Reprinted from the MagPi Issue 25, July 2014. www.themagpi.com

MouseAir

A Control Panel for Cat Entertainment

Building a Control Panel

John Shovic

Guest Writer

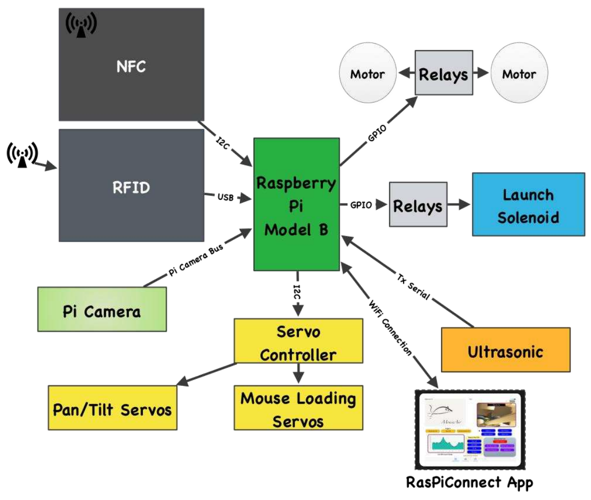
SKILL LEVEL : INTERMEDIATE

Introduction

I have always enjoyed building control panels for my projects. I used to laboriously build physical control panels with switches, meters and lights. Lots of lights. MouseAir is a complex project. 4 Servo Motors, 3 Sensors, 4 relays, 2 DC motors, one mother-of-all solenoids, one camera and one obnoxious cat, all working in sequence. All this to launch toy mice across the room.

I like to be able to change parameters, change modes and cause actions (such as entertaining a cat), whether sitting near the MouseAir launcher or across the world.

MouseAir System Diagram

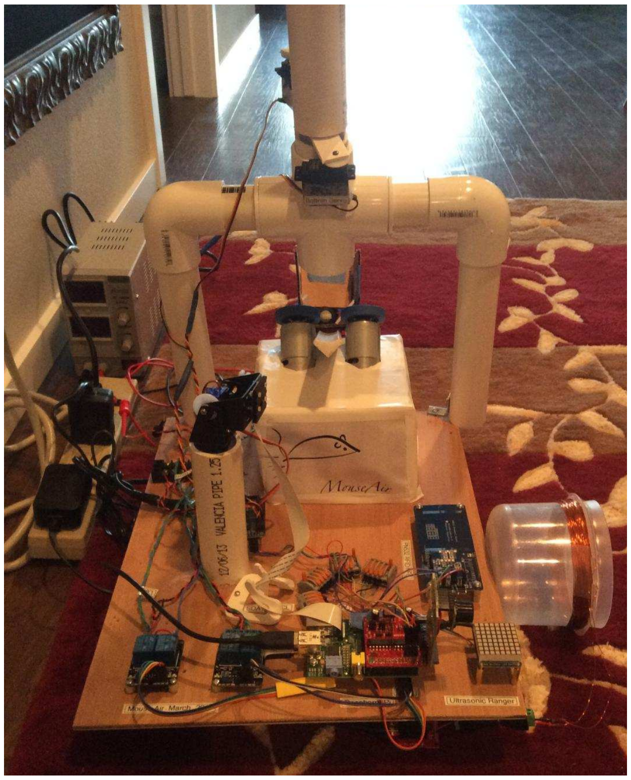


In this project at SwitchDoc Labs, I am using RasPiConnect (www.milocreek.com) to control MouseAir. RasPiConnect allows me to build multiple pages of controls, with graphs, webpages, pictures, streaming video (with a little more work!) and lots of lights and buttons on an iPad or iPhone without having to build and submit an app to the App Store. You build the controls on the phone or tablet and then modify a server on the Raspberry Pi to talk to the control panel. This is the second project for which I am using RasPiConnect. The first is Project Curacao, descripted in early issues of The MagPi.

Description of Mouse Air

The MouseAir system is built around a Raspberry Pi controlling all the devices necessary to launch the toys, connected to the iPad via a WiFi connection. We are using a Pi Camera on a pan / tilt structure to capture the cat events, examine the launching mechanism for jams, motion detection and even streaming video. It uses a solenoid to push a mouse between two rapidly spinning DC motors. Note the hand built 125KHz RFID antenna on the right side in the picture overleaf. See an early mouse launch at: http://www.switchdoc.com/2014/04/mouseair-prototype-video

The motors are a bit of an overkill. A properly loaded mouse can be shot 10 meters!



What Controls to Use

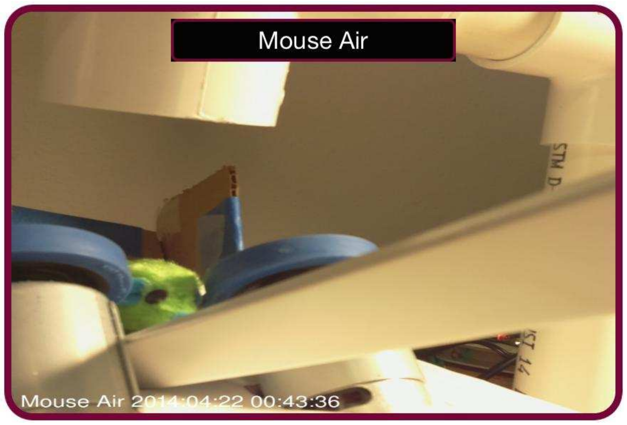
In designing a control panel using RasPiConnect, the first thing you have to do is choose which controls to use and where to place them on the iPad Screen. This is all done within the app on the iPad or iPhone. RasPiConnect has a collection of about 20 controls to choose from, all with special options and behaviours from meters to buttons to LEDs to constantly updated live graphs.

The first thing I did was decide what I wanted to control: which sensors are used for triggering, controlling and viewing the Pi Camera, be able to manually launch mice and manually control any part of the six step procedure to launch the mouse.

Every control in RasPiConnect has an input and response. The app sends an input to your Raspberry Pi (or Arduino, or both!) and the Raspberry Pi sends back a response.

Pi Camera display

For the camera display, I have chosen a Remote Picture Webview control. This control has a title and the response from the Raspberry Pi is in the form of an HTML page. It is very flexible and you can configure it for pictures or for streaming video with a little effort. Note the mouse peeking out.



Buttons for action

In RasPiConnect, there are two types of buttons: Action and Feedback Action. You use Action Buttons to do an action that does not require feedback (however, there are ways to provide feedback by refreshing other controls based on an Action Button tap). For example, "Take Picture".

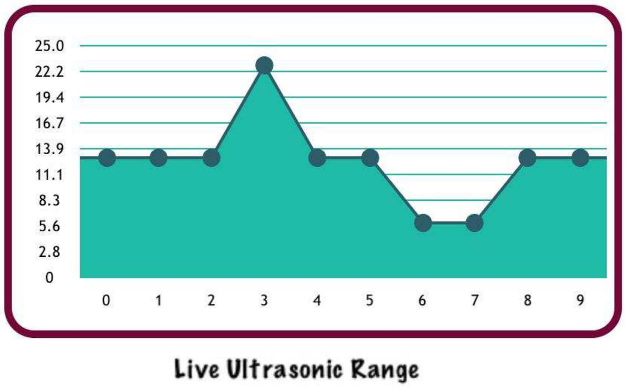
Feedback Buttons for Selections

Feedback Buttons are used for cycling through a set of options (such as "Off" and "On"). Tapping a Feedback Button sends an input to the Raspberry Pi (the current text in the button) and the response text from the Raspberry Pi replaces the button text. For example, tapping the "Ultrasonic Off" button will send "Ultrasonic Off" to the Raspberry Pi, turn the Ultrasonic sensor off and return the text "Ultrasonic On" to set up the button for the next tap. In Project Curacao, some buttons had 8 different states!

Live graphs for real time reports

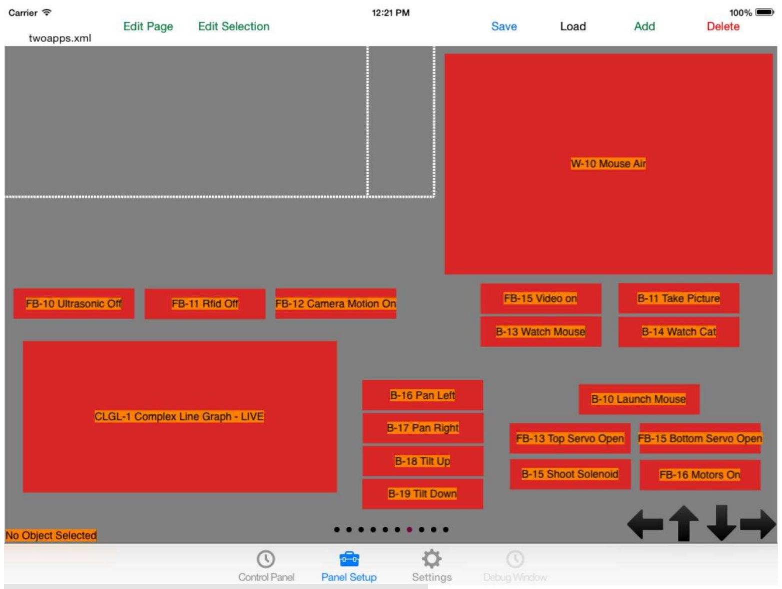
The live controls are a new feature in the lastest version of RasPiConnect. They are a collection of controls that will periodically (interval set by the user) ask the Raspberry Pi for any new information and update the iPad screen. I decided to use a Complex Line Graph Live control to give a continuous display of what distance the ultrasonic sensor is reading (is the cat walking by?)

Each time the iPad app asks for an update of the graph, the Raspberry Pi returns a response of the list of data points to be graphed and what text to use for the x axis labels, I usually set this display to update every second. Note that RasPiConnect will only allow you to use this feature if you are connected to WiFi on the iPad. We don't want to run up the mobile data bill!



Configuring RasPiConnect on the iPad

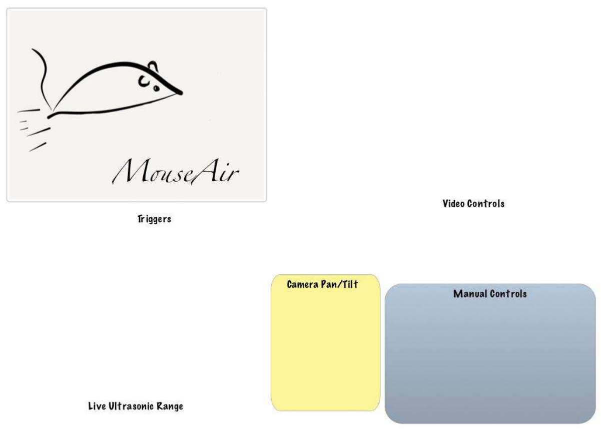
Each of the desired controls is placed on the iPad in the Panel Setup tab within RasPiConnect.



The design screen in RaspiConnect

Backgrounds

To finish off the control panel we add the MouseAir logo and accent text boxes onto the screen. To do this we build a custom background in the page (we used Grafio on an iPad to generate a JPG or PNG file) and then select this background graphic for the page.



The background image loaded into the app

Configuring RasPiConnectServer Software on the Raspberry Pi

The MouseCommand.txt file is first written by the RasPiConnectServer program and then read by MouseAir. When the command is complete, MouseAir writes "DONE" into the command file, telling RasPiConnectServer that it is finished and ready for the next command. Note that the RasPiConnect app keeps all commands in a queue and will not send another command until either a timeout occurs (programmable) or it gets a response from the Raspberry Pi.

The MouseAir Control Software

The MouseAir software operates in a loop, periodically checking for a new command from RasPiConnect, checking for triggers from RFID, Pi Camera motion and checking for an Ultrasonic trigger. The MouseAir software is available on http://github.com/switchdoclabs.

On the MouseAir side, the software for receiving commands from RasPiConnect is contained in `processcommand()`.

```
def processCommand():
  f = 
open("/home/pi/MouseAir/state/MouseComman
d.txt", "r")
  command = f.read()  f.close()

  if (command == "") or (command ==
"DONE"):
    # Nothing to do
    return False

  # Check for our commands
  pclogging.log(pclogging.INFO, __name__,
"Command %s Received" % command)

  print "Processing Command: ", command

  if (command == "FIREMOUSE"):
    fireMouse()    completeCommand()
    return True
  if (command == "TAKEPICTURE"):
    utils.threadTakePicture("Picture
Taken -RasPiConnect Command")
    completeCommand()
    return True

...

def completeCommand():
  f =
open("/home/pi/MouseAir/state/MouseComman
d.txt", "w")  f.write("DONE")
  f.close()
```

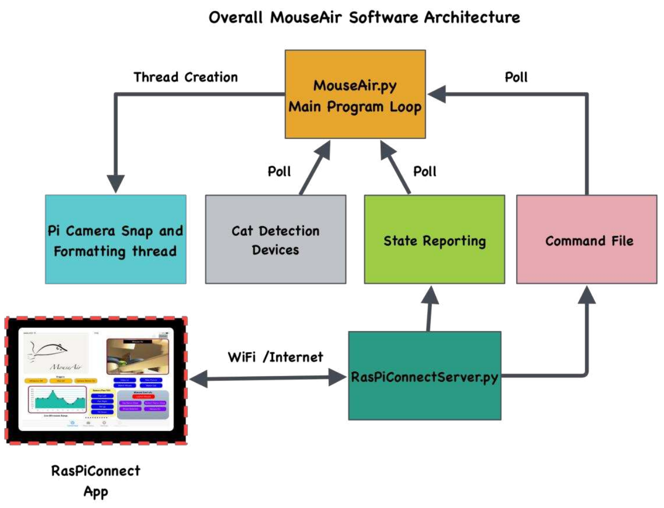
The RasPiConnectServer Software

RasPiConnectServer is a Python program provided for connection from a Raspberry Pi to the RasPiConnect app. Change the file Local.py (in addition config.py) to connect to the MouseAir software. MiloCreek provides full documentation for Local.py and the rest of the server at www.milocreek.com under the documentation link. Each button is pretty simple to do.

There are some other setup items such as setting URLs for your Raspberry Pi that are fully explained in the manual.

All of the software that you need to write is placed in Local.py. There is an example Local.py file provided in the server download. To illustrate how to write the interface, I will follow one entire button through the process.

I will use the Motors On button as an example. This button controls the DC motors that shoot the



mouse up in the air. It has two states, "On" and "Off". This makes it a perfect candidate for a Feedback Action Button.

When you add a control in RasPiConnect, you can set the control code (usually a unique identifier) for the button. By convention, each Feedback Action Button starts with "FB". Our motor control button a control code of "FB-16".

When you tap the button on RasPiConnect an XML message (or optionally a JSON or raw mode message) is sent from the iPad to the Raspberry Pi. The message is parsed by the RasPiConnectServer software and the results are presented to your customized code in the Local.py file. You don't have to deal with any of the parsing or handshaking, the libraries do all of this for us. The button is then presented to Local.py. We wrote one small routine to interface to the MouseAir.py command file.

```
defsendCommandToMouseAirAndWait(command):
```

The next section is the "money code", that is, the code where the functionality of the button is implemented.

```
  # object Type match
  if (objectType ==
FEEDBACK_ACTION_BUTTON_UITYPE):
    if (Config.debug()):
      print "FEEDBACK_ACTION_BUTTON_UTYPE
of %s found" % objectServerID

    # FB-16 -  turn motors on
    if (objectServerID == "FB-16"):
      #check for validate request
      # validate allows RasPiConnect to
verify this object is here
      if (validate == "YES"):
        outgoingXMLData +=
Validate.buildValidateResponse("YES")
        outgoingXMLData +=
BuildResponse.buildFooter()
        return outgoingXMLData
      # not validate request, so execute
      responseData = "XXX"
      if (objectName is None):
        objectName = "XXX"
      lowername = objectName.lower()
      if (lowername == "motors on"):
        print "set Motors On"
```

We now send the command to MouseAir.

```
        status =
sendCommandToMouseAirAndWait("MOTORSON")
        responseData = "motors Off"
        responseData =
responseData.title()
```

Note we have now "toggled" the button by sending "Motors Off" back to the app to set up the button for the next tap.

```
      elif (lowername == "motors off"):
        status =
sendCommandToMouseAirAndWait("MOTORSOFF")
        responseData = "Motors On"
        responseData =
responseData.title()
```

The default section is in case of a time out and the button becomes blank. In that case, we want the motors off.

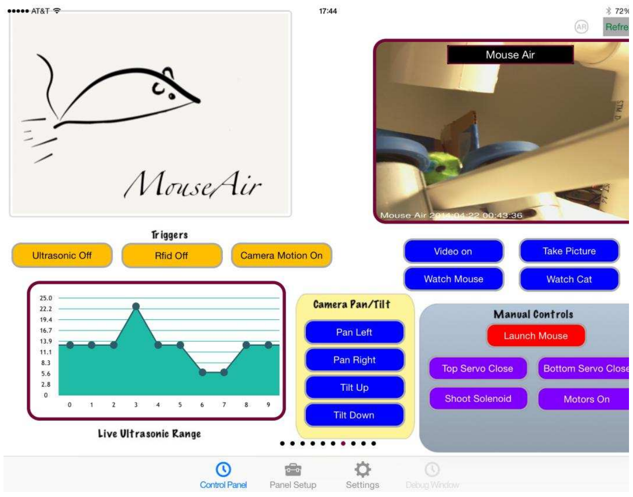
```
      # defaults to Motors off
      else:
        print "Motors Off"
        status =
sendCommandToMouseAirAndWait("MOTORSOFF")
        lowername = "Motors On"
        responseData = lowername.title()
```

FInally, the rest of the XML response is built. By the way, if you somehow screw up the XML, RasPiConnect just rejects it. There is error checking built into the App as well as checksum on each response.

```
      outgoingXMLData +=
BuildResponse.buildResponse(responseData)
      outgoingXMLData +=
BuildResponse.buildFooter()
      return outgoingXMLData
```

Looking at the code, you can see the command that is written to the MouseAir command file. The software then waits for the "DONE" and then sends the response (which is the text used for the button on the iPad. If you sent a "Motor On" command, the response to be sent back would be "Motor Off" and then the "Motor Off" would be displayed on the button on the app.

That is the complete cycle. This design pattern is used for all of the controls.



The completed final control panel for MouseAir

Conclusion

How to setup a control panel for your project is always a challenge. I like being able to control a project locally and across the Internet. MouseAir is a complicated project with a number of different things to adjust and to view. I found that RasPiConnect was a very good match and platform on which to build. I am already planning to use it on SwitchDoc Labs next project.

RasPiConnectServer is available on http://github.com/milocreek and the specific MouseAir RasPiConnect file (Local.py) is available on http://github.com/switchdoclabs.

For more information about MouseAir see the authors blog at www.switchdoc.com.

For more information about RasPiConnect see www.milocreek.com