PanTilt C++ library

v1.0.0

Table of contents

Overview

PanTilt is a C++ library designed to serve as a standard interface for various pan-tilt devices. The library defines data structures and rules to showcase an example structure for similar repositories. The library provides methods to encode/decode commands and encode/decode parameters. **PanTilt.h** file contains list of data structures (**PanTiltCommand enum**, **PanTiltParam enum** and **PanTiltParams class**) and **PanTilt** class declaration. **PanTilt** interface depends on **ConfigReader** library to provide methods to read/write JSON config files.

Versions

Table 1 - Library versions.

Version	Release date	What's new
1.0.0	06.02.2024	- First version of PanTilt library.

Library files

The **PanTilt** is a CMake project. Library files:

```
CMakeLists.txt ------------------- Main CMake file of the library.
3rdparty ------------------------- Folder with third-party libraries.
    CMakeLists.txt --------------- CMake file which includes third-party libraries.
    ConfigReader ----------------- Source code of the ConfigReader library.
test ----------------------------- Folder for internal tests of library.
    CMakeLists.txt --------------- CMake file for tests application.
    main.cpp --------------------- Source code file tests application.
src ------------------------------ Folder with source code of the library.
    CMakeLists.txt --------------- CMake file of the library.
    PanTilt.cpp ------------------ Source code file of the library.
    PanTilt.h -------------------- Header file which includes PanTilt class declaration.
    PanTiltVersion.h ------------- Header file which includes version of the library.
    PanTiltVersion.h.in ---------- CMake service file to generate version file.
```

PanTilt interface class description

Class declaration

PanTilt interface class declared in **PanTilt.h** file. Class declaration:

```
class PanTilt
{
public:

    // Class virtual destructor.
    virtual ~PanTilt();

    // Get the version of the PanTilt class.
    static std::string getVersion();

    // Set the value for a specific library parameter.
    virtual bool setParam(PanTiltParam id, float value) = 0;

    // Get the value of a specific library parameter.
    virtual float getParam(PanTiltParam id) const = 0;

    // Get the structure containing all library parameters.
    virtual void getParams(PanTiltParams& params) const = 0;

    // Execute a PanTilt command.
    virtual bool executeCommand(PanTiltCommand id) = 0;

    // Encode set param command.
    static void encodeSetParamCommand(
        uint8_t* data, int& size, PanTiltParam id, float value);

    // Encode command.
    static void encodeCommand(
        uint8_t* data, int& size, PanTiltCommand id);

    // Decode command.
    static int decodeCommand(uint8_t* data,
        int size,
        PanTiltParam& paramId,
        PanTiltCommand& commandId,
        float& value);

    // Decode and execute command.
    virtual bool decodeAndExecuteCommand(uint8_t* data, int size) = 0;
};
```

getVersion method

getVersion() method returns string of current class version. Method declaration:

```
static std::string getVersion();
```

Method can be used without **PanTilt** class instance:

```
std::cout << "PanTilt version: " << cr::pantilt::PanTilt::getVersion();
```

Console output:

```
PanTilt class version: 1.0.0
```

setParam method

setParam(...) method sets new parameters value. **PanTilt** based library should provide thread-safe **setParam(...)** method call. This means that the **setParam(...)** method can be safely called from any thread. Method declaration:

```
bool setParam(PanTiltParam id, float value);
```

Parameter	Description
id	Parameter ID according to PanTiltParam enum.
value	Parameter value. Value depends on parameter ID.

Returns: TRUE if the parameter was set or FALSE if not.

getParam method

getParam(...) method returns parameter value. **PanTilt** based library should provide thread-safe **getParam(...)** method call. This means that the **getParam(...)** method can be safely called from any thread. Method declaration:

```
float getParam(PanTiltParam id);
```

Parameter	Description
id	Parameter ID according to PanTiltParam enum.

Returns: parameter value or **-1** if the parameters doesn't exist.

getParams method

getParams(...) method is designed to obtain params structure. **PanTilt** based library should provide thread-safe **getParams(...)** method call. This means that the **getParams(...)** method can be safely called from any thread. Method declaration:

```
void getParams(PanTiltParams& params);
```

Parameter	Description
params	Reference to PanTiltParams object to store params.

executeCommand method

executeCommand(...) method executes library command. **PanTilt** based library should provide thread-safe **executeCommand(...)** method call. This means that the **executeCommand(...)** method can be safely called from any thread. Method declaration:

```
bool executeCommand(PanTiltCommand id);
```

Parameter	Description
id	Command ID according to PanTiltCommand enum.

Returns: TRUE if the command executed or FALSE if not.

encodeSetParamCommand method

encodeSetParamCommand(...) static method encodes command to change any PanTilt parameter value. To control a pan-tilt device remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **PanTilt** class contains static methods for encoding the control command. The **PanTilt** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). **encodeSetParamCommand(...)** designed to encode SET_PARAM command. Method declaration:

```
static void encodeSetParamCommand(uint8_t* data, int& size, PanTiltParam id, float
value);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 11.
size	Size of encoded data. Will be 11 bytes.
id	Parameter ID according to **PanTilt enum**.
value	Parameter value.

SET_PARAM command format:

Byte	Value	Description
0	0x01	SET_PARAM command header value.
1	Major	Major version of PanTilt class.
2	Minor	Minor version of PanTilt class.
3	id	Parameter ID **int32_t** in Little-endian format.
4	id	Parameter ID **int32_t** in Little-endian format.
5	id	Parameter ID **int32_t** in Little-endian format.

Byte	Value	Description
6	id	Parameter ID **int32_t** in Little-endian format.
7	value	Parameter value **float** in Little-endian format.
8	value	Parameter value **float** in Little-endian format.
9	value	Parameter value **float** in Little-endian format.
10	value	Parameter value **float** in Little-endian format.

encodeSetParamCommand(...) is static and used without **PanTilt** class instance. This method used on client side (control system). Command encoding example:

```
// Buffer for encoded data.
uint8_t data[11];
// Size of encoded data.
int size = 0;
// Random parameter value.
float outValue = static_cast<float>(rand() % 20);
// Encode command.
PanTilt::encodeSetParamCommand(data, size, PanTiltParam::PAN_ANGLE, outValue);
```

encodeCommand method

encodeCommand(...) static method encodes command for PanTilt remote control. To control a pan-tilt device remotely, the developer has to design his own protocol and according to it encode the command and deliver it over the communication channel. To simplify this, the **PanTilt** class contains static methods for encoding the control command. The **PanTilt** class provides two types of commands: a parameter change command (SET_PARAM) and an action command (COMMAND). **encodeCommand(...)** designed to encode COMMAND command (action command). Method declaration:

```
static void encodeCommand(uint8_t* data, int& size, PanTilt Command id);
```

Parameter	Description
data	Pointer to data buffer for encoded command. Must have size >= 7.
size	Size of encoded data. Will be 7 bytes.
id	Command ID according to PanTiltCommand enum.

COMMAND format:

Byte	Value	Description
0	0x00	COMMAND header value.
1	Major	Major version of PanTilt class.
2	Minor	Minor version of PanTilt class.

Byte	Value	Description
3	id	Command ID **int32_t** in Little-endian format.
4	id	Command ID **int32_t** in Little-endian format.
5	id	Command ID **int32_t** in Little-endian format.
6	id	Command ID **int32_t** in Little-endian format.

encodeCommand(...) is static and used without **PanTilt** class instance. This method used on client side (control system). Command encoding example:

```
// Buffer for encoded data.
uint8_t data[7];
// Size of encoded data.
int size = 0;
// Encode command.
PanTilt::encodeCommand(data, size, PanTilt::GO_TO_PAN_ANGLE);
```

decodeCommand method

decodeCommand(...) static method decodes command on pan-tilt device controller side. Method declaration:

```
static int decodeCommand(uint8_t* data, int size, PanTiltParam& paramId, PanTiltCommand&
commandId, float& value);
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Must be 11 bytes for SET_PARAM and 7 bytes for COMMAND.
paramId	PanTilt parameter ID according to **PanTiltParam enum**. After decoding SET_PARAM command the method will return parameter ID.
commandId	PanTilt command ID according to **PanTiltCommand enum**. After decoding COMMAND the method will return command ID.
value	PanTilt parameter value (after decoding SET_PARAM command).

Returns: 0 - in case decoding COMMAND, **1** - in case decoding SET_PARAM command or **-1** in case errors.

decodeAndExecuteCommand method

decodeAndExecuteCommand(...) method decodes and executes command on pan-tilt device controller side. The particular implementation of the PanTilt controller must provide thread-safe **decodeAndExecuteCommand(...)** method call. This means that the **decodeAndExecuteCommand(...)** method can be safely called from any thread. Method declaration:

```
virtual bool decodeAndExecuteCommand(uint8_t* data, int size) = 0;
```

Parameter	Description
data	Pointer to input command.
size	Size of command. Must be 11 bytes for SET_PARAM or 7 bytes for COMMAND.

Returns: TRUE if command decoded (SET_PARAM or COMMAND) and executed (action command or set param command).

Data structures

PanTiltCommand enum

Enum declaration:

```
enum class PanTiltCommand
{
    /// Restart Pan-Tilt device.
    RESTART = 1,
    /// Stop Pan-Tilt device, block all running commands and left device in current
state.
    STOP,
    /// Go to given pan motor position.
    GO_TO_PAN_POSITION,
    /// Go to given tilt motor position.
    GO_TO_TILT_POSITION,
    /// Go to given pan and tilt motor position.
    GO_TO_PAN_TILT_POSITION,
    /// Go to given pan angle.
    GO_TO_PAN_ANGLE,
    /// Go to given tilt angle.
    GO_TO_TILT_ANGLE,
    /// Go to given pan and tilt angle.
    GO_TO_PAN_TILT_ANGLE,
    /// Go to home position.
    GO_TO_HOME
};
```

Table 2 - Commands description.

Command	Description
RESTART	Restart pan-tilt device.
STOP	Stop Pan-Tilt device, block all running commands and left device in current state.
GO_TO_PAN_POSITION	Go to given pan motor position. Valid values from 0 to 65535 (MAX_UINT_16).

Command	Description
GO_TO_TILT_POSITION	Go to given tilt motor position. Valid values from 0 to 65535 (MAX_UINT_16).
GO_TO_PAN_TILT_POSITION	Go to given pan and tilt motor position. Valid values from 0 to 65535 (MAX_UINT_16).
GO_TO_PAN_ANGLE	Go to given pan angle. Valid values from -180.0° to 180.0°.
GO_TO_TILT_ANGLE	Go to given tilt angle. Valid values from -90.0° to 90.0°.
GO_TO_PAN_TILT_ANGLE	Go to given pan and tilt angle. Valid values from -180.0° to 180.0°.
GO_TO_HOME	Go to home position.

PanTiltParam enum

Enum declaration:

```
enum class PanTiltParam
{
    /// Pan motor position for encoder. Range: 0 - 65535.
    PAN_MOTOR_POSITION = 1,
    /// Tilt motor position for encoder. Range: 0 - 65535.
    TILT_MOTOR_POSITION,
    /// Pan angle. Range: -180.0 - 180.0.
    PAN_ANGLE,
    /// Tilt angle. Range: -90.0 - 90.0.
    TILT_ANGLE,
    /// Pan tilt motor position for encoder. Range: 0 - 65535.
    PAN_TILT_MOTOR_POSITION,
    /// Pan tilt angle. Range: -180.0 - 180.0.
    PAN_TILT_ANGLE,
    /// Pan motor speed. Range: 0.0 - 100.0.
    PAN_MOTOR_SPEED,
    /// Tilt motor speed. Range: 0.0 - 100.0.
    TILT_MOTOR_SPEED,
    /// Pan tilt motor speed. Range: 0.0 - 100.0.
    PAN_TILT_MOTOR_SPEED
};
```

Table 3 - Params description.

Parameter	Access	Description
PAN_MOTOR_POSITION	read / write	Pan motor position for encoder. Range: 0 to 65535.
TILT_MOTOR_POSITION	read / write	Tilt motor position for encoder. Range: 0 to 65535.
PAN_ANGLE	read / write	Pan angle. Range: -180.0 to 180.0.
TILT_ANGLE	read / write	Tilt angle. Range: -90.0 - 90.0.

Parameter	Access	Description
PAN_TILT_MOTOR_POSITION	read / write	Pan tilt motor position for encoder. Range: 0 - 65535.
PAN_TILT_ANGLE	read / write	Pan tilt angle. Range: -180.0 - 180.0.
PAN_MOTOR_SPEED	read / write	Pan motor speed. Range: 0.0 - 100.0.
TILT_MOTOR_SPEED	read / write	Tilt motor speed. Range: 0.0 - 100.0.
PAN_TILT_MOTOR_SPEED	read / write	Pan tilt motor speed. Range: 0.0 - 100.0.

PanTiltParams class description

Class declaration

PanTiltParams class is used to provide pan-tilt parameters structure. Also **PanTiltParams** provides possibility to write/read params from JSON files (**JSON_READABLE** macro) and provides methods to encode and decode params. **PanTiltParams** interface class declared in **PanTilt.h** file. Class declaration:

```
class PanTiltParams
{
public:

    /// Pan motor position for encoder. Range: 0 - 65535.
    int panMotorPosition{ 0 };
    /// Tilt motor position for encoder. Range: 0 - 65535.
    int tiltMotorPosition{ 0 };
    /// Pan angle. Range: -180.0 - 180.0.
    float panAngle{ 0.0f };
    /// Tilt angle. Range: -90.0 - 90.0.
    float tiltAngle{ 0.0f };
    /// Pan tilt motor position for encoder. Range: 0 - 65535.
    int panTiltMotorPosition{ 0 };
    /// Pan tilt angle. Range: -180.0 - 180.0.
    float panTiltAngle{ 0.0f };
    /// Pan motor speed. Range: 0.0 - 100.0.
    float panMotorSpeed{ 0.0f };
    /// Tilt motor speed. Range: 0.0 - 100.0.
    float tiltMotorSpeed{ 0.0f };
    /// Pan tilt motor speed. Range: 0.0 - 100.0.
    float panTiltMotorSpeed{ 0.0f };

    /// Macro from ConfigReader to make params readable/writable from JSON.
    JSON_READABLE(PanTiltParams, panMotorPosition, tiltMotorPosition, panAngle,
        tiltAngle, panTiltMotorPosition, panTiltAngle, panMotorSpeed, tiltMotorSpeed,
        panTiltMotorSpeed)

    /// operator =
        PanTiltParams& operator= (const PanTiltParams& src);
    // Encode (serialize) params.
    bool encode(uint8_t* data, int bufferSize, int& size,
                                        PanTiltParamsMask* mask = nullptr);
    // Decode (deserialize) params.
    bool decode(uint8_t* data, int dataSize);
};
```

Table 4 - **PanTiltParams** class fields description is related to PanTiltParam enum description.

Field	type	Description
panMotorPosition	int	Pan motor position for encoder. Range: 0 - 65535.
tiltMotorPosition	int	Tilt motor position for encoder. Range: 0 - 65535.
panAngle	float	Pan angle. Range: -180.0 - 180.0.
tiltAngle	float	Tilt angle. Range: -90.0 - 90.0.
panTiltMotorPosition	int	Pan tilt motor position for encoder. Range: 0 - 65535.
panTiltAngle	float	Pan tilt angle. Range: -180.0 - 180.0.
panMotorSpeed	float	Pan motor speed. Range: 0.0 - 100.0.
tiltMotorSpeed	float	Tilt motor speed. Range: 0.0 - 100.0.
panTiltMotorSpeed	float	Pan tilt motor speed. Range: 0.0 - 100.0.

None: *PanTiltParams class fields listed in Table 4* ***have to*** *reflect params set/get by methods setParam(...) and getParam(...).*

Serialize PanTilt params

PanTiltParams class provides method **encode(...)** to serialize PanTilt params. Serialization of PanTilt params is necessary in case when PanTilt params have to be sent via communication channels. Method provides options to exclude particular parameters from serialization. To do this method inserts binary mask (1 byte) where each bit represents particular parameter and **decode(...)** method recognizes it. Method declaration:

```
bool encode(uint8_t* data, int bufferSize, int& size, PanTiltParamsMask* mask = nullptr);
```

Parameter	Value
data	Pointer to data buffer. Buffer size must be >= 48 bytes.
bufferSize	Data buffer size. Buffer size must be >= 48 bytes.
size	Size of encoded data.

Parameter	Value
mask	Parameters mask - pointer to **PanTiltParamsMask** structure. **PanTiltParamsMask** (declared in PanTilt.h file) determines flags for each field (parameter) declared in PanTiltParams class. If the user wants to exclude any parameters from serialization, he can put a pointer to the mask. If the user wants to exclude a particular parameter from serialization, he should set the corresponding flag in the **PanTiltParamsMask** structure.

Returns: TRUE if params encoded (serialized) or FALSE if not.

PanTiltParamsMask structure declaration:

```
struct PanTiltParamsMask
{
    bool panMotorPosition{ true };
    bool tiltMotorPosition{ true };
    bool panAngle{ true };
    bool tiltAngle{ true };
    bool panTiltMotorPosition{ true };
    bool panTiltAngle{ true };
    bool panMotorSpeed{ true };
    bool tiltMotorSpeed{ true };
    bool panTiltMotorSpeed{ true };
};
```

Example without parameters mask:

```
// Prepare parameters.
cr::pantilt::PanTiltParams params;
params.panAngle = 160.0f;

// Encode (serialize) params.
int bufferSize = 128;
uint8_t buffer[128];
int size = 0;
params.encode(buffer, bufferSize, size);
```

Example with parameters mask:

```
// Prepare parameters.
cr::pantilt::PanTiltParams params;
params.panAngle = 160.0f;

// Prepare mask.
cr::pantilt::PanTiltParamsMask mask;
// Exclude tiltAngle.
mask.tiltAngle = false;

// Encode (serialize) params.
int bufferSize = 128;
uint8_t buffer[128];
int size = 0;
```

```
params1.encode(buffer, bufferSize, size, &mask);
```

Deserialize PanTilt params

PanTiltParams class provides method **decode(...)** to deserialize params. Deserialization of PanTilt params is necessary in case when it is needed to receive params via communication channels. Method automatically recognizes which parameters were serialized by **encode(...)** method. Method declaration:

```
bool decode(uint8_t* data, int dataSize);
```

Parameter	Value
data	Pointer to data buffer with serialized params.
dataSize	Size of command data.

Returns: TRUE if params decoded (deserialized) or FALSE if not.

Example:

```
// Prepare parameters.
cr::pantilt::PanTiltParams params1;
params1.panAngle = 160.0f;

// Encode (serialize) params.
int bufferSize = 128;
uint8_t buffer[128];
int size = 0;
params1.encode(buffer, bufferSize, size);

// Decode (deserialize) params.
cr::pantilt::PanTiltParams params2;
params2.decode(buffer, size);
```

Read params from JSON file and write to JSON file

PanTilt depends on open source **ConfigReader** library which provides method to read params from JSON file and to write params to JSON file. Example of writing and reading params to JSON file:

```
// Write params to file.
cr::utils::ConfigReader inConfig;
cr::pantilt::PanTiltParams in;
inConfig.set(in, "panTiltParams");
inConfig.writeToFile("PanTiltParams.json");

// Read params from file.
cr::utils::ConfigReader outConfig;
if(!outConfig.readFromFile("PanTiltParams.json"))
{
    cout << "Can't open config file" << endl;
    return false;
}
```

PanTiltParams.json will look like:

```
{
    "panTiltParams":
    {
        "panMotorPosition": 43565,
        "tiltMotorPosition": 10500,
        "panAngle": 30.5f,
        "tiltAngle": 89.9f,
        "panTiltMotorPosition": 300,
        "panTiltAngle": 60.0f,
        "panMotorSpeed": 50.0f,
        "tiltMotorSpeed": 100.0f,
        "panTiltMotorSpeed": 10.5f
    }
}
```

Build and connect to your project

Typical commands to build **PanTilt**:

```
git clone https://github.com/ConstantRobotics-Ltd/PanTilt.git
cd PanTilt
git submodule update --init --recursive
mkdir build
cd build
cmake ..
make
```

If you want connect **PanTilt** to your CMake project as source code you can make follow. For example, if your repository has structure:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
```

You can add repository **PanTilt** as submodule by commands:

```
cd <your respository folder>
git submodule add https://github.com/ConstantRobotics-Ltd/PanTilt.git 3rdparty/PanTilt
git submodule update --init --recursive
```

In your repository folder will be created folder **3rdparty/PanTilt** which contains files of **PanTilt** repository with subrepository **ConfigReader** and **ConfigReader**. New structure of your repository:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    PanTilt
```

Create CMakeLists.txt file in **3rdparty** folder. CMakeLists.txt should contain:

```
cmake_minimum_required(VERSION 3.13)

################################################################################
## 3RD-PARTY
## dependencies for the project
################################################################################
project(3rdparty LANGUAGES CXX)

################################################################################
## SETTINGS
## basic 3rd-party settings before use
################################################################################
# To inherit the top-level architecture when the project is used as a submodule.
SET(PARENT ${PARENT}_YOUR_PROJECT_3RDPARTY)
# Disable self-overwriting of parameters inside included subdirectories.
SET(${PARENT}_SUBMODULE_CACHE_OVERWRITE OFF CACHE BOOL "" FORCE)

################################################################################
## CONFIGURATION
## 3rd-party submodules configuration
################################################################################
SET(${PARENT}_SUBMODULE_PAN_TILT                      ON  CACHE BOOL "" FORCE)
if (${PARENT}_SUBMODULE_PAN_TILT)
    SET(${PARENT}_PAN_TILT                            ON  CACHE BOOL "" FORCE)
    SET(${PARENT}_PAN_TILT_TEST                       OFF CACHE BOOL "" FORCE)
    SET(${PARENT}_PAN_TILT_EXAMPLE                    OFF CACHE BOOL "" FORCE)
endif()

```

```
################################################################################
## INCLUDING SUBDIRECTORIES
## Adding subdirectories according to the 3rd-party configuration
################################################################################
if (${PARENT}_SUBMODULE_PAN_TILT)
    add_subdirectory(PanTilt)
endif()
```

File **3rdparty/CMakeLists.txt** adds folder **PanTilt** to your project and excludes test application and example (PanTilt class test application and example of custom **PanTilt** class implementation) from compiling. Your repository new structure will be:

```
CMakeLists.txt
src
    CMakeList.txt
    yourLib.h
    yourLib.cpp
3rdparty
    CMakeLists.txt
    PanTilt
```

Next you need include folder 3rdparty in main **CMakeLists.txt** file of your repository. Add string at the end of your main **CMakeLists.txt**:

```
add_subdirectory(3rdparty)
```

Next you have to include **PanTilt** library in your **src/CMakeLists.txt** file:

```
target_link_libraries(${PROJECT_NAME} PanTilt)
```

Done!

How to make custom implementation

The **PanTilt** class provides only an interface, data structures, and methods for encoding and decoding commands and params. To create your own implementation of the pan-tilt controller, PanTilt repository has to be included in your project (see [Build and connect to your project]() section). The catalogue **example** (see [Library files]() section) includes an example of the design of the custom pan-tilt controller. All the methods of the PanTilt interface class have to be included. Custom PanTilt class declaration:

```
class CustomPanTilt : public PanTilt
{
public:

    // Class constructor.
    CustomPanTilt();

    // Class destructor.
    ~CustomPanTilt();

```

```cpp
    // Get the version of the PanTilt class.
    static std::string getVersion();

    // Set the value for a specific library parameter.
    bool setParam(PanTiltParam id, float value) override;

    // Get the value of a specific library parameter.
    float getParam(PanTiltParam id) override;

    // Get the structure containing all library parameters.
    void getParams(PanTiltParams& params) override;

    // Execute a PanTilt command.
    bool executeCommand(PanTiltCommand id) override;

    // Decode and execute command.
    bool decodeAndExecuteCommand(uint8_t* data, int size);

private:

    /// Parameters structure (default params).
    PanTiltParams m_params;
    /// Mutex for parameters access.
    std::mutex m_paramsMutex;
};
```