Internal software architecture

Internally, the software is splitted across different modules :

- camera (video streaming, photo, zoom and image settings)
- interface board (motor control, lights on/off etc.) [not implemented yet]
- control (higher level function such as configuration set/get, etc.)

All of the modules are interfaced together on a common Inter-Process Communication (IPC) bus, using shared memory. This bus is only accessible through native C++ code running on the onboard computer.
The external interface to the user software is done through a TCP connection, which is bridged to the IPC bus.
This TCP connection uses a simple protocol based on JSON requests.
The client (user software) sends a command to the server (the IP bridge), which then replies with JSON data containing the result of the command (either an error or some data).

Here is the basic format for a command :

```
{
    "id": "<a user provided id which will be in the response for this command>",
    "module": "<the submodule>",
    "action": "<the command to execute>",
    "args": {
        // an object with command-specific arguments
    }
}
```

Here is the reply format :

```
{
    "id": "<the same id as in the corresponding command>",
    "ok": true, // true if command successful, false if error
    "return": {
        // additional data, for an error it is the error message, for
        //   a command response it may contain some data
    }
}
```

Important note: when sending or receiving JSON over the TCP socket, it *must* be contained within a single line. The `\n` is used as a delimiter between individual JSON packets. Replies will be formatted the same way.

Available commands

Only commands for the `camera` module are available for the moment. They are documented below. As an example, let's look at the `GetZoomInfo` command, used to query the current zoom value as well as the minimum and maximum zoom values allowed by the camera :

Command	Arguments	Reply
`GetZoomInfo`	*(none)*	`min_zoom` *(int)* `max_zoom` *(int)* `current_zoom` *(int)*

In order to execute this command, one must send the JSON object :

```
{
    "id": "my_id_456",
    "module": "camera",
    "action": "GetZoomInfo",
    "args": {}
}
```

As a one-liner :

```
{ "id": "my_id_456", "module": "camera", "action": "GetZoomInfo", "args": {} }
```

The answer will then be for example (if no error), when properly indented :

```
{
    "id": "my_id_456",
    "ok": true,
    "return": {
        "min_zoom": 0,
        "max_zoom": 16384,
        "current_zoom": 5000
    }
}
```

If an error occured, the `ok` field will be set to `false` and the error message stored in `return.message` :

```
{
    "id": "my_id_456",
    "ok": false,
    "return": {
        "message": "[1] NoCamera: no ack received",
        "code": -1 // this field is not guaranteed to be present and must not be used
    }
}
```

Let's take another example with `SetZoom` :

Command	Arguments	Reply
`SetZoom`	`zoom` *(int)*	*(none)*

Sending this command through the TCP socket (on a single line) :

```
{
    "id": "my_id_789",
    "module": "camera",
    "action": "SetZoom",
    "args": {
        "zoom": 1234
    }
}
```

Will, if no error encountered, produce the following response :

```
{
    "id": "my_id_789",
    "ok": true,
    "return": {}
}
```

Commands for the `camera` module :

Command	Arguments	Reply
`StreamOn`	`rtmp_addr` *(string)* `quality` *(string)*	*(none)*
`StreamOff`	*(none)*	*(none)*
`CaptureStillImage`	`tcp_addr` *(string)* `quality` *(string)*	*(none)*

Command	Arguments	Reply
`GetModelString`	*(none)*	`value` *(string)*
`GetVersionString`	*(none)*	`value` *(string)*
`Reset`	*(none)*	*(none)*
`GetZoomInfo`	*(none)*	`min_zoom` *(int)* `max_zoom` *(int)* `current_zoom` *(int)*
`SetZoom`	`zoom` *(int)*	*(none)*
`GetFocusInfo`	*(none)*	`min_focus` *(int)* `max_focus` *(int)* `current_focus` *(int)* `is_auto` *(bool)*
`SetFocus`	`focus` *(int)*	*(none)*
`SetAutoFocus`	*(none)*	*(none)*
`GetShutterInfo`	*(none)*	`min_shutter` *(int)* `max_shutter` *(int)* `current_shutter` *(int)*
`SetShutter`	`shutter` *(int)*	*(none)*
`ShutterReset`	*(none)*	*(none)*
`ShutterUp`	*(none)*	*(none)*
`ShutterDown`	*(none)*	*(none)*
`GetIrisInfo`	*(none)*	`min_iris` *(int)* `max_iris` *(int)* `current_iris` *(int)*
`SetIris`	`iris` *(int)*	*(none)*
`IrisReset`	*(none)*	*(none)*
`IrisUp`	*(none)*	*(none)*
`IrisDown`	*(none)*	*(none)*

Command	Arguments	Reply
`GetGainInfo`	*(none)*	`min_gain` *(int)* `max_gain` *(int)* `current_gain` *(int)*
`SetGain`	`gain` *(int)*	*(none)*
`GainReset`	*(none)*	*(none)*
`GainUp`	*(none)*	*(none)*
`GainDown`	*(none)*	*(none)*
`GetWB`	*(none)*	`mode` *(string)*
`SetWB`	`mode` *(string)*	*(none)*
`GetAE`	*(none)*	`mode` *(string)*
`SetAE`	`mode` *(string)*	*(none)*
`SetContrastAdjustLevel`	`value` *(float)*	*(none)*
`EnableSpotAE`	`x` *(float)* `y` *(float)*	*(none)*
`DisableSpotAE`	*(none)*	*(none)*
`EnableLRReverse`	*(none)*	*(none)*
`DisableLRReverse`	*(none)*	*(none)*
`EnablePictureFlip`	*(none)*	*(none)*
`DisablePictureFlip`	*(none)*	*(none)*
`EnableFlickerReduction`	*(none)*	*(none)*
`DisableFlickerReduction`	*(none)*	*(none)*