Reykjavík University

Mechatronics I

T-411-MECH

Individual Assignment 7

14/09/2020

Students:
Giacomo Menchi

Teacher:
Joseph T. Foley

Contents

1 Tasks

As usual, the code for this assignment is available at the following Github repo:

Link: GitHub Repo

1.1 Task 1

Task: Create a Struct that reads the data coming from a GPS sensor and splits it into its base components.

- Have internal parameters for useful data, fx longitude and latitude.
- Have all functions be implemented in the struct.

Screenshots: Below are some screenshots of the Struct and the reading functions (I posted the main ones, the full code can be found inside the repository above).

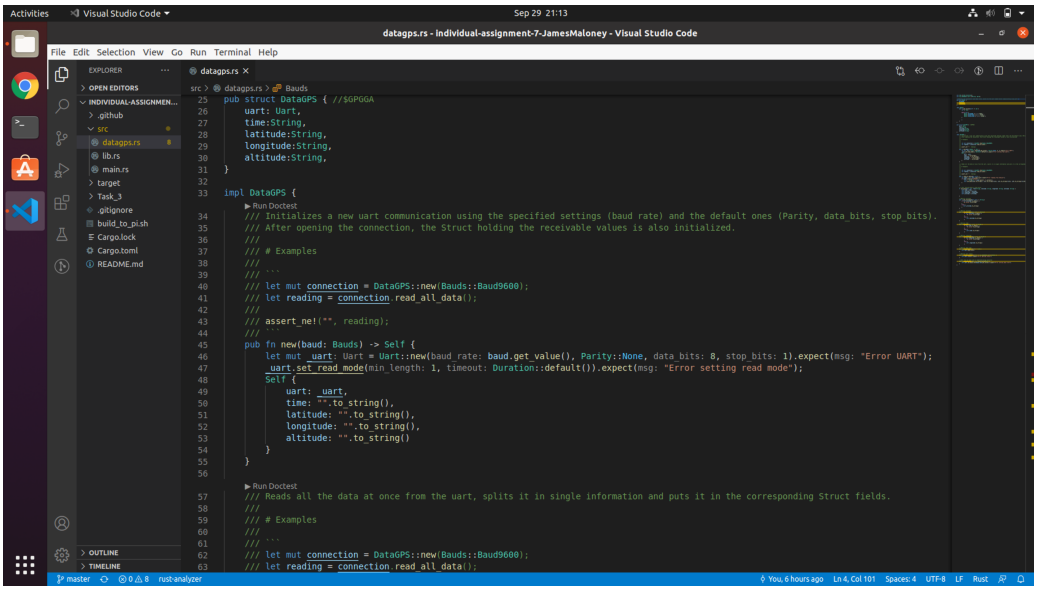

Figure 1: Struct and new() function.

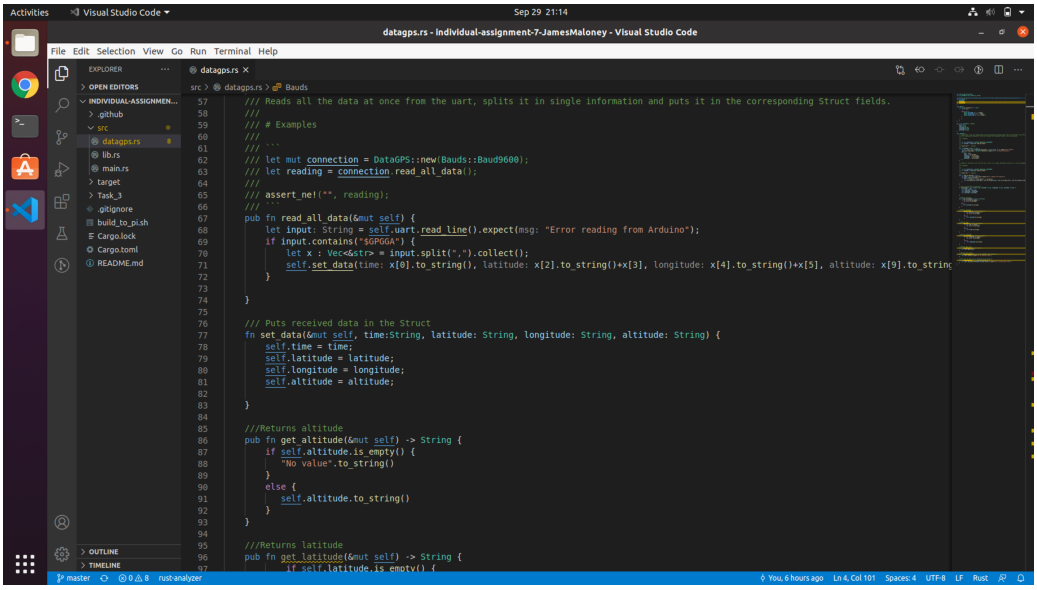

Figure 2: read_all_data() function, some getters and setters.

1.2 Task 2

Task: Use Enums to set all the settings.

- For example baud rate and UART port.

Screenshot: In rpi_embedded crate the only setting to configure when starting a uart connection is the baud rate, since the port is not requested, Parity is always set to "None", data_bits is always 8 and stop_bits is always 1. For this reason, I created an enum just for the baud rate, which allows different configurations (the standard one is 9600, since it is the direct communication mode from GPS, but can also be set to other rates to allow getting data from Arduino or other similar devices, which could act as data interpreters/converters).

```rust
///Defines and implements some Baud rates, which are the most common while communicating using uart.
pub enum Bauds {
    Baud9600,
    Baud57600,
    Baud115200
}

impl Bauds{
    pub fn get_value(&self) -> u32 {
        let v: u32;

        match self {
            Bauds::Baud9600 => { v = 9600 }
            Bauds::Baud57600 => { v = 57600 }
            Bauds::Baud115200 => { v = 115200 }
        }

        v
    }
}
```

Figure 3: Enum with baud rate settings.

1.3 Task 3

Task: Simulate the GPS with an Arduino or test module.

- Take a picture of your code working
- Here is the exact message you get from GPS module, send it with Arduino:

  `$GPGGA,112503.101,6507.4485,N,02255.5355,W,1,04,2.26,17.7,M,60.6,M,,*44`

  `$GPGSA,A,3,28,30,20,15,,,,,,,,,2.45,2.26,0.93*02`

Screenshots: The screenshots below show the uart communication between Arduino and Raspberry Pi, where RPi is receiving data about a simulated GPS module. In the screenshots I only show the code to receive raw data and directly print it, but merging this code with the one from the steps before (which I did in the following tasks) I can also manage it using the Struct and the enum described above.

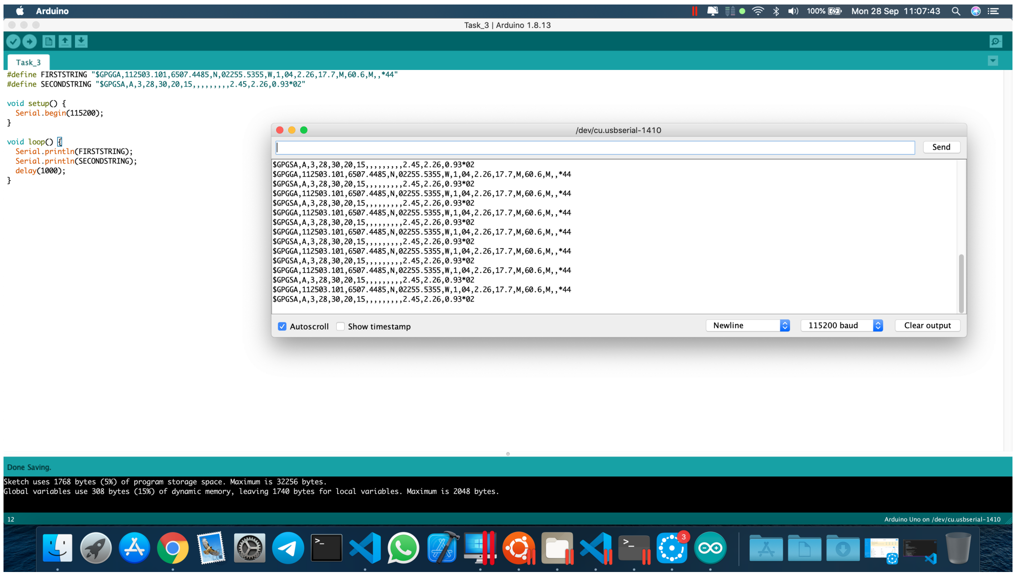

Figure 4: Arduino sending simulation data to Raspberry Pi.

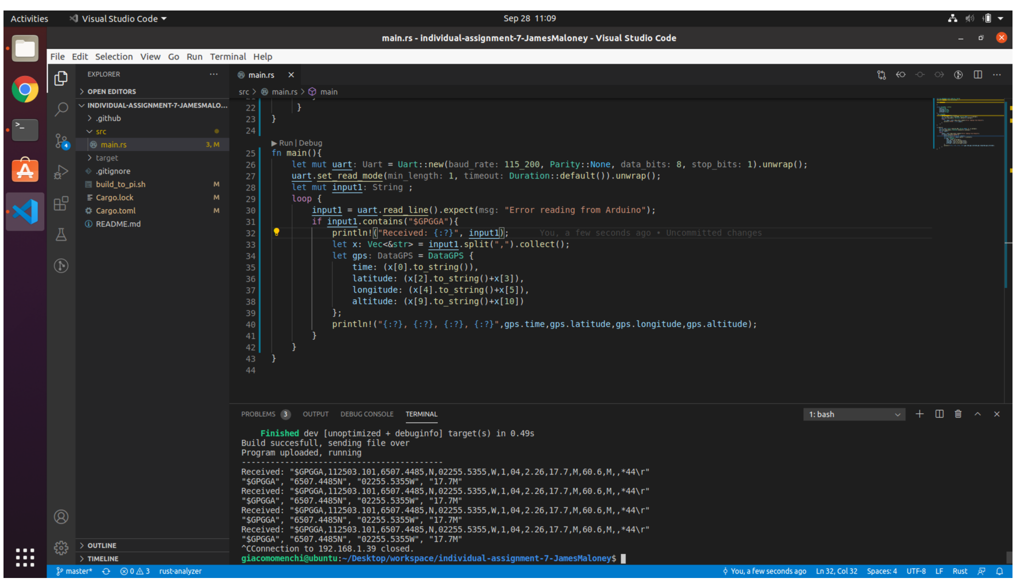

Figure 5: Raspberry Pi receiving raw data from Arduino and printing it out.

1.4 Task 4

Task: Implement some tests to show the functionality of your code have it run when cargo test is called.

- Fx. if the UART works

Screenshot: The screenshots below represent the three implemented tests, which are also thoroughly described in the comments and in the documentation.

- **set_baud_uart_test()** updates the baud rate value during execution and checks if it updates successfully.
- **check_coordinates()** takes already verified coordinates which were obtained from other sources or previous executions and uses them to check if the new ones are correct (in the test the last two digits of the coordinates are not used since the values can fluctuate and thus the test could fail even if it shouldn't).
- **uart_enabled()** tries to read data from an incorrect baud rate, thus panicking (this is actually normal behavior, since the test is set to pass if a panic is detected).

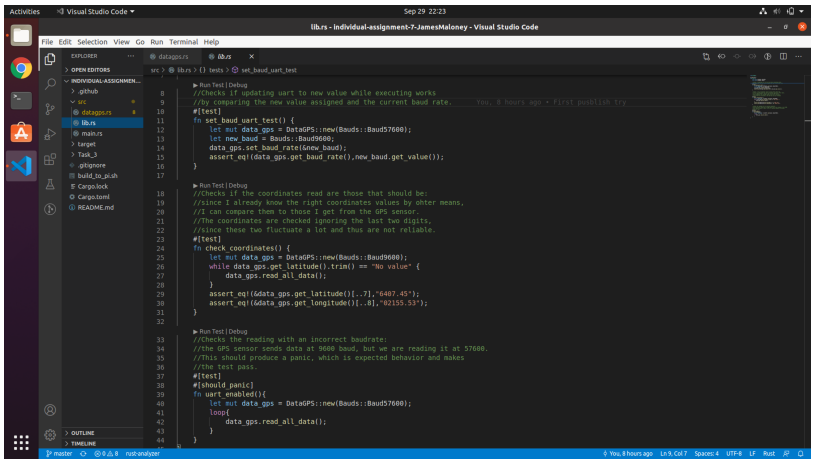

Figure 6: The three implemented tests.

```
    Finished test [unoptimized + debuginfo] target(s) in 0.29s
     Running target/debug/deps/tests-35500ea8d140b93c

running 3 tests
test tests::check_coordinates ... ok
test tests::set_baud_uart_test ... ok
test tests::uart_enabled ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

     Running target/debug/deps/ia7-81a6fc004acd74a2

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

   Doc-tests tests

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

Figure 7: Proof that all the tests are successful.

1.5 Task 5

Task: Publish the crate onto crates.io and send in the handle remember to document how it is used and have examples

- Turn in the name and version that you would set in Cargo.toml [crate_name_example = "0.1.0"]

Explanation: The crate I created is available in crates.io at the link below, and can be added to any project by adding [mech1-gpsreader = "0.4.0"] to the Cargo.toml file. The link below also allows users to access the original repo (which is the same as the one written at the beginning of this report) or to the documentation, which I wrote for publication and that also contains examples.

Link: Crates.io Page

1.6 Video proof

Last, this is a video showing the GPS code working correctly by taking the data from the Adafruit Ultimate GPS (the actual antenna is not shown in the video since it's outside the window to avoid shielded signal) and printing the whole string of received data to terminal.

Link: Video