Lab 1-2: Introduction to Raspberry PI and Soft Real Time Systems

RTES

September 23, 2013

1 Introduction

At theory lectures you have learned several concepts related with scheduling in *real-time* systems. In a *hard real-time system*, missing a dealdine is considered as a total system failure and should never occur. In *soft and firm real-time systems*, we can allow some outputs given after the deadline. Gaming is a nice example of soft real-time systems. In gaming and other applications, the rending is produced at a certain rate (Frames Per Second, FPS), where it is acceptable if some frames are missed and discarded. Please note that, strictly speaking, chess is in fact a hard real-time system.

In this laboratory, we will use a Raspberry Pi system to implement a soft real-time system in an object oriented language.

2 Raspberry Pi

2.1 Board description

The Raspberry Pi (RPi) is a small credit-card-sized computer developed in the United Kingdom by the Raspberry Pi Foundation. The board is of the size of a credit card and was developed with the intention of promoting the teaching of basic computer science in schools.

RPi boards went into mass production at 2012. The RPi foundation and project has the support of Cambridge University and Broadcom Inc.

There are two models of RPis distributed, model A and model B. Their differences are seen in the following table

	Model A	Model B
Price:	$25	$35
SoC:	BCM2835 (CPU + GPU + DSP + SDRAM + USB)	
CPU:	ARM1176JZF-Sx at 700 MHz (ARM11)	
GPU:	BCM VideoCore IV, , OpenGL ES 2.0, -2 y VC-1 (licensed), 1080p30 H.264/MPEG-4 AVC	

Memory:	256 MB (comp. GPU)	512 MB 15/10/2012
Input Video:	MIPI connector CSI (RPF webcam)	
Video:	RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), DSI (LCD)	
Audio:	Jack 3.5 mm, HDMI	
Storage:	Tarjeta SD / MultiMediaCard MMC / SDIO	
Network:	None	10/100 Ethernet (RJ-45) via hub USB
I/O:	8 x General Purpose Input/Output GPIO, Serial Peripheral Interface SPI, Ii^{2}C, Universal Asynchronous Receiver-Transmitter UART	
RTClk:	None	
Power:	500 mA, (2.5 W)	700 mA, (3.5 W)
Input Power:	5 V vía Micro USB o GPIO header	
Size:	85.60mm × 53.98mm (3.370 × 2.125 inch)	
OS:	GNU/Linux: Debian (Raspbian), Fedora (distribución Linux) (Pidora), Arch Linux (Arch Linux ARM) , Slackware Linux . RISC OS	

The board design includes a general purpose input output connector (GPIO) allowing a simple interface to external devices through a simple male insulation-displacement connector (IDC). The connector has a number of different types of connection on them. These are:

- True GPIO (General Purpose Input Output) pins that you can use to turn LEDs on and off etc.
- I^{2}C interface pins that allow you to connect hardware modules with just two control pins
- SPI interface
- Serial Rx and Tx pins for communication with serial peripherals

In addition, some of the pins can be used for PWM (pulse Width Modulation) for power control and another type of pulse generation for controlling servo motors called PPM (Pulse Position Modulation). The complete list of chipset GPIO pins which are available on the GPIO connector is: 0, 1, 4, 7, 8, 9, 10, 11, 14, 15, 17, 18, 21, 22, 23, 24, 25. On the Revision 2.0 this list changes to: 2, 3, 4, 7, 8, 9, 10, 11, 14, 15, 17, 18, 22, 23, 24, 25, 27, with 28, 29, 30, 31 additionally available on the P5 header.

As noted above, P1-03 and P1-05 (SDA0 and SCL0 / SDA1 and SCL1) have 1.8 kohm pull-up resistors to 3.3 V.

RPi commonly runs Debian GNU/Linux from the SD card inserted in the board. As in the laboratory we have a Debian OS installed in the RPi system we can code the board through a number of computing languages, including:

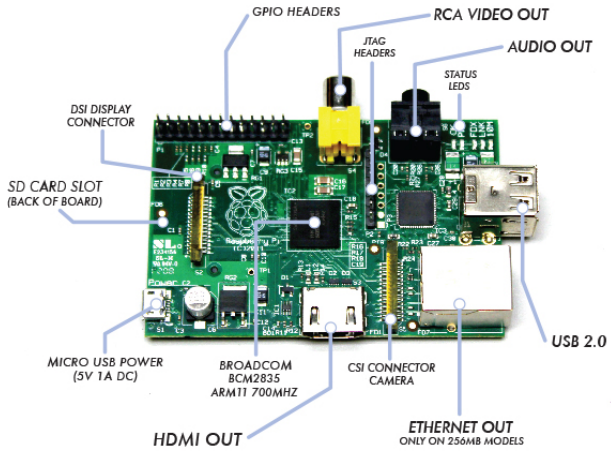

Figure 1: Raspberry Pi picture and Components description.

Scratch Scratch is an entry-level programming language that comes as standard with the Raspberry Pi distribution (Raspbian). Scratch was originally created by the Lifelong Kindergarten Group at the MIT Media Lab in Boston, U.S., with an aim to help young people learn mathematical and computational concepts while having fun making things.

Python Python is one of the primary programming languages hosted on the Raspberry Pi. Python is named after Monty Python's Flying Circus, the comedy team who brought us Life of Brian.

C As you might already know, the C Programming language was written by Dennis Ritchie, using Brian Kernighan's B language as its model.

C++ C++ was developed by the Danish developer Bjarne Stroustrup as a way to add object oriented features to C.

Perl Perl was develooped by Wall at 1987 while working at Unisys. Perl is extremelly fast processing string objects and allowed the first time the inclusion of CGI websites, where the website was composed programmatically at at access time. In fact, this feature enabled ecommerce, so sites such as Amazon and eBay to come into being.

Erlang It was designed by Joe Armstrinc at Ericsson in 2986 to support distributed, fault-tolerant, soft-real-time, non-stop applications. It supports hot swapping, so that code can be changed without stopping a system. It became open source in 1998.

In this laboratory we will focus on the use of python in the Raspberry Pi while coding a simple soft-real-time game.

Connect your RPi to a monitor through the HDMI connector, a keyboard and power the system. After the boot process, it will show a prompt login. Use *pi* as username and *raspberry* as password. Navigate through the filesystem and get familiarized with the board, its components and the OS.

3 Python in Raspberry Pi

3.1 Introduction to pygame

Pygame is a cross-platform library designed to make it easy to write multimedia software, such as games, in Python. Pygame requires the Python language and SDL multimedia library. It can also make use of several other python libraries. `pygame` python modules should be already installed on your RPi's. If there are not, just install them through the following command, provided that you have access to Internet.

```
$ sudo apt-get install python-pygame
```

We will write a simple pygame program. In order to create a pygame application, you should include five steps in your code:

1. Import the pygame library.
2. Initialise the pygame library.
3. Create a window.
4. Initialise game objects.
5. Start an infinite loop.

Importing pygame library is easy:

```
1 import pygame
2 import sys
```

which will import pygame methods into python.

To inizialise the pygame library, just call to

```
1 pygame.init()
```

which initialises all the pygame modules.

After initializing the pygame library, we create a window that will be used to render the game graphics. This can be achieved through the code below:

```
1 screen = pygame.display.set_mode((800, 600))
2 pygame.display.set_caption("My RTES Application")
```

In the fourth step we should load all assets required for our application to run, including images, sounds, initialise object positioning, set up state variables, and others.

Finally, the application loop is just a loop where we continuously handle events, check for input, move objects, and draw things.

Each iteration of the loop is called a frame.

In the design of an application, we should check carefully that each frame takes exactly the same time on any machine. Otherwise the speed at which the application runs will depend on the speed of the system.

```
clk = pygame.time.Clock()
while True:
  clk.tick(50)

  # Process events
  for event in pygame.event.get():
    if event.type == pygame.QUIT:
      sys.exit()

  # Clear the screen
  screen.fill((0, 0, 0))

  # Check input
  # Move objects
  # Draw objects

  # Update the screen
  pygame.display.flip()
```

We have created a `Clock` object and included a call to `clk.tick(50)` to enforce a rate of 50 frames per second. We also check whether there is a `QUIT` event to take care of (the user closed the screen) in order to call to `sys.exit()`. `screen.fill()` fills the `screen` object created in the third step with the colour given by the input tuple (in RGB). After whatever procedures we have invoked, none of them will be active on the `screen` object until we call to the `pygame.display.flip()` method. Remember that in python the indentation is important, as it determines the scope of the control structures such as the *while* command.

Type the previous code and be check that the pygame screen is created.

3.2 Soft Real Time

Next, we will display the rate at which the RPi and pygame are able to control our soft real-time system. We will first modify the previous code so that we draw on screen the measure Frames Per Second of the execution control loop implemented in pygame.

To do so, we will wrap our previous code as a class (PhonePong), which can be approached as follows:

First, initialise your python modules and create some constants that will be used in your program.

```
import pygame
import sys # for sys.exit()

# Define some constants, such as display size
#  and colors used
DISPLAY = 800,600
GREEN         = 25, 200, 50
WHITE         = 255,255,255
BLACK         = 0, 0, 0
YELLOW = 255, 255, 0
```

Let's create a class to draw and control de background on our game.

```
# A class to define the game background
class background:

        def __init__(self,dSize):



                # Our main background will be a surface object
                # dSize: width, heigth
                self.image = pygame.Surface(dSize)

                # which will be green-filled
                self.image.fill(GREEN)

                # and has to have a white band as net
                netWidth = dSize[0] / 80

                # rect(left, top, width, height)
                self.net = pygame.Rect(0,0,netWidth, dSize[1])
                self.net.centerx = dSize[0] / 2
                pygame.draw.rect(self.image, WHITE, self.net)

        def draw(self,display):

                # Do the actual draw
                display.blit(self.image, (0,0))
```

Create our main class, including methods for initialization and destruction of the object.

```
# Our main PhonePong class
class PhonePong:
        # This method is executed when the  object is created
        def __init__(self):
                pygame.init()
                # initialization code in here
                self.screen = pygame.display.set_mode(DISPLAY)
                pygame.display.set_caption("My RTES Phone Pong")

                # Creat our playground. obviously, in green.
                self.green = background(DISPLAY)


        # This method is executed when a new object
        #                 PhonePong is destroyed
        # Include here code to close all file descriptors

        def __del__(self):
                print "exit signal received"
```

Let's code the main method of our code. This method, named run will contain the main loop cycle as suggested in the pygame documentation. Please be aware of the indentation.

```
        # Main method, which contains the soft real-time loop
        def run(self):
                clock = pygame.time.Clock()
                while True:
                        clock.tick(10)

                        # Call to Handle Events method
                        self.handleEvents()

                        self.fps = clock.get_fps()


                        # Clear the screen
                        self.screen.fill(BLACK)
                        self.green.draw(self.screen)


                        # Move objects ...
                        # Draw objects ...

                        # Update texts
                         self.handleText()

                        # Update the screen
                        pygame.display.flip()
```

The next methods control which text is created at each frame.

```
        def handleText(self):
                # pick a font you have and set its size
                myfont = pygame.font.SysFont("Comic Sans MS", 20)
                # instantiate a label with our font
                label = myfont.render("PhonePong "+
                                "(fps: {:.1f})".format(self.fps) ,
                                 1, YELLOW)

                # place the label object on the screen
                #     at coordinates x=10, y=10
                self.screen.blit(label, (10, 10))
```

The last method will control all events happening in the game, including keystrokes, network input, quit signals and others.

```
        # In this method we will handle the game's events
        def handleEvents(self):
                # Network code...
                # Deal for all pending events in the pygame event queue
                for event in pygame.event.get():
                        if event.type == pygame.QUIT:
                                pygame.quit()
                                sys.exit()
```

And finally we include standard python code that controls standalone execution of the class.

```
# Standalone execution
if __name__ == '__main__':
        myGame = PhonePong()
        myGame.run()
```

Try to complete these tasks.

- Describe the code structure.
- Code and run this python program in your RPi.
- Code a second program, so that increases linearly the load on the RPi processor from no load to full cpu load in 1 minute.
- Modify the previous code so that you can store into a file the frames per second the pygame achieves during this minute.
- Plot the figure and discuss the results.
- Explain why this is a soft-real time system.
- Could we implement a hard real time system through python/pygame?

3.3 Phone Pong

In this second session, we will reuse our previous implementation of a soft real time loop in order to code a Pong game. Pong is one of the earliest arcade video games; it is a tennis sports game featuring simple two-dimensional graphics. Pong was one of the first video games to reach mainstream popularity[1]. The game was originally manufactured by Atari Incorporated (Atari), who released it in 1972. Allan Alcorn created Pong as a training exercise assigned to him by Atari co-founder Nolan Bushnell.

In this laboratory we will add a novel way to control the Pong game. In our case we will control each of the two player rackets through the accelerometers present in our modern cell phones. Data will be streamed to the RPi through UDP over the internal network. In order to generate the streams of UDP data

[1] http://en.wikipedia.org/wiki/Pong

Figure 2: An upright cabinet of Pong signed by Pong creator Allan Alcorn.

from the accelerometers we can employ free and open source Android packages such as SensorUDP or Sensorstream IMU+GPS. There is a similar software for IOS named Sensor Data.

To achieve this, we will need a more complete version of our PhonePong code, in this case including some rackets, a ball and a collision detector for each player.

- Look the pygame documentation. Code a Pong game employing pygame on the RPi. Use three sprites to code two rackets and a ball.
- Look the python documentation in order to learn how to implement a UDP server. Implement two UDP sockets for each player so that you can control each racket through UDP. Implement a parser to extract the sensor information from the UDP stream generated by your phone.
- Use and understand the difference of using UDPsocket.setblocking(0) or UDPsocket.setblocking(1) on the UDP server initialisation.
- Discuss the implemented control in terms of a real time system.
- Implement a second game that employs the additional sensors that you cell phone is offering (Optional)

3.4 Report

Please write a Laboratory Report following the structure of this text. The report should contain introduction, goals of each section and provide with a description of the results form each part. Include the graphics you consider relevant so that the development of the report is understandable. The short lab report and python code should be delivered before December 1st.