**AL5D Robot arm project.**

SHOULD WE MAKE THE CONTACT PLATE OURSELVES OUT OF COPPER?

# Problems to be solved

* Implement kinematic algorithm to move robot arm to location of resistor to be sorted.
* Read the value of a resistor using the 2 probes on the modified gripper design.
* Turn electromagnet on.
* Based on the returned value, tell the robot arm to sort the resistor into the appropriate container.
* Turn electromagnet off.
* Implement safety features (master stop).
* Implement Calibration features.
* Possibility of datalogging.

# Technical considerations.

Choice of embedded system.

First the documentation for the Lynxmotion ‘SSC-32u’ arm controller board was analysed. The control system uses a UART serial connection to receive control commands and to send some return data back as well. The UART connection could be configured to a desired baud rate and then controlled from any other device with a serial port.

A basic serial message to move all the servos on the robot arm can be sent in the form of a string, beginning with # and ending with a carriage return <cr>

#0P1425#1P1500#2P1500#3P700T2000<cr>

The first part of the command:

#0P1425

#0 means that servo 0 is being controlled, then p1425 sets that servo to that position. So 1P, 2P, 3P is then controlling another servo.

The last command:

T2000

Sets the travel speed in milliseconds, so in this example the motors will allow move to their desired positions in 2 seconds.

The absolute (maximum) control command was measured to be no greater than 34 Bytes, therefore that is the total length of a message in our system and memory allocation considerations going forward were based on that measurement.

To read the value of resistors, the gripper design is based on 2 probes which act like ohm meter probes to read the resistance of the gripper and an electro-magnet which then picks the resistor up for it to be sorted to a predefined location. For reliable measurements, the PCB to read the value must be mounted as close to the probes as possible, as noise and resistance could affect the analogue reading over longer distances.

Due to the reliability of serial communication over the working envelope of the robot arm:

(<https://www.digikey.com/en/articles/uarts-ensure-reliable-long-haul-industrial-communications#:~:text=With%20appropriate%20line%20drivers%2C%20a,485%20or%20RS%2D422%20interfaces>.)

*“With appropriate line drivers, a UART can work over long distances: from 15 meters (m) for the RS-232 serial data bus to 1000 m for RS-485 or RS-422 interfaces” [22 May 2019]*

The proposed design utilises an ohm meter on the arm itself, to read the value of the resistor and increase the reliability of that analogue reading, then send the processed reading to another “master” microcontroller which can then activate the robot arm. This will mean making 2 PCBs, one which could be mounted on the gripper and one to remain at the base acting as a master controller.

Given that the SSC-32u controller used serial messages to control the board, a serial connection would be needed for communication with the gripper and considering that the system may want to have flexibility to communicate with other devices such as a desktop computer alongside the arm controller. A microcontroller

Due to the recent upsets in the global semiconductor supply chain, certain, more popular microcontrollers, such as STM32F103vgt6 (core of the STM “blue pill”) and Atmega328p (core of the “Arduino Nano”) have become increasingly difficult to source, with back order waiting times sometimes over a year. An alternative solution was required here to bypass this issue.

Two contenders for the role in this case were the ESP32 or the RP2040 (Pi Pico/ Pico W). Seeking to keep the costs low and looking to develop further on a relatively new platform, the RP2040 was chosen due to its low cost and wide availability. The Pi Pico W development board was then set up to allow development of the system using breadboards, protoboard and in-house PCBs, made using a PCB router.

Diagram

Description automatically generated

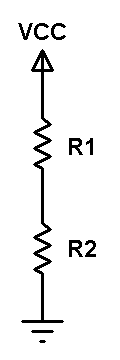
# Ohm Meter

The project required a system of reading values of resistance into a microprocessor, the value then denoting what box to sort the resistor in to.

Utilising an analogue to digital converter (ADC), resistance could be read from an unknown resistor by creating a voltage divider with that and another resistor of known value. Reading the voltage from that divider will give the resistance.

The resistance of known value is not random and does need to be within a relative range of that which is being tested. Thus, a feature which is seen in most multimeters, an auto ranging feature must be implemented to allow a wide range of values to be read without manually changing the resistor of known value.

The auto ranging design was chosen, using five PNP transistors, acting as switches between different known values of resistance, due to its small layout size compared to some other methods such as using an analogue multiplexer and some peripheral logic.



Applying voltage divider equation to the above circuit we get:  
Voltage across resistor R1: VR1 = VCC x R1/(R1 + R2)  
Voltage across resistor R2: VR2 = VCC x R2/(R1 + R2)

Assume that the resistance of R1 is known and R2 is unknown, by applying voltage divider equation we can easily get the value of R2 by measuring the voltage across it where: R2 = VR2 x R1/(VCC – VR2). If VR2 is equal to VCC then R2 = infinite.

**How to measure R2 voltage (VR2):**  
The voltage is an analog signal and we can measure it using ADC (Analog-to-Digital) converter which converts analog data into digital data. The Arduino board with ATmega328P microcontroller (UNO, Nano, Mini …) has a built-in 10-bit ADC module where VCC (+5V) is represented by 1023 and 0 volts is represented by 0 (if VDD is already used as positive voltage reference).

**Do we need fixed voltage reference?**  
Actually, I think there’s no need for a fixed voltage reference and adding it will not give a big change in the results. Because the voltage divider positive terminal (VCC) and the ADC positive reference voltage are connected to the same point and any change of this point voltage will affect both the voltage divider output and the ADC module.

**Multirange ohmmeter:**  
To get a good value of the resistor to be measured, a multirange ohmmeter should be used.  
The multi-range ohmmeter may also use voltage divider technique with different values of resistor R1. The following image shown a simple diagram of multi-range ohmmeter:

**[Diagram, schematic

Description automatically generated](https://simple-circuit.com/wp-content/uploads/2020/01/multirange-ohmmeter.png)**

The multi-range ohmmeter is based on a mechanical switching device with 1 input and multi output. At any time just one known resistor (R11, R12 … R1n) is connected in series with the unknown resistor R2.

**Auto-ranging ohmmeter:**  
The auto-range ohmmeter operates the same as the multi-range ohmmeter except that the mechanical switching device is replaced by electronic switches such as transistors driven by an intelligent device (microprocessor, microcontroller …) that decides which switch is closed according to the resistance value of the unknown resistor.

Project Hardware Required:

* Arduino board
* 5 x PNP transistor ([**2SA1015**](http://www.unisonic.com.tw/datasheet/2SA1015.pdf), [**2N3906**](https://www.onsemi.com/pub/Collateral/2N3906-D.PDF) …)
* 2 x 100nF ceramic capacitor
* 5 x 4.7k ohm resistor
* 2M ohm resisor
* 100k ohm resistor
* 10k ohm resistor
* 1k ohm resistor
* 100 ohm resistor
* 330 ohm resistor
* 10k ohm variable resistor or potentiometer
* Breadboard
* Jumper wires

Arduino autoranging ohmmeter circuit:  
Project circuit diagram is shown below.  
Note that all the grounded terminals should be connected together.

Diagram, schematic

Description automatically generated

The resistor which we want to measure is connected between Arduino analog channel 1 (A1) and GND as shown in the above circuit diagram (Ohmmeter Probes). The 100nF capacitor is used to stabilize the voltage across the unknown resistor.

The five PNP transistors Q1 ~ Q5 are general purpose transistors and they are of the same type 2SA1015, 2N3906 or equivalent. They are used as electronic switches for our autoranging ohmmeter.  
The emitter terminal of the five transistors are connected together to 5V pin of the Arduino board.

The collector of each transistor is connected to a different resistor. At any time there is only one ON transistor whereas the others are OFF. For better accuracy, each of the 5 resistors should have tolerance of 1% or lower.  
Also, each transistor base is connected to Arduino digital pin through 4.7k ohm resistor.

The AREF pin of the Arduino board should be connected to 5V pin with 100nF capacitor between it and GND pin.

MY WORDS:

*In the circuit diagram, the resistor being measured is connected between the A1 pin and GND on an Arduino board. A 100nF capacitor is included to steady the voltage across the unknown resistor. The circuit also includes five PNP transistors (Q1-Q5), which are all of the same type (2SA1015, 2N3906 or equivalent) and function as electronic switches for the ohmmeter. The emitters of the five transistors are all connected to the 5V pin on the Arduino board. Each transistor's collector is connected to a different resistor, with only one transistor turned on at a time. For improved accuracy, it is recommended that each of the five resistors has a tolerance of 1% or lower. The base of each transistor is connected to an Arduino digital pin via a 4.7k ohm resistor. Additionally, the AREF pin on the Arduino board should be connected to the 5V pin and have a 100nF capacitor between it and the GND pin.*

# Master Controller

Since the master controller will require 2 separate UART connections, one for connecting to the ohm meter and another for connecting to the robot arm unit, using an ATMEGA328p based system would not be sufficient. Either an ESP32 (3 UART) or RP2040 (2 UART) based system would overcome this obstacle, both systems can be programmed using Arduino or Platform IO, as well as “bare metal” using their own respective SDKs.

# Electromagnet

A picture containing graphical user interface

Description automatically generated

<https://wiki.keyestudio.com/KS0320_Keyestudio_Electromagnet_Module_(Black_and_Environmental-friendly)>

* Working Voltage: DC 5V
* Working current: 0.3A (maximum)
* Maximum power: 3W

The above electromagnet can be controlled using a PWM signal on its SIG pin, the larger the PWM value the larger the magnetic field (up to 1kg, this needs to be tested).

V & V

Managing bugs:

50% of microelectronics development is V&V as opposed to design. (lol wayyyy more)

Validation are we building the right system?

Verification ae we building the system right? (correctly?)

Diagram

Description automatically generated

Diagram

Description automatically generated

Graphical user interface, application

Description automatically generated

Requirements matrix

Talk about gripper iterations.

First one was too long, so mounted the PCB at 90 degrees to previous design.

Second design did not have correct mounting holes for the current al5d arm.

Third iteration did not have a solid enough infill and was very weak and broke on the first stress test. Potentially should add some sort of composite analysis here? This can be very basic.

Talk about initial building of system using Atmega328p on Arduino Nano, then upgrading the design to the raspberry pi pico, the pico and the RP2040 being much cheaper and readily available where as the Atmega328p is currently difficult to get hold of. Then finally settling on the Pico W with included WiFi capability to allow for possibility of exploring datalogging using JSON and an Amazon AWS server with get/puts.

Talk about the iterations of the base design and the justification for that in terms of allowing the system to be built by anyone using the cloned github repository.

Code snippets and pseudo code to justify the mathematical kinematic model.

Notes make 2nd gantt chart

PCB designs.

Take pictures of stuff

Summaries

What could we do better? Add datalogging and wifi control, better conveyor system. Have our own motor control board so as not to use the SCC-32u

Each row in this array is in the order, (x, y, resistance in ohms), and can be configured to any value between 100R to 2M

long BoxCoordinatesXY[10][3] = {

    // Row 1

    {0, 260, 1000},       //1k

    {0, 330, 4700},       //4.7k

    {0, 390, 10000},      //10k

    // Row 2

    {80, 270, 20000},     //20k

    {80, 330, 47000},     //47k

    {80, 390, 86000},     //86k

    // Row 3

    {160, 270, 100000},   //100k

    {160, 330, 180000},   //180k

    // Row 4

    {240, 270, 470000},   //470k

    {240, 330, 1000000}}; //1M

Calibration function

void startAndCalibrateArm()

{

  Serial.println("#0P1425#1P1500#2P1500#3P700T2000\r"); // starting position

  Serial.println("#2PO-100\r");                 // calibrate

  // DEBUG

  delay(2000);

}

Kinematics string function  
char \*getArmCode(float x, float y, float z)

{

  float grip\_angle\_d = 68;

  // grip angle in radians for use in calculations

  float grip\_angle\_r = radians(grip\_angle\_d);

  // Adjustment so that the gripper hits the floor at z=0;

  z += 70;

  // Base angle and radial distance from x,y coordinates

  // float d = sqrt( x\*x + y\*y );    //distance from base center to x,y coordinates

  float phi = 90.00;

  float bas\_angle\_r = atan2(x, y);

  float L = sqrt((x \* x) + (y \* y));

  // L is y coordinate for the arm

  L -= cos(phi) \* GRIPPER;

  y = L;

  // Grip offsets calculated based on grip angle

  float grip\_off\_z = (sin(grip\_angle\_r)) \* GRIPPER;

  float grip\_off\_y = (cos(grip\_angle\_r)) \* GRIPPER;

  // Wrist position

  float wrist\_z = (z - grip\_off\_z) - BASE\_HGT;

  wrist\_z -= sin(phi) \* GRIPPER;

  float wrist\_y = y - grip\_off\_y;

  // Shoulder to wrist distance ( AKA sw )

  float s\_w = (wrist\_z \* wrist\_z) + (wrist\_y \* wrist\_y);

  float s\_w\_sqrt = sqrt(s\_w);

  // s\_w angle to ground

  // float a1 = atan2( wrist\_y, wrist\_z );

  float a1 = atan2(wrist\_z, wrist\_y);

  // s\_w angle to SHOULDER

  float a2 = acos(((sh\_sq - el\_sq) + s\_w) / (2 \* SHOULDER \* s\_w\_sqrt));

  // shoulder angle

  float shl\_angle\_r = a1 + a2;

  float shl\_angle\_d = degrees(shl\_angle\_r);

  // elbow angle

  float elb\_angle\_r = acos((sh\_sq + el\_sq - s\_w) / (2 \* SHOULDER \* ELBOW));

  float elb\_angle\_d = degrees(elb\_angle\_r);

  float elb\_angle\_dn = -(180.0 - elb\_angle\_d);

  // wrist angle

  float wri\_angle\_d = (grip\_angle\_d - elb\_angle\_dn) - shl\_angle\_d;

  // Servo pulses

  float bas\_servopulse = 1500.0 - ((degrees(bas\_angle\_r)) \* 11.11);

  float shl\_servopulse = 1500.0 + ((shl\_angle\_d - 90.0) \* 6.6);

  float elb\_servopulse = 1500.0 - ((elb\_angle\_d - 90.0) \* 6.6);

  float wri\_servopulse = 1500 - (wri\_angle\_d \* 11.1);

  // Set servo pulses

  // create string for output

  // Added some offsets here, not sure if they are correct, they seem to work though

  sprintf(output, "#0P%ld#1P%ld#2P%ld#3P%ldT2000\r", ftl(1500 - bas\_servopulse + 500), ftl(shl\_servopulse), ftl(elb\_servopulse), ftl(wri\_servopulse + 400));

  return output;

}