Fun with LCD Graphics

ECE 476 Advanced Embedded Systems

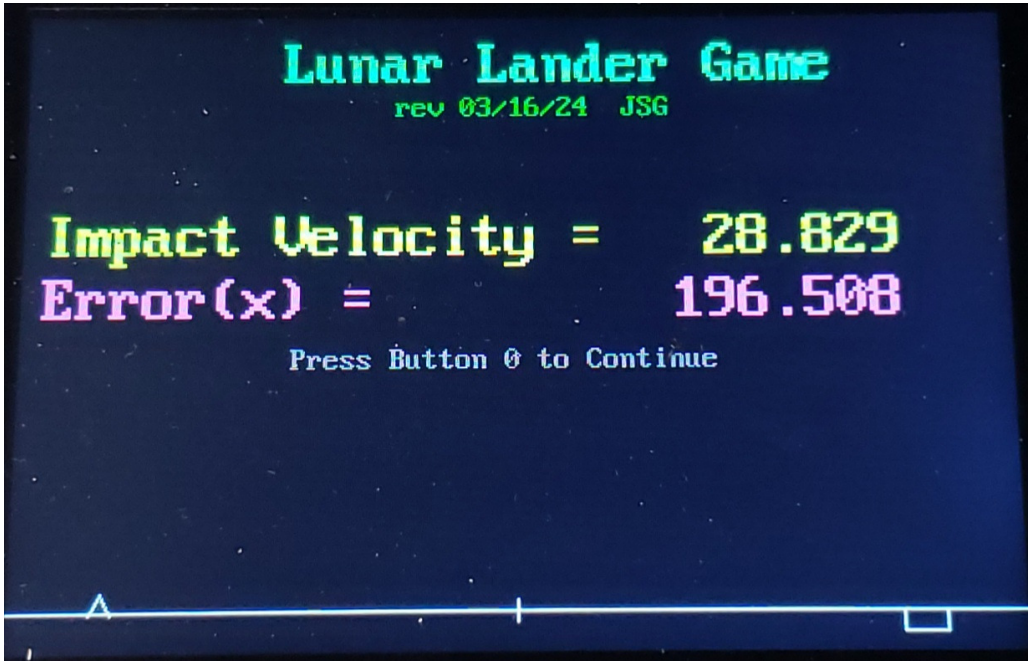
Jake Glower - Lecture #13

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

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

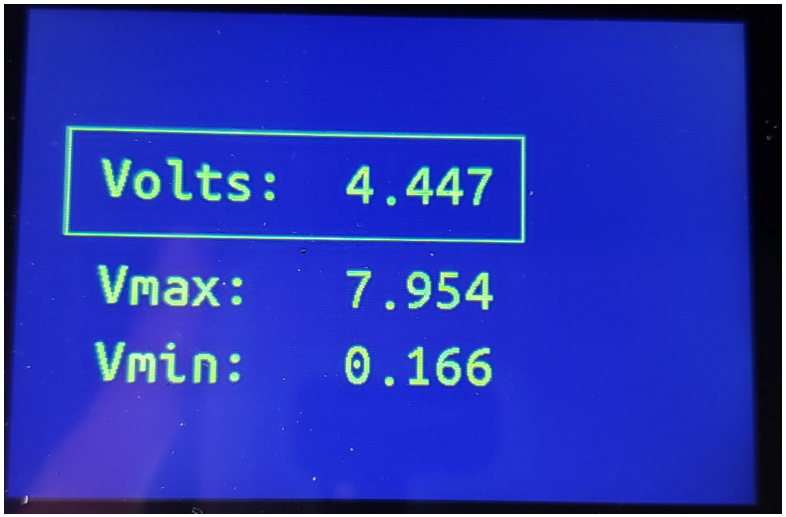
- Output text, such as the voltage or resistance attached to the Pi-Pico
- Display graphics, such as the x-y position of the joystick, and
- Do animation, such as a bouncing ball or a lunar lander game.



Volt Meter

Turn your PIC into a volt meter

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

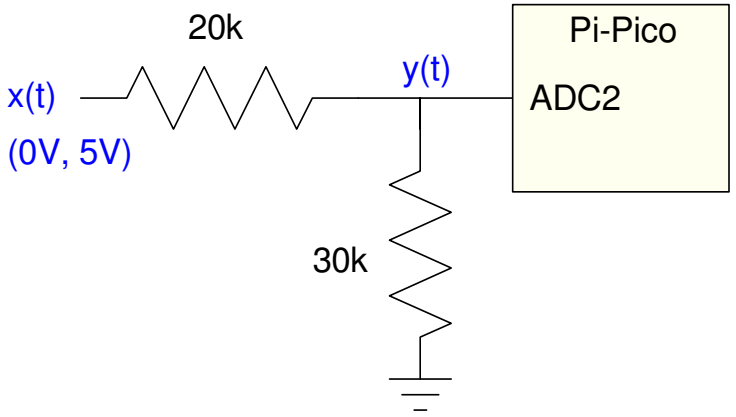


Hardware: 0V to +5V

- A/D input = (0V, 3.3V)
- Reduce the voltage using two resistors

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

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



Hardware: -10V to +10V Inputs:

- Output = 0V to 3.3V

Use three resistors and a weighted average

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

If you have 0V and 3.3V available, rewrite 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

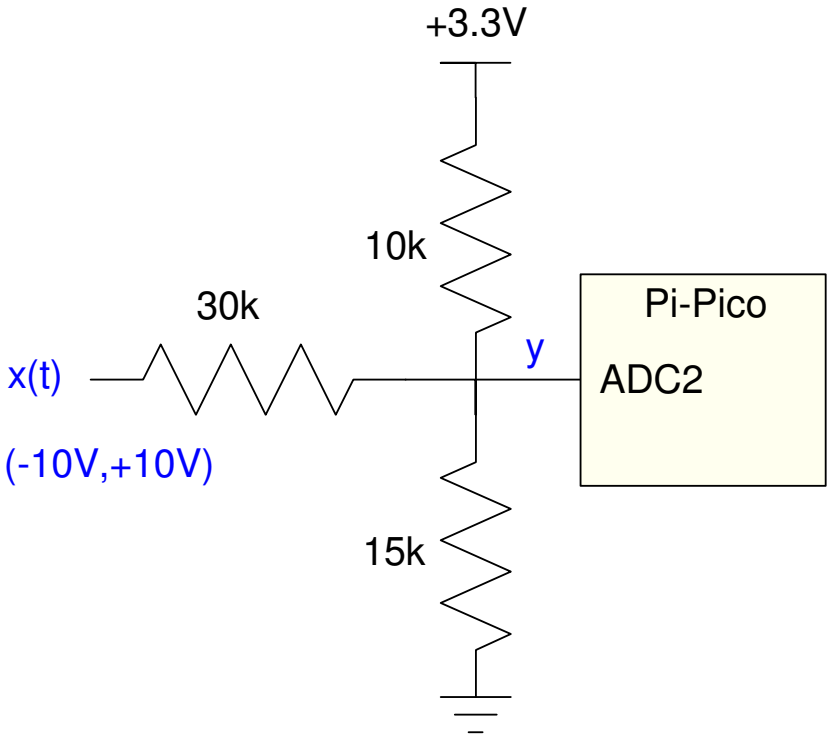
$$R_x = \frac{R}{0.165} = 30.3k \approx 30k$$

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

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

The following circuit converts

- input (x): -10V to +10V
- output (y): 0V to +3.3V



In software, the computed votlage is

$$V_x = \left(\frac{20V}{65{,}535}\right) \cdot A/D - 10$$

Code:

- Input = A/D channel 1

LCD displays

- Voltage
  - -10V to +10V
- max voltage
- min voltage

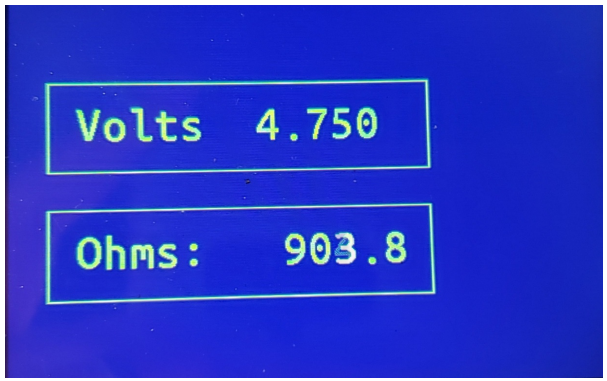
- Resolution = 4.9mV
  - 20V / 4096

```
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 = 20 / 65535
Vmax = -999
Vmin = 999

while(1):
    a0 = a2d0.read_u16()
    Volt = k*a0 - 10
    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

- Convert resistance to voltage



Hardware: A voltage divider works

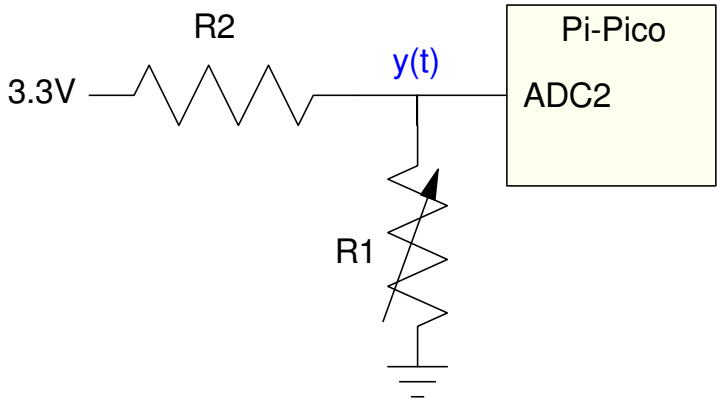
$$V = \left(\frac{R_1}{R_1+R_2}\right) 3.3V$$

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

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

a0 = raw A/D reading.

- Best sensitivity when R1 = R2.



Software

- Volt Meter &
- Ohm-Meter

Resolution = 1 Ohm

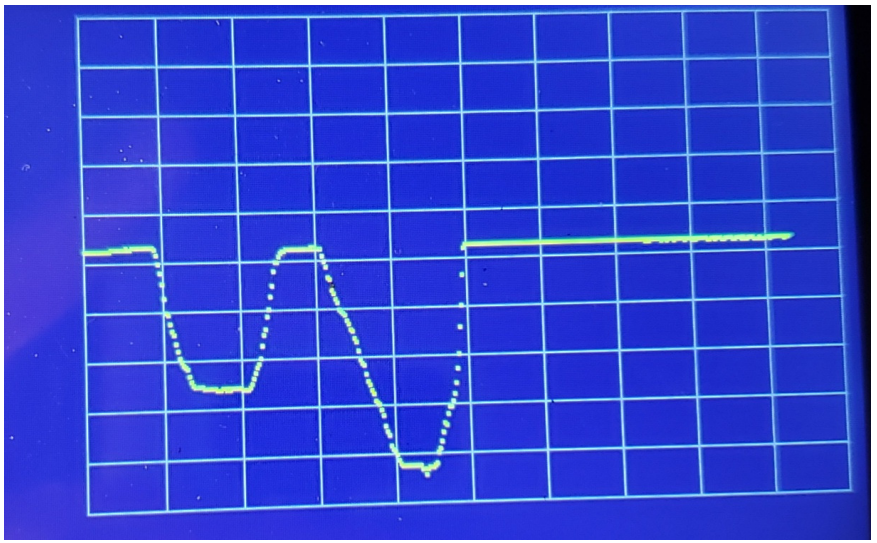
- R1 = R2 = 1k

```
# 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

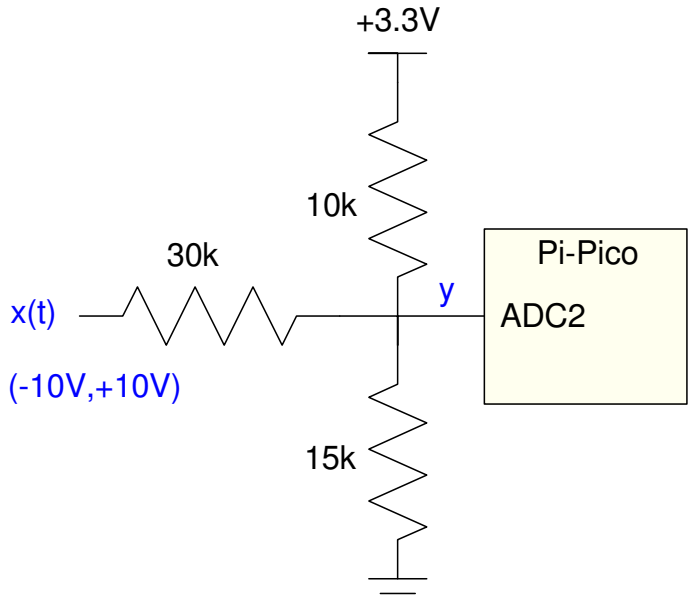
Display

- y axis: voltage
- x axis: time



Hardware: -10V to +10V:

- 0 = -10V
- 65535 = +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.

Software:

Set up a 421x1 array (y)

Each sample = 10ms

Y = voltage

- Ymin to Ymax
- 10 to 280

At each time point

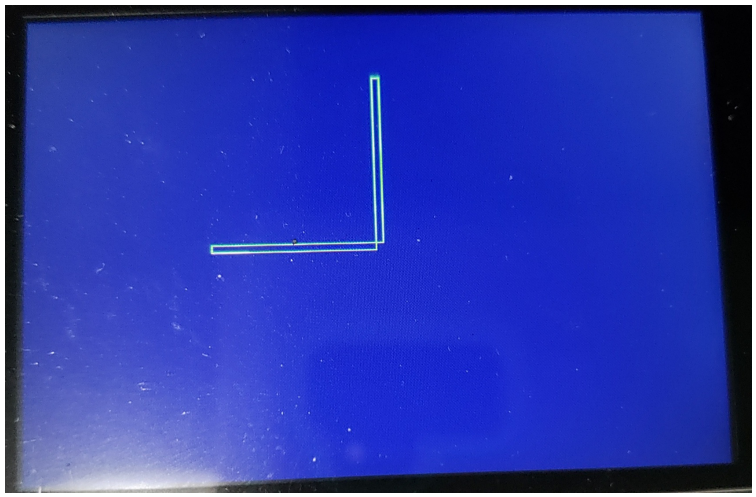
- erase the last point
    - Pixel in Navy
- draw the current point
    - in yellow

```
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

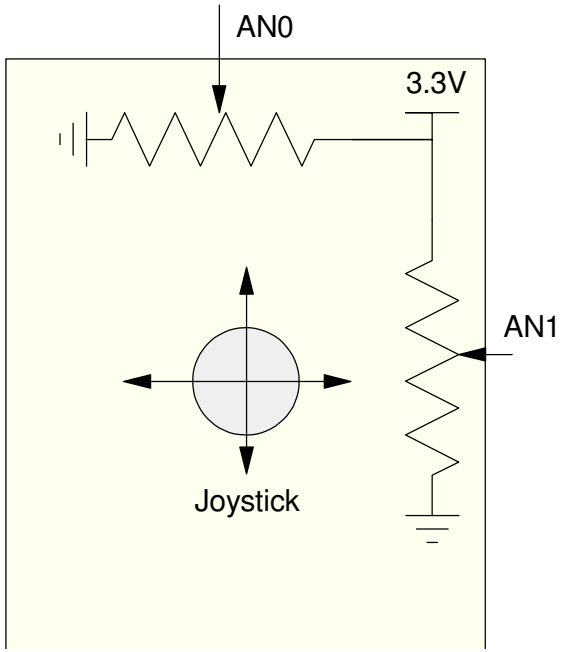
Display the (x,y) position of the joystick

- Display X and Y on the LCD



Hardware:

- Nothing needs to be added
- XY joystick part of Pi-Pico Breadboard
- AN0: left-right motion
  - 0V to 3.3V
- AN1: up-down motion
  - 0V to 3.3V



Software:

On startup, record X and Y

- interprits these as the neutral position

Every 20ms (main loop)

- Read X and Y
- Erase the old box
  - color = navy
- Draw new box to X & Y
  - color = yellow
- horizontal and vertical to speed up code

```
# 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

Simulate a ball bouncing around the display

Software:

- Actually a fairly involved program.

Acceleration is

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

Every 0.1 second (dt), update velocity and position

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

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

Code

- not the entire program

Use Euler integration

- Simple
- Works

When you hit a wall

- Bounce back
- (change sign of velocity)

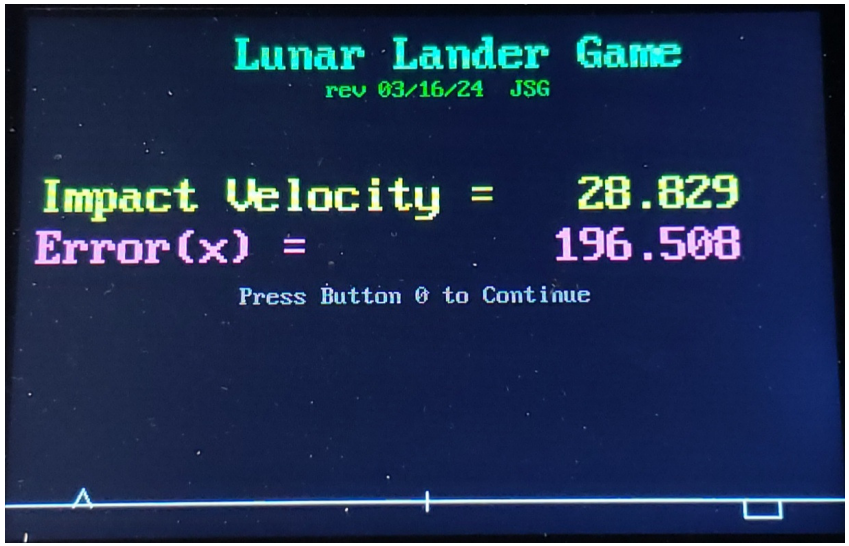
```
import LCD
from time import sleep_ms

:
:

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

Recreate the acade game

Goal:

- Land on the target on the planet
- With minimal velocity
  - In the X direction
  - in the Y direction
- The XY joystick controls the thrust
  - Neutral position = no thrust
  - Can apply in the X and Y directions



Note: The arcade game also had limited fuel

- Not incorporated in the game

Software

The input is thrust

- acceleration set by the joystick
- L/R: thrust in the x direction.
  - proportional to joystick position.
- U/D: thrust in the y direction
  - Also proportional to position

Gravity pulls you down

- 2.35 m/s2 on the moon

Game stops when you hit

- y < 0

```
# Lunar Lander
:
:

    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',10,100,Yellow, Navy)
    :
    :
```

Software (cont'd)

Euler integration is used

- Simple
- Works

Every 10ms

- Clear the last lander position and
- Clear the last thrust XY display
  - color = navy
- Redraw these in white
  - faster than redrawing the whole display

```
# Lunar Lander
:
:

    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',10,100,Yellow, Navy)
    :
    :
```

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.