



Raspberry Pi RFID Attendance System

Submitted by

KANISHKAR A V	(CB.EN.U4AIE20026)
MANESH KARUN R	(CB.EN.U4AIE20035)
NITHESH B	(CB.EN.U4AIE20043)
NITHISH KUMAR R	(CB.EN.U4AIE20044)

For the completion of

21AIE211 - Introduction to Computer Networks

CSE - AI

18th July 2022

Introduction:

Raspberry Pi RFID Attendance System:

In this project, we will be designing a RFID based Attendance System using Raspberry Pi and the information read/received by the RFID Reader will be authenticated against our database by using socket programming (TCP protocol).

Hardware Requirements:

- Raspberry Pi
- Micro SD Card
- Power Supply
- Ethernet Cable or Wi-Fi
- RC522 RFID Reader
 1. The RC522 RFID reader module is designed to create a 13.56MHz electromagnetic field and communicate with RFID tags (ISO 14443A standard tags).
 2. The reader can communicate with a microcontroller over a 4-pin SPI with a maximum data rate of 10 Mbps. It also supports communication over I2C and UART protocols.
 3. The RC522 RFID module can be programmed to generate an interrupt, allowing the module to alert us when a tag approaches it, instead of constantly asking the module "Is there a card nearby?".
 4. The module's operating voltage ranges from 2.5 to 3.3V, but the good news is that the logic pins are 5-volt tolerant, so we can easily connect it to an Arduino or any 5V logic microcontroller without using a logic level converter.

Technical Specifications

Here are the specifications:

Frequency Range	13.56 MHz ISM Band
Host Interface	SPI / I2C / UART
Operating Supply Voltage	2.5 V to 3.3 V
Max. Operating Current	13-26mA
Min. Current(Power down)	10µA

Logic Inputs

5V Tolerant

Read Range

5 cm

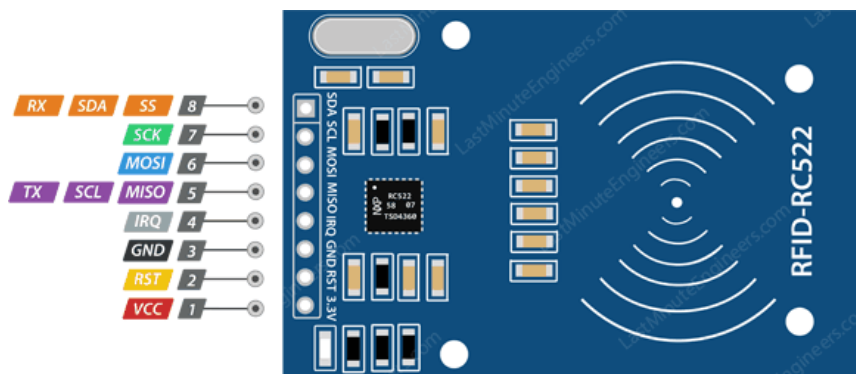
- IR Sensor
- Servo Motor

Implementation:

Preparing Raspbian for your RFID Attendance System

1. To start, we will first ensure that everything is up to date on our Raspbian installation by running the following two commands on the Raspberry Pi.
 - `sudo apt-get update`
 - `sudo apt-get upgrade`
2. We will now install all the packages that we will be relying on for the next few sections. Let's begin by installing build-essential, git, python3-dev, python3-pip, and python3-smbus by running the command below.
 - `sudo apt-get install build-essential git python3-dev python3-pip python3-smbus`

Interfacing RFID RC522 Reader Circuit to Raspberry Pi



1. For wiring the RFID RC522 to the circuit, you will require the following pieces of equipment ready.
 - 1 piece of Male to Male Breadboard Wire
 - 6 pieces of Male to Female Breadboard Wire
 - RFID RC522 Read/Writer
2. Wiring the RFID circuit up to the Raspberry Pi.

VCC - supplies power to the module. This can be anywhere from 2.5 to 3.3 volts. But remember that connecting it to the 5V pin will probably destroy your module!

RST - is an input for reset and power-down. When this pin goes low the module enters power-down mode. In which the oscillator is turned off and the input pins are disconnected from the outside world. Whereas the module is reset on the rising edge of the signal.

GND - is the ground pin and needs to be connected to the GND pin on the Arduino.

IRQ - is an interrupt pin that alerts the microcontroller when an RFID tag is in the vicinity.

MISO - pin acts as master-in-slave-out when SPI interface is enabled, as serial clock when I2C interface is enabled and as serial data output when the UART interface is enabled.

MOSI - is the SPI input to the RC522 module.

SCK - accepts the clock pulses provided by the SPI bus master.

SDA - pin acts as a signal input when the SPI interface is enabled, as serial data when the I2C interface is enabled and as a serial data input when the UART interface is enabled. This pin is usually marked by encasing the pin in a square so that it can be used as a reference to identify other pins.

- SDA connects to GPIO8 (Physical Pin 24)
- SCK connects to GPIO11 (Physical Pin 23)
- MOSI connects to GPIO10 (Physical Pin 19)
- MISO connects to GPIO9 (Physical Pin 21)
- GND connects to Physical Pin 6.
- RST connects to GPIO25 (Physical Pin 22)
- 3.3v connects to 3v3 (Physical Pin 1)

Enabling the SPI interface

1. With the RFID now wired to our Raspberry Pi, we will need to go into the raspi-config tool to enable the SPI interface. This interface is required so that we can communicate with the RC522 module. To do this, you need to first launch the raspi-config tool by running the following command.
 - **sudo raspi-config**
2. Upon running the command, you will see with a screen showing various options that you can configure. For now, we are only interested in activating the SPI interface. If you want to learn more, you can check out our ultimate guide to the Raspi-Config tool. On this screen use your ARROW keys to go down and select “5 Interfacing Options” and press ENTER .
3. On the next screen, you will want to use your ARROW keys to select the “P4 SPI” option, once selected press ENTER . sudo raspi-config Terminal \$ ⊕ Copy
4. You will now need to confirm if you want to enable the SPI Interface. To this, you will want to use your ARROW keys to select “Yes” and then press ENTER once it's selected.

5. The SPI Interface should now be successfully enabled, and you should now see the text “The SPI interface is enabled” appear on the screen. Now before the SPI interface is fully enabled, we will need to restart the Raspberry Pi. We can achieve this by going back to the terminal by pressing ENTER and then ESC . Enter the following command to restart the Raspberry Pi.

- **`sudo reboot`**

6. Once the Raspberry Pi has finished rebooting, you can verify that the SPI interface has been enabled by running the following command. This command will retrieve the list of enabled kernel modules and grab anything from that list that contains the text “spi “. If you see the text “spi_bcm2835” appear in the command line, then you are now ready to proceed to test that the circuit is working correctly. Once that is done we can set up our RFID powered attendance system. If it doesn’t appear, then we recommend you check out our guide on setting up the RFID RC522 for other methods of enabling the correct kernel module.

- **`lsmod | grep spi`**

Preparing the RFID Attendance System Database

Begin by installing MYSQL to your Raspberry Pi by running the following command on your Pi.

- **`sudo apt-get install mysql-server -y`**

Next, we will need to run the “secure installation” script that comes packaged with MYSQL. This script will run you through some processes on making your MYSQL server more secure.

Run this script by running the following command within the terminal on the Raspberry Pi.

- **`sudo mysql_secure_installation`**

When prompted make sure that you set a new password for the root MYSQL server. Additionally, you should answer “Y” to most prompts such as disabling root login access to your MYSQL server.

Now let’s load up into the MYSQL command-line tool by running the following command. You will be prompted to enter the password you set in the previous step.

As MariaDB when installed utilizes “UNIX_SOCKET” as the authentication method by default, we are required to log in using the superuser, do this by using “sudo”.

- **`sudo mysql -u root -p`**

Let’s begin by creating a database where we will be storing all of the data that we will be utilizing for our RFID attendance system.

We will be naming this database, “attendancesystem”. To create this database, run the following command.

- `CREATE DATABASE attendancesystem;`

With our database created, let's now create a user called "attendanceadmin" we will utilize this user in our Python scripts to read from our newly created database.

Make sure you set the password for this to something unique and hard to guess. For our example, we will be just using "pimylifeup" as the password.

- `CREATE USER '<username>'@'localhost' IDENTIFIED BY '<password>';`

Now that we have created our user we need to give it the rights to access our "attendancesystem" database.

We can do this by running the following command. This command will give our "attendanceadmin" user full privileges on any table within our database.

- `GRANT ALL PRIVILEGES ON attendancesystem.* TO '<username>'@'localhost';`

Before we create our tables, we need to utilize the "use" command so that we are directly interacting with the "attendancesystem" database.

Begin interacting with the database by running the following command.

- `use attendancesystem;`

Now that we are dealing directly with the database that we want to utilize we can now start creating the tables where all our data will be stored.

Running the following two commands will create the tables that we will rely on for storing data. We will explain the fields in these tables after we have created them.

```
create table attendance(
    id INT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE,
    user_id INT UNSIGNED NOT NULL,
    clock_in TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY ( id )
);
```

```
create table users(
    id INT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE,
    rfid_uid VARCHAR(255) NOT NULL,
```

```
name VARCHAR(255) NOT NULL,  
created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,  
PRIMARY KEY ( id )  
);
```

Theory:

Socket Programming:

A *socket* is a communications connection point (endpoint) that you can name and address in a network. Socket programming shows how to use socket APIs to establish communication links between remote and local processes.

The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data.

TCP:

Transmission Control Protocol (TCP) is connection-oriented, meaning once a connection has been established, data can be transmitted in two directions. TCP has built-in systems to check for errors and to guarantee data will be delivered in the order it was sent, making it the perfect protocol for transferring information like still images, data files, and web pages.

But while TCP is instinctively reliable, its feedback mechanisms also result in a larger overhead, translating to greater use of the available bandwidth on your network.

Why TCP over UDP?

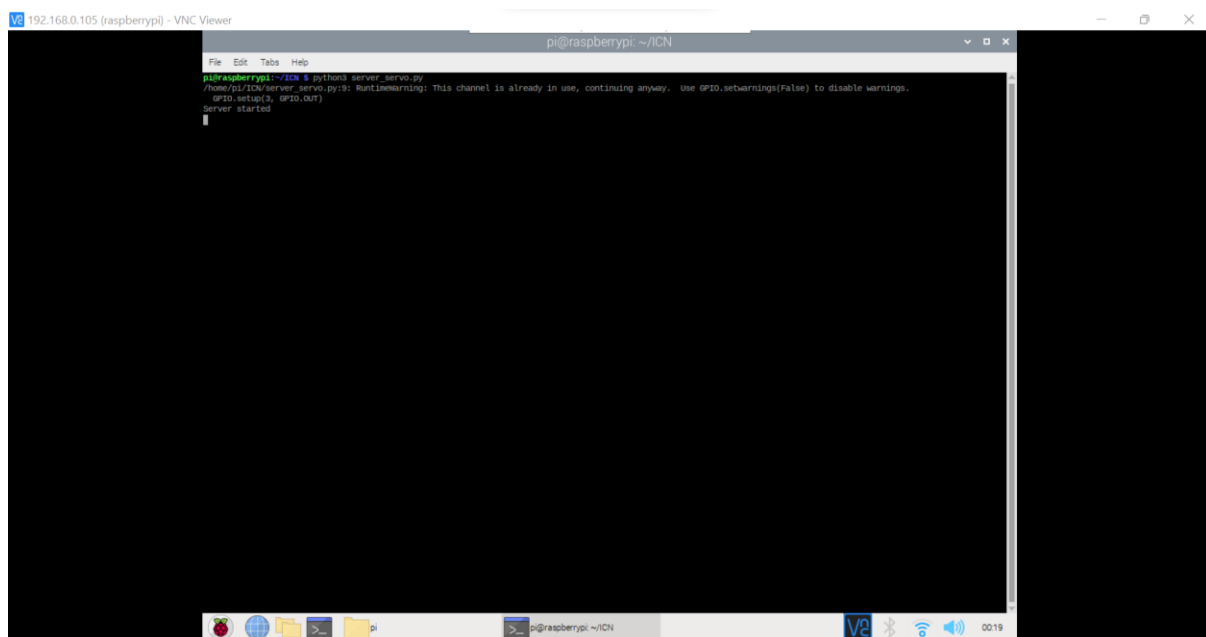
- **Is reliable:** Packets dropped in the network are detected and retransmitted by the sender.
- **Has in-order data delivery:** Data is read by your application in the order it was written by the sender.

Working:

- Firstly, we will be having a IR Sensor which will be monitoring the entry of the object and if the object is detected RFID Reader detects for the incoming RFID card.
- We have then used socket programming (TCP protocol) to send information that is read/received from the RFID Reader through client interface (client .py - in which the client interacts) to server interface.
- Then the information that is received on the server interface (server.py) is authenticated against our database we created for the users.

- Once the information has been authenticated by the server, door/gate opens up which is demonstrated by using a Servo motor and acknowledgement will be shown on the client interface.
- Again we will be monitoring the exit of the object with another IR Sensor, when it detects the object it will consider it as exiting and closes the door/gate.

Result:



```
192.168.0.105 (raspberrypi) - VNC Viewer
pi@raspberrypi: ~/ICN
File Edit Tabs Help
pi@raspberrypi:~/ICN $ python3 server_servo.py
/home/pi/ICN/server_servo.py:9: RuntimeWarning: This channel is already in use, continuing anyway. Use GPID.setwarnings(False) to disable warnings.
  GPID.setup(5, GPID.OUT)
Server started
```

Fig 5.1 Getting server running

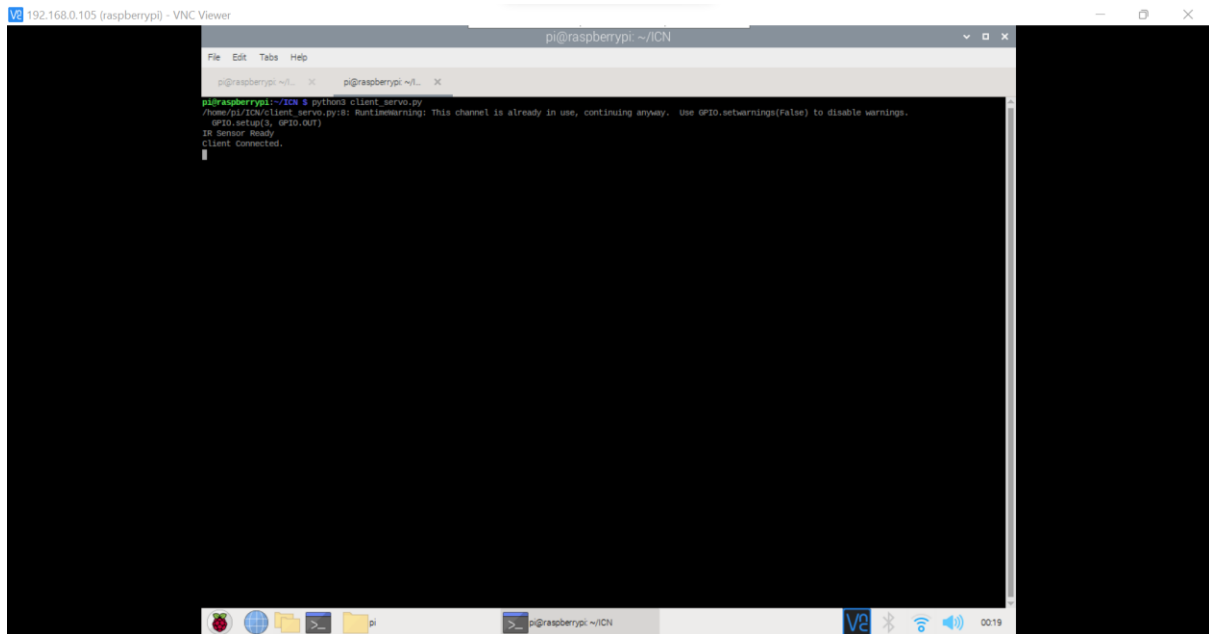


Fig 5.2 Client interface

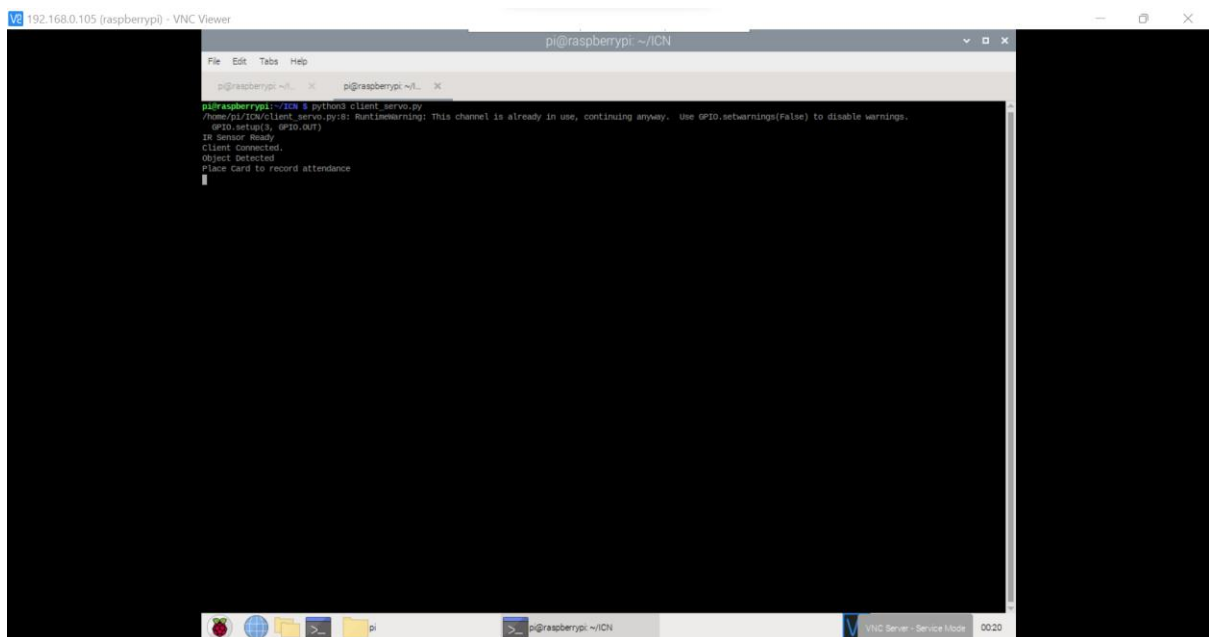
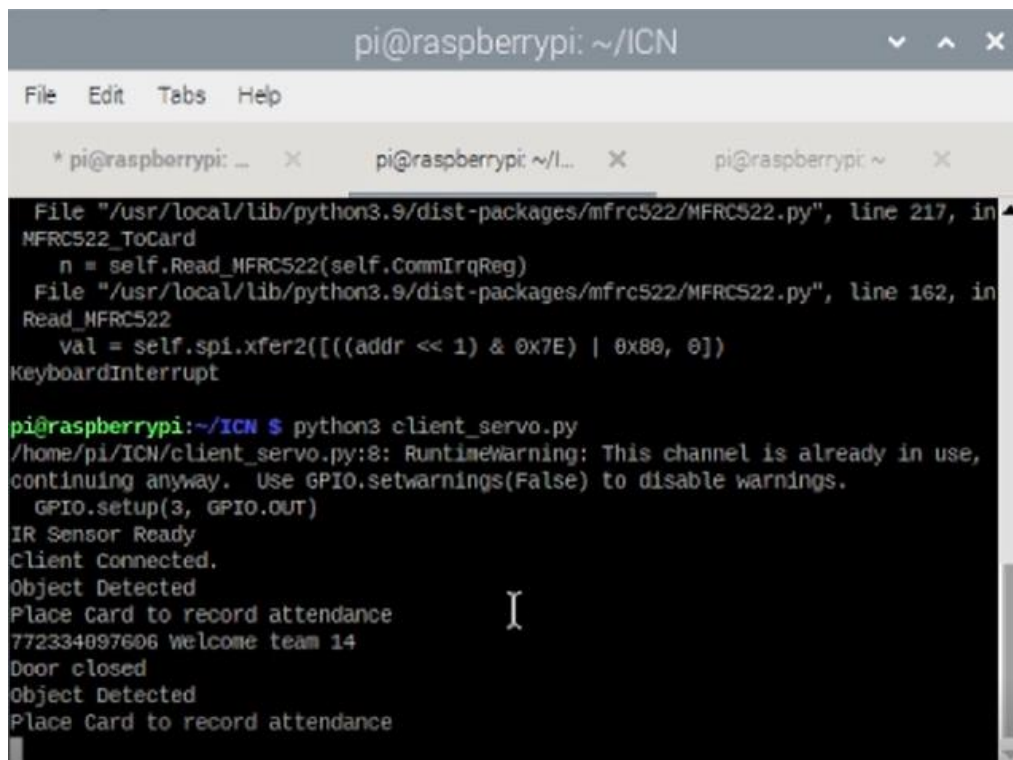


Fig 5.3 Object detected by IR Sensor and prompting RFID input



The screenshot shows a terminal window titled 'pi@raspberrypi: ~/ICN'. The window contains the following text:

```
File "/usr/local/lib/python3.9/dist-packages/mfrc522/MFRC522.py", line 217, in MFRC522_ToCard
    n = self.Read_MFRC522(self.CommIrqReg)
File "/usr/local/lib/python3.9/dist-packages/mfrc522/MFRC522.py", line 162, in Read_MFRC522
    val = self.spi.xfer2([((addr << 1) & 0x7E) | 0x80, 0])
KeyboardInterrupt

pi@raspberrypi:~/ICN $ python3 client_servo.py
/home/pi/ICN/client_servo.py:8: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(3, GPIO.OUT)
IR Sensor Ready
Client Connected.
Object Detected
Place Card to record attendance
772334897606 welcome team 14
Door closed
Object Detected
Place Card to record attendance
```

Fig 5.4 RFID tag door opened

Code

server.py

```
import socket
```

```

import threading
import time
import mysql.connector
import time
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)
GPIO.setup(3, GPIO.OUT)
pwm=GPIO.PWM(3, 50)
pwm.start(0)

db = mysql.connector.connect(
    host="localhost",
    user="admin",
    passwd="admin",
    database="experiment"
)

cursor = db.cursor()

def SetAngle(angle):
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(3, GPIO.OUT)
    duty = angle / 18 + 2
    GPIO.output(3, True)
    pwm.ChangeDutyCycle(duty)
    time.sleep(1)
    GPIO.output(3, False)
    pwm.ChangeDutyCycle(0)

def function(conn, addr):
    try:
        while True:
            message = conn.recv(1024).decode()
            cursor.execute("Select id, name FROM users WHERE rfid_uid="+
message)
            result = cursor.fetchone()

            if cursor.rowcount >= 1:
                print("Authenticated")
                print("Welcome " + result[1])
                SetAngle(90)
                print("Gate Opened")
                message = (message + " Welcome " + result[1])

                cursor.execute("INSERT INTO attendance (user_id) VALUES (%s)",
(result[0],) )
                db.commit()

```

```

        conn.send(message.encode())

    else:
        message = "User does not exist"
        conn.send(message.encode())
    conn.close()
except Exception as e:
    conn.close()

s = socket.socket()

port = 9696

s.bind(('', port))
s.listen(5)
print("Server started")

while True:
    conn, addr = s.accept()
    threading.Thread(target=function, args=(conn, addr)).start()

s.close()

```

client.py

```

import socket
import threading
import time
import RPi.GPIO as GPIO
from mfrc522 import SimpleMFRC522

GPIO.setmode(GPIO.BOARD)
GPIO.setup(3, GPIO.OUT)
pwm=GPIO.PWM(3, 50)
pwm.start(0)

def SetAngle(angle):
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(3, GPIO.OUT)
    duty = angle / 18 + 2
    GPIO.output(3, True)
    pwm.ChangeDutyCycle(duty)
    time.sleep(1)
    GPIO.output(3, False)
    pwm.ChangeDutyCycle(0)

```

```

ir1_sensor = 16
ir2_sensor = 40

GPIO.setmode(GPIO.BOARD)
GPIO.setup(ir1_sensor,GPIO.IN)
GPIO.setup(ir2_sensor,GPIO.IN)

print ("IR Sensor Ready")

reader = SimpleMFRC522()

port = 9696
s = socket.socket()
s.connect(('', port))
print("Client Connected.")

try:
    while True:

        ir1_state = GPIO.input(ir1_sensor)
        if (ir1_state == 0):
            print ("Object Detected")
            #RFID code
            print('Place Card to record attendance')
            id, text = reader.read()

            message = str(id)

            s.send(message.encode())
            message = s.recv(1024).decode()
            print(message)
            time.sleep(2)
            ir2_state = GPIO.input(ir2_sensor)
            if (ir2_state == 0):
                SetAngle(0)
                print("Door closed")
            else:
                print("Door not closed")
        else:
            s.close

except KeyboardInterrupt:
    GPIO.cleanup()

```