Project Description

The aim of this project is to turn Raspberry Pi 3 into a subsystem to be used in home automation. The sensors and drivers needed in home automation will be directly connected to the Raspberry Pi 3. This system will work as a server module, providing serial and digital I/O access to high-level applications over the network. The Project details are presented below.

Step 1: Hardware Design and Implementation

Parts to be used in the design:

- Raspberry Pi 3
- One sensor(GPIO{DC5V, GND, 17}),
- A relay (GPIO{DC5V, GND, 27}),
- A keypad (GPIO{16,20,21,12,06,13,19,26}),
- A sensor modulus (MPU9250) (GPIO{DC3.3V, GND, 02, 03}),
- Ethernet communication interface

Step 2: Software Design and Implementation

At this stage, it is expected to develop different software that will run on both Raspberry Pi and PC. The first of these is the server software that will run on the Raspberry Pi. The second software is the application software that will run on the PC side that will access the Raspberry Pi over Ethernet.

Below is a visual representation of the expected software architecture. The server-side of the software that'll run on the Raspbery Pi will use 3 processes: GyroSensor Node (127.0.0.1:7003), DigitalIO Node (127.0.0.1:7002) and Server Node (>RaspberryPiIP>:7001). And the client-side of the software that'll run on the PC has only one process: Client Node, which will communicate with the server-code running on Raspberry Pi through Ethernet (Socket, TCP/IP at port 7001).

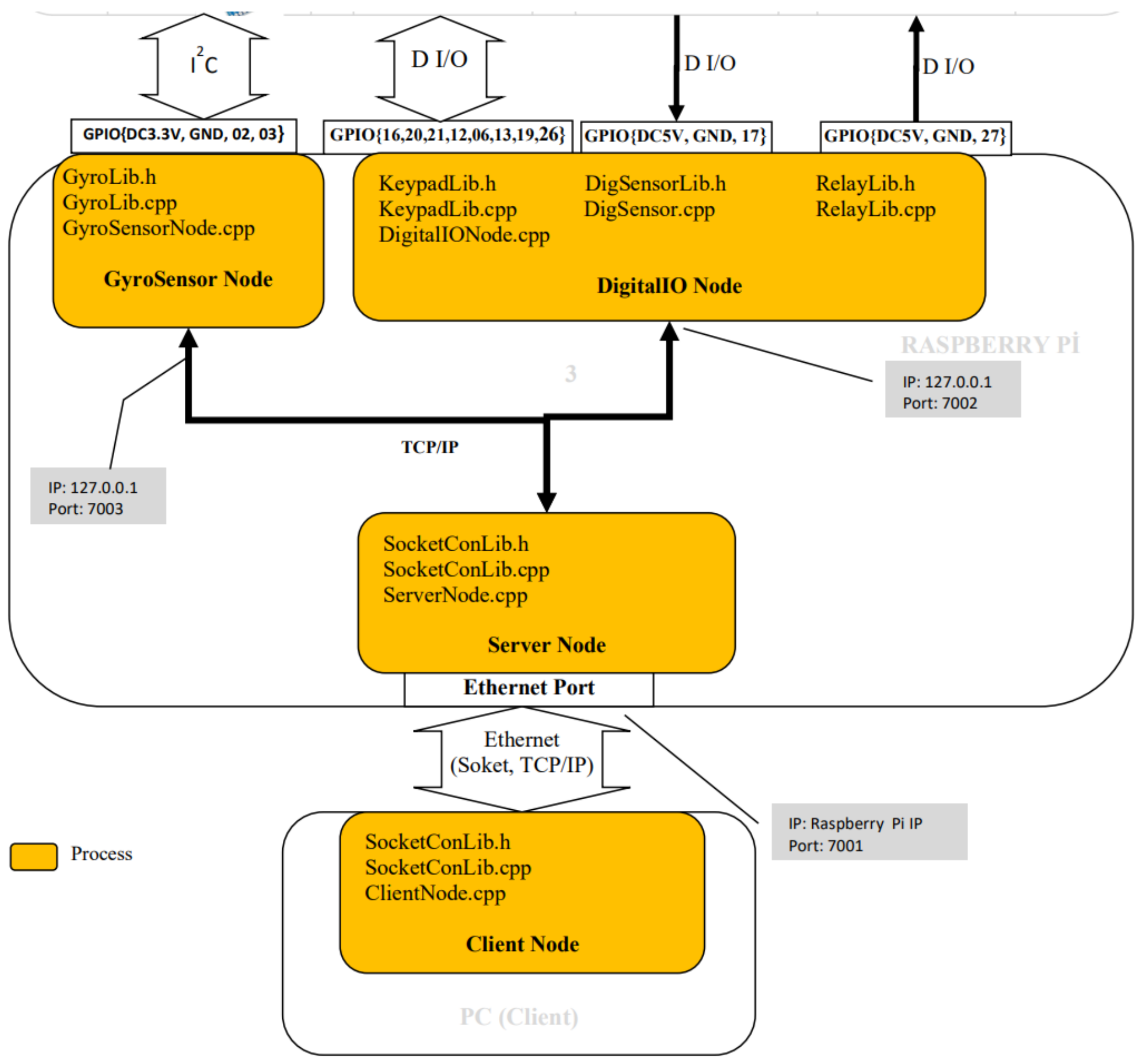

Figure 1 - Software Architecture

Step 2.1: Creating the Libraries

i. Write a gyro connection class. (GyroLib.h & GyroLib.cpp)

Gyro	
+ init(): void	Prepare the sensor to get data
+ getGyroX(): double	Return Gyro X value (deg/s)
+ getGyroY(): double	Return Gyro Y value (deg/s)
+ getGyroZ(): double	Return Gyro Z value (deg/s)
+ getAccX(): double	Return Acceleration X value (m/s^2)
+ getAccY(): double	Return Acceleration Y value (m/s^2)
+ getAccZ(): double	Return Acceleration Z value (m/s^2)
+ getTemp()	Return the Temperature (Degree Celcius)
Expand if needed	

ii. Write a keypad management class. (KeypadLib.h & KeypadLib.cpp)

Keypad	
+ init(): void	Prepare the keypad
+ release()	Ends the keypad usage
+ getKey(): char	Returns the pressed key
Expand if needed	

iii. Write a class for digital sensor management. (DigSensorLib.h & DigSensorLib.cpp)

DigSensor	
+ init(): void	Prepare the sensor
+ release()	Ends the sensor usage
+ read(): bool	Get the sensor value
Expand if needed	

iv. Write a class for relay management. (RelayLib.h & RelayLib.cpp)

Relay	
+ init(): void	Prepare the relay
+ release()	Ends the relay usage
+ set(bool)	Set relay state
Expand if needed	

v. Write a class for socket connection management. (SocketConLib.h & SocketConLib.cpp)

SocketCon	
+ init(): void	Creates the socket connection. If it will run as Server, it listens for a connection and creates the connection with the incoming request. If it will run as a client, it makes a connection request and creates the connection. At this stage, it is expected to develop different software that will run on both Raspberry Pi and PC. The first software is the server software that will run on the Raspberry Pi. The second software is the application software that will run on the PC side that will access the Raspberry Pi over Ethernet.
+ release()	Ends the connection.

+ send(string)	Sends the character string over the established connection.
+ receive(string&)	Retrieves the character string over the established connection.
Expand if needed	

Step 2.2: Writing the code that implements the processes

i. DigitalIO Node (DigitalIONode.cpp)

This process works as a server and provides keypad, digital sensor and driver control according to incoming commands. It performs these operations according to the commands coming through the socket.

- It will be listening on Port 7002. When the connection request is received, the connection will be established and operations will be performed according to the commands received over the connection. The commands it accepts are given in Table 1.
- When the sensor status is requested, the value on the relevant digital pin will be read and sent.
- When an on or off request for the drive is received, the status of the drive will be changed accordingly.
- When a key request is received, the last key entered will be sent.

Table 1 - The commands accepted by the DigitalIO Node

Message Received by Server	Response Message	Action
sensorState:	sensorState <state>:	Package and send the current sensor state
sensorType:	sensorType <sensor type>:	Sends the sensor type
relay <0 or 1>:	relay ok: or relay err:	Relay's state will be set accordingly. Result will be returned.
relayState:	relay <0 or 1>:	Returns the current state of the relay
close:	close ok: or close err:	Once the Message is received by the server, the connection will be closed after the reply message is sent.
key:	key <KEY>:	Returns the key value entered on the keypad. Key is the string of characters entered until the "#" key is pressed on the keypad.
EXPLANATIONS		

<state> unsigned int or bool. Gets the value of either 0 or 1. <sensor type> string (char array). Gets its value as type name. For example: for temperature sensor it will be returned as “TEMPERATURE” or any other explanatory name of choosing. <0 or 1> unsigned int or bool. Gets the value of either 0 or 1.

ii. GyroSensor Node (GyroSensorNode.cpp)

This process runs as a GyroSensor Node server.

- Listening will be done at GyroSensor Node Port 7003. When the connection request is received, the connection will be established and operations will be performed according to the commands received over the connection.
- Here, the GyroSensor Node will work as a server. It will execute operations according to the messages received via the socket. The messages are given in Table 2.
- According to the incoming message, the requested information will be requested via MPU9250 and the received information will be returned via the socket.

Table 2 - The commands accepted by the GyroSensor Node

Message Received by Server	Response Message	Action
temp:	temp <Temperature Value>:	Returns the temperature value
gyro:	gyro <X> <Y> <Z>:	Returns the gyro value
acc:	acc <X> <Y> <Z>:	Returns the acceleration value
close:	close ok: or close err:	Once the Message is received by the server, the connection will be closed after the reply message is sent.

iii. Server Node (ServerNode.cpp)

This process is expected to do the following:

- It will connect to the GyroSensor Node and DigitalIO Node processes through the port they are listening on.
- It will start listening on port 7001. When a connection request is received, the connection will be established.
- It will wait for the messages given in Table 1 and Table 2 to arrive on port 7001.

- If the messages given in Table 1 are received, it will send these messages to the DigitalIO Node process via the socket it is connected to, and send the response from this process to the connection established on port 7001.
- If the messages given in Table 2 are received, it will send these messages to the GyroSensor Node process via the socket it is connected to, and send the response from this process to the connection established on port 7001.
- As can be seen, the message “close” is among the messages to be sent to both processes. In this case, this message will be sent to both processes.

iv. Client Node (ClientNode.cpp)

This program will run on the PC connected to the Raspberry pi via Ethernet. This process will perform the following functions:

- It will establish a connection with the Server Node on port 7001.
- The application will present a text-based menu. In this menu, the user will be able to use the desired function.
    - When the user wants to see the sensor status from the menu, a similar output will come according to the information received from the server (Server Node)
        - Temperature Sensor: ON
    - From the menu, the user will be able to change the drive status. When this option is requested we will see a similar output.
        - Driver Status: ON
        - Press 0 to switch it OFF.

          or
        - Driver Status: OFF
        - Press 1 to switch it ON.
    - The user will be able to select automatic control from the menu. Here, the drive control will be done automatically according to the user's preferred sensor state. When the user wants to close this function, the control will end. For example:
        - Sensor List:
        - Type: TEMPERATURE
        - In which state of the sensor should the driver be ON? (1-ON/0-OFF)

        - > 1
        - Right now the control is on. If the sensor is ON, the driver will also be ON. Press “e” to exit…
    - The user will be able to query the Gyro Sensor data by selecting it from the menu. The process will request the information from the Server Node process and display it on the screen. For example when this information is requested from the menu the output will be similar to;
        - Temperature: 25.3 C
        - Gyro: x: -0.0381679 y:0.671756 z:-0.0839695 [deg/sec]
    - After each operation in the menu, the menu list will be displayed on the screen and the user will be expected to select.
    - If the user wants to close the connection, they will be able to choose from the menu.
    - When Key is selected in the menu, the key entered via the keypad will be displayed in the section given as xxx as follows:
        - KEY: xxxxxxxx

Step 3: Package all source files into one package

Create a directory tree of all source files as given below (RCS: Remote Control System).

Create the contents of the CMakeLists.txt files for the build process.

Compile the package and test all menus by running the processes on the corresponding computer.

- RCS/
    - include/
        - GyroLib.h
        - KeypadLib.h
        - DigSensorLib.h
        - RelayLib.h
        - SocketConLib.h
    - src/
        - GyroLib.cpp
        - KeypadLib.cpp
        - DigSensorLib.cpp
        - RelayLib.cpp
        - SocketConLib.cpp

- ClientNode.cpp
- ServerNode.cpp
- GyroSensorNode.cpp
- DigitalIONode.cpp
- CMakeLists.txt