○Coordinator と Router と Enddevice をギリギリの接続可能な距離に配置し，
Router に sniffer を配置する.これにより Association 等を確認する．

以下はデータが届いた配置
E：院生部屋（デスク）　　R：院生部屋出口（廊下側）
C：木下先生部屋の奥

以下はデータが届かなかった配置
E：院生部屋（デスク）　　R：アルベルト先生部屋と池田先生部屋の間
C：木下先生部屋の奥

また，以下の画像のように A と B のデータの流れを構築可能だが、Wireshark での見やすさと，具体的なアプリケーションを考慮して，A を採用している．

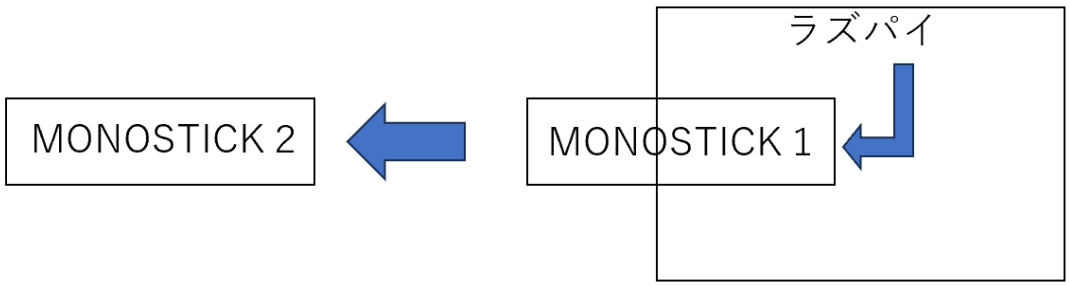
A

B

Wireshark でのデータファイル（1005.pnapng）は添付してある．

今回使用したデバイスの 64bit の MAC アドレス
C : 0x001BC50122016BD5
R : 0x001BC50122016C13
E : 0x001BC50122016BDD

○　【ラズパイから MONOSTICK 1 (NXP)にコマンドを送信する.】
　　↓
　　【MONOSTICK 1 から MONOSTICK 2 に送信する.】
　　↓
　　【MONOSTICK 2 でコマンドを受信する．】



目標として,
ラズパイからシリアル通信で事前に決めたコマンドを送信　→　MONOSTICK で受信

上記の目標達成後、センサ回路を正確に組んで、コマンド＋センサデータを送信可能にする.
現在はセンサから値を取得できているが，値が不変である.

【ラズパイから MONOSTICK 1 (NXP)にコマンドを送信する.】
以下はラズパイから MONOSTICK 1 にコマンドをシリアル通信で送信するプログラムの実行結果とプログラムコードである.

```
KYOHEI@KYOHEI:~/Desktop $ python loopback.py
KYOHEI@KYOHEI:~/Desktop $ python test.py
===== Start Program =====

===== Set Parameter Complete =====

Read Serial:
b''
b''
b''
===== Read Serial Complete =====

===== Write Serial Complete =====

===== End Program =====
```

```python
import serial
import struct
import binascii
import sys

use_port = '/dev/ttyUSB0'

_serial = serial.Serial(use_port)
_serial.baudrate = 9600
_serial.parity = serial.PARITY_NONE
_serial.bytesize = serial.EIGHTBITS
_serial.stopbits = serial.STOPBITS_ONE
_serial.timeout = 5 #sec

#commands = [ 0xB6, 0x01, 0x02, 0x00 ]

#for cmd in commands:
#    data = struct.pack("B", cmd)
#    print("tx: ", data)
#    _serial.write(data) #データ送信

serialCommand = "Rx_Data"
#writeSer.write(serialCommand.encode())
_serial.write(serialCommand.encode()) #データ送信
_serial.flush() #データを送信するまで待機


#rx = _serial.readline() #データ受信
#print("rx: ", rx.hex()) #16進数文字列で表記  0x表記がなくなる

_serial.close()

```

loopback.py

```python
import serial

print("===== Start Program =====\n")

# Set Parameter
deviceName = '/dev/ttyUSB0'
baudrateNum = 9600
timeoutNum = 3
print("===== Set Parameter Complete =====\n")

# # Read Serial
readSer = serial.Serial(deviceName, baudrateNum, timeout=timeoutNum)
c = readSer.read()
string = readSer.read(10)
line = readSer.readline()
print("Read Serial:")
print(c)
print(string)
print(line)
readSer.close()
print("===== Read Serial Complete =====\n")

# Write Serial
serialCommand = "Rx_Data"
writeSer = serial.Serial(deviceName, baudrateNum, timeout=timeoutNum)
writeSer.write(serialCommand.encode())
writeSer.close()
print("===== Write Serial Complete =====\n")

print("===== End Program =====\n")

```

test.py

ラズパイから「Rx_Data」というコマンドを受信すると，確認用データを MONOSTICK2 に送信するプログラムになっている．
今後はセンサ値を加えて送信する必要がある．

【MONOSTICK 1 から MONOSTICK 2 に送信する.】
MONOSTICK 同士のデータ送信は構築済みである．
今回は受信したコマンドを送信したのではなく，受信の確認データを送信した．以下が追加コードである．
今後は受信したデータそのものを送信可能する必要がある．

```
PRIVATE void vReadInputCommand()
{
  commandType currentCommand = NO_COMMAND;
  currentCommand = vReadCommand (); //Utils.cの関数
 // DBG_vPrintf(TRUE, "%d\n", currentCommand); //現在のコマンドを表示

  if (currentCommand == SEND_COMMAND)
    {

       SendData();
       currentCommand = NO_COMMAND;
    }
  else if(currentCommand == RX_COMMAND){ //追加コード

      SendData();
      currentCommand = NO_COMMAND;
  }
}

PUBLIC void SendData(){
```

```
PUBLIC commandType vReadCommand ()
{
  if (strlen(command) == 0)
    {
      return NO_COMMAND;
    }
  ///追加コマンド////
  else if(strcmp(command, "Rx_Data") == 0)
  {
      DBG_vPrintf(TRUE,"OK\r\n");
      memset(command,0,sizeof(command));
        return RX_COMMAND;
  }
  //////////////
  else if (strcmp(command,"send") == 0)
      {
      //DBG_vPrintf(TRUE,"%s Send command \n",command);
      memset(command,0,sizeof(command));
        return SEND_COMMAND;
      }
```

【MONOSTICK 2 でコマンドを受信する．】
今回は確認用データを受信した． これは構築済みである．
今後は受信したコマンドを認識できるようにする必要がある．