**Flashing ECU Software Over the Air – Project Configuration**

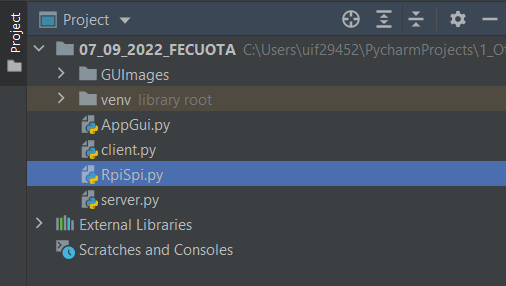
In order to launch the project for the first time, there are some configurations that need to be made.

**1. Pycharm.** The project was mainly developed within Pycharm. The application is made out of 4 script, 2 of which are runnable:

* client.py - runnable
* server.py - runnable
* AppGui.py
* RpiSpi.py

Even though server.py is not meant to be run on the same computer as client.py, it can be done for testing purposes. We’ll include it in our project. For the project to work, besides the 4 python scripts, the folder “GUImages” must also be placed in our project.

After creating the project in pycharm, it should look like in Fig.1.



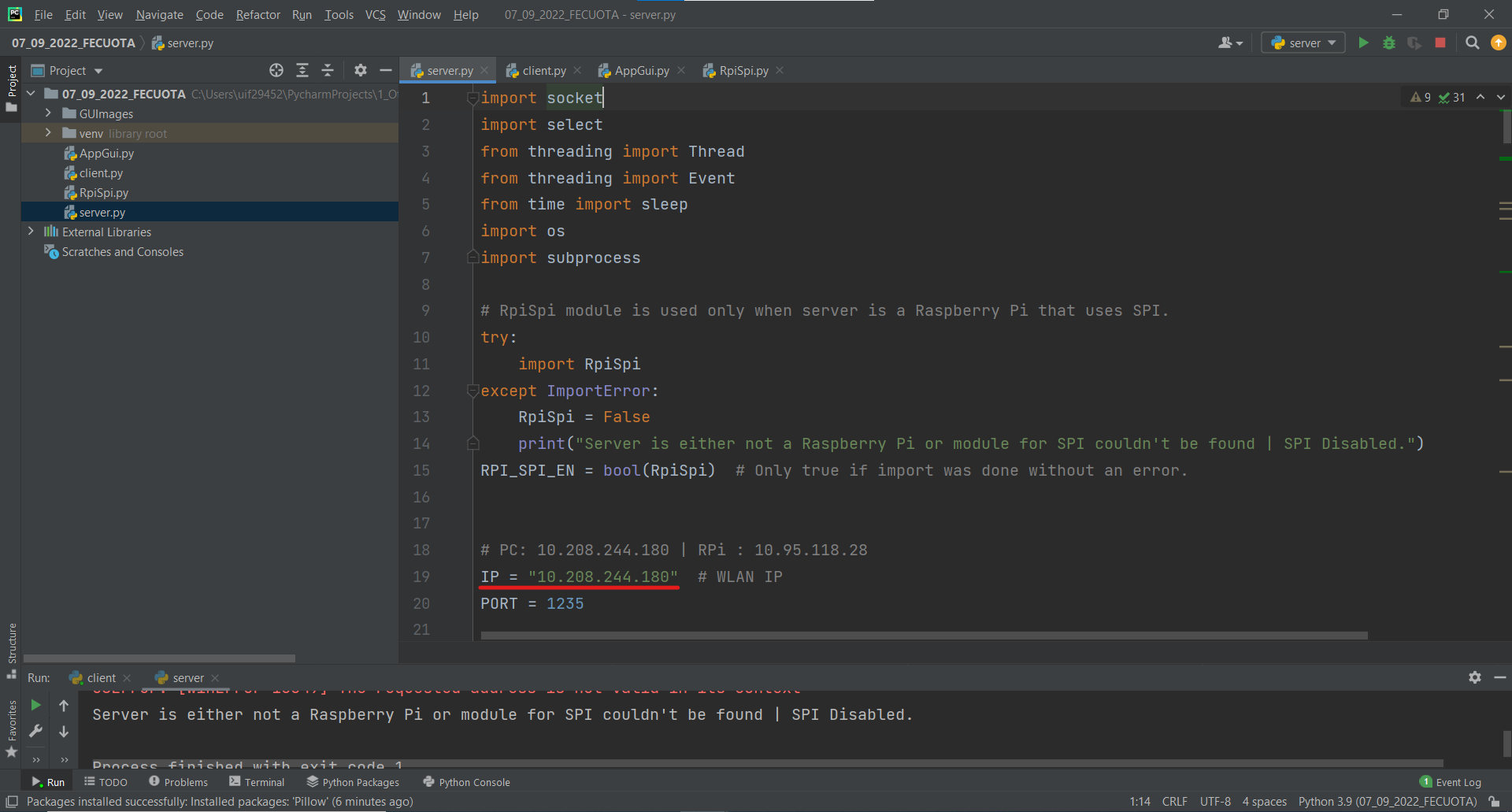
*Fig.1. Pycharm project for the application*

After all files have been placed in our project, we must import the modules used in the project that are not standard like:

* customtkiner
* Pillow

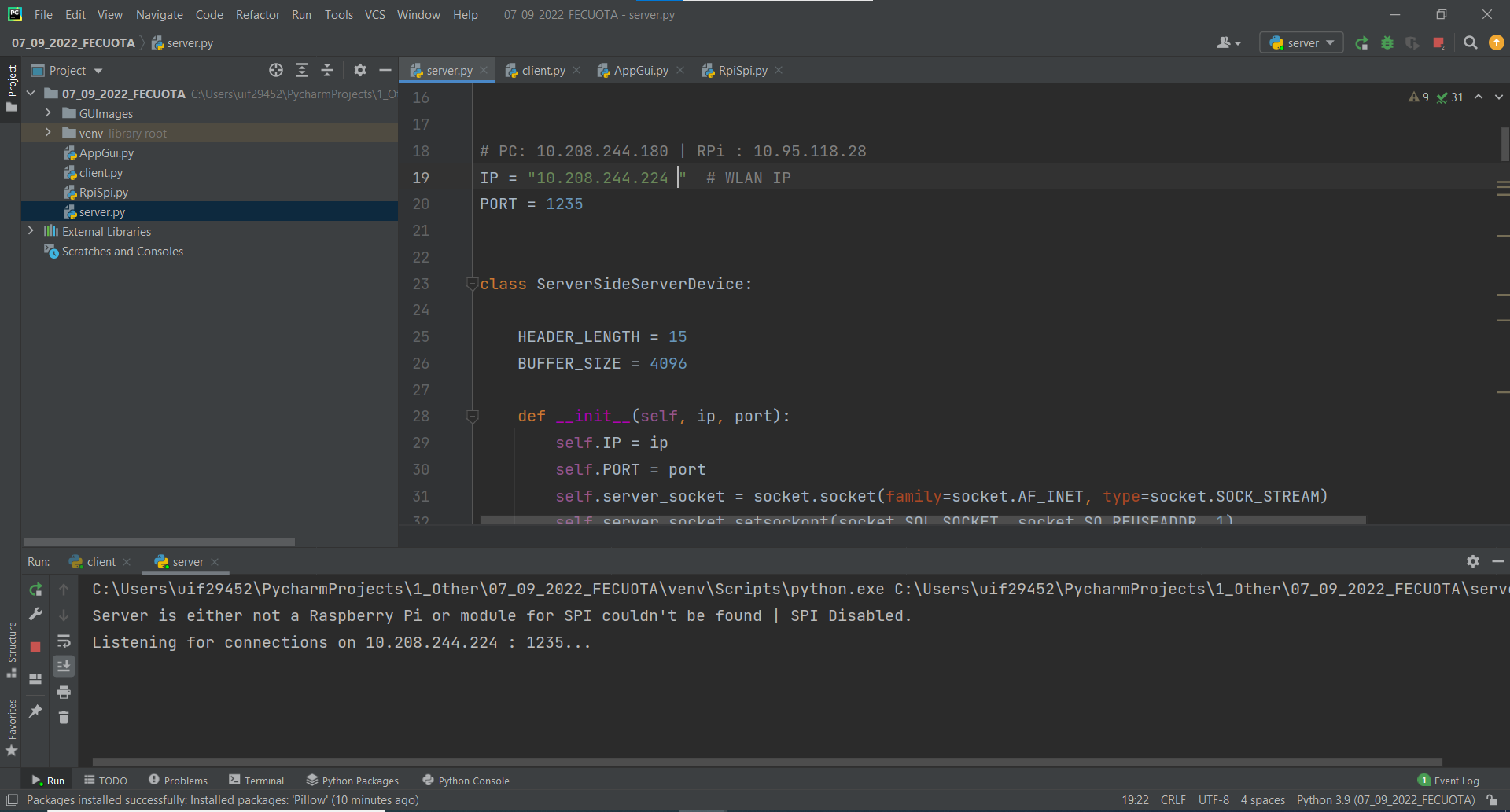
You may observe inside RpiSpi.py that “RPi” module is not imported. “RPi” is standard only on raspberry pi and the “server.py” script in which is used will still work without “RPi”.

In order to successfully run server.py you must configure the IP address(Fig.2.). On windows, get the IP by getting into command prompt and typing “ipconfig”.



*Fig.2. IP Field to be configured for server.py*

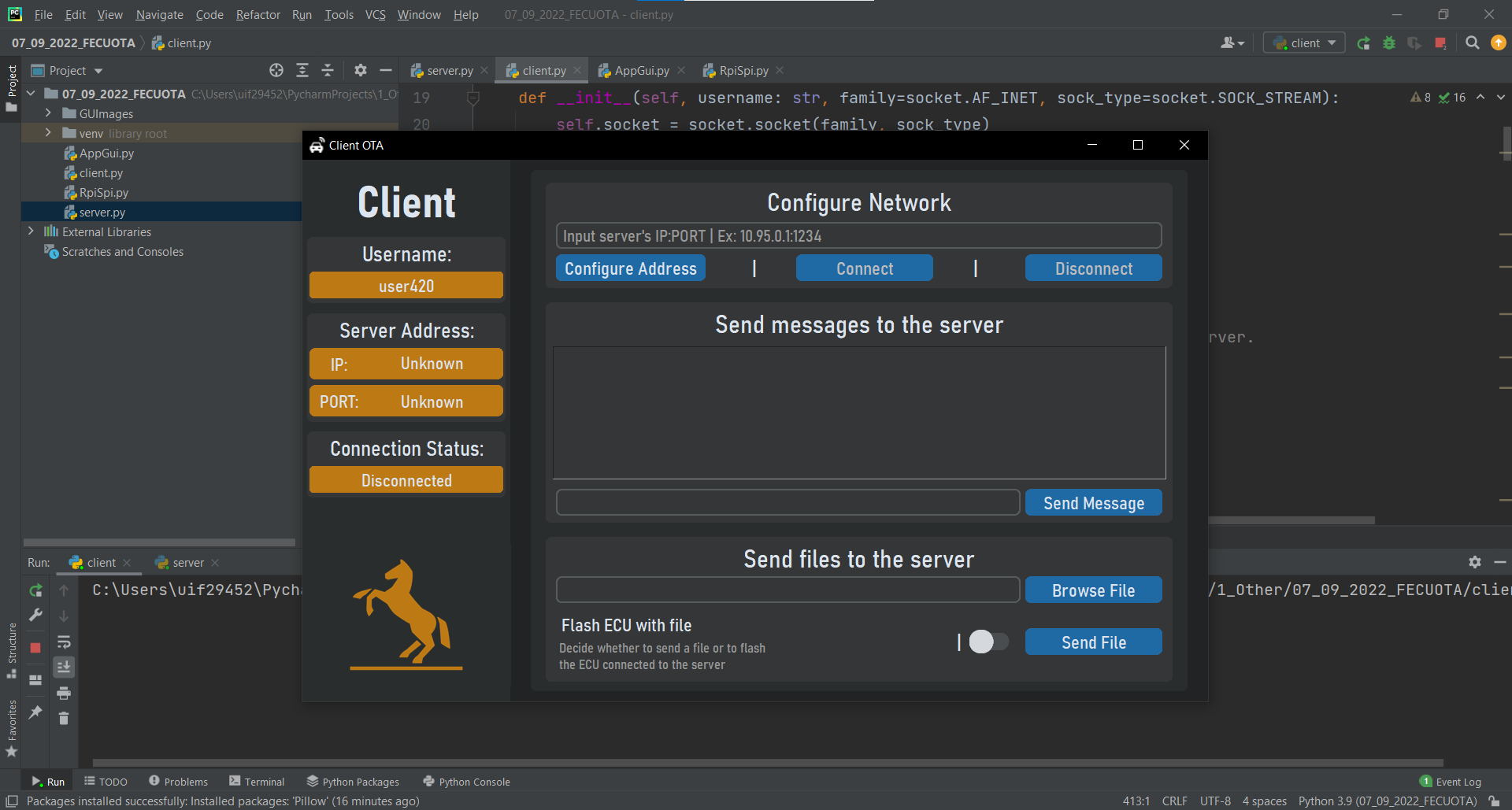
After getting server.py ready, it is now launchable. A successful start of server.py can be seen in Fig.3.



*Fig.3. Successful start of server.py*

There is no configuration to be made for client.py so it should be ready to run.

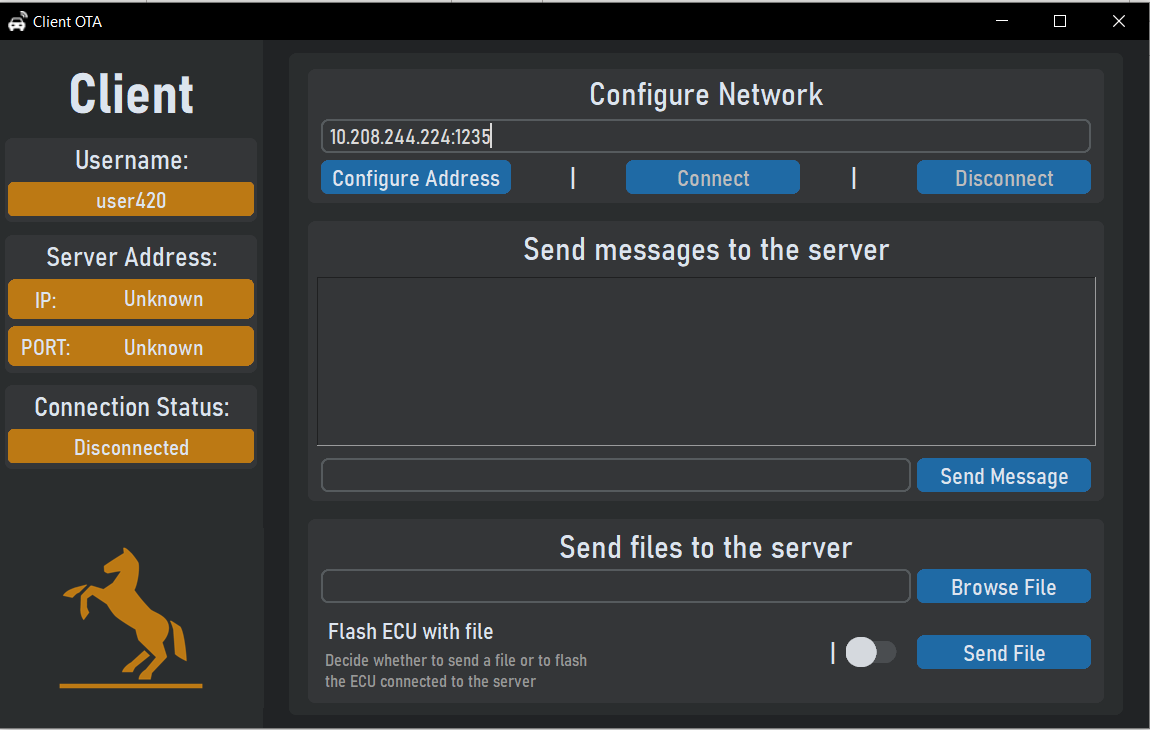
A successful start of client.py can be seen in Fig.4. Compared to server.py, when starting client.py a GUI should pop up with it.



*Fig.4. Successful start of client.py*

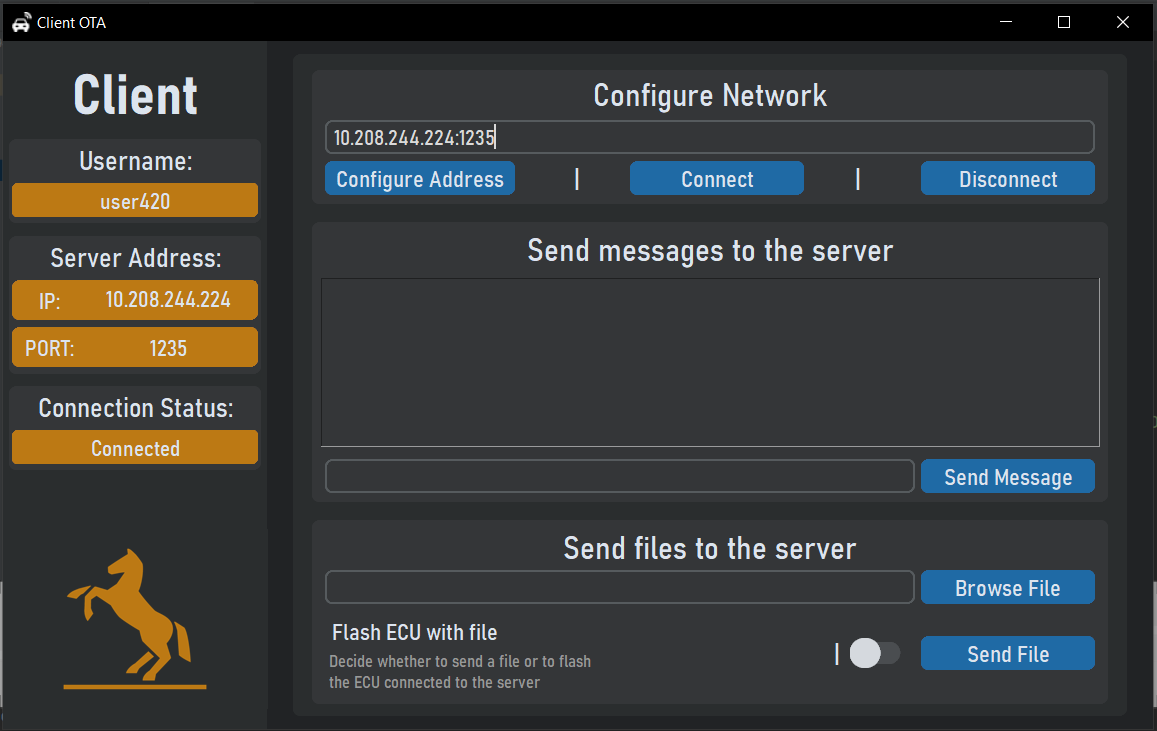
**Testing the application locally**

When testing the application locally, both server.py and client.py are running. Server’s IP address must be entered within the entry in the “Configure Network” menu like in Figure 5.



*Fig.5. Server’s IP enter into GUI*

After server’s address has been entered into the corresponding field, next step is to press “Configure Address” then connect. If everything was successful the fields in the “Client” menu should have changed and should look like in Fig.6.



*Fig.5. GUI after a connection has been established*

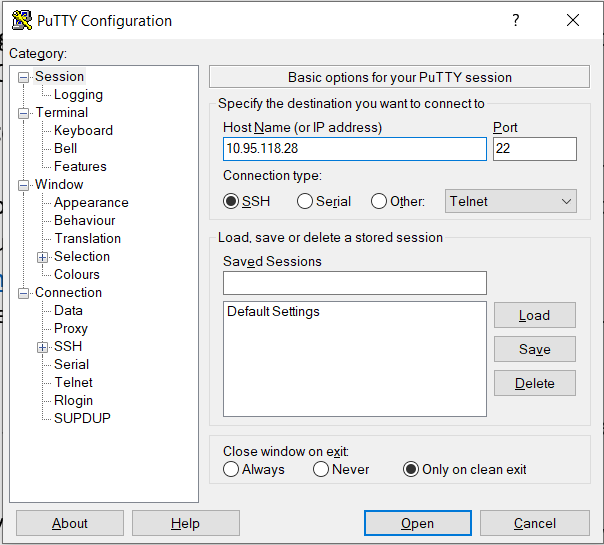
From now on, both messages and files can be sent through the application. We can’t flash anything right now because there’s no ECU connected to our server machine.

**2. Flashing ECU connected to Raspberry Pi via our application**

Configuring the Raspberry Pi. To be able to configure the Rasbperry Pi, we need to access it. For this tutorial we will use Putty because it enables SSH connections. Before getting Putty started make sure the Raspberry is inside your area network.

Get Putty: <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

After launching Putty, enter RPi’s IP address in the corresponding field like in figure 6, then press “Open”.



*Fig.6. Setting up a SSH connection through Putty*

A successful connection will accompanied by a console pop up in which the text “login as:” can be seen (Check fig.7.).



*Fig.6. Successful SSH connection*

Next step is entering the username and password of the user trying to connect. After that, we’ve succeeded in connecting to the RPi.

The ECU we’re going to flash is an ATMEGA328. To flash it, our application will get use of avrdude programming tool.

❗: When making the physical links between Raspberry Pi and ATMEGA328, NEVER use the 5V line. Using it may result in IRREVERSIBLE damage to the Raspberry Pi. ATMEGA328 will work just fine when powered with 3.3V.

The tutorial for configuring avrdude can be found at:

<https://learn.adafruit.com/program-an-avr-or-arduino-using-raspberry-pi-gpio-pins/overview>

In shorthand, the steps that need to be made are:

1. Download necessary dependencies by running following commands in terminal:

sudo apt-get update

sudo apt-get install -y build-essential bison flex automake libelf-dev libusb-1.0-0-dev libusb-dev libftdi-dev libftdi1

1. Download avrdude programming tool and configure linuxgpio. After the configure a text should have popped up in the console saying wether linuxgpio is enabled. It must be.

wget http://download.savannah.gnu.org/releases/avrdude/avrdude-6.1.tar.gz

tar xvfz avrdude-6.1.tar.gz

cd avrdude-6.1

./configure --enable-linuxgpio

1. Install avrdude
2. Verify installation

avrdude -v

make

sudo make install

The next step, after setting up avrdude, is to physically link the Raspberry Pi to the ATMEGA328. The following connections must be made:

* RPi 3V3 – ATMEGA328 VCC
* GND – GND
* GPIO #12 – ICSP RESET
* GPIO #24 – ISCP SCK
* GPIO #23 – ISCP MOSI
* GPIO #18 – ICSP MISO

After finishing the physical layer, configuration of avrdude comes next. If you’ve done the manual install then the config file for avrdude is located in “/usr/loca/etc/avrdude.conf”. Copy this file in the main directory and give it the name “avrdude\_gpio.conf”

cp /usr/local/etc/avrdude.conf ~/avrdude\_gpio.conf

nano ~/avrdude\_gpio.conf

Once in the editor, get all the way down to the very end of the file. You’ll need to add a new section containing the following:

# Linux GPIO configuration for avrdude.

# Change the lines below to the GPIO pins connected to the AVR.

programmer

id = "pi\_1";

desc = "Use the Linux sysfs interface to bitbang GPIO lines";

type = "linuxgpio";

reset = 12;

sck = 24;

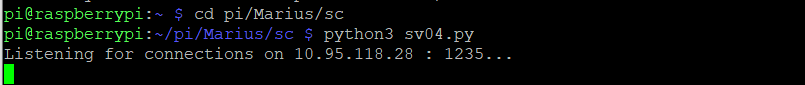
mosi = 23;

miso = 18;

;

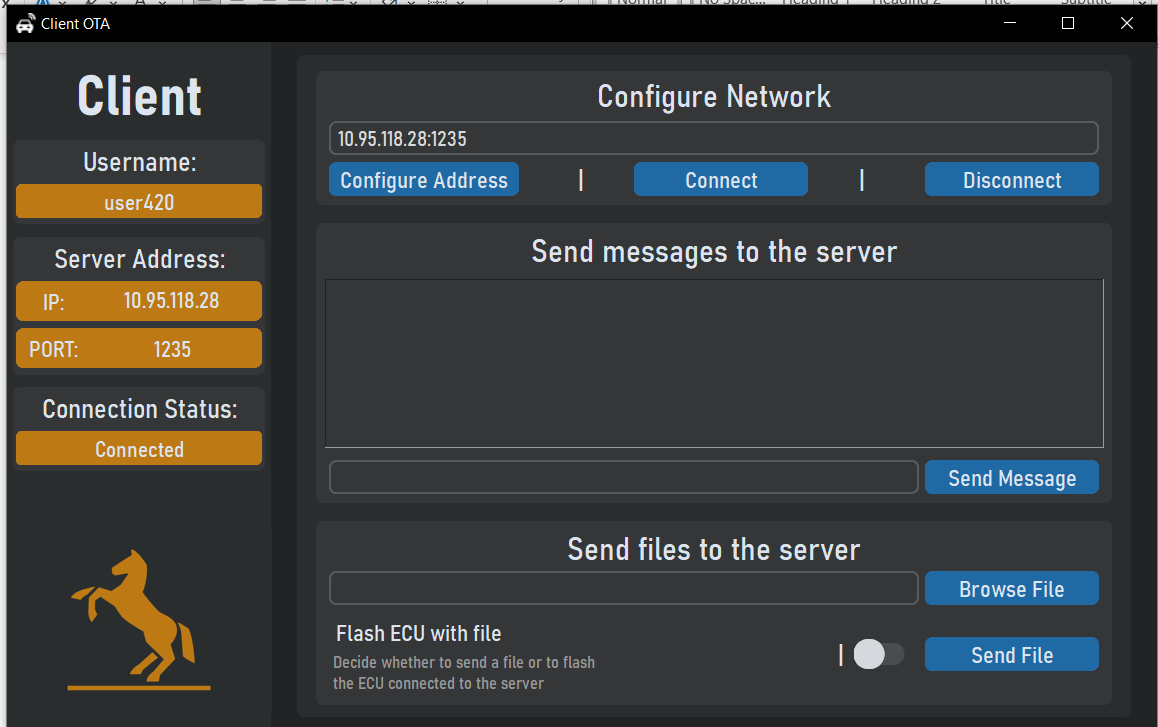
If done, press CTRL+O, enter, CTRL+X. Now avrdude is ready for what we will use it. For more information about avrdude check the [link](https://learn.adafruit.com/program-an-avr-or-arduino-using-raspberry-pi-gpio-pins/overview) given above.

Now that avrdude is all set up, we can launch our server on Raspberry Pi. Copy server.py on the RPi then launch it. After launching the server, the terminal should look like in figure 7.



*Fig.7. Launching server.py on Raspberry Pi*

After executing “server.py”, run “client.py” from your main computer and enter Raspberry’s IP address.



*Fig.8. Connecting the client to Raspberry Pi.*

At this point you should be able to send messages and files to the Raspberry Pi and also to flash the ECU. Our files to flash will be generated by Arduino IDE after configurating it. Further, we will develop an application in Arduino with which to blink a led on ATMEGA’s PB0 pin.

After installing Arduino IDE there is a configuration to be made so it can compile code for ATMEGA.

From <https://www.brennantymrak.com/articles/programming-avr-with-arduino> , those steps must be followed:

* Download the [breadboard-1-6-x configuation here.](https://www.arduino.cc/en/uploads/Tutorial/breadboard-1-6-x.zip)
* Determine the location of your Arduino sketchbook folder by opening the Arduino IDE, going to File, selecting Preferences and then looking at the Sketchbook location.
* Go to the location of your sketchbook folder and create a "hardware" folder if one does not exist already.
* Extract the "breadboard" folder from the "breadboard-1-6-x" zip file and move it into the newly created "hardware" folder. The "hardware" folder should now have the "breadboard" folder in it.
* Restart the Arduino IDE
* In the Arduino IDE, confirm that setup was completed correctly by going to Tools > Board. You should now see "ATmega328 on a breadboard (8 MHz internal clock)" as an option in the boards list.

And then:

* Go to the "hardware" folder we just created when loading the configuration and open the "breadboard" folder, then open the "avr" folder.
* Open the "boards.txt" file
* In the file, find the line that says "atmega328bb.build.f\_cpu=8000000L"
* Change the line to the following "atmega328bb.build.f\_cpu=1000000L"
* Change every instance of “atmega328p” with “atmega328” or any other avr microcontroller you’re using.
* Save and close the file

Now Arduino IDE is ready to compile code.

Our application to blink a led can be found below:

#include <avr/io.h>

#include <util/delay.h>

int main(void){

DDRB = 0b00000001; // PB0 is output

while(1){

PORTB = 0b00000001; // Turns on only PB0

\_delay\_ms(1000); // Delays 1000ms (1s)

PORTB = 0b00000000; // Turns on only PB0

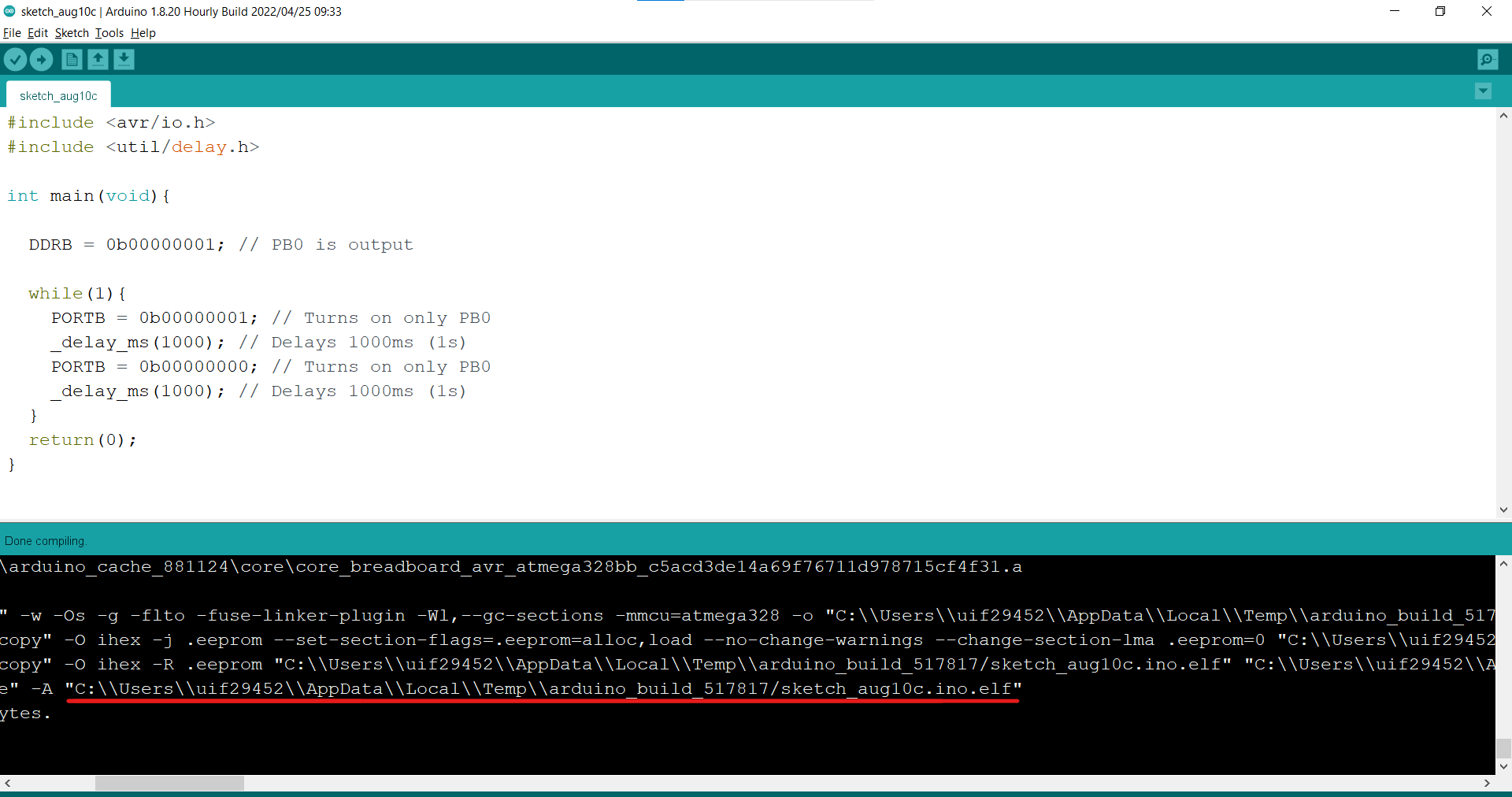
\_delay\_ms(1000); // Delays 1000ms (1s)

}

return(0);

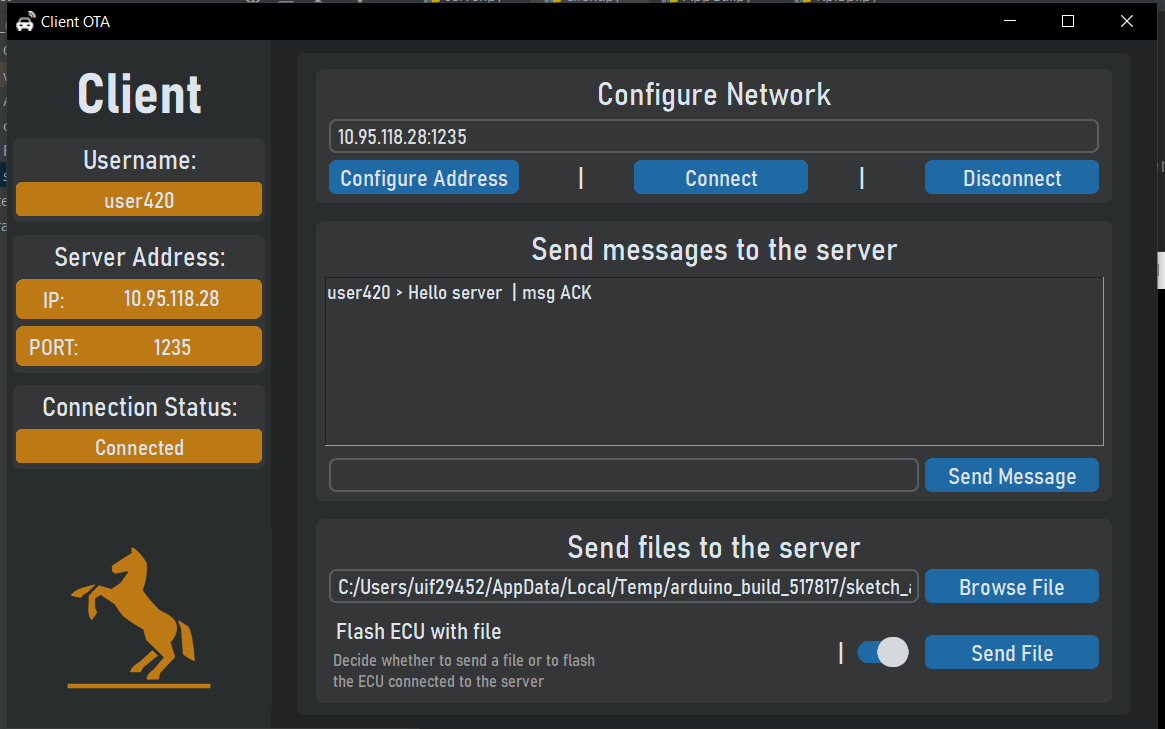
}

After writing the code inside Arduino Ide to find the .hex file generated you’ll need to do the following: Inside File – Preferences, check the boxes right to “Show verbose output during” which are described as “compilation” and “upload”. Compile the code and check the path to .hex file as in figure 9.



*Fig.9. Getting the necessary file to flash on ECU.*

What’s left to do is to browse for that file inside our client application, check to flash the file and watch the magic happen.



*Fig.10. Flashing ECU software OTA*