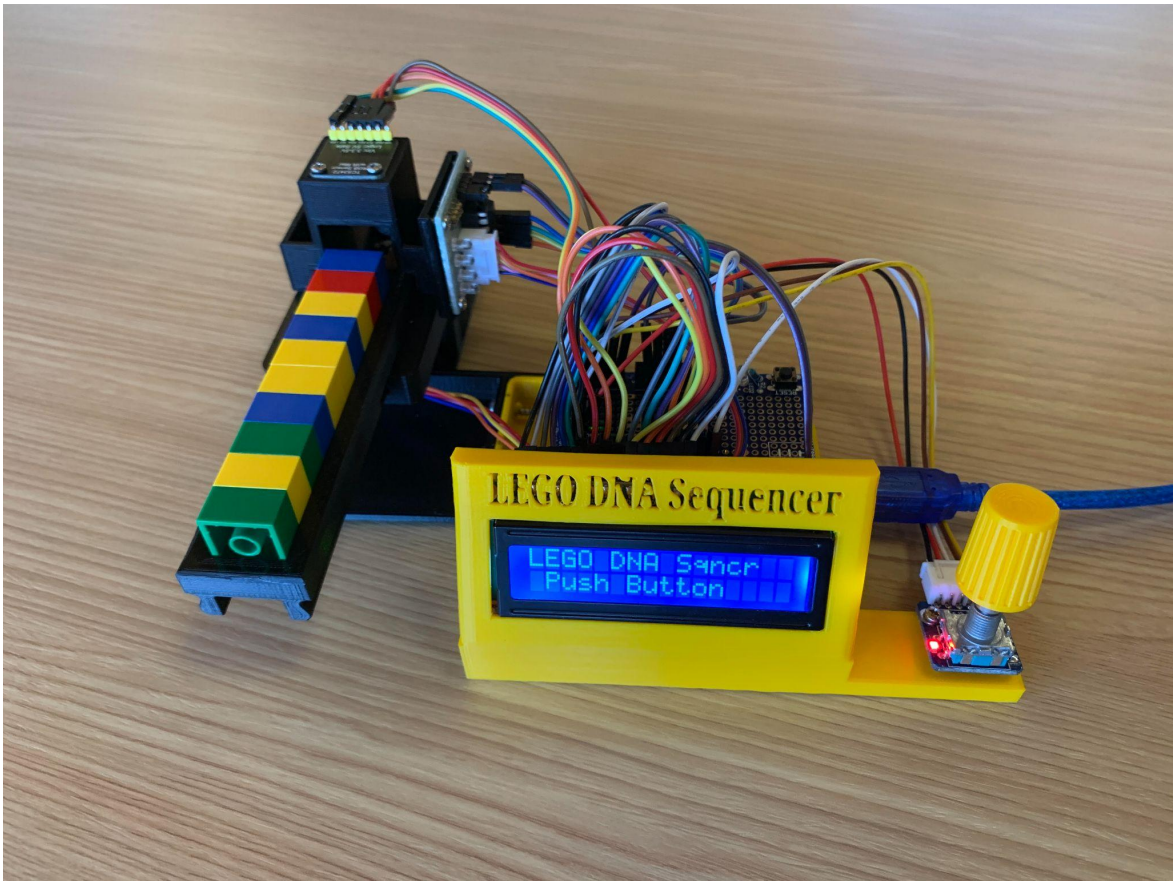
LEGO DNA Sequencer

and DIY Construction

2023-01-16 Rev 0.4

The purpose of this LEGO DNA Sequencer and DIY Construction is to provide a complete blueprint for an open source tool that can be used to explain the complex DNA sequencing operation with a colorful easy to understand model.



Tom Rolander, MSEE
Mentor for Circuit Design, Software, 3D Printing
Miller Library, Fabrication Lab
Hopkins Marine Station, Stanford University,
120 Ocean View Blvd, Pacific Grove, CA 93950
+1 831.915.9526 | rolander@stanford.edu

Table of Contents

Overview

The purpose of this document is to provide all of the necessary information to build your own LEGO DNA Sequencer. The electronics assembly, 3D printed parts, software, and assembly steps are described in detail. Also a complete Users Manual is provided describing the operation of the LEGO DNA Sequencer.

DNA sequencing determines the order of the four chemical building blocks - called "bases" - that make up the DNA molecule. The building blocks are composed of A (adenine), C (cytosine), G (guanine), and T (thymine). These building blocks are represented by LEGO® 2x2 bricks of **Red for G**, **Blue for T**, **Green for A**, and **Yellow for C**. Actual DNA sequencing is greatly simplified in this model by using genomes of only 10 building blocks to identify a species.

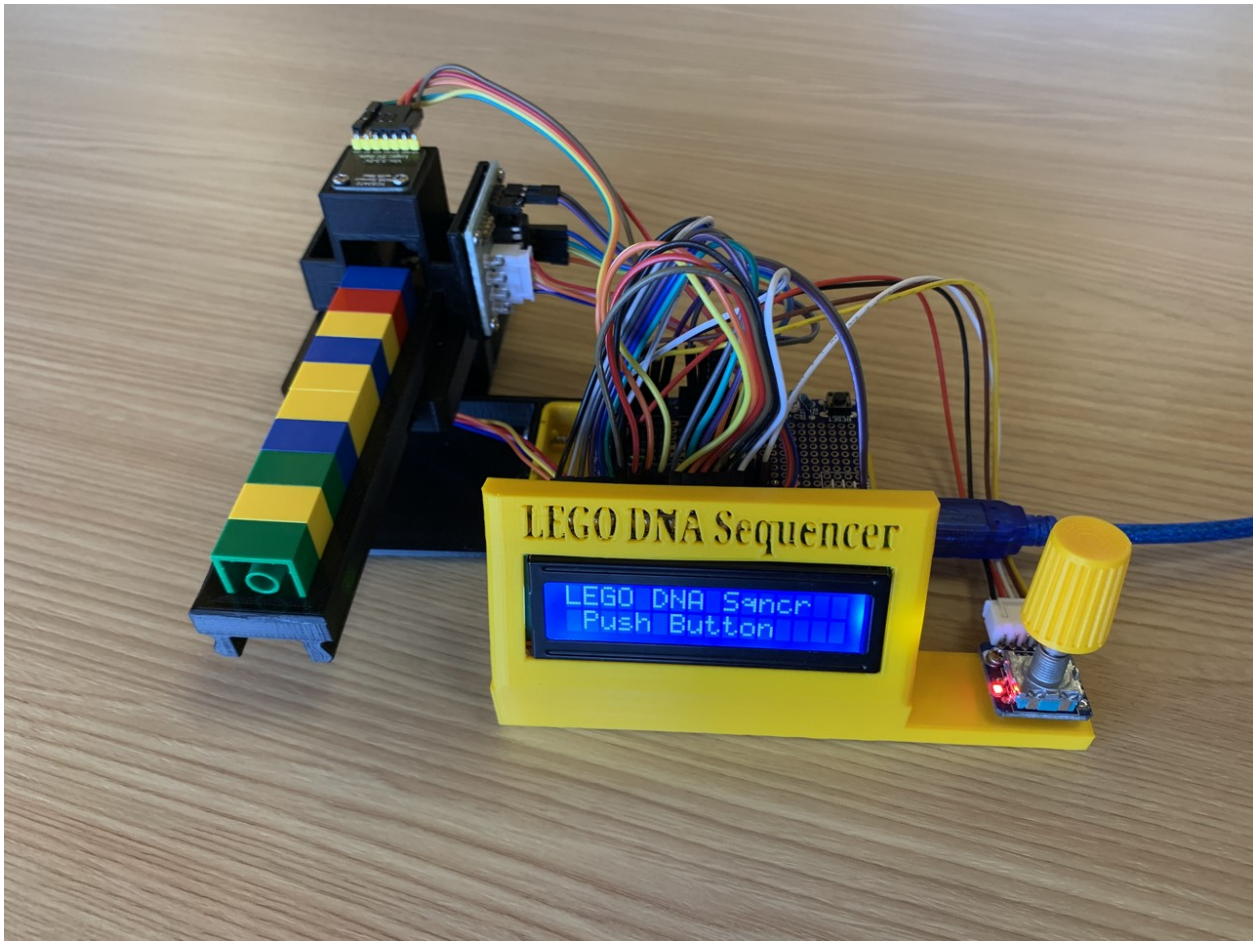
It is beyond the scope of this document to provide more details regarding actual DNA sequencing.

Users Manual

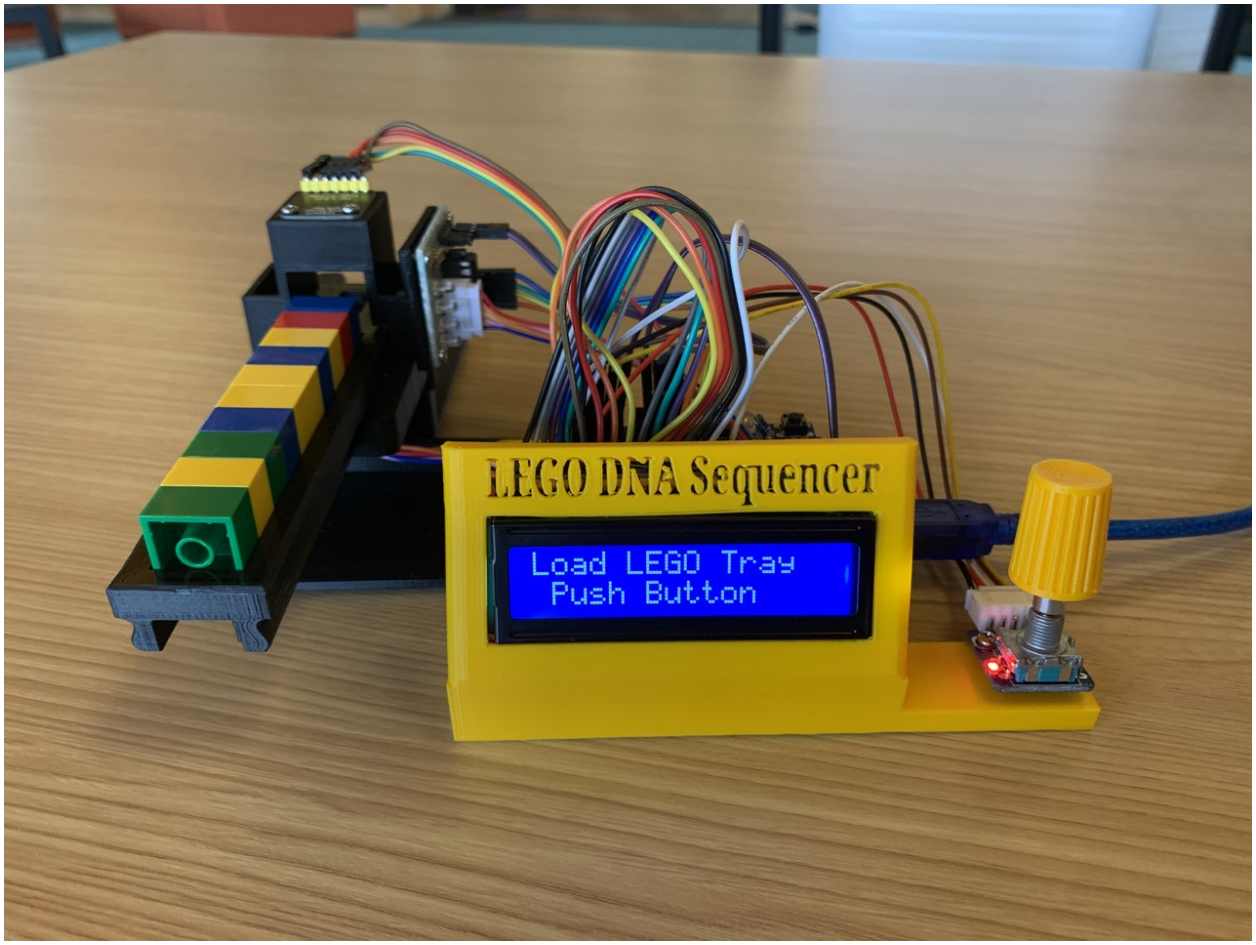
The LEGO DNA Sequencer is operated with a simple user interface. The UI includes an LCD display, a buzzer, and a rotary encoder / button which is much like a car radio knob.

Starting the System

Power on the system by connecting the USB cable from the Arduino into a USB power adapter. There is no ON/OFF switch. When the system is plugged in it will beep three times and display the following screen.



Push the button and you will be ready to perform DNA sequencing.



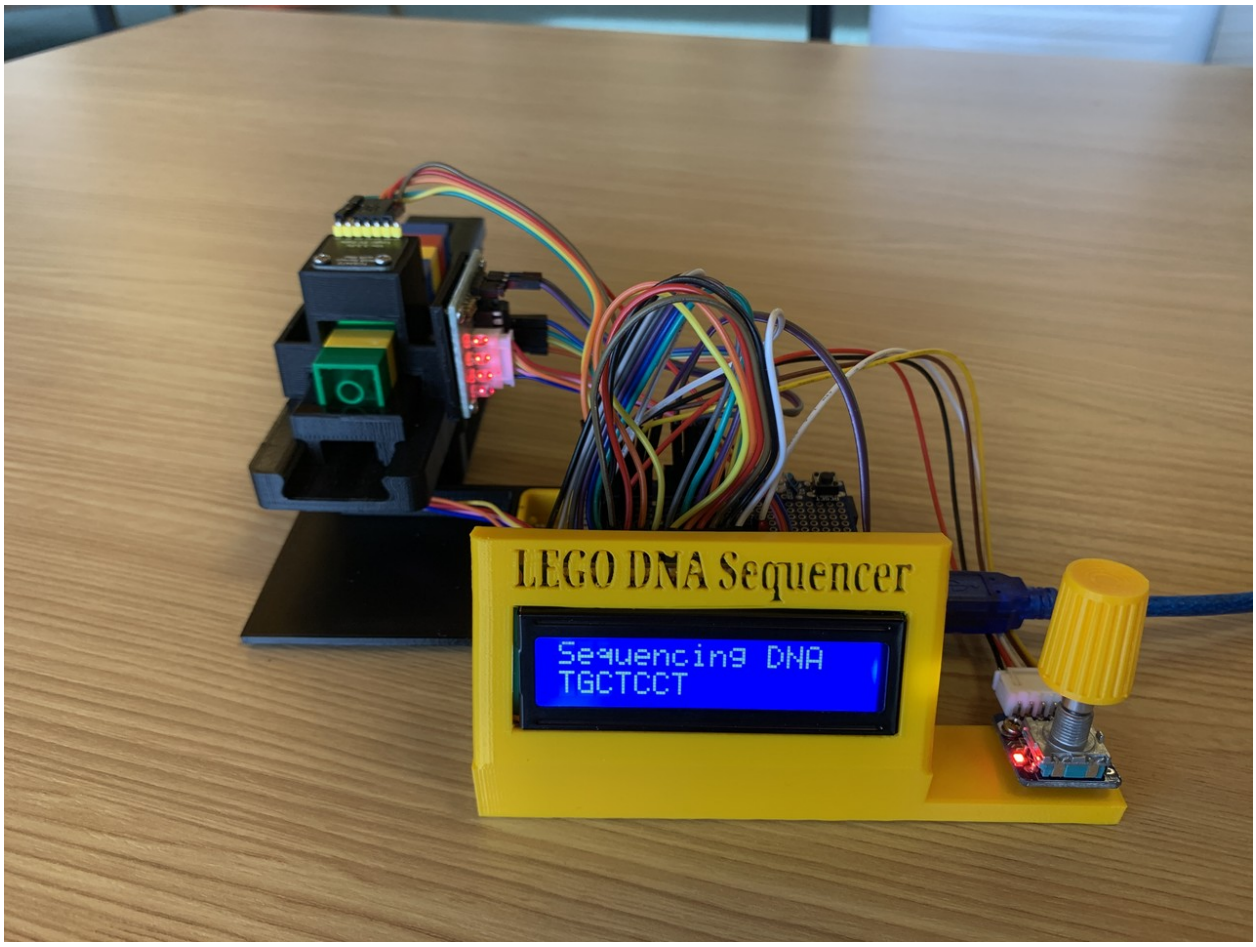
To operate the sequencer you will need to assemble 10 of the LEGO 2x2 bricks into a genome for sequencing. The system has 6 built in genomes that it will recognize as shown in the following illustration.

Loading the LEGO Tray

When you have assembled a genome of 10 LEGO bricks, place them in the slot in the tray with the first brick in the sequence at the forward end of the tray. Note that the first brick will be inside the color sensor detection unit. The beginning of the second brick should align with the face of the color sensor.

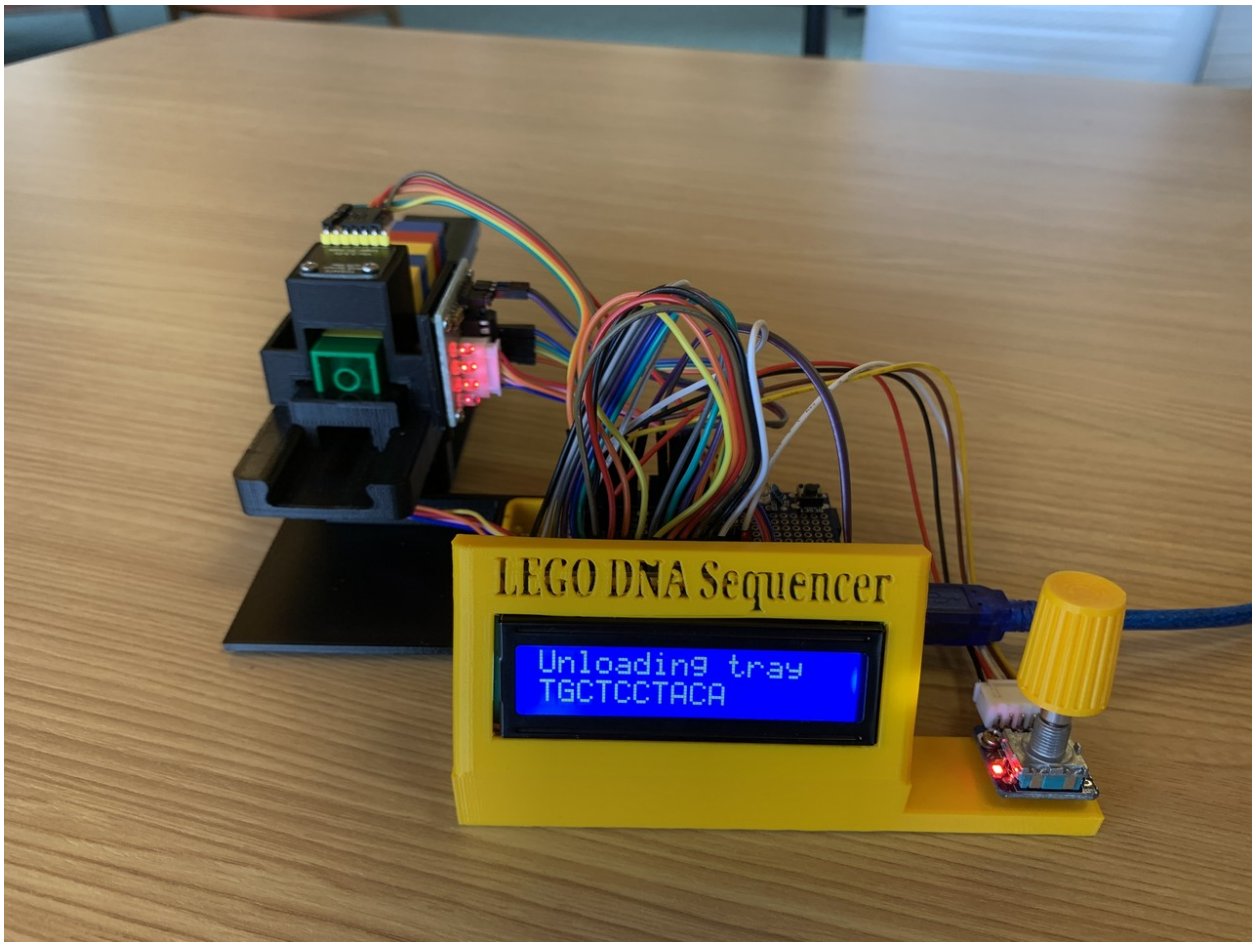
Beginning the Sequencing

Push the button to begin sequencing. You will hear a buzzer click with each brick that is sequenced and the appropriate A, C, G, or T will be displayed.

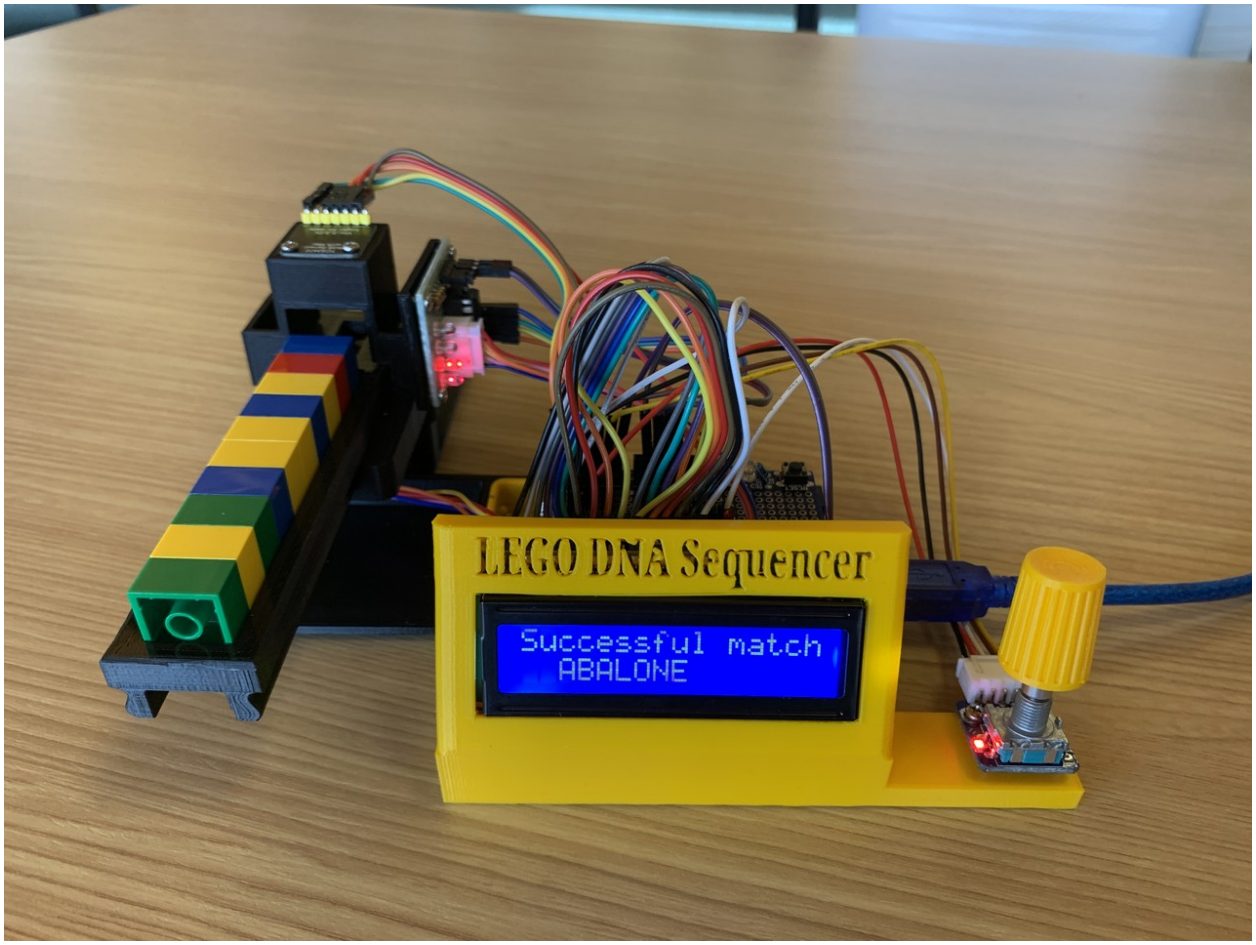


Unloading the LEGO Tray

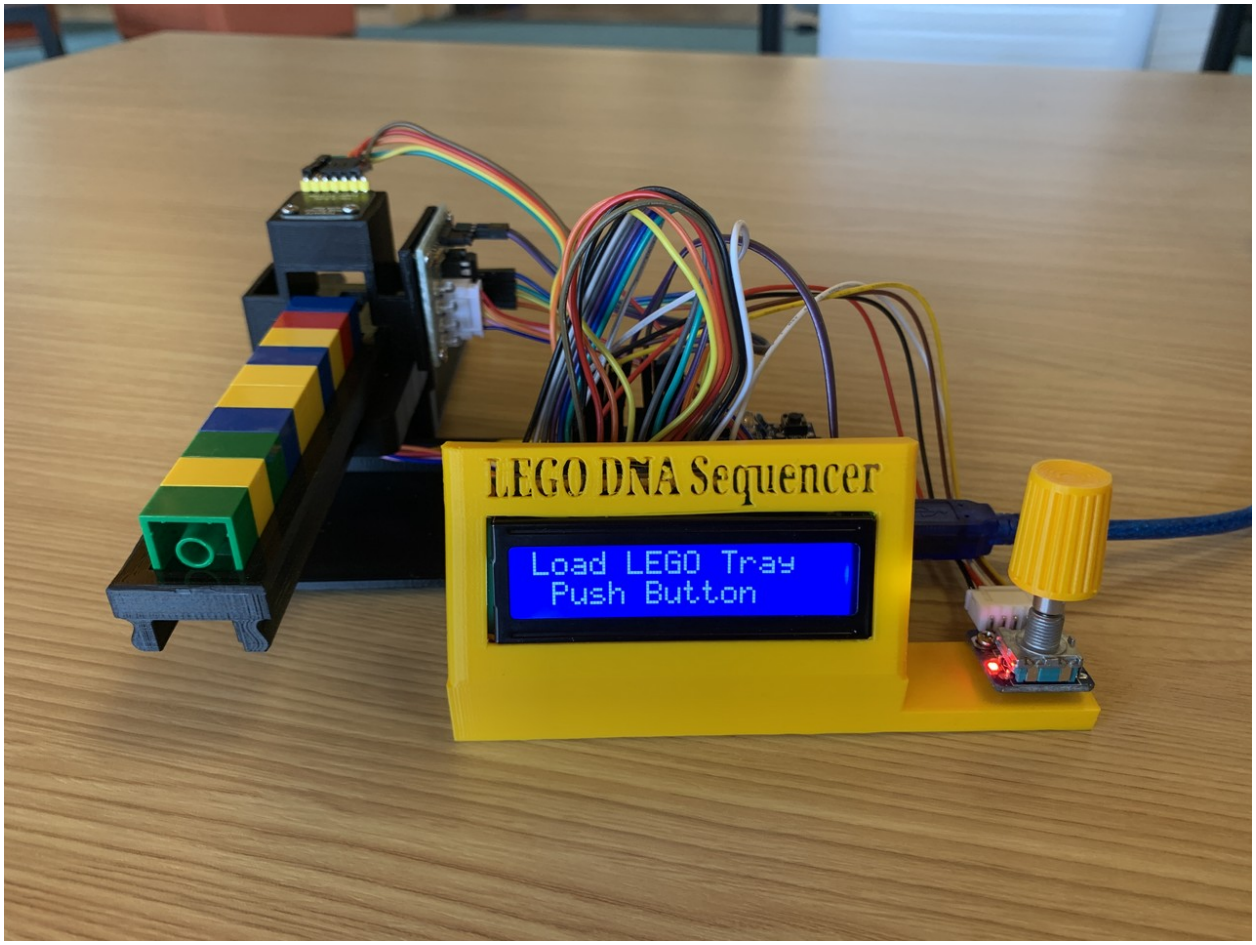
When all 10 of the bricks have been sequenced the LEGO tray will be unloaded, returning the tray to the initial position.



At that point you should see a “Successful match” and the name of the species that has been sequenced.

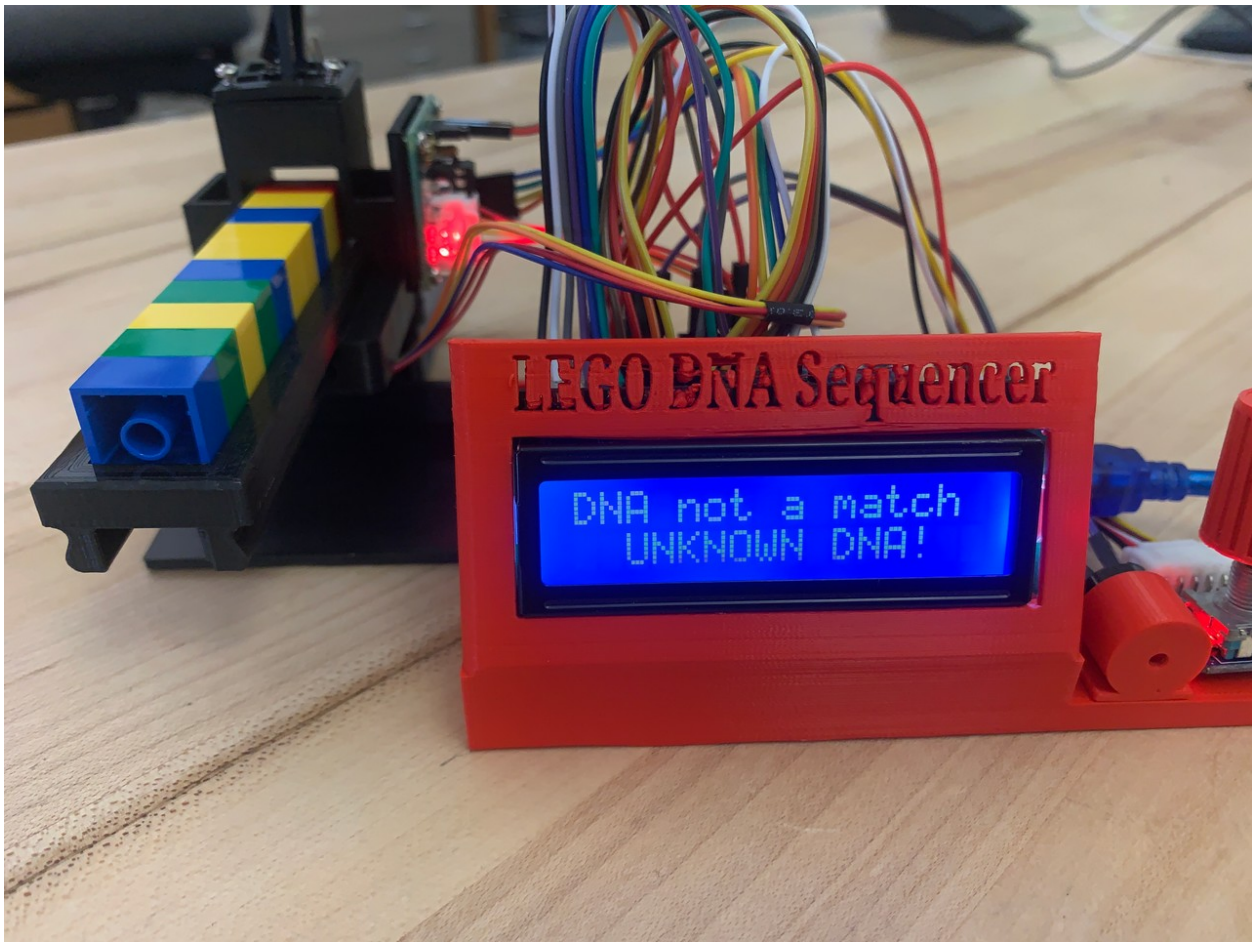


Pushing the button will ready the system to perform another sequencing operation.



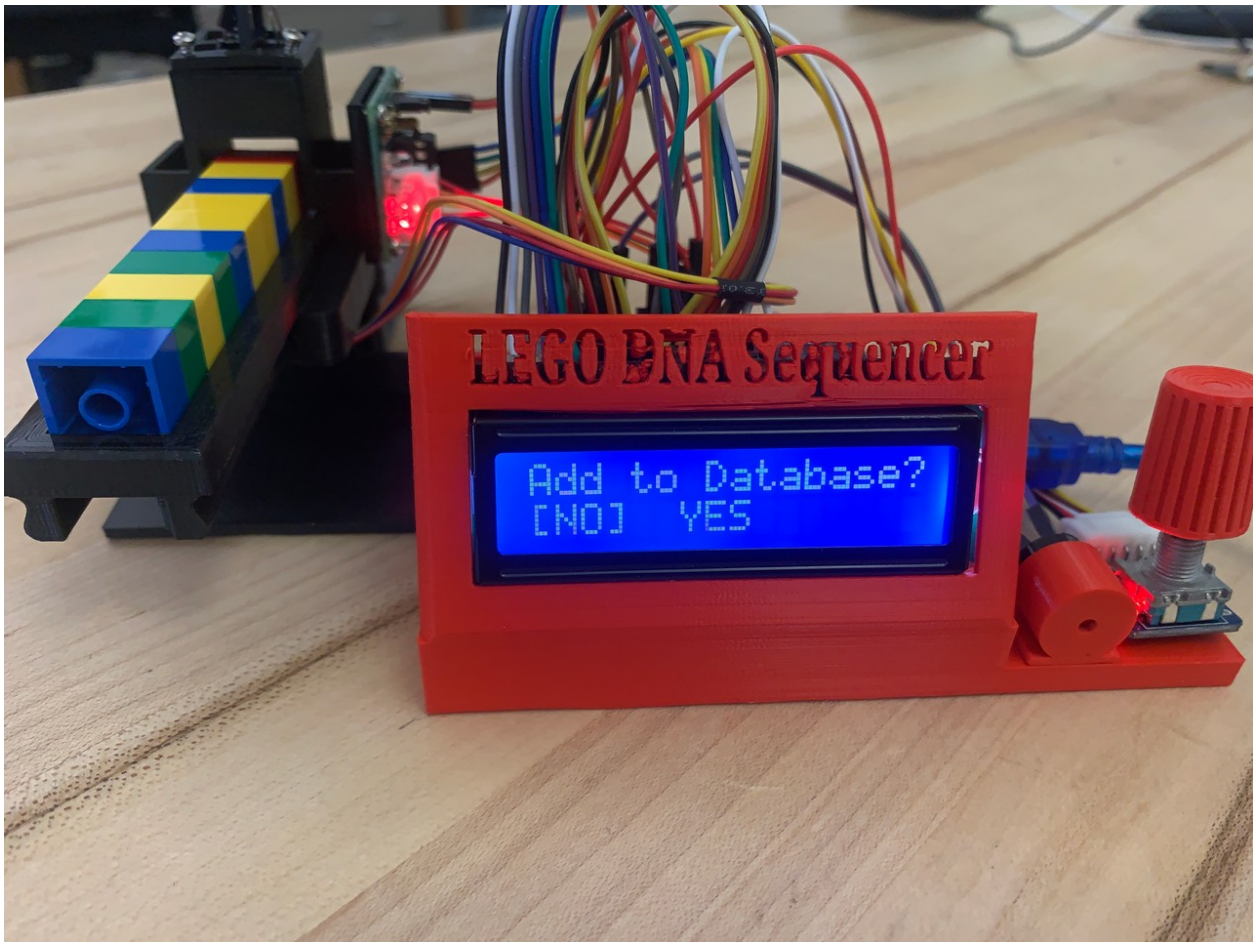
UNKNOWN DNA

In the event that the bricks you have assembled do not match those of a known genome the system will display the following message.

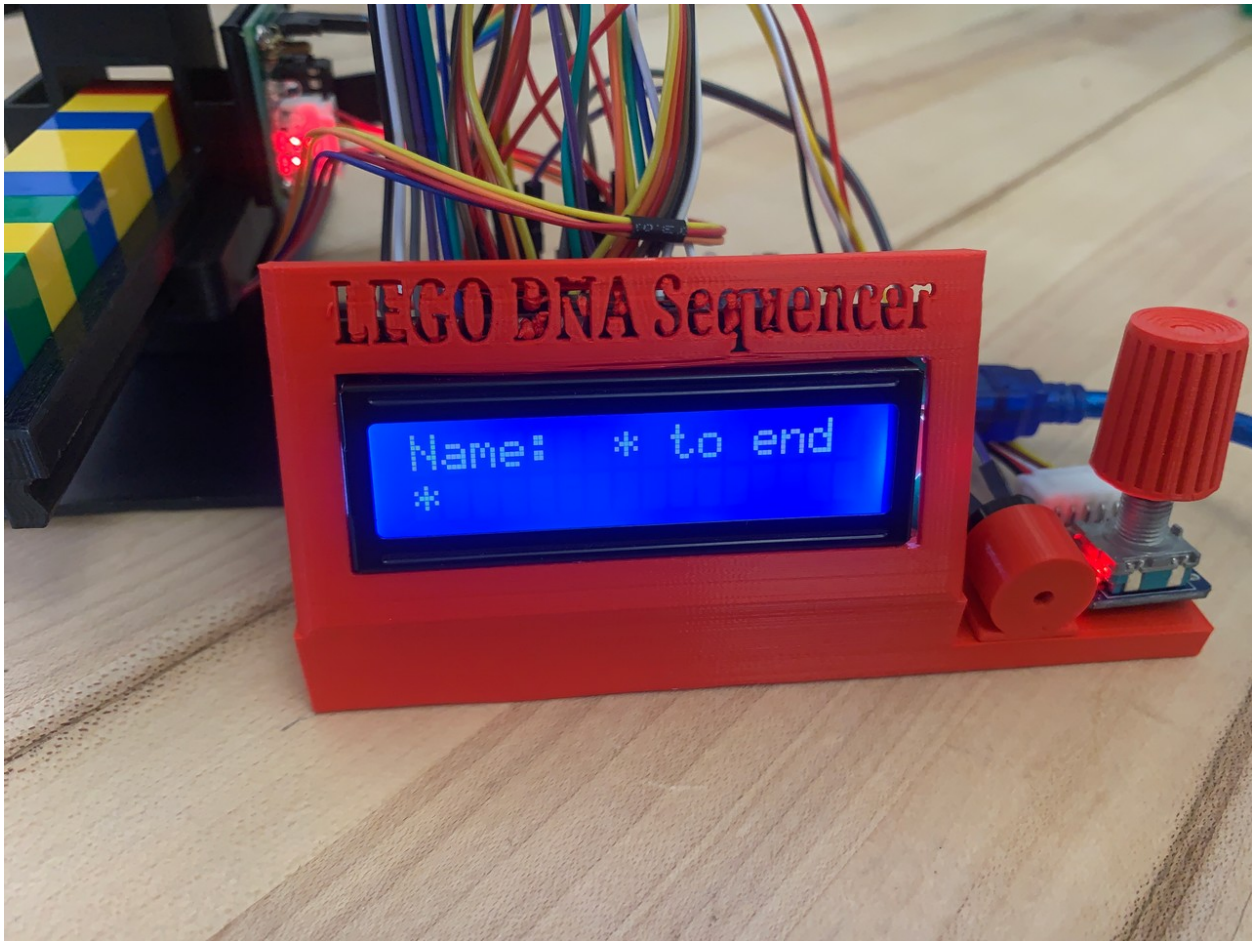


Updating the Database with a new DNA

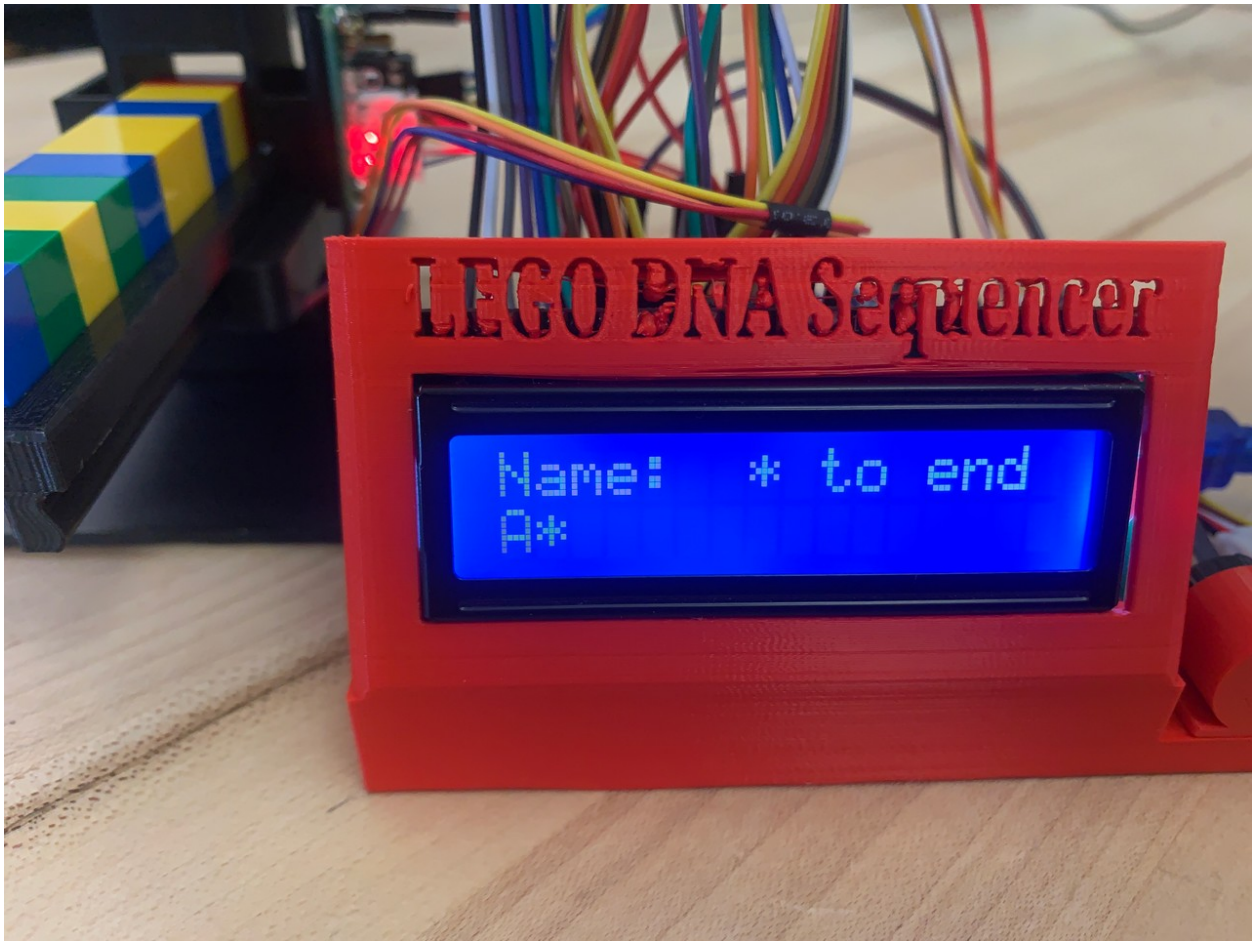
Push the button and you will be asked if you want to add the sequence to the database of known species genomes. Rotating the knob will move the selection back and forth between the choices of NO and YES. If you select NO, which is the default selection, the system will return to the state where it is ready to start another sequence operation.



If you selected YES to add the new sequence to the database, you will be prompted to enter a Name for the new genome. Rotating the knob will show you the characters that you can choose for the next letter of the name: A-Z, 0-9, and ‘ ‘.

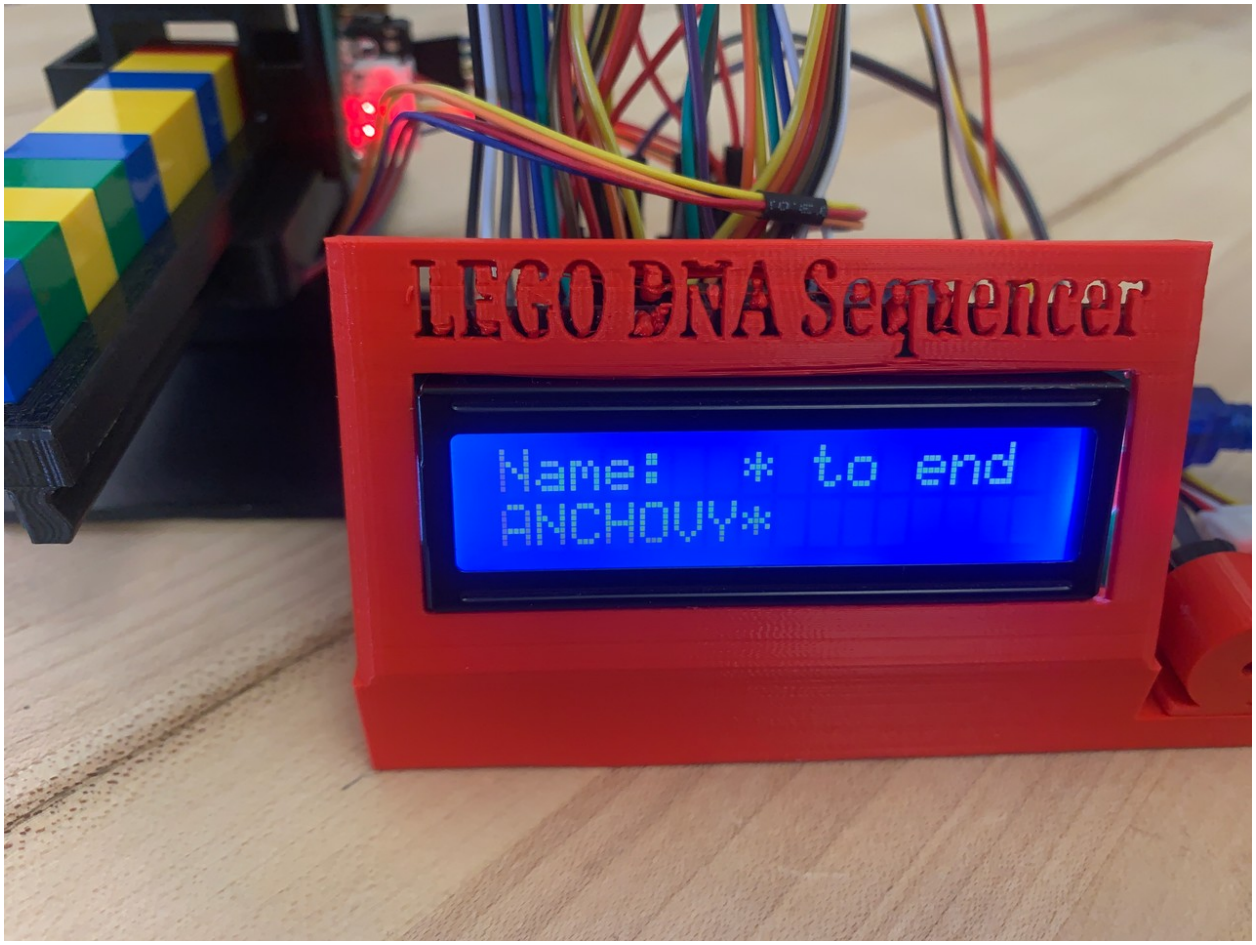


To select a character for the name, press the button and the display will advance for you to enter the next character.



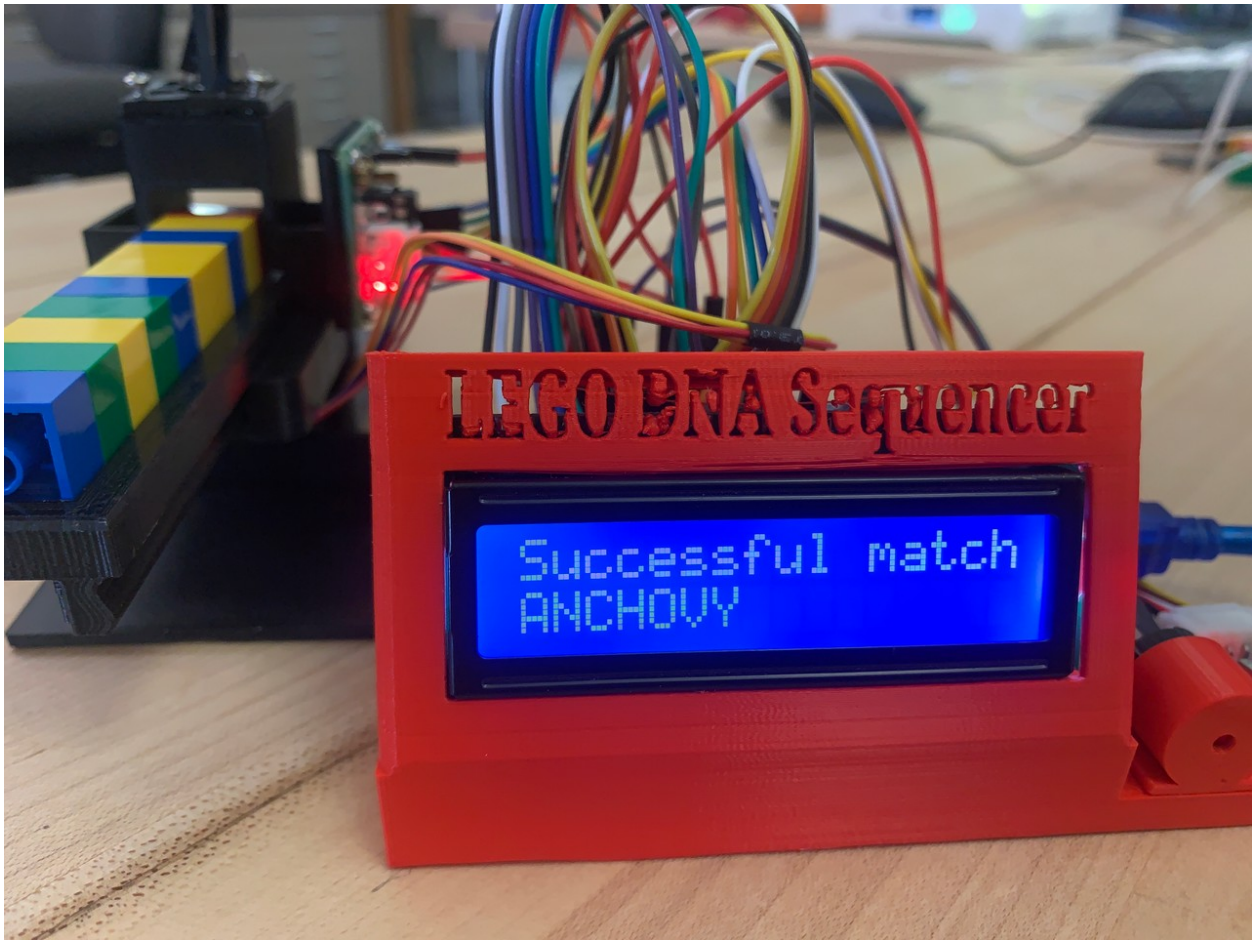
If you wish to delete a previous character, select the ‘<’ character and push the button.

When you have completed entering the name, select the ‘*’ character.



You will then be prompted NO or YES to accept the database entry.

You should then be able sequence the same tray of LEGO bricks and see that your new genome is successfully matched.


LEGO DNA Sequencer
Successful match
ANCHOVY

Troubleshooting

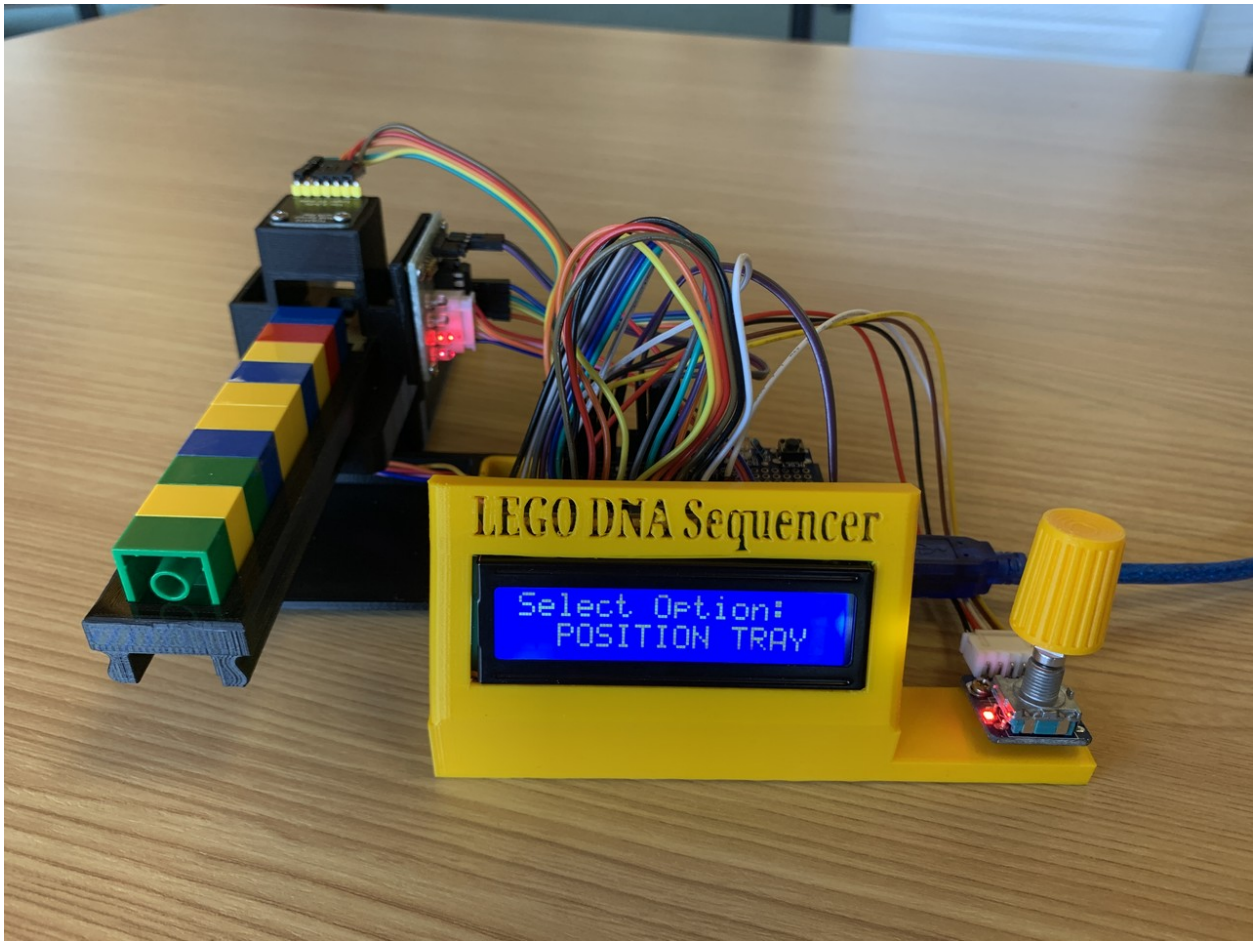
In the event that the sequencer is not correctly identifying a known genome sequence it will be necessary to make some adjustments to either the positioning of the LEGO tray and/or adjustments to the color channel threshold levels.

Adjustment of the LEGO Tray Position

The “home” initial position of the LEGO tray can be manually adjusted with the maintenance User Interface. NOTE: DO NOT MANUALLY FORCE THE TRAY!

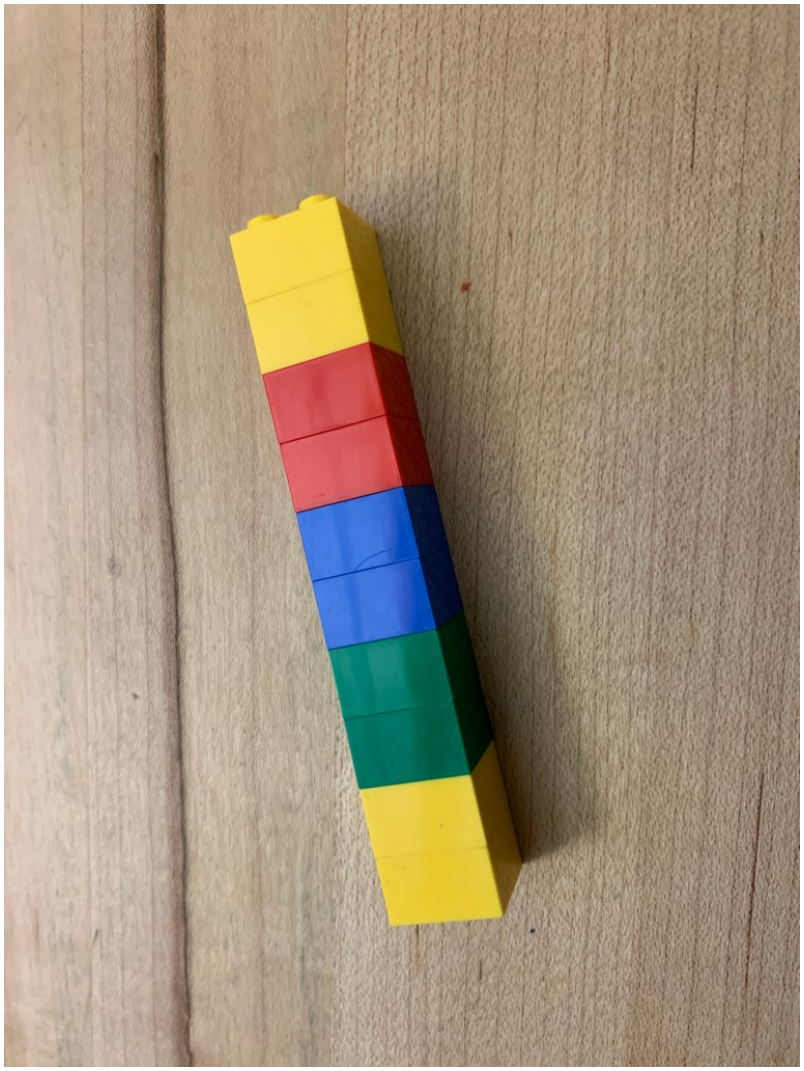
The maintenance User Interface is entered by rotating the knob when the system is in the state where it is ready to begin sequencing with the display showing, “Load LEGO Tray / Push Button”. Each click of the rotating knob will advance to another option. Select “Cancel” to exit from maintenance mode.

When you push the button to select “POSITION TRAY”, rotating the knob clockwise will advance the tray forward and rotating the knob counterclockwise will move the tray backward. Push the button to finish when you have correctly positioned the tray so that the face of the 2nd brick aligns with the face of the color sensor. At that point you should re-sequence and verify the results of your operation. If this fails



Adjustments to the Color Channel Threshold Levels

The first step in adjusting the color channel threshold levels is to assemble and sequence the YYRRBBGGYY test bricks.



Sequencing this known order of bricks will enable the system to adjust the channel threshold levels used to identify the Red, Green, Blue, and Yellow colors detected by the color sensor.

Additional Operations

The Additional Operations are accessed by rotating the knob when the system is ready to begin sequencing with the display showing, “Load LEGO Tray / Push Button”.

CANCEL

Exit from additional operations and return to sequencing LEGO operations.

UNLOAD TRAY

The LEGO tray will be backed out, removing it from the system

LOAD TRAY

The LEGO tray should be inserted until it reaches the gear. This will position the tray about 1” from the face of the color sensor. Gently advance the tray forward when you push the “LOAD TRAY”. The tray should automatically move forward to the position where it should be ready to start sequencing.

POSITION TRAY

Rotating the knob clockwise will advance the tray forward and counterclockwise in reverse direction. Pushing the button will exit positioning the tray.

ZERO NEW DNA

This operation will remove any genomes that were added to the database. Note that there is limited storage for additional genomes.

BUZZER OFF

This will turn OFF the buzzer which is used to announce operations and identify progress during sequencing.

BUZZER ON

This will turn OFF the buzzer which is used to announce operations and identify progress during sequencing.

ADVANCED

This selection provides access to system operations.

Advanced Operations

The Advanced Operations include selections that adjust the color channel threshold levels, auto positioning the LEGO tray, displaying the LCD output on an external monitor, and rebooting the system.

CANCEL

Exit from advanced operations and return to sequencing LEGO operations.

CLEAR CHANNEL

This selection displays the current CLEAR channel threshold level and allows you to edit the value.

RED CHANNEL

This selection displays the current RED channel threshold level and allows you to edit the value.

BLUE CHANNEL

This selection displays the current BLUE channel threshold level and allows you to edit the value.

AUTO CHANNEL

This selection requires that you have scanned the LEGO test bricks YYRRBBGGYY. It will then display the current channel threshold values for CLEAR, RED, and BLUE. It will show the new recommended values and allow you to update the system with those values.

AUTO POSITION

The auto positioning is also accomplished using the LEGO test bricks YYRRBBGGYY. You should begin with the POSTION TRAY operation to get the tray near to the desired start position. This operation advances the tray to the two RED bricks and then backs up to the estimated start position.

REMOTE LCD ON

This selection enables the system serial output to show the contents of the LCD. It requires that the RemoteLCD.py program is operating on the laptop computer used to display the large font output. See the next chapter on Remote LCD.

REMOTE LCD OFF

This selection disables the system serial output to show the contents of the LCD.

REBOOT

This will immediately Reboot the system and return to the startup screen.

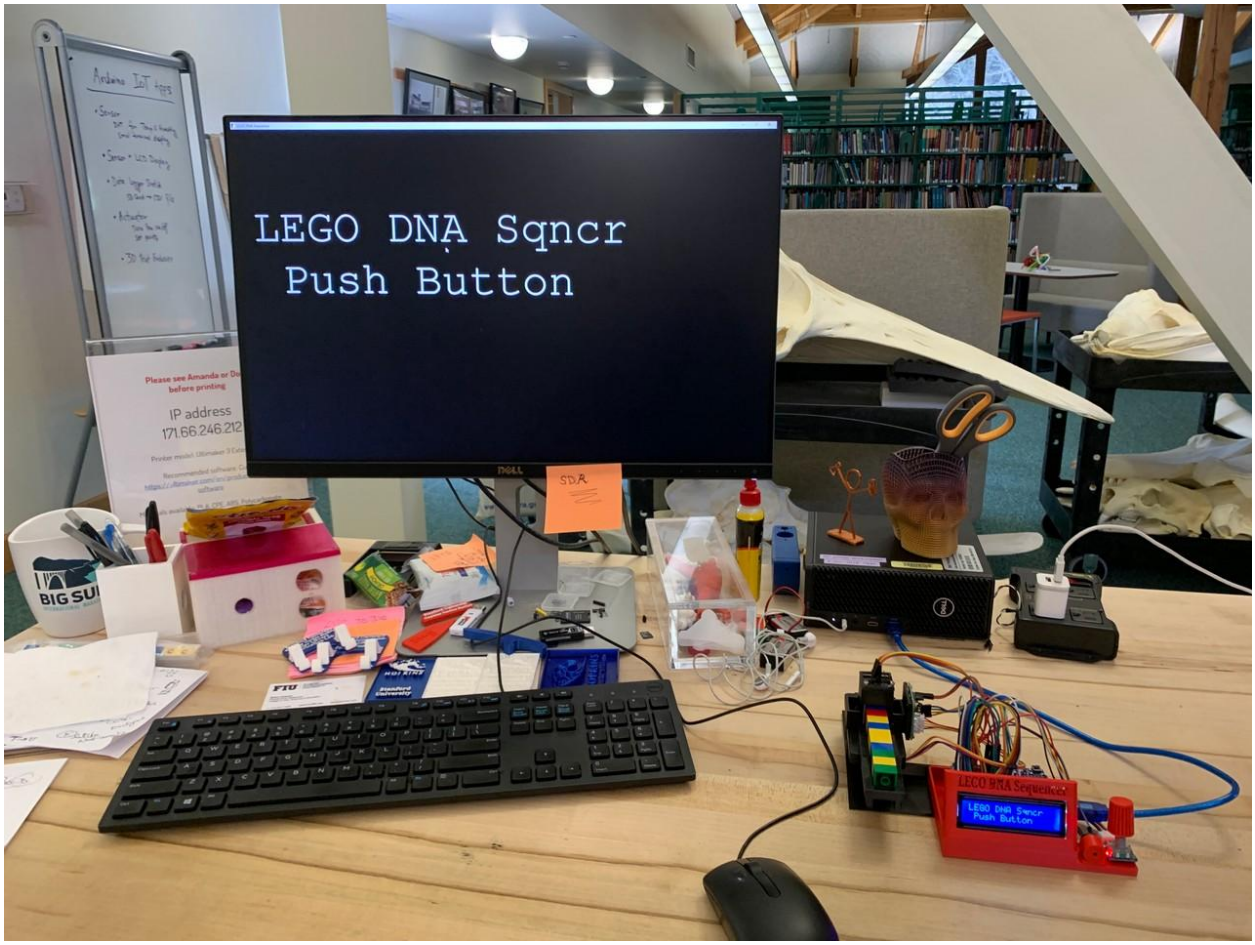
Remote LCD

The Remote LCD is an optional companion Python program which displays the LCD contents of the LEGO DNA Sequencer on a big screen for a larger viewing audience.

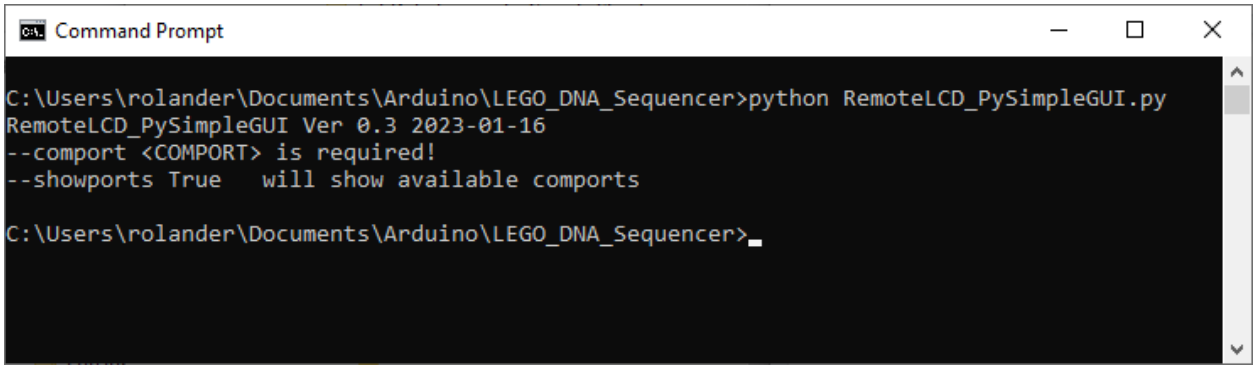
There are two Python versions of the Remote LCD program which require two different GUI libraries:

- PySimpleGUI
- Tkinter

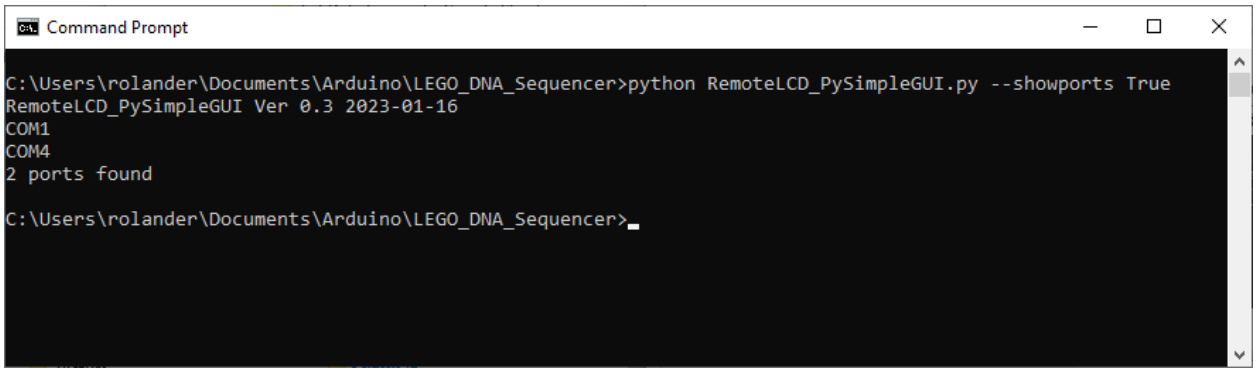
RemoteLCD_PySimpleGUI



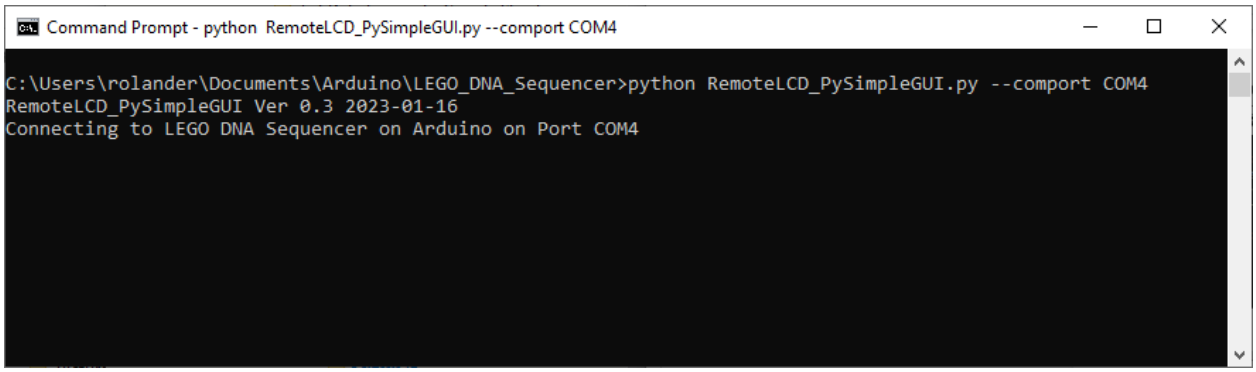
The application is launched with the following command on a Windows PC:



The available comports can be displayed using the following command:



The following command will begin operation of the Remote LCD display:



The Welcome screen will be displayed and the LEGO DNA Sequencer will reboot, showing its startup screen.

From that point on the contents of the LCD screen on the LEGO DNA Sequencer will be displayed on your computer screen.

To terminate the RemoteLCD program enter the following keys:

Alt - F4

The the following screens show the RemoteLCD output:


Sequencing DNA
TGCT
IP address
171.66.246.212
SDA
LEGO DNA Sequencer
Sequencing DNA
TGCT


Unloading tray
TGCTCCTACA
LEGO DNA Sequencer


Successful match
ABALONE
LEGO DNA Sequencer

Installation for Remote LCD for PySimpleGUI

The steps to run the Remote LCD Python program on your laptop computer are as follows:

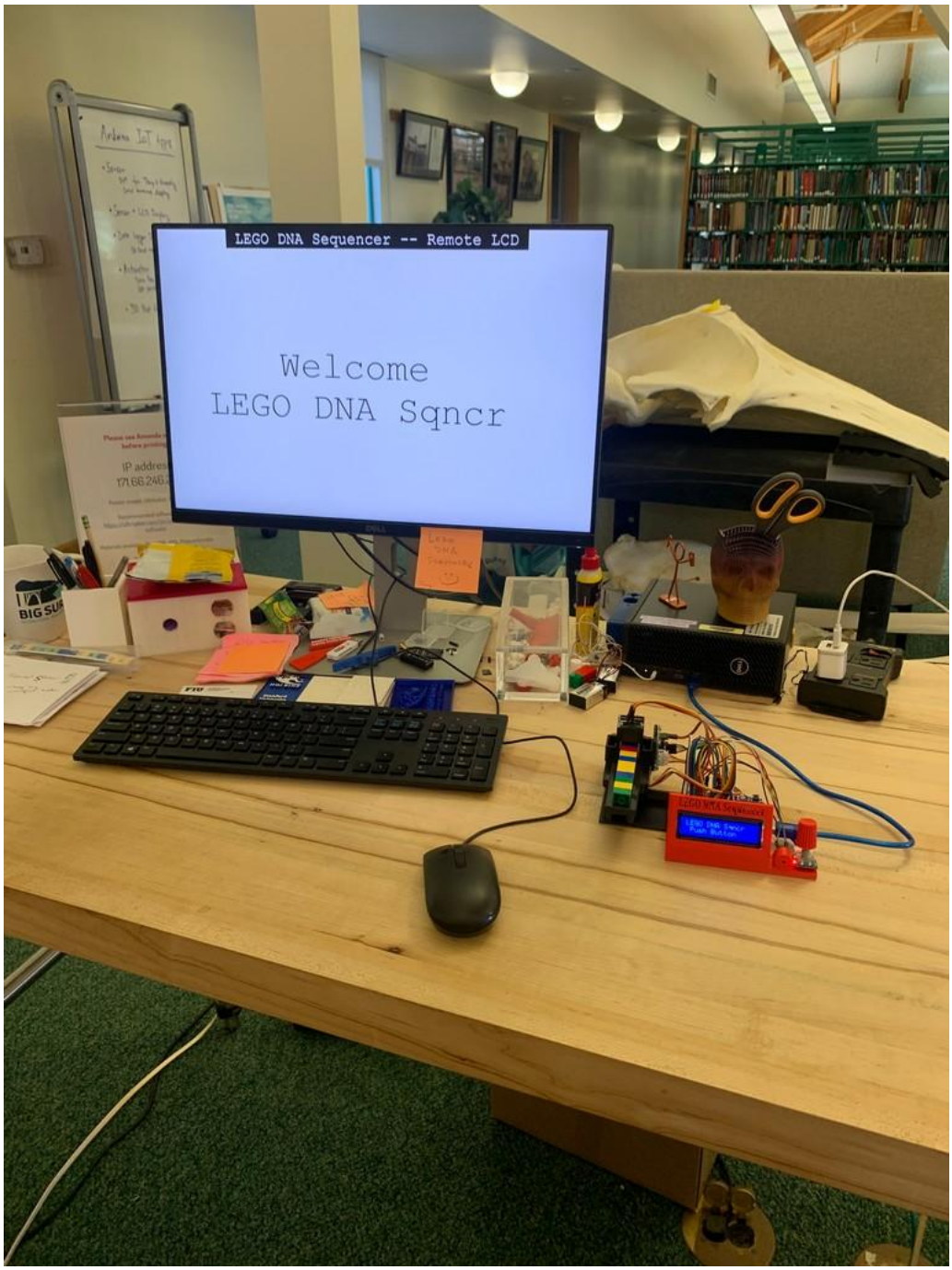
Windows

- Install python
- Install pip
- Install PySimpleGUI
- Install pyserial

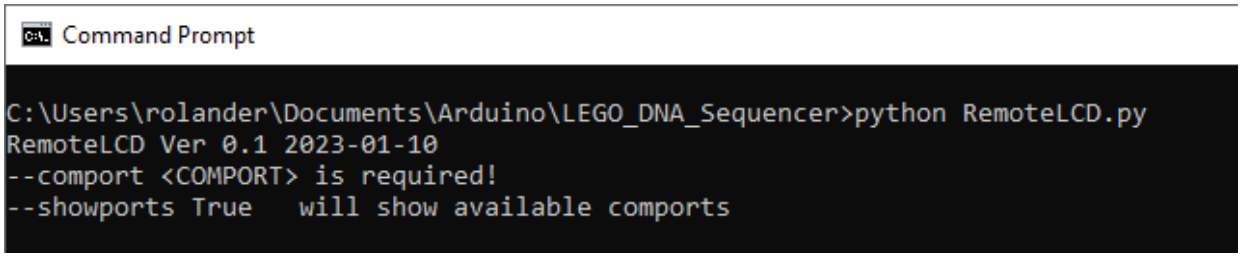
Mac

- Install python
- Install pip
- Install PySimpleGUI
- Install pyserial

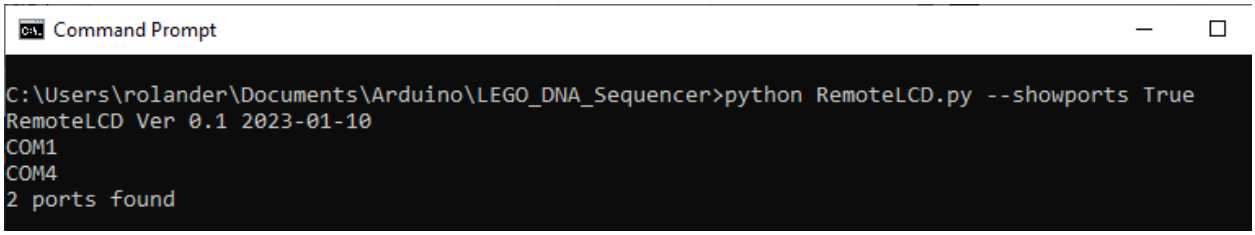
RemoteLCD_tkinter



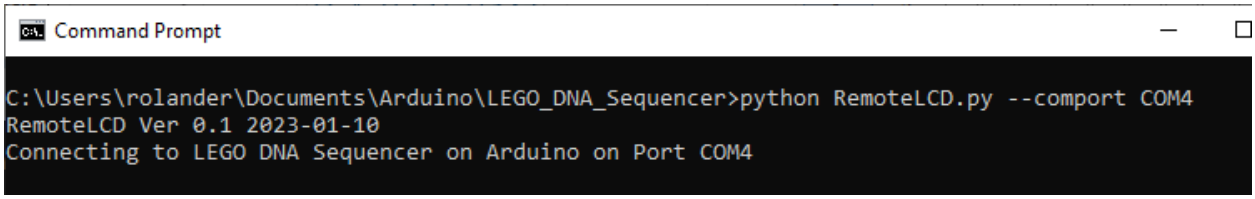
The application is launched with the following command on a Windows PC:



The available comports can be displayed using the following command:



The following command will begin operation of the Remote LCD display:



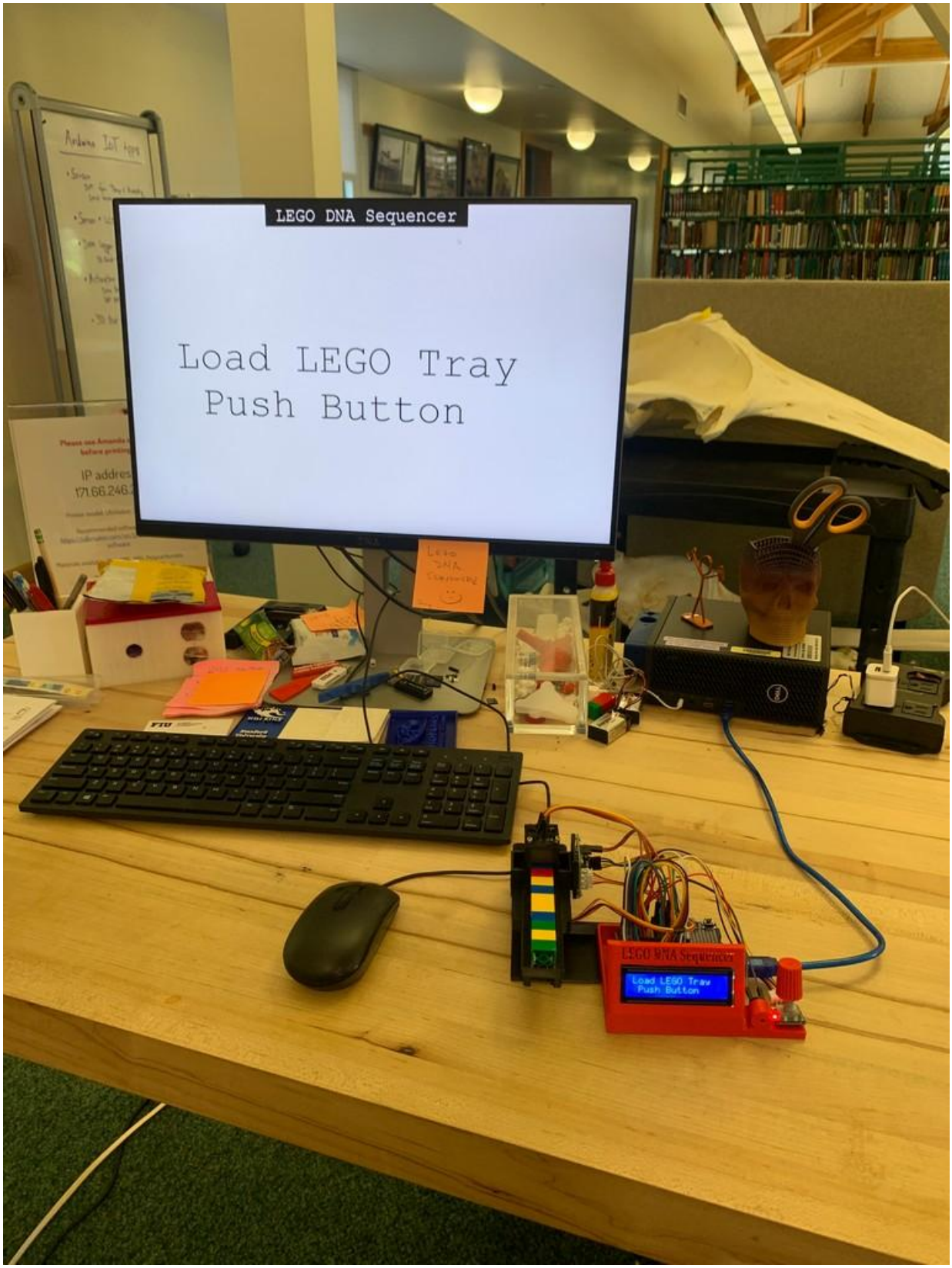
The Welcome screen will be displayed and the LEGO DNA Sequencer will reboot, showing its startup screen.

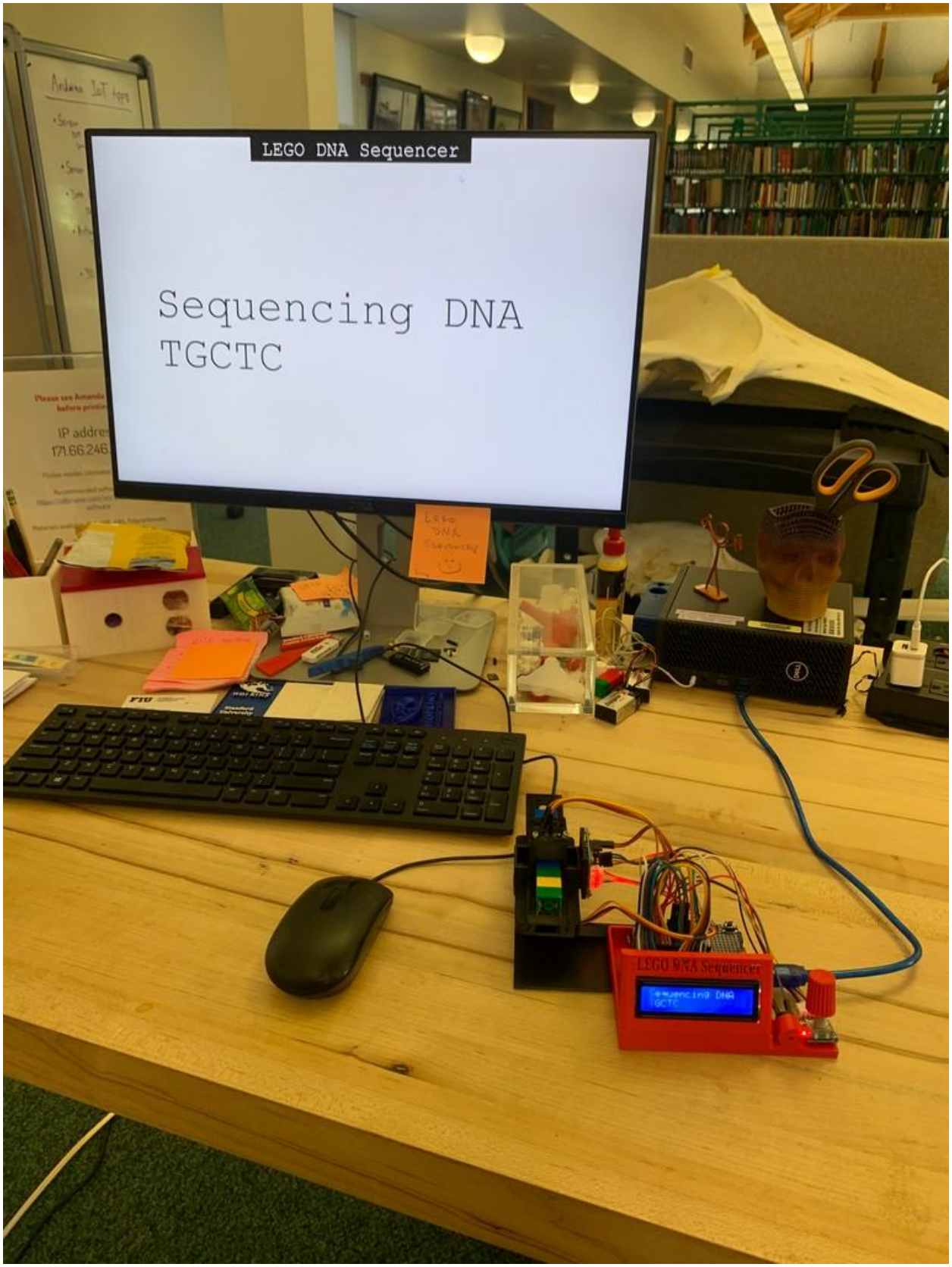
From that point on the contents of the LCD screen on the LEGO DNA Sequencer will be displayed on your computer screen.

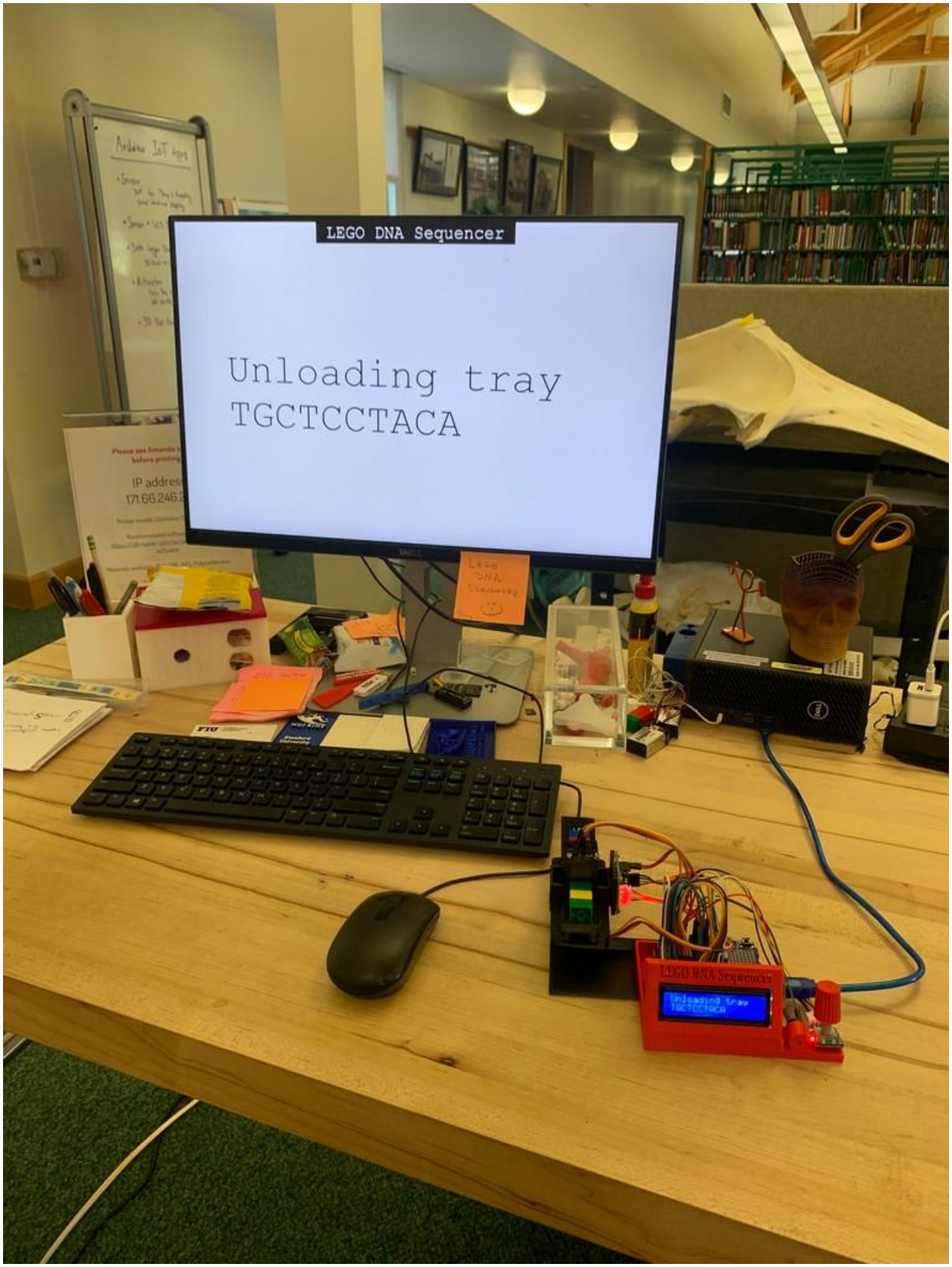
To terminate the RemoteLCD program enter the following keys:

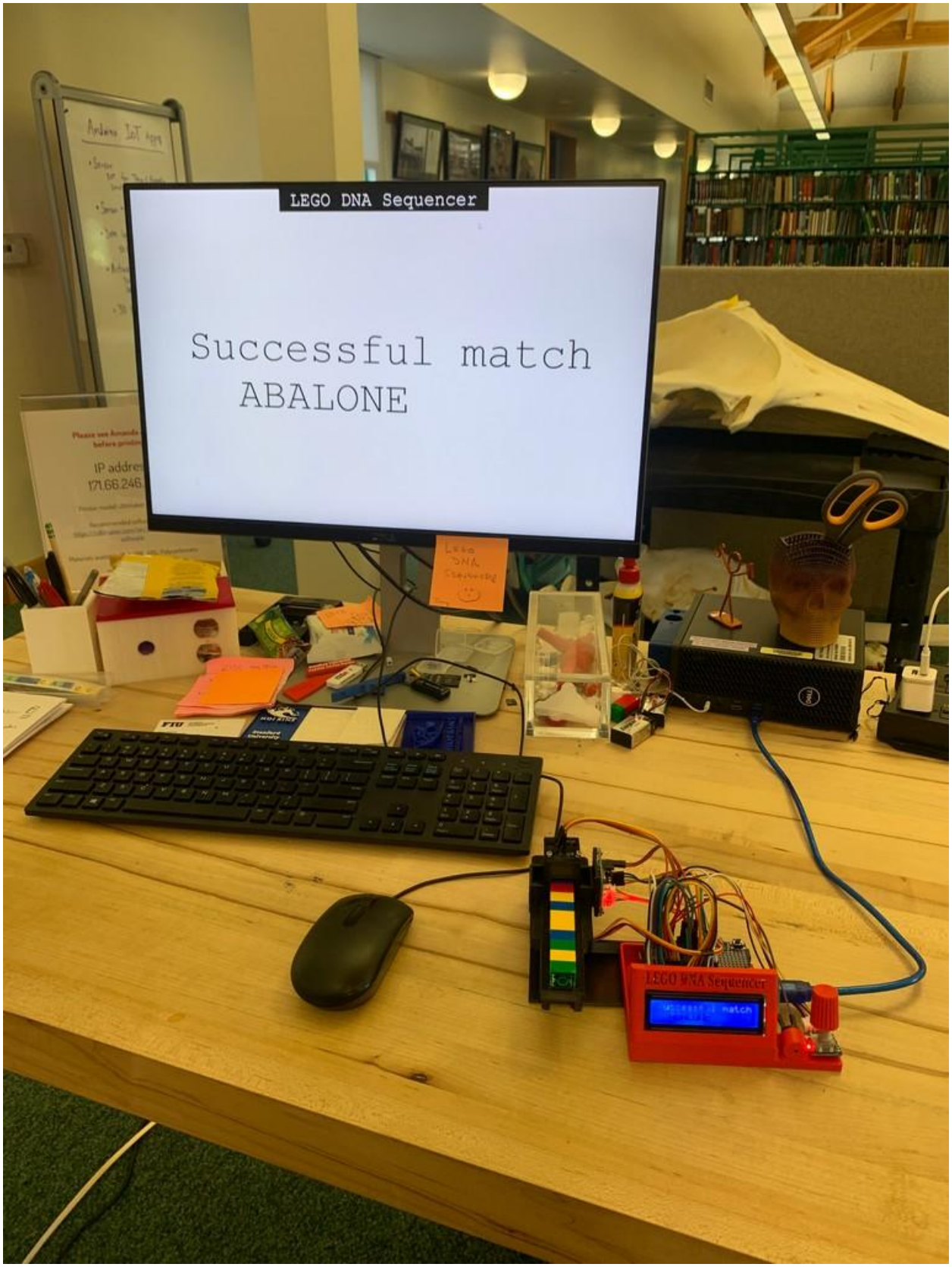
Alt - F4

The the following screens show the RemoteLCD output:


LEGO DNA Sequencer
Load LEGO Tray
Push Button
LEGO DNA Sequencer
Load LEGO Tray
Push Button


LEGO DNA Sequencer
Sequencing DNA
TGCTC
LEGO DNA Sequencer


LEGO DNA Sequencer
Unloading tray
TGCTCCTACA


LEGO DNA Sequencer
Successful match
ABALONE
LEGO DNA Sequencer

Installation for Remote LCD for tkinter

The steps to run the Remote LCD Python program on your laptop computer are as follows:

Windows

- Install python
- Install pip
- Install tk
- Install pyserial

Mac

- Install python
- Install pip
- Install tk
- Install pyserial

DIY: Build Your Own LEGO DNA Sequencer

A primary objective of this project has been to provide all of the documentation required to independently build your own LEGO DNA Sequencer. To that end all of the components of the system are open source.

The DIY elements include:

- Operators Manual for the end users
- Electronics assembly details with:
    - Parts list
    - Breadboard wiring of components
- STL files for all 3D printed parts
- Assembly instructions for the 3D printed parts and electronics
- Software source code and instructions to upload code to Arduino
- System and unit testing and troubleshooting

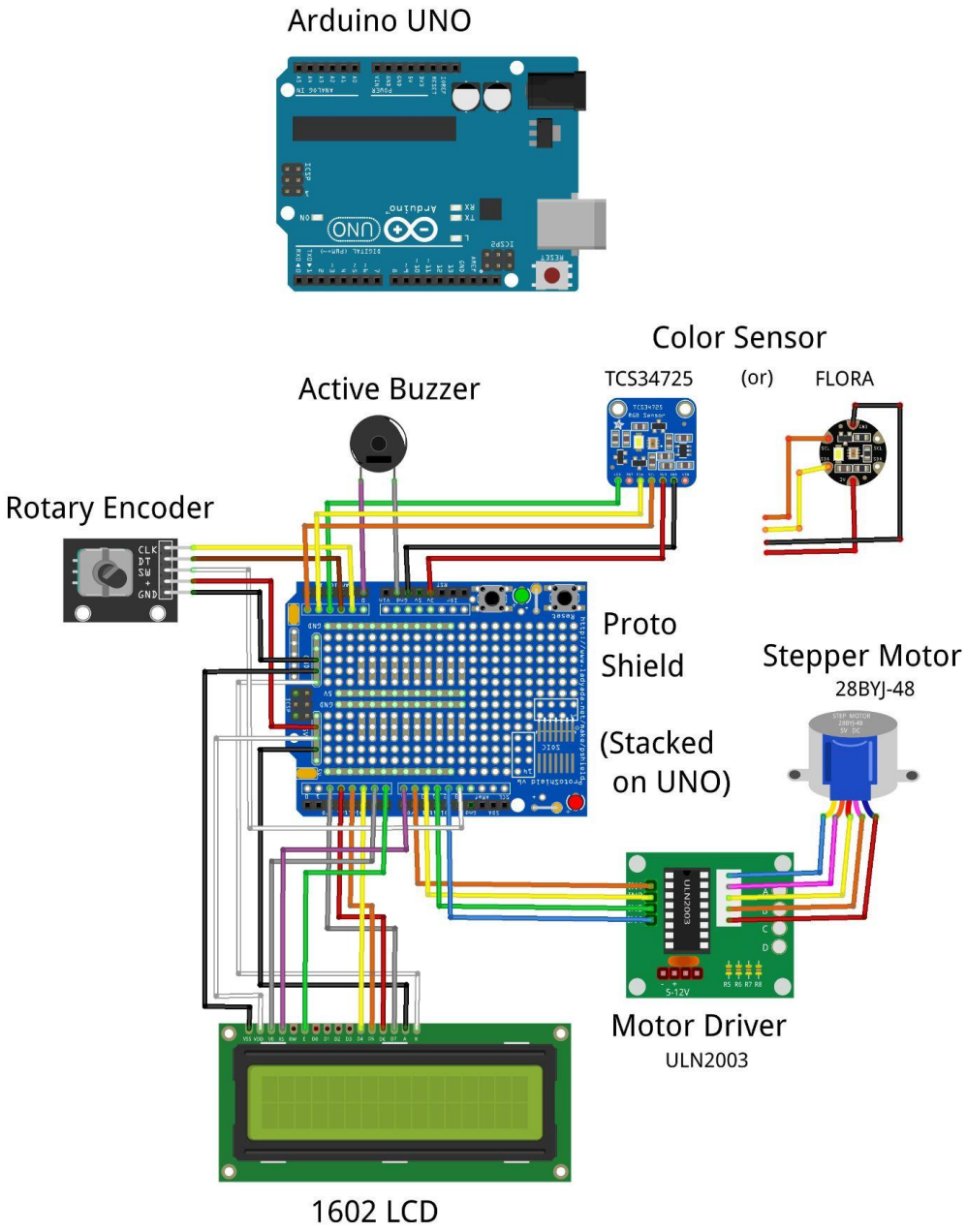
The LEGO DNA Sequencer License

Electronics

The LEGO DNA Sequencer electronics is controlled by an Arduino UNO microcontroller.

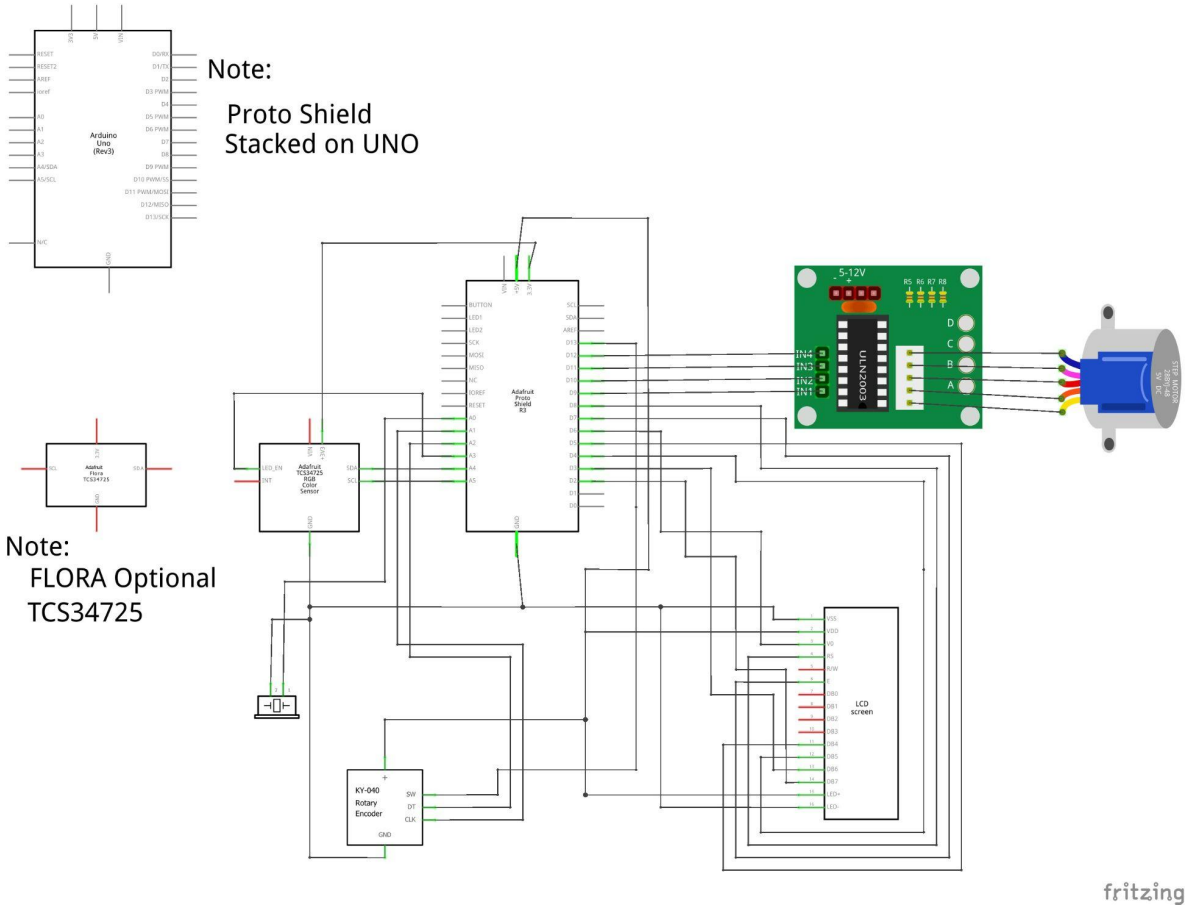
LEGO DNA Sequencer

Breadboard



LEGO DNA Sequencer

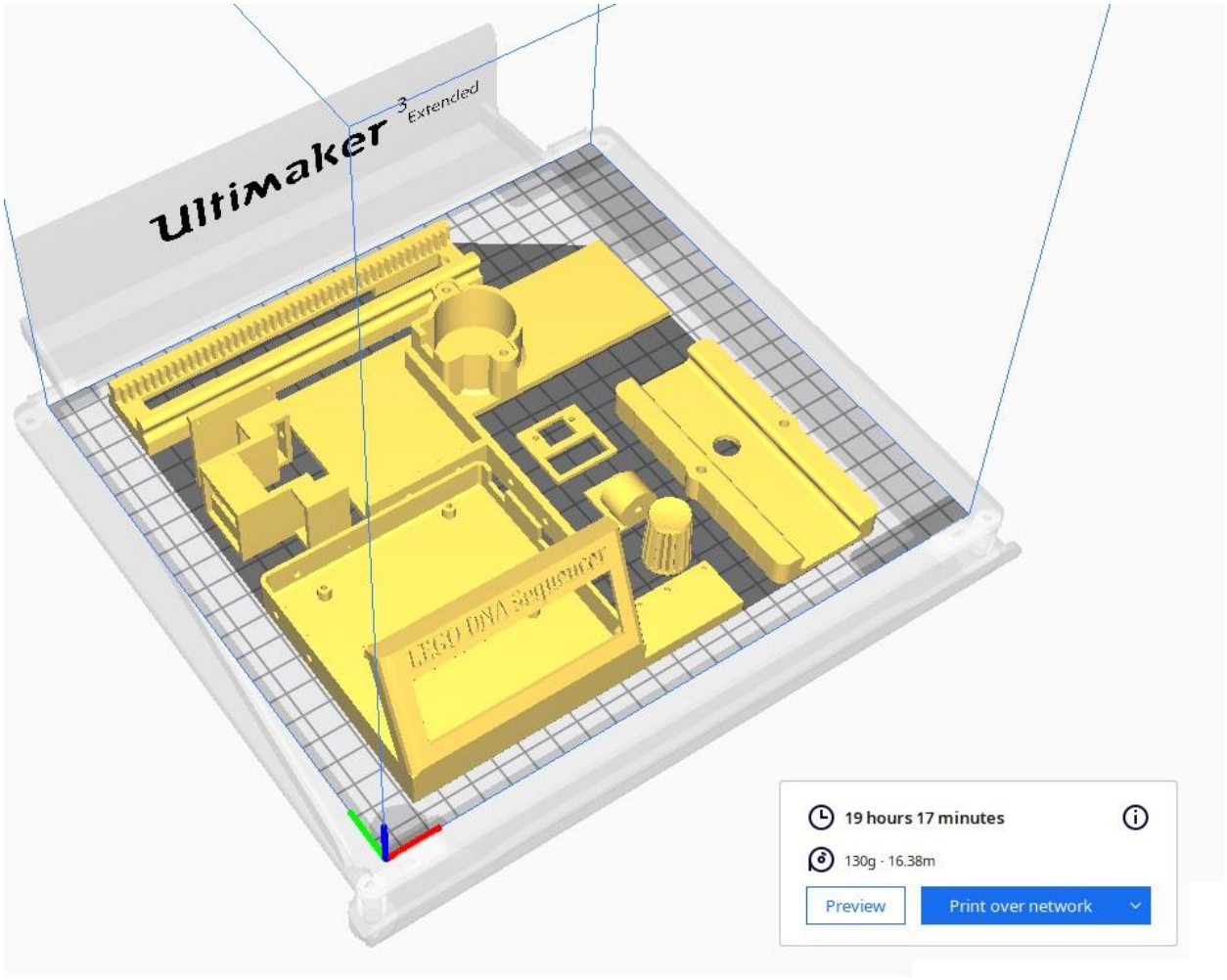
Schematic



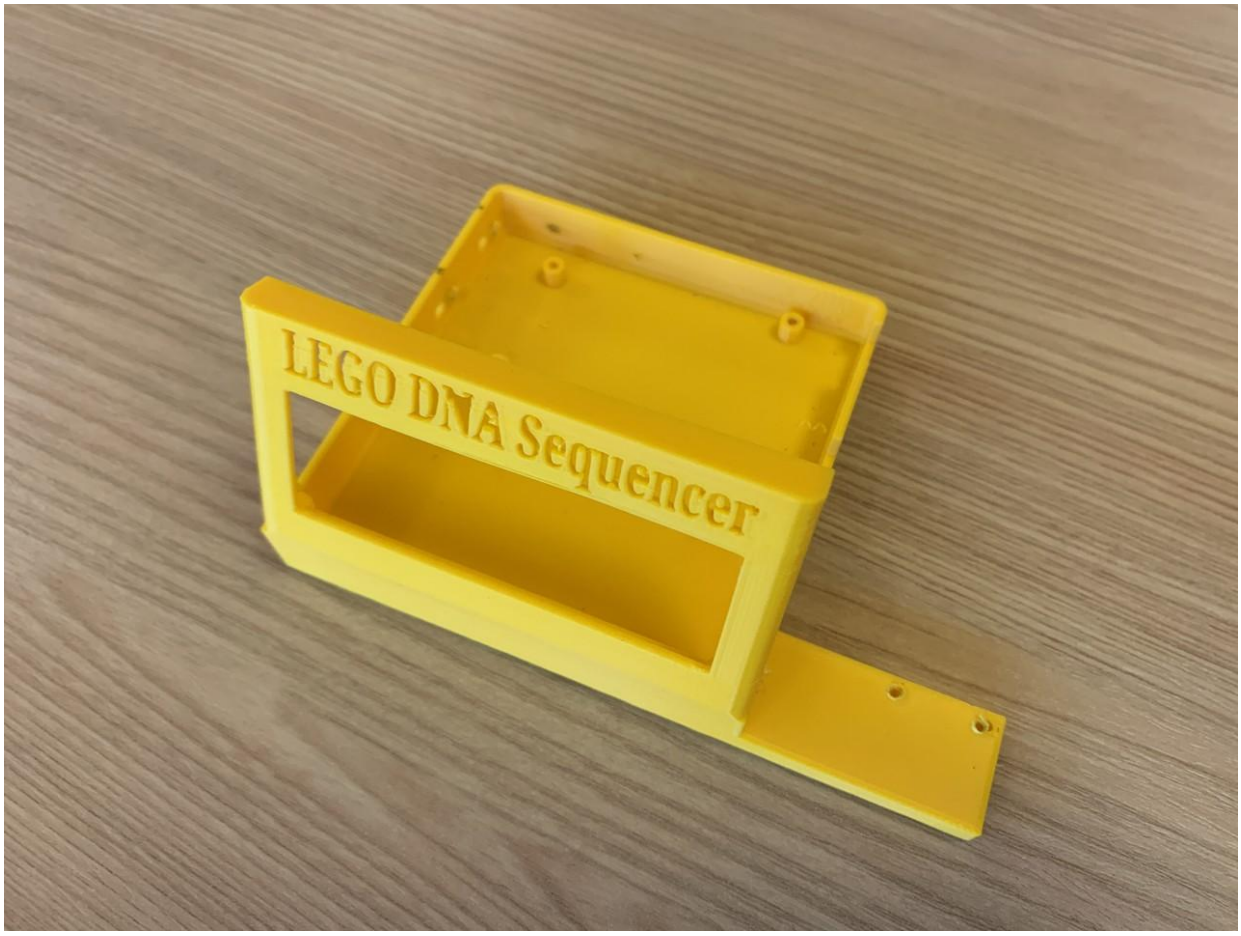
3D Printed Parts

There are a total of 8 parts to be printed.

The 3D printing can be done in a single printing process, or individually.

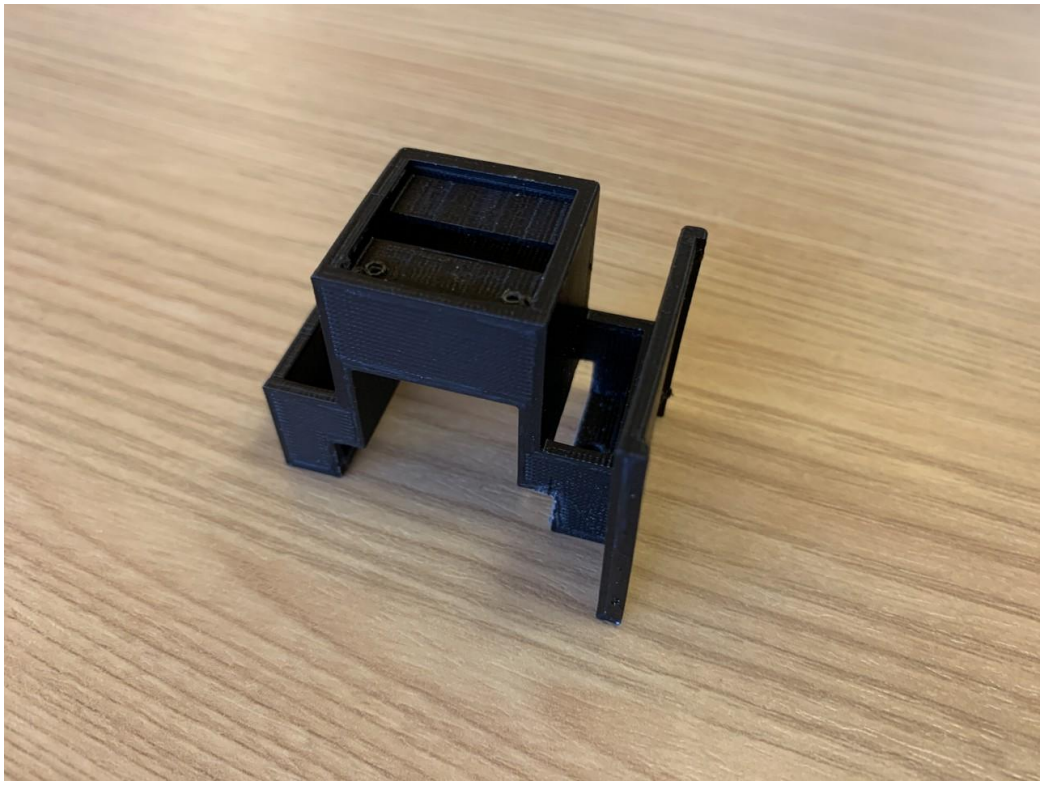


Arduino and LCD Case



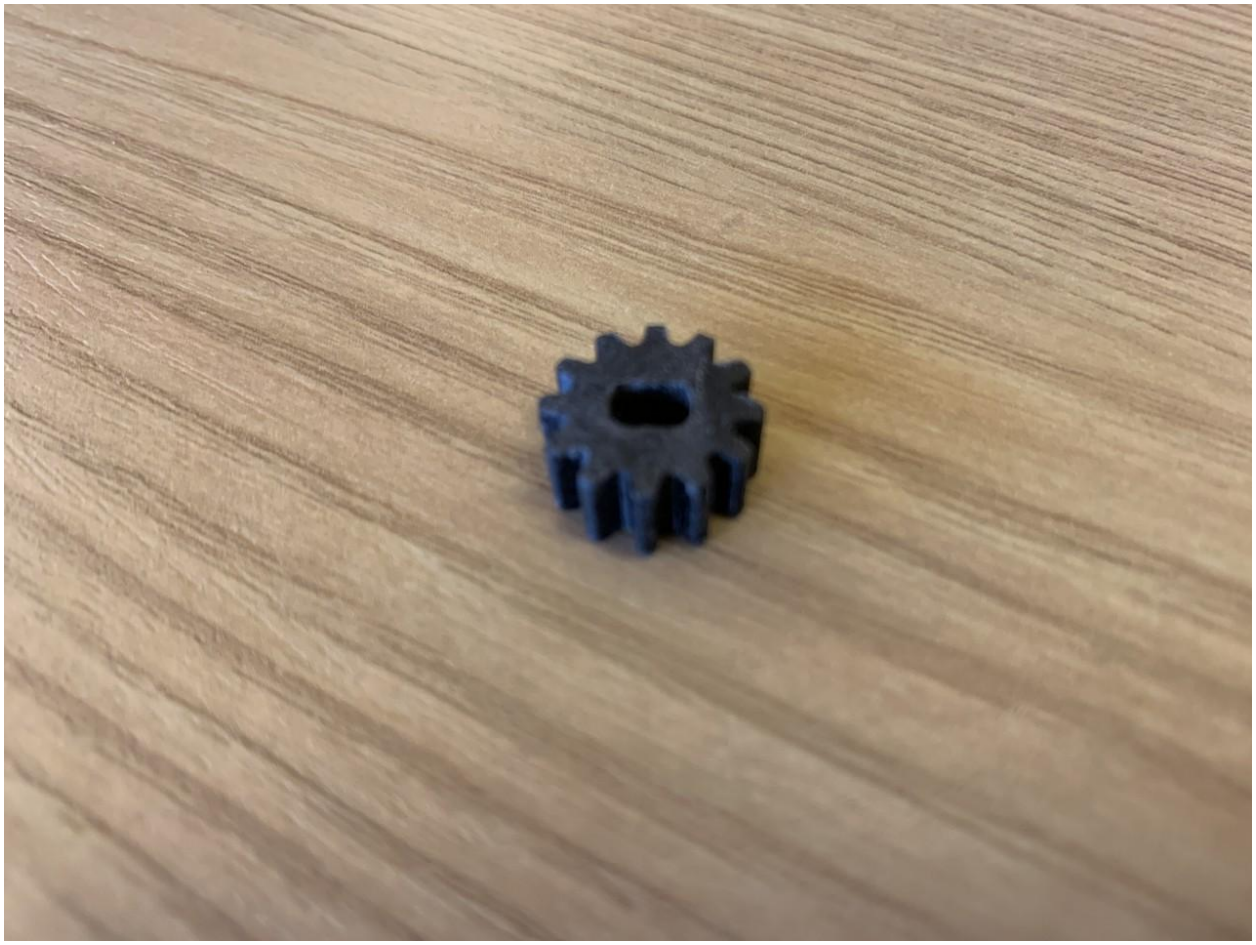
STL File

Color Sensor and Motor Drive Mount



STL File

Gear

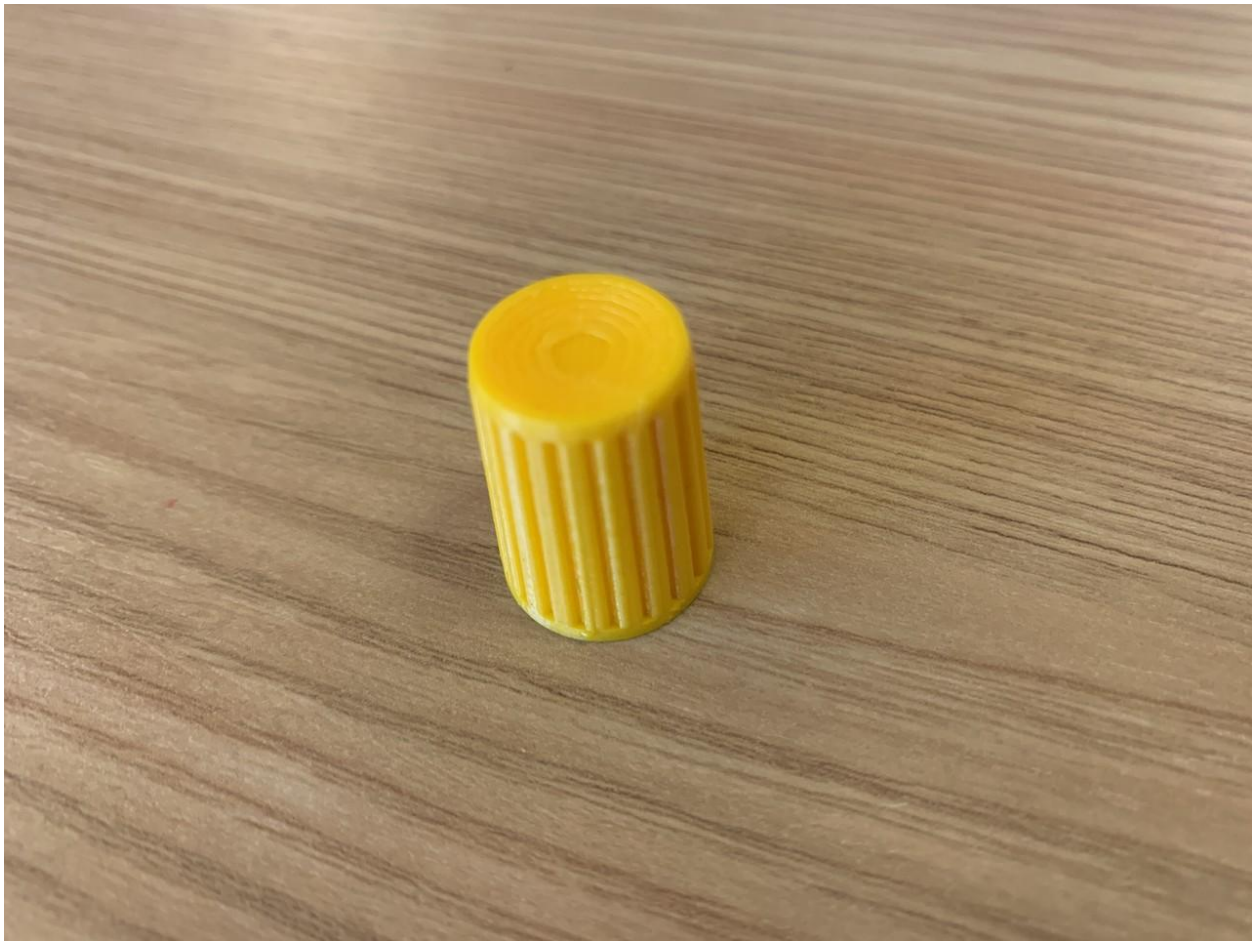


STL File

LEGO Tray

STL File

Rotary Encoder Knob



STL File

Stepper Motor Mount

STL File

Tray Holder and Gear Mount

STL File

Buzzer Mount

STL File

Parts Remixed from Thingiverse.com

Arduino Case with LCD display 1602
28byj stackable linear slide
Rotary Encoder Knob (16 x 21.5mm, 6mm flatted shaft)

License

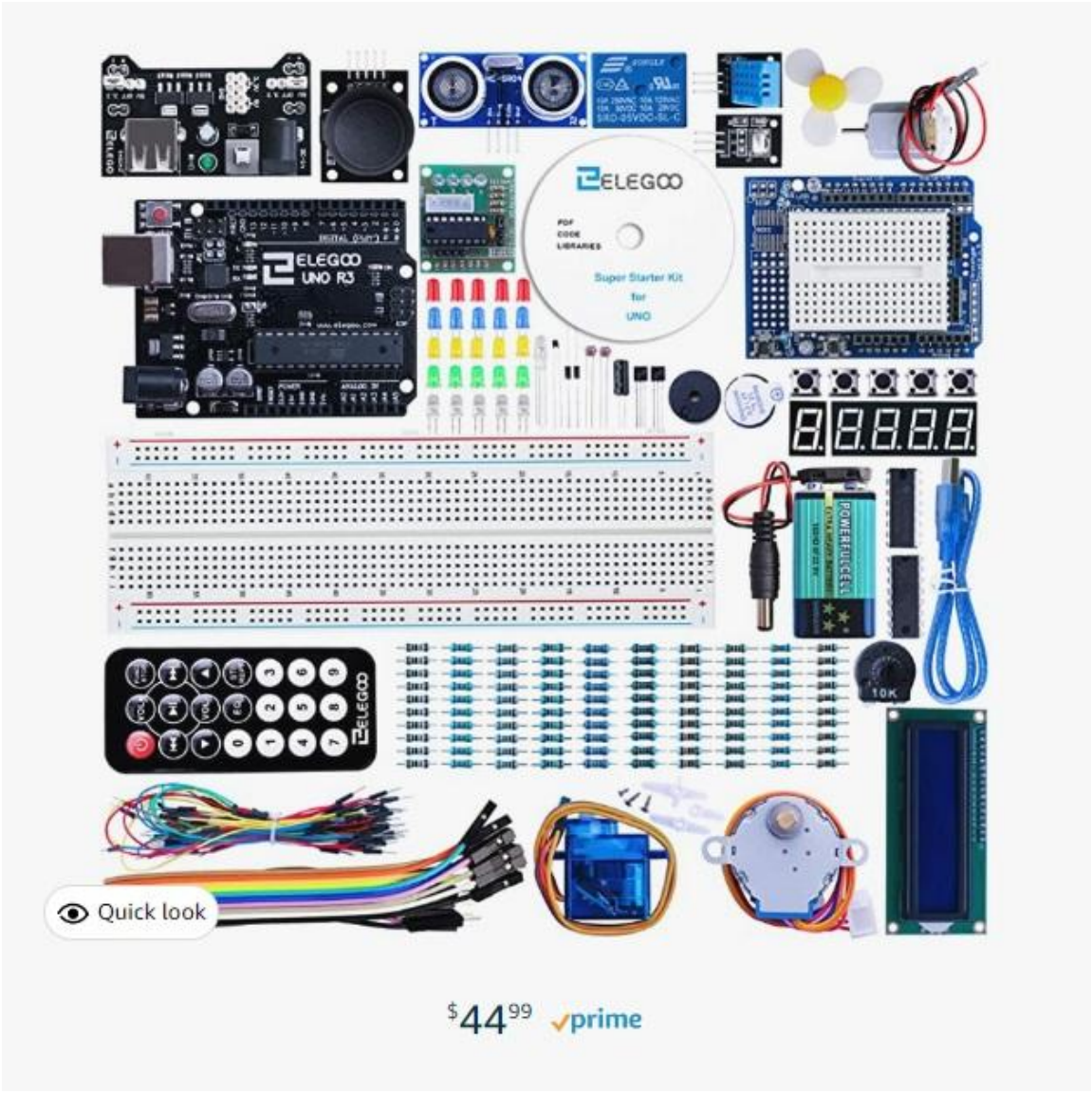
Parts List

The parts required to assemble the LEGO DNA Sequencer include:

Arduino Advanced Starter Kit

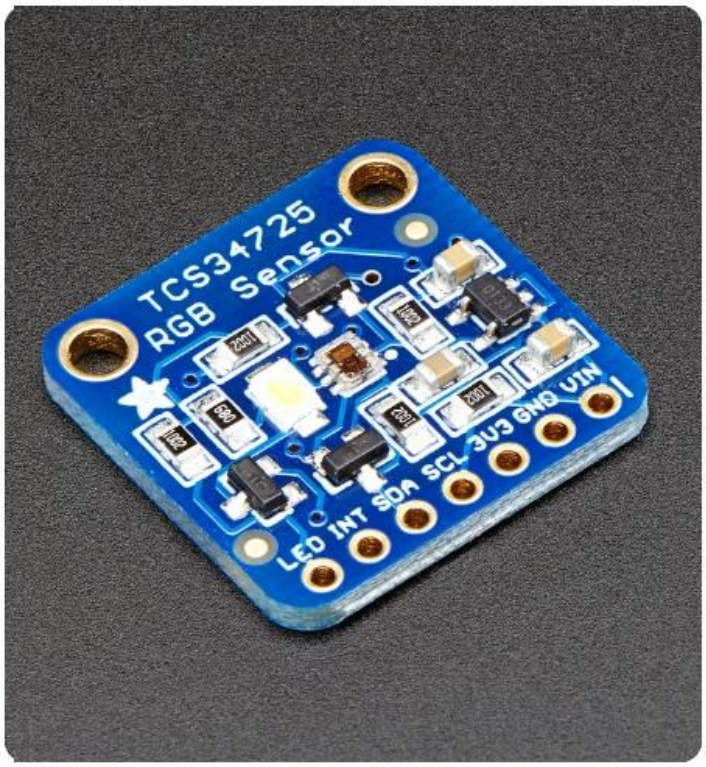
The easiest way to get the parts for the LEGO DNA Sequencer is to use one of the Arduino Advanced Starter Kits. There are several kits available which have all of the electronic parts with the only exception of the TCS34725 Color Sensor.

ELEGOO Upgraded UNO R3 Project Most Complete Starter Kit w/Tutorial Compatible with Arduino IDE (63 Items) V2.0



$44.99 prime

TCS34725 – RGB Color Sensor with IR filter and White LED



RGB Color Sensor with IR filter and White LED - TCS34725

Product ID: 1334

$7.95

Discontinued

You can grab the Adafruit APDS9960 Proximity, Light, RGB, and Gesture Sensor - STEMMA QT / Qwiic instead!

Description

Technical Details

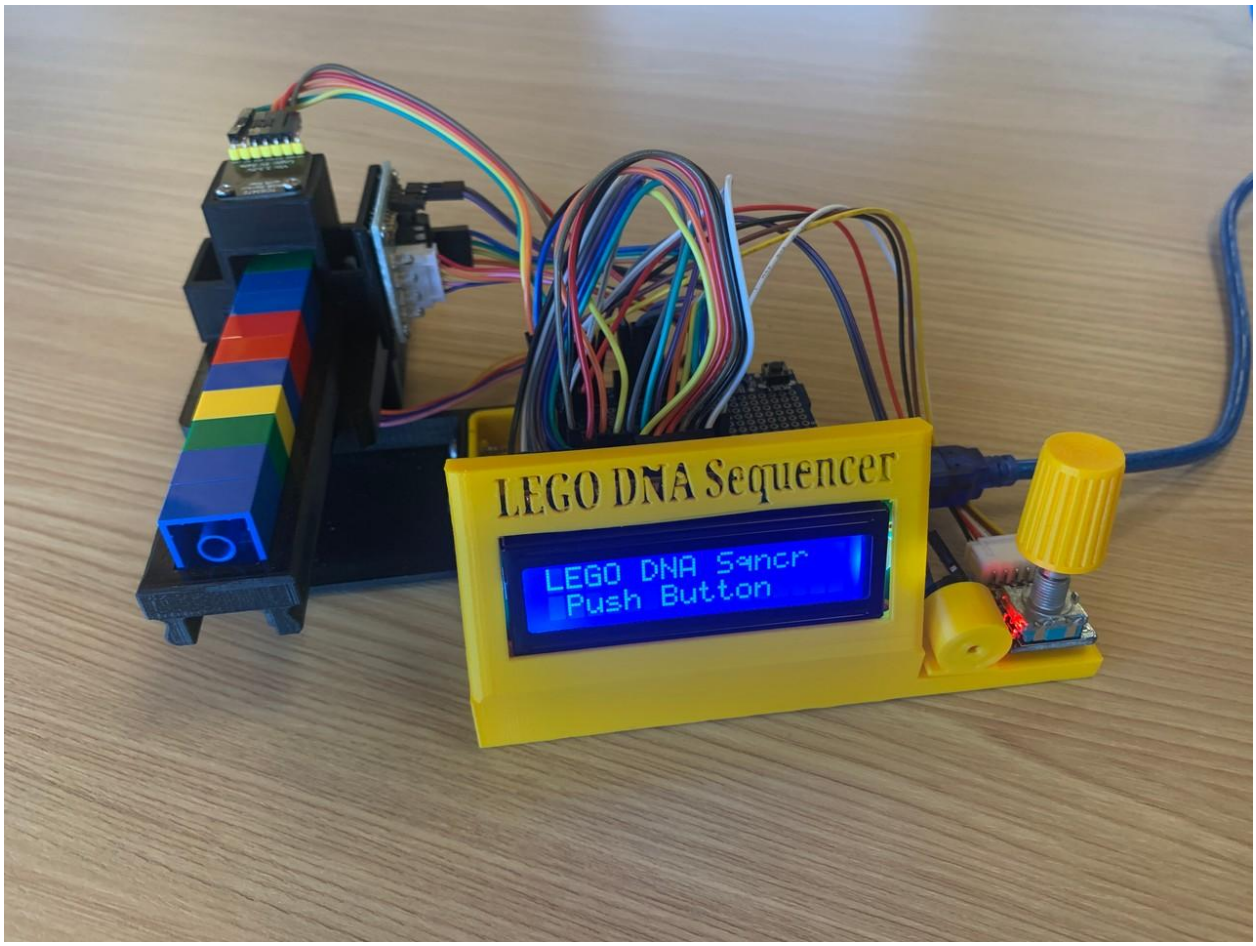
Hardware for the Assembly

Sheet Metal Screws

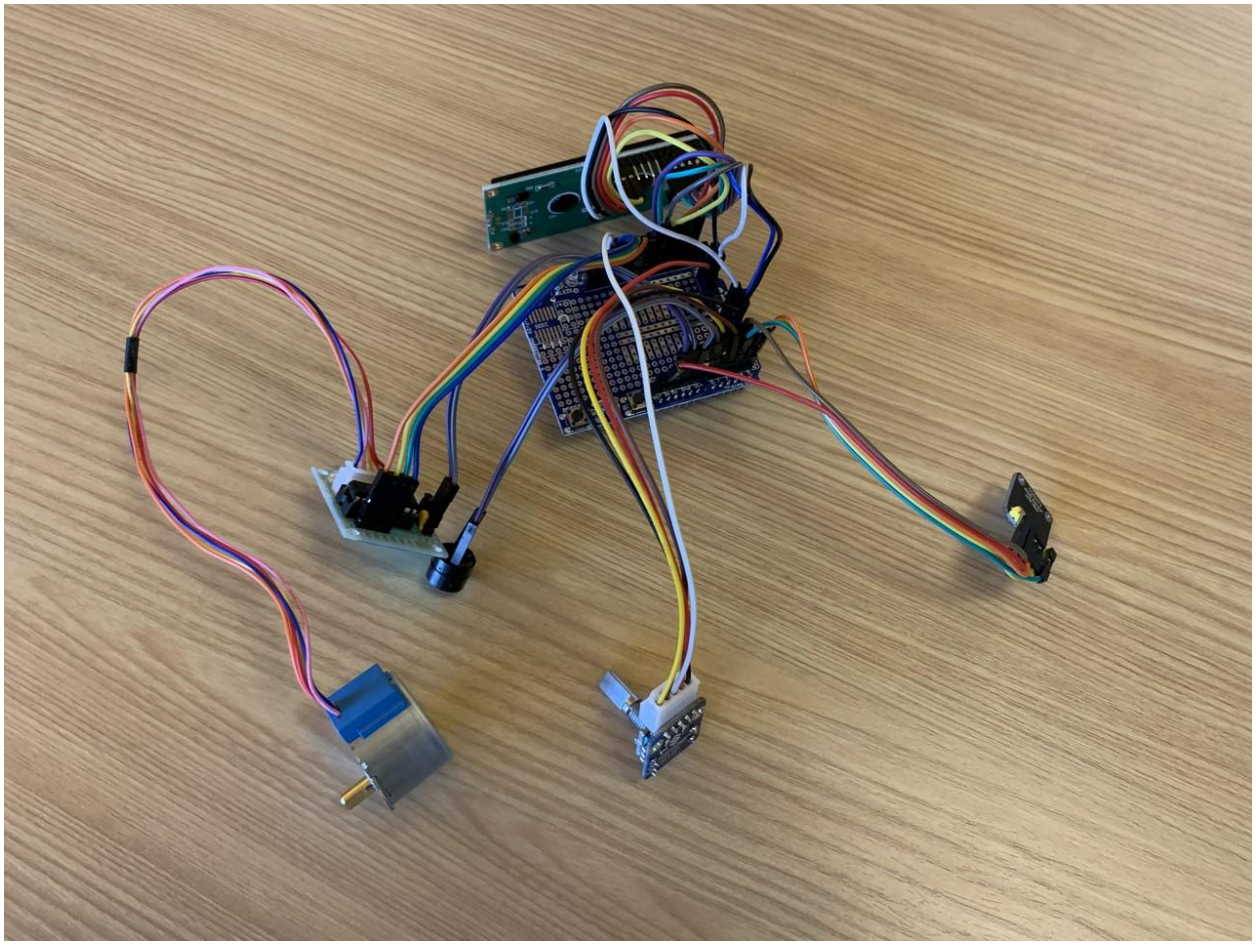
(2) #6 ¾”
(3) #6 ½”
(16) #4 ¼”

Assembly and Construction Step Details

The steps in building a DIY LEGO DNA Sequencer includes the following, printing

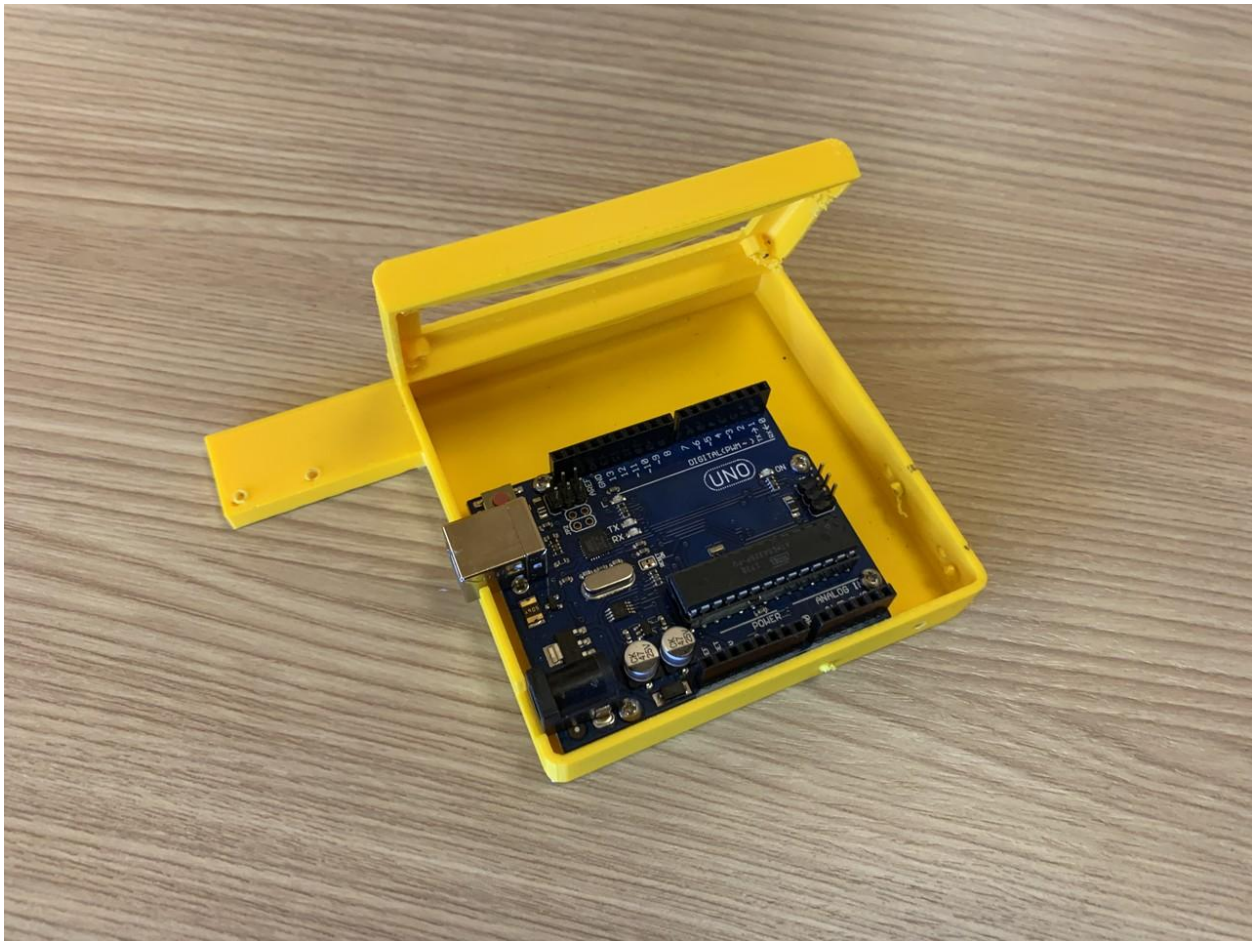


Assemble the electronics using the LEGO DNA Sequencer Breadboard.

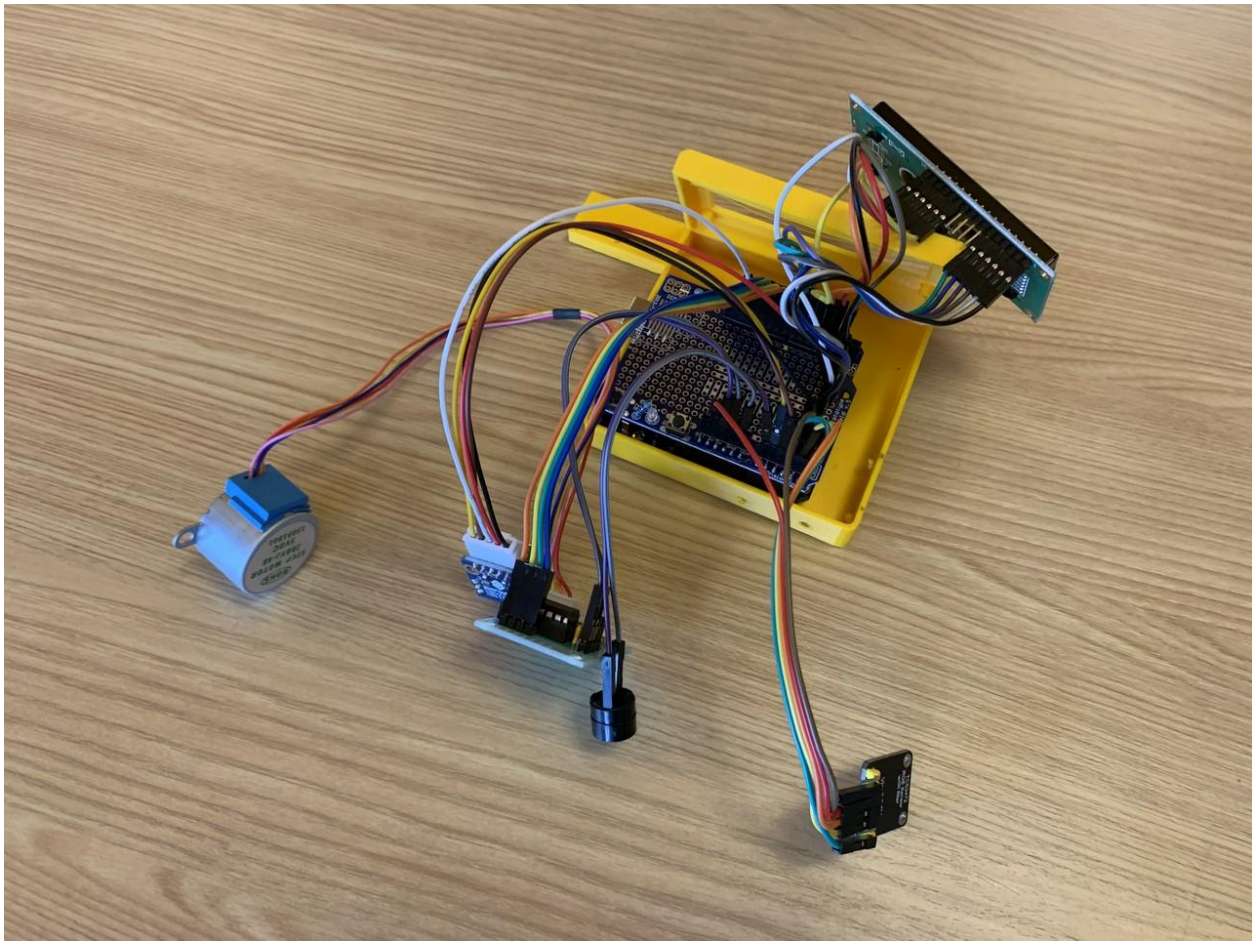


Arduino and LCD Case

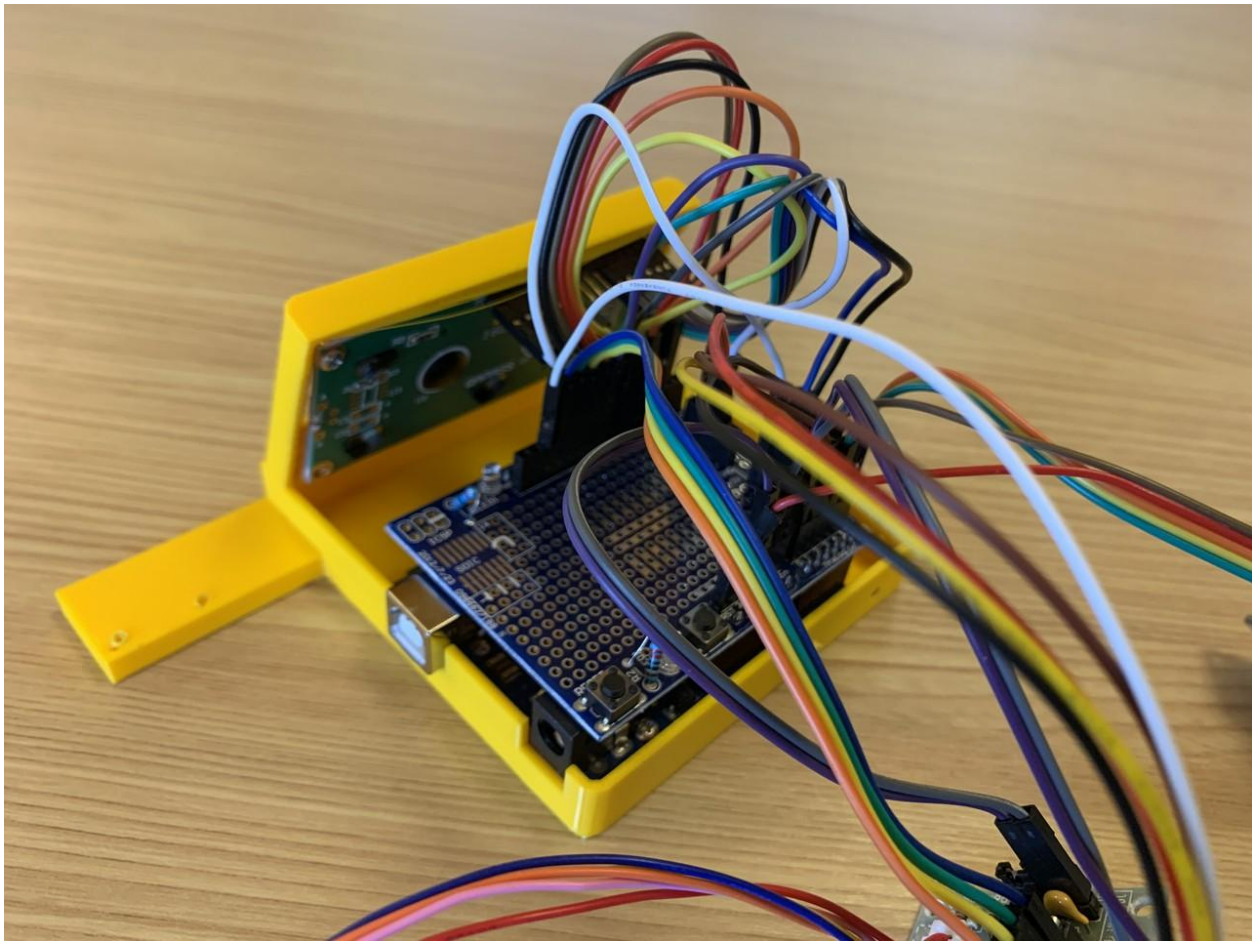
Insert the Arduino in the case and install it with (4) #4 ¼” sheet metal screws.



Insert the Proto Shield on top of the Arduino which has been screwed into the case.

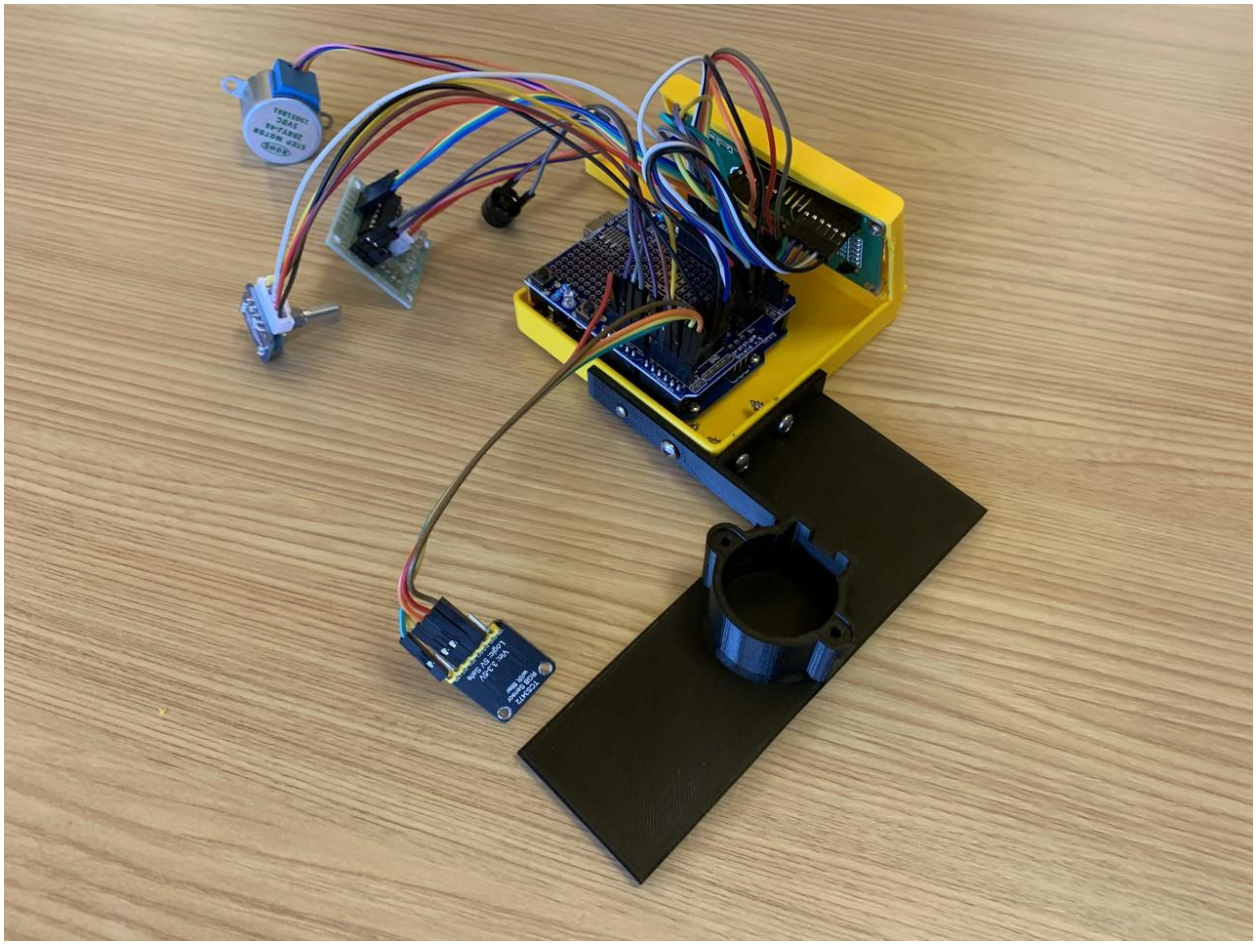


Insert the 1602 LCD Display in the case and install the upper (2) #4 ¼” sheet metal screws. Note that the bottom two mounting screws are not necessary but can also be inserted.

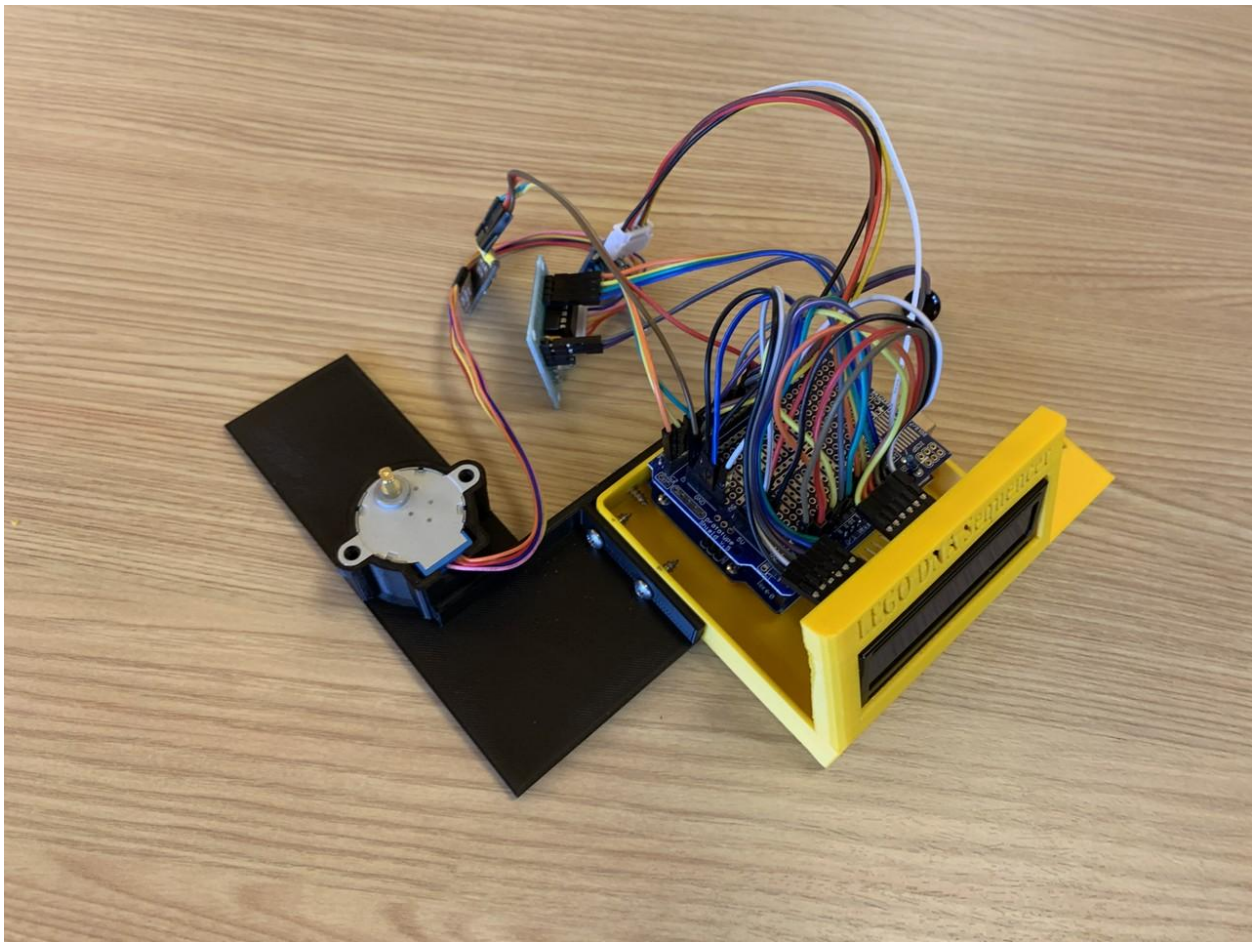


Complete the Assembly

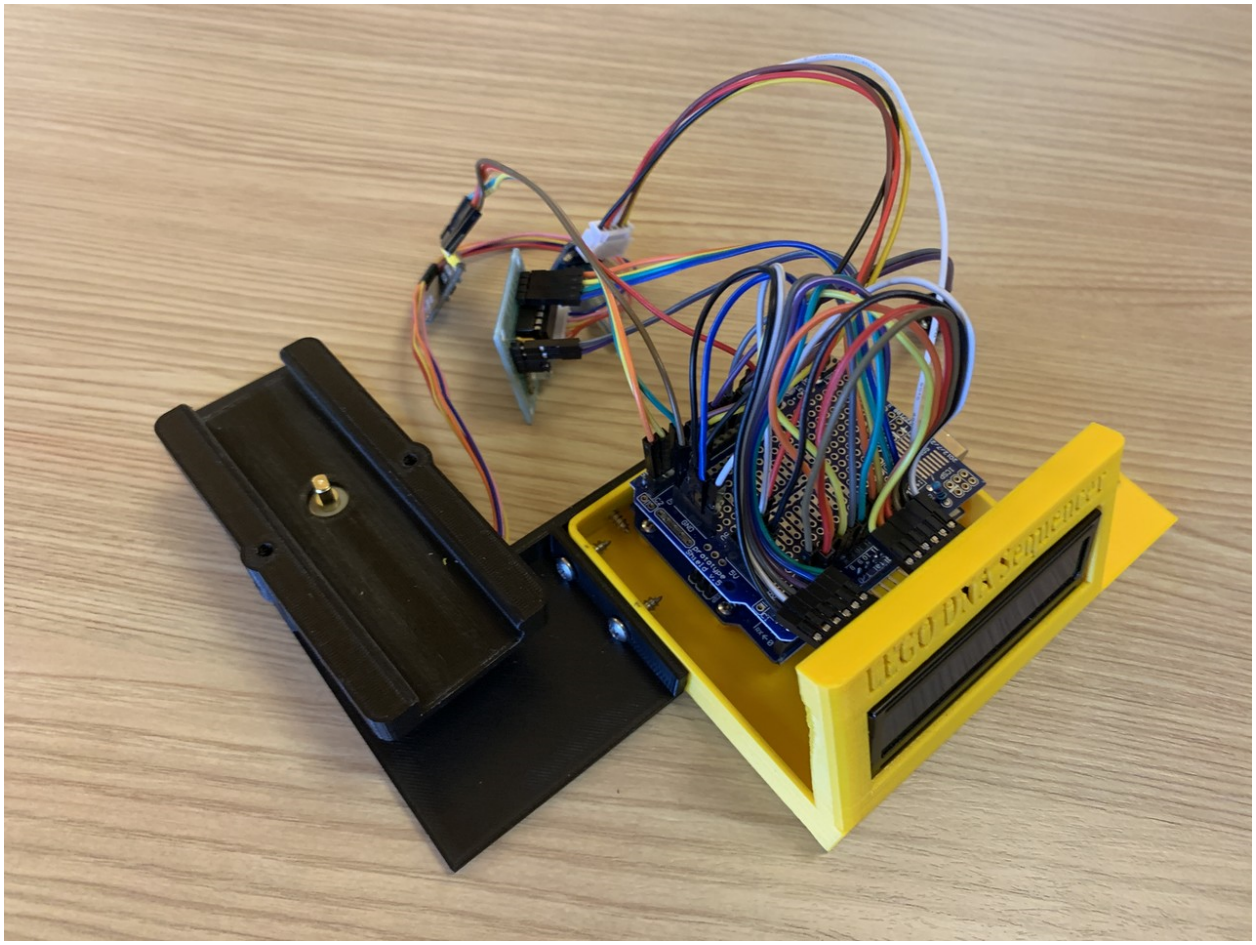
Connect the stepper motor base to the Arduino and LCD case. This is done with (3) #6 ½” sheet metal screws and (1) #4 ¼ sheet metal screw. Note that the shorter screw is used to avoid striking the Arduino UNO.



Insert the 28BYJ-48 Stepper Motor into the base unit.



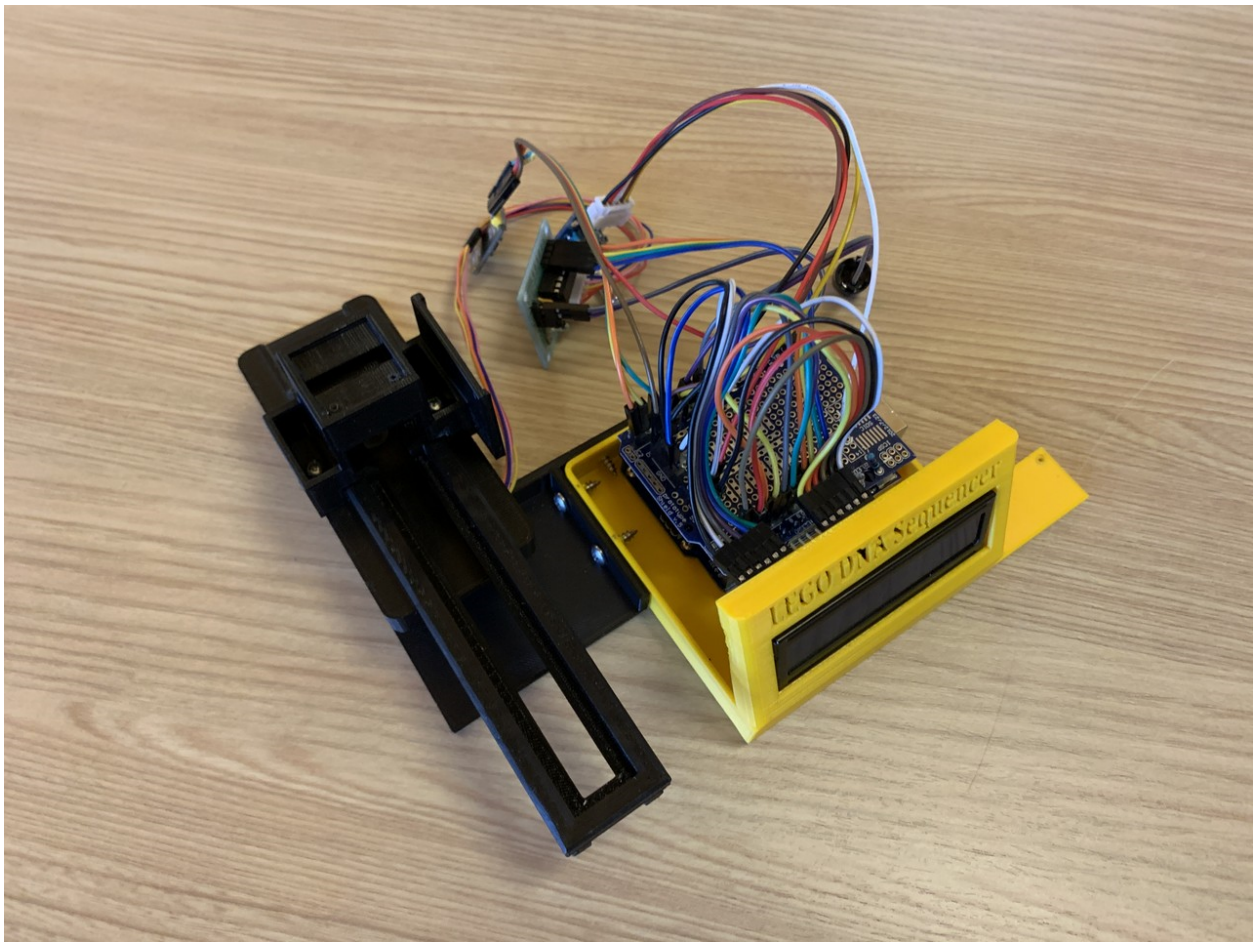
Mount the tray/gear holder on top of the stepper motor, aligning it with the gear shaft and the two mounting holes.



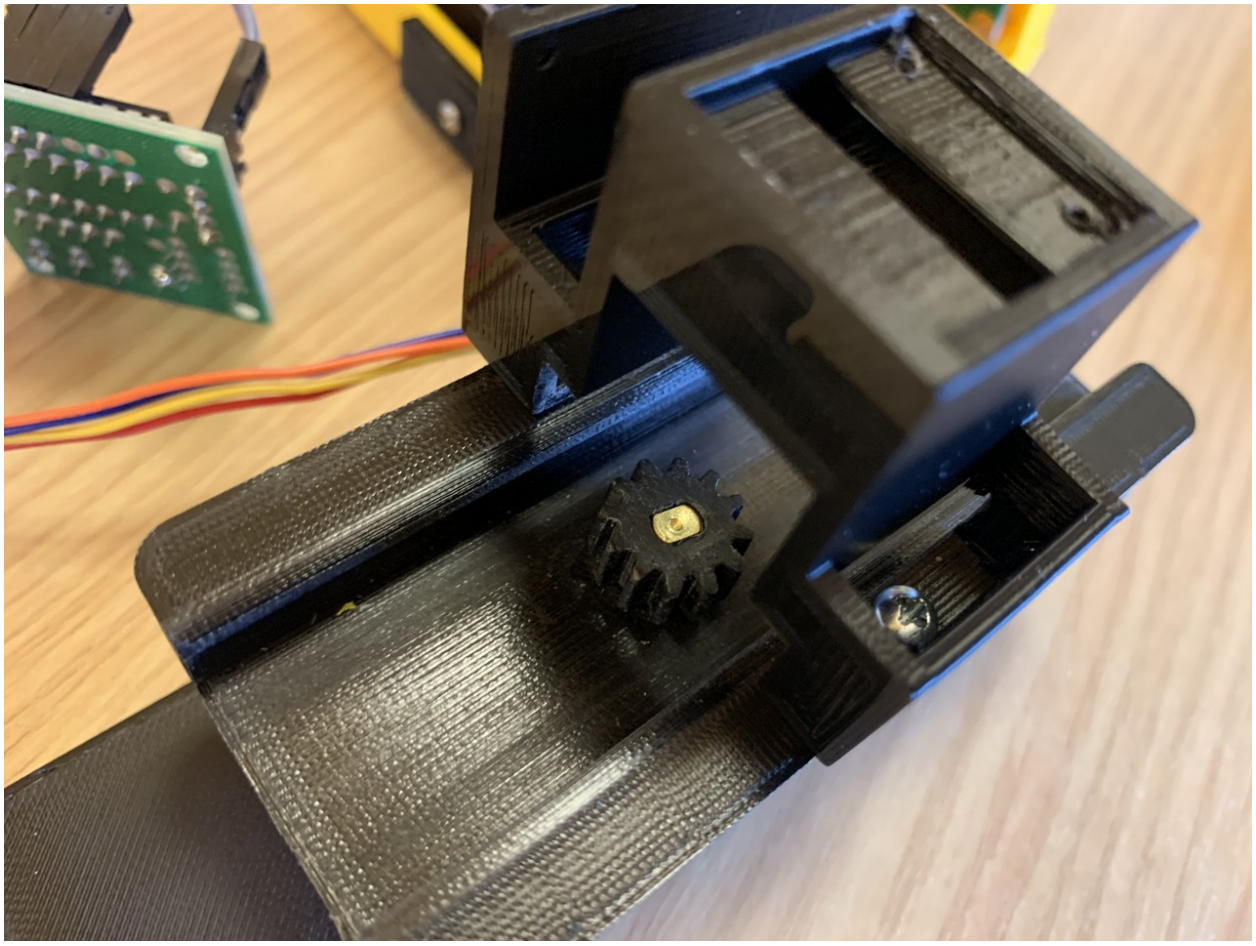
Verify that the gear moves smoothly in the track. This can be done by placing the gear in the opening position of the track and using a flat bladder screwdriver to rotate the gear. You should be able to rotate the gear the entire length of the track. You will need to correct any places where the gear binds and doesn't move smoothly. Sandpaper and a deburring tool may do the job.

Place the color sensor / motor drive on top of the tray holder. Note that the motor drive attachment must be on the Arduino case side. Insert (2) #6 ¾” sheet metal screws into the two mounting holes and gently tighten them, aligning all three elements: base mount, holder mount, and sensor mount.

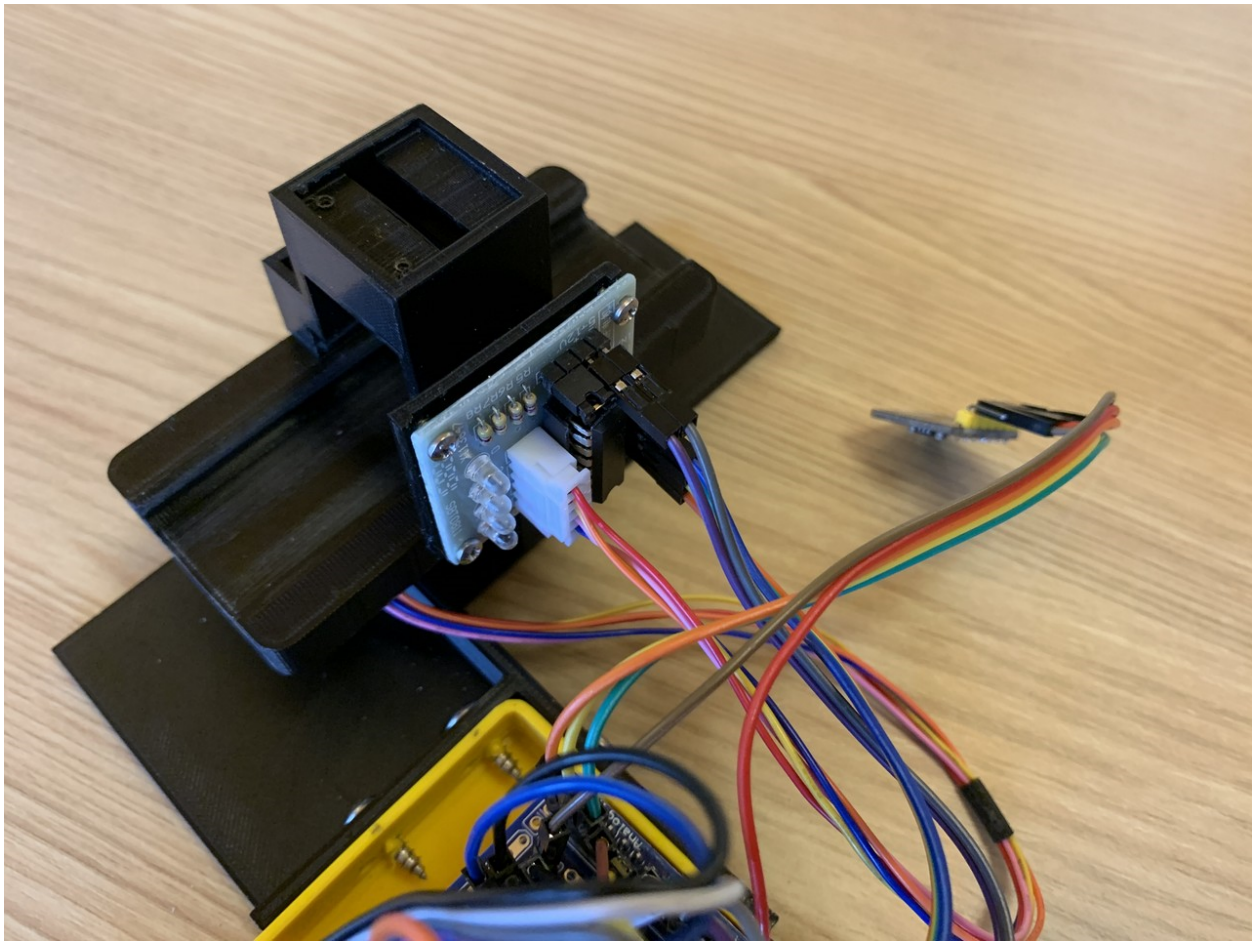
Make certain that the LEGO tray holder slides smoothly back and forth along the entire track.



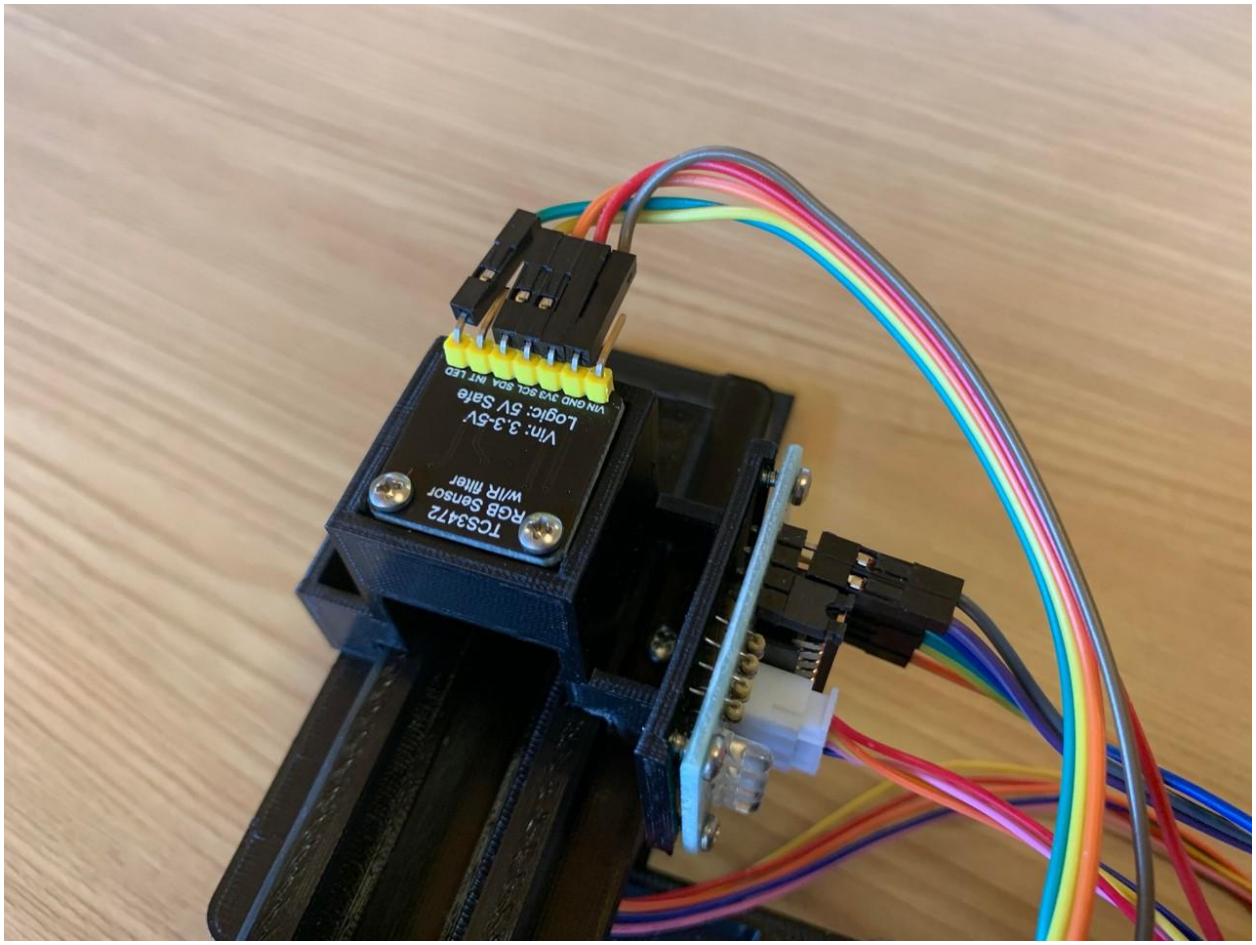
Insert the gear onto the top of the stepper motor shaft.



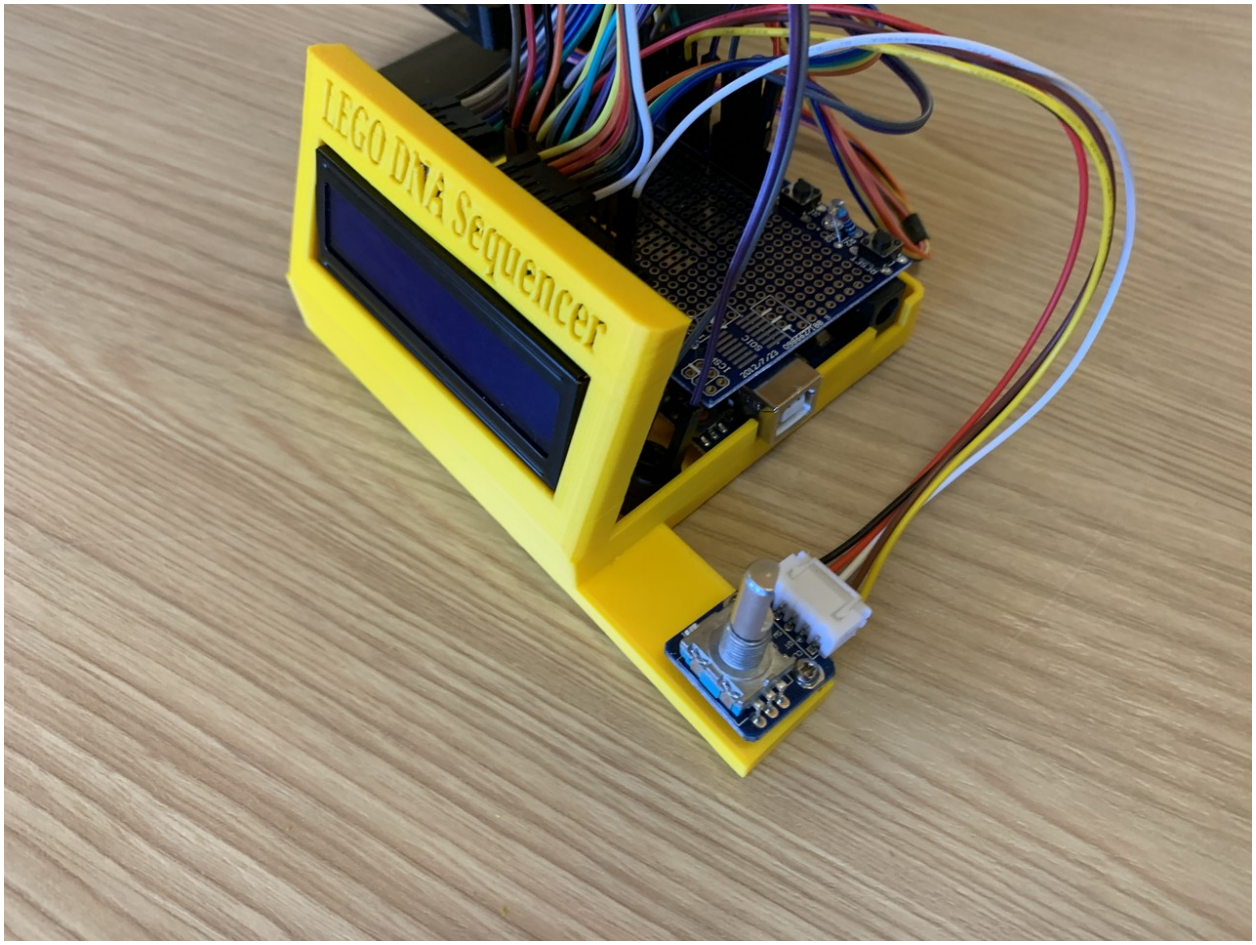
Insert the motor driver on the side of the color sensor and motor driver mount. Make sure that you have oriented the board as shown below.



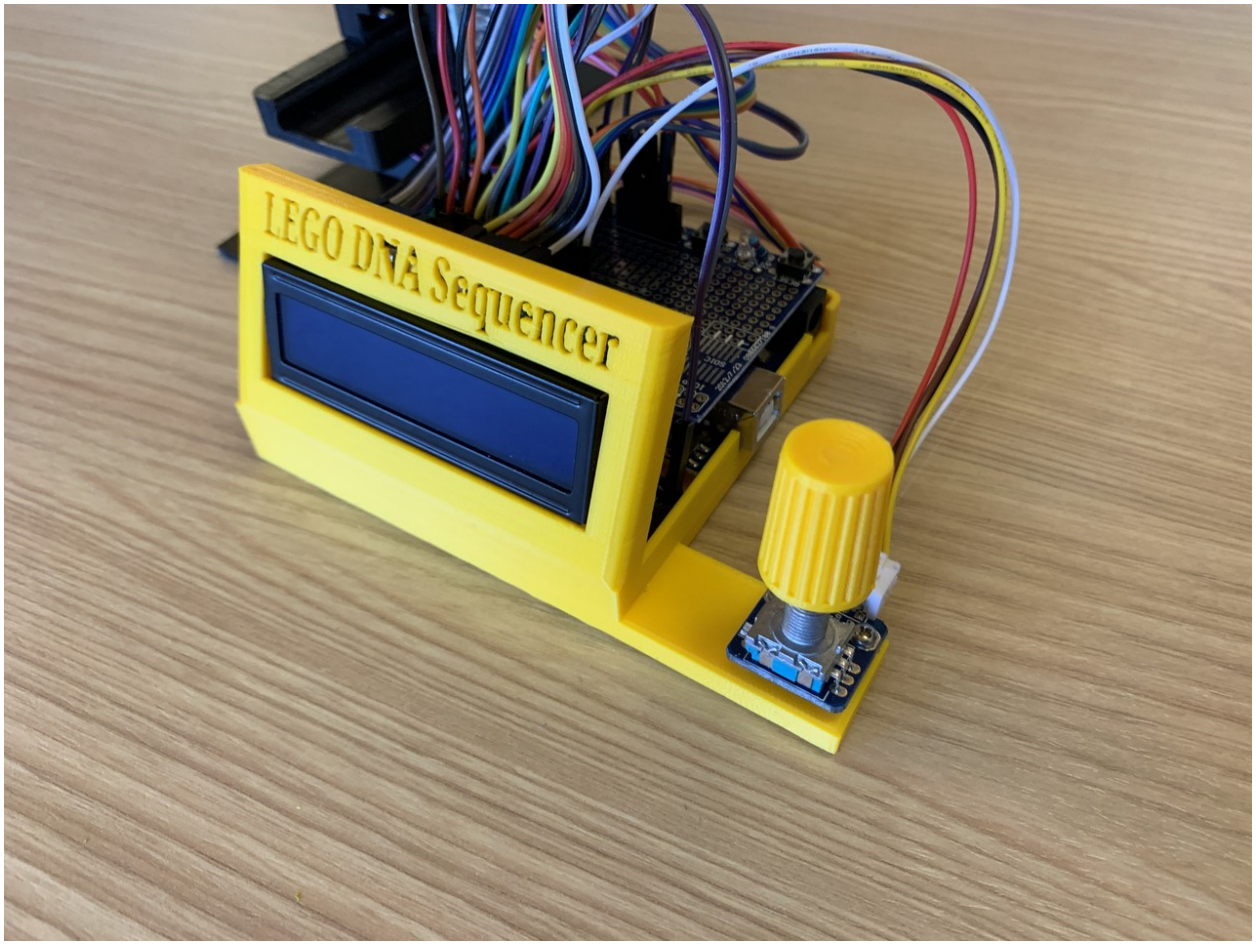
Attach the TCS34725 Color Sensor to the top of the color sensor mount with (2) #4 ¼” sheet metal screws. Note that the cables must protrude from the rear of the assembly.



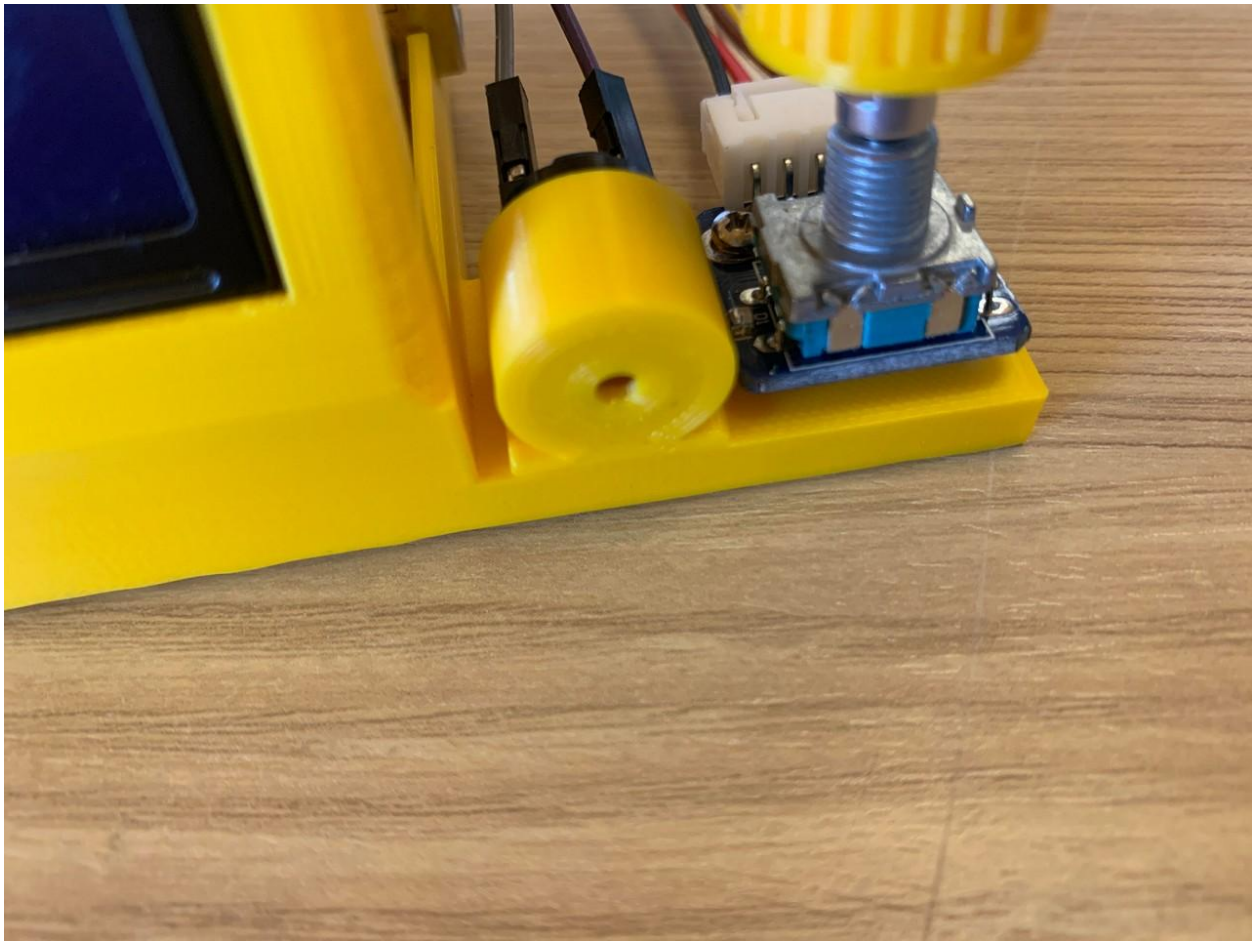
Attach the rotary encoder to the two far right holes in the flange extending from the right side of the Arduino case with (2) #4 ¼” sheet metal screws.



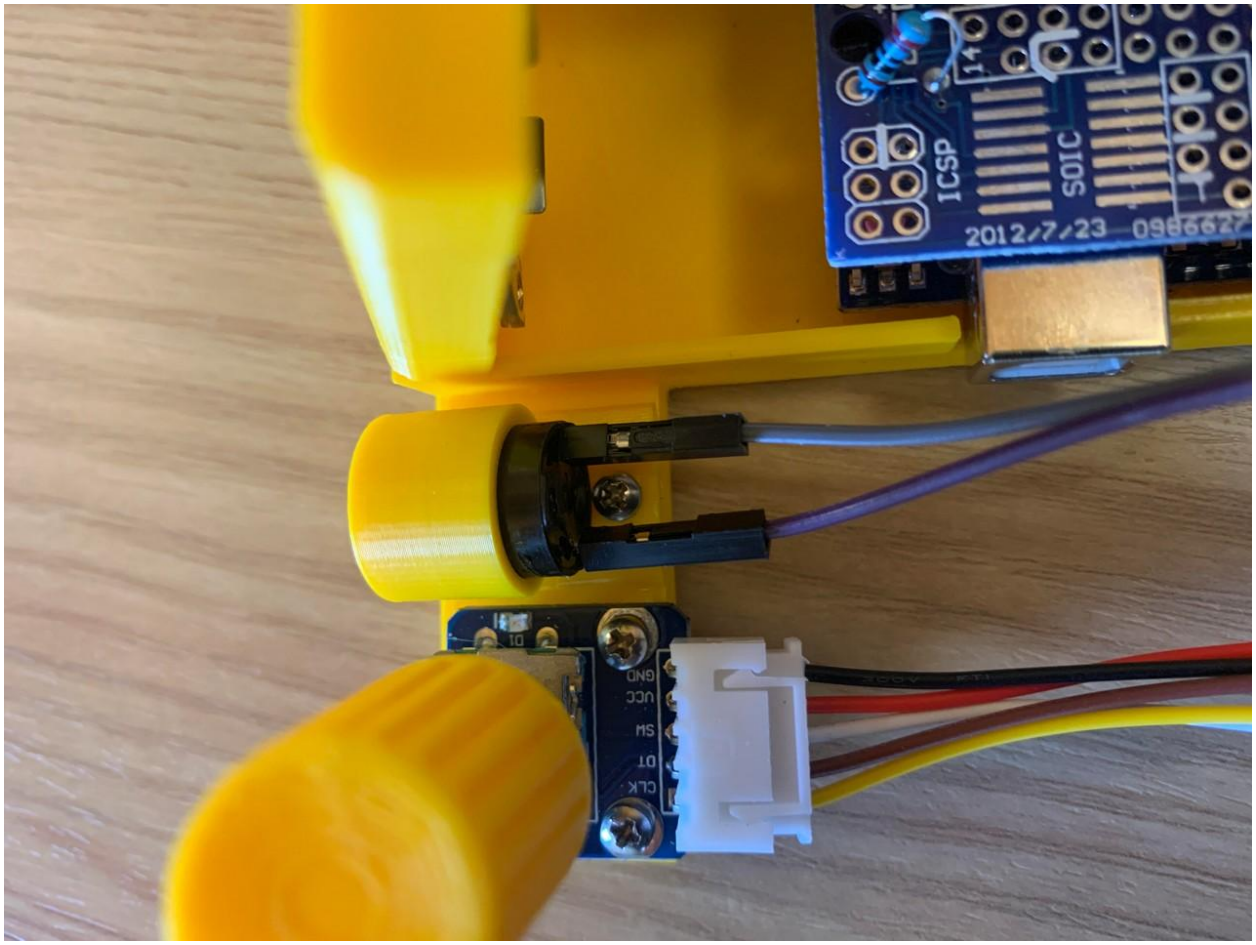
Place the knob on top of the rotary encoder shaft.



Attach the buzzer holder to the remaining hole in the flange extending from the right side of the Arduino case with (1) #4 ¼” sheet metal screw.



Insert the buzzer into the back of the holder with the pins on the aligned so that the sheet metal screw can be inserted between the two pins.



Unit and System Testing

There are Unit Testing Arduino modules for each of the hardware components, including:

- Buzzer
- LCD and Rotary Encoder
- Stepper Motor
- TCS34725

Successful operation of each of the Unit Tests should be done before a System Test with the LEGO DNA Sequencer.

The Arduino source is provided for each of the Unit Tests.

Buzzer Unit Test

```
/*************************************************************************
  LEGO DNA Sequencer
  Buzzer Unit Test
  Original Code:  2022-12-24
  Tom Rolander, MSEE
  Mentor for Circuit Design, Software, 3D Printing
  Miller Library, Fabrication Lab
  Hopkins Marine Station, Stanford University,
  120 Ocean View Blvd, Pacific Grove, CA 93950
  +1 831.915.9526 | rolander@stanford.edu
 *************************************************************************/

#define PROGRAM F("LEGO DNA Sequencer - Buzzer Unit Test")
#define VERSION F("Ver 0.1 2023-01-04")
#define DEBUG_OUTPUT 1
#define DEBUG_MODE   0


int buzzer = A0;//the pin of the active buzzer

void setup()
{
  pinMode(buzzer, OUTPUT); //initialize the buzzer pin as an output
}

void Beeps(int iNmbBeeps, int iMSLengthON, int iMSLengthOFF)
{
  for (int i=0; i<iNmbBeeps; i++)
  {
    digitalWrite(buzzer, HIGH);
    delay(iMSLengthON);//wait for 1ms
    digitalWrite(buzzer, LOW);
    delay(iMSLengthOFF);//wait for 1ms
  }
}

void loop()
{
  Beeps(10, 1, 1000);
  delay(5000);

  Beeps(1, 20, 0);
  delay(5000);

  Beeps(2, 20, 250);
  delay(5000);
```

```
  Beeps(3, 20, 200);
  delay(5000);
}
```

LCD and Rotary Encoder Unit Test

```
/*************************************************************************
  LEGO DNA Sequencer
  LCD and Button Unit Test

  Original Code:  2022-12-24

  Tom Rolander, MSEE
  Mentor for Circuit Design, Software, 3D Printing
  Miller Library, Fabrication Lab
  Hopkins Marine Station, Stanford University,
  120 Ocean View Blvd, Pacific Grove, CA 93950
  +1 831.915.9526 | rolander@stanford.edu

 *************************************************************************/

#define PROGRAM "LEGO DNA Sequencer - LCD and Rotary Encoder Unit Test"
#define VERSION "Ver 0.1 2022-12-24"

#define DEBUG_OUTPUT 1

#define LED_TCS34725  A0

#include <LiquidCrystal.h>
int Contrast = 75;
LiquidCrystal lcd(8, 7, 5, 4, 3, 2);

// Rotary Encoder Inputs
#define CLK A1
#define DT A2
#define SW 13

int counter = 0;
int currentStateCLK;
int lastStateCLK;
String currentDir ="";
unsigned long lastButtonPress = 0;

void setup()
{
  Serial.begin(115200);
  delay(1000);
  Serial.println();
  Serial.println(PROGRAM);
  Serial.println(VERSION);

  // Set encoder pins as inputs
```

```
  pinMode(CLK,INPUT);
  pinMode(DT,INPUT);
  pinMode(SW, INPUT_PULLUP);

  // Read the initial state of CLK
  lastStateCLK = digitalRead(CLK);

  //pinMode(13, INPUT_PULLUP);

  analogWrite(6, Contrast);
  lcd.begin(16, 2);

  pinMode (LED_TCS34725, OUTPUT);
  digitalWrite (LED_TCS34725, LOW);

  lcd.setCursor(0, 0);
//lcd.print("          1     ");
//lcd.print("0123456789012345");
  lcd.print("LCD Unit Testing");

  lcd.setCursor(0, 1);
  lcd.print(" Rotary Encoder ");

  delay(2000);

  lcd.setCursor(0, 0);
  lcd.print(" Rotary Encoder ");
  lcd.setCursor(0, 1);
  lcd.print("                ");
}
void loop()
{
  // Read the current state of CLK
  currentStateCLK = digitalRead(CLK);

  // If last and current state of CLK are different, then pulse occurred
  // React to only 1 state change to avoid double count
  if (currentStateCLK != lastStateCLK  && currentStateCLK == 1){

    // If the DT state is different than the CLK state then
    // the encoder is rotating CCW so decrement
    if (digitalRead(DT) != currentStateCLK) {
      counter --;
      currentDir ="CCW";
    } else {
      // Encoder is rotating CW so increment
      counter ++;
      currentDir ="CW";
    }
```

```
    Serial.print("Direction: ");
    Serial.print(currentDir);
    Serial.print(" | Counter: ");
    Serial.println(counter);

    lcd.setCursor(0, 1);
    lcd.print("                ");
    lcd.setCursor(0, 1);
    lcd.print("Counter = ");
    lcd.print(counter);
  }

  // Remember last CLK state
  lastStateCLK = currentStateCLK;

  // Read the button state
  int btnState = digitalRead(SW);

  //If we detect LOW signal, button is pressed
  if (btnState == LOW) {
    //if 50ms have passed since last LOW pulse, it means that the
    //button has been pressed, released and pressed again
    if (millis() - lastButtonPress > 50) {
      Serial.println("Button pressed!");
      lcd.setCursor(0, 1);
      lcd.print("Button pressed! ");
    }

    // Remember last button press event
    lastButtonPress = millis();
  }

  // Put in a slight delay to help debounce the reading
  delay(1);
}
```

Stepper Motor Unit Test

```
/*************************************************************************
  LEGO DNA Sequencer
  Stepper Motor Unit Test

  Original Code:  2022-12-24

  Tom Rolander, MSEE
  Mentor for Circuit Design, Software, 3D Printing
  Miller Library, Fabrication Lab
  Hopkins Marine Station, Stanford University,
  120 Ocean View Blvd, Pacific Grove, CA 93950
  +1 831.915.9526 | rolander@stanford.edu

 *************************************************************************/

#define PROGRAM "LEGO DNA Sequencer - Stepper Motor Unit Test"
#define VERSION "Ver 0.1 2022-12-24"

#define DEBUG_OUTPUT 1


//Includes the Arduino Stepper Library
#include <Stepper.h>

// Defines the number of steps per rotation
const int stepsPerRevolution = 2038;

#define STEPPER_PIN_1 9
#define STEPPER_PIN_2 10
#define STEPPER_PIN_3 11
#define STEPPER_PIN_4 12

// Creates an instance of stepper class
// Pins entered in sequence IN1-IN3-IN2-IN4 for proper step sequence
Stepper myStepper = Stepper(stepsPerRevolution, STEPPER_PIN_1, STEPPER_PIN_3,
STEPPER_PIN_2, STEPPER_PIN_4);

static int iSteps = 0;

void setup() {
  Serial.begin(115200);
  delay(1000);
  Serial.println();
  Serial.println(PROGRAM);
  Serial.println(VERSION);
```

```
  myStepper.setSpeed(10);

  Serial.println("Hit debugger Send to enter Commands");
  help();
}

void help()
{
  Serial.println("Command Line Mode:");
  Serial.println("  Fxxxx to go Forward  xxxx steps");
  Serial.println("  Rxxxx to go Reverse xxxx steps");
}

void loop()
{
  if (Serial.available() > 0)
  {
    char cBuffer[7];
    int nChars;
    int iValue = 0;

    nChars = Serial.readBytes(cBuffer, sizeof(cBuffer));
    cBuffer[nChars] = '\0';
    Serial.println(cBuffer);
    cBuffer[0] = toupper(cBuffer[0]);
    if (nChars != 5 ||
        (cBuffer[0] != 'F' && cBuffer[0] != 'R'))
    {
      help();
      return;
    }

    for (int i=1; i<5; i++)
    {
      iValue = (iValue * 10) + (cBuffer[i]-'0');
    }
    if (cBuffer[0] == 'R')
      iValue = 0 - iValue;

    iSteps = iSteps + iValue;
    myStepper.step(iValue);
    Serial.print("Position = ");
    Serial.println(iSteps);
  }
}
```

TCS34725 Color Sensor Unit Test

NOTE: To run this Unit Test you should remove the gear from the tray holder and move the tray with LEGOs manually under the color sensor.

```
/**************************************************************************
  LEGO DNA Sequencer
  TCS34725 Color Sensor Unit Test

  Original Code:  2022-12-24

  Tom Rolander, MSEE
  Mentor for Circuit Design, Software, 3D Printing
  Miller Library, Fabrication Lab
  Hopkins Marine Station, Stanford University,
  120 Ocean View Blvd, Pacific Grove, CA 93950
  +1 831.915.9526 | rolander@stanford.edu

 **************************************************************************/

#define PROGRAM "LEGO DNA Sequencer - TCS34725 Color Sensor Unit Test"
#define VERSION "Ver 0.1 2022-12-24"

#define DEBUG_OUTPUT 1

#include <Wire.h>
#include "Adafruit_TCS34725.h"

#define CLEAR   -1
#define RED     3
#define GREEN   2
#define BLUE    1
#define YELLOW  0

#define LED_TCS34725  A3

/* Initialise with specific int time and gain values */
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_16X);

void setup(void) {
  Serial.begin(115200);
  delay(1000);
  Serial.println();
  Serial.println(PROGRAM);
```

```
  Serial.println(VERSION);

  pinMode (LED_TCS34725, OUTPUT);
  digitalWrite (LED_TCS34725, LOW);
  if (tcs.begin()) {
    Serial.println("Found sensor");
    // LED off to prevent overheating
    tcs.setInterrupt(true);
  } else {
    Serial.println("No TCS34725 found ... check your connections");
    while (1);
  }

  Serial.println("Delay 5 seconds, then turn on LED light");
  delay(5000);
  digitalWrite (LED_TCS34725, HIGH);
  tcs.setInterrupt(false);
}

void loop(void) {
  uint16_t r, g, b, c;
  tcs.getRawData(&r, &g, &b, &c);
  Serial.print("Red: "); Serial.print(r, DEC); Serial.print(" ");
  Serial.print("Green: "); Serial.print(g, DEC); Serial.print(" ");
  Serial.print("Blue: "); Serial.print(b, DEC); Serial.print(" ");
  Serial.print("Clear: "); Serial.print(c, DEC); Serial.print(" ");
  Serial.println(" ");

  delay(1000);
}
```

Arduino Source Code

To compile and run the Arduino code the following libraries are needed for the Arduino IDE:

- Adafruit_TCS34725
- MD5 Note: this is only required if you enable `FILTER_BAD_WORDS`

```
/*************************************************************************
  LEGO DNA Sequencer
  Main Program

  Original Code:  2022-12-24

  Tom Rolander, MSEE
  Mentor for Circuit Design, Software, 3D Printing
  Miller Library, Fabrication Lab
  Hopkins Marine Station, Stanford University,
  120 Ocean View Blvd, Pacific Grove, CA 93950
  +1 831.915.9526 | rolander@stanford.edu

 *************************************************************************/

#define PROGRAM F("LEGO DNA Sequencer - Main Program")
#define VERSION F("Ver 0.9 2023-01-11")
#define PROGRAM_SHORT F("LEGO DNA Sqncr  ")
#define VERSION_SHORT F("Ver 0.9 01-11-23")

#define DEBUG_OUTPUT 0
#define DEBUG_MODE   0

#define FILTER_BAD_WORDS  0

#define FLORA 0

#define STATE_START       0
#define STATE_LOAD_TRAY   1
#define STATE_SEQUENCING  2

// Proc to reset the Arduino
void (* re_set)(void) = 0x00;

static bool bRemoteLCDSerial = false;

static int iState = STATE_START;
```

```
static int iStepPosition = 0;

static bool bAutomated = true;

#define BUZZER        A0

#define LED_TCS34725  A3

#if DEBUG_MODE
static bool bCommandLineMode = false;
#endif

// Rotary Encoder Inputs
#define CLK A1
#define DT A2
#define SW 13

static int iRotaryEncoder_Counter = 0;
static int iRotaryEncoder_CurrentStateCLK;
static int iRotaryEncoder_LastStateCLK;
static bool bRotaryEncoder_CurrentDirectionClockwise;
static unsigned long lastButtonPress = 0;

char sNameChars[40] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 <*";
//                     01234567890123456789012345678901234567890
//                     0         1         2         3
#define NMB_NAME_CHARS  39

char sNumberChars[13] = "0123456789<*";
//                       0123456789012
//                       0         1
#define NMB_NUMBER_CHARS  12


#if FILTER_BAD_WORDS
#include <MD5.h>
#define NMB_BAD_WORDS 10
const unsigned char ucBadWords[NMB_BAD_WORDS][16] =
{
  {0x84, 0xb9, 0x06, 0x03, 0x0d, 0x48, 0x52, 0x7c, 0xe0, 0x5c, 0x36, 0x82,
0x0f, 0x94, 0xc9, 0xa3},
  {0x01, 0x2e, 0x3b, 0x42, 0x29, 0xed, 0x77, 0x78, 0x85, 0x07, 0x44, 0xbb,
0x32, 0x7f, 0xa5, 0x54},
  {0x71, 0xb4, 0x74, 0xfa, 0x0a, 0x68, 0x85, 0x32, 0x55, 0x2a, 0x0a, 0x95,
0x46, 0x4a, 0x9a, 0x00},
  {0x4f, 0x79, 0x1f, 0x1f, 0xc2, 0xf1, 0xbc, 0x1a, 0x86, 0xab, 0xc3, 0x8d,
0x46, 0x78, 0x9b, 0x34},
```

```
  {0x00, 0x94, 0xcf, 0x00, 0x63, 0x7a, 0xee, 0x49, 0xb2, 0x3c, 0x4f, 0x44,
0xba, 0xaf, 0xe0, 0xde},
  {0x4a, 0x46, 0x6a, 0x30, 0x2e, 0x55, 0xdd, 0x2e, 0x4e, 0x8d, 0xf6, 0xde,
0xa7, 0xaa, 0xc3, 0x2f},
  {0xbe, 0x97, 0x62, 0x73, 0xca, 0x31, 0x79, 0xad, 0x77, 0x72, 0x63, 0xfb,
0x26, 0xac, 0x08, 0xd4},
  {0x27, 0x50, 0xfe, 0xdb, 0xda, 0x13, 0x41, 0xe9, 0x84, 0x2b, 0xdd, 0xb7,
0x44, 0xc8, 0x62, 0x03},
  {0xa0, 0x15, 0x4f, 0x58, 0xd6, 0xa4, 0x46, 0x1c, 0x9d, 0x35, 0x3d, 0x04,
0xd6, 0x3d, 0xb0, 0x8b},
  {0xf7, 0x3e, 0x87, 0x12, 0x37, 0x3e, 0xba, 0xab, 0xa7, 0xf8, 0xb6, 0x91,
0xa1, 0x92, 0x24, 0xeb}
};
#endif

#include <EEPROM.h>

#define EEPROM_LEGO_SIGNATURE F("LEGO");

// EEPROM state variables
#define EEPROM_SIZE 512

#define EEPROM_SIGNATURE                    0 // 000-003  Signature 'LEGO'
#define EEPROM_CLEAR_CHANNEL_THRESHHOLD     4 // 004-005
iRED_CHANNEL_THRESHHOLD
#define EEPROM_RED_CHANNEL_THRESHHOLD       6 // 006-007
iRED_CHANNEL_THRESHHOLD
#define EEPROM_BLUE_CHANNEL_THRESHHOLD      8 // 008-009
iBLUE_CHANNEL_THRESHHOLD
#define EEPROM_REMOTE_LCD_SERIAL           10 // 010-010  bRemoteLCDSerial
#define EEPROM_NUMBER_OF_LEGO_DNA          11 // 011-011  iNumberOfEEPROM
#define EEPROM_LEGO_DNA                    12 // 012-511  LEGO_DNAs

#define EEPROM_LEGO_DNA_MAX                4
#define EEPROM_LEGO_DNA_ENTRY_SIZE         28

#define LEGO_DNA_MAX  11

static int iNumberOfEEPROM = 0;

#define NMB_LEGO_BRICKS 10

int iBrickSequencing[NMB_LEGO_BRICKS][4];
char sDNASequence[NMB_LEGO_BRICKS + 1] = "";


int iLEGO_DNA_Number = 7;
char sLEGO_DNA_Sequence[LEGO_DNA_MAX][NMB_LEGO_BRICKS + 1] =
{ "CCGGTTAACC",
```

```
  "ATTGGTCATT",
  "TGCTCCTACA",
  "CACAATCTAC",
  "GCTCCCGGGT",
  "CAAATCTTAG",
  "CGTCTACCAA"
};
char sLEGO_DNA_Name[LEGO_DNA_MAX][17] =
{ " YYRRBBGGYY TEST",
  "  MAKO SHARK    ",
  "  ABALONE       ",
  "  SQUID         ",
  "  COPEPOD       ",
  "  SEA STAR      ",
  "  TUNA          "
};


#include <LiquidCrystal.h>
int Contrast = 75;
LiquidCrystal lcd(8, 7, 5, 4, 3, 2);

//Includes the Arduino Stepper Library
#include <Stepper.h>

// Defines the number of steps per rotation
const int stepsPerRevolution = 2038;

#define STEPPER_PIN_1 9
#define STEPPER_PIN_2 10
#define STEPPER_PIN_3 11
#define STEPPER_PIN_4 12

#define OFFSET_TO_FIRST_BRICK 460
#define STEPS_PER_LEGO_BRICK  550
#define STEPS_PER_AUTO_POSITION   25
#define OFFSET_AUTO_POSITION  200

// Creates an instance of stepper class
// Pins entered in sequence IN1-IN3-IN2-IN4 for proper step sequence
Stepper myStepper = Stepper(stepsPerRevolution, STEPPER_PIN_1, STEPPER_PIN_3,
STEPPER_PIN_2, STEPPER_PIN_4);

#if DEBUG_MODE
#define MAINTENANCE_NMB_OPS          10
#else
#define MAINTENANCE_NMB_OPS          9
#endif
#define MAINTENANCE_OP_CANCEL        0
```

```
#if DEBUG_MODE
#define MAINTENANCE_OP_COMMAND_LINE   1
#define MAINTENANCE_OP_UNLOAD_TRAY    2
#define MAINTENANCE_OP_LOAD_TRAY      3
#define MAINTENANCE_OP_POSITION_TRAY  4
#define MAINTENANCE_OP_ZERO_NEW_DNA   5
#define MAINTENANCE_OP_BUZZER_OFF     6
#define MAINTENANCE_OP_BUZZER_ON      7
#define MAINTENANCE_OP_ADVANCED       8
#define MAINTENANCE_OP_VERSION_INFO   9
#endif
#define MAINTENANCE_OP_UNLOAD_TRAY    1
#define MAINTENANCE_OP_LOAD_TRAY      2
#define MAINTENANCE_OP_POSITION_TRAY  3
#define MAINTENANCE_OP_ZERO_NEW_DNA   4
#define MAINTENANCE_OP_BUZZER_OFF     5
#define MAINTENANCE_OP_BUZZER_ON      6
#define MAINTENANCE_OP_ADVANCED       7
#define MAINTENANCE_OP_VERSION_INFO   8

char sMaintenanceOp[MAINTENANCE_NMB_OPS][17] =
{ "  CANCEL        ",
#if DEBUG_MODE
  "  COMMAND LINE  ",
#endif
  "  UNLOAD TRAY   ",
  "  LOAD TRAY     ",
  "  POSITION TRAY ",
  "  ZERO NEW DNA  ",
  "  BUZZER OFF    ",
  "  BUZZER ON     ",
  "  ADVANCED      ",
  "  VERSION INFO  "
};

#define ADVANCED_NMB_OPS            9

#define ADVANCED_OP_CANCEL          0
#define ADVANCED_OP_CLEAR_CHANNEL   1
#define ADVANCED_OP_RED_CHANNEL     2
#define ADVANCED_OP_BLUE_CHANNEL    3
#define ADVANCED_OP_AUTO_CHANNEL    4
#define ADVANCED_OP_AUTO_POSITION   5
#define ADVANCED_OP_REMOTE_LCD_ON   6
#define ADVANCED_OP_REMOTE_LCD_OFF  7
#define ADVANCED_OP_REBOOT          8

char sAdvancedOp[ADVANCED_NMB_OPS][17] =
{ "  CANCEL        ",
```

```
  "  CLEAR CHANNEL ",
  "  RED CHANNEL   ",
  "  BLUE CHANNEL  ",
  "  AUTO CHANNEL  ",
  "  AUTO POSITION ",
  "  REMOTE LCD ON ",
  "  REMOTE LCD OFF",
  "  REBOOT        "
};


static bool bBuzzer = true;

#include <Wire.h>
#include "Adafruit_TCS34725.h"

#define CLEAR   -1
#define RED     'G'
#define GREEN   'A'
#define BLUE    'T'
#define YELLOW  'C'

/* Initialise with specific int time and gain values */
Adafruit_TCS34725 tcs = Adafruit_TCS34725(TCS34725_INTEGRATIONTIME_50MS,
TCS34725_GAIN_16X);

#if FLORA
#define CLEAR_CHANNEL_THRESHHOLD  130
#define RED_CHANNEL_THRESHHOLD  35
#define BLUE_CHANNEL_THRESHHOLD  30
#else
#define CLEAR_CHANNEL_THRESHHOLD  900
#define RED_CHANNEL_THRESHHOLD  170
#define BLUE_CHANNEL_THRESHHOLD  220
#endif

static int iCLEAR_CHANNEL_THRESHHOLD = CLEAR_CHANNEL_THRESHHOLD;
static int iRED_CHANNEL_THRESHHOLD = RED_CHANNEL_THRESHHOLD;
static int iBLUE_CHANNEL_THRESHHOLD = BLUE_CHANNEL_THRESHHOLD;

void setup() {
  Serial.begin(115200);
  delay(1000);

#if DEBUG_OUTPUT
  Serial.println();
  Serial.println(PROGRAM);
  Serial.println(VERSION);
#endif
```

```
  pinMode(BUZZER, OUTPUT); //initialize the buzzer pin as an output

  // Set encoder pins as inputs
  pinMode(CLK, INPUT);
  pinMode(DT, INPUT);
  pinMode(SW, INPUT_PULLUP);

  // Read the initial state of CLK
  iRotaryEncoder_LastStateCLK = digitalRead(CLK);

  analogWrite(6, Contrast);
  lcd.begin(16, 2);

  pinMode (LED_TCS34725, OUTPUT);
  digitalWrite (LED_TCS34725, LOW);

  if (tcs.begin()) {
#if DEBUG_OUTPUT
    Serial.println(F("Found sensor"));
#endif
    // LED off to prevent overheating
    tcs.setInterrupt(true);

  } else {
    Serial.println(F("No TCS34725 found ... check your connections"));
    while (1);
  }

  myStepper.setSpeed(10);

  char cBuffer[5];
  int nChars;
  long lStartTime = millis();

#if DEBUG_MODE
  bCommandLineMode = true;
  lcd.setCursor(0, 0);
  lcd.print(F("LEGO DNA Sqncr  "));
  lcd.setCursor(0, 1);
  lcd.print(F("Debugger Console"));
#else
  StartSequencer();
#endif

#if DEBUG_OUTPUT
  for (int i = 0; i < 128; i++)
  {
    int iByte = EEPROM[i];
```

```
    if (i % 16 == 0)
      Serial.println("");
    if (iByte < 16)
      Serial.print(F("0"));
    Serial.print(iByte, HEX);
    Serial.print(F(" "));
  }
  Serial.println("");
#endif

  LoadEEPROM();

  Beeps(3, 10, 200);  // 3 Beeps, 10MS ON, 200MS OFF
}

void LoadEEPROM()
{
  if (EEPROM[EEPROM_SIGNATURE + 0] == 'L' &&
      EEPROM[EEPROM_SIGNATURE + 1] == 'E' &&
      EEPROM[EEPROM_SIGNATURE + 2] == 'G' &&
      EEPROM[EEPROM_SIGNATURE + 3] == 'O')
  {
    iCLEAR_CHANNEL_THRESHHOLD =
readUnsignedIntFromEEPROM(EEPROM_CLEAR_CHANNEL_THRESHHOLD);
    iRED_CHANNEL_THRESHHOLD =
readUnsignedIntFromEEPROM(EEPROM_RED_CHANNEL_THRESHHOLD);
    iBLUE_CHANNEL_THRESHHOLD =
readUnsignedIntFromEEPROM(EEPROM_BLUE_CHANNEL_THRESHHOLD);
    bRemoteLCDSerial= EEPROM[EEPROM_REMOTE_LCD_SERIAL];
    iNumberOfEEPROM = EEPROM[EEPROM_NUMBER_OF_LEGO_DNA];

    int index;
    int iOffset = EEPROM_LEGO_DNA;
    for (index = 0; index < iNumberOfEEPROM; index++)
    {
      for (int i = 0; i < 11; i++)
      {
        sLEGO_DNA_Sequence[iLEGO_DNA_Number][i] = EEPROM[iOffset];
        iOffset = iOffset + 1;
      }
      for (int i = 0; i < 17; i++)
      {
        sLEGO_DNA_Name[iLEGO_DNA_Number][i] = EEPROM[iOffset];
        iOffset = iOffset + 1;
      }
#if DEBUG_OUTPUT
      Serial.println(F("EEPROM Data"));
      Serial.print(F("Index "));
      Serial.print(iLEGO_DNA_Number);
```

```
      Serial.print(F(" ["));
      Serial.print(&sLEGO_DNA_Sequence[iLEGO_DNA_Number][0]);
      Serial.print(F("] ["));
      Serial.print(&sLEGO_DNA_Name[iLEGO_DNA_Number][0]);
      Serial.println(F("]"));
#endif
      iLEGO_DNA_Number = iLEGO_DNA_Number + 1;
    }
  }
  else
  {
#if DEBUG_OUTPUT
    Serial.println(F("EEPROM LEGO not found"));
#endif
    SetupEEPROM();
  }
#if DEBUG_OUTPUT
  Serial.println(F("EEPROM LEGO found"));
  Serial.print(F("bRemoteLCDSerial = "));
  Serial.println(bRemoteLCDSerial);
  Serial.print(F("iNumberOfEEPROM = "));
  Serial.println(iNumberOfEEPROM);
  Serial.print(F("Clear Channel Threshhold = "));
  Serial.println(iCLEAR_CHANNEL_THRESHHOLD);
  Serial.print(F("Red   Channel Threshhold = "));
  Serial.println(iRED_CHANNEL_THRESHHOLD);
  Serial.print(F("Blue  Channel Threshhold = "));
  Serial.println(iBLUE_CHANNEL_THRESHHOLD);
#endif
}

void writeUnsignedIntIntoEEPROM(int address, unsigned int number)
{
  EEPROM[address] = number >> 8;
  EEPROM[address + 1] = number & 0xFF;
}

unsigned int readUnsignedIntFromEEPROM(int address)
{
  return (EEPROM[address] << 8) + EEPROM[address + 1];
}

void SetupEEPROM()
{
  EEPROM[EEPROM_SIGNATURE + 0] = 'L';
  EEPROM[EEPROM_SIGNATURE + 1] = 'E';
  EEPROM[EEPROM_SIGNATURE + 2] = 'G';
  EEPROM[EEPROM_SIGNATURE + 3] = 'O';
```

```
  writeUnsignedIntIntoEEPROM(EEPROM_CLEAR_CHANNEL_THRESHHOLD,
iCLEAR_CHANNEL_THRESHHOLD);
  writeUnsignedIntIntoEEPROM(EEPROM_RED_CHANNEL_THRESHHOLD,
iRED_CHANNEL_THRESHHOLD);
  writeUnsignedIntIntoEEPROM(EEPROM_BLUE_CHANNEL_THRESHHOLD,
iBLUE_CHANNEL_THRESHHOLD);
  EEPROM[EEPROM_REMOTE_LCD_SERIAL] = bRemoteLCDSerial;
  EEPROM[EEPROM_NUMBER_OF_LEGO_DNA] = iNumberOfEEPROM;


#if DEBUG_OUTPUT
  Serial.println(F("EEPROM LEGO initialized"));
#endif

}
void UpdateLCD(char cColor, int index)
{
  char lcd1[17] = "Sequencing DNA  ";
  char lcd2[17] = "                ";
  lcd2[index] = cColor;
  for(int i=0; i<index; i++)
    lcd2[i] = sDNASequence[i];
  RemoteLCDSerial(lcd1, lcd2);
  lcd.print(cColor);
}

char GetLEGOColor(int index)
{
  char cRetcode = '\0';

  uint16_t r, g, b, c;

  tcs.getRawData(&r, &g, &b, &c);

  iBrickSequencing[index][0] = r;
  iBrickSequencing[index][1] = g;
  iBrickSequencing[index][2] = b;
  iBrickSequencing[index][3] = c;

#if DEBUG_OUTPUT
  Serial.print(iStepPosition, DEC); Serial.print(F(","));
  Serial.print(r, DEC); Serial.print(F(","));
  Serial.print(g, DEC); Serial.print(F(","));
  Serial.print(b, DEC); Serial.print(F(","));
  Serial.print(c, DEC); Serial.print(F(","));
#endif

  if (c >= iCLEAR_CHANNEL_THRESHHOLD)
  {
```

```
    UpdateLCD(YELLOW, index);
    cRetcode = YELLOW;
#if DEBUG_OUTPUT
    Serial.println(F("Y,C"));
#endif
  }
  else if (r >= iRED_CHANNEL_THRESHHOLD)
  {
    UpdateLCD(RED, index);
    cRetcode = RED;
#if DEBUG_OUTPUT
    Serial.println(F("R,G"));
#endif
  }
  else if (b >= iBLUE_CHANNEL_THRESHHOLD)
  {
    UpdateLCD(BLUE, index);
    cRetcode = BLUE;
#if DEBUG_OUTPUT
    Serial.println(F("B,T"));
#endif
  }
  else
    /* if (g > b && g > r) */
  {
    UpdateLCD(GREEN, index);
    cRetcode = GREEN;
#if DEBUG_OUTPUT
    Serial.println(F("G,A"));
#endif
  }

  return (cRetcode);
}

void StartSequencer()
{
  lcd.setCursor(0, 0);
  lcd.print(F("LEGO DNA Sqncr  "));
  lcd.setCursor(0, 1);
  lcd.print(F(" Push Button    "));
  F_RemoteLCDSerial(F("LEGO DNA Sqncr  "),F(" Push Button    "));
}

void help()
{
#if DEBUG_OUTPUT
  Serial.println(F("Command Line Mode:"));
  Serial.println(F("  Lxxxx to go Left  xxxx steps"));
```

```
  Serial.println(F("  Rxxxx to go Right xxxx steps"));
  Serial.println(F("  Sxxxx set RPM to xxxx"));
  Serial.println(F("  C to Read Color Sensor"));
  Serial.println(F("  K Read Color Sensor until Send"));
  Serial.println(F("  1 Color Sensor Light ON"));
  Serial.println(F("  0 Color Sensor Light OFF"));
  Serial.println(F("  M Manual Sequencer"));
  Serial.println(F("  A Automated Sequencer"));
  Serial.println(F("  E to Start Sequencer"));
  Serial.println(F("  Z zero out added DNAs"));
#endif
}

bool GetButtonPressed()
{
  // Read the button state
  int btnState = digitalRead(SW);

  //If we detect LOW signal, button is pressed
  if (btnState == LOW)
  {
    //if 50ms have passed since last LOW pulse, it means that the
    //button has been pressed, released and pressed again
    if (millis() - lastButtonPress > 50) {
      //      lcd.setCursor(0, 1);
      //      lcd.print("Button pressed! ");
    }

    // Wait for button to be released
    while (digitalRead(SW) == LOW)
      ;

    // Remember last button press event
    lastButtonPress = millis();
    return (true);
  }
  return (false);
}

bool GetYesOrNo(bool bInitialResponse, char *sQuestion)
{
  bool bResponse = bInitialResponse;

  char lcd1[17] = "                ";
  char lcd2[17];
  strncpy(lcd1, sQuestion, strlen(sQuestion));

  lcd.setCursor(0, 0);
  lcd.print(F("                "));
```

```
lcd.setCursor(0, 0);
lcd.print(sQuestion);
lcd.setCursor(0, 1);
if (bInitialResponse == false)
{
  lcd.print(F("[NO]  YES       "));
  strcpy(lcd2,"[NO]  YES       ");
}
else
{
  lcd.print(F(" NO  [YES]      "));
  strcpy(lcd2," NO  [YES]      ");
}
if (bRemoteLCDSerial)
{
  delay(500);
  RemoteLCDSerial(lcd1, lcd2);
}

while (GetButtonPressed() == false)
{
  delay(1);
  if (CheckRotaryEncoder())
  {
    if (bRotaryEncoder_CurrentDirectionClockwise == false)
    {
      if (bResponse != false)
      {
        lcd.setCursor(0, 1);
        lcd.print(F("[NO]  YES       "));
        strcpy(lcd2,"[NO]  YES       ");
      }
      bResponse = false;
    }
    else
    {
      if (bResponse != true)
      {
        lcd.setCursor(0, 1);
        lcd.print(F(" NO  [YES]      "));
        strcpy(lcd2," NO  [YES]      ");
      }
      bResponse = true;
    }
    if (bRemoteLCDSerial)
    {
      delay(500);
      RemoteLCDSerial(lcd1, lcd2);
    }
```

```
   }
  }
  while (digitalRead(SW) == LOW)
    ;

  return (bResponse);
}

////////////////////////////////////////////
int GetNumber(int iCurrentValue, char *iName, int iOffsetEEPROM)
{
  lcd.setCursor(0, 0);
  lcd.print(F("Enter "));
  lcd.write((byte) 0xff);
  lcd.print(F(" to end  "));
  lcd.setCursor(0, 1);
  lcd.print(F("                "));
  lcd.setCursor(0, 1);
  lcd.print(iName);
  lcd.print(F("="));

  int iOffset = strlen(iName) + 1;
  char sNewNumber[5] = "";
  char cNextChar;

  sprintf(sNewNumber, "%d", iCurrentValue);
  lcd.print(sNewNumber);

  int index = strlen(sNewNumber);
  while (index < 5)
  {
    cNextChar = GetNextNumber(index + iOffset);
    if (cNextChar == '\0')
    {
      break;
    }
    if (cNextChar == '<')
    {
      if (index > 0)
      {
        lcd.setCursor(index + iOffset, 1);
        lcd.print(' ');
        index = index - 1;
        continue;
      }
    }
    sNewNumber[index] = cNextChar;
    index = index + 1;
    sNewNumber[index] = '\0';
```

```
  }

  if (strlen(sNewNumber) != 0)
  {
    int iValue = 0;
    for (int i = 0; i < 4; i++)
    {
      if (sNewNumber[i] == '\0')
        break;
      iValue = (iValue * 10) + (sNewNumber[i] - '0');
    }
    if (iValue != iCurrentValue)
    {
      char sTemp[17] = "SET ";
      strcat(sTemp, iName);
      strcat(sTemp, "=");
      strcat(sTemp, sNewNumber);
      if (GetYesOrNo(false, sTemp))
      {
        writeUnsignedIntIntoEEPROM(iOffsetEEPROM, iValue);
        return (iValue);
      }
    }
  }
  return (-1);
}
////////////////////////////////////////////

bool CheckRotaryEncoder()
{
  bool bResponse = true;

  // Read the current state of CLK
  iRotaryEncoder_CurrentStateCLK = digitalRead(CLK);

  // If last and current state of CLK are different, then pulse occurred
  // React to only 1 state change to avoid double count
  if (iRotaryEncoder_CurrentStateCLK != iRotaryEncoder_LastStateCLK  &&
iRotaryEncoder_CurrentStateCLK == 1) {

    // If the DT state is different than the CLK state then
    // the encoder is rotating CCW so decrement
    if (digitalRead(DT) != iRotaryEncoder_CurrentStateCLK) {
      iRotaryEncoder_Counter --;
      bRotaryEncoder_CurrentDirectionClockwise = false;
    } else {
      // Encoder is rotating CW so increment
      iRotaryEncoder_Counter ++;
      bRotaryEncoder_CurrentDirectionClockwise = true;
```

```
    }
  }
  else
  {
    bResponse = false;
  }

  // Remember last CLK state
  iRotaryEncoder_LastStateCLK = iRotaryEncoder_CurrentStateCLK;
  delay(10);
  return (bResponse);
}

void loop()
{
#if DEBUG_MODE
  if (bCommandLineMode)
  {
    if (Serial.available() > 0)
    {
      char cBuffer[7];
      int nChars;
      int iValue = 0;

      nChars = Serial.readBytes(cBuffer, sizeof(cBuffer));
      Serial.print(F("nChars = "));
      Serial.println(nChars);
      cBuffer[nChars] = '\0';
      Serial.print(F("cBuffer = "));
      Serial.println(cBuffer);

      cBuffer[0] = toupper(cBuffer[0]);

      if (nChars == 1)
      {
        switch (cBuffer[0])
        {
          case 'Z':
            iNumberOfEEPROM = 0;
            iLEGO_DNA_Number = 6;
            for (int i = 0; i < 512; i++)
              EEPROM[i] = '\0';
            SetupEEPROM();
            break;

          case 'E':
            StartSequencer();
            iState = STATE_START;
            bCommandLineMode = false;
```

```
      return;

    case 'M':
      bAutomated = false;
      break;

    case 'A':
      bAutomated = true;
      break;

    case 'C':
      GetLEGOColor(0);
      break;

    case 'K':
      while (Serial.available() <= 0)
      {
        GetLEGOColor(0);
        delay(500);
      }
      nChars = Serial.readBytes(cBuffer, sizeof(cBuffer));
      help();
      break;

    case '1':
      tcs.setInterrupt(false);
      digitalWrite (LED_TCS34725, HIGH);
      break;

    case '0':
      tcs.setInterrupt(true);
      digitalWrite (LED_TCS34725, LOW);
      break;

    default:
      Serial.println(F("Unrecognized command!"));
      help();
      break;
  }
  return;
}
if (nChars != 5 ||
    (cBuffer[0] != 'L' && cBuffer[0] != 'R' && cBuffer[0] != 'S'))
{
  help();
  return;
}

for (int i = 1; i < 5; i++)
```

```
      {
        iValue = (iValue * 10) + (cBuffer[i] - '0');
      }
      Serial.print(F("Value = "));
      if (cBuffer[0] == 'R')
        iValue = 0 - iValue;
      Serial.println(iValue);

      if (cBuffer[0] == 'S')
        myStepper.setSpeed(iValue);
      else
      {
        MoveTray(iValue);
        GetLEGOColor(0);
      }
    }
    return;
  }
#endif

  if (iState == STATE_START)
  {
    if (GetButtonPressed())
    {
      //      while (digitalRead(SW) == LOW)
      //        ;
      iState = STATE_LOAD_TRAY;
    }
    else
    {
      return;
    }
  }

  if (iState == STATE_LOAD_TRAY)
  {
    F_RemoteLCDSerial(F("Load LEGO Tray  "), F(" Push Button    "));
    lcd.setCursor(0, 0);
    lcd.print(F("Load LEGO Tray  "));
    lcd.setCursor(0, 1);
    lcd.print(F(" Push Button    "));

    Beeps(1, 5, 0);  // 1 Beep, 5MS ON, 0MS OFF

    while (GetButtonPressed() == false)
    {
      if (CheckRotaryEncoder())
      {
        lcd.setCursor(0, 0);
```

```
        lcd.print(F("Select Option:  "));

        int iMaintenanceOp = MAINTENANCE_OP_CANCEL;

        while (true)
        {
          lcd.setCursor(0, 1);
          lcd.print(sMaintenanceOp[iMaintenanceOp]);

          if (GetButtonPressed())
            break;

          int iCounter = iRotaryEncoder_Counter;
          if (CheckRotaryEncoder())
          {
            if (iRotaryEncoder_Counter > iCounter)
            {
              iMaintenanceOp++;
              if (iMaintenanceOp  >= MAINTENANCE_NMB_OPS)
                iMaintenanceOp = MAINTENANCE_OP_CANCEL;
            }
            else
            {
              iMaintenanceOp--;
              if (iMaintenanceOp < 0)
                iMaintenanceOp = MAINTENANCE_OP_VERSION_INFO;
            }
          }
        }

        switch (iMaintenanceOp)
        {
          case MAINTENANCE_OP_CANCEL:
#if DEBUG_OUTPUT
            Serial.println(F("Cancel"));
#endif
            StartSequencer();
            iState = STATE_START;
#if DEBUG_MODE
            bCommandLineMode = false;
#endif
            break;

#if DEBUG_MODE
          case MAINTENANCE_OP_COMMAND_LINE:
            Serial.println(F("Enter Command Line Mode"));
            bCommandLineMode = true;
            break;
#endif
```

```
          case MAINTENANCE_OP_UNLOAD_TRAY:
#if DEBUG_OUTPUT
            Serial.println(F("Unload Tray"));
#endif
            myStepper.step(-4000);
            iStepPosition = 0;
            break;

          case MAINTENANCE_OP_LOAD_TRAY:
#if DEBUG_OUTPUT
            Serial.println(F("Load Tray"));
#endif
            myStepper.step(1640);
            iStepPosition = 0;
            break;

          case MAINTENANCE_OP_POSITION_TRAY:
#if DEBUG_OUTPUT
            Serial.println(F("Position Tray"));
#endif
            {
              int iCounter = iRotaryEncoder_Counter;
              while (GetButtonPressed() == false)
              {
                if (CheckRotaryEncoder())
                {
#if DEBUG_OUTPUT
#if 0
                  Serial.print(F("iCounter = "));
                  Serial.print(iCounter);
                  Serial.print(F("  iRotaryEncoder_Counter = "));
                  Serial.println(iRotaryEncoder_Counter);
#endif
#endif
                  if (iRotaryEncoder_Counter > iCounter)
                    MoveTray(50);
                  else
                    MoveTray(-50);
                  iCounter = iRotaryEncoder_Counter;
                }
              }
            }
            break;

          case MAINTENANCE_OP_ZERO_NEW_DNA:
            iNumberOfEEPROM = 0;
            iLEGO_DNA_Number = 6;
            for (int i = EEPROM_NUMBER_OF_LEGO_DNA; i < 512; i++)
```

```
            EEPROM[i] = '\0';
          SetupEEPROM();
          break;

        case MAINTENANCE_OP_BUZZER_ON:
          bBuzzer = true;
          break;

        case MAINTENANCE_OP_BUZZER_OFF:
          bBuzzer = false;
          break;

        case MAINTENANCE_OP_ADVANCED:
          Advanced();
          break;

        case MAINTENANCE_OP_VERSION_INFO:
          lcd.setCursor(0, 0);
          lcd.print(PROGRAM_SHORT);
          lcd.setCursor(0, 1);
          lcd.print(VERSION_SHORT);
          while (GetButtonPressed() == false &&
                 CheckRotaryEncoder() == false)
          {
            delay(100);
          }
          break;
      }

      return;
    }
  }
  iState = STATE_SEQUENCING;
}

lcd.setCursor(0, 0);
lcd.print(F("Sequencing DNA  "));
lcd.setCursor(0, 1);
lcd.print(F("                "));

F_RemoteLCDSerial(F("Sequencing DNA  "),F("                "));

lcd.setCursor(0, 1);


// STATE_SEQUENCING
tcs.setInterrupt(false);
digitalWrite (LED_TCS34725, HIGH);
```

```
  myStepper.setSpeed(10);
  iStepPosition = 0;
  MoveTray(OFFSET_TO_FIRST_BRICK);

  sDNASequence[NMB_LEGO_BRICKS] = '\0';

#if DEBUG_OUTPUT
  Serial.println("");
  Serial.println(F("---------- CSV ----------"));
  Serial.println(F("P,R,G,B,Y,M,DNA"));
#endif

  for (int i = 0; i < NMB_LEGO_BRICKS; i++)
  {
    if (bAutomated == false)
    {
      lcd.setCursor(0, 0);
      lcd.print(F("Position LEGO # "));
      lcd.print(i);
      lcd.setCursor(0, 1);
      lcd.print(F(" Push Button    "));

      while (GetButtonPressed() == false)
        ;
    }

    delay(100); // Delay a little for visual effect
    sDNASequence[i] = GetLEGOColor(i);

    if (i < 9)
    {
      Beeps(1, 2, 0);   // 1 Beep, 2MS ON, 0MS OFF
      MoveTray(STEPS_PER_LEGO_BRICK);
    }
    else
    {
      tcs.setInterrupt(true);
      digitalWrite (LED_TCS34725, LOW);

      lcd.setCursor(0, 0);
      lcd.print(F("Unloading tray  "));

      char lcd1[17] = "Unloading tray  ";
      char lcd2[17] = "                ";
      for(int i=0; i<NMB_LEGO_BRICKS; i++)
        lcd2[i] = sDNASequence[i];
      RemoteLCDSerial(lcd1, lcd2);

      Beeps(2, 5, 250);  // 2 Beeps, 5MS ON, 250MS OFF
```

```
      myStepper.setSpeed(15);
      MoveTray(-5410);

#if DEBUG_MODE
      Serial.print(F("["));
      Serial.print(sDNASequence);
      Serial.println(F("]"));
      for (int i = 0; i < NMB_LEGO_BRICKS; i++)
      {
        for (int j = 0; j < 4; j++)
        {
          Serial.print(iBrickSequencing[i][j]);
          Serial.print(F(" "));
        }
        Serial.println("");
      }
#endif
      myStepper.setSpeed(10);
      iState = STATE_LOAD_TRAY;

      if ((strcmp(sDNASequence, "AAAAAAAAAA") == 0) ||
          (strcmp(sDNASequence, "CCCCCCCCCC") == 0) ||
          (strcmp(sDNASequence, "GGGGGGGGGG") == 0) ||
          (strcmp(sDNASequence, "TTTTTTTTTT") == 0))
      {
        lcd.setCursor(0, 0);
        lcd.print(F("EMPTY TRAY ERROR"));
        lcd.setCursor(0, 1);
        lcd.print(F("  PUSH BUTTON   "));

        Beeps(4, 10, 200);  // 4 Beeps, 10MS ON, 200MS OFF

        while (GetButtonPressed() == false)
          ;
      }
      else
      {
        int index;
        for (index = 0; index < iLEGO_DNA_Number; index++)
        {
          if (strcmp(sDNASequence, sLEGO_DNA_Sequence[index]) == 0)
            break;
        }

        if (index < iLEGO_DNA_Number)
        {
          lcd.setCursor(0, 0);
          lcd.print(F("Successful match"));
```

```
          lcd.setCursor(0, 1);
          lcd.print(sLEGO_DNA_Name[index]);

          char lcd1[17] = "Successful match";
          char lcd2[17] = "";
          strcpy(lcd2, sLEGO_DNA_Name[index]);
          RemoteLCDSerial(lcd1, lcd2);

          Beeps(3, 5, 200);  // 3 Beeps, 5MS ON, 200MS OFF
        }
        else
        {
          lcd.setCursor(0, 0);
          lcd.print(F("DNA not a match "));
          lcd.setCursor(0, 1);
          lcd.print(F("  UNKNOWN DNA!  "));
          F_RemoteLCDSerial(F("DNA not a match "), F("  UNKNOWN DNA!  "));

          Beeps(4, 10, 200);  // 4 Beeps, 10MS ON, 200MS OFF
        }

        while (GetButtonPressed() == false)
          ;

        if (index >= iLEGO_DNA_Number)
        {
          if (GetYesOrNo(false, "Add to Database?"))
          {
#if DEBUG_OUTPUT
            Serial.println(F("Add to Database!"));
#endif
            lcd.setCursor(0, 0);
            lcd.print(F("Name:  * to end "));
            lcd.setCursor(0, 1);
            lcd.print(F("                "));
            F_RemoteLCDSerial(F("Name:  * to end "), F("                "));

            char sNewName[17] = "";
            char cNextChar;

            int index = 0;
            while (index < 16)
            {
              cNextChar = GetNextChar(index);
              if (cNextChar == '\0')
              {
                break;
              }
              if (cNextChar == '<')
```

```
            {
              if (index > 0)
              {
                lcd.setCursor(index, 1);
                lcd.print(' ');
                index = index - 1;
                continue;
              }
            }
            sNewName[index] = cNextChar;
            index = index + 1;
            sNewName[index] = '\0';

            char lcd1[17] = "Name:  * to end ";
            char lcd2[17] = "                ";
            strncpy(lcd2, sNewName, strlen(sNewName));
            RemoteLCDSerial(lcd1, lcd2);
          }
          for (; index < 16; index++)
          {
            sNewName[index] = ' ';
          }
          sNewName[16] = '\0';

          if (strcmp(sNewName, "                ") != 0)
            if (GetYesOrNo(false, sNewName))
            {
              if (iNumberOfEEPROM >= (EEPROM_LEGO_DNA_MAX - 1))
              {
                lcd.setCursor(0, 0);
                lcd.print(F("EEPROM DB full! "));
                lcd.setCursor(0, 1);
                lcd.print(F(" Push Button    "));

                while (GetButtonPressed() == false)
                  ;
              }
              else
              {
#if DEBUG_OUTPUT
                Serial.print(F("Adding to Database ["));
                Serial.print(sDNASequence);
                Serial.print(F("] = ["));
                Serial.print(sNewName);
                Serial.println(F("]"));
#endif

#if FILTER_BAD_WORDS
                // Check for Bad Word
```

```
                    bool bBadWord = false;
                    for (int i = 0; i < NMB_BAD_WORDS; i++)
                    {
                      char sQuad[5] = "    ";
                      for (int j = 0; j < 13; j++)
                      {
                        int k;
                        strncpy(sQuad, &sNewName[j], 4);
                        //generate the MD5 hash for our string
                        unsigned char* hash = MD5::make_hash(sQuad);

                        for (k = 0; k < 16; k++)
                        {
                          if (hash[k] == ucBadWords[i][k])
                            continue;
                          break;
                        }
                        if (k == 16)
                          bBadWord = true;
                        free(hash);
                        if (bBadWord)
                          break;
                      }
                      if (bBadWord)
                        break;
                    }
                    if (bBadWord)
                    {
                      lcd.setCursor(0, 0);
                      lcd.print(F("Bad Word in Name"));
                      lcd.setCursor(0, 1);
                      lcd.print(F(" Push Button    "));
#if DEBUG_OUTPUT
                      Serial.println(F("BAD WORD!"));
#endif
                      while (GetButtonPressed() == false)
                        ;
                    }
                    else
#endif
                    {
                      // Now add to DATABASE
                      int iOffset = (EEPROM_LEGO_DNA_ENTRY_SIZE *
iNumberOfEEPROM);
                      for (int i = 0; i <= 11; i++)
                      {
                        sLEGO_DNA_Sequence[iLEGO_DNA_Number][i] =
sDNASequence[i];
                        EEPROM[EEPROM_LEGO_DNA + iOffset] = sDNASequence[i];
```

```
                    iOffset = iOffset + 1;
                  }
                  iOffset = iOffset - 1;
                  for (int i = 0; i <= 17; i++)
                  {
                    sLEGO_DNA_Name[iLEGO_DNA_Number][i] = sNewName[i];
                    EEPROM[EEPROM_LEGO_DNA + iOffset] = sNewName[i];
                    iOffset = iOffset + 1;
                  }

                  iNumberOfEEPROM = iNumberOfEEPROM + 1;
                  EEPROM[EEPROM_NUMBER_OF_LEGO_DNA] = iNumberOfEEPROM;

                  iLEGO_DNA_Number = iLEGO_DNA_Number + 1;
                }
              }
            }
          }
        }
        iState = STATE_LOAD_TRAY;
      }
    }
  }

void Advanced()
{
  while (GetButtonPressed() == false)
  {
    while (true)
    {
      lcd.setCursor(0, 0);
      lcd.print(F("Select Option:  "));

      int iAdvancedOp = ADVANCED_OP_CANCEL;

      while (true)
      {
        lcd.setCursor(0, 1);
        lcd.print(sAdvancedOp[iAdvancedOp]);

        if (GetButtonPressed())
          break;

        int iCounter = iRotaryEncoder_Counter;
        if (CheckRotaryEncoder())
        {
          if (iRotaryEncoder_Counter > iCounter)
          {
```

```
            iAdvancedOp++;
            if (iAdvancedOp  >= ADVANCED_NMB_OPS)
              iAdvancedOp = ADVANCED_OP_CANCEL;
          }
          else
          {
            iAdvancedOp--;
            if (iAdvancedOp < 0)
              iAdvancedOp = ADVANCED_OP_REBOOT;
          }
        }
      }

      int iResult;
      switch (iAdvancedOp)
      {
        case ADVANCED_OP_CANCEL:
          return;

        case ADVANCED_OP_CLEAR_CHANNEL:
          iResult = GetNumber(iCLEAR_CHANNEL_THRESHHOLD, "CLEAR",
EEPROM_CLEAR_CHANNEL_THRESHHOLD);
          if (iResult > 0)
            iCLEAR_CHANNEL_THRESHHOLD = iResult;
          break;

        case ADVANCED_OP_RED_CHANNEL:
          iResult = GetNumber(iRED_CHANNEL_THRESHHOLD, "RED",
EEPROM_RED_CHANNEL_THRESHHOLD);
          if (iResult > 0)
            iRED_CHANNEL_THRESHHOLD = iResult;
          break;

        case ADVANCED_OP_BLUE_CHANNEL:
          iResult = GetNumber(iBLUE_CHANNEL_THRESHHOLD, "BLUE",
EEPROM_BLUE_CHANNEL_THRESHHOLD);
          if (iResult > 0)
            iBLUE_CHANNEL_THRESHHOLD = iResult;
          break;

        case ADVANCED_OP_AUTO_CHANNEL:
          if (sDNASequence[0] == '\0')
          {
            lcd.setCursor(0, 0);
            lcd.print(F("SCAN TEST LEGOS!"));
            lcd.setCursor(0, 1);
            lcd.print(F("PUSH BUTTON     "));
            while (GetButtonPressed() == false)
            {
```

```
            delay(100);
          }
          break;
        }
        if (GetYesOrNo(false, "SCAN YYRRBBGGYY?"))
        {
          char sTemp[17] = "";
          sprintf(sTemp,"C%d R%d B%d",iCLEAR_CHANNEL_THRESHHOLD,
iRED_CHANNEL_THRESHHOLD,iBLUE_CHANNEL_THRESHHOLD);
          lcd.setCursor(0, 0);
          lcd.print(F("CURRENT CHANNELS"));
          lcd.setCursor(0, 1);
          lcd.print(F("                "));
          lcd.setCursor(0, 1);
          lcd.print(sTemp);
          while (GetButtonPressed() == false)
          {
            delay(100);
          }
          int iClear = iBrickSequencing[0][3];
          int iRed = iBrickSequencing[2][0];
          int iBlue = iBrickSequencing[4][2];
          if (iBrickSequencing[1][3] < iClear)
            iClear = iBrickSequencing[1][3];
          if (iBrickSequencing[3][0] < iRed)
            iRed = iBrickSequencing[3][0];
          if (iBrickSequencing[5][2] < iBlue)
            iBlue = iBrickSequencing[5][2];
          iClear = iClear - ((iClear * 1)/10);
          iRed = iRed - ((iRed * 1)/10);
          iBlue = iBlue - ((iBlue * 1)/10);
          sprintf(sTemp,"C%d R%d B%d?", iClear, iRed, iBlue);
          if (GetYesOrNo(false, sTemp))
          {
            iCLEAR_CHANNEL_THRESHHOLD = iClear;
            iRED_CHANNEL_THRESHHOLD = iRed;
            iBLUE_CHANNEL_THRESHHOLD = iBlue;
            writeUnsignedIntIntoEEPROM(EEPROM_CLEAR_CHANNEL_THRESHHOLD,
iCLEAR_CHANNEL_THRESHHOLD);
            writeUnsignedIntIntoEEPROM(EEPROM_RED_CHANNEL_THRESHHOLD,
iRED_CHANNEL_THRESHHOLD);
            writeUnsignedIntIntoEEPROM(EEPROM_BLUE_CHANNEL_THRESHHOLD,
iBLUE_CHANNEL_THRESHHOLD);
          }
        }
        break;

      case ADVANCED_OP_AUTO_POSITION:
      {
```

```
            lcd.setCursor(0, 0);
            lcd.print(F("USE TEST LEGO   "));
            lcd.setCursor(0, 1);
            lcd.print(F(" Push Button    "));
            while (GetButtonPressed() == false)
            {
              delay(100);
            }
            // Advance 25 steps at a time until red found and then red lost
            lcd.setCursor(0, 0);
            lcd.print(F("AUTO POSITIONING"));
            lcd.setCursor(0, 1);
            lcd.print(F("                "));

            tcs.setInterrupt(false);
            digitalWrite (LED_TCS34725, HIGH);

            MoveTray(OFFSET_TO_FIRST_BRICK);
            MoveTray(STEPS_PER_LEGO_BRICK);

            iStepPosition = 0;
            bool bWaitingForRED = true;
            int iStartPositionRED = 0;
            int iEndPositionRED = 0;
            while (iStepPosition < 4000)
            {
              MoveTray(STEPS_PER_AUTO_POSITION);
              lcd.setCursor(0, 1);
              char cColor = GetLEGOColor(0);
              if (bWaitingForRED == true &&
                  cColor == RED)
              {
//Serial.print("RED = "); Serial.println(iStepPosition);
                iStartPositionRED = iStepPosition;
                bWaitingForRED = false;
              }
              else
              if (bWaitingForRED == false &&
                  cColor != RED)
              {
//Serial.print("BLUE = "); Serial.println(iStepPosition);
                iEndPositionRED = iStepPosition;
                break;
              }
            }
            tcs.setInterrupt(true);
            digitalWrite (LED_TCS34725, LOW);

            if (iStartPositionRED == 0 || iEndPositionRED == 0)
```

```
            {
              lcd.setCursor(0, 0);
              lcd.print(F("AUTO POSITIONING"));
              lcd.setCursor(0, 1);
              lcd.print(F("   FAILURE      "));
              while (GetButtonPressed() == false)
              {
                delay(100);
              }
              break;
            }
            else
            {
              int iSecondREDStart = (iEndPositionRED + iStartPositionRED)/2;
//Serial.print("2nd RED = "); Serial.println(iSecondREDStart);
              MoveTray(iSecondREDStart-iStepPosition);
              int iMove = 0 - (STEPS_PER_LEGO_BRICK *2);
//Serial.print("iMove = "); Serial.println(iMove);
              iMove = iMove - OFFSET_TO_FIRST_BRICK;
//Serial.print("iMove = "); Serial.println(iMove);
              iMove = iMove - OFFSET_AUTO_POSITION;
//Serial.print("iMove = "); Serial.println(iMove);
              MoveTray(iMove);
            }
          break;
        }

        case ADVANCED_OP_REMOTE_LCD_ON:
          bRemoteLCDSerial = true;
          EEPROM[EEPROM_REMOTE_LCD_SERIAL] = bRemoteLCDSerial;
          re_set();
          break;

        case ADVANCED_OP_REMOTE_LCD_OFF:
          bRemoteLCDSerial = false;
          EEPROM[EEPROM_REMOTE_LCD_SERIAL] = bRemoteLCDSerial;
          re_set();
          break;

        case ADVANCED_OP_REBOOT:
          re_set();
          break;

        default:
          break;
      }
    }
  }
}
```

```
char GetNextChar(int index)
{
  int iCurrentCharIndex = NMB_NAME_CHARS - 1;
  char cCurrentChar = sNameChars[NMB_NAME_CHARS - 1];

  lcd.setCursor(index, 1);
  lcd.print(cCurrentChar);

  int iCurrentCounter = iRotaryEncoder_Counter;
  while (GetButtonPressed() == false)
  {
    if (CheckRotaryEncoder())
    {
      if (iCurrentCounter != iRotaryEncoder_Counter)
      {
        if (iRotaryEncoder_Counter > iCurrentCounter)
        {
          iCurrentCharIndex = iCurrentCharIndex + (iRotaryEncoder_Counter -
iCurrentCounter);
          if (iCurrentCharIndex >= NMB_NAME_CHARS)
            iCurrentCharIndex = 0;
        }
        else
        {
          iCurrentCharIndex = iCurrentCharIndex - (iCurrentCounter -
iRotaryEncoder_Counter);
          if (iCurrentCharIndex < 0)
            iCurrentCharIndex = NMB_NAME_CHARS - 1;
        }
        lcd.setCursor(index, 1);
        cCurrentChar = sNameChars[iCurrentCharIndex];
        lcd.print(cCurrentChar);
        iCurrentCounter = iRotaryEncoder_Counter;
      }
    }
  }
  if (cCurrentChar == '*')
    cCurrentChar = '\0';
  return (cCurrentChar);
}

char GetNextNumber(int index)
{
  int iCurrentCharIndex = NMB_NUMBER_CHARS - 1;
  char cCurrentChar = sNumberChars[NMB_NUMBER_CHARS - 1];

  lcd.setCursor(index, 1);
  lcd.print(cCurrentChar);
```

```
  int iCurrentCounter = iRotaryEncoder_Counter;
  while (GetButtonPressed() == false)
  {
    if (CheckRotaryEncoder())
    {
      if (iCurrentCounter != iRotaryEncoder_Counter)
      {
        if (iRotaryEncoder_Counter > iCurrentCounter)
        {
          iCurrentCharIndex = iCurrentCharIndex + (iRotaryEncoder_Counter -
iCurrentCounter);
          if (iCurrentCharIndex >= NMB_NUMBER_CHARS)
            iCurrentCharIndex = 0;
        }
        else
        {
          iCurrentCharIndex = iCurrentCharIndex - (iCurrentCounter -
iRotaryEncoder_Counter);
          if (iCurrentCharIndex < 0)
            iCurrentCharIndex = NMB_NUMBER_CHARS - 1;
        }
        lcd.setCursor(index, 1);
        cCurrentChar = sNumberChars[iCurrentCharIndex];
        lcd.print(cCurrentChar);
        iCurrentCounter = iRotaryEncoder_Counter;
      }
    }
  }
  if (cCurrentChar == '*')
    cCurrentChar = '\0';
  return (cCurrentChar);
}

void Beeps(int iNmbBeeps, int iMSLengthON, int iMSLengthOFF)
{
  if (bBuzzer == false)
    return;

  for (int i = 0; i < iNmbBeeps; i++)
  {
    digitalWrite(BUZZER, HIGH);
    delay(iMSLengthON);//wait for 1ms
    digitalWrite(BUZZER, LOW);
    delay(iMSLengthOFF);//wait for 1ms
  }
}

void MoveTray(int iSteps)
```

```
{
  iStepPosition = iStepPosition + iSteps;
  myStepper.step(iSteps);
}

#if 0
void TestNameEntry()
{
  Serial.println(F("Add to Database!"));
  lcd.setCursor(0, 0);
  lcd.print(F("Name:  "));
  lcd.write((byte) 0xff);
  lcd.print(F(" to end "));
  lcd.setCursor(0, 1);
  lcd.print(F("                "));

  char sNewName[17] = "";
  char cNextChar;

  int index = 0;
  while (index < 16)
  {
    cNextChar = GetNextChar(index);
    if (cNextChar == '\0')
    {
      break;
    }
    if (cNextChar == '<')
    {
      if (index > 0)
      {
        lcd.setCursor(index, 1);
        lcd.print(' ');
        index = index - 1;
        continue;
      }
    }
    sNewName[index] = cNextChar;
    index = index + 1;
    sNewName[index] = '\0';
  }
  for (; index < 16; index++)
  {
    sNewName[index] = ' ';
  }
  sNewName[16] = '\0';
  Serial.println(sNewName);
}
#endif
```

```
bool F_RemoteLCDSerial(const __FlashStringHelper *lcd1, const
__FlashStringHelper *lcd2)
{
  if (bRemoteLCDSerial == false)
    return (false);

  Serial.print('[');
  Serial.print(lcd1);
  Serial.println("");
  Serial.print(lcd2);
  Serial.print(']');
}

bool RemoteLCDSerial(char *lcd1, char *lcd2)
{
  if (bRemoteLCDSerial == false)
    return (false);

  Serial.print('[');
  Serial.print(lcd1);
  Serial.println("");
  Serial.print(lcd2);
  Serial.print(']');
}
```

Remote LCD Python Source Code

There are two versions of the Remote LCD program which use different GUI libraries:

RemoteLCD_PySimpleGUI

```
"""
/*************************************************************************
  RemoteLDC_PySimpleGUI
    The purpose of this Python program is to remotely display the
  contents of an Arduino 1602 LCD.  It was created to facilitate
  a large group of persons watching the LEGO DNA Sequencer.

  Original Code:  2023-01-08
  Revision:       2022-01-10

  Tom Rolander, MSEE
  Mentor, Circuit Design & Software
  Miller Library, Fabrication Lab
  Hopkins Marine Station, Stanford University,
  120 Ocean View Blvd, Pacific Grove, CA 93950
  +1 831.915.9526 | rolander@stanford.edu

 *************************************************************************/

"""
Program = "RemoteLCD_PySimpleGUI"
Version = "Ver 0.3"
RevisionDate = "2023-01-16"

import sys
import os
import time
import serial
import argparse
from types import NoneType
import PySimpleGUI as sg

def getbackgroundcolor(x):
    if chr(x) == 'A':
        return 'green'
    if chr(x) == 'C':
        return 'yellow'
```

```
    if chr(x) == 'G':
        return 'red'
    if chr(x) == 'T':
        return 'blue'

def gettextcolor(x):
    if chr(x) == 'C':
        return 'black'
    else:
        return 'white'

print (Program, Version, RevisionDate)

parser = argparse.ArgumentParser("RemoteLCD")
parser.add_argument('--comport', type=str, required=False, help="COM port")
parser.add_argument('--showports', required=False, choices=('True','False'))
args = parser.parse_args()

if type(args.showports) is not NoneType:
    os.system("python -m serial.tools.list_ports")
    exit(0)

if type(args.comport) is NoneType:
    print ("--comport <COMPORT> is required!")
    print ("--showports True   will show available comports")
    exit(1)

print ("Connecting to LEGO DNA Sequencer on Arduino on Port",args.comport)

SerialObj = serial.Serial(args.comport) # COMxx   format on Windows
#SerialObj = serial.Serial('COM4') # COMxx   format on Windows
                                   # ttyUSBx format on Linux

SerialObj.baudrate = 115200  # set Baud rate to 9600
SerialObj.bytesize = 8     # Number of data bits = 8
SerialObj.parity   ='N'    # No parity
SerialObj.stopbits = 1     # Number of Stop bits = 1


sg.theme('Black')
layout = [  [sg.Text('LEGO DNA Sqncr', font=("Courier",120), key='LINE1')],
            [sg.Text(' Push Button', font=("Courier",120), key='LINE2'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL0'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL1'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL2'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL3'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL4'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL5'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL6'),
```

```
            sg.Text(' ', font=("Courier",120), key='LINE2-COL7'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL8'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL9'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL10'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL11'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL12'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL13'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL14'),
            sg.Text(' ', font=("Courier",120), key='LINE2-COL15')]
        ]

# Create the Window
window = sg.Window('LEGO DNA Sequencer', layout, margins=(50,250)).Finalize()
window.Maximize()

# Event Loop to process "events"
while True:
    event, values = window.read(timeout=10)
    if event == sg.WIN_CLOSED: # if user closes window
        break
    if SerialObj.in_waiting != 0:
        letter = SerialObj.read()
        if letter == b'[':
            lcd1 = SerialObj.read(16)
            SerialObj.read(2)   #Ignore the '\r\n'
            lcd2 = SerialObj.read(16)
            SerialObj.read(1)   #Ignore the ']'
            window['LINE1'].update(str(lcd1, 'UTF-8'))
            if str(lcd1, 'UTF-8') == "Sequencing DNA  " or str(lcd1, 'UTF-8')
== "Unloading tray  ":
                window['LINE2'].update('  ')
                for x in range(10):
                    window['LINE2-COL'+str(x)].update(background_color='black')
                    window['LINE2-COL'+str(x)].update(' ')
                n = range(10, 16)
                for x in n:
                    window['LINE2-COL'+str(x)].update(background_color='black')
                    window['LINE2-COL'+str(x)].update(' ')
                for x in range(10):
                    backgroundcolor = getbackgroundcolor(lcd2[x])

window['LINE2-COL'+str(x)].update(background_color=backgroundcolor)
                    textcolor = gettextcolor(lcd2[x])
                    window['LINE2-COL'+str(x)].update(text_color=textcolor)
                    window['LINE2-COL'+str(x)].update(font=("Courier",120,
'bold'))
                    window['LINE2-COL'+str(x)].update(chr(lcd2[x]))
            else:
                window['LINE2'].update(background_color='black')
```

```
                window['LINE2'].update(text_color='white')
                window['LINE2'].update(font=("Courier",120, 'normal'))
                window['LINE2'].update(str(lcd2, 'UTF-8')+'        ')

window.close()
```

RemoteLCD_tkinter

```
"""
/*************************************************************************
  RemoteLDC_tkinter
    The purpose of this Python program is to remotely display the
  contents of an Arduino 1602 LCD.  It was created to facilitate
  a large group of persons watching the LEGO DNA Sequencer.

  Original Code:  2023-01-08
  Revision:       2022-01-10

  Tom Rolander, MSEE
  Mentor, Circuit Design & Software
  Miller Library, Fabrication Lab
  Hopkins Marine Station, Stanford University,
  120 Ocean View Blvd, Pacific Grove, CA 93950
  +1 831.915.9526 | rolander@stanford.edu

 *************************************************************************/

"""
Program = "RemoteLCD_tkinter"
Version = "Ver 0.1"
RevisionDate = "2023-01-10"

import sys
import os
import tkinter as tk
import time
import serial
import argparse
from types import NoneType


def Draw():
    global text
    frame=tk.Frame(root,bd=1)
    frame.place(x=10,y=10)
    frame.pack(side="left")
    text=tk.Label(frame,text='', font=("Courier", 120))
    text.pack()
    text.configure(text= "  Welcome\n  LEGO DNA Sqncr")

def Refresher():
    global text
```

```
    if SerialObj.in_waiting != 0:
        letter = SerialObj.read()
        if letter == b'[':
            lcd1 = SerialObj.read(16)
            SerialObj.read(2)   #Ignore the '\r\n'
            lcd2 = SerialObj.read(16)
            SerialObj.read(1)   #Ignore the ']'
            #text.configure(text="0123456789012345\n" + time.asctime())
            text.configure(text= "  " + str(lcd1, 'UTF-8') + "\n  " + str(lcd2,
'UTF-8'))
            #text.configure(text= lcd1 + "\n" + lcd2)
    root.after(1000, Refresher) # every second...

print (Program, Version, RevisionDate)

parser = argparse.ArgumentParser("RemoteLCD")
parser.add_argument('--comport', type=str, required=False, help="COM port")
parser.add_argument('--showports', required=False, choices=('True','False'))
args = parser.parse_args()

if type(args.showports) is not NoneType:
    os.system("python -m serial.tools.list_ports")
    exit(0)

if type(args.comport) is NoneType:
    print ("--comport <COMPORT> is required!")
    print ("--showports True   will show available comports")
    exit(1)

print ("Connecting to LEGO DNA Sequencer on Arduino on Port",args.comport)

SerialObj = serial.Serial(args.comport) # COMxx   format on Windows

#SerialObj = serial.Serial('COM4') # COMxx   format on Windows
                                   # ttyUSBx format on Linux

SerialObj.baudrate = 115200  # set Baud rate to 9600
SerialObj.bytesize = 8     # Number of data bits = 8
SerialObj.parity   ='N'    # No parity
SerialObj.stopbits = 1     # Number of Stop bits = 1

root=tk.Tk()
root.title("LEGO DNA Sequencer")
root.geometry("1540x340")
root.attributes('-fullscreen', True)
label = tk.Label(root, text=" LEGO DNA Sequencer -- Remote LCD ",
font=("Courier", 48), bg="black", fg="white")
label.pack()
```

```
Draw()
Refresher()
root.mainloop()
```

Credits

This project has been clearly inspired by the LEGO DNA Sequencer created by Sam Nicholls and Tom Blanchard at Monster DNA Lab. It is a very well conceived open source project.

Peter Shum was a Post Doc at the Hopkins Marine Station of Stanford University and implemented the MonsterLab Sequencer for a Hopkins Open House Day in 2018.

Dr. Amanda Whitmire is the Head Librarian & Bibliographer of the Miller Library at the Hopkins Marine Station and has provided the facilities of her Fabrication Lab for this project.