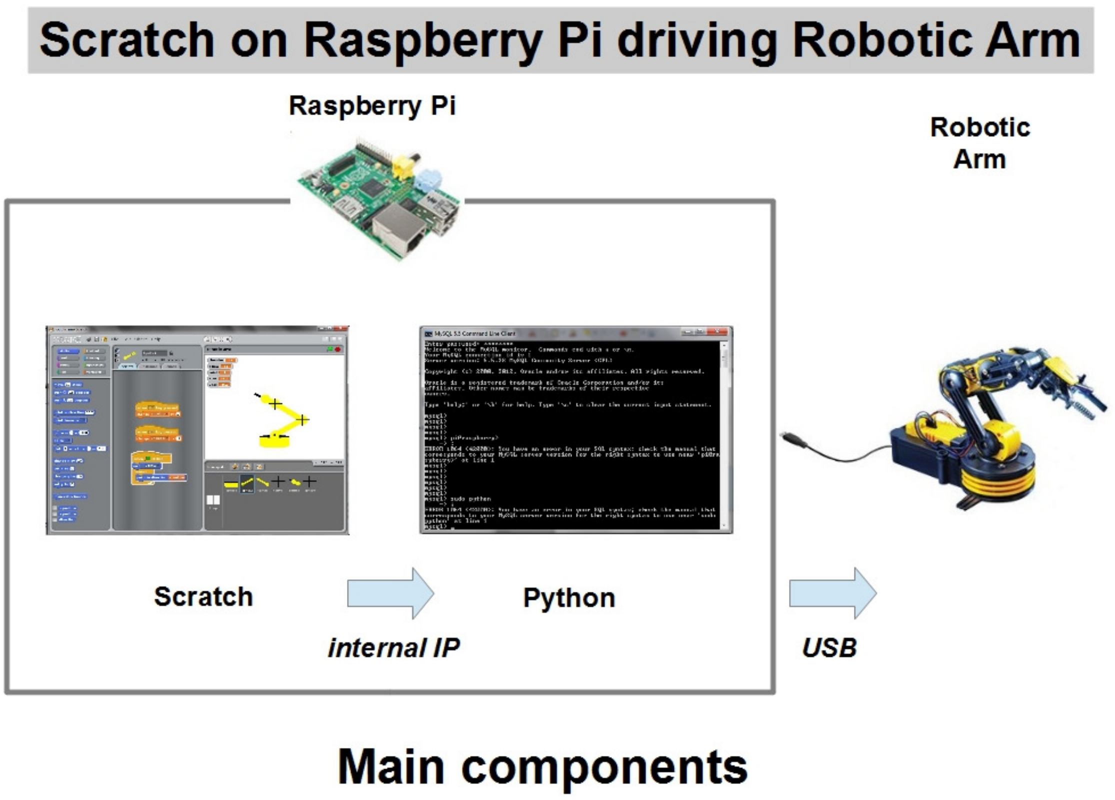
Using Scratch on the Raspberry Pi to drive a Robotic Arm

This document is currently available at www.mattrealm.me.uk/media/arm_doc.pdf.

This project came about as a suitable learning exercise for me on the Raspberry Pi using Python.

Some folk relish the thought of hooking up the Raspberry Pi to motors and suchlike, using its GPIO pins, colourful ribbon cable and breadboards. I may graduate to that one day, but meanwhile the Robotic Arm sold by Maplin features motors and colourful wiring – and, instead of multiple breadboarded connections, a single, neat USB cable.

It can obviously be connected to a Raspberry Pi – but how to make it work? Fortunately Issue 3 of The MagPi in July 2012 gave the game away, with full details of how to drive it from Python. Fine as far as it goes, but I'd like to use it in my after-school Computer Club for Year 4 children (aged 8 to 9) at Oakfield First School, who are into Scratch, not Python.



The answer was to find out how Python can communicate with Scratch and add this capability to the USB-driving Python program. The diagram above shows how it works.

The Robotic Arm is driven by commands issued from Scratch on the Raspberry Pi. These are in the form of specific *broadcast* block messages, which are received by a Python program also running on the Pi and converted to USB signals to control the arm.

The only requirement for Scratch is to enable remote sensor connections, which means it will also send out *broadcast* block messages to any process which has established a data connection via Internet Protocol (IP). All the extra clever bits, including managing the IP connection and driving the arm are within the domain of the Python program – see listing in Appendix A.

The Python program first connects to the arm via USB, then to Scratch via an IP socket. It enters a long loop, awaiting data from Scratch, decoding it and making the appropriate USB library calls to send signals to control the arm.

Scratch sends data whenever it executes a *broadcast* block. The Python program extracts the message broadcast and uses it to determine which USB signals to output.

Each USB signal sent is in effect a command to start or stop moving one of the five arm motors either forward or reverse. It is easier to have each Scratch *broadcast* block message correspond to a limited arm movement and therefore, for each one, the Python program issues a start movement USB signal, waits, and then issues the stop movement signal. The length of wait can be adjusted, from Scratch, between 0.1s (the initial value) and 1s, so longer movements can be carried out with fewer *broadcast* block messages.

The Scratch scripts are presently minimal – each command is assigned a key and generates the appropriate *broadcast* block message.

The current version of the Python program shows its history and could be considerably improved. When run, it must be invoked as `sudo python scratch_my_arm_01.py` as the USB library needs superuser status.

<to be continued!>

Appendix A

Listing of Python program enabling Scratch to drive Robotic Arm

```
# Minimalist version of USB arm control to show how simple it could be! (c) N Polwart,  2011
# scratch_my_arm.py  20131030.1530
#    copied from arm.py which sends arm commands via USB from python keyboard input


import usb.core, time
import socket
import sys

dev = usb.core.find(idVendor=0x1267, idProduct=0x0000)
if dev is None:
    print 'Cannot connect to arm'           # if device not found report an error
    sys.exit()

dev.set_configuration()

PORT = 42001
HOST = 'localhost'               # Scratch must be running on the same machine
##HOST = askstring('Scratch Connector', 'IP:')
##if not HOST:
##   sys.exit()

print("Connecting...")
scratchSock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
scratchSock.connect((HOST, PORT))
print("Connected!")

dur = 0.1               # initial movement duration is 0.1s

going = 1

while (going > 0):

    data = ''
    while len(data) < 4:
       chunk = scratchSock.recv(4 - len(data))
       if chunk == '':
           raise RuntimeError("Connection broken")
       data += chunk
    payload_len = ord(data[3:4]) + 256 * (ord(data[2:3]))
    print payload_len

    payload = ''
    while len(payload) < payload_len:
       incr = scratchSock.recv(payload_len - len(payload))
       if incr == '':
           raise RuntimeError("Connection broken")
       payload += incr

    print "<"+payload+">"

    command = payload[11:payload_len - 1]       # assume payload is <broadcast "command">

##    command = raw_input("Next command: ");
    print  command

    if command == "L1":
        print "turn light on"
        datapack=0,0,1      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

    elif command == "L0":
        print "turn light off"
        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

    elif command == "G1":
        print "close grip"
        datapack=1,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)


    elif command == "G0":
        print "open grip"
        datapack=2,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.
        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

    elif command == "S0":
        print "shoulder up"
        datapack=64,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)


    elif command == "S1":
        print "shoulder down"
        datapack=128,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

    elif command == "E1":
        print "elbow up"
        datapack=16,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)


    elif command == "E0":
        print "elbow down"
        datapack=32,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

    elif command == "W1":
        print "wrist up"
        datapack=4,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)


    elif command == "W0":
        print "wrist down"
        datapack=8,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

    elif command == "R1":
        print "rotate clockwise"
        datapack=0,2,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)


    elif command == "R0":
        print "rotate anticlockwise"
        datapack=0,1,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)

        time.sleep(dur)    # waits for 1 second whilst motors move.

        datapack=0,0,0      # change this to vary the movement
        bytesout=dev.ctrl_transfer(0x40, 6, 0x100, 0, datapack, 1000)


    elif command == "X":
        print "exit"
        going = 0

    elif command == "1":
        print "duration 1/10 sec"
        dur = 0.1

    elif command == "2":
        print "duration 1/5 sec"
        dur = 0.2

    elif command == "5":
        print "duration 1/2 sec"
        dur = 0.5

    elif command == "10":
        print "duration 1 sec"
        dur = 1.0

    else:
        print "command not recognised"

print 'done'
```