Internet of Things: A study of sensor networks, architectures & design of protocol layers

Fall 2018

University of Texas at Dallas

Project Report:

Automated Home Security System

Submitted by:

Aashaar Panchalan (adp170630)

Manva Pradhan (mxp174430)

Pratik Kamath (pxk170010)

Yangtao Hua (yxh172530)

Under the guidance of Prof. Neeraj Gupta

List of Contents:

1. Introduction:

This project report initially describes the problem statement, motivation & objective for implementing our project Automated Home Security System. It then describes the technology used i.e. Zigbee followed by practical implementation steps and working of every module.

1. a) Problem Statement:

Nowadays, the security level is very important, always emphasized and enhanced system. Different ways of security system have been enhanced such as use many security officers, the use of alarms, monitoring system, through the production of electronic hardware and software and much more. All this improvement depends on their usage. One of the most important safety systems and required for all social group is home security. Houses always need to be monitored such as from thefts & burglaries. When residents are out of home, they still desire to keep an eye on their homes to ensure its safety. So, home surveillance system must be upgraded to be more effective to keep up with the increasing crime rate. Various methods can be done to improve home security monitoring including the usage of security officers. However, this method is not suitable for all levels, wasteful and less reliability. All these improvements need to work more effectively, giving advantages to the user and can monitor without any errors that may hinder the security process. At present, a lot of study on smart home systems has been done and it covers all aspects. For example, smart home systems study in terms of multimedia, security monitoring, lighting, temperature control and others. In a smart home system, manual methods are no longer used and replaced by an automated system that helped users to monitor the condition of the house, thus facilitating and speeding up daily works. Automatic system can prevent the effects of human error and saving electricity.

1. b) Objectives of the project:

The purpose of Automated Home security system is to detect presence of an intruder using a sensor and then wirelessly start a camera and light. The camera is required to take some pictures and start a video stream which can be made available to the resident on their smart phones. The objective is to implement a wireless network where are the components are connected to each other using Zigbee protocol.

2. Zigbee:

Zigbee is a low power wireless technological standard created by Zigbee Alliance for WPANs based on IEEE 802.15.4 created for Control & Sensor Networks. It supports data rates up to 250kbps. It supports Star and Mesh topologies.

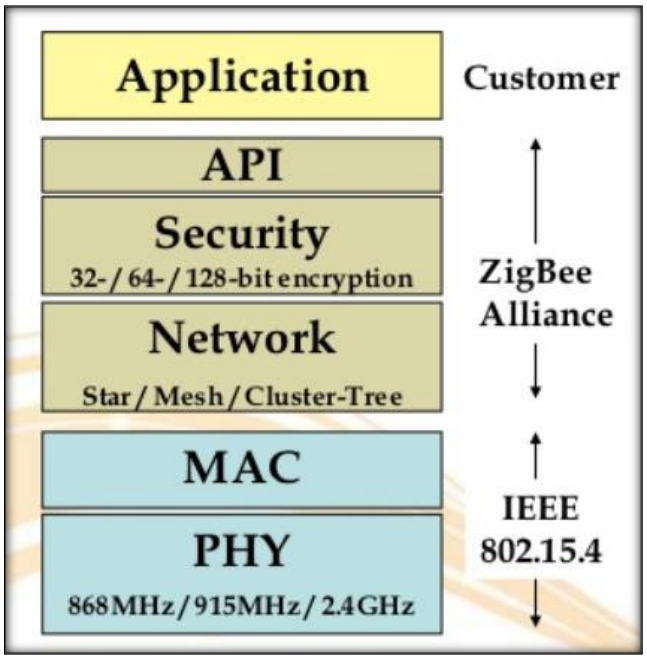
Zigbee consists of 2 types of devices –

- **Fully Functional Device (FFD):** It can communicate with other FFDs & RFDs and can function as a Coordinator, Router or End device.
- **Reduced Functional Devices (RFD):** It can communicate with only FFDs and can function only as an End device.

A Zigbee network consists of 3 components –

- **Coordinator:**
  - Initiates the network
  - Stores information about the network
  - Stores the routing algorithm
  - All devices communicate with it
  - Acts a bridge to other networks
  - WPAN must have exactly 1 FFD acting as Coordinator
- **Router:**
  - Optional component
  - Should be an FFD
  - Capable of extending network coverage area
  - Manages local address allocation & deallocation
- **End Device:**
  - Can either be an FFD or RFD according to the application
  - Optimized for low power applications
  - Cheapest component of the network
  - Communicates only with Coordinator or Router

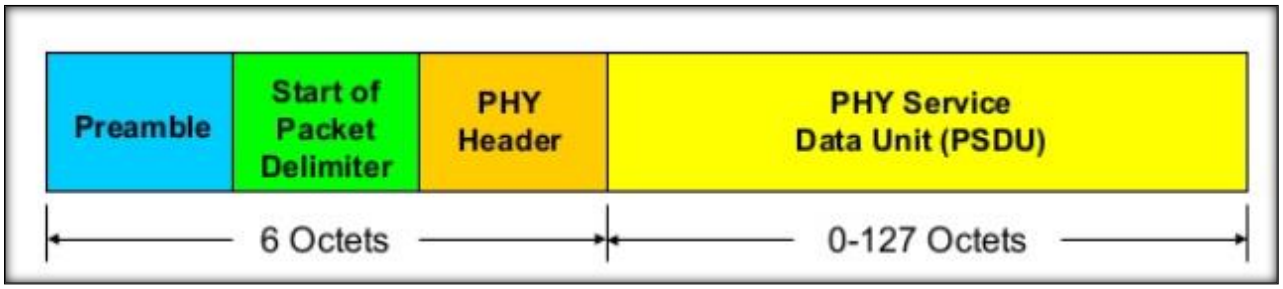
2.a) Zigbee Stack and layers:



Physical Layer: The physical layer performs modulation on outgoing signals and demodulation on incoming signals. It transmits information and receives information from a source. The following table shows the physical layer frequency band, data rate, and channel numbers.

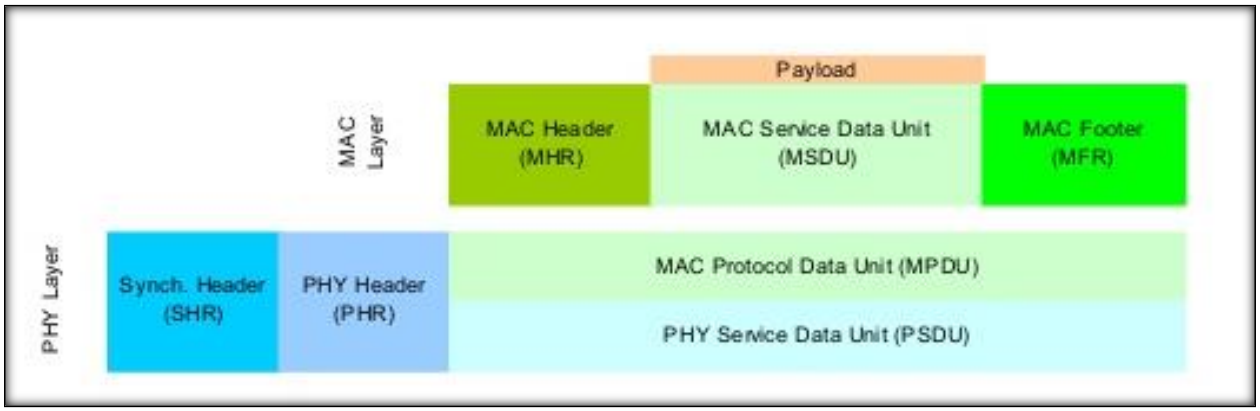
Frequency Band	Country	Data Rate	Channel Numbers
868.3MHz	European countries	20Kbps	0
902–928 MHz	United States	40Kbps	1–10
2.405GHz	Worldwide	250Kbps	11–26

The structure of a packet is as follows:



- Octets = Bytes
- Preamble – 32 bits
- Start of packet – 8 bits
- PHY header – 8 bits – stores PSDU length
- PSDU – 0 to 1016 bits – Data field

MAC Layer: It is responsible for performing functions like channel acquisition, frame security, error correction, specifying traffic mode, etc. The structure of a MAC layer packet is as follows:



Zigbee supports 4 types of frame format:

- Data frame – for normal data transmission
- Beacon frame – for handshakes
- Acknowledgment frame – for sending ACKs
- MAC command frame – for controlling digital pins of hardware XBee module.

2.b) Traffic Types in Zigbee:

- **Periodic:**
  - Sensor wakes up, checks for data & then goes to sleep
  - Application dictates the data rate
- **Intermittent:**
  - Application or some external stimulus determines the data rate (energy saver)
- **Repetitive:**
  - Fixed data rate
  - Guaranteed Time Slots (GTS) are used to operate devices in fixed duration

Network Layer: The network layer is located between the MAC layer and application support sublayer. It provides the following functions:

- Starting a network
- Managing end devices joining or leaving a network
- Route discovery
- Neighbor discovery

Application Support Layer: It acts an interface between Network & Application Layers. Its tasks include:

- Accepting data units from application layer
- Adding the header information of the frame to it
- Transferring the resulting frame to Network layer.

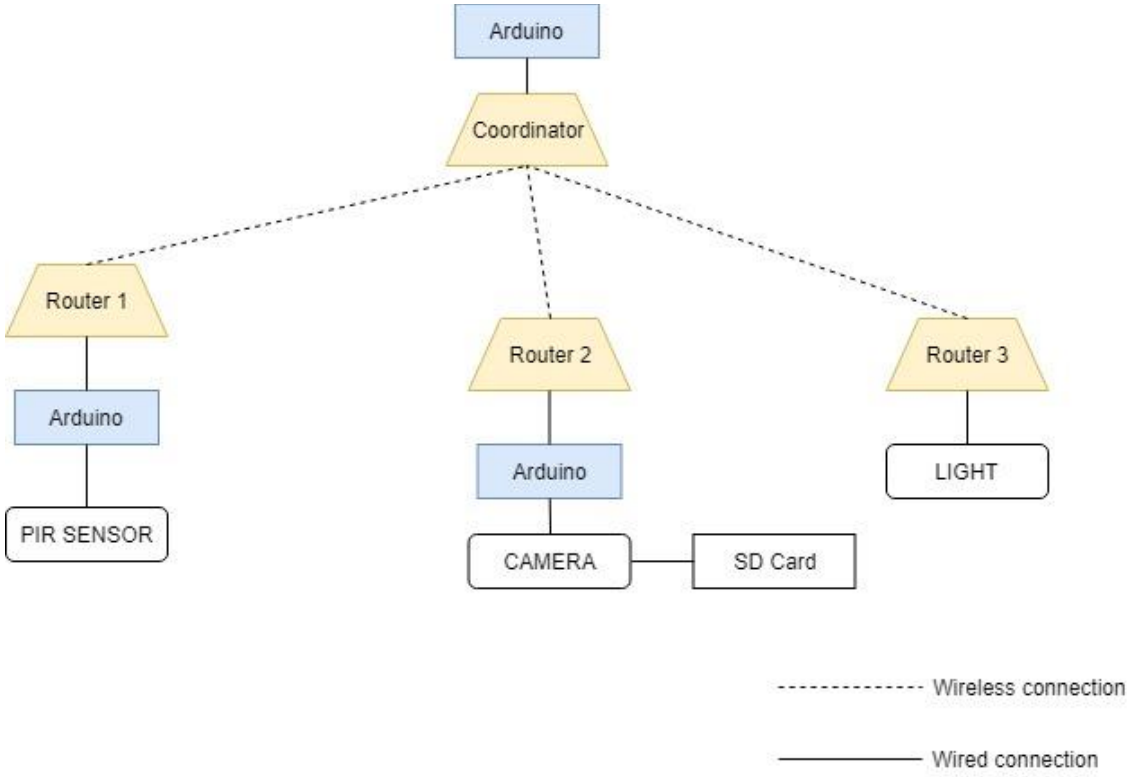
Application Layer: It is specified by the vendor. It consists of application objects which hold user applications & Zigbee device objects (ZDOs). Every device has descriptor called 'ZigBee descriptor'. It contains following information:

- Frequency band
- Power description
- Application flags
- Application versions
- Serial Number
- Manufacturer, etc.

Zigbee Device Object (ZDO): It defines the role of the device in the network – Coordinator, Router or End Device. It also initializes API and Network layers and offers services like device/service discovery.

3. Prototype:

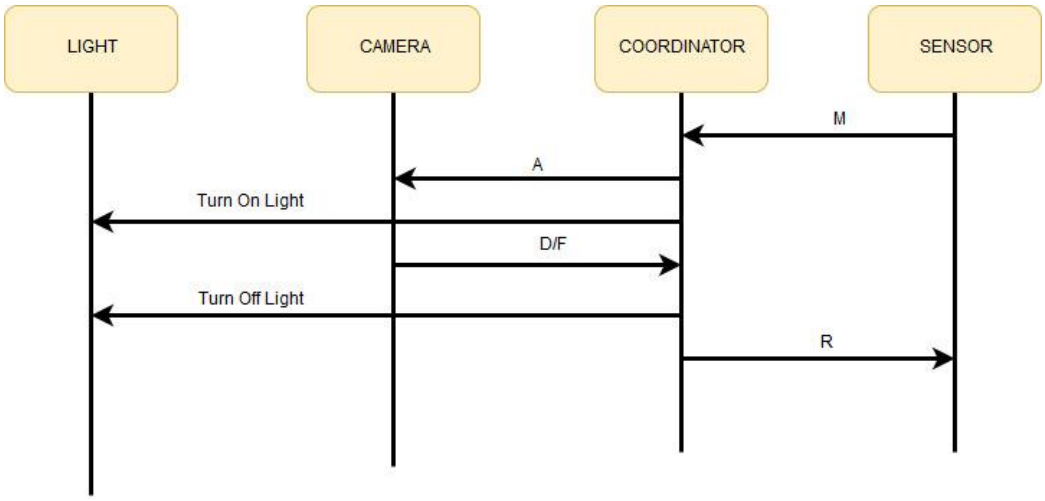
3.a) Architecture Diagram: *(Team)*



3.b) Components used:

- Arduino nano
- Xbee pro S2
- Arduino nano i/o breakout shield
- PIR motion sensor
- TTL serial camera
- Resistors
- SD card
- SD card reader
- Breadboard

3.c) Sequence Diagram: *(Aashaar)*



Messages: *(Aashaar)*

- M – Motion detected – Sensor to Coordinator
- A – Activate camera – Coordinator to Camera
- D – Capturing image Done (Positive ack) – Camera to Coordinator
- F – Capturing image Failed (Negative ack) – Camera to Coordinator
- R – Resume sensing – Coordinator to Sensor

All the router xbees are configured with config in AT mode & the coordinator 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 must be explicitly mentioned. So, we can specify multiple addresses in our Arduino code & send different messages to their corresponding destination XBees.

3.d) Configuring XBee: *(Aashaar)*

The following section describes how to configure Coordinator & Router XBees.

Coordinator:

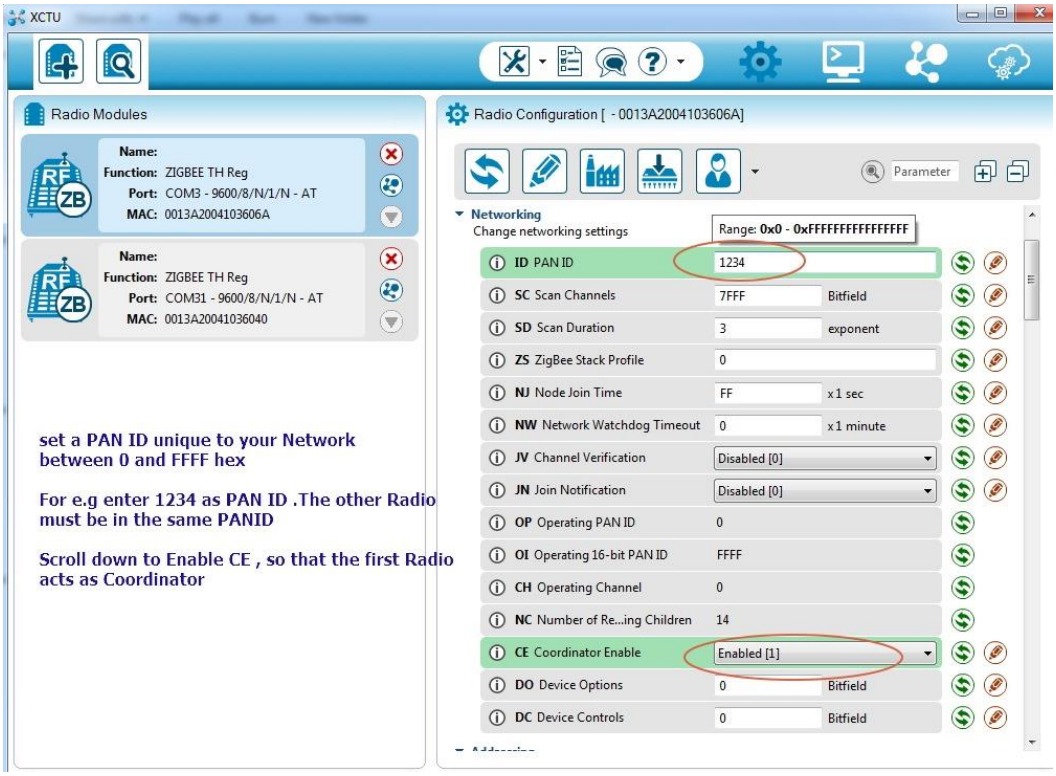
- Connect the XBee to a computer and open XCTU software.
- Upgrade the firmware of the XBee if required.
- Specify a unique Pan ID, like 1002 for this network.

- Set CE Coordinator Enable to 'Enabled [1]'
- In NI Node Identifier, enter a unique name for this XBee ex. Coordinator.
- Set AP API Enable to 'API enabled [1]' – this sets the router in API mode.
- Click on the write changes button on top.
- These settings are saved in a file Coordinator_API_profile.xpro (\\Prototype code\zigbee)

Router:

- Connect the XBee to a computer and open XCTU software.
- Upgrade the firmware of the XBee if required.
- Enter the same PAN ID as specified in the coordinator.
- Set JV Channel verification to 'Enabled [1]'
- In NI Node Identifier, enter a unique name for this XBee ex. Router1 or Router_PIR
- Set AP API Enable to 'Transparent mode [0]' – this sets the router in AT mode.
- Click on the write changes button on top.
- These settings are saved in a file Router_AT_profile.xpro (\\Prototype code\zigbee)

Here's sample screenshot of XCTU screen:



The network architecture consists of 4 parts:

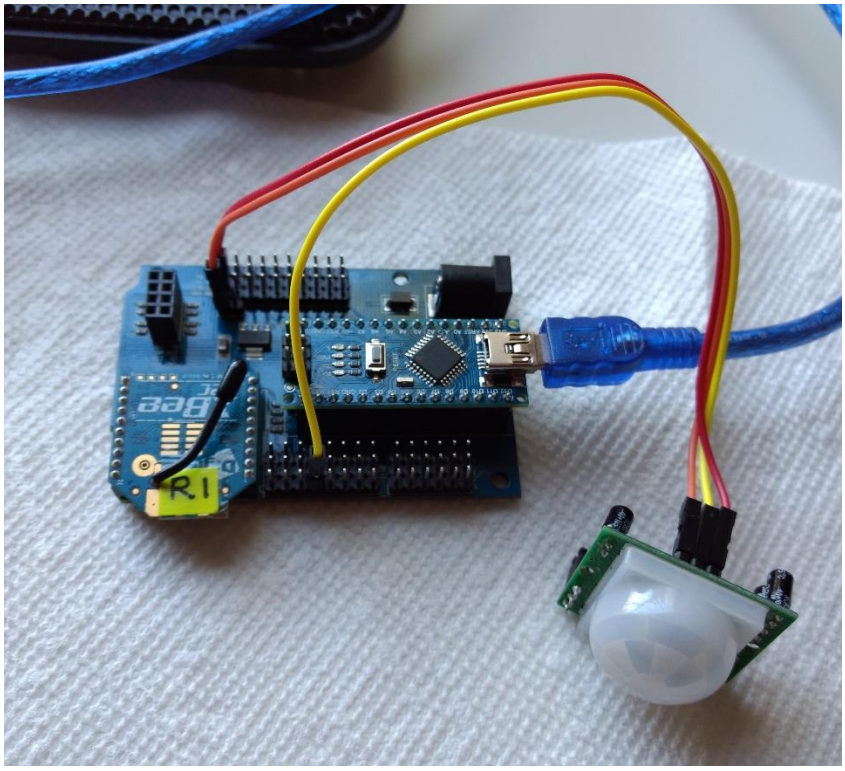
- Motion Sensing & Detection (Router 1)
- Coordinator
- Camera (Router 2)
- Lights (Router 3)

The following section explores all the parts of our network mentioned above:

3.e) Motion Sensing & Detection (Router 1):

Function: Detects motion & sends a wireless message to the coordinator.

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



Architecture: *(Aashaar, Pratik, Manva)*

- Fix the Arduino and Xbee in Arduino Nano breakout shield.
- Connect the Vcc and Gnd of the PIR sensor to arduino's +5V and Gnd respt.
- 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 PIR_Router.ino (\\Prototype code\arudino\PIR_Router1) in Arduino.
- Upload the settings in Router_AT_profile.xpro (\\Prototype code\zigbee) in XBee to configure it as router for the network. The Pan ID is set to 1002 for the network.

- 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.
- Power up the Arduino using usb cable or an external 5V supply.

Working with Arduino: (Aashaar, Pratik)

- Connected the PIR sensor to Arduino using the following: GND to GND, 5V to 5V, output to a digital pin of the Arduino.
- Implemented logic in Arduino to sense motion using PIR sensor by detecting a HIGH signal from the PIR sensor and to sense the end of motion by detecting a LOW signal from the PIR sensor. Also reflected the corresponding output on the Arduino serial monitor.
- Ensured proper operation of PIR sensor since it is a very critical input to our system. This included calibration, adjusting sensitivity, precision, accuracy & duty cycle.

Working with Xbee: *(Aashaar)*

- 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.

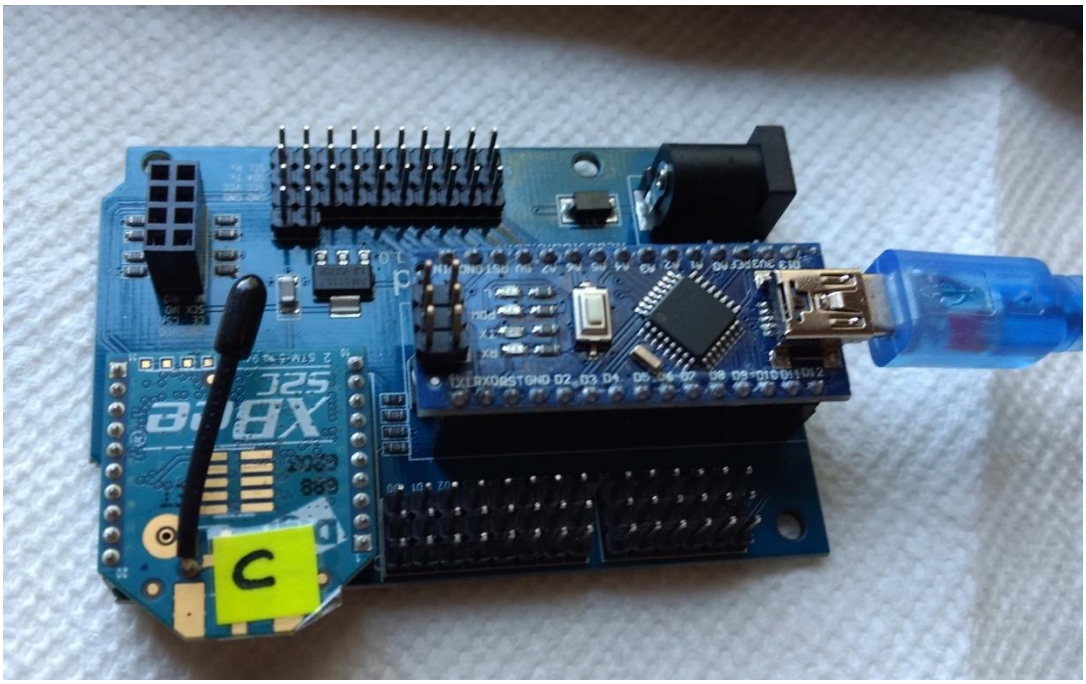
3.f) Coordinator:

Function: Receives message from sensor & sends wireless messages to camera and light to turn them on/off.

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

Architecture: *(Aashaar)*

- Fix the Arduino and Xbee in Arduino Nano breakout shield. (refer Motion Sensing & Detection part for details about breakout shield)
- Upload the code in Coordinator.ino (\\Prototype code\arudino\ Coordinator) in Arduino.
- Upload the settings in Coordinator_API_profile.xpro (\\Prototype code\zigbee) in XBee to configure it the coordinator for the network. The Pan ID is set to 1002.
- Power up the Arduino using usb cable or an external 5V supply.



Working: *(Aashaar)*

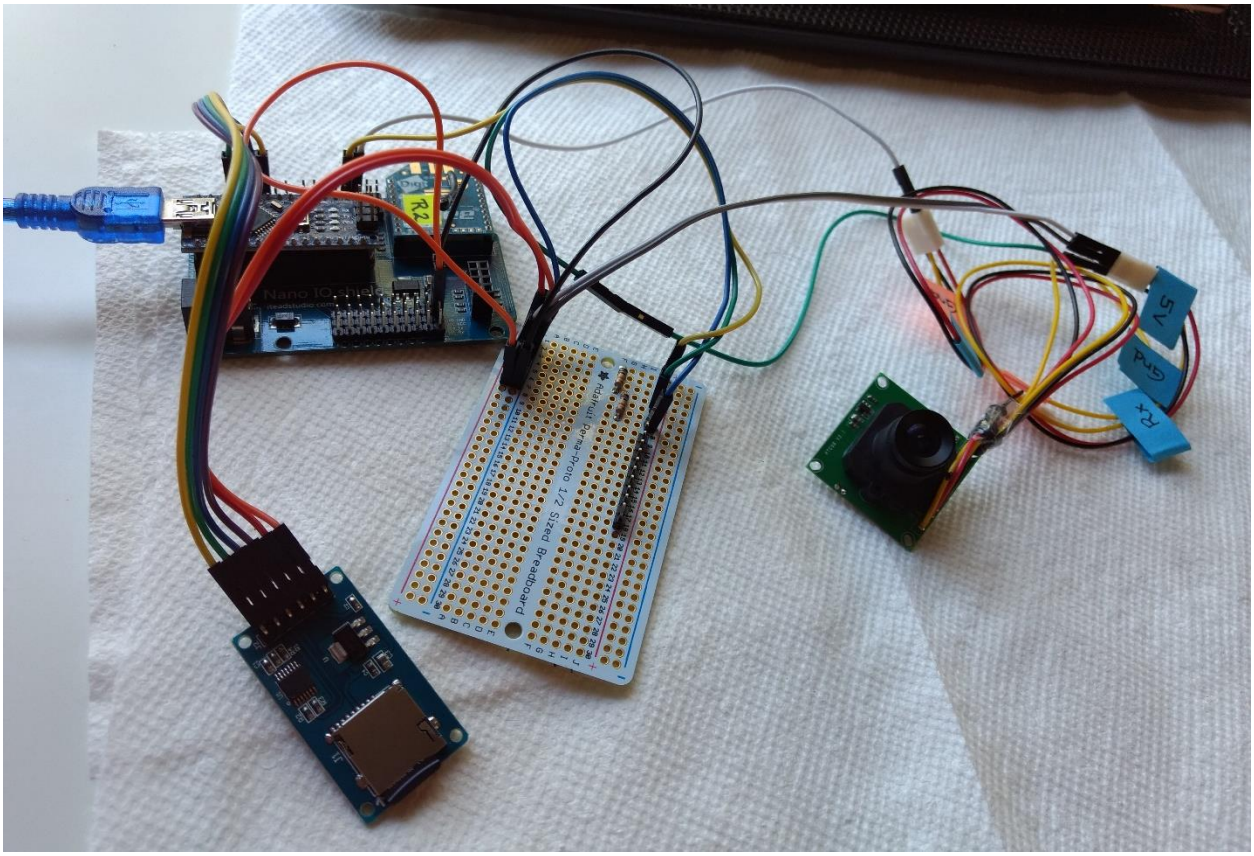
- The Coordinator Xbee will receive message packet from router Xbee of sensing module and Arduino will read it through its serial port.
- If this message has our code (“M”) for detection of motion, a message will be sent to the camera to trigger.
- The Arduino will check & analyze this message to get 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 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 its 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.
- Coordinator calls one more function right after activateCamera() i.e. activateLight()
- activateLight()function wirelessly turns on the LEDs. It is a remote AT command to control Router 3 Xbee from Coordinator.

- We control D3 of Router 3 XBee, to specify pin number as D3 in the packet structure we use the hex converted values 0x44 0x33 while writing bytes in Arduino's serial port.
- To set pin D3 of Xbee to HIGH, write 0x5 as parameter value, to reset it to LOW write 0x4 as parameter value in the packet.

3.g) Camera (Router 2):

Function: Receives message from the coordinator, clicks a picture & stores in the SD card.

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



Architecture: *(Aashaar, Pratik, Manva, Yangtao)*

- 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
- Take 5V and Gnd from Arduino and connect them breadboard to power multiple devices.
- Supply Vcc and Gnd for camera from the breadboard.
- Connect Tx of camera to pin 2 of Arduino. On the breakout board, this pin is denoted by D2.

- Create a voltage divider circuit on the breadboard using the two 10k ohm resistors.
- Connect one end of the voltage divider to pin 3 of Arduino and the other end to Gnd.
- Take a connection from the middle of this voltage divider circuit and connect it to the Rx of the camera. This is required as the camera's high is 3.3V and arduino's high is 5V. So, the voltage divider brings arduino's high of 5V to 2.5V which is safe for the camera.
- Supply Vcc and Gnd for the SD card reader from the breadboard.
- Connect the following pins in SD card reader and camera respectively:

SD Card reader	Arduino
CS	10
DI	11
DO	12
CLK	13

- Insert an SD card inside the card reader.
- If the camera or sdcard reader is not configured properly, it will show specific error in the serial monitor of the Arduino.
- Upload the code in Camera_Router2.ino (\\Prototype code\arudino\ Camera_Router2) in Arduino.
- Upload the settings in Router_AT_profile.xpro (\\Prototype code\zigbee) in XBee to configure it as router for the network. The Pan ID is set to 1002 for the network.
- Power up the Arduino using usb cable or an external 5V supply.

Working with Arduino: *(Aashaar, Pratik)*

- Developed code in Arduino to capture the images using the TTL Serial camera on a timely basis (by setting the log interval) and store the images in the SD card which is connected to the Arduino via SD card reader.
- Developed code from scratch for serial transfer for image data from the SD card using Arduino to PC and decode it on the PC to render the image, however the Arduino serial transfer was very slow, and we shifted to camera which was directly connected to Raspberry Pi.

Working with Xbee: *(Aashaar)*

- 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 arduino 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.
- Once the pic is captured, it displays a message "Picture captured & stored in SD card".

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.

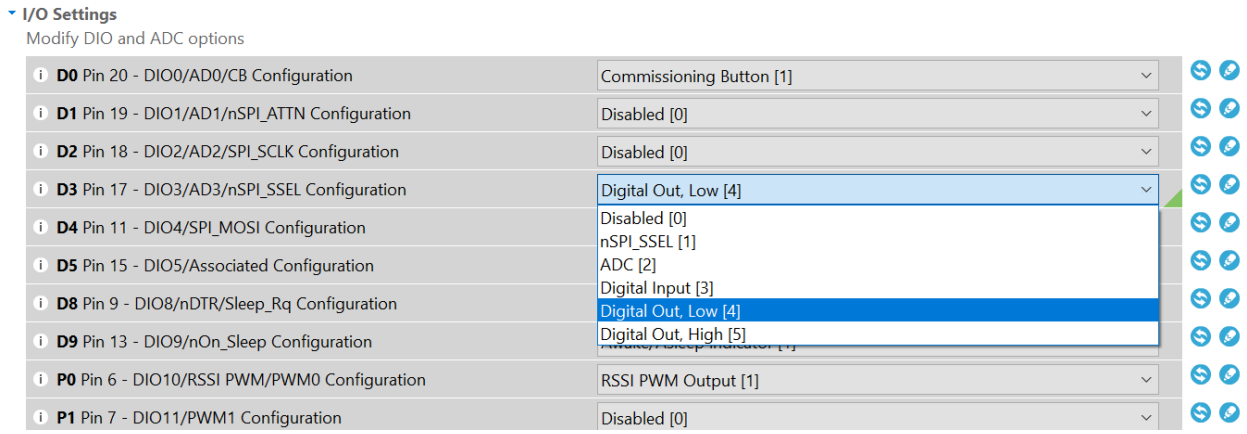
3.h) Lights (Router 3):

Function: Receives message from the coordinator and turns on the LEDs.

Components: XBee module, XBee breakout board, Breadboard, LEDs, connecting wires

Architecture: *(Aashaar)*

- Upload the settings in Router_AT_profile.xpro (\\Prototype code\zigbee) in XBee to configure it as router for the network. The Pan ID is set to 1002 for the network.
- In addition to the above settings, set D3 pin of the XBee to Digital Low[4]. This step is necessary as we want to use this pin to control the LEDs.



- Connect the xbee on breadboard & supply 3.3V to it. Take a connection out from D3 & connect it to LEDs.

Working: *(Aashaar)*

- The Xbee receives the message packet from the coordinator.
- Since this Xbee is configured in AT mode and when it receives a message in Remote AT Command frame format, it decodes the pin number and the value to which it must be set.
- Initially, D3 is configured to be LOW. Once the coordinator gets a message that motion has been detected, it will send a message which will set D3 to HIGH which turns on the LEDs.
- Then, when the camera is done capturing the image and it sends and ACK back to Coordinator, then the Coordinator again sends a message to reset D3 to LOW which turns off the LEDs.

3.i) Packet Structures: *(Aashaar)*

In a setup of XBee and Arduino, if the XBee is in AT mode, then the arduino is required to write only the payload, the XBee fills rest of the required packet fields from it's the values programmed into it during

its initial configuration. However, if the XBee is configured in API mode then the arduino must write every element of packet in its serial port. In our architecture, Coordinator is the only XBee that is configured in API mode. The following section describes the packets sent by Coordinator to other elements of the network.

Activate Camera (A) packet: Sent from Coordinator to Camera

7E	00	0F	10	00	00	13	A2	00	41	5B	8C	BA	FF	FE	00	00	41	checksum

- 7E – Starting delimiter of zigbee packet
- 00 – High part of the length of packet
- 0F – Low part of length of packet
- 10 – API frame type: Transmit request: Message will some payload be sent to specified destination address
- 00 – Frame ID – set to zero for no reply, can be set to 1, if an ack is required.
- 00 13 A2 00 41 5B 8C BA – destination address: MAC address of camera's xbee.
- FF FE – Network address
- 00 – set broadcast radius to default 0
- 00 – Set options to default 0
- 41 – Payload – ASCII value for 'A' is 65 which is '41' in hexadecimal format.
- Checksum: sum of everything that is non-zero after 'low part of length'. Checksum is used to verify the integrity of the packet.

Activate Light packet: Sent from Coordinator to Light

7E	00	10	17	00	00	13	A2	00	41	08	09	DD	FF	FE	02	44	33	05	checksum

- 7E – Starting delimiter of zigbee packet
- 00 – High part of the length of packet
- 0F – Low part of length of packet
- 17 – API frame type: Remote AT Command: Send a wireless command to another xbee.
- 00 – Frame ID – set to zero for no reply, can be set to 1, if an ack is required.
- 00 13 A2 00 41 08 09 DD – destination address: MAC address of light's xbee.
- FF FE – Network address
- 02 – set remote command options to 2: this will apply changes in destination xbee as soon as it receives this packet.
- 44 33 –specify the pin number whose value is to be changed. 44 33 is the hex value for 'D3', thus specifying pin D3.
- 05 – Value for pin specified above. 05 denotes HIGH.
- Checksum: sum of everything that is non-zero after 'low part of length'. Checksum is used to verify the integrity of the packet.

Resume Sensing (R) packet: Sent from Coordinator to PIR sensor

7E	00	0F	10	00	00	13	A2	00	41	08	09	D7	FF	FE	00	00	52	checksum

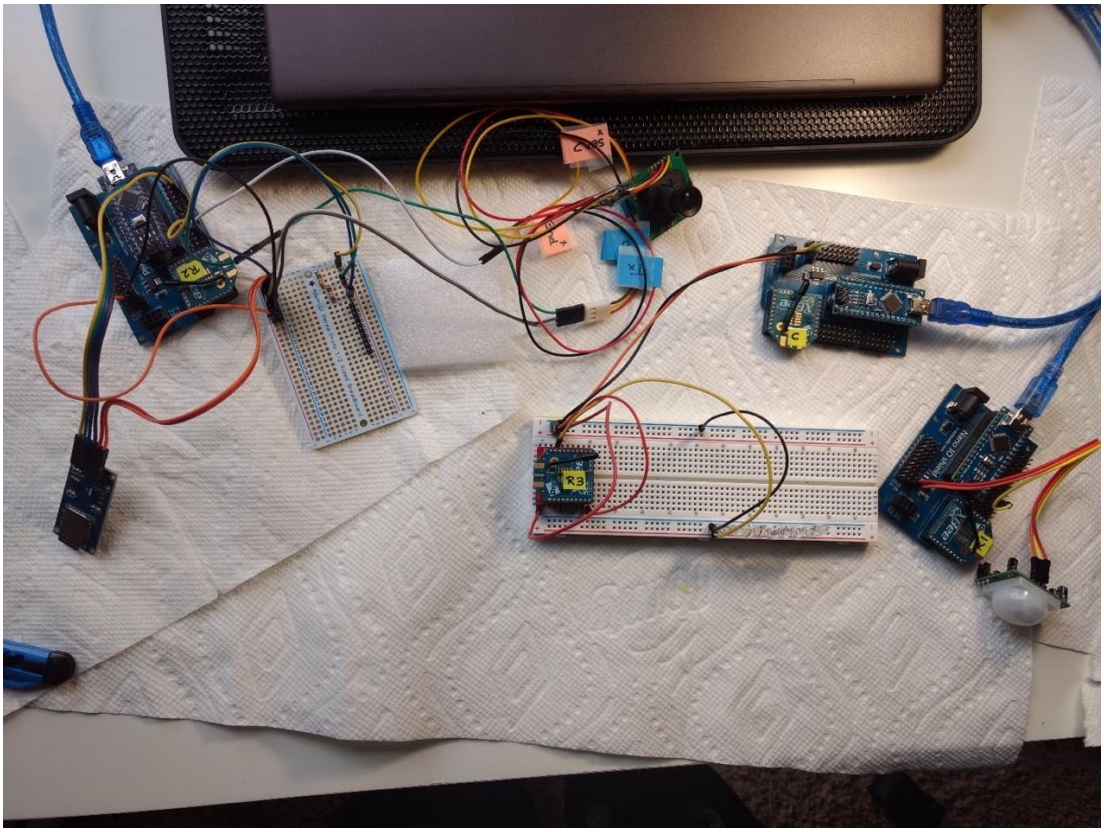
- 7E – Starting delimiter of zigbee packet
- 00 – High part of the length of packet
- 0F – Low part of length of packet
- 10 – API frame type: Transmit request: Message will some payload be sent to specified destination address
- 00 – Frame ID – set to zero for no reply, can be set to 1, if an ack is required.
- 00 13 A2 00 41 08 09 D7 – destination address: MAC address of PIR Sensor's receiver zigbee.
- FF FE – Network address
- 00 – set broadcast radius to default 0
- 00 – Set options to default 0
- 52 – Payload – ASCII value for 'R' is 82 which is '52' in hexadecimal format.
- Checksum: sum of everything that is non-zero after 'low part of length'. Checksum is used to verify the integrity of the packet.

Deactivate Light packet: Sent from Coordinator to Light

7E	00	10	17	00	00	13	A2	00	41	08	09	DD	FF	FE	02	44	33	04	checksum

- 7E – Starting delimiter of zigbee packet
- 00 – High part of the length of packet
- 0F – Low part of length of packet
- 17 – API frame type: Remote AT Command: Send a wireless command to another xbee.
- 00 – Frame ID – set to zero for no reply, can be set to 1, if an ack is required.
- 00 13 A2 00 41 08 09 DD – destination address: MAC address of light's xbee.
- FF FE – Network address
- 02 – set remote command options to 2: this will apply changes in destination xbee as soon as it receives this packet.
- 44 33 –specify the pin number whose value is to be changed. 44 33 is the hex value for 'D3', thus specifying pin D3.
- 04 – Value for pin specified above. 05 denotes LOW.
- Checksum: sum of everything that is non-zero after 'low part of length'. Checksum is used to verify the integrity of the packet.

3.j) Complete hardware setup: *(Aashaar)*



3.k) Image transfer from SD card to Raspberry Pi *(Pratik, Manva)*

- The Arduino has limited powers to multitask and multi-process. The TTL Camera paired with the Arduino stored the images into SD card.
- The Arduino Connected to the TTL Camera captured and stored images in the SD card. We needed that images to be shown on the web server.
- The architecture of Arduino allows us to convert the JPEG file to BITMAP (.bmp) file. Thus, we thought to transfer that BITMAP file from Arduino to Raspberry Pi.
- We used the following code to convert the image file to bitmap. That bitmap was used to transfer serially into Raspberry Pi. It retrieves the image from SD card attached with Arduino into the Raspberry Pi using serial transfer.
- Code:

```
1.  #include <SD.h>
2.
3.  File photoFile;
4.  const int buttonPin = 7;
5.  const int ledPin =  5;
6.
7.
8.  void setup() {
9.
10.   Serial.begin(115200);
11.   Serial.println("initializing sd card");
```

```
//  pinMode(10, OUTPUT);         // CS pin of SD Card Shield

  if (!SD.begin(10)) {
    Serial.print("sd initialzation failed");
    return;
  }
  //Serial.println("sd initialization done");
}


void loop() {

  while (1) {
    // Serial.println("press the button to send picture");
    Serial.flush();
    File photoFile = SD.open("salamander-black-and-white.jpg");
    if (photoFile) {
      while (photoFile.position() < photoFile.size()) {
        Serial.write(photoFile.read());
      }
      photoFile.close();
    }
    else {
      Serial.println("error sending photo");
    }
  }
}
```

- The above code sends the JPEG file from the SD card to Arduino in the bit format. It checks for the files, if they available and then transfers it serially to Pi.
- In consequence, the following code read the bit format and prints it.

```
import serial
ser = serial.Serial('/dev/ttyUSB0', 9600)
while 1:
    if(ser.in_waiting >0):
        line = ser.readline()
        print(line)
```

- This code took 4-5 minutes to transfer a single image. Using the Python code to read the serial output in Raspberry. We found out that we can get the bitmap of the image on the Raspberry Pi. But due to large amount of delay, we decided to change the camera integration and the layout of the architecture.

3.l) Arm/Disarm the system *(Manva)*

- The system that we created needed to be armed and disarmed via the user. To do so, we used the UI in the Raspberry Pi that was hosted on the web server to arm/disarm the system.
- We deployed the web server with the capabilities to login, display images from SD card and arm/disaram the Arduino.
- The logic was to start a function once the user pushes a button and toggle it to disarm once he repushes the button.
- Once the user selects to arm the system, the raspberry pi sends Arduino a message ‘1’ to start the system.

- Sending a '1' to the serial of Arduino signals it to start the sensor and arm the system using `arm()` function.
- Similiarly, the serial '0' from the Raspberry Pi instructs the Arduino to disarm and stop the system by activating the `disarm()` function.

```
#Python code to send a serial '1' and serial '0'
import serial
ser = serial.Serial('/dev/ttyUSB0', 9600)
if response.form['activate'] == 1:
  ser.write(b '1')
else :
  ser.write(b '0')
```

- On the Arduino side, the serial reads the '1' from the Pi and start a function that is analogous to switch "ON". The following code activates/deavtivates the Arduino.

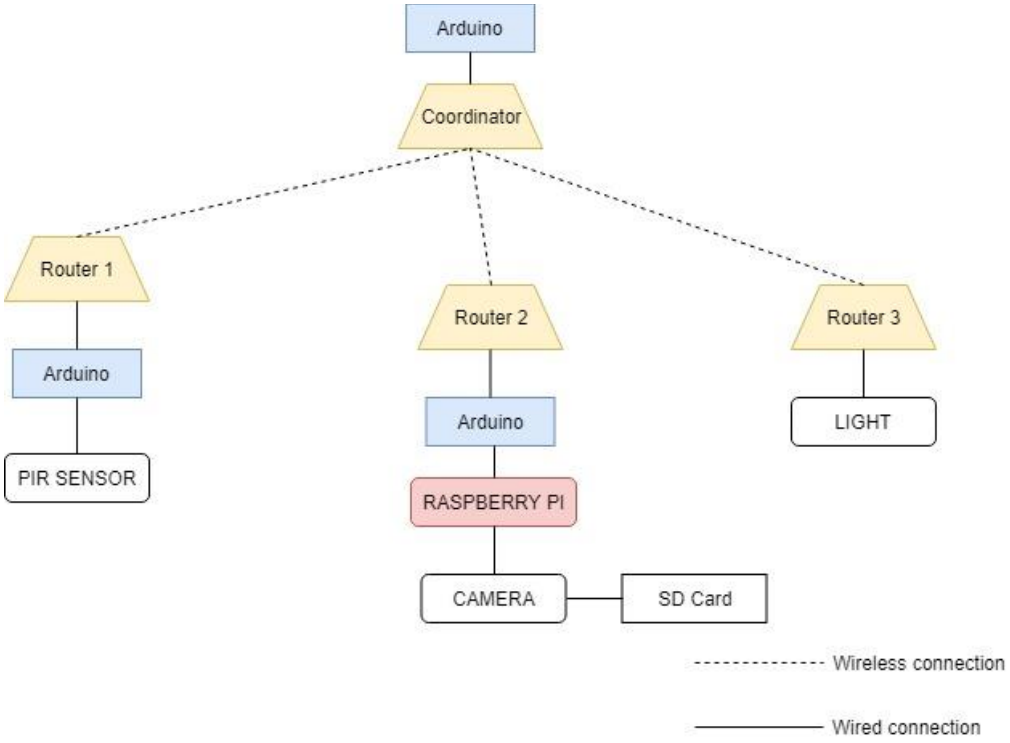
```
int r = 1;
void setup() {
  Serial.begin(9600);
}
void loop() {
  if (Serial.available()) { //From RPi to Arduino
    r = r * (Serial.read() - '0'); //converting the value of chars to integer
    if (r == '1') {
      arm(); //calling the function to arm the system
    } else {
      disarm(); //calling the function to disarm the system
    }
    Serial.println(r);
  }
}
```

3.m) Scope for improvement: *(Team)*

- The Arduino takes a lot of time to capture a picture & store it in the SD card.
- This is because the serial port of Arduino is very slow.
- Since the image capture is so slow, this cannot be enhanced to have live video streaming capabilities.
- The solution is to use a Raspberry Pi instead of an Arduino. Pi has greater significantly higher computational power than Arduino & it can support live video streaming with good response time.

4. Extension with Raspberry Pi (Web Server and Image Capturing):

4.a) Architecture Diagram: *(Team)*



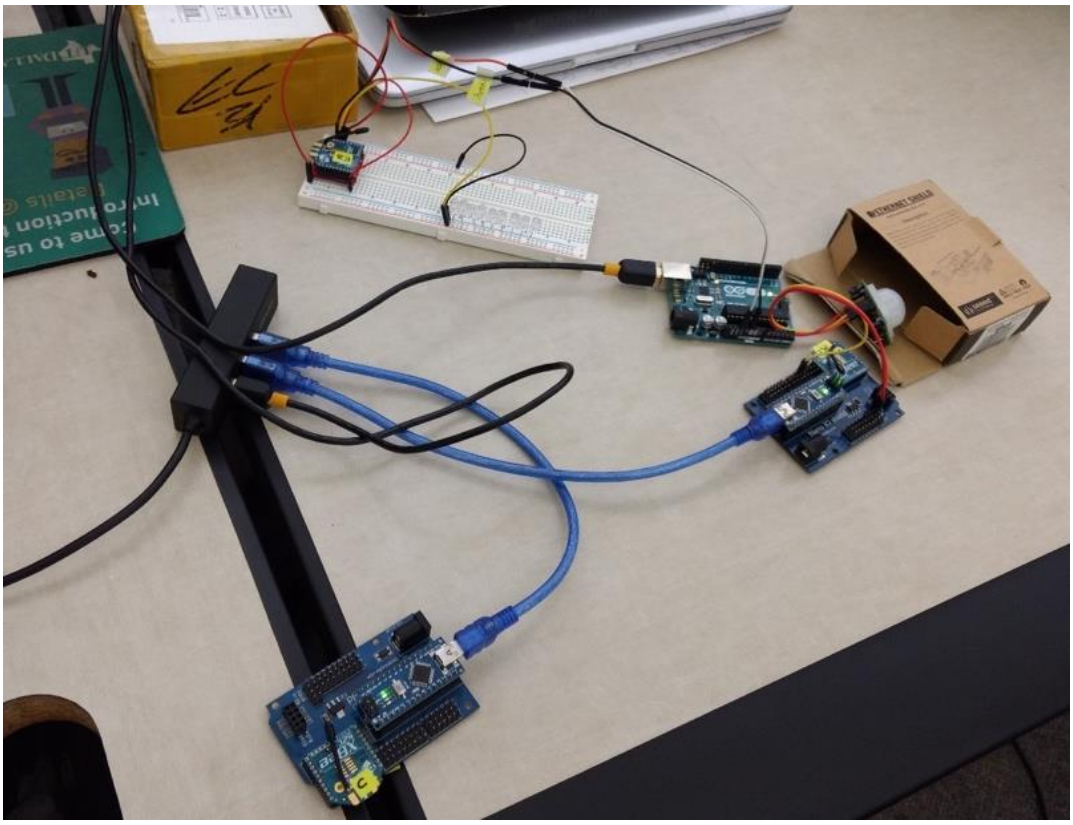
4.b) Camera with Pi (Router 2): (*Yangtao, Manva, Pratik)*

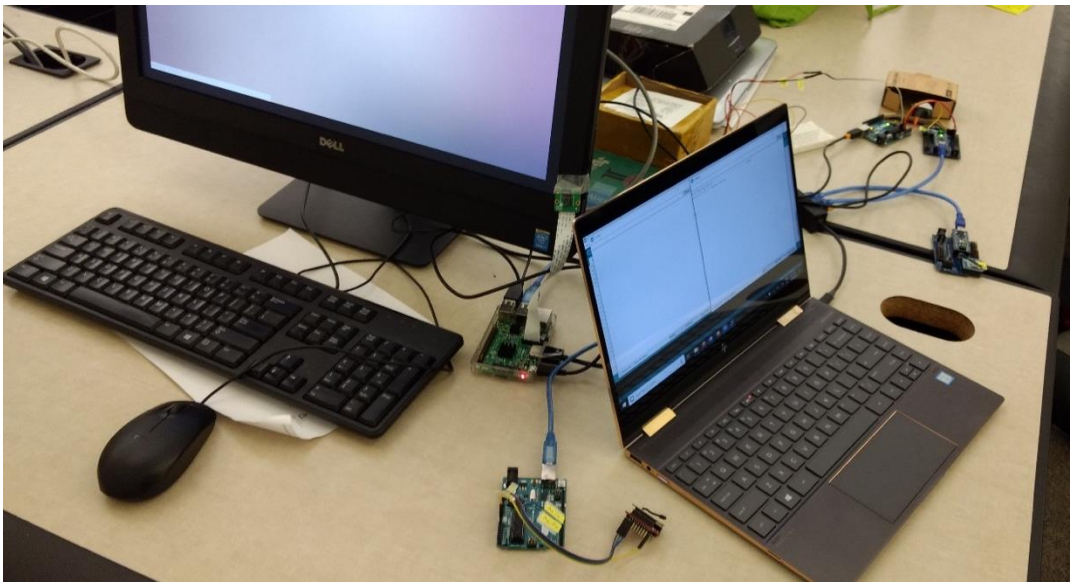
Function: Receives message from the coordinator, clicks pictures and then starts live video streaming on a webpage.

Components: Raspberry Pi, Arduino Uno, XBee module, Camera (model: V2-8 Megapixel), Sd Card.

Architecture: *(Yangtao, Manva)*

- The XBee module is connected to the Arduino. (Rx to Tx & vice-versa)
- The Rasberry Pi is connected to the Arduino (using a USB cable) and a camera.
- Upload the code in Camera_Pi_Router2.ino (\\Enhanced Pi Camera code\arduino\ Camera_Pi_Router2) in Arduino.
- Upload the settings in Router_AT_profile.xpro (\\Prototype code\zigbee) in XBee to configure it as router for the network. The Pan ID is set to 1002 for the network.





Working:

Arduino: *(Aashaar, Manva)*

- The Xbee receives the message packet from the coordinator & Arduino reads it from the serial port. If it receives a packet with payload ‘A’, the Arduino writes ‘A’ in its serial port.
- The Rasberry Pi is constantly reading Arduino’s serial port. When it reads an ‘A’ it triggers the camera function.

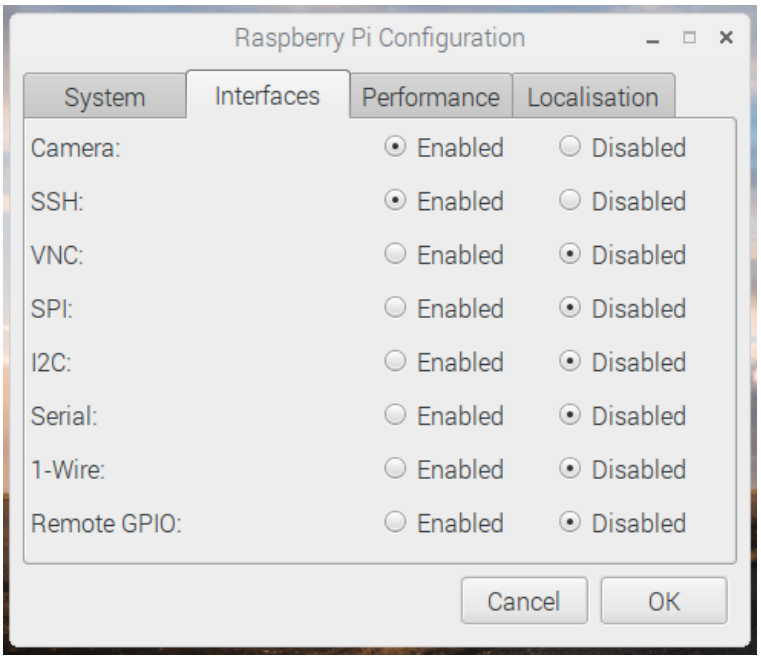
- The arduino is programmed to timeout after 30 seconds, once this interval is over, it kills the camera recording and sends a message with payload “D” back to the coordinator. When the coordinator receives this message, it turns off the lights and sends resume sensing message to the PIR sensor to restart the sensing operation.

Pi to Arduino: *(Manva, Yangtao)*

- Connect Arduino USB Plug to Raspberry PI with USB cable and check the connection between Arduino and Raspberry pi by type "ls /dev/tty*" in Raspberry Pi terminal, normally the result should be content "/dev/ttyACM0", remember it and set it to the code line.
- Let the Rasberry Pi Flask function constantly reading Arduino’s serial port after it received the URL request. When it reads an ‘A’ it will redirect the webpage and trigger camera to capture or take video stream.

Raspberry Pi: *(Manva, Pratik, Yangtao)*

- Ensure that Raspberry Pi is turned off.
- Connect the camera to pi’s camera port & then turn on the Pi.
- Go raspi-config and change the settings to match the following:



4.c) Flask *(Manva, Pratik, Yangtao)*

Introduction:

- Flask is a microframework for Python based on Werkzeug, Jinja 2. It can be used for creating full-stack web applications involving HTML, CSS and Javascript among many other technologies.
- It is called a micro-framework because core can be easily used a very extensible. There is no need for a lot of configuration files or the need to have a lot of files which are interconnected. Some of very good Flask applications can be developed using a single Python file.
- Flask was not the only option in this domain, and there were many alternatives which we could choose in place of Flask.
- However, what encouraged us to choose Flask over other web frameworks is that it is not memory-heavy and since it was to be deployed on a low processing power device like Raspberry pi which has very limited memory, it proved to be quite effective. Also, since not many files are required for development of the website, it's way better than some of its counter-parts such as Django which surely offers way more services than Flask but is heavy on memory and space.
- The main functionality we wanted to achieve from our application on Flask, was that of image capture and video streaming and therefore we needed an application which can take in data from our system and start the image capture and video streaming.

Working: *(Manva, Pratik, Yangtao)*

- We achieved these functionalities, by attaching the Pi Camera to a Raspberry Pi. The Flask server was deployed on the Raspberry Pi.
- To start with we created a login-page in flask and then created a database in SQLite to store the log-in details and other metadata.
- The user will log-in to the application using his username and password.

- After log-in, the user is redirected to the web-page in which he can see any previous photographs captured by the camera and stored in the system. If there is no motion detected, the user will stay on this page because there is nothing to capture.
- However, the application in the back-end is constantly waiting for a serial input from the Arduino on the camera side, which would denote the PIR sensor is sensing motion.
- Once the Flask application gets the signal which states that the motion is detected by the PIR sensor, from the Arduino, the image capture process will start, and the user will be redirected to the page where he can see the image capture being initiated.
- The images will be stored in the Operating System of the Raspberry Pi itself for future reference
- We have made our design in such a way that a photograph will be taken every 3 seconds for a period of 10 seconds and then will be stored on the Raspberry Pi itself, so that it is easily accessible.
- As soon as it is done, the user will be automatically redirected to the video streaming page, where he can clearly see the intruder on video.
- The video streaming will be on for a period of 30 seconds, which will enable us to get clear idea about the intruders.
- After which, the Flask application will again wait for serial input from the Arduino for the status of the PIR sensor and the user will be redirected to the page where he can see the previous images stored on the Raspberry Pi.

4.d) Code for flask:

“HelloWorld.py”

- In order to get acquainted with Flask and learn more about the routing process, we wrote many applications in Flask, first being Hello World. We have mentioned the code for it for future reference.
- We can run this application using the command: python HelloWorld.py then access URL: http://127.0.0.1:5000 to visit the webpage.

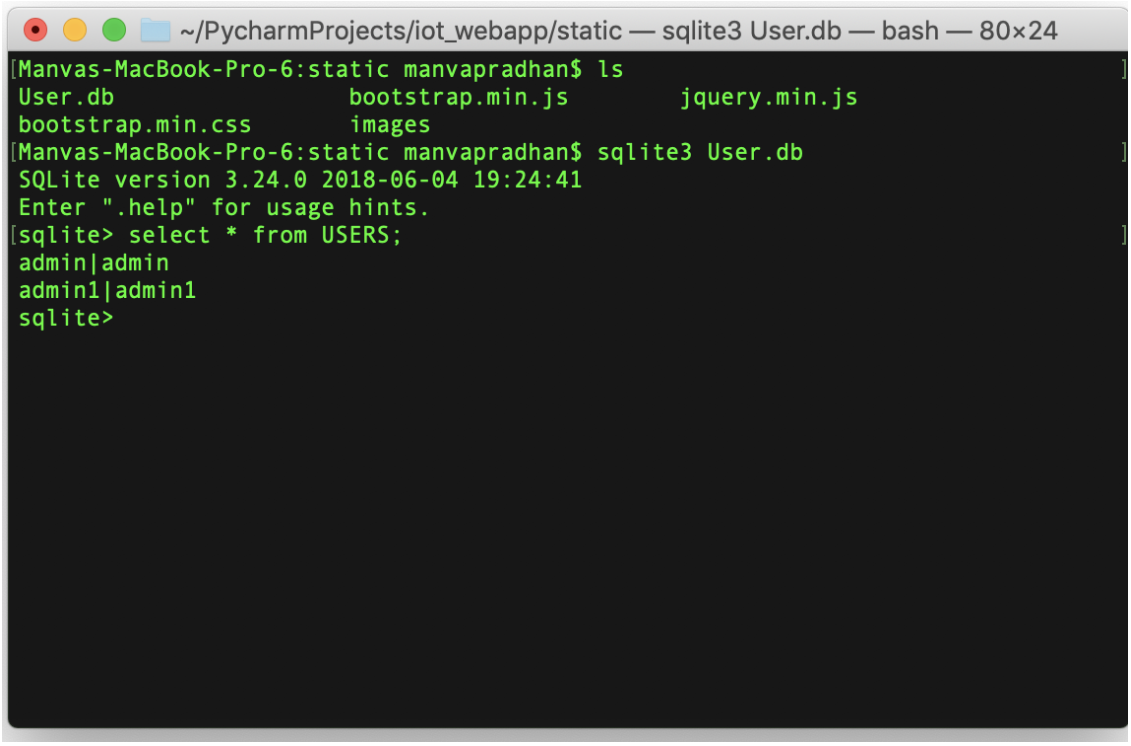
```
1. from flask
2. import Flask
3. app = Flask(__name__)
4. @app.route('/')
5. def index():
6.   return 'Hello world'
7. if __name__ == '__main__':
8.   app.run(debug = True, host = '0.0.0.0')
```

- This is just a simple ‘Hello World’ program.

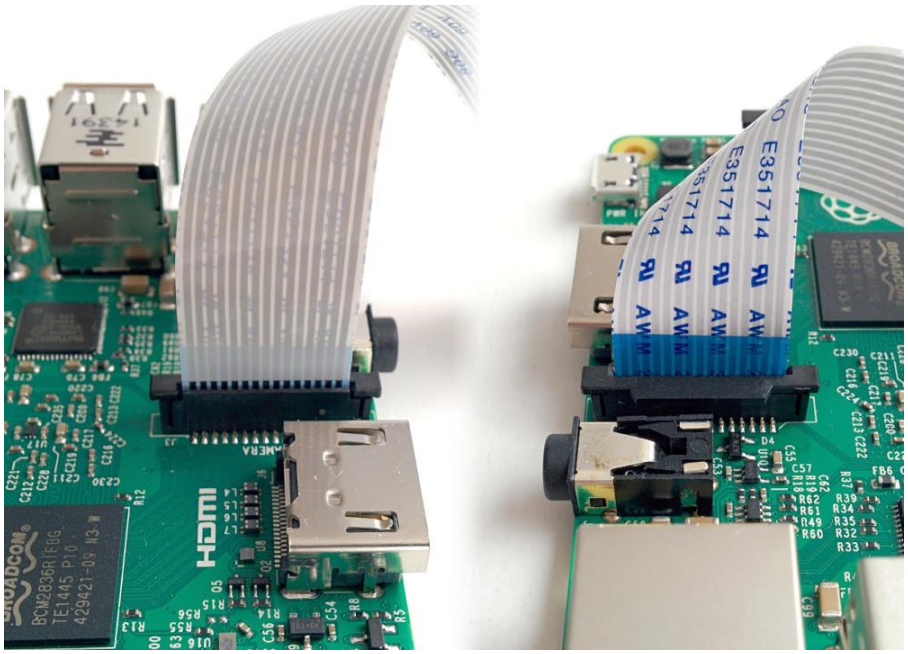
4.e) Setting up the SQLite Database: *(Manva, Pratik, Yangtao)*

- The database was used to store the details of the authenticated users. They had an username and password to add security to the existing architecture.

- The login page in the UI was the entry point for all the system to work i.e. arm and disarm the system.
- For database we used sqlite3 to store the data. The queries used were SELECT and CREATE query.



4.f) Code for camera: "Camera.py" *(Manva, Pratik, Yangtao)*



The camera should give a preview should for 10 seconds, and then close. Move the camera around to preview what the camera sees. The live camera preview should fill the screen.

```
from picamera
import PiCamera
from time
import sleep


camera = PiCamera()
camera.start_preview()
sleep(10)
camera.stop_preview()
```

To record video, amend your code to replace capture() with start_recording() and stop_recording(), run the code, it will record 10 seconds of video and then close the preview:

```
camera.start_preview()
camera.start_recording('/home/pi/video.h264')
sleep(10)
camera.stop_recording()
camera.stop_preview()
```

Core code to show the stream on the webpage, and there we used Bootstrap and Jquery to build the webpage:

Html part: (Manva, Pratik, Yangtao)

```
<div align="center">
    <title>Video Streaming Demo</title></div>
</head>
<style>
        body{
            background-image: url("https://hdwallsource.com/img/2014/9/blur-26347-27038-hd-wallpapers.jpg");
            background-repeat: no-repeat;
            background-position: center;
            background-size: cover;
            padding: 10px;
        }</style>
<body><font face="Verdana">
<h1>Video Streaming Demo</h1>
<div align="center"><img src="{{ url_for('video_feed') }}" width="60%" , height="60%" ></div>
```

Backend python code: (Manva, Pratik, Yangtao)

- *Camera operations:*

```
import time
import threading

try:
    from greenlet import getcurrent as get_ident
```

```
6.  except ImportError:
7.      try:
8.      from thread import get_ident
9.      except ImportError:
10.     from _thread import get_ident
11.
12.
13. class CameraEvent(object):
14.         """An Event-like class that signals all active clients when a new frame is
15.         available.
16.         """
17.     def __init__(self):
18.         self.events = {}
19.     def wait(self):
20.     """Invoked from each client's thread to wait for the next frame."""
21.     ident = get_ident()
22.     if ident not in self.events:
23.     # this is a new client
24.     # add an entry for it in the self.events dict
25.     # each entry has two elements, a threading.Event() and a timestamp
26.     self.events[ident] = [threading.Event(), time.time()]
27.     return self.events[ident][0].wait()
28. def set(self):
29.     """Invoked by the camera thread when a new frame is available."""
30.     now = time.time()
31.     remove = None
32.     for ident, event in self.events.items():
33.     if not event[0].isSet():
34.     # if this client's event is not set, then set it
35. # also update the last set timestamp to now
36.     event[0].set()
37.     event[1] = now
38.                 else:
39.      # if the client's event is already set, it means the client
40.     # did not process a previous frame
41.     # if the event stays set for more than 5 seconds, then assume
42.     # the client is gone and remove it
43.     if now - event[1] > 5:
44.         remove = ident
45.     if remove:
46.         del self.events[remove]
47. def clear(self):
48. """Invoked from each client's thread after a frame was processed."""
49.     self.events[get_ident()][0].clear()
```

Camera class for adding the live video streaming functionality to the system.

```
1.  Class BaseCamera(object):
2.      thread = None  # background thread that reads frames from camera
3.      frame = None  # current frame is stored here by background thread
4.      last_access = 0  # time of last client access to the camera
5.      event = CameraEvent()
6.
7.      def __init__(self):
8.          """Start the background camera thread if it isn't running yet."""
9.          if BaseCamera.thread is None:
10.             BaseCamera.last_access = time.time()
11.
12.             # start background frame thread
13.             BaseCamera.thread = threading.Thread(target=self._thread)
14.             BaseCamera.thread.start()
```

```
15.
16.             # wait until frames are available
17.             while self.get_frame() is None:
18.                 time.sleep(0)
19.
20.     def get_frame(self):
21.         """Return the current camera frame."""
22.         BaseCamera.last_access = time.time()
23.
24.         # wait for a signal from the camera thread
25.         BaseCamera.event.wait()
26.         BaseCamera.event.clear()
27.
28.         return BaseCamera.frame
29.
30.     @staticmethod
31.     def frames():
32.         """Generator that returns frames from the camera."""
33.         raise RuntimeError('Must be implemented by subclasses.')
34.
35.     @classmethod
36.     def _thread(cls):
37.         """Camera background thread."""
38.         print('Starting camera thread.')
39.         frames_iterator = cls.frames()
40.         for frame in frames_iterator:
41.             BaseCamera.frame = frame
42.             BaseCamera.event.set()  # send signal to clients
43.             time.sleep(0)
44.
45.             # if there hasn't been any clients asking for frames in
46.             # the last 10 seconds then stop the thread
47.             if time.time() - BaseCamera.last_access > 10:
48.                 frames_iterator.close()
49.                 print('Stopping camera thread due to inactivity.')
50.                 break
51.         BaseCamera.thread = None
```

- *Pi operation process:*

```
1.  import os
2.  import sqlite3
3.  from importlib import import_module
4.  from datetime import datetime
5.  import serial
6.  from flask import Flask, render_template, redirect, url_for, request
7.  from flask import Response
8.  import time
9.  import capture_image
10. if os.environ.get('CAMERA'):
11.     Camera = import_module('camera_' + os.environ['CAMERA']).Camera
12. else:
13.     from camera import Camera
14.
15. app = Flask(__name__)
16.
17.
18. def check_password(hashed_password, user_password):
19.     return hashed_password == user_password
20.
```

```
21.
22. def validate(username, password):
23.     con = sqlite3.connect('static/User.db')
24.     completion = False
25.     with con:
26.         cur = con.cursor()
27.         cur.execute('SELECT * FROM Users')
28.         rows = cur.fetchall()
29.         for row in rows:
30.             dbUser = row[0]
31.             dbPass = row[1]
32.             if dbUser == username:
33.                 completion = check_password(dbPass, password)
34.     return completion
35.
36.
37. @app.route('/', methods=['GET', 'POST'])
38. def login():
39.     error = None
40.     if request.method == 'POST':
41.         username = request.form['username']
42.         password = request.form['password']
43.         completion = validate(username, password)
44.         if not completion:
45.             error = 'Invalid Credentials. Please try again.'
46.         else:
47.             return redirect(url_for('success'))
48.     return render_template('login.html', error=error)
49.
50.
51. @app.route('/success', methods=['GET', 'POST'])
52. def success():
53.     if request.method == 'POST':
54.         return redirect((url_for('camera')))
55.     return render_template('success.html')
56.
57.
58. @app.route('/camera')
59. def camera():
60.     pics = os.listdir('./static/images')
61.     # hists = [os.path.join(os.getcwd() + '/static/images', file) for file in hists]
62.     return render_template('report.html', hists=pics)
63.
64.
65. @app.route('/video_feed')
66. def video_feed():
67.     '''''Video streaming route. Put this in the src attribute of an img tag.'''
68.     return Response(gen(Camera()), mimetype='multipart/x-mixed-
    replace; boundary=frame')
69.
70.
71. @app.route('/video', methods=['GET','POST'])
72. def index():
73.     '''''Video streaming home page.'''
74.     if request.method=="POST":
75.         return render_template('success.html')
76.
77.    ser = serial.Serial('/dev/ttyACM0', 9600)
78.     while True:
79.         read_serial = str(ser.readline())
80.         with open('log.txt','a+') as f:
81.             f.write(str(datetime.now()))
82.             f.write('\t')
83.             f.write(read_serial)
```

```
            f.write('\n')
        if 'A' in read_serial[2]:
            capture_image.capture_image()
            return render_template('video.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')


if __name__ == '__main__':
    app.run()
```

5. Conclusion:

We implemented an Automated Home security system that detects presence of an intruder and wirelessly starts a camera and light using Zigbee protocol. The prototype camera was able to capture images which was augmented to live video stream by replacing Arduino by Raspberry Pi. We can use this system for surveillance of residential and industrial locations.

6. References:

- 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
- 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
- https://www.instructables.com/id/Raspberry-Pi-Arduino-Serial-Communication/
- https://projects.raspberrypi.org/en/projects/python-web-server-with-flask/2

7. Code for Arduino *(Aashaar)*

PIR_Router1.ino:

```
//the time we give the sensor to calibrate (10-60 secs according to the datasheet)
int calibrationTime = 30;

//the time when the sensor outputs a low impulse
long unsigned int lowIn;

//the amount of milliseconds the sensor has to be low
//before we assume all motion has stopped
```

```
long unsigned int pause = 1000;

boolean lockLow = true;
boolean takeLowTime;

int pirPin = 3;    //the digital pin connected to the PIR sensor's output
int ledPin = 13;
bool flag = true;

////////////////////////////
//SETUP
void setup(){
  Serial.begin(9600);
  pinMode(pirPin, INPUT);
  pinMode(ledPin, OUTPUT);
  digitalWrite(pirPin, LOW);

  //give the sensor some time to calibrate
  Serial.print("calibrating sensor ");
    for(int i = 0; i < calibrationTime; i++){
      Serial.print(".");
      delay(1000);
      }
    Serial.println(" done");
    Serial.println("sensor active");
    delay(50);
  }

////////////////////////////
//LOOP
void loop(){

     if(Serial.available()>0)
     {
        byte receivedMessage = Serial.read();

        Serial.println(receivedMessage);
        if (receivedMessage == 0x52) // check for codeword "R"
        {
          flag = true;
          Serial.println(flag);
        }
     }

     if(digitalRead(pirPin) == HIGH && flag == true){
       digitalWrite(ledPin, HIGH);   //the led visualizes the sensors output pin state
       //if(lockLow){
         //makes sure we wait for a transition to LOW before any further output is made:
         //lockLow = false;
         delay(1500); // delay to let communication between router & cooordinator
         Serial.println("M"); // send codeword to Coordinator to indicate motion is detected
         flag = false;
         //Serial.println("---");
         //Serial.print("motion detected at ");
         //Serial.print(millis()/1000);
         //Serial.println(" sec");
         delay(50);
         //}
         //takeLowTime = true;
       }

     if(digitalRead(pirPin) == LOW){
       digitalWrite(ledPin, LOW);  //the led visualizes the sensors output pin state
```

```
72.
73.  }
74. }
```

Coordinator:

```
1.  byte RFin_bytes[19];
2.  #define MAX_MILLIS_TO_WAIT 15000  //timeout
3.  unsigned long starttime;
4.  #define motion 0x4D
5.  #define positiveAck 0x44 // codeword "D"
6.  #define negativeAck 0x46 // codeword "F"
7.
8.  void setup() {
9.    // put your setup code here, to run once:
10.   Serial.begin(9600);
11.   Serial.println("Coordinator")
12.
13.   //pinMode(ledPin, OUTPUT);
14.
15. }
16.
17. void loop() {
18.   // put your main code here, to run repeatedly:
19.   starttime = millis();
20.   delay(1000);
21.   while (Serial.available())
22.   {
23.     byte startDelimiter = Serial.read();
24.
25.     if (startDelimiter == 0x7E) //check for start delimiter of the API packet.
26.     {
27.
28.       while ( (Serial.available()<18)) // total message size is 19 byte. Since 1 byte is alrea
    dy read, we wait for rest 18 bytes
29.       {
30.          // hang in this loop until we either get 18 bytes of data or 15 seconds(timeout)
31.       }
32.       if(Serial.available() < 18)
33.       {
34.                // if the data didn't come in - handle that problem here
35.          Serial.println("error - Didn't get 18 bytes of data!");
36.       }
37.       else
38.       {
39.          RFin_bytes[0] = 0x7E;   // put first byte StartDelimiter already read
40.          for(int n=1; n<19; n++)
41.             RFin_bytes[n] = Serial.read(); // Then: Get them.
42.
43.          // print entire payload
44.          /*
45.          Serial.println("start of msg");
46.          for(int n=0; n<19; n++)
47.          {
48.             Serial.print(n);
49.             Serial.print(" - ");
50.             Serial.println(RFin_bytes[n]);
51.          }
52.          Serial.println("===============");
53.          */
54.
55.          if (RFin_bytes[15] == motion)
```

```
56.          {
57.             Serial.println("motion detected");
58.             activateCamera(); // function call to send API packet to camera's xbee
59.             delay(1000);
60.             activateLight(); // function call to send API packet to Light's xbee
61.             Serial.println("activation message sent to camera");
62.          }
63.          else if (RFin_bytes[15] == positiveAck)
64.          {
65.             Serial.println("positive acknowledgement received from camera");
66.             resumeSensing(); // function call to send packet to resume sensing
67.             delay(1000);
68.             deactivateLight(); // function call to turn off the light
69.             Serial.println("resume sensing msg sent");
70.          }
71.          else if (RFin_bytes[15] == negativeAck)
72.          {
73.             Serial.println("********** error encountered in capturing image at camera. please
    check the camera. ************");
74.             resumeSensing(); // function call to send packet to resume sensing
75.             delay(1000);
76.             deactivateLight(); // function call to turn off the light
77.             Serial.println("resume sensing msg sent");
78.          }
79.        }
80.     }
81.     /*
82.     else
83.       Serial.print("****************");
84.       Serial.println(startDelimiter);
85.       */
86.   }
87. }
88.
89.
90. void activateCamera()
91. {
92.   Serial.write(0x7E); // start byte
93.   Serial.write((byte)0x00); //high part of length
94.   Serial.write(0x0F); //low part of length, the number of bytes that follow this line excludin
    g Checksum
95.   Serial.write(0x10); // API frame type: Transmit Request
96.   Serial.write((byte)0x0); //frame id set to zero for no reply
97.   // Following 8 bytes are destination address - MAC address of Camera's receiver zigbee - 001
    3A200415B8CBA
98.   Serial.write(0x00);
99.   Serial.write(0x13);
100.         Serial.write(0xA2);
101.         Serial.write(0x00);
102.         Serial.write(0x41);
103.         Serial.write(0x5B);
104.         Serial.write(0x8C);
105.         Serial.write(0xBA);
106.         // destination address ends
107.
108.         // Following 2 bytes are Network address - set to default values - 0xFF 0xFE
109.         Serial.write(0xFF);
110.         Serial.write(0xFE);
111.         // network address ends
112.
113.         Serial.write(0x00); //set broadcast radius to default 0
114.         Serial.write(0x00); // set options to default 0
115.
```

```
116.          Serial.write(0x41); // Payload = "A" -> A denotes 'Activate camera'
117.
118.          //checksum - sum of everything that is non-zero after 'low part of length'
119.          long checksum = 0x10 + 0x13 + 0xA2 + 0x41 + 0x5B + 0x8C + 0xBA + 0xFF + 0xFE + 0x41;

120.          checksum = 0xFF - (checksum & 0xFF);
121.          Serial.write(checksum); //write checksum to Serial
122.        }
123.
124.        void resumeSensing()
125.        {
126.          Serial.write(0x7E); // start byte
127.          Serial.write((byte)0x00); //high part of length
128.          Serial.write(0x0F); //low part of length, the number of bytes that follow this line e
    xcluding Checksum
129.          Serial.write(0x10); // API frame type: Transmit Request
130.          Serial.write((byte)0x0); //frame id set to zero for no reply
131.          // Following 8 bytes are destination address - MAC address of PIR Sensor's receiver z
    igbee - 0013A200410809D7
132.          Serial.write(0x00);
133.          Serial.write(0x13);
134.          Serial.write(0xA2);
135.          Serial.write(0x00);
136.          Serial.write(0x41);
137.          Serial.write(0x08);
138.          Serial.write(0x09);
139.          Serial.write(0xD7);
140.          // destination address ends
141.
142.          // Following 2 bytes are Network address - set to default values - 0xFF 0xFE
143.          Serial.write(0xFF);
144.          Serial.write(0xFE);
145.          // network address ends
146.
147.          Serial.write(0x00); //set broadcast radius to default 0
148.          Serial.write(0x00); // set options to default 0
149.
150.          Serial.write(0x52); // Payload = "R" -> A denotes 'Resume Sensing'
151.
152.          //checksum - sum of everything that is non-zero after 'low part of length'
153.          long checksum = 0x10 + 0x13 + 0xA2 + 0x41 + 0x08 + 0x09 + 0xD7 + 0xFF + 0xFE + 0x52;

154.          checksum = 0xFF - (checksum & 0xFF);
155.          Serial.write(checksum); //write checksum to Serial
156.        }
157.
158.        void activateLight()
159.        {
160.          Serial.write(0x7E); // start byte
161.          Serial.write((byte)0x00); //high part of length
162.          Serial.write(0x10); //low part of length, the number of bytes that follow this line e
    xcluding Checksum
163.          Serial.write(0x17); // API frame type:  Remote AT Command: Send a wireless command to
     another xbee.
164.          Serial.write((byte)0x0); //frame id set to zero for no reply
165.          // Following 8 bytes are destination address - MAC address of PIR Sensor's receiver z
    igbee - 0013A200410809D7
166.          Serial.write(0x00);
167.          Serial.write(0x13);
168.          Serial.write(0xA2);
169.          Serial.write(0x00);
170.          Serial.write(0x41);
171.          Serial.write(0x08);
```

```
172.         Serial.write(0x09);
173.         Serial.write(0xDD);
174.         // destination address ends
175.
176.         // Following 2 bytes are Network address - set to default values - 0xFF 0xFE
177.         Serial.write(0xFF);
178.         Serial.write(0xFE);
179.         // network address ends
180.
181.         Serial.write(0x02); //set remote command options to 02. this will apply changes in de
    stination xbee as soon as it receives this packet.
182.
183.         Serial.write(0x44); // AT command - Pin D3
184.         Serial.write(0x33); // AT command - Pin D3
185.
186.         Serial.write(0x5); // Parameter Value. (HIGH)
187.
188.         //checksum - sum of everything that is non-zero after 'low part of length'
189.         long checksum = 0x17 + 0x13 + 0xA2 + 0x41 + 0x08 + 0x09 + 0xDD + 0xFF + 0xFE + 0x02 +
    0x44 + 0x33 +0x5;
190.         checksum = 0xFF - (checksum & 0xFF);
191.         Serial.write(checksum); //write checksum to Serial
192.       }
193.
194.
195.       void deactivateLight()
196.       {
197.         Serial.write(0x7E); // start byte
198.         Serial.write((byte)0x00); //high part of length
199.         Serial.write(0x10); //low part of length, the number of bytes that follow this line e
    xcluding Checksum
200.         Serial.write(0x17); // API frame type:  Remote AT Command: Send a wireless command to
     another xbee.
201.         Serial.write((byte)0x0); //frame id set to zero for no reply
202.         // Following 8 bytes are destination address - MAC address of PIR Sensor's receiver z
    igbee - 0013A200410809D7
203.         Serial.write(0x00);
204.         Serial.write(0x13);
205.         Serial.write(0xA2);
206.         Serial.write(0x00);
207.         Serial.write(0x41);
208.         Serial.write(0x08);
209.         Serial.write(0x09);
210.         Serial.write(0xDD);
211.         // destination address ends
212.
213.         // Following 2 bytes are Network address - set to default values - 0xFF 0xFE
214.         Serial.write(0xFF);
215.         Serial.write(0xFE);
216.         // network address ends
217.
218.         Serial.write(0x02); //set remote command options to 02. this will apply changes in de
    stination xbee as soon as it receives this packet.
219.
220.         Serial.write(0x44); // AT command - Pin D3
221.         Serial.write(0x33); // AT command - Pin D3
222.
223.         Serial.write(0x4); // Parameter Value. (LOW)
224.
225.         //checksum - sum of everything that is non-zero after 'low part of length'
226.         long checksum = 0x17 + 0x13 + 0xA2 + 0x41 + 0x08 + 0x09 + 0xDD + 0xFF + 0xFE + 0x02 +
    0x44 + 0x33 + 0x4;
227.         checksum = 0xFF - (checksum & 0xFF);
```

```
228.         Serial.write(checksum); //write checksum to Serial
229.       }
```

Camera_Router2 :

```
1.  // This is a basic snapshot sketch using the VC0706 library.
2.  // On start, the Arduino will find the camera and SD card and
3.  // then snap a photo, saving it to the SD card.
4.  // Public domain.
5.
6.  // If using an Arduino Mega (1280, 2560 or ADK) in conjunction
7.  // with an SD card shield designed for conventional Arduinos
8.  // (Uno, etc.), it's necessary to edit the library file:
9.  //   libraries/SD/utility/Sd2Card.h
10. // Look for this line:
11. //   #define MEGA_SOFT_SPI 0
12. // change to:
13. //   #define MEGA_SOFT_SPI 1
14. // This is NOT required if using an SD card breakout interfaced
15. // directly to the SPI bus of the Mega (pins 50-53), or if using
16. // a non-Mega, Uno-style board.
17.
18. #
19. include < Adafruit_VC0706.h > #include < SPI.h > #include < SD.h >
20.   int my = 0;
21. bool flag = true;
22. // comment out this line if using Arduino V23 or earlier
23. #
24. include < SoftwareSerial.h >
25.
26.   // uncomment this line if using Arduino V23 or earlier
27.   // #include <NewSoftSerial.h>
28.
29.   // SD card chip select line varies among boards/shields:
30.   // Adafruit SD shields and modules: pin 10
31.   // Arduino Ethernet shield: pin 4
32.   // Sparkfun SD shield: pin 8
33.   // Arduino Mega w/hardware SPI: pin 53
34.   // Teensy 2.0: pin 0
35.   // Teensy++ 2.0: pin 20
36.   #define chipSelect 10
37.
38. // Pins for camera connection are configurable.
39. // With the Arduino Uno, etc., most pins can be used, except for
40. // those already in use for the SD card (10 through 13 plus
41. // chipSelect, if other than pin 10).
42. // With the Arduino Mega, the choices are a bit more involved:
43. // 1) You can still use SoftwareSerial and connect the camera to
44. //    a variety of pins...BUT the selection is limited.  The TX
45. //    pin from the camera (RX on the Arduino, and the first
46. //    argument to SoftwareSerial()) MUST be one of: 62, 63, 64,
47. //    65, 66, 67, 68, or 69.  If MEGA_SOFT_SPI is set (and using
48. //    a conventional Arduino SD shield), pins 50, 51, 52 and 53
49. //    are also available.  The RX pin from the camera (TX on
50. //    Arduino, second argument to SoftwareSerial()) can be any
51. //    pin, again excepting those used by the SD card.
52. // 2) You can use any of the additional three hardware UARTs on
53. //    the Mega board (labeled as RX1/TX1, RX2/TX2, RX3,TX3),
54. //    but must specifically use the two pins defined by that
55. //    UART; they are not configurable.  In this case, pass the
56. //    desired Serial object (rather than a SoftwareSerial
57. //    object) to the VC0706 constructor.
```

```
58.
59. // Using SoftwareSerial (Arduino 1.0+) or NewSoftSerial (Arduino 0023 & prior):
60. #
61. if ARDUINO >= 100
62. // On Uno: camera TX connected to pin 2, camera RX to pin 3:
63. SoftwareSerial cameraconnection = SoftwareSerial(2, 3);
64. // On Mega: camera TX connected to pin 69 (A15), camera RX to pin 3:
65. //SoftwareSerial cameraconnection = SoftwareSerial(69, 3);
66. #
67. else
68.   NewSoftSerial cameraconnection = NewSoftSerial(2, 3);#
69. endif
70.
71. Adafruit_VC0706 cam = Adafruit_VC0706( & cameraconnection);
72.
73. // Using hardware serial on Mega: camera TX conn. to RX1,
74. // camera RX to TX1, no SoftwareSerial object is required:
75. //Adafruit_VC0706 cam = Adafruit_VC0706(&Serial1);
76.
77. void setup() {
78.
79.   // When using hardware SPI, the SS pin MUST be set to an
80.   // output (even if not connected or used).  If left as a
81.   // floating input w/SPI on, this can cause lockuppage.
82.   #
83.   if !defined(SOFTWARE_SPI)# if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
84.   if (chipSelect != 53) pinMode(53, OUTPUT); // SS on Mega
85.   #
86.   else
87.   if (chipSelect != 10) pinMode(10, OUTPUT); // SS on Uno, etc.
88.   #
89.   endif# endif
90.
91.   Serial.begin(9600);
92.   Serial.println("vc0706 Camera snapshot test");
93.
94.   // see if the card is present and can be initialized:
95.
96. }
97.
98. void loop() {
99.   if (Serial.available() > 0) {
100.           //Serial.println(Serial.read());
101.           byte receivedMessage = Serial.read();
102.
103.           if (receivedMessage == 0x41) // check for codeword "A"
104.           {
105.             if (!SD.begin(chipSelect)) {
106.               Serial.println("card failed, or not present");
107.               flag = false;
108.               Serial.println("F"); // codeword for Failed (negative ack) to be sent to the co
    ordinator.
109.               // don't do anything more:
110.               return;
111.             }
112.
113.             // Try to locate the camera
114.             if (cam.begin()) {
115.               Serial.println("camera found:");
116.             } else {
117.               Serial.println("no camera found?");
118.               flag = false;
```

```
119.               Serial.println("F"); // codeword for Failed (negative ack) to be sent to the co
    ordinator.
120.               return;
121.             }
122.             // Print out the camera version information (optional)
123.             char * reply = cam.getVersion();
124.             if (reply == 0) {
125.               Serial.print("failed to get version");
126.               Serial.println("F"); // codeword for Failed (negative ack) to be sent to the co
    ordinator.
127.             } else {
128.               flag = false;
129.               Serial.println("-----------------");
130.               Serial.print(reply);
131.               Serial.println("-----------------");
132.             }
133.
134.             // Set the picture size - you can choose one of 640x480, 320x240 or 160x120
135.             // Remember that bigger pictures take longer to transmit!
136.
137.             cam.setImageSize(VC0706_640x480); // biggest
138.             //cam.setImageSize(VC0706_320x240);        // medium
139.             //cam.setImageSize(VC0706_160x120);          // small
140.
141.             // You can read the size back from the camera (optional, but maybe useful?)
142.             uint8_t imgsize = cam.getImageSize();
143.             Serial.print("image size: ");
144.             if (imgsize == VC0706_640x480) Serial.println("640x480");
145.             if (imgsize == VC0706_320x240) Serial.println("320x240");
146.             if (imgsize == VC0706_160x120) Serial.println("160x120");
147.
148.             //Serial.println("Snap in 10 secs...");
149.             //delay(10000);
150.
151.             if (!cam.takePicture()) {
152.               Serial.println("failed to snap!");
153.               Serial.println("F"); // codeword for Failed (negative ack) to be sent to the co
    ordinator.
154.             } else
155.               Serial.println("picture taken!");
156.
157.             // Create an image with the name IMAGExx.JPG
158.             char filename[13];
159.             strcpy(filename, "IMAGE00.JPG");
160.             for (int i = 0; i < 100; i++) {
161.               filename[5] = '0' + i / 10;
162.               filename[6] = '0' + i % 10;
163.               // create if does not exist, do not open existing, write, sync after write
164.               if (!SD.exists(filename)) {
165.                 break;
166.               }
167.             }
168.
169.             // Open the file for writing
170.             File imgFile = SD.open(filename, FILE_WRITE);
171.
172.             // Get the size of the image (frame) taken
173.             uint16_t jpglen = cam.frameLength();
174.             Serial.print("storing ");
175.             Serial.print(jpglen, DEC);
176.             Serial.print(" byte image.");
177.
178.             int32_t time = millis();
```

```
179.              pinMode(8, OUTPUT);
180.              // Read all the data up to # bytes!
181.              byte wCount = 0; // For counting # of writes
182.              while (jpglen > 0) {
183.                // read 32 bytes at a time;
184.                uint8_t * buffer;
185.                uint8_t bytesToRead = min(32, jpglen); // change 32 to 64 for a speedup but may
     not work with all setups!
186.                buffer = cam.readPicture(bytesToRead);
187.                imgFile.write(buffer, bytesToRead);
188.                if (++wCount >= 64) { // Every 2K, give a little feedback so it doesn't appear
    locked up
189.                  Serial.print('.');
190.                  wCount = 0;
191.                }
192.                //Serial.print("read ");  Serial.print(bytesToRead, DEC); Serial.println(" byte
    s");
193.                jpglen -= bytesToRead;
194.              }
195.              imgFile.close();
196.
197.              time = millis() - time;
198.              Serial.println("done!");
199.              Serial.println("picture captured & stored in SD card");
200.              Serial.print(time);
201.              Serial.println(" ms elapsed");
202.              delay(1000);
203.              Serial.println("D"); // codeword for Done (positive ack) to be sent to the coordi
    nator.
204.            }
205.          }
206.          //Serial.println("program Complete");
207.
208.        }
```

- End of document