
#define BOX_LONG   216
#define BOX_DIFF   40
#define NUMB_H     30
#define MEASURE_H  55
#define CIRCLE_H   60

// Touchpad pressure min/max
#define MINPRESSURE 1
#define MAXPRESSURE 1000
```

```
// Where to sense for "Auto" selection
#define Y_LOW     110
#define Y_HIGH    0

// Setting up touchscreen, LCD, and motorshield
TouchScreen ts = TouchScreen(XP,YP,XM,YM, 300);
Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// Getting motor object
Adafruit_DCMotor *myMotor = AFMS.getMotor(1);

// Which option is currently selected, 5 is NULL
uint8_t selector = 5;

// Awesome function which turns RGB color
// values into 16 bit values for the LCD
uint16_t getColor(uint8_t red, uint8_t green, uint8_t blue)
{
  red   >>= 3;
  green >>= 2;
  blue  >>= 3;
  return (red << 11) | (green << 5) | blue;
}


void setup(){
    Serial.begin(9600);

    // Clear screen
    tft.reset();

    uint16_t identifier = tft.readID();
    tft.begin(identifier);

    // Begins drawing the screen
    tft.setRotation(1);
    tft.fillScreen(WHITE);
    tft.setTextColor(WHITE); tft.setTextSize(8);

    // 1
    tft.fillRect(23, 0, 80, BOX_SHORT, getColor(255,153,85));
    tft.fillCircle(63,CIRCLE_H,55, getColor(128,0,0));
    tft.setCursor(45,NUMB_H);
    tft.println("1");

    // 2
    tft.fillRect(141, 0, 80, BOX_SHORT, getColor(135,222,205));
    tft.fillCircle(181, CIRCLE_H,55, getColor(0,170,136));
    tft.setCursor(163,NUMB_H);
    tft.println("2");

    // 3
    tft.fillRect(259, 0, 80, BOX_SHORT, getColor(141,211,95));
```

```
    tft.fillCircle(299, CIRCLE_H, 55, getColor(33,120,33));
    tft.setCursor(281,NUMB_H);
    tft.println("3");

    // 4
    tft.fillRect(377, 0, 80, BOX_SHORT, getColor(153,85,255));
    tft.fillCircle(417, CIRCLE_H, 55, getColor(68,0,85));
    tft.setCursor(393,NUMB_H);
    tft.println("4");

    // AUTO
    tft.fillRect(0,230, 480, 75, BLACK);
    tft.setCursor(160, 240);
    tft.println("AUTO");

    // Set up the Motor
    AFMS.begin();
    myMotor->run(RELEASE);

    // Setup sensor pins
    pinMode(REFLECT, INPUT);
    pinMode(OPACITY, INPUT);
}


void loop(void){
    TSPoint p = ts.getPoint();
    if (p.z > MINPRESSURE && p.z < MAXPRESSURE) {
        pinMode(XM, OUTPUT);
        pinMode(YP, OUTPUT);

        // Screen is rotated, so remap X & Y coordinates
        uint16_t y = map(p.x,TS_MINX,TS_MAXX,320,0);
        uint16_t x = map(p.y,TS_MINY,TS_MAXY,0,480);

        //Manual mode is selected
        if ( y < Y_LOW && y > Y_HIGH ){
            if (selector == 4) selector = 3;
            if ( x > 0 && x < 120 && selector != 0 ){
                // General idea here is to:
                Screen_Select(0);    // update the screen
                getGet(selector, 0); // move to position
                selector = 0;        // update current position
                TakeMeasure(selector); // take measurement
            }
            if ( x > 120 && x < 240 && selector != 1){
                Screen_Select(1);
                getGet(selector, 1);
                selector = 1;
                TakeMeasure(selector);
            }
            if (x > 240 && x < 360 && selector != 2){
                Screen_Select(2);
                getGet(selector, 2);
```

```
            selector = 2;
            TakeMeasure(selector);
        }
        if ( x > 360 && x < 480 && selector != 3){
            Screen_Select(3);
            getGet(selector, 3);
            selector = 3;
            TakeMeasure(selector);
        }
    }

    // Auto mode is selected
    if ( y > 230 && selector != 4 ){
        Screen_Select(4);
        getGet(selector, 0);
        TakeMeasure(0);
        getGet(0, 1);
        TakeMeasure(1);
        getGet(1, 2);
        TakeMeasure(2);
        getGet(2, 3);
        TakeMeasure(3);
        selector = 4;
    }
  }
}


// Change what the screen displays in terms of
// which option is currently selected
void Screen_Select(uint8_t select){

    // Clear previous selection without redrawing everything
    tft.fillRect(0, BOX_SHORT, 480, BOX_DIFF, WHITE);
    tft.setTextSize(8); tft.setTextColor(WHITE);
    tft.setCursor(160,240); tft.println("AUTO");

    switch(select){
        case 0:
            tft.fillRect(23, BOX_SHORT, 80, BOX_DIFF, getColor(255,153,85));
            break;
        case 1:
            tft.fillRect(141, BOX_SHORT, 80, BOX_DIFF, getColor(135,222,205));
            break;
        case 2:
            tft.fillRect(259, BOX_SHORT, 80, BOX_DIFF, getColor(141,211,95));
            break;
        case 3:
            tft.fillRect(377, BOX_SHORT, 80, BOX_DIFF, getColor(153,85,255));
            break;
        case 4:
            tft.setTextSize(8); tft.setTextColor(getColor(255,255,0));
            tft.setCursor(160, 240);
            tft.println("AUTO");
```

```
            break;
        default: break;
    }
}

void TakeMeasure( uint8_t select ){
    tft.setTextSize(4); tft.setTextColor(WHITE);

    // This is necessary to avoid drawing over itself
    // Essentially erases previous writes
    switch(select){
        case 0: tft.setCursor(30,NUMB_H + 100);
            tft.fillRect(23, BOX_SHORT-BOX_DIFF-10, 80, BOX_DIFF, getColor(255,153,85));
            break;
        case 1: tft.setCursor(145,NUMB_H + 100);
            tft.fillRect(141, BOX_SHORT-BOX_DIFF-10, 80, BOX_DIFF, getColor(135,222,205));
            break;
        case 2: tft.setCursor(265,NUMB_H + 100);
            tft.fillRect(259, BOX_SHORT-BOX_DIFF-10, 80, BOX_DIFF, getColor(141,211,95));
            break;
        case 3: tft.setCursor(380,NUMB_H + 100);
            tft.fillRect(377, BOX_SHORT-BOX_DIFF-10, 80, BOX_DIFF, getColor(153,85,255));
            break;
    }
    tft.println(analogRead(OPACITY));
}


// Out motor control function
// Goes to position desired
void getGet(int pos, int desired){
  boolean found = false;
  int reflective;
  int refPrev = 0;
  int threshold = 200;
  int direc = 0;

  while (found == false){

    // Check to see if it is quicker to go reverse
    // one petri rather than forward 3
    if(pos - desired == -3 || pos - desired == 1){
      direc = -1;
      moveMotor(-1); //reverse
    }
    else if (pos == desired){
      moveMotor(0); //stop
      found = true;
    }
    else{
      direc = 1;
      moveMotor(1); //forward
    }
```

```
    //check if we're at the next position
    reflective = analogRead(REFLECT);

    // Make sure to not update itself on the same
    // white patch, otherwise update position
    if (reflective <= threshold && refPrev > threshold){
      pos += direc;
      pos = pos%4;
      refPrev = reflective;
    }
    else{
      refPrev = reflective;
    }
  }
}


// Making life easier via functions
void moveMotor(int state){
  int motorSpeed = 200;
  if (state == 0){
    myMotor->setSpeed(0);
  }

  if (state == 1){
    myMotor->run(FORWARD);
    myMotor->setSpeed(motorSpeed);
    delay(10);
  }

  if (state == -1){
    myMotor->run(BACKWARD);
    myMotor->setSpeed(motorSpeed);
    delay(10);
  }
}
```