AC 2009-1418: STUDENT DESIGN AND DEVELOPMENT OF A TACTILE DISPLAY WITH THREE-DIMENSIONAL MOVEMENTS

Andrew Patrick, Texas A&M University

Clint Vigil, Texas A&M University

Ryan Beasley, Texas A&M University

Ben Zoghi, Texas A&M University

Student Design and Development of a Tactile Display with Three Dimensional Movements

Abstract

This paper describes a senior-level class project in which two Electronics Engineering Technology undergraduates designed and implemented a novel tactile display. The display consists of four pins, each attached to a platform moved by three servos. The twelve servos are controlled by a commercial servo controller using software developed in the Python programming language. This project provided the students with experience investigating currently available systems in a rapidly expanding field, proposing electro-mechanical designs, evaluating the proposed designs, constructing a functioning prototype of the chosen design, and documenting the process. Guidance and evaluation throughout the project were provided by the course instructor and a technical expert.

Introduction

Tactile feedback is a common method of providing information to users of a device through the sense of touch, e.g., vibrating cell phones, warning bumps on the shoulders of roads, and force-feedback joysticks. One heavily investigated type of tactile displays uses many individual pins pressed against the user's hand or fingers[1,2]. The ultimate goal with such pin-driven displays is, through sufficient density of pins and high fidelity in control of each pin's position, to recreate on the surface of the user's skin the same shape as is either sensed at a remote location or generated in a simulation. In the first situation, such displays may provide feedback to surgeons controlling medical robots[3] or to police controlling bomb-defusing robots, and in the second situation such displays could assist in rapid design by allowing designers to feel the shapes of potential creations.

Currently available devices are limited to a single degree of movement for each pin. There has been limited research into tactile displays that can move each pin both normally and tangentially to the skin. The use of 3D movement for the pins means the display can mimic more interactions between the skin and various remote or simulated surfaces[4,5]. This paper covers the experiences of two undergraduate Electronics Engineering Technology students in developing and constructing a tactile display with four pins with independent motion in three dimensions. This project was part of a class on control theory, and the students had previously taken classes in analog circuits, digital circuits, and circuit/component testing. Each student spent approximately 20 hours on this project.

Project Flow

This project was completed as part of a senior-level course on control systems. The students chose to work on this project as proposed by the technical expert. Together, the students and the technical expert decided upon appropriate goals, including construction of a working system and documentation. Then the students proceeded to review literature on the topic of pin-driven tactile displays. Once the students understood the design space, the students and technical expert brainstormed possible electro-mechanical designs. The students evaluated the expected performances of the designs based on the metrics used in the literature, and choose one design to construct. The finished system allows a user to explore a digital picture using the sense of touch, and was evaluated based on performance.

System Overview

During the design process, multiple electro-mechanical systems were considered for moving the tactile pins. Servomotors, as used in remote control vehicles, were quickly chosen as the actuator due to their ease of use, low cost, and sufficient force. The difficult part of the design was how to position the servomotors to connect to the pins without the transmissions intersecting and interfering. Most of the initial designs attempted to position the servomotors orthogonally so that each servomotor would control a direction of the pin's movement in an orthogonal basis, but such designs were obviously restricted to displays with a small number of pins.

The design that was selected for construction places the pins on platforms supported by multiple actuators in parallel, based on two considerations. First, it was thought that the servomotors could all be placed on the same plane, simplifying construction. Second, construction of displays with large numbers of pins seemed possible. Third, rough calculations showed that the display could exert sufficient forces and allow the recommended pin density as mentioned in other tactile display work. Fourth, parallel manipulators tend to be less susceptible to errors due to flexing of supports than manipulators where the actuators are in series. Unfortunately, based on the speed and slew rate limit of servomotors, the design was not expected to reach the ideal bandwidth for tactile displays.

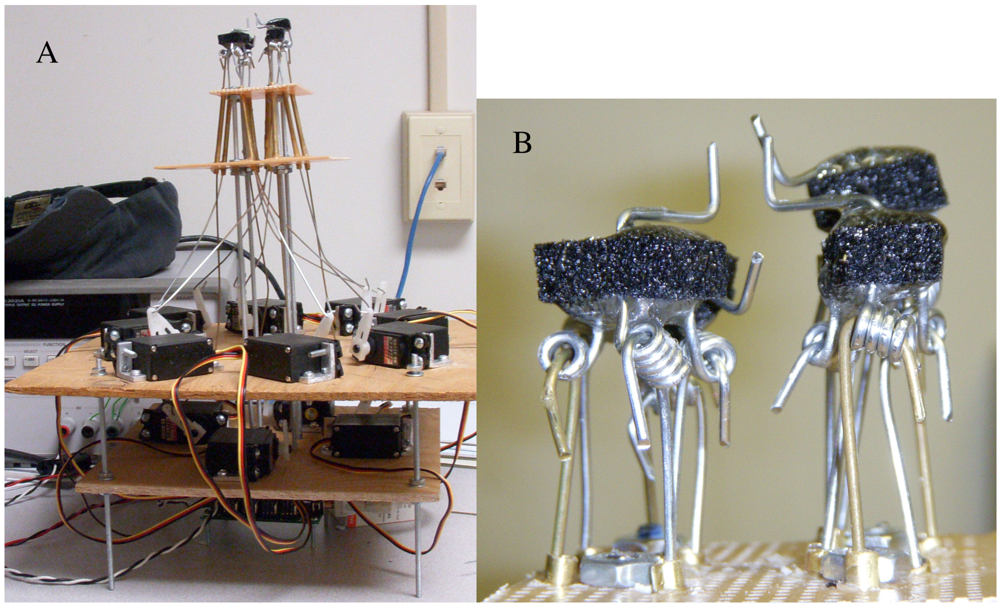

Fig. 1. A) 2x2 tactile display with 3D movements. B) Enlarged view of pins and platforms.

The display (Fig. 1) consists of five main parts: pins, platforms, servomotors, servo controller, and user interface program. Four pins directly interact with the user's fingertip. Looking down from above the display, the pins form the corners of a square. Each pin is attached to a separate platform. Each platform is moved by three servomotors. Note that if six actuators were attached to each platform, we would have a platform capable of motions in all three translations and all three rotations. With just three actuators, only three degrees of freedom are controlled. The servo motors are connected to a servo controller board which powers them and can send pulse-width modulated (PWM) signals to each servomotor. The controller is connected to a PC. The PC runs a program that provides a graphical user interface (GUI) and sends the controller the desired position for each servomotor. The students chose servomotors and a servocontroller based on costs and capabilities. A simple block diagram representing one pin of the display is shown in figure 2.

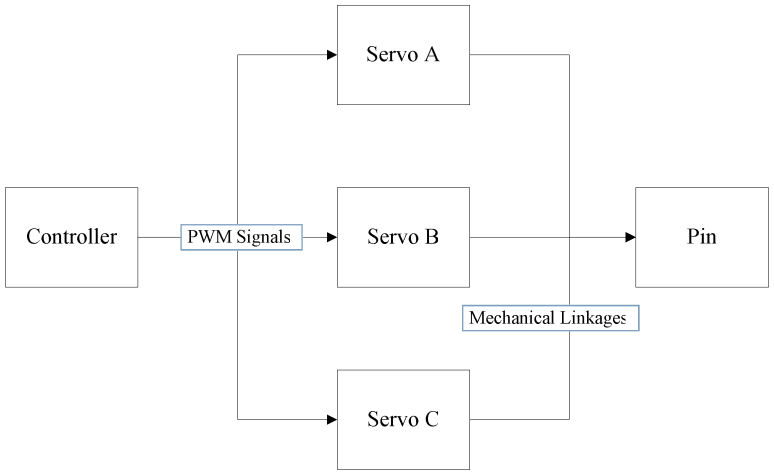

Fig. 2. Block diagram for one display pin.

Hardware

The servo controller board being used in this device is the SSC-32 (firmware version 2) produced by Lynxmotion. The microcontroller on board the servo controller is the Atmel ATMEGA168-20PU. This servo controller has a number of features which made it very attractive for this application. The high resolution outputs (1µS) along with the group move feature provide precise and smooth movements. The output pulse range of the device is 0.50 to 2.50mS providing 180° of servo travel. This range is limited to 0.50-1.50mS or 1.50-2.50mS depending on the mounting orientation of the servo being controlled. Communication with the controlling computer is done using RS-232 standard serial communication at 115.2kbps. The servo controller board has a DB-9 female connector. The board is shown below in figure 3.

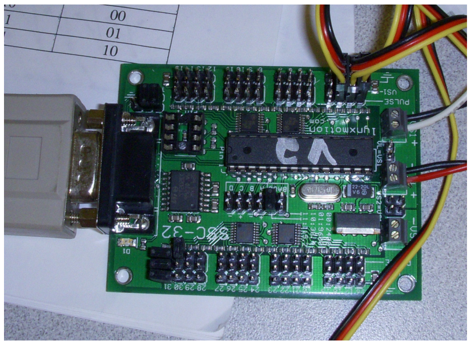

Fig. 3. The Lynxmotion SSC-32 Servo Controller Board.

Power is supplied to the servo controller from an adjustable dual output power supply. The servo controller board was configured to use separate power input for logic power and servo power. The logic power input requires 6V and the servo power input requires 5V. Though the servos can be powered at 6V, doing so would reduce their life. The split power supply setup also insures that any voltage drops caused by high servo current demand will not affect the processor.

The servos used were HS-322 servos made by Hitec. These servos were chosen based on several factors. The servos have a range of 200° which is greater than the range of the servo controller board. The speed of the servos at 0.29 seconds over 90° along with a torque of 42 oz. in. were both adequate. A single arm was used on the output of each servo because the mounting method used for the servos prevented the use of any of the other arms or wheels included with the servos. The servos connect to the servo controller using a 3 wire cable. The three wires supply +5V, ground and a PWM signal. During construction it was determined that separating the servos into two layers would simplify the design. As a result, eight of the servos (two per pin) were placed on one layer, with the remaining four servos placed at a lower layer. This change allowed the servos to be placed closer to the radial center of the device and thus reduced the amount of bending in the transmissions from servos to pins.

The device contains four platforms, consisting of rods connected to a thin layer of stiff foam rubber, each driven by three servos. Foam was easy to work with and provides a limited amount of springiness between the servos and the pins, to reduce the possibility that program errors would cause the pins to injure the user. Metal rods 1mm in diameter are attached to each servo and routed through brass tubes then attached to the foam. The 1.5 inch long brass tubes are placed several inches above the servos and are used for restricting the movement of the connecting rods. By only allowing the rods to move in a linear manner it increases the positional accuracy of the platforms as well as greatly simplifies the software needed to control the device. For the connection between the rods and the platform a hinge type connection was used, a ball and socket joint would be much better for this application but a suitable size one was not available in the limited time frame during construction. Each platform has a small metal L shaped pin that is rigidly attached to its top and extends toward the middle of the device. The ends of all four pins are close enough for one fingertip to rest on all them.

This device utilizes several different types of cabling to supply power and data. Power is supplied to the servo controller board through four 18 gauge stranded cables. Each cable is 2 feet long and attaches to a screw type terminal block on the servo board. The power supply end of each cable is a standard banana plug. For serial communication between the computer and servo controller board a serial cable with on female DB-9 and one male DB-9 is used. Each servo has a built in 3 conductor cable that ends in a female connector which attaches directly to the servo controller board.

Software

A graphical user interface was programmed that would load a picture, and display the mouse cursor as a square that was two pixels wide and two pixels tall. The colors of the four pixels within the cursor were then used as control values for the servos. Each pixel corresponds to one of the four pins. For a given pixel, the red, green, blue (RGB) color values are each used to move one of the three servos connected to the corresponding pin. For the software to work, the servo controller has to be set to a serial setting that matches program using jumpers on the board. The settings were: 115200 Baud Rate, 8 data bits, 1 stop bits 1, no parity, and no flow control.

The GUI is where the user can manipulate the pins by using the mouse and any image. The code is written in Python with the Python Image Library (PIL), for handling the image data, and pySerial, for handling serial communication[6]. Furthermore, pySerial requires Jython and PythonWin.

Once an image is loaded, four windows are shown (Fig. 4). The top window shows the actual image while the three images below shows the red, green, and blue values of the image. When the user moves the mouse cursor over the picture, the cursor becomes a 2x2 square to represent which four pixels are being used to control the servos. Each point gives a red, green, and blue value between 0 and 255. These values are scaled by a factor that is unique to each servo due to the wire moving to each platform. The scaled values are converted to ASCII characters and sent to the servo controller using serial communication. Then the GUI waits for the mouse to move again and repeats the process.

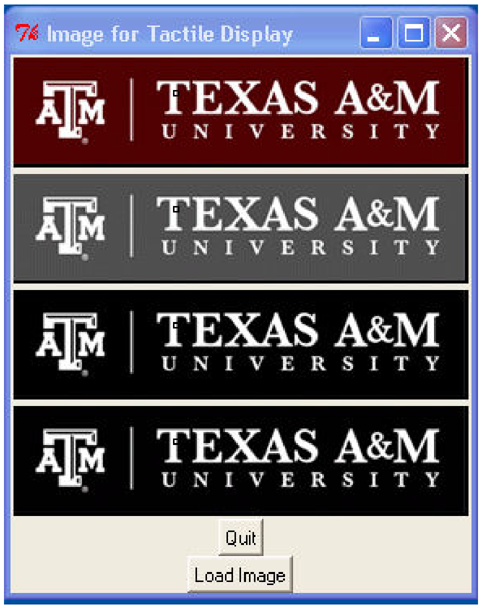

Fig. 4. Graphic User Interface showing four copies (full color, red portion, blue portion, green portion, respectively) of a digital image. The small square box (on the "T" of "Texas") indicates which pixels are currently being used to move the display pins.

Results

The tactile display successfully allowed tactile exploration of color images, exceeding expectations in some areas while failing to meet other expectations. Dramatically, the subjective experience of exploring an image (e.g., map) using both normal and tangential distributed forces was very interesting. Also noticeable is an error in specifying the design constraints; the finished product is over a foot tall, making it rather unwieldy. Though not considered during the design phase, use of the device suggests that such tactile displays must be much shorter (or be placed on short tables) for effective use.

The main problem came from one aspect of the design: several of the rods from the servos to the platforms touch each other. Because the rods were so close together, moving one servo would cause slight movement of the other platforms, i.e., crosstalk. This made accurate pin motions problematic. It also made centering all the platforms to a level state unlikely since the position of a platform depended on the position of the three directly connected servos as well as every other servo with which they cross rods.

The range of movement of the pins exceeded expectations. The design was to have about 0.5 mm tilt for the angle and a vertical travel of 3 mm. The pins in the final product showed movement ranges of 1 to 1.5 inches in all directions. In fact, pinching skin between two pins became a problem.

Conclusion and Future Work

This paper describes a design and development project by two senior undergraduates, in which they produced a proof of concept tactile display that provides both normal and tangential forces distributed across the fingertip. During this project, the students performed guided literature reviews, created their own design constraints, brainstormed designs, selected a design, and constructed a physical device from their design. The scope of this project permits a multitude of materials, designs, and approaches. Furthermore, the materials can be chosen for ease of use, to prevent mechanical considerations from overwhelming non-mechanically-oriented students. In the future the same project can be attempted with different students, a different design, and a different end product.

A redesign is necessary to improve on the capabilities of this device. First, the design does not scale smoothly. For example, constructing a nine (3x3) pin display would necessitate a new layout of the servos and significant work in servo placement to prevent the rods from interacting with each other. The underlying issue is that the servomotors are significantly larger than the inter-pin spacing, such that the actuators take significant space and the transmissions must condense down to a much smaller area. In this design, a collision of the rods resulted, preventing accurate display of information. Addressing this design issue is of primary importance, since more pins are necessary for realistic tactile information. One potential approach, envisioned at the end of this project, is to use springs to push the pins towards one limit of motion, with servo-actuated ropes to pull the pins towards the other limit[2]. Efficient design should provide high pin density without allowing the ropes to interact.

Bibliography

1. Wagner, C.R., S.J. Lederman, and R.D. Howe, "Design and performance of a tactile shape display using RC servomotors," *Haptics-e*, vol. 3, 2003.

2. Sarakoglou, I., M. Bezdicek, N. Tsagarakis, and D.G. Caldwell, “Free to touch: A portable tactile display for 3D surface texture exploration,” *Intl. Conf. Intelligent Robots and Systems*, 3587-3592, 2006.
3. Beasley, R.A., and R. Howe. “Tactile tracking of arteries in robotic surgery,” *IEEE Intl. Conf. on Robotics and Automation*, 4:3801-3806, 2002.
4. Drewing, K., M. Fritschi, R. Zopf, and M.O. Ernst, “First evaluation of a novel tactile display exerting shear force via lateral displacement,” *ACM Trans. Applied Perception*, 2(2):118-131, 2005.
5. Moy, G., and R.S. Fearing, “Effects of shear stress in teletaction and human perception,” *Proc ASME Dyn. Systems and Control Division*, 64:265-272, 1998.
6. Python programming language, www.python.org

Appendix: Software in Python

```
'''This GUI based program loads an image and displays its RGB color layers.  The color values for the image pixels
are then used to drive a tactile pin display (in three dimensions).

When an image is loaded, it is converted to RGB format and then displayed along with a separate image for each
color layer.  Moving the mouse over the top (full-color) image displays a square surrounding a 2x2 selection of
pixels.  A matching square is displayed on the three images of the separate color layers.  That information (2x2x3) is
then sent via serial communication to a servo controller, for driving a 2x2 tactile display in which each pin element
can move in three dimensions.

by Andrew Patrick, Clint Vigil, and Ryan Beasley, 2008.
'''

import Tkinter as tk # give Tkinter a namespace to avoid conflicts with PIL (they both have a class named Image)
from PIL import Image, ImageTk
import tkFileDialog # used to load the image file
import serial

ser = serial.Serial(0,115200) # set up serial port
ser.write("\r") # clear out any previous commands
ser.write("#0 P1370 #1 P1500 #2 P1480 #4 P1230 #5 P1500 #6 P1500 #8 P1500 #9 P1400 #10 P1500 #12 P1360
#13 P1550 #14 P1680 \r") # default all servos up
ser.close()

class App:
    def __init__(self, master):
        self.frame = tk.Frame(master)
        self.frame.pack()

        self.canvas1 = tk.Canvas(self.frame, width=0, height=0, bg="black") # place canvases in the frame for each of
the four images
        self.canvas1.bind("<Motion>", self.onMouseMove) # when the mouse moves over the first canvas, we will
print out the pixel values
        self.canvasR = tk.Canvas(self.frame, width=0, height=0, bg="black")
        self.canvasG = tk.Canvas(self.frame, width=0, height=0, bg="black")
        self.canvasB = tk.Canvas(self.frame, width=0, height=0, bg="black")
        self.canvas1.pack(side='top')
        self.canvasR.pack(side='top')
        self.canvasG.pack(side='top')
        self.canvasB.pack(side='top')

        # button to quit
        self.button2 = tk.Button(self.frame, text='Quit', command=self.frame.quit)
        self.button2.pack(side='top')
        # button to load image
        self.button3 = tk.Button(self.frame, text='Load Image', command=self.load_image)
        self.button3.pack(side='top')

    def load_image(self):
        self.imageFile = tkFileDialog.askopenfile(parent=root, mode='rb', title='Choose an image file') # this opens a
dialog and returns a file pointer
        if self.imageFile != None:
            self.canvas1.delete('image') # clear any old images
            self.canvasR.delete('imageR')
            self.canvasG.delete('imageG')
            self.canvasB.delete('imageB')
            self.im = Image.open(self.imageFile).convert('RGB') # open the image and convert to RGB to get three
layers
            self.image1 = ImageTk.PhotoImage(self.im) # convert to a type that Tk can deal with
            self.w = self.image1.width()
            self.h = self.image1.height()
            self.canvas1.create_image(0, 0, image=self.image1, tags='image', anchor="nw") # put the image on the
canvas
            self.source = self.im.split() # get the three color layers
            self.imR = ImageTk.PhotoImage(self.source[0])
            self.imG = ImageTk.PhotoImage(self.source[1])
            self.imB = ImageTk.PhotoImage(self.source[2])
            self.canvasR.create_image(0, 0, image=self.imR, tags='imageR', anchor="nw")
            self.canvasG.create_image(0, 0, image=self.imG, tags='imageG', anchor="nw")
            self.canvasB.create_image(0, 0, image=self.imB, tags='imageB', anchor="nw")
            self.canvas1.config(width=self.w, height=self.h) # change the size of the canvases to fit the images
            self.canvasR.config(width=self.w, height=self.h)
            self.canvasG.config(width=self.w, height=self.h)
            self.canvasB.config(width=self.w, height=self.h)

    def onMouseMove(self, event):
        x = event.x # location of the mouse in the widget
        y = event.y
        if x<self.w-1 and y<self.h-1: # make sure the 2x2 pixels all fit in the image
            pix = [self.im.getpixel((x,y)), self.im.getpixel((x+1,y)), self.im.getpixel((x,y+1)),
self.im.getpixel((x+1,y+1))] # get the pixel values from the image, these are what need to be sent to the tactile
display
            print x, y, pix # print the cursor location and the pixel values
            pin1 = self.im.getpixel((x,y)) # seperate pixel values for coversion use
            pin2 = self.im.getpixel((x+1,y))
            pin3 = self.im.getpixel((x,y+1))
            pin4 = self.im.getpixel((x+1,y+1))
            self.canvas1.delete('rect') # clear out any previous box (used to show which pixels are being selected
            self.canvas1.create_rectangle(x-1, y-1, x+2, y+2, tags='rect') # draw a box around the selected pixels
            self.canvasR.delete('rect')
            self.canvasR.create_rectangle(x-1, y-1, x+2, y+2, tags='rect')
            self.canvasG.delete('rect')
            self.canvasG.create_rectangle(x-1, y-1, x+2, y+2, tags='rect')
            self.canvasB.delete('rect')
            self.canvasB.create_rectangle(x-1, y-1, x+2, y+2, tags='rect')
            R1 = pin1[0] # retrive individual RGB values from pixel data for pin 1
            G1 = pin1[1]
            B1 = pin1[2]
            R2 = pin2[0] # retrive individual RGB values from pixel data for pin 2
```

```
G2 = pin2[1]
B2 = pin2[2]
R3 = pin3[0] # retrive individual RGB values from pixel data for pin 3
G3 = pin3[1]
B3 = pin3[2]
R4 = pin4[0] # retrive individual RGB values from pixel data for pin 4
G4 = pin4[1]
B4 = pin4[2]
moverA = 1370 - (2.2*R1)  # convert RGB values to XYZ directions
movegA = (300 - (1.18*G1)) + 1550
movebA = 1500 - (2.1*B1)
moverB = 1480 - (1.2*R2)
movegB = 1360 - (1.2*G2)
movebB = (300 - (1.18*B2)) + 1500
moverC = (200 - (0.78*R3)) + 1500
movegC = 1400 - (1.9*G3)
movebC = 1230 - (1.3*B3)
moverD = (300 - (1.18*R4)) + 1500
movegD = 1680 - (1.2*G4)
movebD = 1500 - (1.8*B4)
moverA = int(moverA)  # change values from float to int
movegA = int(movegA)
movebA = int(movebA)
moverB = int(moverB)
movegB = int(movegB)
movebB = int(movebB)
moverC = int(moverC)
movegC = int(movegC)
movebC = int(movebC)
moverD = int(moverD)
movegD = int(movegD)
movebD = int(movebD)
dataXA = int_2_ASCII(moverA) # convert number to ASCII characters for serial comm
dataYA = int_2_ASCII(movegA)
dataZA = int_2_ASCII(movebA)
dataXB = int_2_ASCII(moverB)
dataYB = int_2_ASCII(movegB)
dataZB = int_2_ASCII(movebB)
dataXC = int_2_ASCII(moverC)
dataYC = int_2_ASCII(movegC)
dataZC = int_2_ASCII(movebC)
dataXD = int_2_ASCII(moverD)
dataYD = int_2_ASCII(movegD)
dataZD = int_2_ASCII(movebD)
ser.open()  # open serial communication
ser.write("#0 P")   # move pin A
datawrite(dataXA)
ser.write("#13 P")
datawrite(dataYA)
ser.write("#5 P")
datawrite(dataZA)
ser.write(" \r")
ser.write("#2 P")  # move pin B
datawrite(dataXB)
ser.write("#12 P")
datawrite(dataYB)
```

```
            ser.write("#10 P")
            datawrite(dataZB)
            ser.write(" \r")
            ser.write("#1 P")  # move pin C
            datawrite(dataXC)
            ser.write("#9 P")
            datawrite(dataYC)
            ser.write("#4 P")
            datawrite(dataZC)
            ser.write(" \r")
            ser.write("#6 P")  # move pin D
            datawrite(dataXD)
            ser.write("#14 P")
            datawrite(dataYD)
            ser.write("#8 P")
            datawrite(dataZD)
            ser.write(" \r")
            ser.close() # close serial comm

def int_2_ASCII(number): # convert number to ASCII characters
        temp = [0,0,0,0]
        ascii_var = [' ',' ',' ',' ']
        y = 0
        z = 0
        while(number > 10):
            temp[y] = number%10
            number = number / 10
            y = y+1
        temp[y] = number
        while(y >= 0):
            char = temp[y]+ 48
            ascii_var[z] = chr(char)
            z = z+1
            y = y-1
        return ascii_var

def datawrite(list = []): # write ASCII characters to serial port
        for i in list:
            ser.write(i)

root = tk.Tk() # make a GUI
root.title('Image for Tactile Display')
app = App(root) # put the images and buttons in the GUI
root.mainloop() # run the GUI
```