AGIBOT Genie 02
GENIE Development Kit
V2.1.0 User Manual

AGIBOT

Genie 02 GDK v2.1.0

1. Overview

1.1 Glossary

- Robot Domain Controller
- Local Deployment
- Hybrid Deployment
- Motion Control

2. Main Features

3. Quick Start

3.1 Environmental Requirements

3.1.1 Hardware Requirements

- Complies withISO/IEC 11801:2002andEIA/TIA 568Bor ISO/IEC 11801:2002 and EIA/TIA 568-B.2-1 standard network cables
- Developer computer (Ubuntu 22.04, x86_64, Intel i9 or higher performance CPU)

3.1.2 Network Configuration

Connect the Debug network port of the G2 interface panel and the network port of the developer computer using a network cable, and then configure a static IP for the network interface card of the developer computer:

```
Plain Text
interface config
IP：10.42.1.xxx(10.42.1.10~10.42.1.99)
MASK：255.255.255.0
```

Verify whether the communication with G2 is normal on the developer computer:

```
C++
shell
# 其中 10.42.0.101 为 G2 的固定 IP
ping 10.42.1.101
```

3.2 Install GDK

3.2.1 Local Deployment

Connect to G2 using ssh, password is 1

```
C++
```

```
ssh agi@10.42.1.101
```

Enter the deployment package and declare environment variables.

```
C++
cd app
source env.sh
```

3.2.2 Hybrid Deployment

Execute the following command on the developer computer Console to install GDK:

```
Plain Text
curl -sSL http://10.42.1.101:8849/install.sh | bash
```

3.3 Get sample code

3.3.1 Obtain the Compressed Packet of sample code from G2

```
C++
curl -sSL http://10.42.1.101:8849/install_example.sh | bash
```

3.3.2 Folder Structure

```
C++
├── C++
│   ├── CMakeLists.txt
│   └── src
├── python
│   ├── dds_example.py
│   ├── genie_02_example.py
│   ├── hal_dds_example.py
│   ├── mc_example.py
│   ├── proto
│   └── saved_commands
└── ros
    ├── camera_example
    ├── control_example
    └── genie_msgs
```

3.3.3 Running Example (Python Version Motion Control Example)

3.3.3.1 Declare the Python path

```
Plain Text
source ~/.cache/agibot/app/env.sh
```

3.3.3.2 Run mc_example.py

```
Plain Text
```

```
python3 mc_example.py
```

The robot joints can be controlled via keyboard input to execute action sequences

3.3.3.2.1 Operation Instructions

Button	Function
a/d	Switch Joint
w/s	Adjust joint angle
m	Switch action sequence
p	Play action sequence
q	Exit the program

4. Instructions for Use

4.1 ROS2 Program Development (humble)

Genie 02 Controller internally uses high-performance DDS by default and does not send ROS2 messages (which affects performance) by default. When users use ROS2 communication, they need to start the ROS2 forwarding node of GDK.

During the installation of GDK, the necessary ROS2 message definitions are also installed in the system. After users compile these messages, they can use ROS2 tools or programs to receive/send messages.

4.1.1 Obtain ROS2 message definitions and compile them

4.1.1.1 Hybrid Deployment

```
Plain Text
cd your_workspace
cp -r ~/.cache/agibot/app/gdk/examples/ros/genie_msgs/ ./
colcon build
```

4.1.1.2 Local Deployment

```
C++
cd your_workspace
cp -r /home/agi/app/gdk/examples/ros/genie_msgs/ ./
colcon build
```

4.1.1.3 View messages (taking the lidar node as an example)

4.1.1.3.1 Start the ROS forwarding node for lidar

```
Plain Text
source ~/.cache/agibot/app/gdk/ros_env gdk_lidar
ros2 launch gdk_lidar lidar.launch
```

4.1.1.3.2 Declare ROS environment variables

```
Plain Text
export ROS_DOMAIN_ID=0
export ROS_LOCALHOST_ONLY=1
```

4.1.1.3.3 View node information

```
Plain Text
ros2 topic list
ros2 topic echo /lidar/livox_front
ros2 topic hz /lidar/livox_back
```

4.1.2 Basic Usage (Taking Motion Control as an Example)

4.1.2.1 Create a subscriber for the /hal/joint_state message to obtain information such as the names, angles, and velocities of all current joints of the robot

```
C++
c++
sub_joint_ = this->create_subscription<genie_msgs::msg::JointState>(
            "/hal/joint_state",
            10,
            std::bind(&ControlExampleNode::joint_state_callback, this,
std::placeholders::_1)
);
```

4.1.2.2 Create a publisher for the /MotionControlService/JointPosition/request message

```
C++
c++
//
pub_joint_position_ =
create_publisher<genie_msgs::msg::JointPositionRequst>("/MotionControlService/JointPosition
/request", 10);
```

4.1.2.3 Set your control commands and publish them, and you can control the corresponding joints

```
C++
c++
genie_msgs::msg::JointPositionRequst request;
request.lifetime = 1.0;
request.joint_names = joint_names_;
request.joint_positions = target_positions;
request.joint_velocities = {velocity_};
request.uuid = generate_uuid_boost();
pub_joint_position_->publish(request);
```

4.1.2.4 Start the ROS2 forwarding node

Here, the motion control node package is gdk_controller, and the specific ROS package name can be queried in app/gdk/share

```
C++
source app/gdk/scripts/ros_env.sh gdk_controller # gdk_controller 可根据实际使用改变
ros2 launch gdk_controller controller.launch     # 启动控制节点
```

4.1.2.5 Set ROS communication configuration at the end point

```
C++
export ROS_DOMAIN_ID=0
export ROS_LOCALHOST_ONLY=1
```

4.1.2.6 Start Node

```
C++
source your_pkg/install/setup.bash
ros2 launch your_pkg your_pkg_launch
```

4.2 Python Program Development

4.2.1 Environment Configuration

```
C++
pip install protobuf==5.28.3
```

4.2.2 Declare the Python path

4.2.2.1 Hybrid Deployment

```
C++
source ~/.cache/agibot/app/env.sh
```

4.2.2.2 Local Deployment

```
C++
source /home/agi/app/env.sh
```

4.2.3 Import Python Modules

```
C++
import agibot_gdk
```

For detailed instructions on using the Python interface, please refer to Chapter 5 "Detailed Interface Description".

4.3 C++ Program Development

4.3.1 Header File Inclusion Method

```
C++
#include "gdk/gdk.h"
```

4.3.2 CMake Compile Configuration

The GDK's Header Files and Dynamic Libraries are deployed in the GDK_HOME directory, which defaults to ~/.cache/agibot on the development machine and /home/agi/ on the robot domain controller.

4.3.2.1 Compile in Hybrid Deployment

Add the following content to your CMakeLists.txt and use this method to create a compilation project

```
Bash
CMakeLists.txt
if(NOT DEFINED ENV{GDK_HOME})
    set(GDK_HOME_PATH ~/.cache/agibot)
else()
    set(GDK_HOME_PATH $ENV{GDK_HOME})
endif()

function(add_adapter name)
    add_executable(${name} src/${name}.cpp)

    if(CMAKE_SYSTEM_PROCESSOR STREQUAL "x86_64")
        target_include_directories(${name} PRIVATE
${GDK_HOME_PATH}/app/gdk/build_dep/x86_64/include)
        target_link_libraries(${name} PRIVATE
${GDK_HOME_PATH}/app/gdk/build_dep/x86_64/lib/libgdk_adapter.so)
    elseif(CMAKE_SYSTEM_PROCESSOR STREQUAL "aarch64")
        target_include_directories(${name} PRIVATE
${GDK_HOME_PATH}/app/gdk/build_dep/aarch64/include)
        target_link_libraries(${name} PRIVATE
${GDK_HOME_PATH}/app/gdk/build_dep/aarch64/lib/libgdk_adapter.so)
    endif()
endfunction()
```

4.3.2.2 Compile in Local Deployment

Add the following content to your CMakeLists.txt and use this method to create a compilation project

```
Bash
CMakeLists.txt
set(GDK_HOME_PATH ~/.cache/agibot)

function(add_adapter name)
    add_executable(${name} src/${name}.cpp)
    target_include_directories(${name} PRIVATE
${GDK_HOME_PATH}/app/gdk/build_dep/aarch64/include)
    target_link_libraries(${name} PRIVATE ${GDK_HOME_PATH}/app/gdk/lib/libgdk_adapter.so)
endfunction()
```

4.3.3 Declare the path of the compiled linked library

4.3.3.1 Hybrid Deployment

```
C++
source ~/.cache/agibot/env.sh
```

4.3.3.2 Local Deployment

```
C++
source /home/agi/app/env.sh
```

4.3.4 Compile

```
C++
cmake -B build
make -C build
```

Compiled artifacts can be queried in the build folder

5. Interface Detailed Description

5.1 ROS2

5.1.1 Camera Node

5.1.1.1 Topic

Topic Name	Meaning
`/camera/head_back_fisheye`	Head and Back Fisheye Camera
`/camera/head_left_fisheye`	Left Head Fisheye Camera
`/camera/head_right_fisheye`	Right-side Fisheye Camera on the Head
`/camera/head_stereo_left`	Head Left Eye Camera
`/camera/head_stereo_right`	Head Right Eye Camera
`/camera/hand_left`	Wrist Depth Camera
`/camera/hand_right`	Wrist Depth Camera
`/camera/head_color`	Head RGB Camera
`/camera/head_depth`	Head Depth Camera

5.1.1.2 Message Type

`sensor::msgs::msg::Image`

Fields

Field	Type	Description
`header`	`std_msgs/Header`	Message header, containing timestamp and coordinate system information

`height`	`uint32`	Image height (number of pixel rows)
`width`	`uint32`	Image width (number of pixel columns)
`encoding`	`string`	Encoding format string for pixel data
`is_bigendian`	`bool`	Data byte order (true indicates big-endian)
`step`	`uint32`	Number of bytes occupied by a single row of data
`data`	`uint8[]`	Original pixel data of the image

Single-channel image

Encoding	Description	Bytes per Pixel
`mono8`	8-bit canary release image	1
`mono16`	16-bit canary release image	2
`bayer_bggr8`	Bayer BGGR Format	1
`bayer_rggb8`	Bayer RGGB Format	1

Multi-channel Image

Encoding	Description	Bytes per Pixel
`bgr8`	8-bit BGR color	3
`rgb8`	8-bit RGB color	3
`bgra8`	8-bit BGRA with transparency	4
`rgba8`	8-bit RGBA with transparency	4

Special Format

Encoding	Description	Bytes per Pixel
`32FC1`	32-bit floating-point single-channel	4
`16UC1`	16-bit unsigned integer single-channel	2
`8UC3`	8-bit unsigned integer three-channel	3

5.1.1.3 Start the ROS2 forwarding node

5.1.1.3.1 Hybrid Deployment

```
C++
source ~/.cache/agibot/app/gdk/scripts/ros_env.sh gdk_camera
ros2 launch gdk_camera camera.launch
```

5.1.1.3.2 Local Deployment

```
C++
source app/gdk/scripts/ros_env.sh gdk_camera
ros2 launch gdk_camera camera.launch
```

5.1.2 Controller Node

5.1.2.1 Whole-body state acquisition

5.1.2.1.1 Topic

`/wbc/motion_control_status`

5.1.2.1.2 Message Type

`genie_msgs::msg::MotionControlStatus.msg`

```
Plain Text
MotionControlStatus.msg
std_msgs/Header header  # frame id base_link

# Poses of end effectors
string[] frame_names   #parameters of motion control
geometry_msgs/Pose[] frame_poses

# collision status
# [collision_pair_1,  collision_pair_2]frame name of collision pair
string[] collision_pairs_1
string[] collision_pairs_2

# motion mode
uint8 MODE_STOP=0
uint8 MODE_SERVO=1
uint8 MODE_PLANNING=2
uint8 mode

# 0: no error
uint8 error_code
string error_msg

geometry_msgs/Twist[] frame_twists
geometry_msgs/Wrench[] frame_wrenchs
```

5.1.2.2 Joint Angle Acquisition

5.1.2.2.1 Topic

`/hal/joint_state`

5.1.2.2.2 Message Type

`genie_msgs::msg::JointState`

```
Plain Text
JointState.msg
# The state of each joint (revolute or prismatic) is defined by:
#  * the control mode (csp 0, cst 1)
#  * the position of the joint (rad or m),
#  * the velocity of the joint (rad/s or m/s),
#  * the effort that is applied in the joint (Nm or N),
#  * the position of the motor (rad or m),
#  * the velocity of the motor (rad/s or m/s),
#  * the current of the motor (A),
#  * the error_code of the joint.
#
# TODO: Error code description or joint error code page

std_msgs/Header header

string[] name
uint32[] mode
float64[] position
float64[] velocity
float64[] effort
float64[] motor_position
float64[] motor_velocity
float64[] motor_current
uint32[] error_code
```

5.1.2.3 Joint Angle Control

5.1.2.3.1 Topic

`/MotionControlService/JointPosition/request`

5.1.2.3.2 Message Type

genie_msgs::msg::JointPositionRequst.msg

```
Plain Text
JointPositionRequst.msg
# frame id , timestamp is send time
std_msgs/Header header
float64 lifetime

string[] joint_names
float64[] joint_positions
float64[] joint_velocities

string uuid
string detail
```

5.1.2.4 Control Command Feedback

5.1.2.4.1 Topic

`/MotionControlService/JointPosition/response`

5.1.2.4.2 Message Type

`genie_msg::msg::CommonResponse.msg`

```
Plain Text
CommonResponse.msg
std_msgs/Header header

uint8 data

string uuid
string detail
```

5.1.3 Imu Node

5.1.3.1 Topic

Topic Name	Meaning
`/imu/xt_chest`	Chassis IMU
`/imu/livox_front`	Front Radar IMU
`/imu/livox_back`	Back Radar IMU
`/imu_chassis`	Chest IMU

5.1.3.2 Message Type

`sensor_msgs::msg::Imu`

Field	Type	Description
`header`	`std_msgs/Header`	Message header, containing timestamp and coordinate system information
`orientation`	`geometry_msgs/Quaternion`	Orientation (quaternion) relative to the reference coordinate system
`orientation_covariance`	`float64[9]`	Covariance matrix of direction measurement (row-major)
`angular_velocity`	`geometry_msgs/Vector3`	Three-dimensional angular velocity (rad/s)
`angular_velocity_covariance`	`float64[9]`	Covariance matrix of angular velocity measurement
`linear_acceleration`	`geometry_msgs/Vector3`	Three-dimensional linear acceleration (m/s²)
`linear_acceleration_covariance`	`float64[9]`	Covariance matrix of linear acceleration measurement

5.1.3.3 Start the ROS2 forwarding node

5.1.3.3.1 Hybrid Deployment

```
C++
source ~/.cache/agibot/app/gdk/scripts/ros_env.sh gdk_imu
ros2 launch gdk_imu imu.launch
```

5.1.3.3.2 Local Deployment

```
C++
source app/gdk/scripts/ros_env.sh gdk_imu
ros2 launch gdk_imu imu.launch
```

5.1.4 Lidar Node

5.1.4.1 Topic

Topic Name	Meaning
`/lidar/xt_chest`	Millimeter Wave Radar
`/lidar/livox_front`	Front LiDAR
`/lidar/livox_back`	Rear LiDAR

5.1.4.2 Message Type

`sensor_msgs::msg::PointCloud2`

Field	Type	Description
`header`	`std_msgs/Header`	Message header, containing timestamp and coordinate system information
`height`	`uint32`	Height of the point cloud (1 if the point cloud is unorganized)
`width`	`uint32`	Width of the point cloud (number of points)
`fields`	`PointField[]`	Describe the field structure of each point in the point cloud
`is_bigendian`	`bool`	Data byte order (true indicates big-endian)
`point_step`	`uint32`	Number of bytes occupied by a single point
`row_step`	`uint32`	Number of bytes occupied by a row of data
`data`	`uint8[]`	Original binary data of the point

		cloud
`is_dense`	`bool`	If true, it indicates that all points contain valid data

`PointField`

Each `PointField` describes a field in the point cloud:

Field	Type	Description
`name`	`string`	Field names (e.g., "x", "y", "z", "rgb", etc.)
`offset`	`uint32`	Byte offset of the field in point data
`datatype`	`uint8`	Data type (1 represents INT8, 2 represents UINT8, 3 represents INT16, 4 represents UINT16, 5 represents INT32, 6 represents UINT32, 7 represents FLOAT32, 8 represents FLOAT64)
`count`	`uint32`	Number of elements in this field

5.1.4.3 Start the ROS2 forwarding node

5.1.4.3.1 Hybrid Deployment

```
C++
source ~/.cache/agibot/app/gdk/scripts/ros_env.sh gdk_lidar
ros2 launch gdk_lidar lidar.launch
```

5.1.4.3.2 Local Deployment

```
C++
source app/gdk/scripts/ros_env.sh gdk_lidar
ros2 launch gdk_lidar lidar.launch
```

5.2 Python

5.2.1 Whole Body Status and Command Interface

RobotBody Class

This class encapsulates the main motion control interfaces of the robot chassis.

1. **`get_state()`**

- **Function**: Get the status values of all body actuators
- **Parameter**:

Imported Parameter	Return Value	Remarks
`"body_joint_states`	`{'timestamp':xxx,'nums':xxx,'states':xx`	Timestamp, number of joints,

"	xx}	status value

- **Return Value Description**:
  - `timestamp`: Timestamp (nanoseconds)
  - `nums`: Number of joints
  - `states`: List of joint states, each joint includes:
    - `name`: Joint Name
    - `position`: Joint position (radians)
    - `velocity`: Joint velocity (radians/second)
    - `effort`: Joint torque (N·m)
- **Example**:

```
Python
import agibot_gdk

robot_body = agibot_gdk.RobotBody()

# Get the status of joints
joint_states = robot_body.get_state("body_joint_states")
print(f"关节数量: {joint_states['nums']}")
for state in joint_states['states']:
    print(f"关节: {state['name']}, 位置: {state['position']:.3f}")

#
end_states = robot_body.get_state("end_states")
print(f"左手控制状态: {end_states['left_end_state']['controlled']}")

# get status of whole boday
whole_status = robot_body.get_state("whole_body_status")
print(f"右臂错误码: {whole_status['right_arm_error']}")
```

2. `get_end_state()`

- **Function**: Get end-effector status
- **Parameter**: `"end_state"`
- **Return value** : End-effector status dictionary
- **Return value structure** :

Attribute Name	Type	Description	Example Value
`left_end_state`	`dict`	Left actuator status information	See the detailed structure below
`right_end_state`	`dict`	Right actuator status information	See the detailed structure below

left_end_state/right_end_state 结构:

Attribute Name	Type	Description	Example Value
`controlled`	`bool`	Is it controlled?	`True`

`type`	`int`	Actuator Type	`2`
`names`	`list`	Joint Name List	`['left_gripper_joint1']`
`end_states`	`list`	Joint Status List	See the detailed structure below

Each joint state structure in end_states:

Attribute Name	Type	Description	Unit	Example Value
`id`	`int`	Joint ID	None	`0`
`enable`	`bool`	Enable or not	None	`True`
`position`	`float`	Joint Position	degrees	`120.0`
`velocity`	`float`	Joint Velocity	degrees/second	`0.0`
`effort`	`float`	Joint Torque	N·m	`0.35`
`current`	`float`	Motor Current	A	`0.0`
`voltage`	`float`	Motor Voltage	V	`0.0`
`temperature`	`float`	Motor Temperature	°C	`0.0`
`status`	`int`	Status Code	None	`0`
`err_code`	`int`	Error Code	None	`0`

- **Example**:

```python
import agibot_gdk
import time

robot_body = agibot_gdk.RobotBody()
time.sleep(1)

# get status of effectors
end_state = robot_body.get_end_state("end_state")

# print the left end effector status
left_state = end_state['left_end_state']
print(f"左执行器控制状态: {left_state['controlled']}")
print(f"左执行器类型: {left_state['type']}")
print(f"左执行器关节: {left_state['names']}")

# print the detail status by joint# frame id 无用, timestamp 发送时间
for i, joint_state in enumerate(left_state['end_states']):
    print(f"\n 左执行器关节 {i+1}:")
```

```
    print(f"  关节 ID: {joint_state['id']}")
    print(f"  启用状态: {joint_state['enable']}")
    print(f"  位置: {joint_state['position']:.1f}°")
    print(f"  速度: {joint_state['velocity']:.1f}°/s")
    print(f"  力矩: {joint_state['effort']:.3f} N·m")
    print(f"  电流: {joint_state['current']:.3f} A")
    print(f"  电压: {joint_state['voltage']:.3f} V")
    print(f"  温度: {joint_state['temperature']:.1f}°C")
    print(f"  状态码: {joint_state['status']}")
    print(f"  错误码: {joint_state['err_code']}")

# right effector status
right_state = end_state['right_end_state']
print(f"\n 右执行器控制状态: {right_state['controlled']}")
print(f"右执行器类型: {right_state['type']}")
print(f"右执行器关节: {right_state['names']}")

# 检查执行器状态
print("\n=== 执行器状态检查 ===")
for side in ['left', 'right']:
    state = end_state[f'{side}_end_state']
    if state['controlled']:
        print(f"✓ {side}执行器正在控制中")
    else:
        print(f"✗ {side}执行器未控制")

    for joint_state in state['end_states']:
        if joint_state['err_code'] == 0:
            print(f"✓ {side}执行器关节{joint_state['id']}正常")
        else:
            print(f"✗ {side}执行器关节{joint_state['id']}错误码:
{joint_state['err_code']}")
```

3. `direct_move()`

- **Function**: Control the direct movement of the end effector
- **Parameter**:
  - `key` (str): Control type, such as "gripper"
  - `action` (dict): Dictionary of action parameters
- **action Parameter Structure**:

Parameter Name	Type	Description
`"joint_position"`	`float`	Joint Position (0.0-1.0)
`"joint_velocity"`	`float`	Joint Velocity (rad/s)
`"joint_effort"`	`float`	Joint Torque (N·m)
`"motor_position"`	`float`	Motor position (usually the same as joint_position)
`"motor_velocity"`	`float`	Motor speed (usually the same as

		joint_velocity)
`"motor_current"`	`float`	Motor Current (A)
`"error_code"`	`int`	Error Code (0 indicates normal)

- **Return Value**: 0 indicates success, -1 indicates failure
- **Example**:

```python
import agibot_gdk

robot_body = agibot_gdk.RobotBody()

# 控制末端执行器
action = {
    "left_ee_state": {
        "joint_position": 0.5,    # (0.0-1.0)
        "joint_velocity": 0.2,    # (rad/s)
        "joint_effort": 0.0,      # (N·m)
        "motor_position": 0.5,    # (same with joint_position)
        "motor_velocity": 0.2,    # (same with joint_velocity)
        "motor_current": 0.0,     # (A)
        "error_code": 0           # (0 is no error state)
    },
    "right_ee_state": {
        "joint_position": 0.5,
        "joint_velocity": 0.2,
        "joint_effort": 0.0,
        "motor_position": 0.5,
        "motor_velocity": 0.2,
        "motor_current": 0.0,
        "error_code": 0
    }
}

result = robot_body.direct_move("gripper", action)
if result == 0:
    print("末端执行器控制成功")
else:
    print("末端执行器控制失败")
```

MotionControl Class

This class encapsulates the high-level functional interfaces for motion control.

4. `get_motion_control_status()`

- **Function**: Obtain the operating status of the whole-body actuators
- **Return Value** : Status object, containing the following properties:

Attribute Name	Type	Description
`mode`	`int`	Sports Mode
`error_code`	`int`	Error code, 0 indicates normal

`error_msg`	`str`	Error Message
`frame_names`	`list`	Joint Name List
`collision_pairs_1`	`list`	Collision Pair List 1
`collision_pairs_2`	`list`	Collision Pair List 2

- **Example**:

```python
import agibot_gdk

motion_control = agibot_gdk.MotionControl()

status = motion_control.get_motion_control_status()
print(f"运动模式: {status.mode}")
print(f"错误码: {status.error_code}")
print(f"错误信息: {status.error_msg}")
print(f"关节数量: {len(status.frame_names)}")
print(f"碰撞对数量: {len(status.collision_pairs_1)}")

# 打印所有关节名称
for i, frame_name in enumerate(status.frame_names):
    print(f"关节 {i}: {frame_name}")
```

5. `joint_control_request()`

- **Function**: Whole-body joint control interface
- **Parameter**:
  - `request` (JointControlReq): Joint control request object
- **JointControlReq Parameter Structure**:

Parameter Name	Type	Description
`uuid`	`str`	Request Unique Identifier
`life_time`	`float`	Request Lifecycle (seconds) # Ensure execution is in place in combination with execution speed
`joint_names`	`list`	Joint Name List
`joint_positions`	`list`	Joint Position List (Radians)
`joint_velocities`	`list`	Joint velocity list (radians/second)

- **Return Value**: 0 indicates success, -1 indicates failure
- **Example**:

```python
import agibot_gdk
import uuid
```

```
robot_body = agibot_gdk.RobotBody()
motion_control = agibot_gdk.MotionControl()

# get joint states
joint_states = robot_body.get_state("body_joint_states")
current_positions = [state['position'] for state in joint_states['states']]
target_positions = current_positions.copy()
target_positions[0] += 0.1  # 第一个关节移动 0.1 弧度

# creat control request
joint_control_request = agibot_gdk.JointControlReq()
joint_control_request.uuid = str(uuid.uuid4())
joint_control_request.life_time = 1.0
joint_control_request.joint_names = [state['name'] for state in
joint_states['states']]
joint_control_request.joint_positions = target_positions
joint_control_request.joint_velocities = [0.3] * len(target_positions)

# send command
result = motion_control.joint_control_request(joint_control_request)
if result == 0:
    print("关节控制命令发送成功")
else:
    print("关节控制命令发送失败")
```

Precautions for Use

1. Initialization Wait: After creating the RobotBody and MotionControl objects, it is recommended to wait for 1 second to ensure the DDS connection is established
2. Joint Name: Please confirm the correctness of the joint name before use, which can be obtained via`get_motion_control_status()`.
3. Position Range: Joint positions should be within the safe range to avoid exceeding mechanical limits
4. Speed Limit: Set a reasonable joint speed to avoid danger caused by excessive speed
5. Lifecycle: Set the request lifecycle reasonably to avoid command expiration
6. Error handling: Check return values promptly and handle possible error conditions

Application Scenarios

- Basic Control: Achieve the basic motion control of the robot
- Status Monitoring: Real-time monitoring of the status of each joint of the robot
- Action Recording: Record and playback robot action sequences
- Precise Control: Achieve high-precision joint position control
- Safety Detection: Monitor abnormal states of robots to ensure safe operation

5.2.2 IMU

Imu Class

This class encapsulates the main data acquisition interface for the IMU sensor.

1. `get_latest_imu()`

- **Function**: Obtain the latest IMU data
- **Parameter**:

Parameter Name	Type	Description
`type`	`ImuType`	IMU Type Enumeration Value
`timeout`	`float`	Timeout (milliseconds)

- **Return Value** : `ImuData` object, containing the following attributes:

Attribute Name	Type	Description	Unit
`timestamp_ns`	`int`	Timestamp of Data Acquisition, with a precision of nanoseconds	nanosecond
`orientation`	`Quaternion`	Orientation Quaternion, representing the orientation of the robot in 3D space	Unitless
`angular_velocity`	`Vector3`	Angular velocity, the angular velocity of the robot on three axes	radians/second
`linear_acceleration`	`Vector3`	Linear acceleration, the linear acceleration of the robot on three axes	m/s²
`orientation_covariance`	`list`	Direction covariance matrix, providing uncertainty information of the data	Unitless
`angular_velocity_covariance`	`list`	Angular velocity covariance matrix, used for filtering algorithms	Unitless
`linear_acceleration_covariance`	`list`	Linear acceleration covariance matrix, used for data fusion	Unitless

Detailed Description of ImuData Object

orientation (quaternion object):

- `x`: Quaternion x component
- `y`: Quaternion y component
- `z`: Quaternion z component
- `w`: Quaternion w component

angular_velocity (angular velocity object):

- `x`: Angular velocity of the X-axis

- `y`: Angular velocity of the Y-axis
- `z`: Angular velocity of the Z-axis

linear_acceleration (Linear Acceleration Object):

- `x`: X-axis acceleration
- `y`: Y-axis acceleration
- `z`: Z-axis acceleration

IMU Type:

- `kImuChest`: Chest IMU
- `kImuFront`: Front IMU
- `kImuBack`: Rear IMU
- `kImuChassis`: Chassis IMU

Example:

```
Python
import time
import agibot_gdk

print("IMU 示例程序")
imu = agibot_gdk.Imu()
time.sleep(2)

for i in range(10):
    # 获取最新 IMU 数据
    imu_data = imu.get_latest_imu(agibot_gdk.ImuType.kImuChest, 1000.0)

    if imu_data is not None:
        print(f"\n--- IMU 数据 #{i+1} ---")
        print(f"时间戳: {imu_data.timestamp_ns}")

        # 方向四元数
        print(f"方向四元数: x={imu_data.orientation.x:.4f}, "
              f"y={imu_data.orientation.y:.4f}, "
              f"z={imu_data.orientation.z:.4f}, "
              f"w={imu_data.orientation.w:.4f}")

        # 角速度
        print(f"角速度: x={imu_data.angular_velocity.x:.4f}, "
              f"y={imu_data.angular_velocity.y:.4f}, "
              f"z={imu_data.angular_velocity.z:.4f}")

        # 线性加速度
        print(f"线性加速度: x={imu_data.linear_acceleration.x:.4f}, "
              f"y={imu_data.linear_acceleration.y:.4f}, "
              f"z={imu_data.linear_acceleration.z:.4f}")

        # 协方差矩阵大小
        print(f"方向协方差矩阵大小: {len(imu_data.orientation_covariance)}")
        print(f"角速度协方差矩阵大小: {len(imu_data.angular_velocity_covariance)}")
        print(f"线性加速度协方差矩阵大小: {len(imu_data.linear_acceleration_covariance)}")
    else:
        print(f"未收到 IMU 数据 #{i+1}")
    time.sleep(1.0)
```

2. `get_imu_nearest()`

- **Function**: Retrieve the nearest IMU data around the specified timestamp
- **Parameter**:

Parameter Name	Type	Description
`type`	`ImuType`	IMU Type Enumeration Value
`timestamp`	`int`	Target timestamp (nanoseconds)
`timeout`	`float`	Timeout (milliseconds)

- **Return Value**: `ImuData` object, with the same structure as `get_latest_imu()`
- **Example**:

```python
import time
import agibot_gdk

imu = agibot_gdk.Imu()
imu_type = agibot_gdk.ImuType.kImuChest

for i in range(10):
    # Get IMU data (in 1s)
    imu_data_nearest = imu.get_imu_nearest(imu_type, imu_data.timestamp_ns-1000000000,
1000.0)

    if imu_data_nearest is not None:
        print(f"✅ 最近 IMU 数据: {imu_data_nearest.timestamp_ns}")
        print(f"方向四元数: x={imu_data_nearest.orientation.x:.4f}, "
              f"y={imu_data_nearest.orientation.y:.4f}, "
              f"z={imu_data_nearest.orientation.z:.4f}, "
              f"w={imu_data_nearest.orientation.w:.4f}")
        print(f"角速度: x={imu_data_nearest.angular_velocity.x:.4f}, "
              f"y={imu_data_nearest.angular_velocity.y:.4f}, "
              f"z={imu_data_nearest.angular_velocity.z:.4f}")
        print(f"线性加速度: x={imu_data_nearest.linear_acceleration.x:.4f}, "
              f"y={imu_data_nearest.linear_acceleration.y:.4f}, "
              f"z={imu_data_nearest.linear_acceleration.z:.4f}")
        print(f"方向协方差矩阵大小: {len(imu_data_nearest.orientation_covariance)}")
        print(f"角速度协方差矩阵大小:
{len(imu_data_nearest.angular_velocity_covariance)}")
        print(f"线性加速度协方差矩阵大小:
{len(imu_data_nearest.linear_acceleration_covariance)}")
    else:
        print(f"❌ 未找到最近的 {imu_type} 数据")
    time.sleep(1.0)
```

Precautions for Use

1. Initialization Wait: After creating the Imu object, it is recommended to wait for 2 seconds to ensure the DDS connection is established
2. Timeout Setting: Set an appropriate timeout period according to actual requirements to avoid long-term blocking
3. Data Validity: Please check whether the returned IMU data is None before use
4. Timestamp Precision: The timestamp unit is nanoseconds, which can be used for precise time

synchronization

5. Covariance Matrix: The covariance matrix provides information about the uncertainty of the data and can be used in filtering algorithms

Application Scenarios

- Pose Detection: Obtain the real-time pose of the robot through quaternions
- Motion Analysis: Analyze the motion state of the robot using angular velocity and linear acceleration
- Navigation and Positioning: Perform SLAM and positioning by integrating data from other sensors
- Balance Control: Achieve the robot's balance control based on IMU data
- Data Fusion: Fuse with other sensor data to improve positioning accuracy

5.2.3 Lidar

Lidar Class

This class encapsulates the main data acquisition interface for lidar sensors.

1. `get_latest_pointcloud()`

- **Function**: Obtain the latest point cloud data
- **Parameter**:

Parameter Name	Type	Description
`type`	`LidarType`	Radar Type Enumeration Value
`timeout`	`float`	Timeout (milliseconds)

- **Return Value**: `PointCloud` object, containing the following attributes:

Attribute Name	Type	Description	Unit
`timestamp_ns`	`int`	Timestamp of Point Cloud Data Acquisition	nanosecond
`width`	`int`	Width (number of points) of the point cloud	Points
`height`	`int`	Height (number of points) of the point cloud	Points
`point_step`	`int`	Number of bytes occupied by each point	ByteDance
`row_step`	`int`	Number of bytes occupied per row	ByteDance
`is_bigendian`	`bool`	Whether the data is in big-endian order	Boolean value
`is_dense`	`bool`	Is it a dense point cloud (without invalid points)?	Boolean value

`fields`	`list`	Field information list, defining the attribute structure of each point in the point cloud	Unitless
`data`	`bytes`	Original binary data of the point cloud	ByteDance

Detailed Description of PointCloud Object

fields (field information list):

Each field contains the following attributes:

- `name`: Field name (e.g., "x", "y", "z", "intensity")
- `offset`: Offset in point data
- `datatype`: Data Type
- `count`: The number of elements in this field

data (original data source):

- **Type**: `bytes`
- **Description**: Original binary data of the point cloud
- **Note**: Parsing needs to be performed based on fields information

Radar Type :

- `kLidarChest`: Chest Microwave Radar
- `kLidarFront`: Front Radar
- `kLidarBack`: Rear Radar
- **Example**:

```
Python
import agibot_gdk

lidar = agibot_gdk.Lidar()
pointcloud = lidar.get_latest_pointcloud(agibot_gdk.LidarType.kLidarChest, 1000.0)

if pointcloud is not None:
    print(f"✓ 时间戳: {pointcloud.timestamp_ns}")
    print(f"点云尺寸: {pointcloud.width} x {pointcloud.height}")
    print(f"点步长: {pointcloud.point_step}")
    print(f"行步长: {pointcloud.row_step}")
    print(f"是否大端序: {pointcloud.is_bigendian}")
    print(f"是否密集: {pointcloud.is_dense}")

    # 打印字段信息
    print(f"字段数量: {len(pointcloud.fields)}")
    for j, field in enumerate(pointcloud.fields):
        print(f"  字段 {j+1}: {field.name} (偏移: {field.offset}, "
              f"类型: {field.datatype}, 数量: {field.count})")
else:
    print("未获取到点云数据")
```

2. `get_nearest_pointcloud()`

- **Function**: Obtain the nearest point cloud data near the specified timestamp

- **Parameter**:

Parameter Name	Type	Description
`type`	`LidarType`	Radar Type Enumeration Value
`timestamp`	`int`	Target timestamp (nanoseconds)
`timeout`	`float`	Timeout (milliseconds)

- **Return Value**: `PointCloud` object, with the same structure as `get_latest_pointcloud()`
- **Example**:

```python
import agibot_gdk

lidar = agibot_gdk.Lidar()

# 先获取最新点云数据
pointcloud = lidar.get_latest_pointcloud(agibot_gdk.LidarType.kLidarChest, 1000.0)

if pointcloud is not None:
    # 获取历史点云数据（往前 1 秒）
    pointcloud_nearest = lidar.get_nearest_pointcloud(
        agibot_gdk.LidarType.kLidarChest,
        pointcloud.timestamp_ns - 1000000000,
        1000.0
    )

    if pointcloud_nearest is not None:
        print(f"✅ 最近点云数据: {pointcloud_nearest.timestamp_ns}")
        print(f"点云尺寸: {pointcloud_nearest.width} x {pointcloud_nearest.height}")
        print(f"点步长: {pointcloud_nearest.point_step}")
        print(f"行步长: {pointcloud_nearest.row_step}")
    else:
        print("❌ 未找到最近的点云数据")
else:
    print("未获取到最新点云数据")
```

Precautions for Use

1. **Initialization Wait**: After creating the Lidar object, it is recommended to wait for 1 second to ensure the DDS connection is established
2. **Timeout Setting**: Set an appropriate timeout period based on actual requirements to avoid long-term blocking
3. **Data Validity**: Before use, please check whether the returned point cloud data is None
4. **Timestamp Precision**: The timestamp unit is nanoseconds, which can be used for precise time synchronization
5. **Point Cloud Processing**: Point cloud data volume is large, pay attention to memory usage during processing
6. **Radar Selection** : Select the appropriate radar type (chest/front/rear) based on the application scenario

Application Scenarios

- **SLAM Mapping**: Simultaneous Localization and Mapping using point cloud data
- **Obstacle Detection**: Real-time detection of obstacles in the environment

- **Navigation Obstacle Avoidance**: Provides environmental perception information for robot navigation
- **Environmental Modeling**: Constructing a 3D Environmental Model
- **Object Recognition**: Perform object detection and recognition in combination with point cloud data
- **Path Planning**: Plan a safe path based on point cloud data
- **Data Fusion**: Fuse with other sensor data to improve perception accuracy

5.2.4 Camera

Camera Class

This class encapsulates the main data acquisition interface for camera sensors.

1. `get_latest_image()`

- **Function**: Obtain the latest image data
- **Parameter**:

Parameter Name	Type	Description
`type`	`CameraType`	Camera Type Enumeration Value
`timeout`	`float`	Timeout (milliseconds)

- **Return Value**: `Image` object, containing the following properties:

Attribute Name	Type	Description	Unit
`timestamp_ns`	`int`	Timestamp of image acquisition	nanosecond
`width`	`int`	Image width (pixels)	pixel
`height`	`int`	Height of the image (pixels)	pixel
`encoding`	`str`	Encoding format of the image	string
`color_format`	`str`	Color format of the image	string
`bit_depth`	`int`	Bits per Pixel	bit
`data`	`bytes`	Original pixel data of the image	ByteDance

Detailed description of Image object

Encoding (Encoding Format):

- **Type**: `str`
- **Common Values**:

- `"UNCOMPRESSED"`: Uncompressed
- `"JPEG"`: jpeg image
- `"PNG"`: PNG image

color_format (Color Format):

- **Type**: `str`
- **Common Values**:
  - `"RGB"`: Red, Green, Blue
  - `"BGR"`: Blue Green Red
  - `"GRAY8"`: Canary Release

bit_depth (Bit Depth):

- **Type**: `int`
- **Common Values**:
  - `8`: 8-bit (0-255)
  - `16`: 16-bit (0-65535)
  - `32`: 32-bit (floating point)

data (image data):

- **Type**: `bytes`
- **Description**: Original pixel data of the image
- **Usage**: Image processing, display, and storage
- **Note**: Parsing needs to be performed based on encoding and size

Camera Type:

- `kHeadCenterFisheye`: Head Center Fisheye Camera
- `kHeadLeftFisheye`: Head Left Fisheye Camera
- `kHeadRightFisheye`: Head Right Fisheye Camera
- `kHeadStereoLeft`: Head Stereo Left Camera
- `kHeadStereoRight`: Head Stereo Right Camera
- `kHandLeft`: Left Hand Camera
- `kHandRight`: Right Hand Camera
- `kHeadColor`: Head Color Camera
- `kHeadDepth`: Head depth camera (output is depth map)
- **Example**:

```
Python
import agibot_gdk
import time

camera = agibot_gdk.Camera()
time.sleep(3)  # it is necessary for camare init

image = camera.get_latest_image(agibot_gdk.CameraType.kHeadStereoLeft, 1000.0)
if image is not None:
    print(f"✓ 时间戳: {image.timestamp_ns}")
    print(f"图像尺寸: {image.width} x {image.height}")
    print(f"编码格式: {image.encoding}")
    print(f"颜色格式: {image.color_format}")
```

```
    print(f"位深度: {image.bit_depth}")
else:
    print("未获取到图像数据")
```

2. `get_nearest_image()`

- **Function**: Retrieve the nearest image data around the specified timestamp
- **Parameter**:

Parameter Name	Type	Description
`type`	`CameraType`	Camera Type Enumeration Value
`timestamp`	`int`	Target timestamp (nanoseconds)
`timeout`	`float`	Timeout (milliseconds)

- **Return Value**: `Image` object, with the same structure as `get_latest_image()`
- **Example**:

```
Python
import agibot_gdk
import time

camera = agibot_gdk.Camera()
time.sleep(3)

image = camera.get_latest_image(agibot_gdk.CameraType.kHeadStereoLeft, 1000.0)

if image is not None:

    image_nearest = camera.get_nearest_image(
        agibot_gdk.CameraType.kHeadStereoLeft,
        image.timestamp_ns - 1000000000,
        1000.0
    )

    if image_nearest is not None:
        print(f"✅ 最近图像数据: {image_nearest.timestamp_ns}")
        print(f"图像尺寸: {image_nearest.width} x {image_nearest.height}")
        print(f"编码格式: {image_nearest.encoding}")
    else:
        print("❌ 未找到最近的图像数据")
else:
    print("未获取到最新图像数据")
```

3. `get_image_shape()`

- **Function**: Get image data size
- **Parameter**:

Parameter Name	Type	Description
`type`	`CameraType`	Camera Type Enumeration Value

- **Return value** : `tuple` , a tuple containing the image width and height `(width, height)`
- **Example**:

```python
import agibot_gdk

camera = agibot_gdk.Camera()

shape = camera.get_image_shape(agibot_gdk.CameraType.kHeadStereoLeft)
print(f"图像尺寸: {shape[0]} x {shape[1]}")
```

4. `get_image_fps()`

- **Function**: Get image capture frame rate
- **Parameter**:

Parameter Name	Type	Description
`type`	`CameraType`	Camera Type Enumeration Value

- **Return Value** : `float` , Image Frame Rate (FPS)
- **Example**:

```python
import agibot_gdk

camera = agibot_gdk.Camera()

fps = camera.get_image_fps(agibot_gdk.CameraType.kHeadStereoLeft)
print(f"图像帧率: {fps} FPS")
```

5. `get_image_latency()`

- **Function**: Obtain image latency statistics
- **Parameter**:

Parameter Name	Type	Description
`type`	`CameraType`	Camera Type Enumeration Value
`window_seconds`	`float`	Statistical Window Time (seconds)

- **Return Value** : `LatencyStats` object, containing latency statistics
- **Example**:

```python
import agibot_gdk

camera = agibot_gdk.Camera()

latency = camera.get_image_latency(agibot_gdk.CameraType.kHeadStereoLeft, 1.0)
print(f"最大延迟: {latency.max_latency_ms} ms")
print(f"平均延迟: {latency.avg_latency_ms} ms")
```

Precautions for Use

1. **Initialization Wait**: After creating the Camera object, it is recommended to wait 3 seconds to ensure the camera initialization is completed
2. **Timeout Setting**: Set an appropriate timeout period based on actual requirements to avoid long-term blocking
3. **Data Validity**: Please check whether the returned image data is None before use
4. **Timestamp Precision**: The timestamp unit is nanoseconds, which can be used for precise time synchronization
5. **Image Processing** : The amount of image data is large, so pay attention to memory usage during processing
6. **Camera Selection**: Select the appropriate camera type (fisheye/stereo/depth, etc.) based on the application scenario
7. **Frame Rate Control**: Pay attention to the camera's frame rate limit to avoid excessive requests

Application Scenarios

- **Object Detection**: Use image data for object recognition and detection
- **Visual Navigation**: Provides visual information for robot navigation
- **SLAM Mapping**: Simultaneous Localization and Mapping (SLAM) combined with image data
- **Environmental Monitoring**: Real-time monitoring of changes in the surrounding environment
- **Depth Perception**: Use depth cameras to obtain 3D environment information
- **Stereo Vision** : Distance measurement using binocular cameras
- **Image Recognition** : Perform object classification and recognition
- **Data Fusion**: Fuse with other sensor data to improve perception accuracy

5.2.5 Slam

Slam Class

This class encapsulates the main functional interfaces of the SLAM system.

1. `get_slam_state()`

- **Function**: Get the current state of the SLAM system
- **Parameter**: None
- **Return Value** : SLAM Status Information

Attribute Name	Type	Description
`state`	`int`	SLAM Status Code
`message`	`str`	Status description information

- **Example**:

```python
import agibot_gdk

slam = agibot_gdk.Slam()

# 获取 SLAM 状态
slam_state = slam.get_slam_state()
```

```
print(f"SLAM 状态: {slam_state}")
```

2. `start_mapping()`

- **Function**: Start mapping mode
- **Parameter**: None
- **Return Value** : Operation Result
- **Example**:

```
Python
import agibot_gdk
import time

slam = agibot_gdk.Slam()

# 开始建图
result = slam.start_mapping()
print(f"开始建图结果: {result}")

# 检查建图状态
time.sleep(2)
slam_state = slam.get_slam_state()
print(f"建图状态: {slam_state}")
```

3. `stop_mapping()`

- **Function**: Stop Mapping Mode
- **Parameter**: None
- **Return Value** : Operation Result
- **Example**:

```
Python
import agibot_gdk
import time

slam = agibot_gdk.Slam()

# 停止建图
result = slam.stop_mapping()
print(f"停止建图结果: {result}")

# 检查建图状态
time.sleep(2)
slam_state = slam.get_slam_state()
print(f"建图状态: {slam_state}")
```

4. `get_odom_info()`

- **Function**: Obtain odometry information
- **Parameter**: None
- **Return Value** : Odometry Information Object

Attribute Name	Type	Description	Unit

`pose.pose.position.x`	`float`	Current Position X Coordinate	m
`pose.pose.position.y`	`float`	Current Position Y Coordinate	m
`pose.pose.position.z`	`float`	Z coordinate of the current position	m
`pose.pose.orientation.x`	`float`	Current orientation quaternion X component	Unitless
`pose.pose.orientation.y`	`float`	Current orientation quaternion Y component	Unitless
`pose.pose.orientation.z`	`float`	Z component of the current orientation quaternion	Unitless
`pose.pose.orientation.w`	`float`	Current orientation quaternion W component	Unitless

- **Example**:

```
Python
import agibot_gdk
import time

slam = agibot_gdk.Slam()
time.sleep(1)

#
odom_info = slam.get_odom_info()
print(f"Odom pose: ({odom_info.pose.pose.position.x},
{odom_info.pose.pose.position.y}, {odom_info.pose.pose.position.z})")
print(f"Odom orientation: ({odom_info.pose.pose.orientation.x},
{odom_info.pose.pose.orientation.y}, {odom_info.pose.pose.orientation.z},
{odom_info.pose.pose.orientation.w})")
```

5. Complete Usage Example

- **Function**: Demonstrate the complete usage process of the SLAM module
- **Example**:

```
Python
import agibot_gdk
import time

# 初始化 SLAM 模块
slam = agibot_gdk.Slam()
time.sleep(1)

# 开始建图
slam.start_mapping()

# 建图过程中监控状态和位置
```

```
for i in range(10):
    odom_info = slam.get_odom_info()
    print(f"Odom pose: ({odom_info.pose.pose.position.x},
{odom_info.pose.pose.position.y}, {odom_info.pose.pose.position.z})")
    print(f"Odom orientation: ({odom_info.pose.pose.orientation.x},
{odom_info.pose.pose.orientation.y}, {odom_info.pose.pose.orientation.z},
{odom_info.pose.pose.orientation.w})")
    slam_state = slam.get_slam_state()
    print(f"SLAM state: {slam_state}")
    time.sleep(1)

# 获取当前地图信息
map = agibot_gdk.Map()
time.sleep(1)
map_name = map.get_curr_map()
print(f"Current map: {map_name.id}")

# 停止建图
slam.stop_mapping()
```

Precautions for Use

1. **Initialization Wait**: After creating the Slam object, it is recommended to wait for 1 second to ensure the system initialization is completed
2. **Mapping Environment**: Ensure that the mapping environment has sufficient feature points, and avoid mapping in open or highly repetitive environments
3. **Status Monitoring**: Check the SLAM status in a timely manner to ensure the normal operation of the system
4. **Data Quality**: Ensure the quality of sensor data and avoid mapping when the sensor malfunctions
5. **Computing Resources**: The SLAM algorithm has a large computational load, so pay attention to the usage of system resources

Application Scenarios

- **Environmental Mapping**: Construct a detailed map of the robot's working environment
- **Autonomous Positioning**: Achieve precise positioning in a known environment
- **Navigation Service**: Provides environmental information for path planning
- **Environmental Monitoring**: Real-time monitoring of environmental changes
- **Data Acquisition**: Collect environmental data for subsequent analysis

5.2.6 PNC

Pnc Class

This class encapsulates the main interfaces for robot path planning and navigation control.

1. `get_task_state()`

- **Function**: Get the current task status
- **Parameter**: None
- **Return Value** : Task Status Information

Attribute Name	Type	Description

`state`	`int`	Task Status Code
`message`	`str`	Status description information

- **Example**:

```python
import agibot_gdk

pnc = agibot_gdk.Pnc()

# 获取任务状态
task_state = pnc.get_task_state()
print(f"PNC 任务状态: {task_state}")
```

2. `normal_navi()`

- **Function**: Perform normal navigation to the specified target point
- **Parameter**:

Parameter Name	Type	Description
`target`	`NaviReq`	Navigation Request Object

- **NaviReq Object Structure**:

Attribute Name	Type	Description	Unit
`target.position.x`	`float`	Target Position X Coordinate	m
`target.position.y`	`float`	Target Position Y Coordinate	m
`target.position.z`	`float`	Z Coordinate of Target Position	m
`target.orientation.x`	`float`	X component of the target orientation quaternion	Unitless
`target.orientation.y`	`float`	Y component of the target orientation quaternion	Unitless
`target.orientation.z`	`float`	Z component of the target orientation quaternion	Unitless
`target.orientation.w`	`float`	Quaternion W Component of Target Direction	Unitless

- **Return Value** : Navigation Execution Result

- **Example**:

```python
import agibot_gdk

pnc = agibot_gdk.Pnc()

# 创建导航目标
target = agibot_gdk.NaviReq()
target.target.position.x = -0.14157561800950003
target.target.position.y = 0.015152394013126735
target.target.position.z = 0.0040338473100211417
target.target.orientation.x = 0.01146358383671978
target.target.orientation.y = -0.01720065085681078
target.target.orientation.z = 0.83847410642918951
target.target.orientation.w = 0.54454926012574167

# 执行导航
result = pnc.normal_navi(target)
print(f"导航结果: {result}")
```

Precautions for Use

1. **Initialization waiting**: After creating the Pnc object, it is recommended to wait for 1 second to ensure the system initialization is completed
2. **Target Setting**: Ensure the target position is within the reachable range of the robot
3. **Coordinate System**: Ensure the correct coordinate system (usually the robot body coordinate system) is used
4. **Task Status** : Check task status in a timely manner and handle possible error conditions
5. **Safety Considerations** : Pay attention to the safety of the surrounding environment during navigation

Application Scenarios

- **Autonomous Navigation**: Enables the robot to move autonomously in the environment
- **Path Planning**: Plan the optimal path from the starting point to the end point
- **Task Scheduling**: Manage the execution of multiple navigation tasks
- **Position Control** : Precise control of the robot to reach the specified position and attitude
- **Obstacle Avoidance Navigation**: Achieving Safe Navigation in Dynamic Environments

5.2.7 Map

Map Class

This class encapsulates the main functional interfaces for map management.

1. `get_map()`

- **Function**: Get map information for the specified ID
- **Parameter**:

Parameter Name	Type	Description
`map_id`	`int`	Map ID

- **Return Value** : Map Information Object

Attribute Name	Type	Description
`id`	`int`	Map ID
`name`	`str`	Map Name
`size`	`int`	Map Size (ByteDance)
`created_time`	`str`	Creation Time
`modified_time`	`str`	Modification Time

- **Example**:

```python
import agibot_gdk

map_manager = agibot_gdk.Map()

# 获取地图信息
map_info = map_manager.get_map(0)
print(f"地图信息: {map_info}")
print(f"地图 ID: {map_info.id}")
print(f"地图名称: {map_info.name}")
```

2. `get_curr_map()`

- **Function**: Obtain information about the currently used map
- **Parameter**: None
- **Return Value**: Current map information object, with the same structure as`get_map()`
- **Example**:

```python
import agibot_gdk

map_manager = agibot_gdk.Map()

# 获取当前地图
current_map = map_manager.get_curr_map()
print(f"当前地图: {current_map}")
print(f"当前地图 ID: {current_map.id}")
```

3. `get_all_map()`

- **Function**: Get a list of all available maps
- **Parameter**: None
- **Return Value** : Map List

Attribute Name	Type	Description

`maps`	`list`	Map Information List
`count`	`int`	Number of Maps

- **Example**:

```python
import agibot_gdk

map_manager = agibot_gdk.Map()

# 获取所有地图
all_maps = map_manager.get_all_map()
print(f"所有地图: {all_maps}")
print(f"地图数量: {all_maps.count}")

for i, map_info in enumerate(all_maps.maps):
    print(f"地图 {i}: ID={map_info.id}, 名称={map_info.name}")
```

4. `switch_map()`

- **Function**: Switch to the specified map
- **Parameter**:

Parameter Name	Type	Description
`map_id`	`int`	Target Map ID

- **Return Value** : Switch Result
- **Example**:

```python
import agibot_gdk

map_manager = agibot_gdk.Map()

# 切换到指定地图
result = map_manager.switch_map(1)
print(f"切换地图结果: {result}")

# 验证切换结果
current_map = map_manager.get_curr_map()
print(f"当前地图 ID: {current_map.id}")
```

5. `remove_map()`

- **Function**: Delete the specified map
- **Parameter**:

Parameter Name	Type	Description
`map_id`	`int`	Map ID to be deleted

- **Return Value** : Deletion Result

- **Example**:

```python
import agibot_gdk

map_manager = agibot_gdk.Map()

# 删除指定地图
result = map_manager.remove_map(2)
print(f"删除地图结果: {result}")

# 验证删除结果
all_maps = map_manager.get_all_map()
print(f"剩余地图数量: {all_maps.count}")
```

Precautions for Use

1. **Initialization Waiting**: After creating the Map object, it is recommended to wait for 1 second to ensure the system initialization is completed
2. **Map ID**: Ensure to use a valid map ID and avoid accessing non-existent maps
3. **Map Switching**: Ensure that there are no ongoing navigation tasks when switching maps
4. **Map Deletion**: Before deleting a map, confirm that it is no longer needed, as the deletion operation is irreversible
5. **Storage Space** : Pay attention to the storage space occupied by map files and clean up unnecessary maps in a timely manner

Application Scenarios

- **Map Management**: Manage multiple maps built by the robot
- **Environment Switching** : Switching maps between different working environments
- **Map Backup**: Save and restore important map data
- **Map Sharing**: Share map information among multiple robots
- **Storage Optimization** : Clean up unnecessary maps to save storage space

5.3 C++

All interface declarations need to wait 1-2 seconds for service initialization

```cpp
#include <thread>
#include <chrono>
#include "gdk/gdk.h"

int main()
{
    agibot::gdk::Camera gdk_camera;
    std::this_thread::sleep_for(std::chrono::seconds(2));
}
```

5.3.1 Whole Body Status and Command Interface

MotionControl Class

This class encapsulates the high-level functional interfaces for motion control.

1. `GDKRes GetMotionControlStatus(MotionControlStatus& status)`

- **Function**: Obtain the operating status of the whole-body actuators
- **Parameter**:

Parameter Name	Type	Description
`status`	`MotionControlStatus&`	Output parameter, motion control status object

- **Return Value**: `GDKRes`, the status code of the operation result. Returns `GDKRes::kSuccess` when successful, `status` parameter contains motion control status information

Detailed Description of MotionControlStatus Object

The MotionControlStatus structure contains the following members:

Member Name	Type	Description	Unit
`frame_names`	`std::vector<std::string>`	Joint Name List	List of strings
`frame_poses`	`std::vector<Pose>`	Joint Pose List	Pose List
`collision_pairs_1`	`std::vector<std::string>`	Collision Pair List 1	List of strings
`collision_pairs_2`	`std::vector<std::string>`	Collision Pair List 2	List of strings
`mode`	`uint8_t`	Sports Mode	Unitless
`error_code`	`uint8_t`	Error code, 0 indicates normal	Unitless
`error_msg`	`std::string`	Error Message	string
`twists`	`std::vector<Twist>`	Speed Information List	Speed List
`wrenches`	`std::vector<Wrench>`	Force/Torque Information List	Force/Torque List

```
C++
MotionControlStatus
struct MotionControlStatus {
  std::vector<std::string> frame_names{};
  std::vector<Pose> frame_poses{};
  std::vector<std::string> collision_pairs_1{};
  std::vector<std::string> collision_pairs_2{};
  uint8_t mode{};
  uint8_t error_code{};
  std::string error_msg{};
  std::vector<Twist> twists{};
  std::vector<Wrench> wrenches{};
```

```
};
```

Sample Code

```
C++
#include <csignal>
#include "gdk/gdk.h"
#include <iostream>
#include <chrono>
#include <thread>

void signal_handler(int signum) {
    std::cout << "Interrupt signal (" << signum << ") received.\n";
    exit(signum);
}

int main() {
    agibot::gdk::MotionControl mc;
    std::cout << "motion_control init" << std::endl;

    signal(SIGINT, signal_handler);

    agibot::gdk::MotionControlStatus status;
    std::this_thread::sleep_for(std::chrono::seconds(1));
    mc.GetMotionControlStatus(status);

    std::cout << "mode: " << status.mode << std::endl;
    for (auto& frame_name : status.frame_names) {
        std::cout << "frame_name: " << frame_name << std::endl;
    }
    for (auto& frame_pose : status.frame_poses) {
        std::cout << "frame_pose: x: " << frame_pose.position.x << " y: " <<
frame_pose.position.y << " z: " << frame_pose.position.z << std::endl;
        std::cout << "frame_orientation: x: " << frame_pose.orientation.x << " y: " <<
frame_pose.orientation.y << " z: " << frame_pose.orientation.z << " w: " <<
frame_pose.orientation.w << std::endl;
    }
    for (auto& collision_pair_1 : status.collision_pairs_1) {
        std::cout << "collision_pair_1: " << collision_pair_1 << std::endl;
    }
    for (auto& collision_pair_2 : status.collision_pairs_2) {
        std::cout << "collision_pair_2: " << collision_pair_2 << std::endl;
    }
    std::cout << "error_code: " << status.error_code << std::endl;
    std::cout << "error_msg: " << status.error_msg << std::endl;


    return 0;
}
```

2. `GDKRes JointControl(const JointControlReq& joint_control_req)`

- **Function**: Whole-body joint control interface
- **Parameter**:

Parameter Name	Type	Description
`joint_control_req`	`const JointControlReq&`	Joint control request object

- **Return Value** : `GDKRes` , the status code of the operation result. Returns `GDKRes::kSuccess` when successful

Detailed Description of JointControlReq Object

The JointControlReq structure contains the following members :

Member Name	Type	Description	Unit
`uuid`	`std::string`	Request Unique Identifier	string
`life_time`	`double`	Request Lifecycle	seconds
`joint_names`	`std::vector<std::string>`	Joint Name List	List of strings
`joint_positions`	`std::vector<double>`	Joint Position List	radians
`joint_velocities`	`std::vector<double>`	Joint Velocity List	radians/second
`detail`	`std::string`	Details	string

```
C++
JointControlReq
struct JointControlReq {
  double life_time{0.0};  ///< joint control request lifetime
  std::vector<std::string>
      joint_names{};  ///< joint control request joint names
  std::vector<double>
      joint_positions{};  ///< joint control request joint positions
  std::vector<double> joint_velocities{};

  std::string uuid{};  ///< joint control request uuid, match with rsp
  std::string detail{};
};
```

Sample Code

```
C++
#include <csignal>
#include "gdk/gdk.h"
#include <iostream>
#include <chrono>
#include <thread>
#include <random>
#include <sstream>
#include <vector>

void signal_handler(int signum) {
    std::cout << "Interrupt signal (" << signum << ") received.\n";
    exit(signum);
}

std::string generate_uuid() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<uint32_t> dis(0, 0xFFFFFFFF);
```

```
    std::stringstream ss;
    ss << std::hex;
    ss << dis(gen);
    ss << "-";
    ss << ((dis(gen) & 0xFFFF) | 0x4000); // version 4
    ss << "-";
    ss << ((dis(gen) & 0x0FFF) | 0x4000); // variant
    ss << "-";
    ss << (dis(gen) & 0xFFFF);
    ss << "-";
    ss << dis(gen) << dis(gen);

    return ss.str();
}

int main(int argc, char* argv[]) {
    agibot::gdk::MotionControl mc;
    std::cout << "motion_control init" << std::endl;

    signal(SIGINT, signal_handler);

    std::this_thread::sleep_for(std::chrono::seconds(1));
    if (argc < 3 || argc % 2 == 0) {
        std::cout << "用法: " << argv[0] << " joint_name1 pos1 [joint_name2 pos2 ...]" <<
std::endl;
        return 1;
    }

    agibot::gdk::JointControlReq joint_control;
    joint_control.life_time = 5.0;

    for (int i = 1; i + 1 < argc; i += 2) {
        std::string joint_name = argv[i];
        double joint_pos = std::stod(argv[i + 1]);
        joint_control.joint_names.push_back(joint_name);
        joint_control.joint_positions.push_back(joint_pos);
    }

    joint_control.joint_velocities = {0.3};
    joint_control.uuid = generate_uuid();
    mc.JointControl(joint_control);

    std::cout << "已发送关节控制指令 "  << std::endl;

    return 0;
}
```

3. `GDKRes EndEffectorPoseControl(const EndEffectorPose &end_pose)`

- **Function**: End-effector pose control interface
- Note: This interface does not have collision detection. Please ensure the target position is safe.
- **Parameter**:

Parameter Name	Type	Description
`end_pose`	`const EndEffectorPose&`	End-effector pose control

		object

- **Return Value** : `GDKRes` , the status code of the operation result. Returns `GDKRes::kSuccess` when successful

Detailed Description of EndEffectorPose Object

The EndEffectorPose structure contains the following members :

Member Name	Type	Description	Unit
`life_time`	`double`	Control Lifecycle	seconds
`group`	`int32_t`	cgroup, 0: unknown, 4: left arm, 8: right arm, 12: both arms	Unitless
`left_end_effector_pose`	`Pose`	Left end-effector pose	Pose
`right_end_effector_pose`	`Pose`	Right end-effector pose	Pose

Usage Example

```
C++
#include "gdk/mc.h"
  #include "gdk/types.h"
  #include <iostream>
  #include <chrono>
  #include <thread>
  #include <cmath>
  #include <vector>
  #include <algorithm>

  using namespace agibot::gdk;

  //
  const std::string LEFT_NAME = "arm_l_end_link";
  const std::string RIGHT_NAME = "arm_r_end_link";
  const int32_t CONTROL_GROUP = 12;  // kBothArms

  //
  const Pose TARGET_LEFT = {
      {0.4, 0.2, 0.7},  // position
      {-0.33599605263418142, 0.65399750063564066, -0.57994185990686253,
0.35080102751325265}  // orientation
  };

  const Pose TARGET_RIGHT = {
      {0.6, -0.3, 0.6},  // position
      {0.34224044579728985, 0.61915121432275388, 0.6181931276723881, 0.34257919954626864}
// orientation
  };

  // parameters of MotionControl
  const double MAX_STEP_CM = 0.1;  // cm
  const double LIFETIME = 0.02;    // s
  const double RATE_HZ = 50.0;     // Hz
  const bool HOLD_FINAL = true;    // true is hold the final poses
```

```
class EndEffectorController {
private:
    MotionControl motion_control_;

    // Quaternion SLERP interpolation (strictly corresponds to the Python version)`
    void slerp(const double q0[4], const double q1[4], double t, double result[4]) {
        double dot = q0[0]*q1[0] + q0[1]*q1[1] + q0[2]*q1[2] + q0[3]*q1[3];

        // If the dot product is negative, negate q1 to ensure the shortest path
        if (dot < 0.0) {
            dot = -dot;
            for (int i = 0; i < 4; i++) {
                result[i] = q0[i] + t * (-q1[i] - q0[i]);
            }
        } else {
            for (int i = 0; i < 4; i++) {
                result[i] = q0[i] + t * (q1[i] - q0[i]);
            }
        }

        // Limit the range of dot products
        dot = std::clamp(dot, -1.0, 1.0);

        if (dot > 0.9995) {
            // linear interpolation
            double norm = 0.0;
            for (int i = 0; i < 4; i++) {
                norm += result[i] * result[i];
            }
            norm = std::sqrt(norm);
            if (norm > 0.0) {
                for (int i = 0; i < 4; i++) {
                    result[i] /= norm;
                }
            }
        } else {
            // Spherical linear interpolation
            double theta_0 = std::acos(dot);
            double sin_theta_0 = std::sin(theta_0);
            double theta = theta_0 * t;
            double sin_theta = std::sin(theta);
            double s0 = std::cos(theta) - dot * sin_theta / sin_theta_0;
            double s1 = sin_theta / sin_theta_0;

            for (int i = 0; i < 4; i++) {
                result[i] = s0 * q0[i] + s1 * q1[i];
            }
        }
    }

//Calculate the distance between two points (corresponding to Python version)
    double distanceBetweenPoints(const Position& p1, const Position& p2) {
        double dx = p2.x - p1.x;
        double dy = p2.y - p1.y;
        double dz = p2.z - p1.z;
        return std::sqrt(dx*dx + dy*dy + dz*dz);
    }

//Calculate steps (corresponding to Python version)
    int calculateNSteps(const Position& start, const Position& goal, double max_step_cm)
```

```
{
          double dist_cm = distanceBetweenPoints(start, goal) * 100.0;
          return std::max(static_cast<int>(std::ceil(dist_cm / max_step_cm)), 1);
      }

//Plan trajectory (corresponding to Python version)
      std::vector<Pose> planTrajectory(const Pose& start, const Pose& goal, int n_steps) {
          std::vector<Pose> trajectory;

          for (int i = 0; i < n_steps; i++) {
              double t = static_cast<double>(i) / (n_steps - 1);
              Pose pose;

//Linear interpolation of position
              pose.position.x = start.position.x + t * (goal.position.x -
start.position.x);
              pose.position.y = start.position.y + t * (goal.position.y -
start.position.y);
              pose.position.z = start.position.z + t * (goal.position.z -
start.position.z);

//Quaternion SLERP interpolation
              double q0[4] = {start.orientation.x, start.orientation.y,
start.orientation.z, start.orientation.w};
              double q1[4] = {goal.orientation.x, goal.orientation.y, goal.orientation.z,
goal.orientation.w};
              double result[4];
              slerp(q0, q1, t, result);

              pose.orientation.x = result[0];
              pose.orientation.y = result[1];
              pose.orientation.z = result[2];
              pose.orientation.w = result[3];

              trajectory.push_back(pose);
          }

          return trajectory;
      }

//Search for poses by name (corresponding to Python version)
      Pose findPoseByName(const std::vector<std::string>& frame_names,
                          const std::vector<Pose>& frame_poses,
                          const std::string& target_name) {
          for (size_t i = 0; i < frame_names.size(); i++) {
              if (frame_names[i] == target_name) {
                  return frame_poses[i];
              }
          }
          throw std::runtime_error("Frame name " + target_name + " not found");
      }

  public:
//Execute end effector control (strictly corresponding to Python version)
      void executeEndPoseControl() {
          std::this_thread::sleep_for(std::chrono::milliseconds(1000));  // 等待 1 秒

            //Get current status
          MotionControlStatus status;
          GDKRes result = motion_control_.GetMotionControlStatus(status);
          if (result != GDKRes::kSuccess) {
```

```
            std::cout << "获取运动控制状态失败" << std::endl;
            return;
        }

       //Obtain the starting pose
        Pose start_left_pose = findPoseByName(status.frame_names, status.frame_poses,
LEFT_NAME);
        Pose start_right_pose = findPoseByName(status.frame_names, status.frame_poses,
RIGHT_NAME);

       //Calculate steps
        int n_left = calculateNSteps(start_left_pose.position, TARGET_LEFT.position,
MAX_STEP_CM);
        int n_right = calculateNSteps(start_right_pose.position, TARGET_RIGHT.position,
MAX_STEP_CM);
        int n_steps = std::max(n_left, n_right);

        std::cout << "左臂步数: " << n_left << ", 右臂步数: " << n_right << ", 总步数: " <<
n_steps << std::endl;

//Plan the trajectory
        std::vector<Pose> traj_left = planTrajectory(start_left_pose, TARGET_LEFT,
n_steps);
        std::vector<Pose> traj_right = planTrajectory(start_right_pose, TARGET_RIGHT,
n_steps);

//Execution trajectory
        double dt = 1.0 / RATE_HZ;
        for (int i = 0; i < n_steps; i++) {
            EndEffectorPose end_pose;
            end_pose.life_time = LIFETIME;
            end_pose.group = CONTROL_GROUP;
            end_pose.left_end_effector_pose = traj_left[i];
            end_pose.right_end_effector_pose = traj_right[i];

            result = motion_control_.EndEffectorPoseControl(end_pose);
            if (result != GDKRes::kSuccess) {
                std::cout << "控制命令发送失败，步数: " << i << std::endl;
                return;
            }

            std::this_thread::sleep_for(std::chrono::milliseconds(static_cast<int>(dt *
1000)));
        }

//Maintain the final pose (corresponding to Python version)
        if (HOLD_FINAL) {
            std::cout << "进入末端位姿保持（Ctrl+C 结束）..." << std::endl;
            try {
                while (true) {
                    EndEffectorPose end_pose;
                    end_pose.life_time = LIFETIME;
                    end_pose.group = CONTROL_GROUP;
                    end_pose.left_end_effector_pose = traj_left.back();
                    end_pose.right_end_effector_pose = traj_right.back();

                    result = motion_control_.EndEffectorPoseControl(end_pose);
                    if (result != GDKRes::kSuccess) {
                        std::cout << "保持位姿失败" << std::endl;
                        break;
                    }
```

```
std::this_thread::sleep_for(std::chrono::milliseconds(static_cast<int>(dt * 1000)));
                }
            } catch (const std::exception& e) {
                std::cout << "已中断保持: " << e.what() << std::endl;
            }
        }
    }
};

int main() {
    EndEffectorController controller;
    controller.executeEndPoseControl();
    return 0;
}
```

RobotBody Class

This class encapsulates the main motion control interfaces of the robot chassis.

1. `GDKRes GetState(const std::string& key, JointStates& joint_states)`

- **Function**: Get the status values of all body actuators
- **Parameter**:

Parameter Name	Type	Description
`key`	`const std::string&`	State key-value, such as "body_joint_states"
`joint_states`	`JointStates&`	Output parameter, joint state information object

- **Return Value**: `GDKRes`, the status code of the operation result. Returns `GDKRes::kSuccess` when successful, `joint_states` parameter contains status information

The JointControlReq structure contains the following members :

Member Name	Type	Description	Unit
`uuid`	`std::string`	Request Unique Identifier	string
`life_time`	`double`	Request Lifecycle	seconds
`joint_names`	`std::vector<std::string>`	Joint Name List	List of strings
`joint_positions`	`std::vector<double>`	Joint Position List	radians
`joint_velocities`	`std::vector<double>`	Joint Velocity List	radians/second
`detail`	`std::string`	Details	string

```
C++
```

```
JointControlReq
struct JointControlReq {
  double life_time{0.0};  ///< joint control request lifetime
  std::vector<std::string>
      joint_names{};  ///< joint control request joint names
  std::vector<double>
      joint_positions{};  ///< joint control request joint positions
  std::vector<double> joint_velocities{};

  std::string uuid{};  ///< joint control request uuid, match with rsp
  std::string detail{};
};
```

Usage Example

```
C++
#include "gdk/gdk.h"
#include <iostream>
#include <chrono>
#include <thread>

int main()
{
    agibot::gdk::RobotBody robot_body;
    std::cout << "Robot body init" << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(1));

    agibot::gdk::JointStates body_joint_state;
    if (robot_body.GetState("body_joint_states",body_joint_state) !=
agibot::gdk::GDKRes::kSuccess) {
        std::cout << "Failed to get body joint state" << std::endl;
    } else {
        for (auto &item : body_joint_state.states) {
            std::cout << "Body joint state: " << item.name << " " << item.position <<
std::endl;
        }
    }
    return 0;

}
```

2. `GDKRes GetWholeBodyStatus(WholeBodyStatus& whole_body_status);`

- **Function**: Get the status of all body actuators
- **Parameter**:

Parameter Name	Type	Description
`whole_body_status`	`WholeBodyStatus&`	Output parameter, whole body state information object

- **Return Value**: `GDKRes`, the status code of the operation result. Returns `GDKRes::kSuccess` when successful, `whole_body_status` parameter contains whole body status information

Detailed Description of WholeBodyStatus Object

The WholeBodyStatus structure contains the following members :

Member Name	Type	Description	Unit
`right_arm_error`	`uint32_t`	Right Arm Error Code	Unitless
`left_arm_error`	`uint32_t`	Left Arm Error Code	Unitless
`right_arm_control`	`bool`	Right Arm Control Status	Boolean value
`left_arm_control`	`bool`	Left Arm Control Status	Boolean value
`right_arm_estop`	`bool`	Right arm emergency stop state	Boolean value
`left_arm_estop`	`bool`	Left arm emergency stop state	Boolean value
`right_end_error`	`uint32_t`	Right end-effector error code	Unitless
`left_end_error`	`uint32_t`	Left End-Effector Error Code	Unitless
`right_end_model`	`std::string`	Right end effector model	string
`left_end_model`	`std::string`	Left end effector model	string
`waist_error`	`uint32_t`	Lumbar Error Code	Unitless
`lift_error`	`uint32_t`	Lift Error Code	Unitless
`neck_error`	`uint32_t`	Neck Error Code	Unitless
`chassis_error`	`uint32_t`	Chassis error code	Unitless
`timestamp`	`uint64_t`	timestamp	nanosecond

C++
WholeBodyStatus

```
struct WholeBodyStatus {
  uint32_t right_arm_error{0};
  uint32_t left_arm_error{0};
  bool right_arm_control{false};
  bool left_arm_control{false};
  bool right_arm_estop{false};
  bool left_arm_estop{false};
  uint32_t right_end_error{0};
  uint32_t left_end_error{0};

  std::string right_end_model{};  ///< right end effector type
  std::string left_end_model{};   ///< left end effector type

  uint32_t waist_error{0};  ///< waist state
```

```
  uint32_t lift_error{0};  ///< lift state

  uint32_t neck_error{0};  ///< neck state

  uint32_t chassis_error{0};  ///< chassis state

  uint64_t timestamp{};
};
```

Usage Example

```
C++
#include "gdk/gdk.h"
#include <iostream>
#include <chrono>
#include <thread>

int main()
{
    agibot::gdk::RobotBody robot_body;
    std::cout << "Robot body init" << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(1));

    agibot::gdk::WholeBodyStatus whole_body_status;
    if (robot_body.GetWholeBodyStatus(whole_body_status) != agibot::gdk::GDKRes::kSuccess)
{
        std::cout << "Failed to get whole body status" << std::endl;
    } else {
        std::cout << "right_arm_error: " << whole_body_status.right_arm_error << std::endl;
        std::cout << "left_arm_error: " << whole_body_status.left_arm_error << std::endl;
        std::cout << "right_arm_control: " << whole_body_status.right_arm_control <<
std::endl;
        std::cout << "left_arm_control: " << whole_body_status.left_arm_control <<
std::endl;
        std::cout << "right_arm_estop: " << whole_body_status.right_arm_estop << std::endl;
        std::cout << "left_arm_estop: " << whole_body_status.left_arm_estop << std::endl;
        std::cout << "right_end_error: " << whole_body_status.right_end_error << std::endl;
        std::cout << "left_end_error: " << whole_body_status.left_end_error << std::endl;
        std::cout << "waist_error: " << whole_body_status.waist_error << std::endl;
        std::cout << "neck_error: " << whole_body_status.neck_error << std::endl;
        std::cout << "lift_error: " << whole_body_status.lift_error << std::endl;
        std::cout << "chassis_error: " << whole_body_status.chassis_error << std::endl;
        std::cout << "timestamp: " << whole_body_status.timestamp << std::endl;
        std::cout << "Whole body status: " << whole_body_status.left_end_model<< " " <<
whole_body_status.right_end_model << std::endl;
    }
    return 0;
}
```

3. `GDKRes GetEndState(const std::string& key, DualEndState& end_state)`

- **Function**: Get end-effector status
- **Parameter**:

Parameter Name	Type	Description
`key`	`const std::string&`	State key-value, such as

		"end_states"
`end_state`	`DualEndState&`	Output parameter, double-end effector status information object

- **Return Value**: `GDKRes`, the status code of the operation result. Returns `GDKRes::kSuccess` when successful, `end_state` parameter contains the end-effector status information

Detailed Description of DualEndState Object

The DualEndState structure contains the following members:

Member Name	Type	Description	Unit
`left_end_state`	`EndState`	Left End-Effector Status	Terminal State
`right_end_state`	`EndState`	Right End-Effector Status	Terminal State

```
C++
DualEndState
struct DualEndState {
  EndState left_end_state{};
  EndState right_end_state{};
};
```

EndState Structure:

Member Name	Type	Description	Unit
`controlled`	`bool`	Is it controlled?	Boolean value
`type`	`uint32_t`	End effector type	Unitless
`names`	`std::vector<std::string>`	Joint Name List	List of strings
`end_states`	`std::vector<MotorState>`	Motor Status List	Motor Status List

```
C++
EndState
struct EndState {
  bool controlled{};
  uint32_t type{};
  std::vector<std::string> names{};
  std::vector<MotorState> end_states{};
};
```

MotorState Structure:

Member Name	Type	Description	Unit
`id`	`uint32_t`	Motor ID	Unitless

`enable`	`bool`	Enable or not	Boolean value
`position`	`double`	Motor Position	radians
`velocity`	`double`	Motor Speed	radians/second
`effort`	`double`	Motor Torque	N·m
`current`	`float`	Motor Current	A
`voltage`	`float`	Motor Voltage	V
`temperature`	`float`	Motor Temperature	°C
`status`	`uint32_t`	Motor Status	Unitless
`err_code`	`uint32_t`	Error Code	Unitless

```
C++
MotorState
struct MotorState {
  uint32_t id = 0;
  bool enable = false;
  double position = 0.0;
  double velocity = 0.0;
  double effort = 0.0;
  float current = 0.0f;
  float voltage = 0.0f;
  float temperature = 0.0f;
  uint32_t status = 0;
  uint32_t err_code = 0;
};
```

4. `GDKRes DirectMove(const std::string& key, JointStates& joint_states)`

- **Function**: Control the direct movement of the end effector
- **Parameter**:

Parameter Name	Type	Description
key	`const std::string&`	Control type, such as "gripper"
`joint_states`	`JointStates&`	Joint state control parameters

```
C++
JointStates
struct JointStates {
  size_t nums{};                    ///< number of joint states
  std::vector<JointState> states{};  ///< joint states
  uint64_t timestamp{};             ///< joint states timestamp(ns)
};
```

Usage Example

```
C++
#include "gdk/gdk.h"
#include <iostream>
#include <chrono>
#include <thread>

int main()
{
    agibot::gdk::RobotBody robot_body;
    std::cout << "Robot body init" << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(1));

    agibot::gdk::JointStates move_joint_state;
    move_joint_state.states.resize(2);
    move_joint_state.states[0].name = "left_gripper_joint1";
    move_joint_state.states[0].position = 0.0;
    move_joint_state.states[1].name = "right_gripper_joint1";
    move_joint_state.states[1].position = 0.0;
    if (robot_body.DirectMove("gripper",move_joint_state) != agibot::gdk::GDKRes::kSuccess)
{
        std::cout << "Failed to direct move" << std::endl;
    } else {
        std::cout << "Direct move success" << std::endl;
    }
    return 0;
}
```

Precautions for Use

1. Initialization Wait: After creating the Imu object, it is recommended to wait for 2 seconds to ensure the DDS connection is established
2. Timeout Setting: Set an appropriate timeout period according to actual requirements to avoid long-term blocking
3. Data Validity: Please check whether the returned IMU data is None before use
4. Timestamp Precision: The timestamp unit is nanoseconds, which can be used for precise time synchronization
5. Covariance Matrix: The covariance matrix provides information about the uncertainty of the data and can be used in filtering algorithms

Application Scenarios

- Pose Detection: Obtain the real-time pose of the robot through quaternions
- Motion Analysis: Analyze the motion state of the robot using angular velocity and linear acceleration
- Navigation and Positioning: Perform SLAM and positioning by integrating data from other sensors
- Balance Control: Achieve the robot's balance control based on IMU data
- Data Fusion: Fuse with other sensor data to improve positioning accuracy

5.3.2 IMU

Imu Class

This class encapsulates the main data acquisition interface for the IMU sensor.

1. `GDKRes Imu::GetLatestImu(const ImuType& imu_type, const float timeout_ms,`

`std::shared_ptr<ImuData>& imu)`

- **Function**: Obtain the latest IMU data
- **Parameter**:

Parameter Name	Type	Description
`imu_type`	`const ImuType&`	IMU Type Enumeration Value
`timeout_ms`	`const float`	Timeout (milliseconds)
`imu`	`std::shared_ptr<ImuData>&`	Output parameter, IMU data pointer

- `ImuData` object, which contains the following attributes:

Attribute Name	Type	Description	Unit
`timestamp_ns`	`int`	Timestamp of Data Acquisition, with a precision of nanoseconds	nanosecond
`orientation`	`Quaternion`	Orientation Quaternion, representing the orientation of the robot in 3D space	Unitless
`angular_velocity`	`Vector3`	Angular velocity, the angular velocity of the robot on three axes	radians/second
`linear_acceleration`	`Vector3`	Linear acceleration, the linear acceleration of the robot on three axes	m/s²
`orientation_covariance`	`list`	Direction covariance matrix, providing uncertainty information of the data	Unitless
`angular_velocity_covariance`	`list`	Angular velocity covariance matrix, used for filtering algorithms	Unitless
`linear_acceleration_covariance`	`list`	Linear acceleration covariance matrix, used for data fusion	Unitless

```
C++
ImuData
struct ImuData {
  Orientation orientation{};                    ///< imu current orientation
  std::vector<double> orientation_covariance{};  ///< imu orientation covariance
  Vector3 angular_velocity{};  ///< imu current angular velocity
  std::vector<double>
```

```
      angular_velocity_covariance{};  ///< imu angular velocity covariance
  Vector3 linear_acceleration{};      ///< imu current linear acceleration
  std::vector<double>
      linear_acceleration_covariance{};  ///< imu linear acceleration covariance
  uint64_t timestamp_ns{0};              ///< imu timestamp(ns)
};
```

Detailed Description of ImuData Object

orientation (quaternion object):

- `x`: Quaternion x component
- `y`: Quaternion y component
- `z`: Quaternion z component
- `w`: Quaternion w component

```
C++
Orientation
struct Orientation {
  double x{};
  double y{};
  double z{};
  double w{};
};
```

angular_velocity (angular velocity object):

- `x`: Angular velocity of the X-axis
- `y`: Angular velocity of the Y-axis
- `z`: Angular velocity of the Z-axis

```
C++
Vector3
struct Vector3 {
  double x{};
  double y{};
  double z{};
};
```

linear_acceleration (Linear Acceleration Object):

- `x`: X-axis acceleration
- `y`: Y-axis acceleration
- `z`: Z-axis acceleration

The data structure is the same as above

IMU Type:

- `kImuChest`: Chest IMU
- `kImuFront`: Front IMU
- `kImuBack`: Rear IMU
- `kImuChassis`: Chassis IMU

```
C++
ImuType
enum class ImuType {
```

```
  kImuUnknown = 0,  ///< unknown imu
  kImuChest = 1,    ///< chest imu
  kImuFront = 2,    ///< front imu
  kImuBack = 3,     ///< back imu
  kImuChassis = 4,  ///< chassis imu
};
```

Example:

```
C++
#include <iostream>
#include <chrono>
#include <thread>
#include "gdk/gdk.h"

int main()
{
    std::cout<< "IMU 示例程序" << std::endl;
    agibot::gdk::Imu imu;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Wait for 2 seconds to ensure
DDS connection is established
    std::shared_ptr<agibot::gdk::ImuData> imu_data;
    imu.GetLatestImu(agibot::gdk::ImuType::kImuChassis, 500.0, imu_data);

    if (imu_data != nullptr) {
        std::cout << "\n--- IMU 数据 ---" << std::endl;
        std::cout << "时间戳: " << imu_data->timestamp_ns << std::endl;

//Directional quaternion
        std::cout << "方向四元数: x=" << imu_data->orientation.x << ", "
                  << "y=" << imu_data->orientation.y << ", "
                  << "z=" << imu_data->orientation.z << ", "
                  << "w=" << imu_data->orientation.w << std::endl;

//Angular velocity
        std::cout << "角速度: x=" << imu_data->angular_velocity.x << ", "
                  << "y=" << imu_data->angular_velocity.y << ", "
                  << "z=" << imu_data->angular_velocity.z << std::endl;

//Linear acceleration
        std::cout << "线性加速度: x=" << imu_data->linear_acceleration.x << ", "
                  << "y=" << imu_data->linear_acceleration.y << ", "
                  << "z=" << imu_data->linear_acceleration.z << std::endl;

//Covariance matrix size
        std::cout << "方向协方差矩阵大小: " << imu_data->orientation_covariance.size() <<
std::endl;
        std::cout << "角速度协方差矩阵大小: " << imu_data->angular_velocity_covariance.size()
<< std::endl;
        std::cout << "线性加速度协方差矩阵大小: " << imu_data-
>linear_acceleration_covariance.size() << std::endl;
    } else {
        std::cout << "未收到 IMU 数据" << std::endl;
    }

    return 0;
}
```

2. GDKRes GetNearestImu(const ImuType& *imu_type*, const uint64_t *timestamp_ns*, const

`float timeout_ms, std::shared_ptr<ImuData>& imu);`

- **Function**: Retrieve the nearest IMU data around the specified timestamp
- **Parameter**:

Parameter Name	Type	Description
`type`	`ImuType`	IMU Type Enumeration Value
`timestamp`	`int`	Target timestamp (nanoseconds)
`timeout`	`float`	Timeout (milliseconds)

- **Return Value**: `ImuData` object, with the same structure as `GetLatestImu()`.
- **Example**:

```
C++
#include <iostream>
#include <iomanip>
#include <chrono>
#include <thread>
#include "gdk/gdk.h"

int main()
{
    std::cout << "IMU 示例程序" << std::endl;
    agibot::gdk::Imu imu;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // 等待 1 秒以确保 DDS 连接建立

    agibot::gdk::ImuType imu_type = agibot::gdk::ImuType::kImuChassis;
    std::shared_ptr<agibot::gdk::ImuData> imu_data;
    imu.GetLatestImu(imu_type, 500.0, imu_data);

    if (imu_data != nullptr) {
        std::cout << "\n--- IMU 数据 ---" << std::endl;
        std::cout << "时间戳: " << imu_data->timestamp_ns << std::endl;

        // 方向四元数
        std::cout << "方向四元数: x=" << imu_data->orientation.x << ", "
                  << "y=" << imu_data->orientation.y << ", "
                  << "z=" << imu_data->orientation.z << ", "
                  << "w=" << imu_data->orientation.w << std::endl;

        // 角速度
        std::cout << "角速度: x=" << imu_data->angular_velocity.x << ", "
                  << "y=" << imu_data->angular_velocity.y << ", "
                  << "z=" << imu_data->angular_velocity.z << std::endl;

        // 线性加速度
        std::cout << "线性加速度: x=" << imu_data->linear_acceleration.x << ", "
                  << "y=" << imu_data->linear_acceleration.y << ", "
                  << "z=" << imu_data->linear_acceleration.z << std::endl;

        // 协方差矩阵大小
        std::cout << "方向协方差矩阵大小: " << imu_data->orientation_covariance.size()
<< std::endl;
        std::cout << "角速度协方差矩阵大小: " << imu_data-
```

```
>angular_velocity_covariance.size() << std::endl;
        std::cout << "线性加速度协方差矩阵大小: " << imu_data-
>linear_acceleration_covariance.size() << std::endl;

        // 查找最近的 IMU 数据
        for (int i = 0; i < 10; ++i) {
            std::shared_ptr<agibot::gdk::ImuData> imu_data_nearest;
            agibot::gdk::GDKRes res = imu.GetNearestImu(
                imu_type,
                imu_data->timestamp_ns - 1000000000LL, // 往前 1 秒
                1000.0,
                imu_data_nearest
            );
            if (res == agibot::gdk::GDKRes::kSuccess && imu_data_nearest != nullptr) {
                std::cout << "✅ 最近 IMU 数据: " << imu_data_nearest->timestamp_ns <<
std::endl;
                std::cout << std::fixed << std::setprecision(4);
                std::cout << "方向四元数: x=" << imu_data_nearest->orientation.x
                          << ", y=" << imu_data_nearest->orientation.y
                          << ", z=" << imu_data_nearest->orientation.z
                          << ", w=" << imu_data_nearest->orientation.w << std::endl;
                std::cout << "角速度: x=" << imu_data_nearest->angular_velocity.x
                          << ", y=" << imu_data_nearest->angular_velocity.y
                          << ", z=" << imu_data_nearest->angular_velocity.z <<
std::endl;
                std::cout << "线性加速度: x=" << imu_data_nearest-
>linear_acceleration.x
                          << ", y=" << imu_data_nearest->linear_acceleration.y
                          << ", z=" << imu_data_nearest->linear_acceleration.z <<
std::endl;
                std::cout << "方向协方差矩阵大小: " << imu_data_nearest-
>orientation_covariance.size() << std::endl;
                std::cout << "角速度协方差矩阵大小: " << imu_data_nearest-
>angular_velocity_covariance.size() << std::endl;
                std::cout << "线性加速度协方差矩阵大小: " << imu_data_nearest-
>linear_acceleration_covariance.size() << std::endl;
            } else {
                std::cout << "❌ 未找到最近的 IMU 数据" << std::endl;
            }
            std::this_thread::sleep_for(std::chrono::seconds(1));
        }
    } else {
        std::cout << "未收到 IMU 数据" << std::endl;
    }
    return 0;
}
```

Precautions for Use

1. **Initialization Wait**: After creating the Imu object, it is recommended to wait for 2 seconds to ensure the DDS connection is established

2. **Timeout Setting**: Set an appropriate timeout period based on actual requirements to avoid long-term blocking

3. **Data Validity**: Please check whether the returned IMU data is None before use

4. **Timestamp Precision**: The timestamp unit is nanoseconds, which can be used for precise time synchronization

5. **Covariance Matrix** : The covariance matrix provides information about the uncertainty of the data and can be used in filtering algorithms

Application Scenarios

- **Attitude Detection**: Obtain the real-time attitude of the robot through quaternions
- **Motion Analysis** : Analyze the motion state of the robot using angular velocity and linear acceleration
- **Navigation and Positioning**: Perform SLAM and positioning by integrating data from other sensors
- **Balance Control**: Achieve the balance control of the robot based on IMU data
- **Data Fusion**: Fuse with other sensor data to improve positioning accuracy

5.3.3 Lidar

Lidar Class

This class encapsulates the main data acquisition interface for lidar sensors.

1. `GDKRes GetLatestPointCloud(const LidarType& lidar_type,`

```
                            const float timeout_ms,
                            std::shared_ptr<PointCloud>& pointcloud);
```

- **Function**: Obtain the latest point cloud data
- **Parameter**:

Parameter Name	Type	Description
`lidar_type`	`const LidarType&`	LiDAR type enumeration value
`timeout_ms`	`const float`	Timeout (milliseconds)
`pointcloud`	`std::shared_ptr<PointCloud>&`	Output parameter, point cloud data pointer

- `PointCloud` object, which contains the following attributes:

Attribute Name	Type	Description	Unit
`timestamp_ns`	`uint64_t`	Timestamp of Point Cloud Data Acquisition	nanosecond
`width`	`int`	Width (number of points) of the point cloud	Points
`height`	`int`	Height (number of points) of the point cloud	Points
`point_step`	`int`	Number of bytes occupied by each point	ByteDance
`row_step`	`int`	Number of bytes occupied per row	ByteDance
`is_bigendian`	`bool`	Whether the data is in big-endian order	Boolean value

`is_dense`	`bool`	Is it a dense point cloud (without invalid points)?	Boolean value
`fields`	`std::vector<PointField>`	Field information list, defining the attribute structure of each point in the point cloud	Unitless
`data_view`	`DataView`	Original binary data of the point cloud	ByteDance

```
C++
PointCloud
struct PointCloud {
  int width{0};   ///< point cloud width
  int height{0};  ///< point cloud height

  std::vector<PointField> fields{};  ///< point cloud fields
  int point_step{0};                 ///< point step
  int row_step{0};                   ///< row step
  bool is_bigendian{false};          ///< is bigendian
  bool is_dense{true};               ///< is dense

  DataView data_view{};      ///< point cloud data view
  uint64_t timestamp_ns{0};  ///< point cloud timestamp(ns)
};
```

Detailed Description of PointCloud Object

fields (field information list):

Each field contains the following attributes:

- `name`: Field name (e.g., "x", "y", "z", "intensity")
- `offset`: Offset in point data
- `datatype`: Data Type
- `count`: The number of elements in this field

```
C++
PointField
struct PointField {
  std::string name{};   // point field name (x, y, z, intensity, etc.)
  uint32_t offset{0};   // point field offset
  uint8_t datatype{0};  // point field datatype
  uint32_t count{1};    // point field count
};
```

data_view (original data source):

- **Type**: DataView
- **Description**: Original binary data of the point cloud
- **Note**: Parsing needs to be performed based on fields information

```
C++
DataView
class DataView {
```

```
public:
 /// @enum OwnershipType
 /// @brief ownership type
 /// @details ownership type related information
 enum class OwnershipType { OWNED, BORROWED };

 DataView() = default;

 DataView(const void* data, size_t size);

 explicit DataView(const std::vector<uint8_t>& vec);

 DataView(const DataView& other);

 DataView& operator=(const DataView& other);
 DataView(DataView&& other) noexcept;
 DataView& operator=(DataView&& other) noexcept;

 /// @brief data
 /// @details use to get the data of the data view
 /// @return the data of the data view
 const uint8_t* data() const { return data_; }

 /// @brief mutable_data
 /// @details use to get the mutable data of the data view
 /// @return the mutable data of the data view
 uint8_t* mutable_data() { return const_cast<uint8_t*>(data_); }

 /// @brief size
 /// @details use to get the size of the data view
 /// @return the size of the data view
 size_t size() const { return size_; }

 /// @brief IsOwned
 /// @details use to check if the data view is owned
 /// @return true if the data view is owned, false otherwise
 bool IsOwned() const { return ownership_ == OwnershipType::OWNED; }

 /// @brief Clone
 /// @details use to clone the data view
 /// @return the cloned data view
 DataView Clone() const;

 /// @brief CreateOwnedFrom
 /// @details use to create a data view from owned data
 /// @param data the data of the data view, input parameter
 /// @param size the size of the data view, input parameter
 /// @return the created data view
 static DataView CreateOwnedFrom(const void* data, size_t size);

 /// @brief AssignOwnedData
 /// @details use to assign owned data to the data view
 /// @param data the data of the data view, input parameter
 /// @param size the size of the data view, input parameter
 void AssignOwnedData(const void* data, size_t size);

private:
 const uint8_t* data_ = nullptr;
 size_t size_ = 0;
 OwnershipType ownership_ = OwnershipType::BORROWED;
 std::vector<uint8_t> owned_buffer_;
```

```
};
```

Radar Type :

- `kLidarChest`: Chest Microwave Radar
- `kLidarFront`: Front Radar
- `kLidarBack`: Rear Radar

```
C++
enum class LidarType {
  kLidarUnknown = 0,  ///< unknown lidar
  kLidarChest = 1,    ///< chest lidar
  kLidarFront = 2,    ///< front lidar
  kLidarBack = 3,     ///< back lidar
};
```

- **Example**:

```
C++
#include <iostream>
#include <chrono>
#include <thread>
#include <memory>
#include "gdk/gdk.h"

int main() {
    agibot::gdk::Lidar lidar;

//Optional types: kLidarChest (chest radar), kLidarFront (front radar), kLidarBack
(rear radar)
    agibot::gdk::LidarType lidar_type = agibot::gdk::LidarType::kLidarChest;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Wait for 1 seconds to
ensure DDS connection is established
    std::shared_ptr<agibot::gdk::PointCloud> pointcloud =
std::make_shared<agibot::gdk::PointCloud>();
    lidar.GetLatestPointCloud(lidar_type, 500.0, pointcloud);

    if (pointcloud != nullptr) {
        std::cout << "✅ 时间戳: " << pointcloud->timestamp_ns << std::endl;
        std::cout << "点云尺寸: " << pointcloud->width << " x " << pointcloud->height
<< std::endl;
        std::cout << "点步长: " << pointcloud->point_step << std::endl;
        std::cout << "行步长: " << pointcloud->row_step << std::endl;
        std::cout << "是否大端序: " << (pointcloud->is_bigendian ? "是" : "否") <<
std::endl;
        std::cout << "是否密集: " << (pointcloud->is_dense ? "是" : "否") << std::endl;

        // 打印字段信息
        std::cout << "字段数量: " << pointcloud->fields.size() << std::endl;
        for (size_t j = 0; j < pointcloud->fields.size(); ++j) {
            const auto& field = pointcloud->fields[j];
            std::cout << "  字段 " << (j + 1) << ": " << field.name
                      << " (偏移: " << field.offset
                      << ", 类型: " << field.datatype
                      << ", 数量: " << field.count << ")" << std::endl;
        }
    } else {
        std::cout << "未获取到点云数据" << std::endl;
```

```
    }
    return 0;
}
```

2. `GDKRes GetNearestPointCloud(const LidarType& lidar_type, const uint64_t timestamp_ns, const float timeout_ms, std::shared_ptr<PointCloud>& pointcloud);`

- **Function**: Obtain the nearest point cloud data near the specified timestamp
- **Parameter**:

Parameter Name	Type	Description
`type`	`LidarType`	Radar Type Enumeration Value
`timestamp`	`int`	Target timestamp (nanoseconds)
`timeout`	`float`	Timeout (milliseconds)

- **Return value** : `PointCloud` object, with the same structure as `GetLatestPointCloud ()`
- **Example**:

```
C++
#include <iostream>
#include <chrono>
#include <thread>
#include <memory>
#include "gdk/gdk.h"

int main() {
    agibot::gdk::Lidar lidar;

//Optional types: kLidarChest (chest radar), kLidarFront (front radar), kLidarBack
(rear radar)
    agibot::gdk::LidarType lidar_type = agibot::gdk::LidarType::kLidarChest;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Wait for 1 seconds to
ensure DDS connection is established
    std::shared_ptr<agibot::gdk::PointCloud> pointcloud =
std::make_shared<agibot::gdk::PointCloud>();
    lidar.GetLatestPointCloud(lidar_type, 500.0, pointcloud);

    if (pointcloud != nullptr) {
        std::cout << "✅ 时间戳: " << pointcloud->timestamp_ns << std::endl;
        std::cout << "点云尺寸: " << pointcloud->width << " x " << pointcloud->height
<< std::endl;
        std::cout << "点步长: " << pointcloud->point_step << std::endl;
        std::cout << "行步长: " << pointcloud->row_step << std::endl;
        std::cout << "是否大端序: " << (pointcloud->is_bigendian ? "是" : "否") <<
std::endl;
        std::cout << "是否密集: " << (pointcloud->is_dense ? "是" : "否") << std::endl;

        // 打印字段信息
        std::cout << "字段数量: " << pointcloud->fields.size() << std::endl;
        for (size_t j = 0; j < pointcloud->fields.size(); ++j) {
            const auto& field = pointcloud->fields[j];
            std::cout << "  字段 " << (j + 1) << ": " << field.name
                      << " (偏移: " << field.offset
```

```
                    << ", 类型: " << field.datatype
                    << ", 数量: " << field.count << ")" << std::endl;
        }

        // 查找最近的点云数据
        std::shared_ptr<agibot::gdk::PointCloud> pointcloud_nearest =
std::make_shared<agibot::gdk::PointCloud>();
        agibot::gdk::GDKRes res = lidar.GetNearestPointCloud(
            lidar_type,
            pointcloud->timestamp_ns - 1000000000LL, // 往前 1 秒
            1000.0,
            pointcloud_nearest
        );
        if (res == agibot::gdk::GDKRes::kSuccess && pointcloud_nearest != nullptr) {
            std::cout << "✅ 最近点云数据: " << pointcloud_nearest->timestamp_ns <<
std::endl;
            std::cout << "点云尺寸: " << pointcloud_nearest->width << " x " <<
pointcloud_nearest->height << std::endl;
        } else {
            std::cout << "❌ 未找到最近的 Chest 雷达数据" << std::endl;
        }
    } else {
        std::cout << "未获取到点云数据" << std::endl;
    }
    return 0;
}
```

Precautions for Use

1. **Initialization Wait**: After creating the Lidar object, it is recommended to wait for 1 second to ensure the DDS connection is established
2. **Timeout Setting**: Set an appropriate timeout period based on actual requirements to avoid long-term blocking
3. **Data Validity**: Before use, please check whether the returned point cloud data is None
4. **Timestamp Precision**: The timestamp unit is nanoseconds, which can be used for precise time synchronization
5. **Point Cloud Processing**: Point cloud data volume is large, pay attention to memory usage during processing
6. **Radar Selection** : Select the appropriate radar type (chest/front/rear) based on the application scenario

Application Scenarios

- **SLAM Mapping**: Simultaneous Localization and Mapping using point cloud data
- **Obstacle Detection**: Real-time detection of obstacles in the environment
- **Navigation Obstacle Avoidance**: Provides environmental perception information for robot navigation
- **Environmental Modeling**: Constructing a 3D Environmental Model
- **Object Recognition**: Perform object detection and recognition in combination with point cloud data
- **Path Planning**: Plan a safe path based on point cloud data
- **Data Fusion**: Fuse with other sensor data to improve perception accuracy

5.3.4 Camera

Camera Class

This class encapsulates the main data acquisition interface for camera sensors.

1. `GDKRes GetLatestImage(const CameraType& camera_type, const float timeout_ms, std::shared_ptr<Image>& image);`

- **Function**: Obtain the latest image data
- **Parameter**:

Parameter Name	Type	Description
`camera_type`	`const CameraType&`	Camera Type Enumeration Value
`timeout_ms`	`const float`	Timeout (milliseconds)
`image`	`std::shared_ptr<Image>&`	Output parameter, image data pointer

- `Image` object, containing the following properties:

Member Name	Type	Description	Unit
`timestamp_ns`	`uint64_t`	Timestamp of image acquisition	nanosecond
`width`	`uint32_t`	Image width (pixels)	pixel
`height`	`uint32_t`	Height of the image (pixels)	pixel
`encoding`	`Encoding`	Image encoding format enumeration	Enumerated value
`color_format`	`ColorFormat`	Image color format enumeration	Enumerated value
`bit_depth`	`uint8_t`	Bits per Pixel	bit
`data_view`	`DataView`	Original pixel data view of the image	Data View

```
C++
Image
struct Image {
  uint32_t width{0};   ///< image width
  uint32_t height{0};  ///< image height

  enum class Encoding : uint8_t {
    UNCOMPRESSED,  ///< uncompressed
    JPEG,          ///< JPEG
    PNG            ///< PNG
  } encoding{Encoding::UNCOMPRESSED};

  enum class ColorFormat : uint8_t {
    RGB,
    BGR,
    RGBA,
```

```
    BGRA,

    YUV420,
    YUV422,
    YUV444,
    NV12,
    NV21,

    GRAY8,
    GRAY16,

    BAYER_RGGB,
    BAYER_BGGR,
    BAYER_GBRG,
    BAYER_GRBG,

    RS2_FORMAT_Z16
  } color_format{ColorFormat::RGB};

  uint8_t bit_depth{8};  ///< bit depth

  DataView data_view{};      ///< image data view
  uint64_t timestamp_ns{0};  ///< image timestamp(ns)
};
```

Detailed description of Image object

Encoding (Encoding Format):

- **Type**: Encoding
- **Common Values**:
  - `Encoding::UNCOMPRESSED`: Uncompressed
  - `Encoding::JPEG`: JPEG Compression
  - `Encoding::PNG`: PNG Compression

```
C++
Encoding
  enum class Encoding : uint8_t {
    UNCOMPRESSED,  ///< uncompressed
    JPEG,          ///< JPEG
    PNG            ///< PNG
  } encoding{Encoding::UNCOMPRESSED};
```

color_format (Color Format):

- **Type**: ColorFormat
- **Common Values**:
  - `ColorFormat::RGB`: Red, Green, Blue
  - `ColorFormat::BGR`: Blue Green Red
  - `ColorFormat::RGBA`: Red, Green, Blue, Alpha
  - `ColorFormat::BGRA`: Blue Green Red Alpha
  - `ColorFormat::GRAY8`: 8-bit canary release
  - `ColorFormat::GRAY16`: 16-bit canary release
  - `ColorFormat::YUV420`: YUV420 format

- `ColorFormat::YUV422`: YUV422 format
- `ColorFormat::YUV444`: YUV444 format
- `ColorFormat::NV12`: NV12 Format
- `ColorFormat::NV21`: NV21 Format
- `ColorFormat::BAYER_RGGB`: RGGB Bayer Pattern
- `ColorFormat::BAYER_BGGR`: BGGR Bayer pattern
- `ColorFormat::BAYER_GBRG`: GBRG Bayer pattern
- `ColorFormat::BAYER_GRBG`: GRBG Bayer pattern
- `ColorFormat::RS2_FORMAT_Z16`: RealSense Z16 Depth Format

```
C++
ColorFormat
  enum class ColorFormat : uint8_t {
    RGB,
    BGR,
    RGBA,
    BGRA,

    YUV420,
    YUV422,
    YUV444,
    NV12,
    NV21,

    GRAY8,
    GRAY16,

    BAYER_RGGB,
    BAYER_BGGR,
    BAYER_GBRG,
    BAYER_GRBG,

    RS2_FORMAT_Z16
  } color_format{ColorFormat::RGB};
```

bit_depth (Bit Depth):

- **Type**: `int`
- **Common Values**:
  - `8`: 8-bit (0-255)
  - `16`: 16-bit (0-65535)
  - `32`: 32-bit (floating point)

data_view (image data):

- **Type**: DataView
- **Description**: Original pixel data of the image
- **Usage**: Image processing, display, and storage
- **Note**: Parsing needs to be performed based on encoding and size

Camera Type:

- `kHeadCenterFisheye`: Head Center Fisheye Camera
- `kHeadLeftFisheye`: Head Left Fisheye Camera

- `kHeadRightFisheye`: Head Right Fisheye Camera
- `kHeadStereoLeft`: Head Stereo Left Camera
- `kHeadStereoRight`: Head Stereo Right Camera
- `kHandLeft`: Left Hand Camera
- `kHandRight`: Right Hand Camera
- `kHeadColor`: Head Color Camera
- `kHeadDepth`: Head depth camera (output is depth map)
- **Example**:

```
C++
#include <iostream>
#include <thread>
#include "gdk/gdk.h"

int main() {
    agibot::gdk::Camera gdk_camera;
    std::cout << "Camera init" << std::endl;

    std::this_thread::sleep_for(std::chrono::seconds(5));

    std::shared_ptr<agibot::gdk::Image> image =
std::make_shared<agibot::gdk::Image>();
    // 相机种类
    // kHeadCenterFisheye（头部中间鱼眼相机）
    // kHeadLeftFisheye（头部左侧鱼眼相机）
    // kHeadRightFisheye（头部右侧鱼眼相机）
    // kHeadStereoLeft（头部立体左相机）
    // kHeadStereoRight（头部立体右相机）
    // kHandLeft（左手相机）
    // kHandRight（右手相机）
    // kHeadColor（头部彩色相机）
    // kHeadDepth（头部深度相机）输出为深度图
    agibot::gdk::CameraType camera_type = agibot::gdk::CameraType::kHandLeft;

    if (gdk_camera.GetLatestImage(camera_type, 500, image) !=
agibot::gdk::GDKRes::kSuccess) {
        std::cout << "Failed to get latest image" << std::endl;
    } else {
        std::cout << "Image shape: " << image->width << "x" << image->height <<
std::endl;
    }
    return 0;
}
```

2. `GDKRes GetNearestImage(const CameraType& camera_type, const uint64_t timestamp_ns, const float timeout_ms, std::shared_ptr<Image>& image);`

- **Function**: Retrieve the nearest image data around the specified timestamp
- **Parameter**:

Parameter Name	Type	Description
`camera_type`	`const CameraType&`	Camera Type Enumeration Value

`timestamp_ns`	`const uint64_t`	Target timestamp (nanoseconds)
`timeout_ms`	`const float`	Timeout (milliseconds)
`image`	`std::shared_ptr<Image>&`	Output parameter, image data pointer

- **Return Value** : `Image` object, with the same structure as `GetLatestImage ()`
- **Example**:

```
C++
#include <iostream>
#include <thread>
#include "gdk/gdk.h"

int main() {
    agibot::gdk::Camera gdk_camera;
    std::cout << "Camera init" << std::endl;

    std::this_thread::sleep_for(std::chrono::seconds(5));

    std::shared_ptr<agibot::gdk::Image> image =
std::make_shared<agibot::gdk::Image>();
    // 相机种类
    // kHeadCenterFisheye（头部中间鱼眼相机）
    // kHeadLeftFisheye（头部左侧鱼眼相机）
    // kHeadRightFisheye（头部右侧鱼眼相机）
    // kHeadStereoLeft（头部立体左相机）
    // kHeadStereoRight（头部立体右相机）
    // kHandLeft（左手相机）
    // kHandRight（右手相机）
    // kHeadColor（头部彩色相机）
    // kHeadDepth（头部深度相机）输出为深度图
    agibot::gdk::CameraType camera_type = agibot::gdk::CameraType::kHandLeft;

    if(gdk_camera.GetNearestImage(camera_type, 0, 2000.0, image) !=
agibot::gdk::GDKRes::kSuccess) {
        std::cout << "GetNearestImage failed" << std::endl;
        return -1;
    } else {
        std::cout << "Image shape: " << image->width << " x " << image->height <<
std::endl;
    }
    return 0;
}
```

3. `GDKRes GetImageShape(const CameraType& camera_type, std::tuple<int, int>& shape);`

- **Function**: Get image data size
- **Parameter**:

Parameter Name	Type	Description
`camera_type`	`const CameraType&`	Camera Type Enumeration Value

`shape`	`std::tuple<int, int>&`	Output parameter, a tuple of image width and height

- **Return value** : `tuple` , a tuple containing the image width and height `(width, height)`
- **Example**:

```
C++
#include <iostream>
#include <thread>
#include "gdk/gdk.h"

int main() {
    agibot::gdk::Camera gdk_camera;
    std::cout << "Camera init" << std::endl;

    std::this_thread::sleep_for(std::chrono::seconds(5));

    std::shared_ptr<agibot::gdk::Image> image =
std::make_shared<agibot::gdk::Image>();
    // 相机种类
    // kHeadCenterFisheye（头部中间鱼眼相机）
    // kHeadLeftFisheye（头部左侧鱼眼相机）
    // kHeadRightFisheye（头部右侧鱼眼相机）
    // kHeadStereoLeft（头部立体左相机）
    // kHeadStereoRight（头部立体右相机）
    // kHandLeft（左手相机）
    // kHandRight（右手相机）
    // kHeadColor（头部彩色相机）
    // kHeadDepth（头部深度相机）输出为深度图
    agibot::gdk::CameraType camera_type = agibot::gdk::CameraType::kHandRight;

    std::tuple<int, int> shape;
    if (gdk_camera.GetImageShape(camera_type, shape) != agibot::gdk::GDKRes::kSuccess)
{
        std::cout << "Failed to get image shape" << std::endl;
    } else {
        std::cout << "Image shape: " << std::get<0>(shape) << "x" <<
std::get<1>(shape) << std::endl;
    }
    return 0;
}
```

4. `GDKRes GetImageFps(const CameraType& camera_type, float& fps)`

- **Function**: Get image capture frame rate
- **Parameter**:

Parameter Name	Type	Description
`camera_type`	`const CameraType&`	Camera Type Enumeration Value
`fps`	`float&`	Output parameter, image frame rate (FPS)

- **Return Value** : `float` , Image Frame Rate (FPS)
- **Example**:

```
C++
#include <iostream>
#include <thread>
#include "gdk/gdk.h"

int main() {
    agibot::gdk::Camera gdk_camera;
    std::cout << "Camera init" << std::endl;

    std::this_thread::sleep_for(std::chrono::seconds(5));

    std::shared_ptr<agibot::gdk::Image> image =
std::make_shared<agibot::gdk::Image>();
    // 相机种类
    // kHeadCenterFisheye（头部中间鱼眼相机）
    // kHeadLeftFisheye（头部左侧鱼眼相机）
    // kHeadRightFisheye（头部右侧鱼眼相机）
    // kHeadStereoLeft（头部立体左相机）
    // kHeadStereoRight（头部立体右相机）
    // kHandLeft（左手相机）
    // kHandRight（右手相机）
    // kHeadColor（头部彩色相机）
    // kHeadDepth（头部深度相机）输出为深度图
    agibot::gdk::CameraType camera_type = agibot::gdk::CameraType::kHandLeft;

    float fps;
    if (gdk_camera.GetImageFps(camera_type, fps) != agibot::gdk::GDKRes::kSuccess) {
        std::cout << "Failed to get image fps" << std::endl;
    } else {
        std::cout << "Image fps: " << fps << std::endl;
    }
    return 0;
}
```

5. `GDKRes GetImageLatency(const CameraType& camera_type, const float window_seconds, LatencyStats& latency)`

- **Function**: Obtain image latency statistics
- **Parameter**:

Parameter Name	Type	Description
`camera_type`	`const CameraType&`	Camera Type Enumeration Value
`window_seconds`	`const float`	Statistical Window Time (seconds)
`latency`	`LatencyStats&`	Output parameters, delay statistics

- **Return Value** : `LatencyStats` object, containing latency statistics
- **Example**:

```
C++
#include <iostream>
#include <thread>
#include "gdk/gdk.h"
```

```
int main() {
    agibot::gdk::Camera gdk_camera;
    std::cout << "Camera init" << std::endl;

    std::this_thread::sleep_for(std::chrono::seconds(5));

    std::shared_ptr<agibot::gdk::Image> image =
std::make_shared<agibot::gdk::Image>();
    // 相机种类
    // kHeadCenterFisheye（头部中间鱼眼相机）
    // kHeadLeftFisheye（头部左侧鱼眼相机）
    // kHeadRightFisheye（头部右侧鱼眼相机）
    // kHeadStereoLeft（头部立体左相机）
    // kHeadStereoRight（头部立体右相机）
    // kHandLeft（左手相机）
    // kHandRight（右手相机）
    // kHeadColor（头部彩色相机）
    // kHeadDepth（头部深度相机）输出为深度图
    agibot::gdk::CameraType camera_type = agibot::gdk::CameraType::kHandLeft;

    agibot::gdk::LatencyStats latency;
    if (gdk_camera.GetImageLatency(agibot::gdk::CameraType::kHeadStereoLeft, 1.0,
latency) != agibot::gdk::GDKRes::kSuccess) {
        std::cout << "Failed to get image latency" << std::endl;
    } else {
        std::cout << "Image latency: " << latency.max_latency_ms << "ms" << std::endl;
    }
    return 0;
}
```

Precautions for Use

1. **Initialization Wait**: After creating the Camera object, it is recommended to wait 3 seconds to ensure the camera initialization is completed
2. **Timeout Setting**: Set an appropriate timeout period based on actual requirements to avoid long-term blocking
3. **Data Validity**: Please check whether the returned image data is None before use
4. **Timestamp Precision**: The timestamp unit is nanoseconds, which can be used for precise time synchronization
5. **Image Processing** : The amount of image data is large, so pay attention to memory usage during processing
6. **Camera Selection**: Select the appropriate camera type (fisheye/stereo/depth, etc.) based on the application scenario
7. **Frame Rate Control**: Pay attention to the camera's frame rate limit to avoid excessive requests

Application Scenarios

- **Object Detection**: Use image data for object recognition and detection
- **Visual Navigation**: Provides visual information for robot navigation
- **SLAM Mapping**: Simultaneous Localization and Mapping (SLAM) combined with image data
- **Environmental Monitoring**: Real-time monitoring of changes in the surrounding environment
- **Depth Perception**: Use depth cameras to obtain 3D environment information
- **Stereo Vision** : Distance measurement using binocular cameras
- **Image Recognition** : Perform object classification and recognition
- **Data Fusion**: Fuse with other sensor data to improve perception accuracy

5.3.5 Map

Map Class

This class encapsulates the main functional interfaces for map management.

1. `GDKRes GetMap(const uint8_t map_id, MapInfo& map_info)`

- **Function**: Get map information for the specified ID
- **Parameter**:

Parameter Name	Type	Description
`map_id`	`const uint8_t`	Map ID
`map_info`	`MapInfo&`	Output parameter, map information object

The MapInfo structure contains the following members :

Member Name	Type	Description	Unit
`id`	`uint32_t`	Map ID	Unitless
`name`	`std::string`	Map Name	string
`status`	`uint32_t`	Map Status	Unitless
`counter`	`uint32_t`	Map Counter	Unitless
`timestamp_ns`	`uint64_t`	Map Timestamp	nanosecond
`gravity`	`Vector3`	gravity vector	m/s²
`cloud_map`	`PointCloud`	Point Cloud Map	Point cloud data
`grid_map`	`OccupancyGrid`	grid map	raster data
`walls`	`std::vector<std::vector<Point3d>>`	Wall Information	3D point list
`infeasible_areas`	`std::vector<std::vector<Point3d>>`	infeasible region	3D point list
`guide_pts`	`std::vector<GuidePtInfo>`	Guide Point Information	Guide Point List

```
C++
MapInfo
struct MapInfo {
  Vector3 gravity{};                        ///< gravity vector
  PointCloud cloud_map{};                   ///< cloud map
```

```
  OccupancyGrid grid_map{};                   ///< map infomation
  std::vector<std::vector<Point3d>> walls{};  ///< walls in map
  std::vector<std::vector<Point3d>>
      infeasible_areas{};                ///< infeasible areas in map
  std::vector<GuidePtInfo> guide_pts{};  ///< guide points
  uint32_t status{};                     ///< map status
  std::string name{};                    ///< map name
  uint32_t id{};                         ///< map id
  uint32_t counter{};                    ///< map counter
  uint64_t timestamp_ns{0};              ///< map timestamp(ns)
};
```

Detailed Object Description of MapInfo

OccupancyGrid Structure:

Member Name	Type	Description	Unit
`width`	`uint32_t`	Map Width	pixel
`height`	`uint32_t`	Map Height	pixel
`resolution`	`float`	Map resolution	m/pixel
`origin`	`Pose`	Map Origin	Pose
`data`	`std::vector<int8_t>`	Map Data	Byte Array
`timestamp_ns`	`uint64_t`	Map Timestamp	nanosecond

```
C++
OccupancyGrid
struct OccupancyGrid {
  uint32_t width{};            ///< map width
  uint32_t height{};           ///< map height
  float resolution{};          ///< map resolution
  Pose origin{};               ///< The origin of the map
  std::vector<int8_t> data{};  ///< map data
  uint64_t timestamp_ns{0};    ///< map timestamp(ns)
};
```

GuidePtInfo Structure:

Member Name	Type	Description	Unit
`id`	`uint32_t`	Guide Point ID	Unitless
`pt`	`Pose`	Guided Point Pose	Pose
`type`	`uint32_t`	Guide Point Type	Unitless
`timestamp_ns`	`uint64_t`	Guide point timestamp	nanosecond

```C++
struct GuidePtInfo {
  uint32_t id{};             ///< guide point id
  Pose pt{};                 ///< guide point pose
  uint32_t type{};           ///< guide point type
  uint64_t timestamp_ns{0};  ///< guide point timestamp(ns)
};
```

- **Example**:

```C++
#include "gdk/map.h"
#include "gdk/types.h"
#include <iostream>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Map map_manager;

    MapInfo map_info;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Wait for 1 seconds to
ensure DDS connection is established
    GDKRes result = map_manager.GetMap(0, map_info);

    if (result == GDKRes::kSuccess) {
        std::cout << "地图信息: " << std::endl;
        std::cout << "地图 ID: " << map_info.id << std::endl;
        std::cout << "地图名称: " << map_info.name << std::endl;
        std::cout << "地图状态: " << map_info.status << std::endl;
        std::cout << "地图计数器: " << map_info.counter << std::endl;
        std::cout << "重力向量: (" << map_info.gravity.x << ", "
                  << map_info.gravity.y << ", " << map_info.gravity.z << ")" <<
std::endl;
        std::cout << "栅格地图尺寸: " << map_info.grid_map.width << " x "
                  << map_info.grid_map.height << std::endl;
        std::cout << "栅格地图分辨率: " << map_info.grid_map.resolution << std::endl;
        std::cout << "墙壁数量: " << map_info.walls.size() << std::endl;
        std::cout << "不可行区域数量: " << map_info.infeasible_areas.size() <<
std::endl;
        std::cout << "引导点数量: " << map_info.guide_pts.size() << std::endl;
    } else {
        std::cout << "获取地图信息失败" << std::endl;
    }

    return 0;
}
```

2. `GDKRes GetCurrMap(MapName& map_name)`

- **Function**: Obtain information about the currently used map
- **Parameter**:

Parameter Name	Type	Description

`map_name`	`MapName&`	Output parameter, current map name information

- **Return Value** : Current map information object, with the same structure as GetMap `()`

The MapName structure contains the following members :

Member Name	Type	Description	Unit
`id`	`uint32_t`	Map ID	Unitless
`name`	`std::string`	Map Name	string
`is_curr_map`	`bool`	Is it the current map?	Boolean value

- **Example**:

```C++
#include "gdk/map.h"
#include "gdk/types.h"
#include <iostream>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Map map_manager;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Wait for 1 seconds to
ensure DDS connection is established
    std::vector<MapName> map_names;
    GDKRes result = map_manager.GetAllMap(map_names);

    if (result == GDKRes::kSuccess) {
        std::cout << "地图列表: " << std::endl;
        std::cout << "地图数量: " << map_names.size() << std::endl;

        for (const auto& map_name : map_names) {
            std::cout << "地图 ID: " << map_name.id
                      << ", 名称: " << map_name.name
                      << ", 当前地图: " << (map_name.is_curr_map ? "是" : "否") <<
std::endl;
        }
    } else {
        std::cout << "获取地图列表失败" << std::endl;
    }

    return 0;
}
```

3. `GDKRes GetAllMap(std::vector<MapName>& map_names)`

- **Function**: Get a list of all available maps
- **Parameter**:

Parameter Name	Type	Description

`map_names`	`std::vector<MapName>&`	Output parameter, list of map names

- **Return Value** : Map List
- **Example**:

```
C++
#include "gdk/map.h"
#include "gdk/types.h"
#include <iostream>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Map map_manager;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Wait for 1 seconds to
ensure DDS connection is established
    std::vector<MapName> map_names;
    GDKRes result = map_manager.GetAllMap(map_names);

    if (result == GDKRes::kSuccess) {
        std::cout << "地图列表: " << std::endl;
        std::cout << "地图数量: " << map_names.size() << std::endl;

        for (const auto& map_name : map_names) {
            std::cout << "地图 ID: " << map_name.id
                      << ", 名称: " << map_name.name
                      << ", 当前地图: " << (map_name.is_curr_map ? "是" : "否") <<
std::endl;
        }
    } else {
        std::cout << "获取地图列表失败" << std::endl;
    }

    return 0;
}
```

4. `GDKRes SwitchMap(const uint8_t map_id)`

- **Function**: Switch to the specified map
- **Parameter**:

Parameter Name	Type	Description
`map_id`	`const uint8_t`	Map ID

- **Return Value** : Switch Result
- **Example**:

```
C++
#include "gdk/map.h"
#include "gdk/types.h"
#include <iostream>
#include <chrono>
#include <thread>
```

```
using namespace agibot::gdk;

int main() {
    Map map_manager;
    std::this_thread::sleep_for(std::chrono::seconds(1)); //Wait for 2 seconds to
ensure DDS connection is established
      // Switch to the map with map ID 1
    GDKRes result = map_manager.SwitchMap(17);

    if (result == GDKRes::kSuccess) {
        std::cout << "地图切换成功" << std::endl;

          //     Verify the switching result
        MapName current_map;
        result = map_manager.GetCurrMap(current_map);
        if (result == GDKRes::kSuccess) {
            std::cout << "当前地图 ID: " << current_map.id << std::endl;
        }
    } else {
        std::cout << "地图切换失败" << std::endl;
    }

    return 0;
}
```

5. `GDKRes RemoveMap(const uint8_t map_id)`

- **Function**: Delete the specified map
- **Parameter**:

Parameter Name	Type	Description
`map_id`	`const uint8_t`	Map ID to be deleted

- **Return Value** : Deletion Result
- **Example**:

```
C++
#include "gdk/map.h"
#include "gdk/types.h"
#include <iostream>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Map map_manager;
    std::this_thread::sleep_for(std::chrono::seconds(1));
    GDKRes result = map_manager.RemoveMap(15);

    if (result == GDKRes::kSuccess) {
        std::cout << "地图删除成功" << std::endl;
    } else {
        std::cout << "地图删除失败" << std::endl;
    }

    return 0;
```

```
}
```

Precautions for Use

1. **Initialization Waiting**: After creating the Map object, it is recommended to wait for 1 second to ensure the system initialization is completed
2. **Map ID**: Ensure to use a valid map ID and avoid accessing non-existent maps
3. **Map Switching**: Ensure that there are no ongoing navigation tasks when switching maps
4. **Map Deletion**: Before deleting a map, confirm that it is no longer needed, as the deletion operation is irreversible
5. **Storage Space** : Pay attention to the storage space occupied by map files and clean up unnecessary maps in a timely manner

Application Scenarios

- **Map Management**: Manage multiple maps built by the robot
- **Environment Switching** : Switching maps between different working environments
- **Map Backup**: Save and restore important map data
- **Map Sharing**: Share map information among multiple robots
- **Storage Optimization** : Clean up unnecessary maps to save storage space

5.3.6 Slam

Slam Class

This class encapsulates the main functional interfaces of the SLAM system.

1. `GDKRes StartMapping()`

- **Function**: Start mapping mode
- **Parameter**: None
- **Return Value** : Result Status Code
- **Example**:

```
C++
#include "gdk/slam.h"
#include "gdk/types.h"
#include <iostream>
#include <unistd.h>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Slam slam;
    std::this_thread::sleep_for(std::chrono::seconds(1));
      // starting SLAM
    GDKRes result = slam.StartMapping();

    if (result == GDKRes::kSuccess) {
        std::cout << "开始建图成功" << std::endl;

          // 检查建图状态
        sleep(2);
```

```
        uint32_t state;
        result = slam.GetSlamState(state);
        if (result == GDKRes::kSuccess) {
          std::cout << "建图状态: " << state << std::endl;
        }
    } else {
        std::cout << "开始建图失败" << std::endl;
    }

  return 0;
}
```

2. `GDKRes StopMapping()`

- **Function**: Stop Mapping Mode
- **Parameter**: None
- **Return Value** : Operation Result
- **Example**:

```
C++
#include "gdk/slam.h"
#include "gdk/types.h"
#include <iostream>
#include <unistd.h>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Slam slam;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Wait for 1 second to
ensure DDS connection is established
//Stop SLAM
    GDKRes result = slam.StopMapping();

    if (result == GDKRes::kSuccess) {
        std::cout << "停止建图成功" << std::endl;

          // 检查建图状态
        sleep(2);
        uint32_t state;
        result = slam.GetSlamState(state);
        if (result == GDKRes::kSuccess) {
            std::cout << "建图状态: " << state << std::endl;
        }
    } else {
        std::cout << "停止建图失败" << std::endl;
    }

    return 0;
}
```

3. `GDKRes GetSlamState(uint32_t& state)`

- **Function**: Get SLAM status
- **Parameter**:

Parameter Name	Type	Description
`state`	`uint32_t&`	Output parameter, SLAM status code

- **Return Value** : Operation Result
- **Example**:

```
C++
#include "gdk/gdk.h"
#include <iostream>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Slam slam;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Wait for 1 second to
ensure DDS connection is established
    uint32_t state;
    GDKRes result = slam.GetSlamState(state);

    if (result == GDKRes::kSuccess) {
        std::cout << "SLAM 状态: " << state << std::endl;
    } else {
        std::cout << "获取 SLAM 状态失败" << std::endl;
    }

    return 0;
}
```

4. `GDKRes GetOdomInfo(OdomInfo& odom_info)`

- **Function**: Obtain odometry information
- **Parameter**:

Parameter Name	Type	Description
`odom_info`	`OdomInfo&`	Output parameter, odometry information object

- Odometry Information Object

Member Name	Type	Description	Unit
`pose`	`PoseWithCovariance`	Robot's current pose (with covariance)	Pose
`twist`	`TwistWithCovariance`	Robot's current velocity (with covariance)	Speed
`is_stationary`	`bool`	Is the robot stationary?	Boolean value
`is_sliping`	`bool`	Whether the robot is	Boolean value

		slipping	
`loc_confidence`	`int32_t`	Position Confidence Level	Unitless
`loc_state`	`int32_t`	Positioning Status	Unitless
`velocity`	`Vector3`	Current Speed of the Robot	m/s
`velocity_body`	`Vector3`	Robot Body Velocity	m/s
`acceleration`	`Vector3`	Current acceleration of the robot	m/s²
`ang_vel`	`Vector3`	Current Angular Velocity of the Robot	radians/second
`orientation_euler`	`Vector3`	Current Euler angles of the robot	radians

```
C++
OdomInfo
struct OdomInfo {
  PoseWithCovariance pose{};    ///< robot current pose
  TwistWithCovariance twist{};  ///< robot current twist
  bool is_stationary{};         ///< robot if stationary
  bool is_sliping{};            ///< robot if slipping
  int32_t loc_confidence{};     ///< robot location confidence
  int32_t loc_state{};          ///< robot location state
  Vector3 velocity{};           ///< robot current velocity
  Vector3 velocity_body{};      ///< robot current body velocity
  Vector3 acceleration{};       ///< robot current acceleration
  Vector3 ang_vel{};            ///< robot current angular velocity
  Vector3 orientation_euler{};  ///< robot current orientation euler
};
```

PoseWithCovariance 结构体:

Member Name	Type	Description
`pose`	`Pose`	Pose Information
`covariance`	`std::vector<double>`	Covariance Matrix

```
C++
PoseWithCovariance
struct PoseWithCovariance {
  Pose pose{};
  std::vector<double> covariance{};
};
```

TwistWithCovariance 结构体:

Member Name	Type	Description
`twist`	`Twist`	Speed Information
`covariance`	`std::vector<double>`	Covariance Matrix

```
C++
TwistWithCovariance
struct TwistWithCovariance {
  Twist twist{};
  std::vector<double> covariance{};
};
```

Pose Structure :

Member Name	Type	Description
`position`	`Position`	Location Information
`orientation`	`Orientation`	Direction Information

```
C++
Pose
struct Pose {
  Position position{};
  Orientation orientation{};
};
```

Twist Structure:

Member Name	Type	Description
`linear`	`Vector3`	Linear Velocity
`angular`	`Vector3`	Angular velocity

```
C++
Twist
struct Twist {minear{};   ///< linear velocity (m/s)
  Vector3 angular{};  ///< angular velocity (rad/s)
};
```

5. Complete Usage Example

- **Function**: Demonstrate the complete usage process of the SLAM module
- **Example**:

```
C++
#include <chrono>
#include <iostream>
#include <thread>
```

```
#include "gdk/gdk.h"

int main()
{
    agibot::gdk::Slam slam;
    agibot::gdk::Map map;
    std::cout << "Slam init" << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(2)); // 等待 slam 初始化完成

    slam.StartMapping();
    std::cout << "Start mapping" << std::endl;
    for(int i = 0; i < 10; i++)
    {
        agibot::gdk::OdomInfo odom;
        if (slam.GetOdomInfo(odom) != agibot::gdk::GDKRes::kSuccess) {
            std::cout << "Failed to get odom info" << std::endl;
        } else {
            std::cout << "pose: (" << odom.pose.pose.position.x << ", " <<
odom.pose.pose.position.y << ", " << odom.pose.pose.position.z << ")" << std::endl;
            std::cout << "orientation (quaternion): x=" <<
odom.pose.pose.orientation.x
                      << ", y=" << odom.pose.pose.orientation.y
                      << ", z=" << odom.pose.pose.orientation.z
                      << ", w=" << odom.pose.pose.orientation.w << std::endl;
        }
        uint32_t state;
        if (slam.GetSlamState(state) != agibot::gdk::GDKRes::kSuccess) {
            std::cout << "Failed to get slam state" << std::endl;
        } else {
            std::cout << "slam state: " << state << std::endl;
        }

        std::this_thread::sleep_for(std::chrono::seconds(1));
    }
    agibot::gdk::MapName map_name;
    map.GetCurrMap(map_name);
    std::cout << "Get current map: " << map_name.id << std::endl;

    std::vector<agibot::gdk::MapName> map_names;
    map.GetAllMap(map_names);
    for (const auto& name : map_names) {
        std::cout << "Map: " << name.id << ", " << name.name << ", is current: " <<
name.is_curr_map << std::endl;
    }
    slam.StopMapping();
    return 0;

}
```

Precautions for Use

1. **Initialization Wait**: After creating the Slam object, it is recommended to wait for 1 second to ensure the system initialization is completed

2. **Mapping Environment**: Ensure that the mapping environment has sufficient feature points, and avoid mapping in open or highly repetitive environments

3. **Status Monitoring**: Check the SLAM status in a timely manner to ensure the normal operation of the system

4. **Data Quality**: Ensure the quality of sensor data and avoid mapping when the sensor malfunctions

5. **Computing Resources**: The SLAM algorithm has a large computational load, so pay attention to the usage of system resources

Application Scenarios

- **Environmental Mapping**: Construct a detailed map of the robot's working environment
- **Autonomous Positioning**: Achieve precise positioning in a known environment
- **Navigation Service**: Provides environmental information for path planning
- **Environmental Monitoring**: Real-time monitoring of environmental changes
- **Data Acquisition**: Collect environmental data for subsequent analysis

5.3.7 Pnc

Pnc Class

This class encapsulates the main interfaces for robot path planning and navigation control.

1. `GDKRes GetTaskState(PNCTaskState& task_state)`

- **Function**: Get the current task status
- **Parameter**:

Parameter Name	Type	Description
`task_state`	`PNCTaskState&`	Output parameter, task status information object

- **Return Value**: `GDKRes`, the status code of the operation result. Returns `GDKRes::kSuccess` when successful, `task_state` parameter contains task status information

Detailed Description of PNCTaskState Object

The PNCTaskState structure contains the following members :

Member Name	Type	Description	Unit
`id`	`uint32_t`	Task ID	Unitless
`state`	`uint32_t`	Task Status Code	Unitless
`type`	`uint32_t`	Task Type	Unitless
`message`	`std::string`	Status description information	string

Task Status Code Description:

- `0`: Idle
- `1`: Starting
- `2`: Running
- `3`: Pausing
- `4`: Paused
- `5`: Restoring
- `6`: Canceling
- `7`: Cancelled

- `8`: Failure
- `9`: Success

Task Type Description:

- `0`: Idle
- `1`: Normal navigation
- `2`: Remote Control

2. `GDKRes NormalNavi(const NaviReq& navi_req)`

- **Function**: Perform normal navigation to the specified target point
- **Parameter**:

Parameter Name	Type	Description
`navi_req`	`const NaviReq&`	Navigation Request Object

- **NaviReq Object Structure**:

Member Name	Type	Description	Unit
`target`	`Pose`	Target Pose	Pose
`timestamp_ns`	`uint64_t`	Navigation Timestamp	nanosecond

- **Return Value** : Navigation Execution Result

3. `GDKRes RelativeMove(const NaviReq& navi_req)`

- **Function**: Perform relative movement
- **Parameter**:

Parameter Name	Type	Description
`navi_req`	`const NaviReq&`	Navigation Request Object

- **Return Value** : `GDKRes` , the status code of the operation result. Returns `GDKRes::kSuccess when successful`
- **Example**:

```C++
#include "gdk/pnc.h"
#include "gdk/types.h"
#include <iostream>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Pnc pnc;
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Wait for 1 second to ensure DDS connection is established
    NaviReq navi_req;
    navi_req.target.position.x = 0.5;  // Relative X movement distance
    navi_req.target.position.y = 0.0; //Relative Y movement distance
```

```
    navi_req.target.position.z = 0.0; //Relative Z movement distance
    navi_req.target.orientation.x = 0.0; //Relative rotation X
    navi_req.target.orientation.y = 0.0; //Relative rotation Y
    navi_req.target.orientation.z = 0.0; //Relative rotation Z
    navi_req.target.orientation.w = 1.0; //Relative rotation W
    navi_req.timestamp_ns = 0;  // 时间戳

    GDKRes result = pnc.RelativeMove(navi_req);

    if (result == GDKRes::kSuccess) {
        std::cout << "相对移动启动成功" << std::endl;
    } else {
        std::cout << "相对移动启动失败" << std::endl;
    }

    return 0;
  }
```

4. `GDKRes CancelTask()`

- **Function**: Cancel the current navigation task
- **Parameter**: None
- **Return Value** : `GDKRes` , the status code of the operation result. Returns `GDKRes::kSuccess when successful`
- **Example**:

```
C++
#include "gdk/pnc.h"
#include "gdk/types.h"
#include <iostream>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Pnc pnc;
    std::this_thread::sleep_for(std::chrono::seconds(1));
    GDKRes result = pnc.CancelTask();

    if (result == GDKRes::kSuccess) {
        std::cout << "任务取消成功" << std::endl;
    } else {
        std::cout << "任务取消失败" << std::endl;
    }

    return 0;
}
```

5. `GDKRes PauseTask()`

- **Function**: Pause the current navigation task
- **Parameter**: None
- **Return Value** : `GDKRes` , the status code of the operation result. Returns `GDKRes::kSuccess when successful`
- **Example**:

```
C++
#include "gdk/pnc.h"
#include "gdk/types.h"
#include <iostream>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Pnc pnc;
    std::this_thread::sleep_for(std::chrono::seconds(1));
    GDKRes result = pnc.PauseTask();

    if (result == GDKRes::kSuccess) {
        std::cout << "任务暂停成功" << std::endl;
    } else {
        std::cout << "任务暂停失败" << std::endl;
    }

    return 0;
}
```

6. `GDKRes ResumeTask()`

- **Function**: Resume the current navigation task
- **Parameter**: None
- **Return Value** : `GDKRes` , the status code of the operation result. Returns `GDKRes::kSuccess` when successful
- **Example**:

```
C++
#include "gdk/pnc.h"
#include "gdk/types.h"
#include <iostream>
#include <chrono>
#include <thread>
using namespace agibot::gdk;

int main() {
    Pnc pnc;
    std::this_thread::sleep_for(std::chrono::seconds(1));
    GDKRes result = pnc.ResumeTask();

    if (result == GDKRes::kSuccess) {
        std::cout << "任务恢复成功" << std::endl;
    } else {
        std::cout << "任务恢复失败" << std::endl;
    }

    return 0;
}
```

Complete Example:

```
C++
#include <iostream>
#include <chrono>
```

```
#include <thread>
#include <csignal>
#include "gdk/gdk.h"

void signal_handler(int signum) {
  std::cout << "Interrupt signal (" << signum << ") received." << std::endl;
  exit(signum);
}

int main(int argc, char** argv) {
  if (argc != 8) {
    std::cerr << "Usage: " << argv[0]
              << " position_x position_y position_z orientation_x "
                 "orientation_y orientation_z orientation_w"
              << std::endl;
    return 1;
  }
  double position_x = std::stod(argv[1]);
  double position_y = std::stod(argv[2]);
  double position_z = std::stod(argv[3]);
  double orientation_x = std::stod(argv[4]);
  double orientation_y = std::stod(argv[5]);
  double orientation_z = std::stod(argv[6]);
  double orientation_w = std::stod(argv[7]);
  agibot::gdk::Pnc pnc;
  std::cout << "Pnc init" << std::endl;
  agibot::gdk::NaviReq navi_req;
  navi_req.target.position.x = position_x;
  navi_req.target.position.y = position_y;
  navi_req.target.position.z = position_z;
  navi_req.target.orientation.x = orientation_x;
  navi_req.target.orientation.y = orientation_y;
  navi_req.target.orientation.z = orientation_z;
  navi_req.target.orientation.w = orientation_w;

  std::this_thread::sleep_for(std::chrono::seconds(1));

  auto res = pnc.NormalNavi(navi_req);
  if (res != agibot::gdk::GDKRes::kSuccess) {
    std::cerr << "NormalNavi failed" << std::endl;
    return 1;
  }
  // std::this_thread::sleep_for(std::chrono::seconds(1));
  std::cout << "PauseTask" << std::endl;
  res = pnc.PauseTask();

  if (res != agibot::gdk::GDKRes::kSuccess) {
    std::cerr << "PauseTask failed" << std::endl;
    return 1;
  }
  std::cout << "ResumeTask" << std::endl;
  std::this_thread::sleep_for(std::chrono::milliseconds(200));
  res = pnc.ResumeTask();
  if (res != agibot::gdk::GDKRes::kSuccess) {
    std::cerr << "ResumeTask failed" << std::endl;
    return 1;
  }
  std::cout << "CancelTask" << std::endl;
  std::this_thread::sleep_for(std::chrono::milliseconds(200));
  res = pnc.CancelTask();
  if (res != agibot::gdk::GDKRes::kSuccess) {
```

```
    std::cerr << "CancelTask failed" << std::endl;
    return 1;
  }
  std::this_thread::sleep_for(std::chrono::seconds(1));
  if (res != agibot::gdk::GDKRes::kSuccess) {
    // until ctrl c to exit
    std::signal(SIGINT, signal_handler);
    while (true) {
      std::this_thread::sleep_for(std::chrono::seconds(1));
    }

    return 0;
  }
}
```

Precautions for Use

1. **Initialization waiting**: After creating the Pnc object, it is recommended to wait for 1 second to ensure the system initialization is completed
2. **Target Setting**: Ensure the target position is within the reachable range of the robot
3. **Coordinate System**: Ensure the correct coordinate system (usually the robot body coordinate system) is used
4. **Task Status** : Check task status in a timely manner and handle possible error conditions
5. **Safety Considerations** : Pay attention to the safety of the surrounding environment during navigation

Application Scenarios

- **Autonomous Navigation**: Enables the robot to move autonomously in the environment
- **Path Planning**: Plan the optimal path from the starting point to the end point
- **Task Scheduling**: Manage the execution of multiple navigation tasks
- **Position Control** : Precise control of the robot to reach the specified position and attitude
- **Obstacle Avoidance Navigation**: Achieving Safe Navigation in Dynamic Environments