Fun with LCD Graphics

Introduction

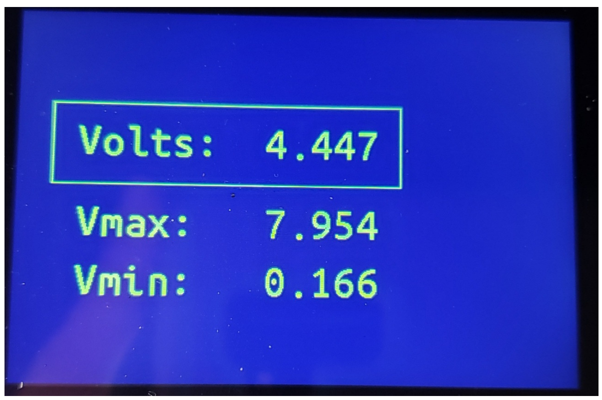
Once you get some graphics routines working, you can start using the graphics display to output information. This lecture goes over using the LCD display to

- Output text, such as the voltage or resistance attached to the Pi-Pico
- Display graphics, such as the x-y position of the joystick, and
- Do animation, such as a bouncing ball or a lunar lander game.

Volt Meter

Problem: Turn your PIC into a volt meter able to read

- 0V to 5V, or
- -10V to +10V

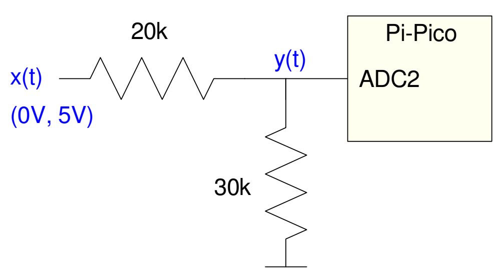


Using a Pi-Pico as a volt meter. The max & min voltages recorded are also displayed.

Hardware: 0V to +5V Inputs: The PIC only allows 0 - 3.3V inputs. With a voltage divider, you can convert 0-5V to 0-3.3V

$$y = \left(\frac{3.3V}{5.0V}\right)x = 0.660x$$

$$y = \left(\frac{R_1}{R_1+R_2}\right)x$$



Hardware for converting (0V,5V) to (0V, 3.3V) Full-scale (y = 3.3V) corresponds to x(t) = 5.5V

Hardware: -10V to +10V Inputs: A similar circuit will convert (-10V, +10V) to (0V, 3.3V). One way to come up with this circuit is to use three resistors for a weighted average. The function you want to implement is

$$y = \left(\frac{3.3V}{20V}\right)x + 1.65$$

Assuming you have access to a 3.3V source and a 0V source, this can be rewritten as

$$y = 0.165x + 0.5(3.3V)$$

Adding a term times 0V to make the coefficients add up to 1.000

$$y = 0.165(x) + 0.5(3.3V) + 0.335(0V)$$

Pick your favorite resistor value, such as R = 5k. The weighted average then has

$$R_x = \frac{R}{0.165} = 30.3k \approx 30k$$

$$R_{3.3V} = \frac{R}{0.5} = 10k$$

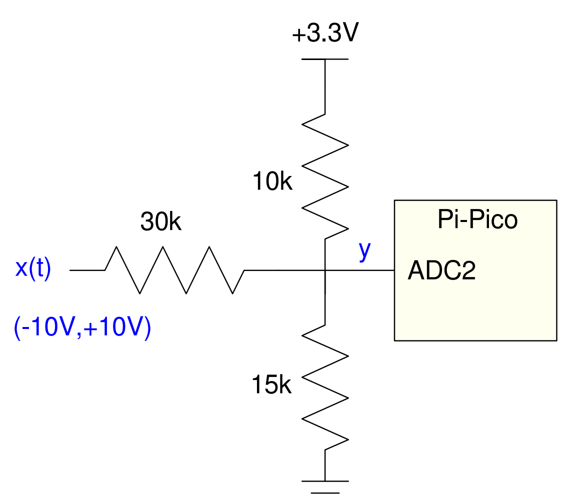
$$R_{0V} = \frac{R}{0.335} = 14.9k \approx 15k$$

A circuit which converts (-10V, +10V) to (0V, 3.3V) is then as follows. The relation between x and y in terms of voltage is

$$V_x = 6V_y - 9.9$$

In terms of the raw A/D reading:

$$V_x = 6 \cdot \left(\frac{3.3V}{65{,}535}\right) \cdot A/D - 9.9$$



Circuit for converting (-10V, +10V) to (0V, 3.3V)

Software: Several fonts are available. The following code uses 24x32 characters for displaying data (line #1). This font takes up about half of the available memory on the Pi-Pico, which is why it's not the default font.

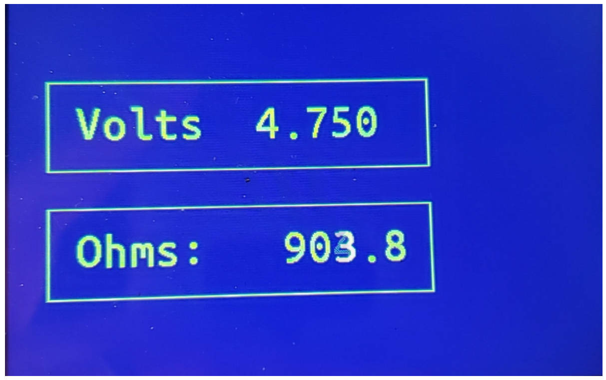
Other fonts available are:

- 8x16: library LCD, LCD.Text()
- 16x32: double the size of the 8x16 font. Library LCD, LCD.Text2()
- 16x24: library LCD_16x24, LCD_Text3().
- 24x32: library LCD_24x32, LCD_Text4()

```
import LCD_24x32 as LCD

from machine import ADC
from time import sleep_ms

a2d0 = machine.ADC(1)

Navy = LCD.RGB(0,0,10)
Yellow = LCD.RGB(150,150,0)

LCD.Init()
LCD.Clear(Navy)
LCD.Box(30, 80, 330, 150, Yellow)

k = 6.0 * 3.3 / 65535
Vmax = -999
Vmin = 999

while(1):
    a0 = a2d0.read_u16()
    Volt = k*a0 - 9.9
    if(Volt > Vmax):
        Vmax = Volt
    if(Volt < Vmin):
        Vmin = Volt
    LCD.Text4('Volts:', 50, 100, Yellow, Navy)
    LCD.Number4(Volt, 5, 3, 170, 100, Yellow, Navy)

    LCD.Text4('Vmax:', 50, 170, Yellow, Navy)
    LCD.Number4(Vmax, 5, 3, 170, 170, Yellow, Navy)

    LCD.Text4('Vmin:', 50, 220, Yellow, Navy)
    LCD.Number4(Vmin, 5, 3, 170, 220, Yellow, Navy)

    print(Volt)
    sleep_ms(200)
```

Ohm-Meter

If you can measure voltage, you can measure resistance. The trick is to convert ohms to volts. Once done, the A/D can read the voltage.



Using a Pi-Pico as an Ohm-Meter

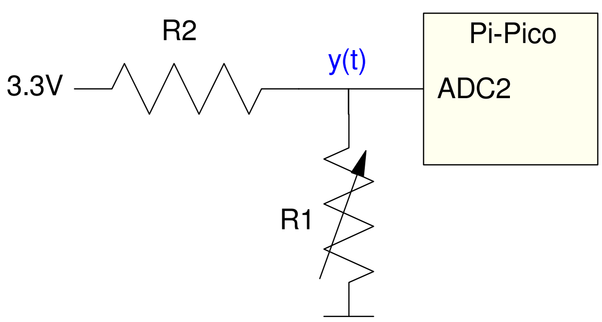
Hardware: A simple way to convert resistance to voltage is to use a voltage divider.

$$V = \left(\frac{R_1}{R_1+R_2}\right)3.3V$$

Once you measure the voltage, the resistance, R1, can be found

$$R_1 = \left(\frac{V}{3.3-V}\right)R_2$$

You get the best sensitivity when R1 = R2.



Configuration to use a PiPico as an ohm-meter. R1 can vary while R2 is fixed (1k in the following code)

Software:

Resistance can be found using the computed voltage:

$$R_1 = \left(\frac{V}{3.3-3}\right) R_2$$

or the raw A/D reading

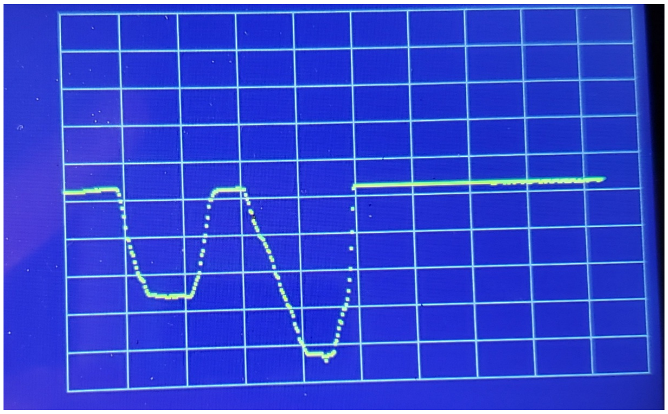
$$R_1 = \left(\frac{a_0}{65535-a_0}\right) R_2$$

The following code uses the latter to reduce the error in the computations.

```
# Ohm Meter
import LCD_24x32 as LCD

from machine import ADC
from time import sleep_ms

a2d0 = machine.ADC(1)

Navy = LCD.RGB(0,0,10)
Yellow = LCD.RGB(150,150,0)

LCD.Init()
LCD.Clear(Navy)
LCD.Box(30, 80, 330, 150, Yellow)
LCD.Box(30, 180, 330, 250, Yellow)

k = 3.3 / 65535

while(1):
    a0 = a2d0.read_u16()
    Volt = k*a0
    Ohms = a0 / (65535 - a0) * 1000.0
    LCD.Text4('Volts:', 50, 100, Yellow, Navy)
    LCD.Number4(Volt, 5, 3, 150, 100, Yellow, Navy)

    LCD.Text4('Ohms:', 50, 200, Yellow, Navy)
    LCD.Number4(Ohms, 6, 1, 150, 200, Yellow, Navy)
    print(Volt, Ohms)
    sleep_ms(200)
```

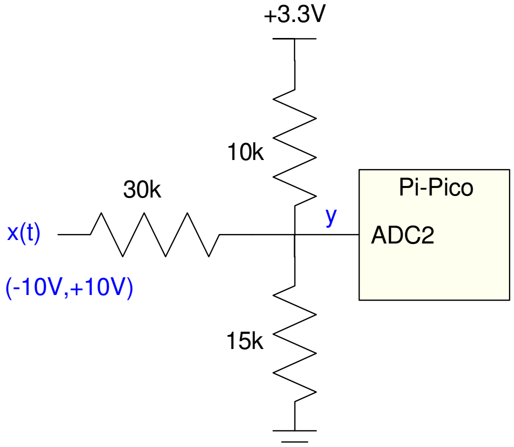
Oscilloscope

An oscilloscope is simply a volt-meter which displays the voltage (y axis) vs time (x axis).



Displaying voltage (y-axis) vs. time (x-axis)

Hardware: -10V to +10V: The required hardware is the same as you had for using a PIC as a volt meter. If you want to measure -10V to +10V, the following circuit converts this signal to (0V, 3.3V) for the Pi-Pico



Circuit to allow a Pi-Pico to read -10V to +10V

Software: One trick to speed up the program execution time is at each time-point,

- Erase the previous voltage at that time and
- Draw in the newly measured voltage

To do this, an a 421x1 array of values is stored (the y-coordinate of the pixel). When updating the display at time-point x,

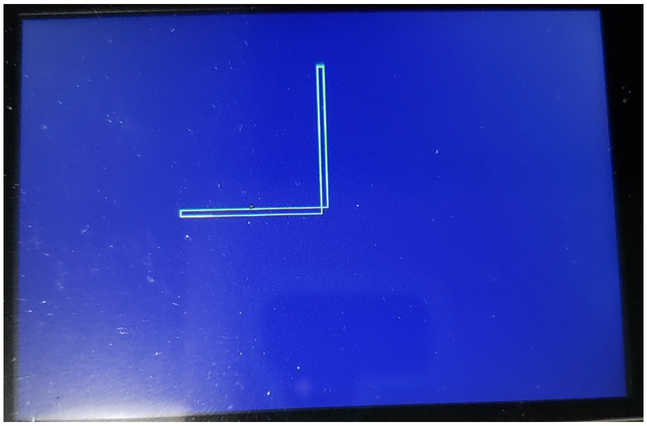
- The previous value of y(x) is set to the background color, and
- The current value of y(x) is set to yellow

This speeds up program execution - although it also means you're erasing the grid lines over time.

```
# Oscilloscop Code

import LCD

from machine import ADC
from time import sleep_ms

a2d0 = machine.ADC(1)

Navy = LCD.RGB(0,0,10)
Yellow = LCD.RGB(150,150,0)
Grey = LCD.RGB(50,50,50)

Xmin = 50
Xmax = 470
Ymin = 10
Ymax = 280
dX = (Xmax - Xmin)/10
dY = (Ymax - Ymin)/10

LCD.Init()
LCD.Clear(Navy)
for i in range(0,11):
    LCD.Line(Xmin, int(Ymin+i*dY), Xmax, int(Ymin+i*dY), Grey)
    LCD.Line(int(Xmin+i*dX), Ymin, int(Xmin+i*dX), Ymax, Grey)

Y = []
for i in range(Xmin, Xmax+1):
    Y.append(0)

k = (Ymax - Ymin) / 65535

X = Xmin
i = 0

while(1):
    a0 = a2d0.read_u16()
    LCD.Pixel2(int(X), int(Y[i]), Navy)
    Y[i] = k*a0 + Ymin
    LCD.Pixel2(int(X), int(Y[i]), Yellow)
    X += 1
    i += 1
    if(X > Xmax):
        X = Xmin
        i = 0
    sleep_ms(10)
```

Joystick X&Y

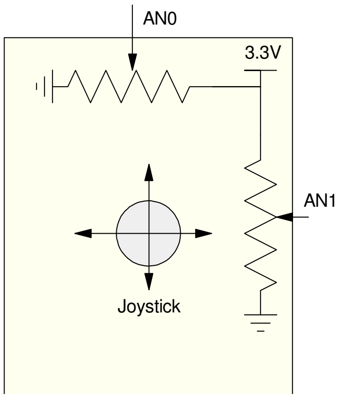
Just for fun, display the (x,y) position of the joystick on the Pico Breakout Board.



Displaying the (x,y) position of the joystick

Hardware: No additional hardware is needed with the Pico Breakout Board - the joystick is already wired up. As you move the joystick, the voltage applied to the analog inputs varies from 0V to 3.3V

- AN0: left-right motion
- AN1: up-down motion



The joystick is connected to two potentiometers, which vary the voltage on AN0 and AN1 based upon the joystick position

Software: The actual voltage sent to the analog inputs varies from 0V to 3.3V. This can be converted to a positive and negative signal by

- recording the voltages at the neutral position, and then
- subtracting this voltage from all subsequent readings

That is what the following code does

- On startup, it records the A/D readings (variable x0, y0), interpreting these as the neutral position
- All subsequent readings subtract this from the value you are currently reading

In addition, to speed up code execution, two rectangles are drawn

- A horizontal rectangle for the left-right motion (AN0)
- A vertical rectangle for the up-down motion (AN1)

Drawing on the diagonal is avoided since this is 10x to 20x slower than horizontal / vertical lines.

Also also, each rectangle is drawn twice:

- The first time is drawn in the background color (Navy) to erase the previous image
- The second time is in yellow to show the current joystick position.

It is faster to erase the previous rectangle in this manner than to erase the entire display.

```
# Joystick Position

import LCD

from machine import ADC
from time import sleep_ms

a2d0 = machine.ADC(0)
a2d1 = machine.ADC(1)

Navy = LCD.RGB(0,0,10)
Yellow = LCD.RGB(150,150,0)
Grey = LCD.RGB(50,50,50)

LCD.Init()
LCD.Clear(Navy)

k = 300 / 65535
x0 = a2d0.read_u16()
y0 = a2d1.read_u16()
x = 0
y = 0
while(1):
    a0 = a2d0.read_u16()
    a1 = a2d1.read_u16()
    LCD.Box(240,160,240+x,165,Navy)
    LCD.Box(240,160,245,160+y,Navy)
    x = int((a0 - x0)*k)
    y = -int((a1 - y0)*k)
    LCD.Box(240,160,240+x,165,Yellow)
    LCD.Box(240,160,245,160+y,Yellow)
    sleep_ms(20)
```

Bouncing Ball

The graphics display on the Pi-Pico can do pretty good animation as well - so long as you keep the figures fairly simple. For example, draw a ball bouncing around the display

Bouncing Ball simulation. The ball will bounce when it hits a wall

Software: This is actually a fairly involved program.

The acceleration on the ball at any given time is

- 0 in the x-direction
- -9.8 m/s2 in the y-direction (gravity)

Every 0.1 second (dt), velocity and position is updated using integration

$$\dot{x}(t) = \int \ddot{x}(t) \cdot dt$$

$$x(t) = \int \dot{x}(t) \cdot dt$$

In code, Euler integration is used since it is simple and doesn't require knowledge of previous values. Other (and better) forms of integration could be used

```
dx = dx + ddx * dt
x = x + dx * dt
```

To model reflection, the sign of the velocity is flipped when you encounter a wall. This creates a lossless system where the ball keeps bouncing around forever and ever.

Code:

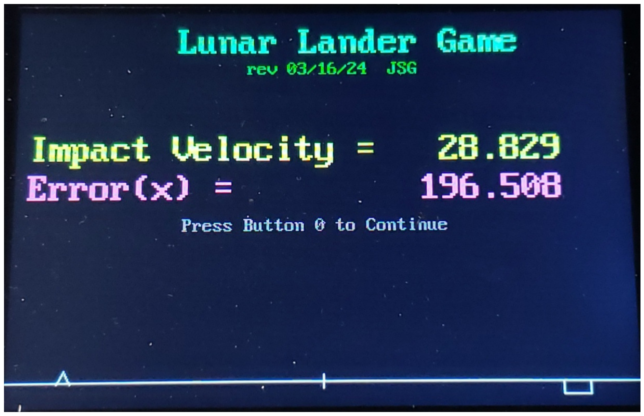
```
# Bouncing Ball

import LCD
from time import sleep_ms

Navy = LCD.RGB(0,0,10)
Yellow = LCD.RGB(150,150,0)
Grey = LCD.RGB(50,50,50)

LCD.Init()
LCD.Clear(Navy)

Xmin = 10
Xmax = 470
Ymin = 10
Ymax = 310
LCD.Box(Xmin,Ymin,Xmax,Ymax,Yellow)

x = 10
y = 300

dx = 10
dy = 0

dt = 0.1

zx = x
zy = y

# ball radius
r = 5

while(1):
    ddy = -9.8
    ddx = 0

    dy += ddy*dt
    dx += ddx*dt

    y += dy*dt
    x += dx*dt

    if(x+r > Xmax):
        dx = -abs(dx)
    if(x-r < Xmin):
        dx = abs(dx)
    if(y+r > Ymax):
        dy = -abs(dy)
    if(y-r < Ymin):
        dy = abs(dy)

    LCD.Circle(zx, 320-zy, r, Navy)
    zx = x
    zy = y
    LCD.Circle(x, 320-y, r, Yellow)
    sleep_ms(10)
```

Lunar Lander Game

Finally, an old arcade game *Lunar Lander*. The goal here is land at the center of the screen with

- minimum error in the left-right direction, and
- minimal velocity upon impact.

Impact is defined as the time when the y-position of the lander is zero or less.



Lunar Lander Game: Use the joystick and guide the lunar lander to the target

Software: The input is thrust (acceleration) set by the joystick

- left-right applies thrust in the left-right direction. Thrust is proportional to joystick position.
- up-down applies thrust in the up-down direction.

In addition, you have gravity (2.35 m/s2 on the moon) pulling you down.

In the following code, the thrust is displayed using bar-graphs. In the arcade game, you only have a limited amount of fuel as well. This latter feature is not incorporated in the following code.

Similar to the bouncing ball, Euler integration is used to update velocity and position.

$$\dot{x}(t) = \int \ddot{x}(t) \cdot dt$$

```
dx += ddx * dt
```

$$x(t) = \int \dot{x}(t) \cdot dt$$

```
x += dx * dt
```

Euler integration isn't a great form of numerical integration, but it's simple, it doesn't need old data, and it's good enough for this application.

In addition, this program uses a variable *flag* to kick out of the main while-loop. When the y-coordinate of the lander becomes negative, the game ends and your impact velocity and error in the x-position is displayed.

```
# Lunar Lander

from machine import Pin, SPI
import time
import utime
import LCD

a2d0 = machine.ADC(0)
a2d1 = machine.ADC(1)

B0 = Pin(15, Pin.IN, Pin.PULL_UP)
B1 = Pin(14, Pin.IN, Pin.PULL_UP)

led = Pin(17, Pin.OUT)
led2 = Pin(16, Pin.OUT)

a2d0 = machine.ADC(0)
a2d1 = machine.ADC(1)

DataX = [];
DataY = [];

# Main Routine

led.value(1)
LCD.Init()

Navy = LCD.RGB(0, 0, 5)
White = LCD.RGB(150,150,150)
LtGreen = LCD.RGB(50,150,50)
DkGreen = LCD.RGB(0,100,0)
Yellow = LCD.RGB(150,150,0)
Pink = LCD.RGB(150,50,100)
Grey = LCD.RGB(50,50,50)

flag =0
while(flag == 0):
    if(B1.value() == 0):
        flag = 1

    LCD.Clear(Navy)
    LCD.Text2('Lunar Lander Game',125, 10, LtGreen, Navy)
    LCD.Text('rev 03/16/24  JSG',180, 40, DkGreen, Navy)

    LCD.Line(0,300,480,300,White)
    LCD.Line(240,305,240,295,White)

    x = 10.0
    y = 250.0
    dx = 0.0
    dy = 0.0
    dt = 0.1

    x0 = a2d0.read_u16()/2000
    y0 = a2d1.read_u16()/2000
    by = 320
```

```
bx = 320

    while( y > 0 ):
        fx = a2d0.read_u16()/2000 - x0
        fy = a2d1.read_u16()/2000 - y0

        ddx = fx
        ddy = fy - 2.35

        LCD.Lander(x,300-y,Navy)
        x = x + dx*dt
        y = y + dy*dt

        dx = dx + ddx*dt
        dy = dy + ddy*dt

        LCD.Lander(x,300-y,White)

        LCD.Box(400,300,420,bx,Navy)
        LCD.Box(422,300,442,by,Navy)
        by = int(300-fy*10)
        bx = int(300-fx*10)
        LCD.Box(400,300,420,bx,White)
        LCD.Box(422,300,442,by,White)

        time.sleep(0.01)


    LCD.Text2('Impact Velocity = ',10,100,Yellow, Navy)
    LCD.Number2(abs(dy), 6, 3, 300, 100, Yellow, Navy)
    LCD.Text2('Error(x) = ',10,132, Pink, Navy)
    LCD.Number2(abs(x-240), 6, 3, 300, 132, Pink, Navy)
    LCD.Text('Press Button 0 to Continue', 130, 170, Grey, Navy)

    while( (B0.value() == 1) & (B1.value() == 1)):
        if(B1.value() == 0):
            flag = 1

print('Game Over')
```

Summary

Once you have a graphics display, getting information out is pretty easy, and the results look good. There are limitations on the graphics display, however:

- It takes about 100ms to clear the entire display. This causes flicker and slows down the entire program if you keep clearing and redrawing images.
- Text can be output - but the prettier and larger fonts take up a lot of program memory and are slow to output.
- Graphics can be output - but horizontal and vertical lines are a lot faster to update than diagonal lines.
- It's usually faster to erase part of an image (redraw using the background color) than to clear the entire display.

References

Pi-Pico and MicroPython

- https://github.com/geeekpi/pico_breakboard_kit
- https://micropython.org/download/RPI_PICO/
- https://learn.pimoroni.com/article/getting-started-with-pico
- https://www.w3schools.com/python/default.asp
- https://docs.micropython.org/en/latest/pyboard/tutorial/index.html
- https://docs.micropython.org/en/latest/library/index.html
- https://www.fredscave.com/02-about.html

Pi-Pico Breadboard Kit

- https://wiki.52pi.com/index.php?title=EP-0172

Other

- https://docs.sunfounder.com/projects/sensorkit-v2-pi/en/latest/
- https://electrocredible.com/raspberry-pi-pico-external-interrupts-button-micropython/
- https://peppe8o.com/adding-external-modules-to-micropython-with-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-raspberry-pi-pico/
- https://randomnerdtutorials.com/projects-esp32-esp8266-micropython/