We are keeping zigbees in AT mode as of now, so that we don’t have to specify the destination address

Tested sending msg from router to coordinator. They are able to communicate to each other without Arduino.

NOTES SECTION:

Tx of Arduino to Din(Rx) of XBee & Rx of Arduino to Dout(Tx) of XBee

Disconnect Tx & Rx pins of Arduino while dumping code, else it may throw an error.

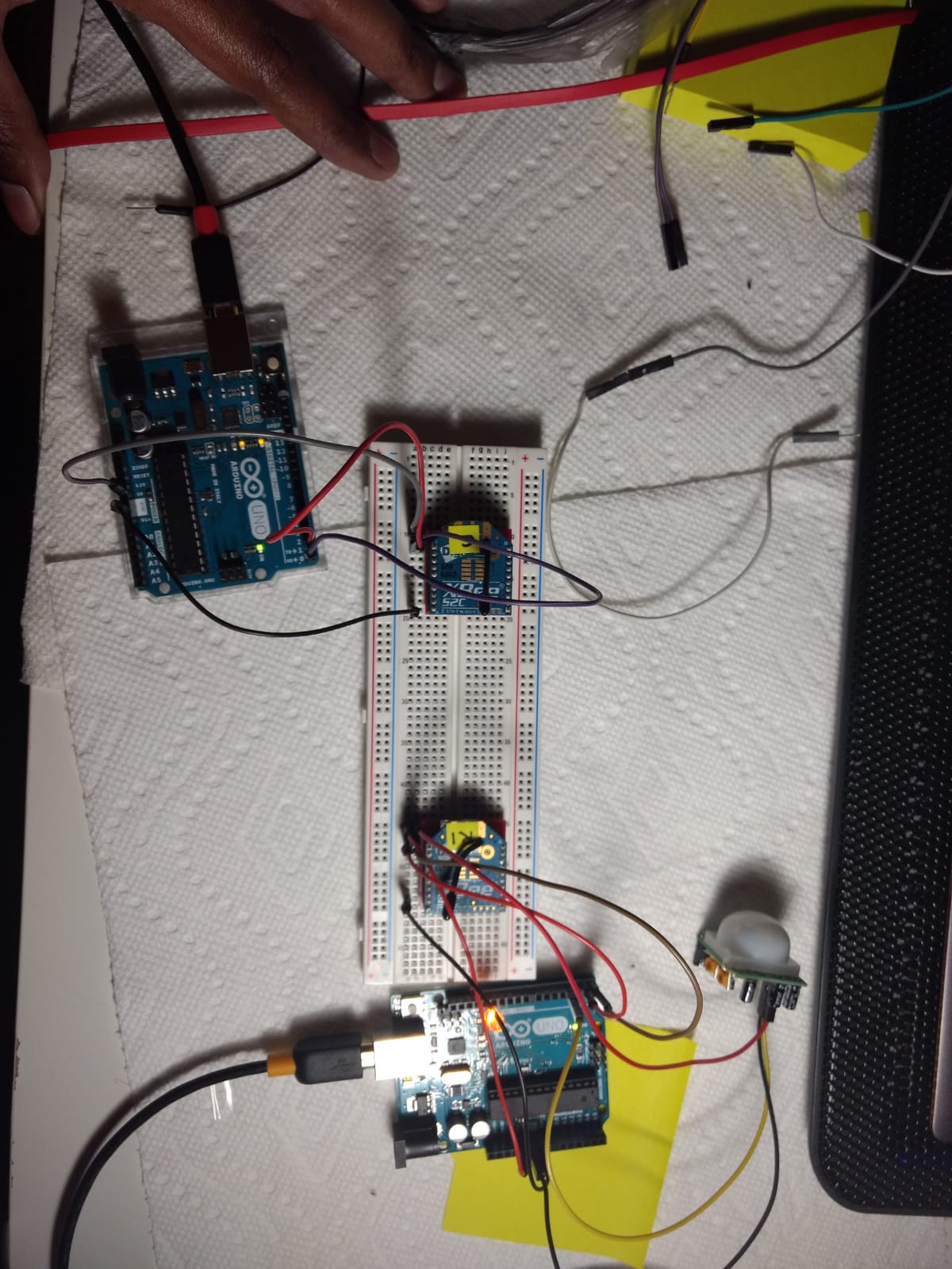
TESTING:

Test1] <https://www.youtube.com/watch?annotation_id=annotation_135090&feature=iv&src_vid=odekkumB3WQ&v=mPx3TjzvE9U>

Tried putting router in the type C shield and coordinator on breadboard with an Arduino for each. 🡪 Failed reasons unknown

Test 2]

Brought both coordinator and router on breadboard with an Arduino for each. Router code was using XBee library (temperature sensor example on youtube) 🡪 Failed



Test 3 ]

Maintained hardware setup as in test 2. Changed Zigbee modes to AT mode (In XCTU- change API Enable to Transparent Mode) with hello world code 🡪 **Success** (code files- arduino\03 & 04) Bug: Coordinator’s serial monitor was showing multiple copies of msg from zigbee

In transparent mode, the destination addresses us zero so router will send the package to the coordinator directly. In API mode, we must specify the destination address.

Test 4]

Setup same as test 2 🡪 Changed code format as shown in above youtube link   
ie. No XBee library in router code, simple serial writes. They are being caught by serial read at coordinator.(code files- arduino\02 & 03).

A screen shot of a computer

Description generated with very high confidence

Test 5]

Router is on breadboard with an Arduino (with code in file: arduino\02) and coordinator (with code in file : arduino\03)is directly connected with USB shield to the laptop. Router’s msgs of motion detection can be seen in Serial monitor of Arduino to which it is connected. And Coordinator receives these msgs which can be seen in XCTU in CONSOLE tab after we click on CLOSE serial connection.

The bug in test 5 is also solved. There is no repetition of msgs. **SUCCESS.**

**A computer sitting on a table

Description generated with very high confidenceA circuit board

Description generated with very high confidenceA circuit board

Description generated with very high confidenceA screen shot of a computer

Description generated with very high confidenceA computer

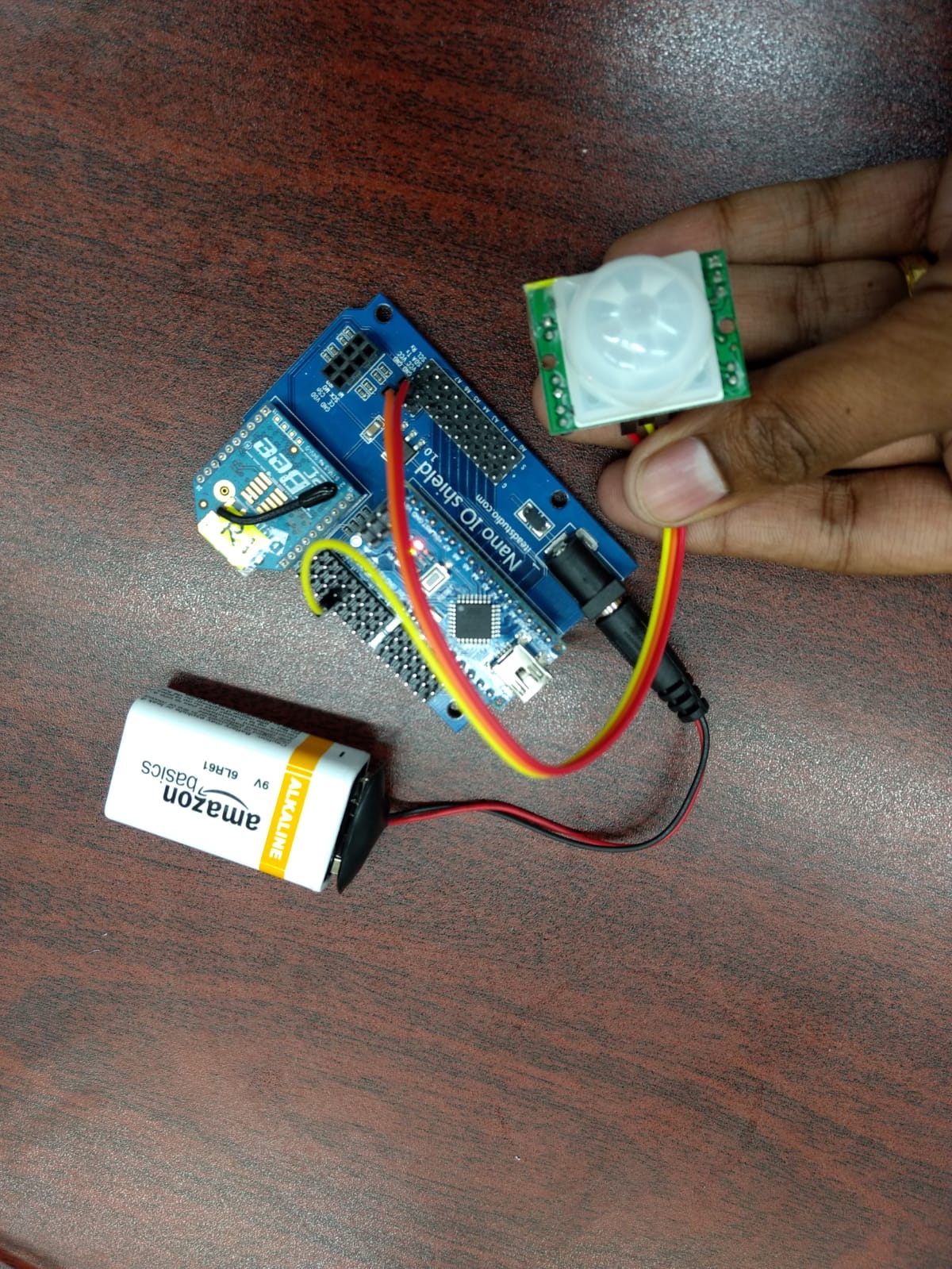
Description generated with very high confidence**

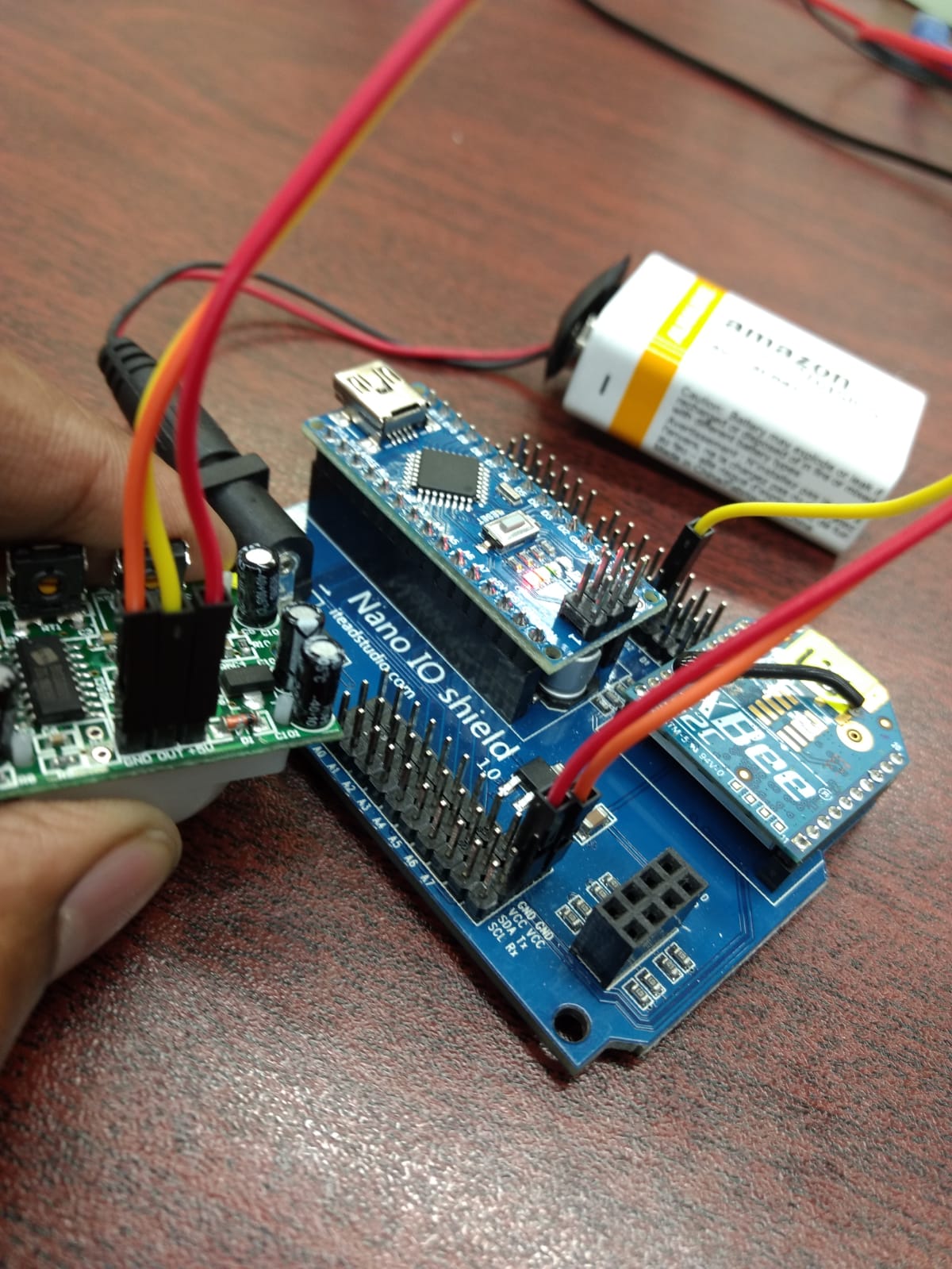
// include a flag at the Coordinator side – ack if the package is sent to the the laptop.

**PHASE 3:**

* Testing with Arduino Nano and Nano IO shield – Modular & Portable version:
* Connect sensor’s Vcc and Gnd to Vcc and Gnd of shield
* Connect PIR’s OUT to PIN3 of shield (‘S’ line)
* Connect Router Xbee Shield to the shield.
* The Arduino’s Tx is connected to Zigbee’s Rx and Arduino’s Rx is connected to Zigbee’s Tx internally – We don’t need to do it explicitly as we did it in the breadboard version.
* Pin 13 (O/P pin in our code has a red LED labelled ‘L’ on Arduino Uno board)
* Working perfect ! – **SUCCESS**

**Notes :** **Ensure that you are correct info of the board in Tools menu. When using Arduino Nano board, make sure that Tools Menu> Boards > is set to Arduino Nano and Processor is set to Atmega 328P(old bootloader). Else it will throw out of sync error.**





**TTL Camera:**

* Put ethernet shield (<https://www.adafruit.com/product/2971> ) on arudino.
* Connections as shown in the link ( <https://www.instructables.com/id/Arduino-Ethernet-Camera/> )
* Download & install all the libraries below in Arudino’s library folder (as mentioned in above link). The links to download the libraries are provided with the library names.
  + Adafruit\_VC0706.h - <https://github.com/adafruit/Adafruit-VC0706-Serial-Camera-Library>
  + SdFat.h - <https://www.arduinolibraries.info/libraries/sd-fat>
  + SdFatUtil.h - <https://upam.uni.lu/content/download/3076/15046/file/SdFatUtil.h>
  + Ethernet.h - <https://github.com/arduino-libraries/Ethernet>
  + SoftwareSerial.h - <https://github.com/PaulStoffregen/SoftwareSerial>
* MAC address – 2C:F7:F1:08:23:F9

**Repeating issue at coordinator:**

@mathtutorsnm I spent a lot of time banging my head on the table but ended up realizing that it was some sort of snowball effect.

1: The xbee reads "Hello world".

2: Arduino gets it via serial from the xbee board.

3: Arduino prints the "Hello world" recieved via serial (now here is the problem I found)

4: Since your arduino is connected to both the computer and the xbee board, via serial, it will print into both of these serial ports.

5: Computer recieves 1 "Hello world", but so does your xbee, get it? the xbee recieved the "Hello world", which it just sent out! (several milliseconds later)

6: Xbee does what it's supposed too, which is send out the signal it recieved

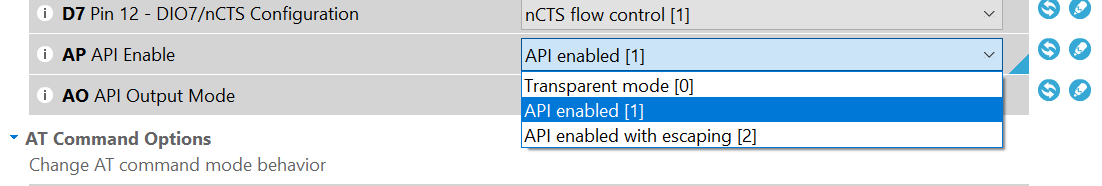
7: Arduino recieves another "Hello world", sends it out again, recieves it again, and so fourth.

8: You now have 1 "Hello world", bouncing between the shield and arduino board, and 1 more being created each time, which is sent to the computer.

9: The speed of the "Hello world" is based on your arduino, xbee, computer serial port and so fourth processing speed. So this is my hypothesis, I don't think you can read with a computer from Serial since the Serial is already in use by the Xbee shield, unless you find out a way to mess with the "USB and XBEE" button on the xbee shield. And manage not to burn out on fiddling with that shit, I know I did. What you can still do is control drones, robotic arms(which I effectively managed to do) but reading USB from an Xbee is not convenient with the shield, if you want to do that I suggest you ditch the Xbee shield on the reciever(coordinator)  end and use the configuration shown in the video.

FIXED IT :

Changed mode of Coordinator Xbee to API ENABLED[1]



Output is now stabilized. One Hello World from Router generates only 1 Hello World in the Coordinator (with arduino). The issue was caused bcoz the Coordinator XBee was in AT mode. Coordinator received it and broadcasted the same message. Since it was using Serial of Arduino it got the message back on it’s RX pin, so the message was bouncing between Arduino and Xbee back and forth.

We changed Coordinator to API mode. In API mode, it doesn’t broadcast & sends only to a particular destination address. So the loop breaks.

Another solution could be to use Arduino Mega at the Coordinator. It has multiple Serial pins. We can connect Xbee to one pair of Tx Rx pins and perform Serial.write() on another set of TxRx pins.

**Working model :**

* Refer video at <https://drive.google.com/open?id=1_I4qdFdRLx8LytTAbI8lZ5kM3tRpaNQ1> for a demo of the working.
* We have 3 parts in our network.
  + Motion Sensing & Detection (Router 1)
  + Central Actuator (Coordinator)
  + Camera (Router 2)
* Both router xbees are configured with config in Router\_AT\_profile.xpro
* The co-ordinator is configured in API mode.
  + In AT mode, an XBee would broadcast any message it receives to its destination address (which is set to coordinator by default). Once the destination address is set, there is no provision for that XBee to send messages to any other destinations.
  + We do not configure the coordinator in AT mode as we don’t want it to broadcast every message it gets.
  + Also, API mode is suitable if the XBee is going to communicate with multiple XBees. This is because in every API packet, destination address has to be explicitly mentioned. So we can specify multiple addresses in our Arduino code & send different messages to their corresponding destination Xbees.
* Let us look into the 3 parts of our network as mentioned above:

**Motion Sensing & Detection (Router 1):**

**Components:** PIR sensor, Arduino Nano, XBee module, Arduino Nano breakout shield.

**Architecture:**

* Fix the Arduino and Xbee in Arduino Nano breakout shield.
* The Out of PIR sensor is connected to Pin 2 of Arduino.
* Arduino Nano breakout board takes care of the connections between Arduino and XBee, so we don’t have to do anything explicitly.
* Upload the code in <filename> to the arduino.(09)
* While uploading the code to Arduino, make sure you disconnect it from the nano board else it will throw errors.
* Ensure that you are correct info of the board in Tools menu. When using Arduino Nano board, make sure that Tools Menu> Boards > is set to Arduino Nano and Processor is set to Atmega 328P(old bootloader). Else it will throw out of sync error.

**Working:**

* When PIR sensor detects motion, Arduino writes a message into its Serial Port.
* To keep our message short, we have used “M” as our message payload. M here is our codeword for Motion Detected. The coordinator is programmed to look this message to trigger the camera.
* Since this router configured in AT mode, we don’t have to explicitly write every byte of the packet in Arduino’s Serial port. We are only are required to write the payload in Serial, the Xbee encodes other bytes from its internal profile settings.
* The Router XBee connected to the Arduino’s serial port, sends this message/packet wirelessly to the Coordinator. The size of this packet is 19 bytes.

Example packet: 7E 00 0F 90 00 13 A2 00 41 08 09 D7 8E 9B 01 4D 0D 0A 03

**Actuator (Coordinator):**

**Components:** Arduino Nano, XBee module, Arduino Nano breakout shield.

**Architecture:**

* Fix the Arduino and Xbee in Arduino Nano breakout shield. (refer Motion Sensing & Detection part for details about breakout shield)
* Upload the code in <filename> to the arduino.(08)

**Working:**

* The Coordinator Xbee will receive message packet from router Xbee of sensing module and Arduino will read it through it’s serial port.
* If this message has our code (“M”) for detection of motion, a message will be sent to the camera to trigger.
* The Arduino will check & analyze this message to get the information out of it.
* It first checks for the Start delimiter of message i.e. 0x7E which is standard for all Zigbee packets.
* It reads the rest 18 packets and stores them in a byte array RFin\_bytes.
* It then looks if the payload of this packet has “M” in it. If so, it calls the activateCamera() function to trigger the camera.
* In activateCamera() function, we create an API packet for ‘Transmit Request’. Transmit Request is generally used to send a message to a particular destination.
* Refer the packet structure of API Transmit Request to understand the construction.
* Arduino will construct this packet by writing it byte by byte in it’s Serial port.
* The XBee will transmit this message from its serial port.
* Every byte encoding in the code has comments with it indicating the part of the packet that byte corresponds to.
* Here we encode the letter “A” into the payload of the message packet denoting ‘Activate camera’.
* Notes: The 3rd byte written is the length of the message. The length of the message is calculated by counting everything in the packet after this 3rd byte excluding the checksum. In the end checksum is calculated, it a sum of every non-zero byte after the length byte of the packet.
* Check sum marks the last byte of the message packet sent.

**Camera (Router 2):**

**Components:** TTL Serial Camera, Arduino Nano, XBee module, Arduino Nano breakout shield, Sd Card, Sd Card reader, Breadboard, Resistors

**Architecture:**

* Fix the Arduino and Xbee in Arduino Nano breakout shield. (refer Motion Sensing & Detection part for details about breakout shield)
* For pin connections of Arduino, Camera and SD card reader, please refer: <https://learn.adafruit.com/ttl-serial-camera/arduino-usage>
* Upload the code in <filename> to the arduino. (10)

**Working:**

* The Xbee receives the message packet from the coordinator.
* Since this Xbee is configured in AT mode, it directly extracts the payload of the message packet and writes it in Serial port of the Arduino.
* The arudino checks if the payload is equal to “A” (the code for activating the camera), if so it clicks a picture from the camera and stores it in the SD card.
* We can change the resolution of the image in the code.
* It takes around xx secs to store an image of approx. 50kb.
* Once the pic is captured, it displays a message “Picture captured & stored in SD card”,

**Messages:**

M – **M**otion detected – Sensor to Coordinator

A – **A**ctivate camera – Coordinator to Camera

D – Capturing image **D**one (Positive ack) – Camera to Coordinator

F – Capturing image **F**ailed (Negative ack) – Camera to Coordinator

R – **R**esume sensing – Coordinator to Sensor

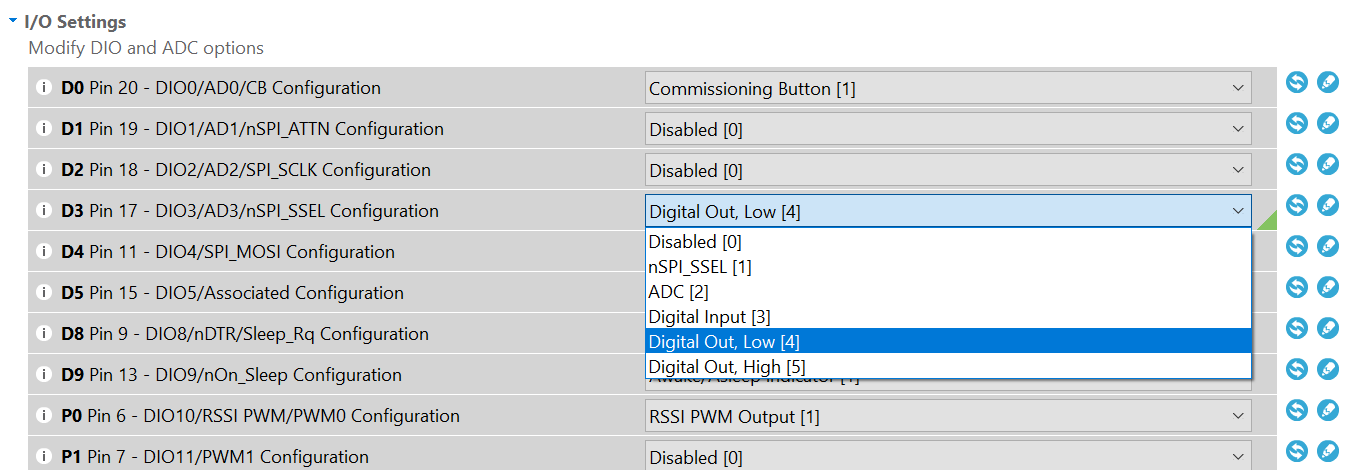
**Imp Note : Make sure that Arduino is not writing these codewords in any other Serial.print() or Serial.write() command. It would break the logic as the receiving Xbee would get a false alarm that this message is generated by the Arduino. One trick is to use any uppercase letters only for codewords & lowercase for any other messages that you want Arduino to print in its Serial port.**

**Light turn on:**

Function to start and stop light.

Write the bytes all in hex. We use D3.

While configuring the router xbee, take all the settings like other routers in addition to one setting. Set D3 to Digital Out Low.



In the coordinator, write the frames in Hex . D3 in hex is 0x44 0x33.  
High is 0x5 & Low is 0x4. Connect the xbee on breadboard & supply 3.3V to it. Take out from D3 & connect it to LEDs.

**Pi part**

Config VCN viewer for connection:

On your Raspberry Pi, run the following commands to make sure you have the latest version of VNC Connect:

sudo apt-get update

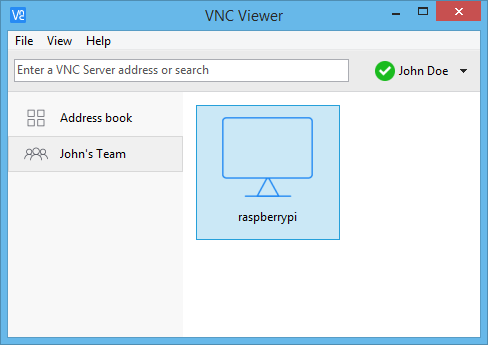
sudo apt-get install realvnc-vnc-server realvnc-vnc-viewer

Enabling VNC Server graphically:

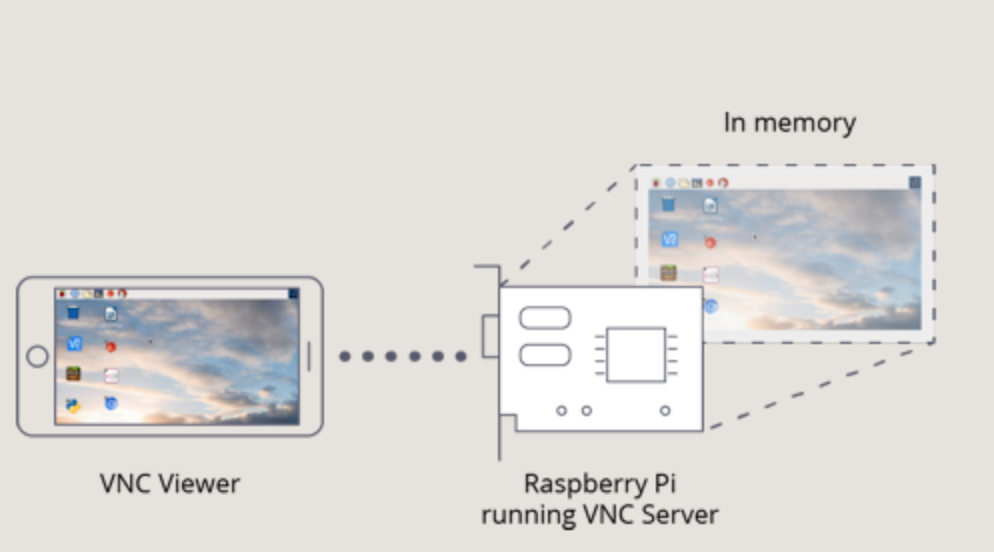
1. On your Raspberry Pi, boot into the graphical desktop.
2. Select Menu > Preferences > Raspberry Pi Configuration > Interfaces.
3. Ensure VNC is Enabled.

Or, you can configure it directly in the panel;

sudo raspi-config



Now you can use your laptop, even phone to access the pi.



<https://www.raspberrypi.org/documentation/remote-access/vnc/>

Website:

Build a Flask web site on the raspberry Pi:

[**https://projects.raspberrypi.org/en/projects/python-web-server-with-flask/2**](https://projects.raspberrypi.org/en/projects/python-web-server-with-flask/2)

To install the Flask into Pi, first you should try the command below:

sudo apt-get install flask

then start your code like this in a python file.

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return 'Hello world'

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, host='0.0.0.0')

run it use command like: python youcode.py

then access URL: <http://127.0.0.1:5000> to visit the webpage.

**DataBase:**

For database we used sqlite3 to store the data.

$ **sqlite3 ex1**

SQLite version 3.8.5

Enter ".help" for usage hints.

sqlite> **create table tbl1(one varchar(10), two smallint);**

sqlite> **insert into tbl1 values('hello!',10);**

sqlite> **insert into tbl1 values('goodbye', 20);**

sqlite> **select \* from tbl1;**

hello!|10

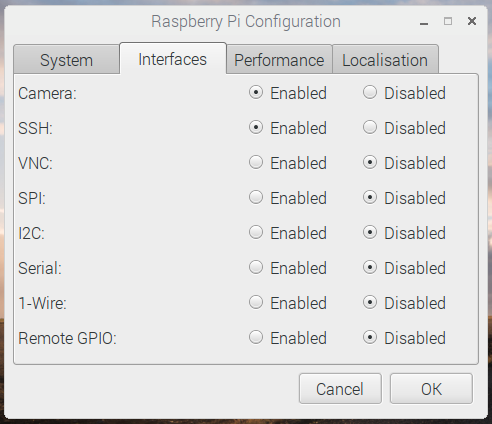
goodbye|20

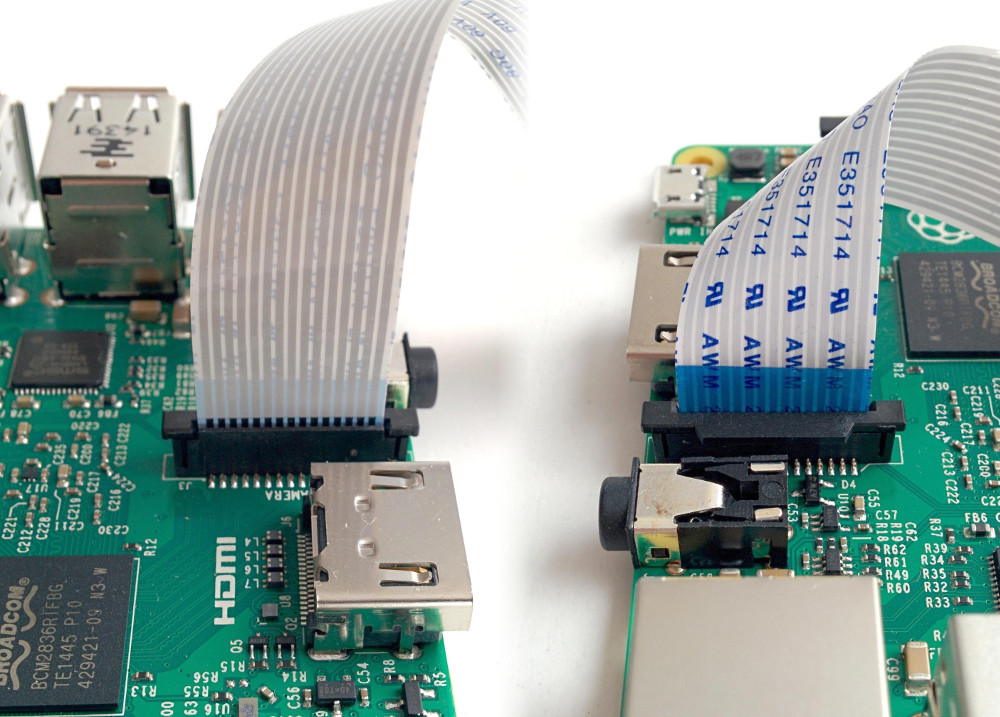
sqlite>

Make sure you type a semicolon at the end of each SQL command! The sqlite3 program looks for a semicolon to know when your SQL command is complete. If you omit the semicolon, sqlite3 will give you a continuation prompt and wait for you to enter more text to be added to the current SQL command. In our program, we build a User.db and inserted user name and password like (‘admin’, ‘password’).

**Camera:**

First of all, with the Pi switched off, you’ll need to connect the Camera Module to the Raspberry Pi’s camera port, then start up the Pi and ensure the software is enabled. The connection part can be pulled out and push it back to make sure cable are fixed firmly. Then enable the Camera in raspi-config.





How to use it:

Open a new file and save it as camera.py. It’s important that you do not save it as picamera.py, because it will have the same name as the package.

Enter the following code:

from picamera import PiCamera

from time import sleep

camera = PiCamera()

camera.start\_preview()

sleep(10)

camera.stop\_preview()

Save with Ctrl + S and run with F5. The camera preview should be shown for 10 seconds, and then close. Move the camera around to preview what the camera sees. The live camera preview should fill the screen.

There is also some parameters that you can used to configure the camera.

camera.saturation = 80

camera.brightness = 50

camera.shutter\_speed = 6000000

camera.iso = 800

camera.sharpness = 0

camera.framrate = 32

camera.hflip = Ture

camera.vflip = False

camera.rotation = 0

camera.resolution = (280,160)

a\_gain = camera.analog\_gain

d\_gain = camera.digital\_gain

camera.led = False

The most common use for the Camera Module is taking still pictures. Amend your code to reduce the sleep and add a camera.capture() line:

camera.start\_preview()

sleep(5)

camera.capture('/home/pi/Desktop/image.jpg')

camera.stop\_preview()

It’s important to sleep for at least 2 seconds before capturing, to give the sensor time to set its light levels. Run the code and you’ll see the camera preview open for 5 seconds before capturing a still picture. You’ll see the preview adjust to a different resolution momentarily as the picture is taken.

To recording video, amend your code to replace capture() with start\_recording() and stop\_recording():

camera.start\_preview()

camera.start\_recording('/home/pi/video.h264')

sleep(10)

camera.stop\_recording()

camera.stop\_preview()

Run the code; it will record 10 seconds of video and then close the preview.

To play the video, you’ll need to open a terminal window by clicking the black monitor icon in the taskbar, Open terminal:

pen terminal

Type the following command and press **Enter** to play the video:

omxplayer video.h264

In our project, we used the frame to display live stream on the webpages, and capture 5 images every time when it detected a motion there. Flask main frame code are shown below:

**from** flask **import** Flask, render\_template, Response  
  
**from** camera\_pi **import** Camera  
  
app **=** Flask**(**\_\_name\_\_**)  
  
  
@**app.route**('/')  
def index():  
 """Video streaming home page."""  
 return** render\_template**('index.html')  
  
  
def gen(**camera**):  
 """Video streaming generator function."""  
 while True:** frame **=** camera.get\_frame**()  
 yield (b'--frame\r\n'  
 b'Content-Type: image/jpeg\r\n\r\n' +** frame **+ b'\r\n')  
  
  
@**app.route**('/video\_feed')  
def video\_feed():  
 """Video streaming route. Put this in the src attribute of an img tag."""  
 return** Response**(**gen**(**Camera**())**,  
 mimetype**='multipart/x-mixed-replace; boundary=frame')  
  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** app.run**(**host**='0.0.0.0'**, debug**=True**, threaded**=True)**

**References:**

**Zigbee router-coordinator-pir:**

<https://www.youtube.com/watch?v=uBkQUph9EKM>

<https://www.youtube.com/watch?annotation_id=annotation_135090&feature=iv&src_vid=odekkumB3WQ&v=mPx3TjzvE9U>

<https://www.youtube.com/watch?v=wtal7SWZek0>

<https://arduino.stackexchange.com/questions/16603/why-i-cannot-connect-directly-arduino-uno-and-xbee-s2>

<https://www.bananarobotics.com/shop/ITEAD-Arduino-Nano-IO-Shield>

<https://forum.arduino.cc/index.php?topic=46652.0>

<http://forum.arduino.cc/index.php?topic=44307.0>

sd card tutorial : <https://www.youtube.com/watch?v=sS_oW81NweI>

<https://www.digi.com/resources/documentation/Digidocs/90001942-13/concepts/c_xbee_comparing_at_api_modes.htm?TocPath=How%20XBee%20devices%20work%7CSerial%20communication%7C_____2>

**TTL camera ethernet shield arudino:**