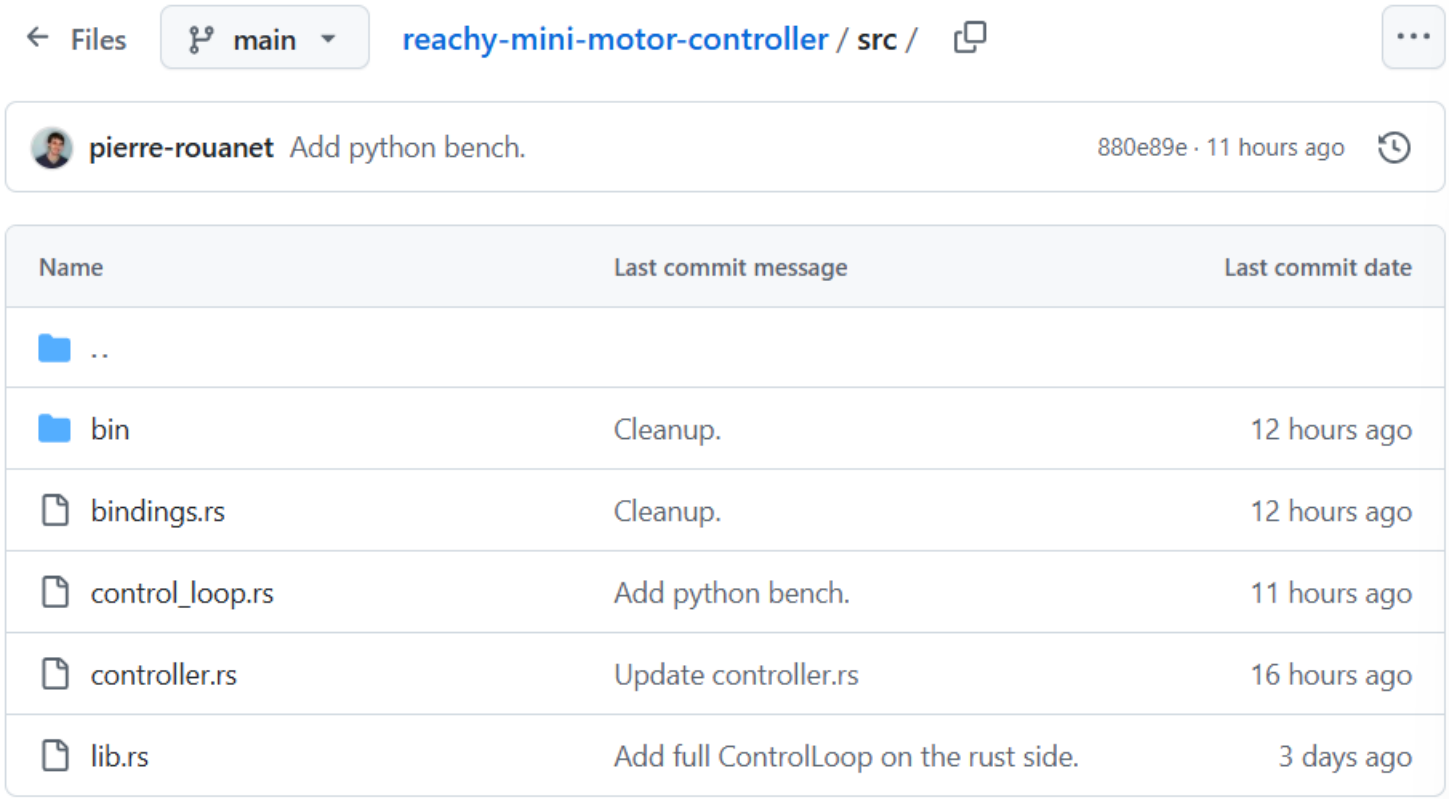


bindings.rs:
파이썬에서 Rust코드 사용하기 위한 PyO3 바인딩 정의 파일
파이썬에서 호출 가능한 메서드 제공

control_loop.rs:
실제 제어 루프 구현 부분
모터 상태 읽기/쓰기, 목표 위치 설정, 통계 측정 기능 등

controller.rs:
모터 제어함수
위치읽기, 위치 설정, 토크제어, 동작모드 설정

lib.rs:
라이브러리 엔트리 포인트
Python 바인딩에 노출할 API 설정

구성요소	역할
Struct ReachyMiniControlLoop	제어 루프 전체를 감싸는 핵심 구조체 (스레드 생성, 통신 관리)
Enum MotorCommand	명령 큐로 전달되는 제어 명령 열거형
Struct FullBodyPosition	9개 모터의 상태를 담은 구조체 (몸체 + 스튜어트 6축 + 안테나 2개)
run()	실제 제어 루프가 실행되는 비동기 스레드 루프
handle_commands()	명령 큐에서 받은 명령 실행
read_pos()	모터 위치 읽기
get_stats()	루프 성능 통계 가져오기

struct ReachyMiniControlLoop

```rust
use tokio::{
    sync::mpsc::{self, Sender},
    time,
};
```

```rust
pub struct ReachyMiniControlLoop {
    tx: Sender<MotorCommand>,
    last_position: Arc<Mutex<Result<FullBodyPosition, String>>>,
    last_stats: Option<(Duration, Arc<Mutex<ControlLoopStats>>)>
}
```

Self

tx
모터 명령 전달. 다른 쓰레드/비동기 코드가 tx를 통해 명령을 보냄

last_position
마지막으로 읽은 포지션 데이터 저장

last_stats
period, read_dt, write_dt 주기적으로 저장

struct ReachyMiniControlLoop

```rust
impl ReachyMiniControlLoop {
    pub fn new(
        serialport: String,
        read_position_loop_period: Duration,
        stats_pub_period: Option<Duration>,
        read_allowed_retries: u64,
    ) -> Result<Self, Box<dyn std::error::Error>> {
        let (tx, rx) = mpsc::channel(100);
```

ReachyMiniControlLoop 생성자
시리얼포트 주소,
position 읽기 주기,
통계 주기(Option이라 없을수도 있음),
읽기 실패시 재시도 횟수

성공시 Self 반환
실패시 Error 반환

struct ReachyMiniControlLoop

```rust
impl ReachyMiniControlLoop {
    pub fn new(
        serialport: String,
        read_position_loop_period: Duration,
        stats_pub_period: Option<Duration>,
        read_allowed_retries: u64,
    ) -> Result<Self, Box<dyn std::error::Error>> {
        let (tx, rx) = mpsc::channel(100);

        let last_stats = stats_pub_period.map(|period| {
            (
                period,
                Arc::new(Mutex::new(ControlLoopStats {
                    period: Vec::new(),
                    read_dt: Vec::new(),
                    write_dt: Vec::new(),
                })),
            )
        });
        let last_stats_clone = last_stats.clone();

        let mut c = ReachyMiniMotorController::new(serialport.as_str()).unwrap();
```

<명령 채널 생성>
tx: 외부에서 명령을 보낼 송신자
rx: 내부에서 명령을 받을 수신자
버퍼크기 100

<통계 구조 초기화>
controlLoopStats 생성

<시리얼 포트로 모터 컨트롤러 초기화>
실질적으로 하드웨어 제어를 담당하는 객체 생성

struct ReachyMiniControlLoop

```rust
// Init last position by trying to read current positions
// If the init fails, it probably means we have an hardware issue
// so it's better to fail.
let last_position = read_pos_with_retries(&mut c, read_allowed_retries)?;
// .map_err(|e| format!("Failed to read initial positions: {}", e))?;

let last_position = Arc::new(Mutex::new(Ok(last_position)));
let last_position_clone = last_position.clone();

std::thread::spawn(move || {
    run(
        c,
        rx,
        last_position_clone,
        last_stats_clone,
        read_position_loop_period,
        read_allowed_retries,
    );
});
```

<초기 position 정보 읽기>
현재 모터의 position 정보 읽어옴
실패하면 즉시 에러 반환
성공하면 FullBodyPosition값 반환

<상태 저장 필드>
읽어온 last_position값을 다른 스레드에서 공유할 수 있도록
Arc로 감싸고
동기화를 위해 Mutex 사용

<루프 실행>
새로운 쓰레드에서 실제 제어루프인 run() 실행
여기에 모든 공유 상태들을 넘김

struct ReachyMiniControlLoop

```
        Ok(ReachyMiniControlLoop {
            tx,
            last_position,
            last_stats,
        })
    }
```

마지막으로 초기화된 구조체 인스턴스 리턴

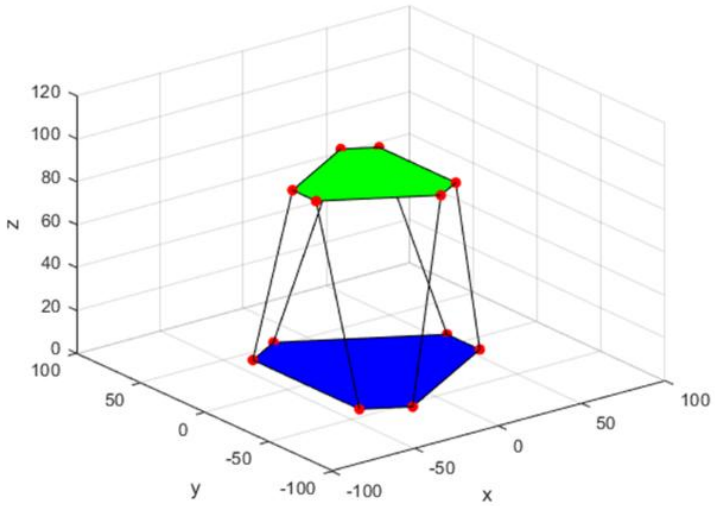
enum MotorCommand

```rust
#[derive(Debug, Clone)]
pub enum MotorCommand {
    SetAllGoalPositions { positions: FullBodyPosition },
    SetStewartPlatformPosition { position: [f64; 6] },
    SetBodyRotation { position: f64 },
    SetAntennasPositions { positions: [f64; 2] },
    EnableTorque(),
    DisableTorque(),
    SetStewartPlatformGoalCurrent { current: [i16; 6] },
    SetStewartPlatformOperatingMode { mode: u8 },
    SetAntennasOperatingMode { mode: u8 },
    SetBodyRotationOperatingMode { mode: u8 },
    EnableStewartPlatform { enable: bool },
    EnableBodyRotation { enable: bool },
    EnableAntennas { enable: bool },
}
```

SetAllGoalPositions { positions }	전체 바디(몸통, 스튜어트 플랫폼, 안테나)에 목표 위치 설정
SetStewartPlatformPosition { position }	6축 스튜어트 플랫폼에 개별 위치 설정
SetBodyRotation { position }	바디의 회전 위치 설정 (Yaw 축 회전)
SetAntennasPositions { positions }	양쪽 안테나에 위치 설정
EnableTorque()	모터 토크(전류) 활성화
DisableTorque()	모터 토크 비활성화
SetStewartPlatformGoalCurrent { current }	스튜어트 플랫폼 각 축에 대해 목표 전류 설정
SetStewartPlatformOperatingMode { mode }	스튜어트 플랫폼의 운용 모드 설정
SetAntennasOperatingMode { mode }	안테나 운용 모드 설정
SetBodyRotationOperatingMode { mode }	몸통 회전 운용 모드 설정
EnableStewartPlatform { enable }	스튜어트 플랫폼 활성화 여부 설정
EnableBodyRotation { enable }	바디 회전 활성화 여부 설정
EnableAntennas { enable }	안테나 활성화 여부 설정

struct FullBodyPosition

```rust
#[gen_stub_pyclass]
#[pyclass]
#[derive(Debug, Clone, Copy)]
pub struct FullBodyPosition {
    #[pyo3(get)]
    pub body_yaw: f64,
    #[pyo3(get)]
    pub stewart: [f64; 6],
    #[pyo3(get)]
    pub antennas: [f64; 2],
    #[pyo3(get)]
    pub timestamp: f64, // seconds since UNIX epoch
}
```



body_yaw: 몸통 회전값

stewart: 6축의 각 position

antennas: 안테나 position

timestamp: 현재 시간, UNIX epoch 기준 초 단위

handle_commands

```rust
fn handle_commands(
    controller: &mut ReachyMiniMotorController,
    command: MotorCommand,
) -> Result<(), Box<dyn std::error::Error>> {
    use MotorCommand::*;
```

MotorCommand의 다양한 명령들을 받아서
그에 해단하는 하드웨어 제어 명령 실행

controller: 실질적으로 하드웨어를 제어하는 객체

command: 실행할 명령(enum MotorCommand 타입)

반환값: 성공하면 Ok, 실패하면 Err

handle_commands

```
match command {
    SetAllGoalPositions { positions } => controller.set_all_goal_positions([
        positions.body_yaw,
        positions.antennas[0],
        positions.antennas[1],
        positions.stewart[0],
        positions.stewart[1],
        positions.stewart[2],
        positions.stewart[3],
        positions.stewart[4],
        positions.stewart[5],
    ]),
    SetStewartPlatformPosition { position } => {
        controller.set_stewart_platform_position(position)
    }
    SetBodyRotation { position } => controller.set_body_rotation(position),
    SetAntennasPositions { positions } => controller.set_antennas_positions(positions),
    EnableTorque() => controller.enable_torque(),
    DisableTorque() => controller.disable_torque(),
    SetStewartPlatformGoalCurrent { current } => {
        controller.set_stewart_platform_goal_current(current)
    }
    SetStewartPlatformOperatingMode { mode } => {
        controller.set_stewart_platform_operating_mode(mode)
    }
    SetAntennasOperatingMode { mode } => controller.set_antennas_operating_mode(mode),
    SetBodyRotationOperatingMode { mode } => controller.set_body_rotation_operating_mode(mode),
    EnableStewartPlatform { enable } => controller.enable_stewart_platform(enable),
    EnableBodyRotation { enable } => controller.enable_body_rotation(enable),
    EnableAntennas { enable } => controller.enable_antennas(enable),
}
```

명령에 따라 다음 동작들 실행

전체 관절위치 설정

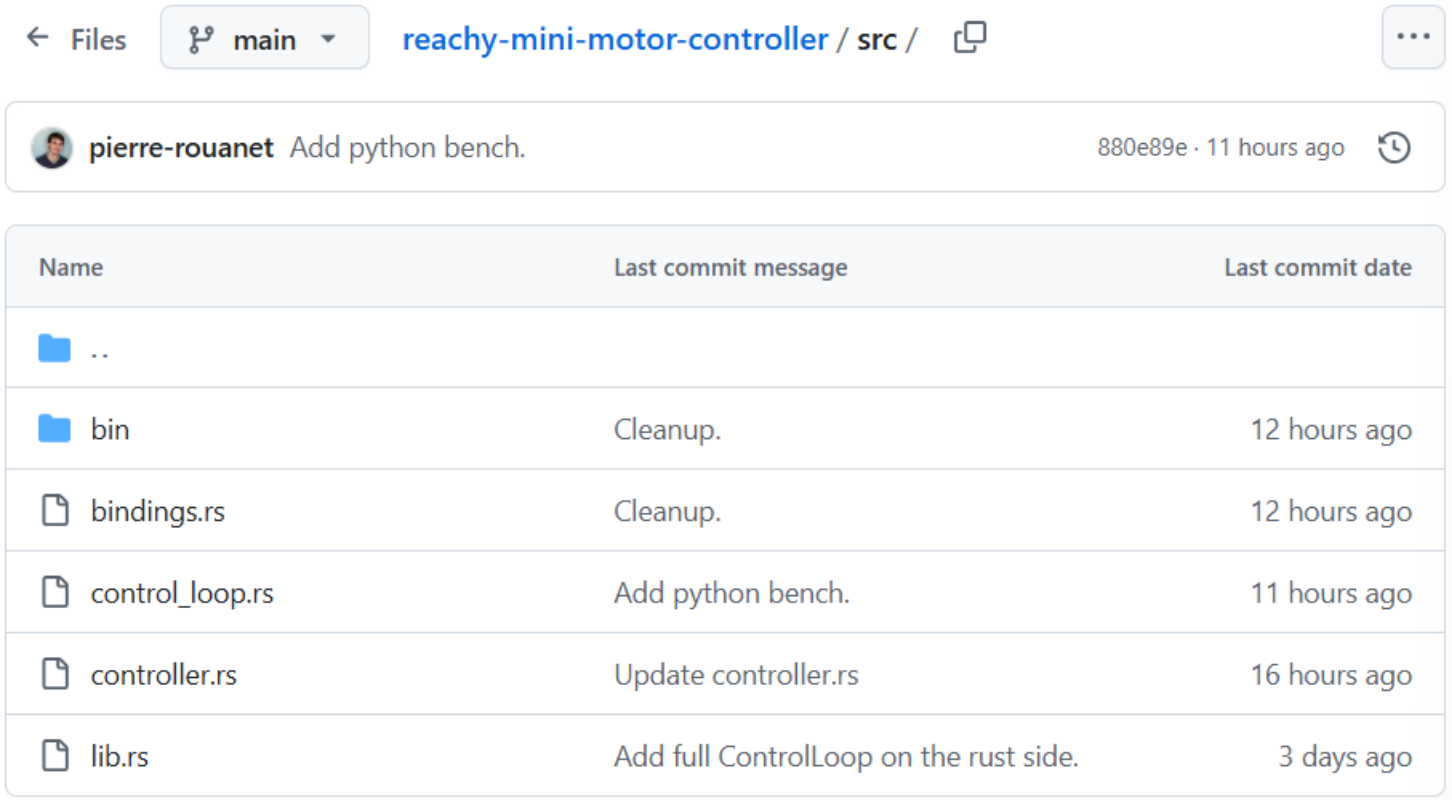
스튜어트 플랫폼 목표위치 설정

몸통 회전 목표위치 설정

안테나의 목표위치 설정

토크 활성화/비활성화

…



bindings.rs:
파이썬에서 Rust코드 사용하기 위한 PyO3 바인딩 정의 파일
파이썬에서 호출 가능한 메서드 제공

control_loop.rs:
실제 제어 루프 구현 부분
모터 상태 읽기/쓰기, 목표 위치 설정, 통계 측정 기능 등

controller.rs:
모터 제어함수
위치읽기, 위치 설정, 토크제어, 동작모드 설정

lib.rs:
라이브러리 엔트리 포인트
Python 바인딩에 노출할 API 설정

controller.rs

impl ReachyMiniMotorController

- func new
- func read_all_positions
- func set_all_goal_positions
- func set_antennas_positions
- func set_stewart_platform_positi...
- func set_body_rotation
- func enable_torque
- func disable_torque
- func set_torque
- func set_stewart_platform_goal_...
- func read_stewart_platform_curr...
- func set_stewart_platform_opera...
- func read_stewart_platform_ope...
- func set_antennas_operating_m...
- func set_body_rotation_operatin...
- func enable_body_rotation
- func enable_antennas
- func enable_stewart_platform

controller.rs

```rust
use rustypot::servo::{dynamixel::xl330, feetech::sts3215};

pub struct ReachyMiniMotorController {
    dph_v1: rustypot::DynamixelProtocolHandler,
    dph_v2: rustypot::DynamixelProtocolHandler,
    serial_port: Box<dyn serialport::SerialPort>,
}

impl ReachyMiniMotorController {
    pub fn new(serialport: &str) -> Result<Self, Box<dyn std::error::Error>> {
        let dph_v1 = rustypot::DynamixelProtocolHandler::v1();
        let dph_v2 = rustypot::DynamixelProtocolHandler::v2();

        let serial_port = serialport::new(serialport, 1_000_000)
            .timeout(Duration::from_millis(10))
            .open()?;

        Ok(Self {
            dph_v1,
            dph_v2,
            serial_port,
        })
    }
```

dph_v1: 다이나믹셀 프로토콜 v1 사용 (Feetech sts3215용)
dph_v2: 다이나믹셀 프로토콜 v2 사용 (Dynamixel xl330용)
serial_port: 실제 직렬 통신 포트

받은 serial port 이름으로 시리얼포트 열고
dph_v1, v2 초기화

성공시 ReachMiniController 반환

controller.rs

```rust
pub fn read_all_positions(&mut self) -> Result<[f64; 9], Box<dyn std::error::Error>> {
    let mut pos = Vec::new();

    pos.extend(sts3215::sync_read_present_position(
        &self.dph_v1,
        self.serial_port.as_mut(),
        &[11, 21, 22],
    )?);
    pos.extend(xl330::sync_read_present_position(
        &self.dph_v2,
        self.serial_port.as_mut(),
        &[1, 2, 3, 4, 5, 6],
    )?);

    Ok(pos.try_into().unwrap())
}
```

sts3215에서 몸통+안테나
xl330에서 스튜어트 플랫폼 읽어옴

```rust
pub fn set_all_goal_positions(
    &mut self,
    positions: [f64; 9],
) -> Result<(), Box<dyn std::error::Error>> {
    sts3215::sync_write_goal_position(
        &self.dph_v1,
        self.serial_port.as_mut(),
        &[11, 21, 22],
        &positions[0..3],
    )?;
    xl330::sync_write_goal_position(
        &self.dph_v2,
        self.serial_port.as_mut(),
        &[1, 2, 3, 4, 5, 6],
        &positions[3..9],
    )?;

    Ok(())
}
```

sts3215에서 몸통+안테나
xl330에서 6개 (스튜어트 플랫폼)
총 9개 모터 포지션 설정

controller.rs

```rust
pub fn set_antennas_positions(
    &mut self,
    positions: [f64; 2],
) -> Result<(), Box<dyn std::error::Error>> {
    sts3215::sync_write_goal_position(
        &self.dph_v1,
        self.serial_port.as_mut(),
        &[21, 22],
        &positions,
    )?;

    Ok(())
}

pub fn set_stewart_platform_position(
    &mut self,
    position: [f64; 6],
) -> Result<(), Box<dyn std::error::Error>> {
    xl330::sync_write_goal_position(
        &self.dph_v2,
        self.serial_port.as_mut(),
        &[1, 2, 3, 4, 5, 6],
        &position,
    )?;

    Ok(())
}
```

sts3215에서 안테나 2개 설정

xl330에서 6개 (스튜어트 플랫폼) 설정

```rust
pub fn set_body_rotation(&mut self, position: f64) -> Result<(), Box<dyn std::error::Error>> {
    sts3215::sync_write_goal_position(
        &self.dph_v1,
        self.serial_port.as_mut(),
        &[11],
        &[position],
    )?;

    Ok(())
}

pub fn enable_torque(&mut self) -> Result<(), Box<dyn std::error::Error>> {
    self.set_torque(true)
}
pub fn disable_torque(&mut self) -> Result<(), Box<dyn std::error::Error>> {
    self.set_torque(false)
}
```

sts3215에서 안테나 2개 설정

xl330에서 6개 (스튜어트 플랫폼) 설정

controller.rs

```rust
fn set_torque(&mut self, enable: bool) -> Result<(), Box<dyn std::error::Error>> {
    sts3215::sync_write_torque_enable(
        &self.dph_v1,
        self.serial_port.as_mut(),
        &[11, 21, 22],
        &[enable; 3],
    )?;
    xl330::sync_write_torque_enable(
        &self.dph_v2,
        self.serial_port.as_mut(),
        &[1, 2, 3, 4, 5, 6],
        &[enable; 6],
    )?;

    Ok(())
}
```

```rust
pub fn set_stewart_platform_goal_current(
    &mut self,
    current: [i16; 6],
) -> Result<(), Box<dyn std::error::Error>> {
    xl330::sync_write_goal_current(
        &self.dph_v2,
        self.serial_port.as_mut(),
        &[1, 2, 3, 4, 5, 6],
        &current,
    )?;

    Ok(())
}
```

위치제어 대신 전류제어 사용
6개의 모터에 전류 목표값을 설정

```rust
pub fn read_stewart_platform_current(
    &mut self,
) -> Result<[i16; 6], Box<dyn std::error::Error>> {
    let currents = xl330::sync_read_present_current(
        &self.dph_v2,
        self.serial_port.as_mut(),
        &[1, 2, 3, 4, 5, 6],
    )?;

    Ok(currents.try_into().unwrap())
}
```

현재 모터가 얼마의 전류로 움직이고 있는지 읽음

controller.rs

```rust
pub fn set_stewart_platform_operating_mode(
    &mut self,
    mode: u8,
) -> Result<(), Box<dyn std::error::Error>> {
    xl330::sync_write_operating_mode(
        &self.dph_v2,
        self.serial_port.as_mut(),
        &[1, 2, 3, 4, 5, 6],
        &[mode; 6],
    )?;

    Ok(())
}
```

```rust
pub fn read_stewart_platform_operating_mode(
    &mut self,
) -> Result<[u8; 6], Box<dyn std::error::Error>> {
    let modes = xl330::sync_read_operating_mode(
        &self.dph_v2,
        self.serial_port.as_mut(),
        &[1, 2, 3, 4, 5, 6],
    )?;

    Ok(modes.try_into().unwrap())
}
```

2. 4. 7. Operating Mode(11)

Value	Operating Mode	Description
0	Current Control Mode	DYNAMIXEL only controls current(torque) regardless of speed and position. This mode is ideal for a gripper or a system that only uses current(torque) control or a system that has additional velocity/position controllers.
1	Velocity Control Mode	This mode controls velocity. This mode is identical to the Wheel Mode(endless) from existing DYNAMIXEL. This mode is ideal for wheel-type robots.
3(Default)	Position Control Mode	This mode controls position. This mode is identical to the Joint Mode from existing DYNAMIXEL. Operating position range is limited by the Max Position Limit(48) and the Min Position Limit(52). This mode is ideal for articulated robots that each joint rotates less than 360 degrees.
4	Extended Position Control Mode(Multi-turn)	This mode controls position. This mode is identical to the Multi-turn Position Control from existing DYNAMIXEL. 512 turns are supported(-256[rev] ~ 256[rev]). This mode is ideal for multi-turn wrists or conveyer systems or a system that requires an additional reduction gear. Note that Max Position Limit(48), Min Position Limit(52) are not used on Extended Position Control Mode.
5	Current-based Position Control Mode	This mode controls both position and current(torque). Up to 512 turns are supported(-256[rev] ~ 256[rev]). This mode is ideal for a system that requires both position and current control such as articulated robots or grippers.
16	PWM Control Mode (Voltage Control Mode)	This mode directly controls PWM output. (Voltage Control Mode)

https://emanual.robotis.com/docs/en/dxl/x/xl330-m288/#operating-mode-11