

Translation Stage Design Deliverables

Name: Jason Cao

ME240 Wednesday

Team laboratory project

Abstract

The goal for this experiment is to precisely control a machine to move in a 2D translation motion. Linear Variable Differential Transformers (LVDTs) is used in this experiment, and we will design a MATLAB code to manipulate the LVDTs to move into the desired positions. We will use the university logo “R” as a target image shown in Figure 1.a and the vertices of the “R” will be the desired positions. To draw the “R” logo, we will calibrate the LVDTs and convert the pixel position into real distance unit and then convert the distance to voltage and finally convert the voltage to degrees which is the input that LVDTs can recognize. The final figure is shown in Figure 1.b.

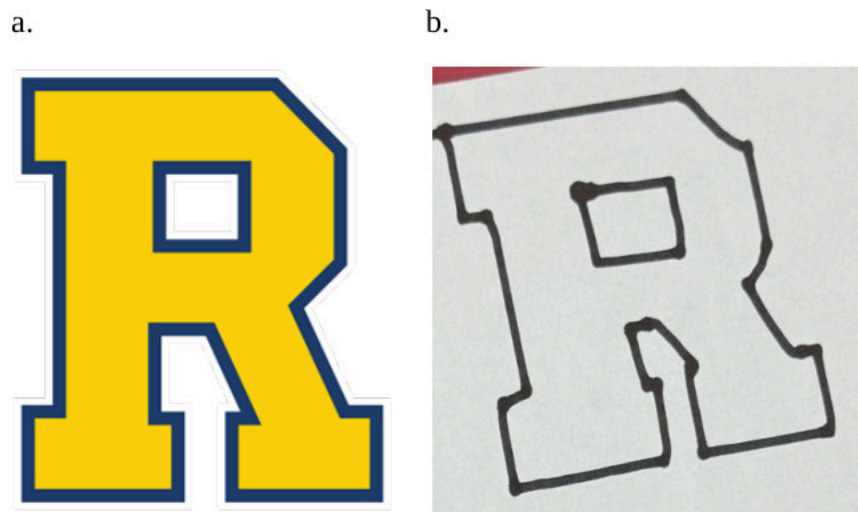


Figure 1: **a**, University of Rochester logo R. **b**, the real image taken by a tilted angle drawn by LVDTs.

Introduction

Precision linear translation stage strategy is widely used in many machines, such as 3D printers and CNC (computer numerical control) mills and lathes. To use these two machines, we need to import a G code to them, and the G code will list all the positions that the machines need to go in sequence. It is the same mechanism in this experiment. We will manually pick the pixel positions in x and y direction from Figure 1 a., and, instead of using G code, we will generate a MATLAB code to list the positions that the LVDTs need to move in sequence.

The translation stage layout is shown in Figure 2. The servo motors can move the half-rounded plates which will push the red stage in x and y directions. The LVDTs are connected to the pins with name labeled on it and it will output voltage. In this figure the pins are A1 and A0. In order to read the LVDTs and give commands to the servo motors, we need to use an Arduino Uno microcontroller board which can be programmed using MATLAB. If we push the pin, the Arduino will provide a voltage correspond to the displacement of the pin. It is forbidden to give a voltage exceeding 5V or the Arduino will be destroyed.

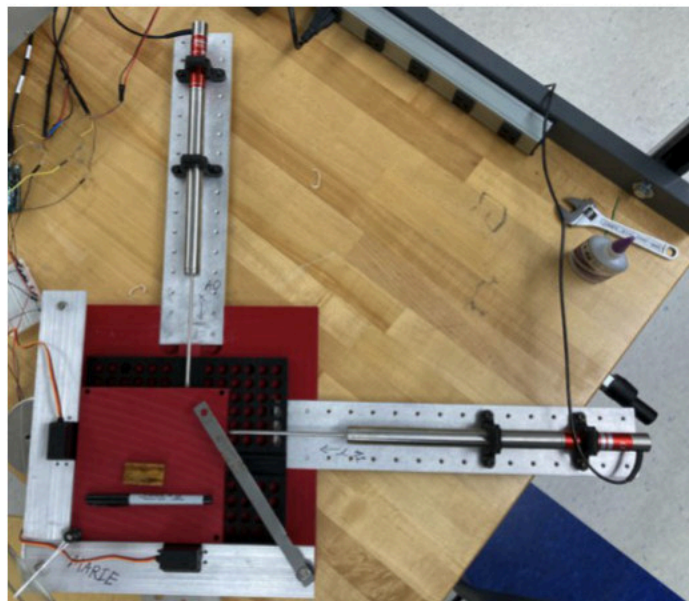


Figure 2. Layout of LVDTs. Red block in the center is the stage. Half-rounded plate is connected to servo motors. Arduino pin is connected to the LVDTs.

Method

At first, we need to find the inner and outer vertices of the R shown in Figure 1a. Load the image in MATLAB using `imread` and `imshow`. Then use data tip to mark the vertices shown in Figure 3. Record the pixel position and then measure the length of the paper we will use in drawing. In our case is 65 mm x 65 mm paper. The image is in 300-pixel x 300-pixel frame. Therefore, we can convert the pixels into unit of mm using equation (1).

$$\text{Location_mm} = [\text{pixel position}] * 65 \text{ mm} / 300\text{-pixel} \quad (1)$$

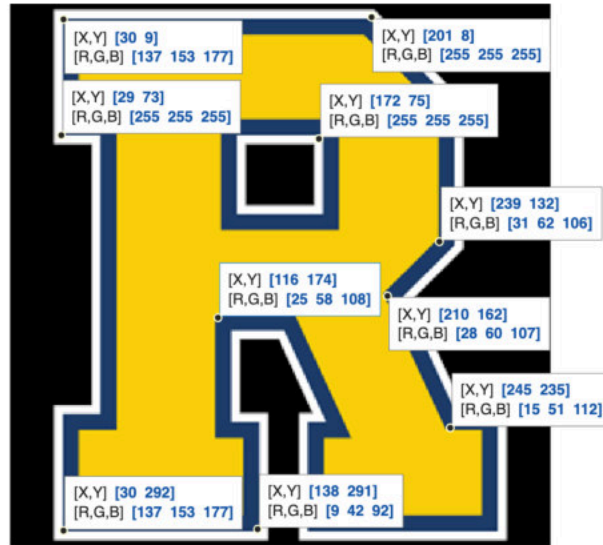


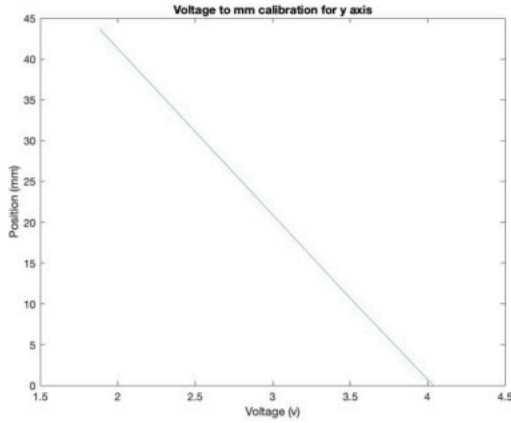
Figure 3. Using data tip to get the pixel position of the vertices of the R. This is only an example to show how to pick the vertices and get the position. The figure does not contain all vertices.

After that, we will use MATLAB to initialize the Arduino microcontroller board and test if the servo motor can move. Please see first two sections in Translation Lab code appendix. Note, we need to check which pin are we using. In our case, we have D3 for x axis and D 11 for y axis.

After we check the servo motors, we should calibrate the LVDTs. First, we need to check the port connecting the Arduino and LVDT. The white signal wire corresponding to x axis is connected to A0, and the signal wire for y axis should be connected to A5. The ground black wire should be connected to Arduino ground. Then our group decide to have a full-scale grid showing the relationship between voltage and angle inputs shown in appendix Translation Lab code section 3 and 4. We choose a resolution of 2000 steps from 0 degrees to 270 degrees. We put a pause after servo motor moves allowing the Arduino to read the voltage. Therefore, each step will have a corresponding voltage output, and we will use this grid to convert voltage to angle inputs later in the experiment. This process will take 20 minutes. After we have the grid, we will use a copper

block to get the calibration from mm to voltage. We should measure the three dimensions of the block and pull the pin to put the block between the stage and pin. After inserting the block, we will read the voltage using readVotage command and record the voltage every time. This method should be applied to both the x axis and y axis. After we have the voltages in x and y axis, we need to plot a figure showing the relationship between the distance and voltage shown in Figure 4. The code for this part is in Appendix Translation Lab code section 8.

a.



b.

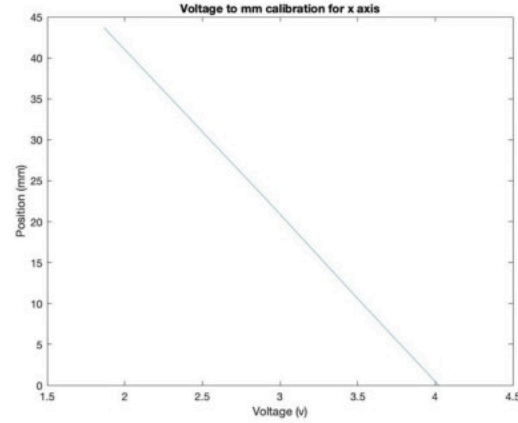


Figure 4: Voltage to mm calibration. Calibration for y axis in a. Calibration for x axis in b.

In theory, the voltage should be linearly related with the distance position. The figures confirmed the linear relationship between voltage and position from LVDT. Then we will perform a polyfit to get the slopes and intercepts for the relationship for x and y axis. Thus, we can use the linear equation to convert any voltage to mm by equation (2).

$$\text{Position (mm)} = \text{slope} * \text{voltage} + \text{intercept} \quad (2)$$

After this, we want to draw some ‘goal’ positions. Before that, we need to convert the pixel positions to angles. We use equation (1) to get pixel positions into mm positions, codes shown in Appendix Translation Lab code section section 9. The x and y represent the outer vertices of the R, and xm and ym represent the inner vertices for rectangle of the R. Then, we need to convert mm to voltage using equation (2), by plugging in the slope and intercept we found in section 8. This code is in section 10. Then we need to convert the voltage to angles so that we can input to the servo motors. In section 10, since the resolution is very small, the Arduino can’t capture the changes in voltage, so we first manually choose the range that the voltage has a clear change with the resolution. Because the half-rounded plate is doing circular motion, after it reaches the maximum from minimum position, the motion is just repeating the motion before it reaches the maximum. Therefore, our range is set to be the first chosen voltage to the minimum voltage. And then we use a for loop to find the indices of converted voltage in our chosen voltage range grid.

Ideally, we want to find the voltage from our range equals to the converted voltage, and then select the index multiplying the resolution to get the angle input. However, in the real life case, it's hard to get the exact value in our voltage grid, so we define a tiny range of 0.04 V for the converted voltage, and then use find command to get all the indices in our grid that are within the defined range, but we only would use the first index among all the indices. We deliberately increased the range for the outer y vertices to 0.1 V because the converted voltages are lower than the voltage values in our range, so we scaled them up by 0.1 V. Then we get the angle inputs for those converted voltages, but we need to add the abandoned voltages to get the angles in global coordinate system. We test the one-dimensional move in section 6 by drawing a square.

After that we used a function called diagonal to draw the final image. The code is shown in Appendix Diagonal Function. We used a scheme of finding the delta x and y and add it to the previous x and y position so it will draw a zig-zag line. We didn't add the pause for drawing the x and y because the pause will enforce the motor to draw 1D motion in x or y because the other coordinate is locked by the pause, but after drawing each line, we should pause in case the machine can't read the input angles. We call the diagonal function in the script, and the final image is shown in Figure 1.b.

Result

The code we designed successfully draw the R logo shown in Figure 5.

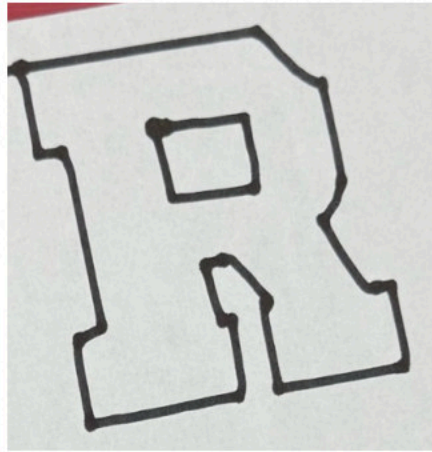


Figure 5. The real image taken by a tilted angle drawn by LVDT.

The figure is not exactly the same as the original R logo because the Arduino can't read the voltage precisely with the resolution we used. What's more, when we get the pixel vertices, we are only using human eyes to find the pixel position which will generate bias to the results, so the image can be off by the original logo.

Conclusion

This exercise successfully characterized a linear translation stage, producing a reliable calibration. We used voltage grid to get a full-scale information for the voltage and angle, and we successfully converted the pixel position to mm but this process has a large uncertainty by measuring the paper and find the vertices. We also used copper block to calibrate the mm to voltage. Use the input angles, we sent the command to LVDT and the image is produced.

Future research should use a better measurement system (like a laser interferometer) to independently track the stage's actual position while isolating the sensor error from the actuation error. This would permit a more thorough validation of the calibration and a complete assessment of error. It would also be beneficial to study dynamic errors by recording the position overshoot and settling time during a movement. Our results raise several additional questions: For instance, is there a change in repeatability with different load or speeds? What is the size of the system's backlash? Analyzing these questions would provide greater insight into the stage's performance potential during real-world use.

APPENDIX

```
%% Translation Lab code
a = arduino("COM31","Uno"); %initialize Arduino
servoX = servo(a,'D3') % initialize servo on Arduino pin D3
readPosition(servoX)
writePosition(servoX,1) % sends the servomotor to its 270° position
readPosition(servoX)
writePosition(servoX,0) % sends the servomotor to its 0° position
readPosition(servoX)
writePosition(servoX,0.52) % stop
%% Y-axis
servoY = servo(a,'D11') % initialize servo on Arduino pin D3
readPosition(servoY)
writePosition(servoY,1) % sends the servomotor to its 270° position
readPosition(servoY)
writePosition(servoY,0) % sends the servomotor to its 0° position
readPosition(servoY)
writePosition(servoY,0.52) % stop

%% Calibration
a = arduino("COM31","Uno");
servoX = servo(a,'D3');
x = linspace(0,1,2000);%angle step
resolution = 1/2001;
voltageX = [];%initialize voltage
for i = 1:length(x)
    writePosition(servoX, x(i));%roate with angle step
    readVoltage(a,'A0')
    pause(.001)
    voltageX(i)=readVoltage(a,'A0');%read voltage
    pause(.1)
end
save("Voltage_x_direction.mat","voltageX");
%%
y = linspace(0,1,2000);%angle step
voltageY = [];%initialize voltage
for i = 1:length(y)
    writePosition(servoY, y(i));%roate with angle step
    readVoltage(a,'A5')
    pause(.001)
    voltageY(i)=readVoltage(a,'A5');%read voltage
    pause(.1)
end
save("voltage_y_direction.mat","voltageY");
%%
max_voltage_x = max(voltageX); %position .005
```

```

max_voltage_y = max(voltagey); %position .1361
min_voltage_x = min(voltagex); %position .6113
min_voltage_y = min(voltagey); %position .7009
%% drwing square to set coordinate
writePosition(servoY, .7009);
pause(.5)
writePosition(servoX, .6113);
pause(.5)
writePosition(servoY, .1361);
pause(.5)
writePosition(servoX, .005);
%% Calibration from voltage to mm
writePosition(servoX, .005);% set the LDTV to max x postion with max voltage
readVoltage(a,'A0');%read first to get better result
pause(0.001);
Vx_0 = readVoltage(a,'A0'); % maximum voltage in x (fully extended)
writePosition(servoY, .1361);% set the LDTV to max y postion with max voltage
readVoltage(a,'A5');%read first to get better result
pause(0.001);
Vy_0 = readVoltage(a,'A5'); % maximum voltage in y (fully extended)
%% click every time after inserting block
readVoltage(a,'A1')
% plot four data points
volt_calibrate_A0 = [1.866,2.9374,3.4751,4.0225];%v
volt_calibrate_A1 = [1.8817,2.9374,3.4800,4.0371]
dimensions= [43.7,22.2,11.1,0];%mm
figure;
% relationship between voltage and mm
plot(volt_calibrate_A0,dimensions);
xlabel("Voltage (v)");ylabel("position (mm)");title("voltage to mm calibration x axis");
plor(volt_calibrate_A1,dimensions);
xlabel("Voltage (v)");ylabel("position (mm)");title("voltage to mm calibration y axis");
p = polyfit(volt_calibrate_A0,dimensions,1);
p2 = polyfit(volt_calibrate_A1,dimensions,1);
slope_A0 = p(1)%unit(mm/V);
interception_A0 = p(2);% mm
slope_A1 = p2(1);interception_A1 = p2(2);
%% converting pixels to mm
%outter edge
x = [26,26,46,46,26,26,142,142,127,127,149,171,162,162,275,275,252,223,251,251,203,26] * 65/300;
y = [5,77,77,224,224,296,296,228,228,185,185,227,228,296,296,224,224,164,135,55,6,5]* 65/300;
% Middle Edges
x_m = [116,116,184,184,116]* 65/300;
y_m = [65,125,125,65,65]* 65/300;
%% converting mm to voltage
converted_x_to_v = (x-interception_A0)./slope_A0;
converted_y_to_v = (y-interception_A1)./slope_A1;

```

```

converted_x_m_to_v = (x_m - interception_A0)./slope_A0;
converted_y_m_to_v = (y_m - interception_A1)./slope_A1;
%% convert voltage to x&y angle
range_voltages_x = voltages_x(3:1232);% hard code range of x
range_voltages_y = voltages_y(253:1401);% hard code range of y
position_x = [];
position_xm = [];
position_y = [];
position_ym = [];
% converting edge voltage to angle
for i=1:length(converted_x_to_v)
    ind_x_in_degree(i)=find((range_voltages_x < (converted_x_to_v(i) + .02)) & (range_voltages_x >
(converted_x_to_v(i) - .02) ),1,'first');
    position_x(i) = (3*resolution) + ind_x_in_degree(i)*resolution;% positon x in global coordinate
end
for i = 1:length(converted_x_to_v)
    ind_y_in_degree(i)=find((range_voltages_y <=(converted_y_to_v(i) + .1) ),1,'first');
    position_y(i) = (253*resolution) + ind_y_in_degree(i)*resolution;% positon y in global coordinate
end
% converting middle edge voltage to angle
for i=1:length(converted_x_m_to_v)
    ind_xm_in_degree(i)=find((range_voltages_x < (converted_x_m_to_v(i) + .02)) & (range_voltages_x >
(converted_x_m_to_v(i) - .02) ),1,'first');
    position_xm(i) = (3*resolution) + ind_xm_in_degree(i)*resolution;% positon y in global
coordinate
end
for i=1:length(converted_y_m_to_v)
    ind_ym_in_degree(i)=find((range_voltages_y < (converted_y_m_to_v(i) + .02)) & (range_voltages_y >
(converted_y_m_to_v(i) - .02) ),1,'first');
    position_ym(i) = (253*resolution) + ind_ym_in_degree(i)*resolution;% positon y in global
coordinate
end
%% draing diagnoal lines for the outter edges
writePosition(servoY,.1361);
pause(.5)
writePosition(servoX,.005);
pause(5);
diagonal(servoX,servoY,position_x,position_y,100)
%% initial position
writePosition(servoX, position_xm(1));
writePosition(servoY, position_ym(1));
pause(5)
%% drawing the inner square
for i = 1:length(position_xm)
    distance_xm_move = position_xm(i);
    distance_ym_move = position_ym(i);
    writePosition(servoX, distance_xm_move);

```

```
    pause(.2);  
    writePosition(servoY, distance_ym_move);  
    pause(.2)  
end
```

Diagonal Function

```
function []= diagonal(servoX,servoY,x,y,resolution)  
dx = [];  
dy = [];  
for i = 1:length(x)-1  
    dx(i) = (x(i+1)-x(i))/resolution;  
    dx(22) = (x(22)-x(21))/resolution;  
end  
for i = 1:length(y)-1  
    dy(i) = (y(i+1)-y(i))/resolution;  
    dy(22) = (y(22)-y(21))/resolution;  
end  
for i = 1:length(dx)  
    writePosition(servoX, x(i)+dx(i));  
    writePosition(servoY,y(i)+dy(i));  
    pause(.5);  
end
```