[Coding Task] Software Roboticist Engineer

Create a driver for custom control protocol using IPC socket communication between publisher and consumer.

Tech stack

- Language: python / C++ / C / Rust
- use git for version control
- send link to the repository with the solution (or zip the repository and send it via email)
- clean commit history is appreciated

Requirements

- solution is compatible with Debian Linux (e.g. Ubuntu)
- application is error prone and can handle invalid input, unexpected exceptions, timeouts, retries etc.
- application is verbose (e.g. logs)
- application is configurable using CLI arguments
- simple README for setup and running
- [Optional] publisher and consumer run as RT tasks

Overview

Example usage

```
publisher --socket-path /path/ --log-level INFO --frequency-hz 500
```

```
consumer --socket-path /path/ --log-level INFO --timeout-ms 100
```

Use any other suitable arguments according to your needs

Publisher

- sends random IMU sensors value as a vector of 12 values, based on the struct definition (3 sensors: accelerometer, gyroscope, magnetometer):

```
typedef struct
{
    float xAcc;     // Acceleration [mg, g=9.81]
    float yAcc;     // Acceleration [mg, g=9.81]
    float zAcc;     // Acceleration [mg, g=9.81]
    uint32_t timestampAcc; // Time stamp of accelerometer measuremen

    int32_t xGyro;  // Gyro rate of rotation around x [mDeg/s]
    int32_t yGyro;  // Gyro rate of rotation around y [mDeg/s]
    int32_t zGyro;  // Gyro rate of rotation around z [mDeg/s]
    uint32_t timestampGyro; // Time stamp of gyro measurement

    float xMag;     // Magnetic induction x axis    [mGauss]
    float yMag;     // Magnetic induction y axis    [mGauss]
    float zMag;     // Magnetic induction z axis    [mGauss]
    uint32_t timestampMag; // Time stamp of magnetometer measurement
} __attribute__((packed)) Payload_IMU_t;
```

- e.g. `0.4343,0.9443,0.2225,1721931959,0.5446,0.7978,0.2211,1721931959,0.9756,0.1212,0.8567,1721931959`
- sending frequency is configurable
- path to the socket is configurable

Consumer

- receives the data, extracts each value
- process and filter the data in order to extract rotation information (euler angles / quaternions)
    - per each sensor and/or as a sensor fusion

    - can be programmed or just explained using pseudo code
    - try to include gimbal lock avoidance
- wait for the data with a configured `timeout` value
- path to the socket is configurable

Comments

- in case of a doubt, make your own assumptions and document them
- if some of the requirement doesn't seem like a good idea, change it and document the reason
- you can use any libraries and tools you want