

DrumRobot

다음에 의해 생성됨 : Doxygen 1.10.0



<b>1 계통도 색인</b>	<b>1</b>
1.1 클래스 계통도	1
<b>2 클래스 색인</b>	<b>3</b>
2.1 클래스 목록	3
<b>3 파일 색인</b>	<b>5</b>
3.1 파일 목록	5
<b>4 클래스 문서화</b>	<b>7</b>
4.1 CanManager 클래스 참조	7
4.1.1 상세한 설명	9
4.1.2 생성자 & 소멸자 문서화	9
4.1.2.1 CanManager()	9
4.1.2.2 ~CanManager()	9
4.1.3 멤버 함수 문서화	10
4.1.3.1 activateCanPort()	10
4.1.3.2 checkAllMotors()	10
4.1.3.3 checkCanPortsStatus()	11
4.1.3.4 checkConnection()	11
4.1.3.5 clearCanBuffer()	12
4.1.3.6 clearReadBuffers()	12
4.1.3.7 createSocket()	13
4.1.3.8 deactivateCanPort()	13
4.1.3.9 distributeFramesToMotors()	13
4.1.3.10 getCanPortStatus()	14
4.1.3.11 initializeCAN()	15
4.1.3.12 list_and_activate_available_can_ports()	15
4.1.3.13 readFramesFromAllSockets()	16
4.1.3.14 recvToBuff()	16
4.1.3.15 restartCanPorts()	17
4.1.3.16 rxFrame()	17
4.1.3.17 sendAndRecv()	18
4.1.3.18 sendFromBuff()	18
4.1.3.19 setMotorsSocket()	18
4.1.3.20 setSocketsTimeout()	19
4.1.3.21 setSocketTimeout()	20
4.1.3.22 txFrame()	20
4.1.4 멤버 데이터 문서화	21
4.1.4.1 ERR_SOCKET_CONFIGURE_FAILURE	21
4.1.4.2 ERR_SOCKET_CREATE_FAILURE	21
4.1.4.3 ifnames	21
4.1.4.4 isConnected	21

4.1.4.5 maxoncmd	21
4.1.4.6 maxonCnt	22
4.1.4.7 motors	22
4.1.4.8 sockets	22
4.1.4.9 tempFrames	22
4.1.4.10 tmotorcmd	22
4.2 DrumRobot 클래스 참조	23
4.2.1 상세한 설명	24
4.2.2 생성자 & 소멸자 문서화	25
4.2.2.1 DrumRobot()	25
4.2.3 멤버 함수 문서화	25
4.2.3.1 checkUserInput()	25
4.2.3.2 clearMotorsSendBuffer()	26
4.2.3.3 DeactivateControlTask()	26
4.2.3.4 displayAvailableCommands()	27
4.2.3.5 idealStateRoutine()	27
4.2.3.6 initializecanManager()	27
4.2.3.7 initializeMotors()	28
4.2.3.8 initializePathManager()	29
4.2.3.9 kbhit()	30
4.2.3.10 MaxonDisable()	30
4.2.3.11 MaxonEnable()	31
4.2.3.12 motorSettingCmd()	32
4.2.3.13 parse_and_save_to_csv()	33
4.2.3.14 printCurrentPositions()	34
4.2.3.15 processInput()	34
4.2.3.16 RecieveLoop()	35
4.2.3.17 recvLoopForThread()	35
4.2.3.18 save_to_txt_inputData()	36
4.2.3.19 SendLoop()	36
4.2.3.20 sendLoopForThread()	38
4.2.3.21 SendReadyLoop()	38
4.2.3.22 setMaxonMode()	39
4.2.3.23 stateMachine()	39
4.2.4 멤버 데이터 문서화	40
4.2.4.1 canManager	40
4.2.4.2 homeManager	41
4.2.4.3 isBack	41
4.2.4.4 isReady	41
4.2.4.5 maxoncmd	41
4.2.4.6 motors	41
4.2.4.7 NUM_FRAMES	41

4.2.4.8 pathManager	42
4.2.4.9 sensor	42
4.2.4.10 systemState	42
4.2.4.11 testManager	42
4.2.4.12 TIME_THRESHOLD_MS	42
4.2.4.13 tmotorcmd	42
4.2.4.14 writeFailCount	43
4.3 GenericMotor 클래스 참조	43
4.3.1 상세한 설명	44
4.3.2 멤버 함수 문서화	44
4.3.2.1 clearReceiveBuffer()	44
4.3.2.2 clearSendBuffer()	44
4.3.3 멤버 데이터 문서화	45
4.3.3.1 currentPos	45
4.3.3.2 currentTor	45
4.3.3.3 currentVel	45
4.3.3.4 cwDir	45
4.3.3.5 desPos	45
4.3.3.6 desTor	45
4.3.3.7 desVel	46
4.3.3.8 interFaceName	46
4.3.3.9 isConected	46
4.3.3.10 isHomed	46
4.3.3.11 Kd	46
4.3.3.12 Kp	46
4.3.3.13 nodeId	47
4.3.3.14 recieveBuffer	47
4.3.3.15 rMax	47
4.3.3.16 rMin	47
4.3.3.17 sendBuffer	47
4.3.3.18 socket	47
4.4 HomeManager 클래스 참조	48
4.4.1 상세한 설명	49
4.4.2 생성자 & 소멸자 문서화	49
4.4.2.1 HomeManager()	49
4.4.3 멤버 함수 문서화	49
4.4.3.1 displayHomingStatus()	49
4.4.3.2 FixMotorPosition()	50
4.4.3.3 HomeTMotor()	50
4.4.3.4 mainLoop()	52
4.4.3.5 MaxonDisable()	52
4.4.3.6 MaxonEnable()	53

4.4.3.7 MoveTMotorToSensorLocation()	54
4.4.3.8 PromptUserForHoming()	55
4.4.3.9 RotateTMotor()	56
4.4.3.10 SetMaxonHome()	57
4.4.3.11 setMaxonMode()	58
4.4.3.12 SetTmotorHome()	59
4.4.3.13 UpdateHomingStatus()	59
4.4.4 멤버 데이터 문서화	60
4.4.4.1 canManager	60
4.4.4.2 maxoncmd	60
4.4.4.3 motors	60
4.4.4.4 sensor	60
4.4.4.5 systemState	60
4.4.4.6 tmotorcmd	61
4.5 MaxonCommandParser 클래스 참조	61
4.5.1 상세한 설명	61
4.5.2 멤버 함수 문서화	62
4.5.2.1 getCheck()	62
4.5.2.2 getCSPMode()	62
4.5.2.3 getCSTMode()	62
4.5.2.4 getCSVMode()	63
4.5.2.5 getCurrentThreshold()	63
4.5.2.6 getEnable()	63
4.5.2.7 getFlowingErrorWindow()	64
4.5.2.8 getHomeMode()	64
4.5.2.9 getHomeoffsetDistance()	64
4.5.2.10 getHomePosition()	65
4.5.2.11 getHomingMethodL()	65
4.5.2.12 getHomingMethodR()	65
4.5.2.13 getOperational()	66
4.5.2.14 getPosOffset()	66
4.5.2.15 getQuickStop()	66
4.5.2.16 getStartHoming()	67
4.5.2.17 getStop()	67
4.5.2.18 getSync()	67
4.5.2.19 getTargetPosition()	67
4.5.2.20 getTargetTorque()	68
4.5.2.21 getTargetVelocity()	68
4.5.2.22 getTorqueOffset()	69
4.5.2.23 getVelOffset()	69
4.5.2.24 parseRecieveCommand()	69
4.6 MaxonMotor 클래스 참조	70

4.6.1 상세한 설명	71
4.6.2 생성자 & 소멸자 문서화	71
4.6.2.1 MaxonMotor()	71
4.6.3 멤버 함수 문서화	72
4.6.3.1 clearReceiveBuffer()	72
4.6.3.2 clearSendBuffer()	72
4.6.4 멤버 데이터 문서화	72
4.6.4.1 canReceiveId	72
4.6.4.2 canSendId	72
4.6.4.3 currentPos	72
4.6.4.4 currentTor	73
4.6.4.5 currentVel	73
4.6.4.6 cwDir	73
4.6.4.7 desPos	73
4.6.4.8 desTor	73
4.6.4.9 desVel	73
4.6.4.10 interFaceName	73
4.6.4.11 isConected	74
4.6.4.12 isHomed	74
4.6.4.13 Kd	74
4.6.4.14 Kp	74
4.6.4.15 nodeId	74
4.6.4.16 recieveBuffer	74
4.6.4.17 rMax	75
4.6.4.18 rMin	75
4.6.4.19 rxPdolds	75
4.6.4.20 sendBuffer	75
4.6.4.21 socket	75
4.6.4.22 txPdolds	75
4.7 PathManager 클래스 참조	76
4.7.1 상세한 설명	77
4.7.2 생성자 & 소멸자 문서화	77
4.7.2.1 PathManager()	77
4.7.3 멤버 함수 문서화	77
4.7.3.1 ApplyDir()	77
4.7.3.2 connect()	78
4.7.3.3 fkfun()	78
4.7.3.4 GetArr()	78
4.7.3.5 getDrummingPosAndAng()	79
4.7.3.6 GetDrumPositoin()	79
4.7.3.7 getMotorPos()	80
4.7.3.8 GetMusicSheet()	81

4.7.3.9 getQ1AndQ2()	81
4.7.3.10 getQ3AndQ4()	82
4.7.3.11 iconnect()	83
4.7.3.12 IKfun()	84
4.7.3.13 Motors_sendBuffer()	85
4.7.3.14 PathLoopTask()	86
4.7.4 멤버 데이터 문서화	86
4.7.4.1 backarr	86
4.7.4.2 bpm	86
4.7.4.3 c_L	87
4.7.4.4 c_MotorAngle	87
4.7.4.5 c_R	87
4.7.4.6 canManager	87
4.7.4.7 ElbowAngle_hit	87
4.7.4.8 ElbowAngle_ready	87
4.7.4.9 l_wrist	87
4.7.4.10 LA	88
4.7.4.11 left_inst	88
4.7.4.12 LF	88
4.7.4.13 line	88
4.7.4.14 motor_dir	88
4.7.4.15 motor_mapping	89
4.7.4.16 motors	89
4.7.4.17 MParser	89
4.7.4.18 n_inst	89
4.7.4.19 p	89
4.7.4.20 P1	89
4.7.4.21 P2	89
4.7.4.22 p_L	90
4.7.4.23 p_R	90
4.7.4.24 Q1	90
4.7.4.25 Q2	90
4.7.4.26 Q3	90
4.7.4.27 Q4	90
4.7.4.28 R	90
4.7.4.29 r_wrist	91
4.7.4.30 RA	91
4.7.4.31 RF	91
4.7.4.32 right_inst	91
4.7.4.33 s	91
4.7.4.34 standby	91
4.7.4.35 systemState	91



4.7.4.36 time_arr	92
4.7.4.37 total	92
4.7.4.38 TParser	92
4.7.4.39 v	92
4.7.4.40 wrist	92
4.7.4.41 WristAngle_hit	92
4.7.4.42 WristAngle_ready	92
4.7.4.43 z0	93
4.8 SystemState 구조체 참조	93
4.8.1 상세한 설명	93
4.8.2 생성자 & 소멸자 문서화	93
4.8.2.1 SystemState()	93
4.8.3 멤버 데이터 문서화	94
4.8.3.1 homeMode	94
4.8.3.2 main	94
4.9 TestManager 클래스 참조	94
4.9.1 상세한 설명	96
4.9.2 생성자 & 소멸자 문서화	96
4.9.2.1 TestManager()	96
4.9.3 멤버 함수 문서화	96
4.9.3.1 dct_fun()	96
4.9.3.2 FixMotorPosition()	97
4.9.3.3 InitializeParameters()	98
4.9.3.4 kbhit()	99
4.9.3.5 mainLoop()	99
4.9.3.6 mkArr()	100
4.9.3.7 move()	102
4.9.3.8 multiTestLoop()	102
4.9.3.9 parse_and_save_to_csv()	105
4.9.3.10 SendLoop()	106
4.9.3.11 setMaxonMode()	107
4.9.3.12 TestArr()	108
4.9.3.13 TestStick()	109
4.9.3.14 TestStickLoop()	111
4.9.3.15 TuningLoopTask()	112
4.9.3.16 TuningMaxonCSP()	115
4.9.3.17 TuningMaxonCST()	117
4.9.3.18 TuningMaxonCSV()	119
4.9.3.19 TuningTmotor()	121
4.9.4 멤버 데이터 문서화	123
4.9.4.1 canManager	123
4.9.4.2 InputData	123

4.9.4.3 maxoncmd	123
4.9.4.4 motors	123
4.9.4.5 systemState	123
4.9.4.6 tmotorcmd	124
4.10 TMotor 클래스 참조	124
4.10.1 상세한 설명	125
4.10.2 생성자 & 소멸자 문서화	125
4.10.2.1 TMotor()	125
4.10.3 멤버 함수 문서화	125
4.10.3.1 clearReceiveBuffer()	125
4.10.3.2 clearSendBuffer()	126
4.10.4 멤버 데이터 문서화	126
4.10.4.1 currentPos	126
4.10.4.2 currentTor	126
4.10.4.3 currentVel	126
4.10.4.4 cwDir	126
4.10.4.5 desPos	126
4.10.4.6 desTor	127
4.10.4.7 desVel	127
4.10.4.8 interFaceName	127
4.10.4.9 isConected	127
4.10.4.10 isHomed	127
4.10.4.11 Kd	127
4.10.4.12 Kp	128
4.10.4.13 motorType	128
4.10.4.14 nodeId	128
4.10.4.15 recieveBuffer	128
4.10.4.16 rMax	128
4.10.4.17 rMin	128
4.10.4.18 sendBuffer	129
4.10.4.19 sensorBit	129
4.10.4.20 socket	129
4.11 TMotorCommandParser 클래스 참조	129
4.11.1 상세한 설명	130
4.11.2 멤버 함수 문서화	130
4.11.2.1 floatToUInt()	130
4.11.2.2 getCheck()	131
4.11.2.3 getControlMode()	131
4.11.2.4 getExit()	131
4.11.2.5 getQuickStop()	132
4.11.2.6 getZero()	132
4.11.2.7 parseRecieveCommand()	132

4.11.2.8 parseSendCommand()	133
4.11.2.9 setMotorLimits()	134
4.11.2.10 uintToFloat()	135
4.11.3 멤버 데이터 문서화	135
4.11.3.1 GLOBAL_KD_MAX	135
4.11.3.2 GLOBAL_KD_MIN	135
4.11.3.3 GLOBAL_KP_MAX	135
4.11.3.4 GLOBAL_KP_MIN	136
4.11.3.5 GLOBAL_P_MAX	136
4.11.3.6 GLOBAL_P_MIN	136
4.11.3.7 GLOBAL_T_MAX	136
4.11.3.8 GLOBAL_T_MIN	136
4.11.3.9 GLOBAL_V_MAX	136
4.11.3.10 GLOBAL_V_MIN	136
<b>5 파일 문서화</b>	<b>137</b>
5.1 CanManager.hpp	137
5.2 HomeManager.hpp	138
5.3 PathManager.hpp	139
5.4 TestManager.hpp	141
5.5 CommandParser.hpp	142
5.6 Motor.hpp	143
5.7 DrumRobot.hpp	144
5.8 SystemState.hpp	145
5.9 CanManager.cpp	145
5.10 CommandParser.cpp	152
5.11 DrumRobot.cpp	159
5.12 HomeManager.cpp	171
5.13 main.cpp	179
5.14 Motor.cpp	180
5.15 PathManager.cpp	180
5.16 Sensor.cpp	188
5.17 TestManager.cpp	189
<b>Index</b>	<b>211</b>



# Chapter 1

## 계통도 색인

### 1.1 클래스 계통도

이 상속 목록은 완전하진 않지만 알파벳순으로 대략적으로 정렬되어있습니다.:

CanManager . . . . .	7
DrumRobot . . . . .	23
GenericMotor . . . . .	43
MaxonMotor . . . . .	70
TMotor . . . . .	124
HomeManager . . . . .	48
MaxonCommandParser . . . . .	61
PathManager . . . . .	76
SystemState . . . . .	93
TestManager . . . . .	94
TMotorCommandParser . . . . .	129



## Chapter 2

# 클래스 색인

### 2.1 클래스 목록

다음은 클래스, 구조체, 공용체 그리고 인터페이스들입니다. (간략한 설명만을 보여줍니다) :

<a href="#">CanManager</a>	CAN 통신을 통해 모터와의 연결을 관리하고, 데이터 송수신 및 명령 실행을 담당하는 클래스입니다 . . . . .	7
<a href="#">DrumRobot</a>	드럼 로봇의 메인 제어 클래스 . . . . .	23
<a href="#">GenericMotor</a>	모든 모터 타입의 기본이 되는 범용 모터 클래스입니다 . . . . .	43
<a href="#">HomeManager</a>	모터의 홈 위치 설정 및 관리를 담당하는 클래스입니다 . . . . .	48
<a href="#">MaxonCommandParser</a>	Maxon 모터 명령어를 파싱하는 클래스입니다 . . . . .	61
<a href="#">MaxonMotor</a>	Maxon 모터를 위한 클래스입니다. GenericMotor를 상속받습니다 . . . . .	70
<a href="#">PathManager</a>	. . . . .	76
<a href="#">SystemState</a>	시스템의 전반적인 상태를 관리합니다 . . . . .	93
<a href="#">TestManager</a>	모터의 성능 테스트 및 파라미터 튜닝을 위한 클래스입니다 . . . . .	94
<a href="#">TMotor</a>	TMotor를 위한 클래스입니다. GenericMotor를 상속받습니다 . . . . .	124
<a href="#">TMotorCommandParser</a>	TMotor 명령어를 파싱하는 클래스입니다 . . . . .	129





## Chapter 3

# 파일 색인

### 3.1 파일 목록

다음은 문서화된 모든 파일에 대한 목록입니다. (간략한 설명만을 보여줍니다) :

include/managers/CanManager.hpp	137
include/managers/HomeManager.hpp	138
include/managers/PathManager.hpp	139
include/managers/TestManager.hpp	141
include/motors/CommandParser.hpp	142
include/motors/Motor.hpp	143
include/tasks/DrumRobot.hpp	144
include/tasks/SystemState.hpp	145
src/CanManager.cpp	145
src/CommandParser.cpp	152
src/DrumRobot.cpp	159
src/HomeManager.cpp	171
src/main.cpp	179
src/Motor.cpp	180
src/PathManager.cpp	180
src/Sensor.cpp	188
src/TestManager.cpp	189



## Chapter 4

# 클래스 문서화

### 4.1 CanManager 클래스 참조

CAN 통신을 통해 모터와의 연결을 관리하고, 데이터 송수신 및 명령 실행을 담당하는 클래스입니다.

```
#include <CanManager.hpp>
```

#### Public 멤버 함수

- `CanManager` (`std::map< std::string, std::shared_ptr< GenericMotor > > &motorsRef`)  
`CanManager` 클래스의 생성자.
- `~CanManager` ()  
`CanManager` 클래스의 소멸자.
- `void initializeCAN` ()  
CAN 포트를 초기화하고, 모터와의 통신을 준비합니다.
- `void restartCanPorts` ()  
모든 CAN 포트를 재시작합니다.
- `void setSocketsTimeout` (int sec, int usec)  
소켓의 타임아웃을 설정합니다.
- `void checkCanPortsStatus` ()  
모든 CAN 포트의 상태를 확인하고, 연결 상태를 업데이트합니다.
- `void setMotorsSocket` ()  
모터와의 통신을 위한 소켓을 설정합니다.
- `bool checkConnection` (`std::shared_ptr< GenericMotor > motor`)  
지정된 모터와의 연결 상태를 확인합니다.
- `bool checkAllMotors` ()  
모든 모터와의 연결 상태를 확인합니다.
- `bool sendAndRecv` (`std::shared_ptr< GenericMotor > &motor`, `struct can_frame &frame`)  
지정된 모터로 데이터를 송신하고 응답을 수신합니다.
- `bool sendFromBuff` (`std::shared_ptr< GenericMotor > &motor`)  
지정된 모터의 송신 버퍼에서 데이터를 송신합니다.
- `bool recvToBuff` (`std::shared_ptr< GenericMotor > &motor`, int readCount)  
지정된 모터의 수신 버퍼로 데이터를 수신합니다.
- `bool txFrame` (`std::shared_ptr< GenericMotor > &motor`, `struct can_frame &frame`)  
모터로 데이터 프레임을 송신합니다.

- bool `rxFrame` (`std::shared_ptr< GenericMotor > &motor`, `struct can_frame &frame`)  
모터로부터 데이터 프레임을 수신합니다.
- void `readFramesFromAllSockets` ()  
모든 소켓에서 CAN 프레임을 읽습니다.
- void `distributeFramesToMotors` ()  
수신된 CAN 프레임을 적절한 모터에 분배합니다.
- void `clearReadBuffers` ()  
읽기 버퍼를 비웁니다.

## Public 속성

- `std::map< std::string, int > sockets`  
모터와 통신하는 소켓의 맵.
- `std::map< std::string, bool > isConnected`  
모터의 연결 상태를 나타내는 맵.
- int `maxonCnt` =0  
연결된 Maxon 모터의 수.

## 정적 Public 속성

- static const int `ERR_SOCKET_CREATE_FAILURE` = -1  
소켓 생성 실패시 반환되는 오류 코드입니다.
- static const int `ERR_SOCKET_CONFIGURE_FAILURE` = -2  
소켓 설정 실패시 반환되는 오류 코드입니다.

## Private 멤버 함수

- bool `getCanPortStatus` (`const char *port`)  
특정 CAN 포트의 상태를 반환합니다.
- void `activateCanPort` (`const char *port`)  
특정 CAN 포트를 활성화합니다.
- void `list_and_activate_available_can_ports` ()  
사용 가능한 모든 CAN 포트를 활성화합니다.
- void `deactivateCanPort` (`const char *port`)  
특정 CAN 포트를 비활성화합니다.
- int `createSocket` (`const std::string &ifname`)  
특정 인터페이스에 대한 소켓을 생성합니다.
- int `setSocketTimeout` (`int socket`, `int sec`, `int usec`)  
소켓의 타임아웃을 설정합니다.
- void `clearCanBuffer` (`int canSocket`)  
특정 CAN 소켓의 버퍼를 비웁니다.

**Private 속성**

- `std::vector< std::string > ifnames`  
사용 가능한 인터페이스 이름 목록.
- `std::map< std::string, std::shared_ptr< GenericMotor > > & motors`  
연결된 모터들의 참조를 저장하는 맵.
- `TMotorCommandParser tmotorcmd`  
T 모터 명령 파서.
- `MaxonCommandParser maxoncmd`  
Maxon 모터 명령 파서.
- `std::map< int, std::vector< can_frame > > tempFrames`  
임시 프레임 저장소.

**4.1.1 상세한 설명**

CAN 통신을 통해 모터와의 연결을 관리하고, 데이터 송수신 및 명령 실행을 담당하는 클래스입니다.

이 클래스는 모터와의 통신을 위한 소켓 설정, 연결 상태 확인, 데이터 송수신 등의 기능을 제공합니다. 모터 제어 명령을 생성하고, CAN 프레임을 통해 모터로 전송하는 역할을 합니다.

[CanManager.hpp](#) 파일의 38 번째 라인에서 정의되었습니다.

**4.1.2 생성자 & 소멸자 문서화****4.1.2.1 CanManager()**

```
CanManager::CanManager (
    std::map< std::string, std::shared_ptr< GenericMotor > > & motorsRef )
```

[CanManager](#) 클래스의 생성자.

매개변수

<code>motorsRef</code>	모터 객체들의 참조를 저장하는 맵. 키는 모터의 이름, 값은 모터 객체에 대한 <code>shared_ptr</code> 입니다.
------------------------	--

[CanManager.cpp](#) 파일의 2 번째 라인에서 정의되었습니다.

```
00003 : motors(motorsRef)
00004 {
00005 }
```

**4.1.2.2 ~CanManager()**

```
CanManager::~CanManager ( )
```

[CanManager](#) 클래스의 소멸자.

[CanManager.cpp](#) 파일의 7 번째 라인에서 정의되었습니다.

```
00008 {
```

```

00009      // 모든 소켓 닫기
00010      for (const auto &socketPair : sockets)
00011      {
00012          if (socketPair.second >= 0)
00013          {
00014              close(socketPair.second);
00015          }
00016      }
00017      sockets.clear();
00018 }

```

### 4.1.3 멤버 함수 문서화

#### 4.1.3.1 activateCanPort()

```

void CanManager::activateCanPort (
    const char * port ) [private]

```

특정 CAN 포트를 활성화합니다.

[CanManager.cpp](#) 파일의 179 번째 라인에서 정의되었습니다.

```

00180 {
00181     char command1[100], command2[100];
00182     snprintf(command1, sizeof(command1), "sudo ip link set %s type can bitrate 1000000 sample-point
0.850", port);
00183     snprintf(command2, sizeof(command2), "sudo ip link set %s up", port);
00184
00185     int ret1 = system(command1);
00186     int ret2 = system(command2);
00187
00188     if (ret1 != 0 || ret2 != 0)
00189     {
00190         fprintf(stderr, "Failed to activate port: %s\n", port);
00191         exit(1); // 또는 다른 에러 처리
00192     }
00193 }

```

#### 4.1.3.2 checkAllMotors()

```

bool CanManager::checkAllMotors ( )

```

모든 모터와의 연결 상태를 확인합니다.

반환값

모든 모터와의 연결이 성공적이면 true, 하나라도 실패하면 false를 반환합니다.

[CanManager.cpp](#) 파일의 557 번째 라인에서 정의되었습니다.

```

00558 {
00559     bool allMotorsChecked = true;
00560     for (auto &motorPair : motors)
00561     {
00562         std::string name = motorPair.first;
00563         auto &motor = motorPair.second;
00564
00565         if (!checkConnection(motor))
00566         {
00567             allMotorsChecked = false;
00568         }
00569     }
00570     return allMotorsChecked;
00571 }

```

## 4.1.3.3 checkCanPortsStatus()

```
void CanManager::checkCanPortsStatus ( )
```

모든 CAN 포트의 상태를 확인하고, 연결 상태를 업데이트합니다.

CanManager.cpp 파일의 91 번째 라인에서 정의되었습니다.

```
00092 {
00093
00094     for (const auto &ifname : this->ifnames)
00095     {
00096         isConnected[ifname] = getCanPortStatus(ifname.c_str());
00097
00098         if (!isConnected[ifname])
00099         {
00100             std::cout << "Port " << ifname << " is NOT CONNECTED" << std::endl;
00101         }
00102     }
00103
00104     // 모든 포트가 연결된 경우 1, 아니면 0 반환
00105 }
```

## 4.1.3.4 checkConnection()

```
bool CanManager::checkConnection (
    std::shared_ptr< GenericMotor > motor )
```

지정된 모터와의 연결 상태를 확인합니다.

매개변수

motor	연결 상태를 확인할 모터의 shared_ptr.
-------	----------------------------

반환값

연결이 성공적이면 true, 실패하면 false를 반환합니다.

CanManager.cpp 파일의 507 번째 라인에서 정의되었습니다.

```
00508 {
00509     struct can_frame frame;
00510     setSocketTimeout(0, 5000 /*5ms*/);
00511     clearReadBuffers();
00512
00513     if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00514     {
00515         tMotorcmd.getControlMode(*tMotor, &frame);
00516         if (sendAndRecv(motor, frame))
00517         {
00518             std::tuple<int, float, float, float> parsedData = tMotorcmd.parseRecieveCommand(*tMotor,
00519 &frame);
00519             motor->currentPos = std::get<1>(parsedData);
00520             motor->currentVel = std::get<2>(parsedData);
00521             motor->currentTor = std::get<3>(parsedData);
00522             motor->isConected = true;
00523         }
00524         else
00525         {
00526             return false;
00527         }
00528     }
00529     else if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00530     {
00531         maxoncmd.getSync(&frame);
00532         txFrame(motor, frame);
00533         motor->clearReceiveBuffer();
00534         if (recvToBuff(motor, maxonCnt))
00535         {
00536             while (!motor->recieveBuffer.empty())
```

```

00537         {
00538             frame = motor->recieveBuffer.front();
00539             if (frame.can_id == maxonMotor->rxPdoIds[0])
00540             {
00541                 std::tuple<int, float, float> parsedData =
maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
00542                 motor->currentPos = std::get<1>(parsedData);
00543                 motor->currentTor = std::get<2>(parsedData);
00544                 motor->isConected = true;
00545             }
00546             motor->recieveBuffer.pop();
00547         }
00548     }
00549     else
00550     {
00551         return false;
00552     }
00553 }
00554 return true;
00555 }

```

#### 4.1.3.5 clearCanBuffer()

```

void CanManager::clearCanBuffer (
    int canSocket ) [private]

```

특정 CAN 소켓의 버퍼를 비웁니다.

CanManager.cpp 파일의 273 번째 라인에서 정의되었습니다.

```

00274 {
00275     struct can_frame frame;
00276     fd_set readSet;
00277     struct timeval timeout;
00278
00279     // 수신 대기 시간 설정
00280     timeout.tv_sec = 0;
00281     timeout.tv_usec = 0; // 즉시 반환
00282
00283     while (true)
00284     {
00285         FD_ZERO(&readSet);
00286         FD_SET(canSocket, &readSet);
00287
00288         // 소켓에서 읽을 데이터가 있는지 확인
00289         int selectRes = select(canSocket + 1, &readSet, NULL, NULL, &timeout);
00290
00291         if (selectRes > 0)
00292         {
00293             // 수신 버퍼에서 데이터 읽기
00294             ssize_t nbytes = read(canSocket, &frame, sizeof(struct can_frame));
00295
00296             if (nbytes <= 0)
00297             {
00298                 // 읽기 실패하거나 더 이상 읽을 데이터가 없음
00299                 break;
00300             }
00301         }
00302         else
00303         {
00304             // 읽을 데이터가 없음
00305             break;
00306         }
00307     }
00308 }

```

#### 4.1.3.6 clearReadBuffers()

```

void CanManager::clearReadBuffers ( )

```

읽기 버퍼를 비웁니다.

CanManager.cpp 파일의 264 번째 라인에서 정의되었습니다.

```

00265 {
00266     for (const auto &socketPair : sockets)
00267     {
00268         int socket_fd = socketPair.second;
00269         clearCanBuffer(socket_fd);
00270     }
00271 }

```



#### 4.1.3.7 createSocket()

```
int CanManager::createSocket (
    const std::string & ifname ) [private]
```

특정 인터페이스에 대한 소켓을 생성합니다.

CanManager.cpp 파일의 144 번째 라인에서 정의되었습니다.

```
00145 {
00146     int result;
00147     struct sockaddr_can addr;
00148     struct ifreq ifr;
00149
00150     int localSocket = socket(PF_CAN, SOCK_RAW, CAN_RAW); // 지역 변수로 소켓 생성
00151     if (localSocket < 0)
00152     {
00153         return ERR_SOCKET_CREATE_FAILURE;
00154     }
00155
00156     memset(&ifr, 0, sizeof(struct ifreq));
00157     memset(&addr, 0, sizeof(struct sockaddr_can));
00158
00159     strcpy(ifr.ifr_name, ifname.c_str());
00160     result = ioctl(localSocket, SIOCGIFINDEX, &ifr);
00161     if (result < 0)
00162     {
00163         close(localSocket);
00164         return ERR_SOCKET_CREATE_FAILURE;
00165     }
00166
00167     addr.can_ifindex = ifr.ifr_ifindex;
00168     addr.can_family = AF_CAN;
00169
00170     if (bind(localSocket, (struct sockaddr *)&addr, sizeof(addr)) < 0)
00171     {
00172         close(localSocket);
00173         return ERR_SOCKET_CREATE_FAILURE;
00174     }
00175
00176     return localSocket; // 생성된 소켓 디스크립터 반환
00177 }
```

#### 4.1.3.8 deactivateCanPort()

```
void CanManager::deactivateCanPort (
    const char * port ) [private]
```

특정 CAN 포트를 비활성화합니다.

CanManager.cpp 파일의 253 번째 라인에서 정의되었습니다.

```
00254 {
00255     char command[100];
00256     sprintf(command, sizeof(command), "sudo ip link set %s down", port);
00257     int ret = system(command);
00258     if (ret != 0)
00259     {
00260         fprintf(stderr, "Failed to down port: %s\n", port);
00261     }
00262 }
```

#### 4.1.3.9 distributeFramesToMotors()

```
void CanManager::distributeFramesToMotors ( )
```

수신된 CAN 프레임을 적절한 모터에 분배합니다.

CanManager.cpp 파일의 468 번째 라인에서 정의되었습니다.

```
00469 {
00470     for (auto &motor_pair : motors)
```

```

00471     {
00472         auto &motor = motor_pair.second;
00473
00474         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00475         {
00476             // TMotor 처리
00477             for (auto &frame : tempFrames[motor->socket])
00478             {
00479                 if (frame.data[0] == tMotor->nodeId)
00480                 {
00481                     std::tuple<int, float, float, float> parsedData =
00482                         tmotorcmd.parseRecieveCommand(*tMotor, &frame);
00483                     tMotor->currentPos = std::get<1>(parsedData);
00484                     tMotor->currentVel = std::get<2>(parsedData);
00485                     tMotor->currentTor = std::get<3>(parsedData);
00486                     tMotor->recieveBuffer.push(frame);
00487                 }
00488             }
00489             else if (std::shared_ptr<MaxonMotor> maxonMotor =
00490                 std::dynamic_pointer_cast<MaxonMotor>(motor))
00491             {
00492                 // MaxonMotor 처리
00493                 for (auto &frame : tempFrames[motor->socket])
00494                 {
00495                     if (frame.can_id == maxonMotor->txPdoIds[0])
00496                     {
00497                         std::tuple<int, float, float> parsedData =
00498                             maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
00499                         maxonMotor->currentPos = std::get<1>(parsedData);
00500                         maxonMotor->currentTor = std::get<2>(parsedData);
00501                         maxonMotor->recieveBuffer.push(frame);
00502                     }
00503                 }
00504             }
00505             tempFrames.clear(); // 프레임 분배 후 임시 배열 비우기
00506         }
00507     }

```

#### 4.1.3.10 getCanPortStatus()

```

bool CanManager::getCanPortStatus (
    const char * port ) [private]

```

특정 CAN 포트의 상태를 반환합니다.

CanManager.cpp 파일의 111 번째 라인에서 정의되었습니다.

```

00112 {
00113     char command[50];
00114     snprintf(command, sizeof(command), "ip link show %s", port);
00115
00116     FILE *fp = popen(command, "r");
00117     if (fp == NULL)
00118     {
00119         perror("Error opening pipe");
00120         return false;
00121     }
00122
00123     char output[1024];
00124     while (fgets(output, sizeof(output) - 1, fp) != NULL)
00125     {
00126         if (strstr(output, "DOWN") || strstr(output, "does not exist"))
00127         {
00128             pclose(fp);
00129             return false;
00130         }
00131         else if (strstr(output, "UP"))
00132         {
00133             pclose(fp);
00134             return true;
00135         }
00136     }
00137
00138     perror("fgets failed");
00139     printf("Errno: %d\n", errno); // errno 값을 출력
00140     pclose(fp);
00141     return false;
00142 }

```

## 4.1.3.11 initializeCAN()

```
void CanManager::initializeCAN ( )
```

CAN 포트를 초기화하고, 모터와의 통신을 준비합니다.

CanManager.cpp 파일의 24 번째 라인에서 정의되었습니다.

```
00025 {
00026     list_and_activate_available_can_ports();
00027     for (const auto &ifname : this->ifnames)
00028     {
00029         std::cout << "Processing interface: " << ifname << std::endl;
00030         int hsocket = createSocket(ifname);
00031         if (hsocket < 0)
00032         {
00033             std::cerr << "Socket creation error for interface: " << ifname << std::endl;
00034             exit(EXIT_FAILURE);
00035         }
00036         sockets[ifname] = hsocket;
00037         isConnected[ifname] = true;
00038         std::cout << "Socket created for " << ifname << ": " << hsocket << std::endl;
00039     }
00040 }
00041 }
```

## 4.1.3.12 list\_and\_activate\_available\_can\_ports()

```
void CanManager::list_and_activate_available_can_ports ( ) [private]
```

사용 가능한 모든 CAN 포트를 활성화합니다.

CanManager.cpp 파일의 195 번째 라인에서 정의되었습니다.

```
00196 {
00197     int portCount = 0; // CAN 포트 수를 세기 위한 변수
00198
00199     FILE *fp = popen("ip link show | grep can", "r");
00200     if (fp == nullptr)
00201     {
00202         perror("No available CAN port");
00203         exit(1);
00204     }
00205
00206     char output[1024];
00207     while (fgets(output, sizeof(output) - 1, fp) != nullptr)
00208     {
00209         std::string line(output);
00210         std::istringstream iss(line);
00211         std::string skip, port;
00212         iss >> skip >> port;
00213
00214         // 콜론 제거
00215         if (!port.empty() && port.back() == ':')
00216         {
00217             port.pop_back();
00218         }
00219
00220         // 포트 이름이 유효한지 확인
00221         if (!port.empty() && port.find("can") == 0)
00222         {
00223             portCount++;
00224             if (!getCanPortStatus(port.c_str()))
00225             {
00226                 printf("%s is DOWN, activating...\n", port.c_str());
00227                 activateCanPort(port.c_str());
00228             }
00229             else
00230             {
00231                 printf("%s is already UP\n", port.c_str());
00232             }
00233
00234             this->ifnames.push_back(port); // 포트 이름을 ifnames 벡터에 추가
00235         }
00236     }
00237
00238     if (feof(fp) == 0)
00239     {
00240         perror("fgets failed");
```

```

00241         printf("Errno: %d\n", errno);
00242     }
00243
00244     pclose(fp);
00245
00246     if (portCount == 0)
00247     {
00248         printf("No CAN port found. Exiting...\n");
00249         exit(1);
00250     }
00251 }

```

#### 4.1.3.13 readFramesFromAllSockets()

```
void CanManager::readFramesFromAllSockets ( )
```

모든 소켓에서 CAN 프레임을 읽습니다.

CanManager.cpp 파일의 455 번째 라인에서 정의되었습니다.

```

00456 {
00457     struct can_frame frame;
00458     for (const auto &socketPair : sockets)
00459     {
00460         int socket_fd = socketPair.second;
00461         while (read(socket_fd, &frame, sizeof(frame)) == sizeof(frame))
00462         {
00463             tempFrames[socket_fd].push_back(frame);
00464         }
00465     }
00466 }

```

#### 4.1.3.14 recvToBuff()

```

bool CanManager::recvToBuff (
    std::shared_ptr< GenericMotor > & motor,
    int readCount )

```

지정된 모터의 수신 버퍼로 데이터를 수신합니다.

매개변수

motor	데이터를 수신할 모터의 shared_ptr.
readCount	읽을 데이터 프레임의 수.

반환값

데이터 수신이 성공적이면 true, 실패하면 false를 반환합니다.

CanManager.cpp 파일의 371 번째 라인에서 정의되었습니다.

```

00372 {
00373     struct can_frame frame;
00374     for (int i = 0; i < readCount; i++)
00375     {
00376         if (rxFrame(motor, frame))
00377         {
00378             motor->recieveBuffer.push(frame);
00379         }
00380         else
00381         {
00382             return false;
00383         }
00384     }
00385     return true;
00386 }

```

## 4.1.3.15 restartCanPorts()

```
void CanManager::restartCanPorts ( )
```

모든 CAN 포트를 재시작합니다.

CanManager.cpp 파일의 43 번째 라인에서 정의되었습니다.

```
00044 {
00045     // 먼저 모든 포트를 down 시킵니다.
00046     for (const auto &port : ifnames)
00047     {
00048         deactivateCanPort(port.c_str());
00049
00050         int socket_fd = sockets[port];
00051         if (socket_fd >= 0)
00052         {
00053             close(socket_fd); // 기존 소켓을 닫습니다.
00054             sockets[port] = -1; // 소켓 디스크립터 값을 초기화합니다.
00055         }
00056     }
00057
00058     // 각 포트에 대해 새로운 소켓을 생성하고 디스크립터를 업데이트합니다.
00059     for (const auto &port : ifnames)
00060     {
00061         usleep(100000); // 100ms 대기
00062         activateCanPort(port.c_str());
00063
00064         int new_socket_fd = createSocket(port);
00065         if (new_socket_fd < 0)
00066         {
00067             // 새로운 소켓 생성에 실패한 경우 처리
00068             fprintf(stderr, "Failed to create a new socket for port: %s\n", port.c_str());
00069         }
00070         else
00071         {
00072             sockets[port] = new_socket_fd; // 소켓 디스크립터 값을 업데이트합니다.
00073         }
00074     }
00075
00076     setMotorsSocket();
00077 }
```

## 4.1.3.16 rxFrame()

```
bool CanManager::rxFrame (
    std::shared_ptr< GenericMotor > & motor,
    struct can_frame & frame )
```

모터로부터 데이터 프레임을 수신합니다.

매개변수

motor	데이터 프레임을 수신할 모터의 shared_ptr.
frame	수신할 CAN 프레임.

반환값

데이터 프레임 수신에 성공적이면 true, 실패하면 false를 반환합니다.

CanManager.cpp 파일의 339 번째 라인에서 정의되었습니다.

```
00340 {
00341
00342     if (read(motor->socket, &frame, sizeof(frame)) != sizeof(frame))
00343     {
00344         // perror("CAN read error");
00345         return false;
00346     }
00347     return true;
00348 }
```

#### 4.1.3.17 sendAndRecv()

```
bool CanManager::sendAndRecv (
    std::shared_ptr< GenericMotor > & motor,
    struct can_frame & frame )
```

지정된 모터로 데이터를 송신하고 응답을 수신합니다.

매개변수

motor	데이터 송수신할 모터의 shared_ptr.
frame	송신할 CAN 프레임.

반환값

데이터 송수신이 성공적이면 true, 실패하면 false를 반환합니다.

CanManager.cpp 파일의 350 번째 라인에서 정의되었습니다.

```
00351 {
00352     if (!txFrame(motor, frame) || !rxFrame(motor, frame))
00353     {
00354         // perror("Send and receive error");
00355         return false;
00356     }
00357     return true;
00358 }
```

#### 4.1.3.18 sendFromBuff()

```
bool CanManager::sendFromBuff (
    std::shared_ptr< GenericMotor > & motor )
```

지정된 모터의 송신 버퍼에서 데이터를 송신합니다.

매개변수

motor	데이터를 송신할 모터의 shared_ptr.
-------	--------------------------

반환값

데이터 송신이 성공적이면 true, 실패하면 false를 반환합니다.

CanManager.cpp 파일의 360 번째 라인에서 정의되었습니다.

```
00361 {
00362     if (!motor->sendBuffer.empty())
00363     {
00364         struct can_frame frame = motor->sendBuffer.front();
00365         motor->sendBuffer.pop();
00366         return txFrame(motor, frame);
00367     }
00368     return false;
00369 }
```

#### 4.1.3.19 setMotorsSocket()

```
void CanManager::setMotorsSocket ( )
```

모터와의 통신을 위한 소켓을 설정합니다.

CanManager.cpp 파일의 388 번째 라인에서 정의되었습니다.

```
00389 {
00390     struct can_frame frame;
00391     setSocketsTimeout(0, 10000);
00392
00393     // 모든 소켓에 대해 각 모터에 명령을 보내고 응답을 확인
00394     for (const auto &socketPair : sockets)
00395     {
00396         int socket_fd = socketPair.second;
00397
00398         for (auto &motor_pair : motors)
00399         {
00400             auto &motor = motor_pair.second;
00401             clearReadBuffers();
00402
00403             // TMotor 및 MaxonMotor에 대해 적절한 명령 설정
00404             if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00405             {
00406                 tmotorcmd.getCheck(*tMotor, &frame);
00407             }
00408             else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor))
00409             {
00410                 maxoncmd.getCheck(*maxonMotor, &frame);
00411             }
00412             usleep(10000);
00413             // 모터의 현재 소켓을 임시 소켓으로 설정
00414             int original_socket = motor->socket;
00415             motor->socket = socket_fd;
00416             usleep(10000);
00417             // 소켓에 CAN 프레임 보내고 응답 확인
00418             if (sendAndRecv(motor, frame))
00419             {
00420                 motor->isConected = true;
00421             }
00422             else
00423             {
00424                 motor->socket = original_socket;
00425             }
00426         }
00427     }
00428
00429     // 모든 소켓에 대한 검사가 완료된 후, 모터 연결 상태 확인 및 삭제
00430     for (auto it = motors.begin(); it != motors.end(); )
00431     {
00432         std::string name = it->first;
00433         std::shared_ptr<GenericMotor> motor = it->second;
00434         if (motor->isConected)
00435         {
00436             std::cerr << "-----> Motor [" << name << "] is Connected." << std::endl;
00437             ++it;
00438         }
00439         else
00440         {
00441             std::cerr << "Motor [" << name << "] Not Connected." << std::endl;
00442             it = motors.erase(it);
00443         }
00444     }
00445
00446     for (auto &motor_pair : motors)
00447     {
00448         if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00449         {
00450             maxonCnt++;
00451         }
00452     }
00453 }
```

#### 4.1.3.20 setSocketsTimeout()

```
void CanManager::setSocketsTimeout (
    int sec,
    int usec )
```

소켓의 타임아웃을 설정합니다.

매개변수

sec	초 단위의 타임아웃 시간.
usec	마이크로초 단위의 타임아웃 시간.

CanManager.cpp 파일의 79 번째 라인에서 정의되었습니다.

```
00080 {
00081     for (const auto &socketPair : sockets)
00082     {
00083         int socket_fd = socketPair.second;
00084         if (setSocketTimeout(socket_fd, sec, usec) != 0)
00085         {
00086             std::cerr << "Failed to set socket timeout for " << socketPair.first << std::endl;
00087         }
00088     }
00089 }
```

#### 4.1.3.21 setSocketTimeout()

```
int CanManager::setSocketTimeout (
    int socket,
    int sec,
    int usec ) [private]
```

소켓의 타임아웃을 설정합니다.

CanManager.cpp 파일의 310 번째 라인에서 정의되었습니다.

```
00311 {
00312     struct timeval timeout;
00313     timeout.tv_sec = sec;
00314     timeout.tv_usec = usec;
00315     if (setsockopt(socket, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout, sizeof(timeout)) < 0)
00316     {
00317         perror("setsockopt failed");
00318         return -1;
00319     }
00320     return 0;
00321 }
00322
00323 }
```

#### 4.1.3.22 txFrame()

```
bool CanManager::txFrame (
    std::shared_ptr< GenericMotor > & motor,
    struct can_frame & frame )
```

모터로 데이터 프레임을 송신합니다.

매개변수

motor	데이터 프레임을 송신할 모터의 shared_ptr.
frame	송신할 CAN 프레임.

반환값

데이터 프레임 송신이 성공적이면 true, 실패하면 false를 반환합니다.



[CanManager.cpp](#) 파일의 329 번째 라인에서 정의되었습니다.

```
00330 {
00331     if (write(motor->socket, &frame, sizeof(frame)) != sizeof(frame))
00332     {
00333         // perror("CAN write error");
00334         return false;
00335     }
00336     return true;
00337 }
```

## 4.1.4 멤버 데이터 문서화

### 4.1.4.1 ERR\_SOCKET\_CONFIGURE\_FAILURE

```
const int CanManager::ERR_SOCKET_CONFIGURE_FAILURE = -2 [static]
```

소켓 설정 실패시 반환되는 오류 코드입니다.

[CanManager.hpp](#) 파일의 42 번째 라인에서 정의되었습니다.

### 4.1.4.2 ERR\_SOCKET\_CREATE\_FAILURE

```
const int CanManager::ERR_SOCKET_CREATE_FAILURE = -1 [static]
```

소켓 생성 실패시 반환되는 오류 코드입니다.

[CanManager.hpp](#) 파일의 41 번째 라인에서 정의되었습니다.

### 4.1.4.3 ifnames

```
std::vector<std::string> CanManager::ifnames [private]
```

사용 가능한 인터페이스 이름 목록.

[CanManager.hpp](#) 파일의 154 번째 라인에서 정의되었습니다.

### 4.1.4.4 isConnected

```
std::map<std::string, bool> CanManager::isConnected
```

모터의 연결 상태를 나타내는 맵.

[CanManager.hpp](#) 파일의 150 번째 라인에서 정의되었습니다.

### 4.1.4.5 maxoncmd

```
MaxonCommandParser CanManager::maxoncmd [private]
```

Maxon 모터 명령 파서.

[CanManager.hpp](#) 파일의 158 번째 라인에서 정의되었습니다.

#### 4.1.4.6 maxonCnt

```
int CanManager::maxonCnt =0
```

연결된 Maxon 모터의 수.

[CanManager.hpp](#) 파일의 151 번째 라인에서 정의되었습니다.

#### 4.1.4.7 motors

```
std::map<std::string, std::shared_ptr<GenericMotor> >& CanManager::motors [private]
```

연결된 모터들의 참조를 저장하는 맵.

[CanManager.hpp](#) 파일의 155 번째 라인에서 정의되었습니다.

#### 4.1.4.8 sockets

```
std::map<std::string, int> CanManager::sockets
```

모터와 통신하는 소켓의 맵.

[CanManager.hpp](#) 파일의 149 번째 라인에서 정의되었습니다.

#### 4.1.4.9 tempFrames

```
std::map<int, std::vector<can_frame> > CanManager::tempFrames [private]
```

임시 프레임 저장소.

[CanManager.hpp](#) 파일의 160 번째 라인에서 정의되었습니다.

#### 4.1.4.10 tmotorcmd

```
TMotorCommandParser CanManager::tmotorcmd [private]
```

T 모터 명령 파서.

[CanManager.hpp](#) 파일의 157 번째 라인에서 정의되었습니다.

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- include/managers/CanManager.hpp
- src/CanManager.cpp

## 4.2 DrumRobot 클래스 참조

드럼 로봇의 메인 제어 클래스.

```
#include <DrumRobot.hpp>
```

### Public 멤버 함수

- `DrumRobot` (`SystemState` &systemStateRef, `CanManager` &canManagerRef, `PathManager` &pathManagerRef, `HomeManager` &homeManagerRef, `TestManager` &testManagerRef, `std::map<std::string, std::shared_ptr<GenericMotor>>` &motorsRef)  
`DrumRobot` 클래스의 생성자.
- void `stateMachine` ()  
 상태 머신을 실행하는 메소드.
- void `sendLoopForThread` ()  
 송신 루프를 별도의 스레드로 실행하는 메소드.
- void `recvLoopForThread` ()  
 수신 루프를 별도의 스레드로 실행하는 메소드.

### Private 멤버 함수

- void `displayAvailableCommands` () const  
 사용 가능한 명령어를 표시하는 메소드.
- bool `processInput` (const `std::string` &input)  
 사용자 입력을 처리하는 메소드.
- void `idealStateRoutine` ()  
 이상 상태 루틴을 실행하는 메소드.
- void `checkUserInput` ()  
 사용자 입력을 확인하는 메소드.
- void `printCurrentPositions` ()  
 현재 모터 위치를 출력하는 메소드.
- int `kbhit` ()  
 키보드 입력이 있는지 확인하는 메소드.
- void `initializeMotors` ()  
 모터 초기화 메소드.
- void `initializcanManager` ()  
 CAN 매니저 초기화 메소드.
- void `DeactivateControlTask` ()  
 제어 태스크 비활성화 메소드.
- void `motorSettingCmd` ()  
 모터 설정 명령어 메소드.
- void `setMaxonMode` (`std::string` targetMode)  
 Maxon 모드 설정 메소드.
- void `MaxonEnable` ()  
 Maxon 모터 활성화 메소드.
- void `MaxonDisable` ()  
 Maxon 모터 비활성화 메소드.
- void `SendLoop` ()  
 송신 루프 메소드.

- void `save_to_txt_inputData` (const string &csv\_file\_name)  
입력 데이터를 txt 파일로 저장하는 메소드.
- void `SendReadyLoop` ()  
준비 상태 송신 루프 메소드.
- void `initializePathManager` ()  
경로 매니저 초기화 메소드.
- void `clearMotorsSendBuffer` ()  
모터 송신 버퍼 클리어 메소드.
- void `RecieveLoop` ()  
수신 루프 메소드.
- void `parse_and_save_to_csv` (const std::string &csv\_file\_name)  
파싱 후 CSV 파일로 저장하는 메소드.

### Private 속성

- `SystemState` & `systemState`  
시스템 상태 참조.
- `CanManager` & `canManager`  
CAN 매니저 참조.
- `PathManager` & `pathManager`  
경로 매니저 참조.
- `HomeManager` & `homeManager`  
홈 매니저 참조.
- `TestManager` & `testManager`  
테스트 매니저 참조.
- std::map< std::string, std::shared\_ptr< `GenericMotor` > > & `motors`  
모터 객체들의 맵 참조.
- `TMotorCommandParser` `tmotorcmd`  
T 모터 명령 파서.
- `MaxonCommandParser` `maxoncmd`  
Maxon 명령 파서.
- Sensor `sensor`  
센서 객체.
- bool `isReady`  
준비 상태 플래그.
- bool `isBack`  
되돌아가기 플래그.
- int `writeFailCount`  
송신 실패 카운트.
- const int `NUM_FRAMES`  
처리할 프레임 수.
- const int `TIME_THRESHOLD_MS`  
시간 임계값.

### 4.2.1 상세한 설명

드럼 로봇의 메인 제어 클래스.

이 클래스는 드럼 로봇 시스템의 메인 제어 로직을 포함하며, 다양한 매니저 클래스와 상태를 관리합니다.

`DrumRobot.hpp` 파일의 43 번째 라인에서 정의되었습니다.

## 4.2.2 생성자 & 소멸자 문서화

### 4.2.2.1 DrumRobot()

```
DrumRobot::DrumRobot (
    SystemState & systemStateRef,
    CanManager & canManagerRef,
    PathManager & pathManagerRef,
    HomeManager & homeManagerRef,
    TestManager & testManagerRef,
    std::map< std::string, std::shared_ptr< GenericMotor > > & motorsRef )
```

[DrumRobot](#) 클래스의 생성자.

매개변수

systemStateRef	시스템 상태에 대한 참조.
canManagerRef	CAN 매니저에 대한 참조.
pathManagerRef	경로 매니저에 대한 참조.
homeManagerRef	홈 매니저에 대한 참조.
testManagerRef	테스트 매니저에 대한 참조.
motorsRef	모터 객체들의 맵 참조.

[DrumRobot.cpp](#) 파일의 4 번째 라인에서 정의되었습니다.

```
00010 : systemState(systemStateRef),
00011 : canManager(canManagerRef),
00012 : pathManager(pathManagerRef),
00013 : homeManager(homeManagerRef),
00014 : testManager(testManagerRef),
00015 : motors(motorsRef)
00016 {
00017 }
```

## 4.2.3 멤버 함수 문서화

### 4.2.3.1 checkUserInput()

```
void DrumRobot::checkUserInput ( ) [private]
```

사용자 입력을 확인하는 메소드.

[DrumRobot.cpp](#) 파일의 255 번째 라인에서 정의되었습니다.

```
00256 {
00257     if (kbhit())
00258     {
00259         char input = getchar();
00260         if (input == 'q')
00261             systemState.main = Main::Pause;
00262         else if (input == 'e')
00263         {
00264             isReady = false;
00265             systemState.main = Main::Ready;
00266             pathManager.line = 0;
00267         }
00268         else if (input == 'r')
00269             systemState.main = Main::Perform;
00270     }
00271     usleep(500000);
00272 }
```

#### 4.2.3.2 clearMotorsSendBuffer()

```
void DrumRobot::clearMotorsSendBuffer ( ) [private]
```

모터 송신 버퍼 클리어 메소드.

[DrumRobot.cpp](#) 파일의 953 번째 라인에서 정의되었습니다.

```
00954 {
00955     for (auto motor_pair : motors)
00956         motor_pair.second->clearSendBuffer();
00957 }
```

#### 4.2.3.3 DeactivateControlTask()

```
void DrumRobot::DeactivateControlTask ( ) [private]
```

제어 태스크 비활성화 메소드.

[DrumRobot.cpp](#) 파일의 454 번째 라인에서 정의되었습니다.

```
00455 {
00456     struct can_frame frame;
00457
00458     canManager.setSocketsTimeout(0, 50000);
00459
00460     for (auto &motorPair : motors)
00461     {
00462         std::string name = motorPair.first;
00463         auto &motor = motorPair.second;
00464
00465         // 타입에 따라 적절한 캐스팅과 초기화 수행
00466         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00467         {
00468             tmotorcmd.getCheck(*tMotor, &frame);
00469             canManager.sendAndRecv(motor, frame);
00470
00471             tmotorcmd.getExit(*tMotor, &frame);
00472             if (canManager.sendAndRecv(motor, frame))
00473                 std::cout << "Exiting for motor [" << name << "]" << std::endl;
00474             else
00475                 std::cerr << "Failed to exit control mode for motor [" << name << "]." << std::endl;
00476         }
00477         else if (std::shared_ptr<MaxonMotor> maxonMotor =
00478             std::dynamic_pointer_cast<MaxonMotor>(motor))
00479         {
00480             maxoncmd.getQuickStop(*maxonMotor, &frame);
00481             canManager.txFrame(motor, frame);
00482
00483             maxoncmd.getSync(&frame);
00484             canManager.txFrame(motor, frame);
00485             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00486             {
00487                 while (!motor->recieveBuffer.empty())
00488                 {
00489                     frame = motor->recieveBuffer.front();
00490                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00491                     {
00492                         std::cout << "Exiting for motor [" << name << "]" << std::endl;
00493                         break;
00494                     }
00495                     motor->recieveBuffer.pop();
00496                 }
00497             }
00498             else
00499                 std::cerr << "Failed to exit for motor [" << name << "]." << std::endl;
00500         }
00501     }
```

## 4.2.3.4 displayAvailableCommands()

```
void DrumRobot::displayAvailableCommands ( ) const [private]
```

사용 가능한 명령어를 표시하는 메소드.

DrumRobot.cpp 파일의 146 번째 라인에서 정의되었습니다.

```
00147 {
00148     std::cout << "Available Commands:\n";
00149
00150     if (systemState.main == Main::Ideal)
00151     {
00152         if (systemState.homeMode == HomeMode::NotHome)
00153         {
00154             std::cout << "- h : Start Homing Mode\n";
00155             std::cout << "- x : Make home state by user\n";
00156         }
00157         else if (systemState.homeMode == HomeMode::HomeDone)
00158         {
00159             std::cout << "- r : Move to Ready Position\n";
00160             std::cout << "- t : Start tuning\n";
00161         }
00162     }
00163     else if (systemState.main == Main::Ready)
00164     {
00165         std::cout << "- p : Start Perform\n";
00166         std::cout << "- t : Start tuning\n";
00167     }
00168     std::cout << "- s : Shut down the system\n";
00169     std::cout << "- c : Check Motors position\n";
00170 }
```

## 4.2.3.5 idealStateRoutine()

```
void DrumRobot::idealStateRoutine ( ) [private]
```

이상 상태 루틴을 실행하는 메소드.

DrumRobot.cpp 파일의 237 번째 라인에서 정의되었습니다.

```
00238 {
00239     int ret = system("clear");
00240     if (ret == -1)
00241         cout << "system clear error" << endl;
00242
00243     displayAvailableCommands();
00244
00245     std::string input;
00246     std::cout << "Enter command: ";
00247     std::getline(std::cin, input);
00248
00249     if (!processInput(input))
00250         std::cout << "Invalid command or not allowed in current state!\n";
00251
00252     usleep(2000);
00253 }
```

## 4.2.3.6 initializecanManager()

```
void DrumRobot::initializecanManager ( ) [private]
```

CAN 매니저 초기화 메소드.

DrumRobot.cpp 파일의 447 번째 라인에서 정의되었습니다.

```
00448 {
00449     canManager.initializeCAN();
00450     canManager.checkCanPortsStatus();
00451     canManager.setMotorsSocket();
00452 }
```

### 4.2.3.7 initializeMotors()

```
void DrumRobot::initializeMotors ( ) [private]
```

모터 초기화 메소드.

DrumRobot.cpp 파일의 304 번째 라인에서 정의되었습니다.

```
00305 {
00306     motors["waist"] = make_shared<TMotor>(0x007, "AK10_9");
00307     motors["R_arm1"] = make_shared<TMotor>(0x001, "AK70_10");
00308     motors["L_arm1"] = make_shared<TMotor>(0x002, "AK70_10");
00309     motors["R_arm2"] = make_shared<TMotor>(0x003, "AK70_10");
00310     motors["R_arm3"] = make_shared<TMotor>(0x004, "AK70_10");
00311     motors["L_arm2"] = make_shared<TMotor>(0x005, "AK70_10");
00312     motors["L_arm3"] = make_shared<TMotor>(0x006, "AK70_10");
00313     motors["L_wrist"] = make_shared<MaxonMotor>(0x009);
00314     motors["R_wrist"] = make_shared<MaxonMotor>(0x008);
00315     motors["maxonForTest"] = make_shared<MaxonMotor>(0x00A);
00316
00317     for (auto &motor_pair : motors)
00318     {
00319         auto &motor = motor_pair.second;
00320
00321         // 타입에 따라 적절한 캐스팅과 초기화 수행
00322         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00323         {
00324             // 각 모터 이름에 따른 멤버 변수 설정
00325             if (motor_pair.first == "waist")
00326             {
00327                 tMotor->cwDir = 1.0f;
00328                 tMotor->rMin = -M_PI / 2.0f; // -90deg
00329                 tMotor->rMax = M_PI / 2.0f; // 90deg
00330                 tMotor->Kp = 400;
00331                 tMotor->Kd = 3.5;
00332                 tMotor->isHomed = true;
00333                 tMotor->interFaceName = "can0";
00334             }
00335             else if (motor_pair.first == "R_arm1")
00336             {
00337                 tMotor->cwDir = -1.0f;
00338                 tMotor->sensorBit = 3;
00339                 tMotor->rMin = -M_PI; // -180deg
00340                 tMotor->rMax = 0.0f; // 0deg
00341                 tMotor->Kp = 200;
00342                 tMotor->Kd = 2.5;
00343                 tMotor->isHomed = false;
00344                 tMotor->interFaceName = "can1";
00345             }
00346             else if (motor_pair.first == "L_arm1")
00347             {
00348                 tMotor->cwDir = 1.0f;
00349                 tMotor->sensorBit = 0;
00350                 tMotor->rMin = 0.0f; // 0deg
00351                 tMotor->rMax = M_PI; // 180deg
00352                 tMotor->Kp = 200;
00353                 tMotor->Kd = 2.5;
00354                 tMotor->isHomed = false;
00355                 tMotor->interFaceName = "can0";
00356             }
00357             else if (motor_pair.first == "R_arm2")
00358             {
00359                 tMotor->cwDir = 1.0f;
00360                 tMotor->sensorBit = 4;
00361                 tMotor->rMin = -M_PI / 4.0f; // -45deg
00362                 tMotor->rMax = M_PI / 2.0f; // 90deg
00363                 tMotor->Kp = 350;
00364                 tMotor->Kd = 3.5;
00365                 tMotor->isHomed = false;
00366                 tMotor->interFaceName = "can1";
00367             }
00368             else if (motor_pair.first == "R_arm3")
00369             {
00370                 tMotor->cwDir = -1.0f;
00371                 tMotor->sensorBit = 5;
00372                 tMotor->rMin = -M_PI * 0.75f; // -135deg
00373                 tMotor->rMax = 0.0f; // 0deg
00374                 tMotor->Kp = 250;
00375                 tMotor->Kd = 3.5;
00376                 tMotor->isHomed = false;
00377                 tMotor->interFaceName = "can1";
00378             }
00379             else if (motor_pair.first == "L_arm2")
00380             {
```



```

00381         tMotor->cwDir = -1.0f;
00382         tMotor->sensorBit = 1;
00383         tMotor->rMin = -M_PI / 2.0f; // -90deg
00384         tMotor->rMax = M_PI / 4.0f; // 45deg
00385         tMotor->Kp = 350;
00386         tMotor->Kd = 3.5;
00387         tMotor->isHomed = false;
00388         tMotor->interFaceName = "can0";
00389     }
00390     else if (motor_pair.first == "L_arm3")
00391     {
00392         tMotor->cwDir = -1.0f;
00393         tMotor->sensorBit = 2;
00394         tMotor->rMin = -M_PI * 0.75f; // -135deg
00395         tMotor->rMax = 0.0f; // 0deg
00396         tMotor->Kp = 250;
00397         tMotor->Kd = 3.5;
00398         tMotor->isHomed = false;
00399         tMotor->interFaceName = "can0";
00400     }
00401 }
00402 else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor))
00403 {
00404     // 각 모터 이름에 따른 멤버 변수 설정
00405     if (motor_pair.first == "L_wrist")
00406     {
00407         maxonMotor->cwDir = -1.0f;
00408         maxonMotor->rMin = -M_PI * 0.75f; // -120deg
00409         maxonMotor->rMax = M_PI / 2.0f; // 90deg
00410         maxonMotor->isHomed = false;
00411         maxonMotor->txPdoIds[0] = 0x209; // Controlword
00412         maxonMotor->txPdoIds[1] = 0x309; // TargetPosition
00413         maxonMotor->txPdoIds[2] = 0x409; // TargetVelocity
00414         maxonMotor->txPdoIds[3] = 0x509; // TargetTorque
00415         maxonMotor->rxPdoIds[0] = 0x189; // Statusword, ActualPosition, ActualTorque
00416         maxonMotor->interFaceName = "can2";
00417     }
00418     else if (motor_pair.first == "R_wrist")
00419     {
00420         maxonMotor->cwDir = 1.0f;
00421         maxonMotor->rMin = 0.0f; // 0deg
00422         maxonMotor->rMax = M_PI; // 180deg
00423         maxonMotor->isHomed = false;
00424         maxonMotor->txPdoIds[0] = 0x208; // Controlword
00425         maxonMotor->txPdoIds[1] = 0x308; // TargetPosition
00426         maxonMotor->txPdoIds[2] = 0x408; // TargetVelocity
00427         maxonMotor->txPdoIds[3] = 0x508; // TargetTorque
00428         maxonMotor->rxPdoIds[0] = 0x188; // Statusword, ActualPosition, ActualTorque
00429         maxonMotor->interFaceName = "can2";
00430     }
00431     else if (motor_pair.first == "maxonForTest")
00432     {
00433         maxonMotor->cwDir = 1.0f;
00434         maxonMotor->rMin = 0.0f; // 0deg
00435         maxonMotor->rMax = M_PI; // 180deg
00436         maxonMotor->isHomed = false;
00437         maxonMotor->txPdoIds[0] = 0x20A; // Controlword
00438         maxonMotor->txPdoIds[1] = 0x30A; // TargetPosition
00439         maxonMotor->txPdoIds[2] = 0x40A; // TargetVelocity
00440         maxonMotor->txPdoIds[3] = 0x50A; // TargetTorque
00441         maxonMotor->rxPdoIds[0] = 0x18A; // Statusword, ActualPosition, ActualTorque
00442     }
00443 }
00444 }
00445 };

```

#### 4.2.3.8 initializePathManager()

```
void DrumRobot::initializePathManager ( ) [private]
```

경로 매니저 초기화 메소드.

DrumRobot.cpp 파일의 946 번째 라인에서 정의되었습니다.

```

00947 {
00948     pathManager.ApplyDir();
00949     pathManager.GetDrumPositoin();
00950     pathManager.GetMusicSheet();
00951 }

```

#### 4.2.3.9 kbhit()

```
int DrumRobot::kbhit ( ) [private]
```

키보드 입력이 있는지 확인하는 메소드.

[DrumRobot.cpp](#) 파일의 274 번째 라인에서 정의되었습니다.

```
00275 {
00276     struct termios oldt, newt;
00277     int ch;
00278     int oldf;
00279
00280     tcgetattr(STDIN_FILENO, &oldt);
00281     newt = oldt;
00282     newt.c_lflag &= ~(ICANON | ECHO);
00283     tcsetattr(STDIN_FILENO, TCSANOW, &newt);
00284     oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
00285     fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);
00286
00287     ch = getchar();
00288
00289     tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
00290     fcntl(STDIN_FILENO, F_SETFL, oldf);
00291
00292     if (ch != EOF)
00293     {
00294         ungetc(ch, stdin);
00295         return 1;
00296     }
00297
00298     return 0;
00299 }
```

#### 4.2.3.10 MaxonDisable()

```
void DrumRobot::MaxonDisable ( ) [private]
```

Maxon 모터 비활성화 메소드.

[DrumRobot.cpp](#) 파일의 724 번째 라인에서 정의되었습니다.

```
00725 {
00726     struct can_frame frame;
00727
00728     canManager.setSocketsTimeout(0, 50000);
00729
00730     for (auto &motorPair : motors)
00731     {
00732         std::string name = motorPair.first;
00733         auto &motor = motorPair.second;
00734
00735         if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00736         {
00737             maxoncmd.getQuickStop(*maxonMotor, &frame);
00738             canManager.txFrame(motor, frame);
00739
00740             maxoncmd.getSync(&frame);
00741             canManager.txFrame(motor, frame);
00742             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00743             {
00744                 while (!motor->recieveBuffer.empty())
00745                 {
00746                     frame = motor->recieveBuffer.front();
00747                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00748                         break;
00749                     motor->recieveBuffer.pop();
00750                 }
00751             }
00752             else
00753                 std::cerr << "Failed to exit for motor [" << name << "]." << std::endl;
00754         }
00755     }
00756 }
```

## 4.2.3.11 MaxonEnable()

```
void DrumRobot::MaxonEnable ( ) [private]
```

Maxon 모터 활성화 메소드.

DrumRobot.cpp 파일의 646 번째 라인에서 정의되었습니다.

```
00647 {
00648     struct can_frame frame;
00649     canManager.setSocketsTimeout(2, 0);
00650
00651     int maxonMotorCount = 0;
00652     for (const auto &motor_pair : motors)
00653     {
00654         // 각 요소가 MaxonMotor 타입인지 확인
00655         if (std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00656             maxonMotorCount++;
00657     }
00658
00659     // 제어 모드 설정
00660     for (const auto &motorPair : motors)
00661     {
00662         std::string name = motorPair.first;
00663         std::shared_ptr<GenericMotor> motor = motorPair.second;
00664         if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00665         {
00666
00667             maxoncmd.getOperational(*maxonMotor, &frame);
00668             canManager.txFrame(motor, frame);
00669
00670             maxoncmd.getEnable(*maxonMotor, &frame);
00671             canManager.txFrame(motor, frame);
00672
00673             maxoncmd.getSync(&frame);
00674             canManager.txFrame(motor, frame);
00675
00676             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00677             {
00678                 while (!motor->recieveBuffer.empty())
00679                 {
00680                     frame = motor->recieveBuffer.front();
00681                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00682                         std::cout << "Maxon Enabled \n";
00683                     motor->recieveBuffer.pop();
00684                 }
00685             }
00686
00687             maxoncmd.getQuickStop(*maxonMotor, &frame);
00688             canManager.txFrame(motor, frame);
00689
00690             maxoncmd.getSync(&frame);
00691             canManager.txFrame(motor, frame);
00692
00693             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00694             {
00695                 while (!motor->recieveBuffer.empty())
00696                 {
00697                     frame = motor->recieveBuffer.front();
00698                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00699                         std::cout << "Maxon Quick Stopped\n";
00700                     motor->recieveBuffer.pop();
00701                 }
00702             }
00703
00704             maxoncmd.getEnable(*maxonMotor, &frame);
00705             canManager.txFrame(motor, frame);
00706
00707             maxoncmd.getSync(&frame);
00708             canManager.txFrame(motor, frame);
00709
00710             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00711             {
00712                 while (!motor->recieveBuffer.empty())
00713                 {
00714                     frame = motor->recieveBuffer.front();
00715                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00716                         std::cout << "Maxon Enabled \n";
00717                     motor->recieveBuffer.pop();
00718                 }
00719             }
00720         }
00721     }
00722 };
```

#### 4.2.3.12 motorSettingCmd()

```
void DrumRobot::motorSettingCmd ( ) [private]
```

모터 설정 명령어 메소드.

DrumRobot.cpp 파일의 556 번째 라인에서 정의되었습니다.

```
00557 {
00558     struct can_frame frame;
00559     canManager.setSocketsTimeout(2, 0);
00560     for (const auto &motorPair : motors)
00561     {
00562         std::string name = motorPair.first;
00563         std::shared_ptr<GenericMotor> motor = motorPair.second;
00564         if (std::shared_ptr<MaxonMotor> maxonMotor =
00565             std::dynamic_pointer_cast<MaxonMotor>(motorPair.second))
00566         {
00567             // CSP Settings
00568             maxoncmd.getCsvMode(*maxonMotor, &frame);
00569             canManager.sendAndRecv(motor, frame);
00570
00571             maxoncmd.getPosOffset(*maxonMotor, &frame);
00572             canManager.sendAndRecv(motor, frame);
00573
00574             maxoncmd.getTorqueOffset(*maxonMotor, &frame);
00575             canManager.sendAndRecv(motor, frame);
00576
00577             // CSV Settings
00578             maxoncmd.getCsvMode(*maxonMotor, &frame);
00579             canManager.sendAndRecv(motor, frame);
00580
00581             maxoncmd.getVelOffset(*maxonMotor, &frame);
00582             canManager.sendAndRecv(motor, frame);
00583
00584             // CST Settings
00585             maxoncmd.getCstMode(*maxonMotor, &frame);
00586             canManager.sendAndRecv(motor, frame);
00587
00588             maxoncmd.getTorqueOffset(*maxonMotor, &frame);
00589             canManager.sendAndRecv(motor, frame);
00590
00591             // HMM Settings
00592             maxoncmd.getHomeMode(*maxonMotor, &frame);
00593             canManager.sendAndRecv(motor, frame);
00594
00595             if (name == "L_wrist")
00596             {
00597                 maxoncmd.getHomingMethodL(*maxonMotor, &frame);
00598                 canManager.sendAndRecv(motor, frame);
00599
00600                 maxoncmd.getHomeoffsetDistance(*maxonMotor, &frame, 0);
00601                 canManager.sendAndRecv(motor, frame);
00602
00603                 maxoncmd.getHomePosition(*maxonMotor, &frame, -90);
00604                 canManager.sendAndRecv(motor, frame);
00605             }
00606             else if (name == "R_wrist")
00607             {
00608                 maxoncmd.getHomingMethodR(*maxonMotor, &frame);
00609                 canManager.sendAndRecv(motor, frame);
00610
00611                 maxoncmd.getHomeoffsetDistance(*maxonMotor, &frame, 0);
00612                 canManager.sendAndRecv(motor, frame);
00613
00614                 maxoncmd.getHomePosition(*maxonMotor, &frame, -90);
00615                 canManager.sendAndRecv(motor, frame);
00616             }
00617             else if (name == "maxonForTest")
00618             {
00619                 maxoncmd.getHomingMethodL(*maxonMotor, &frame);
00620                 canManager.sendAndRecv(motor, frame);
00621
00622                 maxoncmd.getHomeoffsetDistance(*maxonMotor, &frame, 20);
00623                 canManager.sendAndRecv(motor, frame);
00624
00625                 maxoncmd.getHomePosition(*maxonMotor, &frame, 90);
00626                 canManager.sendAndRecv(motor, frame);
00627             }
00628
00629             maxoncmd.getCurrentThreshold(*maxonMotor, &frame);
00630             canManager.sendAndRecv(motor, frame);
00631         }
00632     }
00633 }
```

```

00632         else if (std::shared_ptr<TMotor> tmotor = std::dynamic_pointer_cast<TMotor>(motorPair.second))
00633         {
00634             if (name == "waist")
00635             {
00636                 tmotorcmd.getZero(*tmotor, &frame);
00637                 canManager.sendAndRecv(motor, frame);
00638             }
00639             usleep(5000);
00640             tmotorcmd.getControlMode(*tmotor, &frame);
00641             canManager.sendAndRecv(motor, frame);
00642         }
00643     }
00644 }

```

#### 4.2.3.13 parse\_and\_save\_to\_csv()

```

void DrumRobot::parse_and_save_to_csv (
    const std::string & csv_file_name ) [private]

```

파싱 후 CSV 파일로 저장하는 메소드.

DrumRobot.cpp 파일의 998 번째 라인에서 정의되었습니다.

```

00999 {
01000     // CSV 파일 열기. 파일이 없으면 새로 생성됩니다.
01001     std::ofstream ofs(csv_file_name, std::ios::app);
01002     if (!ofs.is_open())
01003     {
01004         std::cerr << "Failed to open or create the CSV file: " << csv_file_name << std::endl;
01005         return;
01006     }
01007
01008     // 파일이 새로 생성되었으면 CSV 헤더를 추가
01009     ofs.seekp(0, std::ios::end);
01010     if (ofs.tellp() == 0)
01011         ofs << "CAN_ID,p_act,tff_des,tff_act\n";
01012
01013     // 각 모터에 대한 처리
01014     for (const auto &pair : motors)
01015     {
01016         auto &motor = pair.second;
01017         if (!motor->recieveBuffer.empty())
01018         {
01019             can_frame frame = motor->recieveBuffer.front();
01020             motor->recieveBuffer.pop();
01021
01022             int id = motor->nodeId;
01023             float position, speed, torque;
01024
01025             // TMotor 또는 MaxonMotor에 따른 데이터 파싱 및 출력
01026             if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
01027             {
01028                 std::tuple<int, float, float, float> parsedData =
01029                 tmotorcmd.parseRecieveCommand(*tMotor, &frame);
01030                 position = std::get<1>(parsedData);
01031                 speed = std::get<2>(parsedData);
01032                 torque = std::get<3>(parsedData);
01033             }
01034             else if (std::shared_ptr<MaxonMotor> maxonMotor =
01035                 std::dynamic_pointer_cast<MaxonMotor>(motor))
01036             {
01037                 std::tuple<int, float, float> parsedData = maxoncmd.parseRecieveCommand(*maxonMotor,
01038                 &frame);
01039                 position = std::get<1>(parsedData);
01040                 torque = std::get<2>(parsedData);
01041                 speed = 0.0;
01042             }
01043
01044             // 데이터 CSV 파일에 쓰기
01045             ofs << "0x" << std::hex << std::setw(4) << std::setfill('0') << id << ", "
01046                 << std::dec << position << ", " << speed << ", " << torque << "\n";
01047         }
01048     }
01049     ofs.close();
01050     std::cout << "연주 txt_OutData 파일이 생성되었습니다." << csv_file_name << std::endl;
01051 }

```

#### 4.2.3.14 printCurrentPositions()

```
void DrumRobot::printCurrentPositions ( ) [private]
```

현재 모터 위치를 출력하는 메소드.

DrumRobot.cpp 파일의 503 번째 라인에서 정의되었습니다.

```
00504 {
00505     for (auto &motorPair : motors)
00506     {
00507         std::string name = motorPair.first;
00508         auto &motor = motorPair.second;
00509         std::cout << "[" << std::hex << motor->nodeId << std::dec << "]" << " ";
00510         std::cout << name << " : " << motor->currentPos << endl;
00511     }
00512
00513     vector<double> P(6);
00514     P = pathManager.fkfun();
00515
00516     cout << "Right Hand Position : { " << P[0] << " , " << P[1] << " , " << P[2] << " }\n";
00517     cout << "Left Hand Position : { " << P[3] << " , " << P[4] << " , " << P[5] << " }\n";
00518     printf("Print Enter to Go Home\n");
00519     getchar();
00520 }
```

#### 4.2.3.15 processInput()

```
bool DrumRobot::processInput (
    const std::string & input ) [private]
```

사용자 입력을 처리하는 메소드.

DrumRobot.cpp 파일의 172 번째 라인에서 정의되었습니다.

```
00173 {
00174     if (systemState.main == Main::Ideal)
00175     {
00176         if (input == "h" && systemState.homeMode == HomeMode::NotHome)
00177         {
00178             systemState.main = Main::Homing;
00179             return true;
00180         }
00181         else if (input == "t" && systemState.homeMode == HomeMode::HomeDone)
00182         {
00183             systemState.main = Main::Tune;
00184             return true;
00185         }
00186         else if (input == "r" && systemState.homeMode == HomeMode::HomeDone)
00187         {
00188             systemState.main = Main::Ready;
00189             return true;
00190         }
00191         else if (input == "x" && systemState.homeMode == HomeMode::NotHome)
00192         {
00193             systemState.homeMode = HomeMode::HomeDone;
00194             return true;
00195         }
00196         else if (input == "c")
00197         {
00198             systemState.main = Main::Check;
00199             return true;
00200         }
00201         else if (input == "s")
00202         {
00203             if (systemState.homeMode == HomeMode::NotHome)
00204                 systemState.main = Main::Shutdown;
00205             else if (systemState.homeMode == HomeMode::HomeDone)
00206                 systemState.main = Main::Back;
00207             return true;
00208         }
00209     }
00210     else if (systemState.main == Main::Ready)
00211     {
00212         if (input == "p")
00213         {
00214             systemState.main = Main::Perform;
00215             return true;
00216         }
00217     }
00218 }
```

```

00216     }
00217     else if (input == "s")
00218     {
00219         systemState.main = Main::Back;
00220         return true;
00221     }
00222     else if (input == "t")
00223     {
00224         systemState.main = Main::Tune;
00225         return true;
00226     }
00227     else if (input == "c")
00228     {
00229         systemState.main = Main::Check;
00230         return true;
00231     }
00232 }
00233
00234 return false;
00235 }

```

#### 4.2.3.16 RecieveLoop()

```
void DrumRobot::RecieveLoop ( ) [private]
```

수신 루프 메소드.

DrumRobot.cpp 파일의 963 번째 라인에서 정의되었습니다.

```

00964 {
00965     chrono::system_clock::time_point external = std::chrono::system_clock::now();
00966
00967     canManager.setSocketsTimeout(0, 50000);
00968     canManager.clearReadBuffers();
00969
00970     sensor.connect();
00971     if (!sensor.connected)
00972         cout << "Sensor initialization failed. Skipping sensor related logic." << endl;
00973
00974     while (systemState.main == Main::Perform || systemState.main == Main::Pause)
00975     {
00976         if (systemState.main == Main::Pause)
00977             continue;
00978
00979         /*if (sensor.connected && (sensor.ReadVal() & 1) != 0)
00980         {
00981             cout << "Motors at Sensor Location please check!!!\n";
00982             systemState.runMode = RunMode::Pause;
00983         }*/
00984
00985         chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00986         chrono::milliseconds elapsed_time = chrono::duration_cast<chrono::milliseconds>(internal -
external);
00987         if (elapsed_time.count() >= TIME_THRESHOLD_MS)
00988         {
00989             external = std::chrono::system_clock::now();
00990             canManager.readFramesFromAllSockets();
00991             canManager.distributeFramesToMotors();
00992         }
00993     }
00994
00995     parse_and_save_to_csv("../..//READ/DrumData_out.txt");
00996 }

```

#### 4.2.3.17 recvLoopForThread()

```
void DrumRobot::recvLoopForThread ( )
```

수신 루프를 별도의 스레드로 실행하는 메소드.

DrumRobot.cpp 파일의 123 번째 라인에서 정의되었습니다.

```

00124 {
00125
00126     while (systemState.main != Main::Shutdown)
00127     {

```

```

00128         usleep(50000);
00129         if (systemState.main == Main::Ideal)
00130         {
00131             canManager.checkCanPortsStatus();
00132             canManager.checkAllMotors();
00133             sleep(3);
00134         }
00135         else if (systemState.main == Main::Perform)
00136         {
00137             canManager.clearReadBuffers();
00138             RecieveLoop();
00139         }
00140     }
00141 }

```

#### 4.2.3.18 save\_to\_txt\_inputData()

```

void DrumRobot::save_to_txt_inputData (
    const string & csv_file_name ) [private]

```

입력 데이터를 txt 파일로 저장하는 메소드.

DrumRobot.cpp 파일의 856 번째 라인에서 정의되었습니다.

```

00857 {
00858     // CSV 파일 열기
00859     std::ofstream csvFile(csv_file_name);
00860
00861     if (!csvFile.is_open())
00862         std::cerr << "Error opening CSV file." << std::endl;
00863
00864     // 헤더 추가
00865     csvFile << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
00866
00867     // 2차원 벡터의 데이터를 CSV 파일로 쓰기
00868     for (const auto &row : pathManager.p)
00869     {
00870         for (const double cell : row)
00871         {
00872             csvFile << std::fixed << std::setprecision(5) << cell;
00873             if (&cell != &row.back())
00874                 csvFile << ","; // 쉼표로 셀 구분
00875         }
00876         csvFile << "\n"; // 다음 행으로 이동
00877     }
00878
00879     // CSV 파일 닫기
00880     csvFile.close();
00881
00882     std::cout << "연주 DrumData_in 파일이 생성되었습니다." << csv_file_name << std::endl;
00883
00884     std::cout << "SendLoop terminated\n";
00885 }

```

#### 4.2.3.19 SendLoop()

```

void DrumRobot::SendLoop ( ) [private]

```

송신 루프 메소드.

DrumRobot.cpp 파일의 761 번째 라인에서 정의되었습니다.

```

00762 {
00763     struct can_frame frameToProcess;
00764     std::string maxonCanInterface;
00765     std::shared_ptr<GenericMotor> virtualMaxonMotor;
00766
00767     int maxonMotorCount = 0;
00768     for (const auto &motor_pair : motors)
00769     {
00770         // 각 요소가 MaxonMotor 타입인지 확인
00771         if (std::shared_ptr<MaxonMotor> maxonMotor =
00772             std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00773         {
00774             maxonMotorCount++;
00775         }
00776     }
00777 }

```



```

00774         maxonCanInterface = maxonMotor->interFaceName;
00775         virtualMaxonMotor = motor_pair.second;
00776     }
00777 }
00778 chrono::system_clock::time_point external = std::chrono::system_clock::now();
00779
00780 while (systemState.main == Main::Perform || systemState.main == Main::Pause)
00781 {
00782     if (systemState.main == Main::Pause)
00783         continue;
00784
00785     bool isAnyBufferLessThanTen = false;
00786     for (const auto &motor_pair : motors)
00787     {
00788         if (motor_pair.second->sendBuffer.size() < 10)
00789         {
00790             isAnyBufferLessThanTen = true;
00791             break;
00792         }
00793     }
00794     if (isAnyBufferLessThanTen)
00795     {
00796         if (pathManager.line < pathManager.total)
00797         {
00798             std::cout << "line : " << pathManager.line << ", total : " << pathManager.total << "\n";
00799             pathManager.PathLoopTask();
00800             pathManager.line++;
00801         }
00802         else if (pathManager.line == pathManager.total)
00803         {
00804             std::cout << "Perform Done\n";
00805             systemState.main = Main::Ready;
00806             pathManager.line = 0;
00807         }
00808     }
00809
00810     bool allBuffersEmpty = true;
00811     for (const auto &motor_pair : motors)
00812     {
00813         if (!motor_pair.second->sendBuffer.empty())
00814         {
00815             allBuffersEmpty = false;
00816             break;
00817         }
00818     }
00819
00820     // 모든 모터의 sendBuffer가 비었을 때 성능 종료 로직 실행
00821     if (allBuffersEmpty)
00822     {
00823         std::cout << "Performance is Over\n";
00824         systemState.main = Main::Ideal;
00825     }
00826
00827     chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00828     chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
00829 external);
00830
00831     if (elapsed_time.count() >= 5000) // 5ms
00832     {
00833         external = std::chrono::system_clock::now();
00834
00835         for (auto &motor_pair : motors)
00836         {
00837             shared_ptr<GenericMotor> motor = motor_pair.second;
00838             canManager.sendFromBuff(motor);
00839         }
00840
00841         if (maxonMotorCount != 0)
00842         {
00843             maxoncmd.getSync(&frameToProcess);
00844             canManager.txFrame(virtualMaxonMotor, frameToProcess);
00845         }
00846     }
00847 }
00848
00849 // CSV 파일명 설정
00850 std::string csvFileName = "../..//READ/DrumData_in.txt";
00851
00852 // input 파일 저장
00853 save_to_txt_inputData(csvFileName);
00854 }

```

#### 4.2.3.20 sendLoopForThread()

```
void DrumRobot::sendLoopForThread ( )
```

송신 루프를 별도의 스레드로 실행하는 메소드.

[DrumRobot.cpp](#) 파일의 107 번째 라인에서 정의되었습니다.

```
00108 {
00109     initializePathManager();
00110     while (systemState.main != Main::Shutdown)
00111     {
00112         usleep(50000);
00113         if (systemState.main == Main::Perform)
00114         {
00115             if (canManager.checkAllMotors())
00116             {
00117                 SendLoop();
00118             }
00119         }
00120     }
00121 }
```

#### 4.2.3.21 SendReadyLoop()

```
void DrumRobot::SendReadyLoop ( ) [private]
```

준비 상태 송신 루프 메소드.

[DrumRobot.cpp](#) 파일의 887 번째 라인에서 정의되었습니다.

```
00888 {
00889     cout << "Settig...\n";
00890     struct can_frame frameToProcess;
00891     std::string maxonCanInterface;
00892     std::shared_ptr<GenericMotor> virtualMaxonMotor;
00893
00894     int maxonMotorCount = 0;
00895     for (const auto &motor_pair : motors)
00896     {
00897         // 각 요소가 MaxonMotor 타입인지 확인
00898         if (std::shared_ptr<MaxonMotor> maxonMotor =
00899             std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00900         {
00901             maxonMotorCount++;
00902             maxonCanInterface = maxonMotor->interFaceName;
00903             virtualMaxonMotor = motor_pair.second;
00904         }
00905         chrono::system_clock::time_point external = std::chrono::system_clock::now();
00906
00907         bool allBuffersEmpty;
00908         do
00909         {
00910             allBuffersEmpty = true;
00911             for (const auto &motor_pair : motors)
00912             {
00913                 if (!motor_pair.second->sendBuffer.empty())
00914                 {
00915                     allBuffersEmpty = false;
00916                     break;
00917                 }
00918             }
00919             if (!allBuffersEmpty)
00920             {
00921                 chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00922                 chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
00923                     external);
00924
00925                 if (elapsed_time.count() >= 5000) // 5ms
00926                 {
00927                     external = std::chrono::system_clock::now();
00928
00929                     for (auto &motor_pair : motors)
00930                     {
00931                         shared_ptr<GenericMotor> motor = motor_pair.second;
00932                         canManager.sendFromBuff(motor);
00933                     }
00934                 }
00935             }
00936         } while (allBuffersEmpty);
00937     }
00938 }
```

```

00934
00935         if (maxonMotorCount != 0)
00936         {
00937             maxoncmd.getSync(&frameToProcess);
00938             canManager.txFrame(virtualMaxonMotor, frameToProcess);
00939         }
00940     }
00941 }
00942 } while (!allBuffersEmpty);
00943 canManager.clearReadBuffers();
00944 }

```

#### 4.2.3.2 setMaxonMode()

```

void DrumRobot::setMaxonMode (
    std::string targetMode ) [private]

```

Maxon 모드 설정 메소드.

DrumRobot.cpp 파일의 522 번째 라인에서 정의되었습니다.

```

00523 {
00524     struct can_frame frame;
00525     canManager.setSocketsTimeout(0, 10000);
00526     for (const auto &motorPair : motors)
00527     {
00528         std::string name = motorPair.first;
00529         std::shared_ptr<GenericMotor> motor = motorPair.second;
00530         if (std::shared_ptr<MaxonMotor> maxonMotor =
00531             std::dynamic_pointer_cast<MaxonMotor>(motorPair.second))
00532         {
00533             if (targetMode == "CSV")
00534             {
00535                 maxoncmd.getCSVMode(*maxonMotor, &frame);
00536                 canManager.sendAndRecv(motor, frame);
00537             }
00538             else if (targetMode == "CST")
00539             {
00540                 maxoncmd.getCSTMode(*maxonMotor, &frame);
00541                 canManager.sendAndRecv(motor, frame);
00542             }
00543             else if (targetMode == "HMM")
00544             {
00545                 maxoncmd.getHomeMode(*maxonMotor, &frame);
00546                 canManager.sendAndRecv(motor, frame);
00547             }
00548             else if (targetMode == "CSP")
00549             {
00550                 maxoncmd.getCSPMode(*maxonMotor, &frame);
00551                 canManager.sendAndRecv(motor, frame);
00552             }
00553         }
00554     }

```

#### 4.2.3.23 stateMachine()

```

void DrumRobot::stateMachine ( )

```

상태 머신을 실행하는 메소드.

DrumRobot.cpp 파일의 23 번째 라인에서 정의되었습니다.

```

00024 {
00025     while (systemState.main != Main::Shutdown)
00026     {
00027         switch (systemState.main.load())
00028         {
00029             case Main::SystemInit:
00030                 initializeMotors();
00031                 initializecanManager();
00032                 motorSettingCmd();
00033                 std::cout << "System Initialize Complete [ Press Enter ]\n";
00034                 getchar();
00035         }

```

```

00036         systemState.main = Main::Ideal;
00037         break;
00038
00039     case Main::Ideal:
00040         idealStateRoutine();
00041         break;
00042
00043     case Main::Homing:
00044         homeManager.mainLoop();
00045         break;
00046
00047     case Main::Perform:
00048         checkUserInput();
00049         break;
00050
00051     case Main::Check:
00052         canManager.checkAllMotors();
00053         printCurrentPositions();
00054         systemState.main = Main::Ideal;
00055         break;
00056
00057     case Main::Tune:
00058         testManager.mainLoop();
00059         break;
00060
00061     case Main::Shutdown:
00062         break;
00063
00064     case Main::Ready:
00065         if (!isReady)
00066         {
00067             if (canManager.checkAllMotors())
00068             {
00069                 MaxonEnable();
00070                 setMaxonMode("CSP");
00071                 cout << "Get Ready...\n";
00072                 clearMotorsSendBuffer();
00073                 pathManager.GetArr(pathManager.standby);
00074                 SendReadyLoop();
00075                 isReady = true;
00076             }
00077         }
00078         else
00079             idealStateRoutine();
00080         break;
00081
00082     case Main::Back:
00083         if (!isBack)
00084         {
00085             if (canManager.checkAllMotors())
00086             {
00087                 cout << "Get Back...\n";
00088                 clearMotorsSendBuffer();
00089                 pathManager.GetArr(pathManager.backarr);
00090                 SendReadyLoop();
00091                 isBack = true;
00092             }
00093         }
00094         else
00095         {
00096             systemState.main = Main::Shutdown;
00097             DeactivateControlTask();
00098         }
00099         break;
00100     case Main::Pause:
00101         checkUserInput();
00102         break;
00103     }
00104 }
00105 }

```

## 4.2.4 멤버 데이터 문서화

### 4.2.4.1 canManager

`CanManager& DrumRobot::canManager [private]`

CAN 매니저 참조.

[DrumRobot.hpp](#) 파일의 68 번째 라인에서 정의되었습니다.

#### 4.2.4.2 homeManager

```
HomeManager& DrumRobot::homeManager [private]
```

홈 매니저 참조.

[DrumRobot.hpp](#) 파일의 70 번째 라인에서 정의되었습니다.

#### 4.2.4.3 isBack

```
bool DrumRobot::isBack [private]
```

되돌아가기 플래그.

[DrumRobot.hpp](#) 파일의 87 번째 라인에서 정의되었습니다.

#### 4.2.4.4 isReady

```
bool DrumRobot::isReady [private]
```

준비 상태 플래그.

[DrumRobot.hpp](#) 파일의 86 번째 라인에서 정의되었습니다.

#### 4.2.4.5 maxoncmd

```
MaxonCommandParser DrumRobot::maxoncmd [private]
```

Maxon 명령 파서.

[DrumRobot.hpp](#) 파일의 75 번째 라인에서 정의되었습니다.

#### 4.2.4.6 motors

```
std::map<std::string, std::shared_ptr<GenericMotor> >& DrumRobot::motors [private]
```

모터 객체들의 맵 참조.

[DrumRobot.hpp](#) 파일의 72 번째 라인에서 정의되었습니다.

#### 4.2.4.7 NUM\_FRAMES

```
const int DrumRobot::NUM_FRAMES [private]
```

처리할 프레임 수.

[DrumRobot.hpp](#) 파일의 107 번째 라인에서 정의되었습니다.

#### 4.2.4.8 pathManager

```
PathManager& DrumRobot::pathManager [private]
```

경로 매니저 참조.

[DrumRobot.hpp](#) 파일의 69 번째 라인에서 정의되었습니다.

#### 4.2.4.9 sensor

```
Sensor DrumRobot::sensor [private]
```

센서 객체.

[DrumRobot.hpp](#) 파일의 76 번째 라인에서 정의되었습니다.

#### 4.2.4.10 systemState

```
SystemState& DrumRobot::systemState [private]
```

시스템 상태 참조.

[DrumRobot.hpp](#) 파일의 67 번째 라인에서 정의되었습니다.

#### 4.2.4.11 testManager

```
TestManager& DrumRobot::testManager [private]
```

테스트 매니저 참조.

[DrumRobot.hpp](#) 파일의 71 번째 라인에서 정의되었습니다.

#### 4.2.4.12 TIME\_THRESHOLD\_MS

```
const int DrumRobot::TIME_THRESHOLD_MS [private]
```

시간 임계값.

[DrumRobot.hpp](#) 파일의 108 번째 라인에서 정의되었습니다.

#### 4.2.4.13 tmotorcmd

```
TMotorCommandParser DrumRobot::tmotorcmd [private]
```

T 모터 명령 파서.

[DrumRobot.hpp](#) 파일의 74 번째 라인에서 정의되었습니다.

## 4.2.4.14 writeFailCount

```
int DrumRobot::writeFailCount [private]
```

송신 실패 카운트.

[DrumRobot.hpp](#) 파일의 102 번째 라인에서 정의되었습니다.

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- include/tasks/DrumRobot.hpp
- src/DrumRobot.cpp

## 4.3 GenericMotor 클래스 참조

모든 모터 타입의 기본이 되는 범용 모터 클래스입니다.

```
#include <Motor.hpp>
```

## Public 멤버 함수

- GenericMotor (uint32\_t [nodeId](#), const std::string &[interFaceName](#))
- void [clearSendBuffer](#) ()  
    송신 버퍼를 클리어합니다.
- void [clearReceiveBuffer](#) ()  
    수신 버퍼를 클리어합니다.

## Public 속성

- uint32\_t [nodeId](#)  
    모터의 노드 ID.
- std::string [interFaceName](#)  
    모터가 연결된 인터페이스의 이름.
- float [desPos](#)
- float [desVel](#)
- float [desTor](#)  
    목표 위치, 속도, 토크.
- float [currentPos](#)
- float [currentVel](#)
- float [currentTor](#)  
    현재 위치, 속도, 토크.
- float [cwDir](#)  
    시계 방향 회전을 나타내는 방향 값.
- bool [isHomed](#)
- bool [isConected](#)  
    홈 위치에 있는지, 연결되어 있는지의 상태.
- float [rMin](#)
- float [rMax](#)  
    회전 범위의 최소, 최대 값.

- int `socket`  
모터가 연결된 소켓의 식별자.
- int `Kp`  
비례 제어 게인.
- double `Kd`  
미분 제어 게인.
- `std::queue< can_frame >` `sendBuffer`  
송신 버퍼.
- `std::queue< can_frame >` `recieveBuffer`  
수신 버퍼.

### 4.3.1 상세한 설명

모든 모터 타입의 기본이 되는 범용 모터 클래스입니다.

이 클래스는 모터의 기본 속성과 기능을 정의합니다. 모든 특정 모터 타입 클래스는 이 클래스를 상속받습니다.

`Motor.hpp` 파일의 24 번째 라인에서 정의되었습니다.

### 4.3.2 멤버 함수 문서화

#### 4.3.2.1 `clearReceiveBuffer()`

```
void GenericMotor::clearReceiveBuffer ( )
```

수신 버퍼를 클리어합니다.

`Motor.cpp` 파일의 16 번째 라인에서 정의되었습니다.

```
00017 {
00018     while (!recieveBuffer.empty())
00019     {
00020         recieveBuffer.pop();
00021     }
00022 }
```

#### 4.3.2.2 `clearSendBuffer()`

```
void GenericMotor::clearSendBuffer ( )
```

송신 버퍼를 클리어합니다.

`Motor.cpp` 파일의 8 번째 라인에서 정의되었습니다.

```
00009 {
00010     while (!sendBuffer.empty())
00011     {
00012         sendBuffer.pop();
00013     }
00014 }
```



### 4.3.3 멤버 데이터 문서화

#### 4.3.3.1 currentPos

```
float GenericMotor::currentPos
```

[Motor.hpp](#) 파일의 30 번째 라인에서 정의되었습니다.

#### 4.3.3.2 currentTor

```
float GenericMotor::currentTor
```

현재 위치, 속도, 토크.

[Motor.hpp](#) 파일의 30 번째 라인에서 정의되었습니다.

#### 4.3.3.3 currentVel

```
float GenericMotor::currentVel
```

[Motor.hpp](#) 파일의 30 번째 라인에서 정의되었습니다.

#### 4.3.3.4 cwDir

```
float GenericMotor::cwDir
```

시계 방향 회전을 나타내는 방향 값.

[Motor.hpp](#) 파일의 31 번째 라인에서 정의되었습니다.

#### 4.3.3.5 desPos

```
float GenericMotor::desPos
```

[Motor.hpp](#) 파일의 29 번째 라인에서 정의되었습니다.

#### 4.3.3.6 desTor

```
float GenericMotor::desTor
```

목표 위치, 속도, 토크.

[Motor.hpp](#) 파일의 29 번째 라인에서 정의되었습니다.

#### 4.3.3.7 desVel

```
float GenericMotor::desVel
```

[Motor.hpp](#) 파일의 29 번째 라인에서 정의되었습니다.

#### 4.3.3.8 interFaceName

```
std::string GenericMotor::interFaceName
```

모터가 연결된 인터페이스의 이름.

[Motor.hpp](#) 파일의 28 번째 라인에서 정의되었습니다.

#### 4.3.3.9 isConected

```
bool GenericMotor::isConected
```

홈 위치에 있는지, 연결되어 있는지의 상태.

[Motor.hpp](#) 파일의 32 번째 라인에서 정의되었습니다.

#### 4.3.3.10 isHomed

```
bool GenericMotor::isHomed
```

[Motor.hpp](#) 파일의 32 번째 라인에서 정의되었습니다.

#### 4.3.3.11 Kd

```
double GenericMotor::Kd
```

미분 제어 게인.

[Motor.hpp](#) 파일의 36 번째 라인에서 정의되었습니다.

#### 4.3.3.12 Kp

```
int GenericMotor::Kp
```

비례 제어 게인.

[Motor.hpp](#) 파일의 35 번째 라인에서 정의되었습니다.

#### 4.3.3.13 nodeId

```
uint32_t GenericMotor::nodeId
```

모터의 노드 ID.

[Motor.hpp](#) 파일의 27 번째 라인에서 정의되었습니다.

#### 4.3.3.14 recieveBuffer

```
std::queue<can_frame> GenericMotor::recieveBuffer
```

수신 버퍼.

[Motor.hpp](#) 파일의 38 번째 라인에서 정의되었습니다.

#### 4.3.3.15 rMax

```
float GenericMotor::rMax
```

회전 범위의 최소, 최대 값.

[Motor.hpp](#) 파일의 33 번째 라인에서 정의되었습니다.

#### 4.3.3.16 rMin

```
float GenericMotor::rMin
```

[Motor.hpp](#) 파일의 33 번째 라인에서 정의되었습니다.

#### 4.3.3.17 sendBuffer

```
std::queue<can_frame> GenericMotor::sendBuffer
```

송신 버퍼.

[Motor.hpp](#) 파일의 37 번째 라인에서 정의되었습니다.

#### 4.3.3.18 socket

```
int GenericMotor::socket
```

모터가 연결된 소켓의 식별자.

[Motor.hpp](#) 파일의 34 번째 라인에서 정의되었습니다.

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- include/motors/Motor.hpp
- src/Motor.cpp

## 4.4 HomeManager 클래스 참조

모터의 홈 위치 설정 및 관리를 담당하는 클래스입니다.

```
#include <HomeManager.hpp>
```

### Public 멤버 함수

- `HomeManager` (`SystemState` &systemStateRef, `CanManager` &canManagerRef, `std::map< std::string, std::shared_ptr< GenericMotor > >` &motorsRef)  
`HomeManager` 클래스의 생성자입니다.
- void `mainLoop` ()  
 모터 홈 위치 설정의 메인 루프를 실행합니다.

### Private 멤버 함수

- void `displayHomingStatus` ()  
 모터의 홈 위치 설정 상태를 표시합니다.
- void `UpdateHomingStatus` ()  
 모터의 홈 위치 설정 상태를 업데이트합니다.
- void `SetTmotorHome` (`vector< std::shared_ptr< GenericMotor > >` &motors, `vector< std::string >` &motorNames)  
 T모터를 홈 위치로 설정합니다.
- void `HomeTMotor` (`vector< std::shared_ptr< GenericMotor > >` &motors, `vector< std::string >` &motorNames)  
 T모터의 홈 위치를 설정하는 작업을 수행합니다.
- `vector< float >` `MoveTMotorToSensorLocation` (`vector< std::shared_ptr< GenericMotor > >` &motors, `vector< std::string >` &motorNames, `vector< int >` &sensorsBit)  
 T모터를 센서 위치로 이동시킵니다.
- void `RotateTMotor` (`vector< std::shared_ptr< GenericMotor > >` &motors, `vector< std::string >` &motorNames, `vector< double >` &directions, `vector< double >` &degrees, `vector< float >` &midpoints)  
 T모터를 회전시킵니다.
- bool `PromptUserForHoming` (const `std::string` &motorName)  
 사용자에게 홈 위치 설정을 위한 확인을 요청합니다.
- void `SetMaxonHome` (`vector< std::shared_ptr< GenericMotor > >` &motors)  
 Maxon 모터를 홈 위치로 설정합니다.
- void `setMaxonMode` (`std::string` targetMode)  
 Maxon 모터의 작동 모드를 설정합니다.
- void `MaxonEnable` ()  
 Maxon 모터를 활성화합니다.
- void `FixMotorPosition` (`std::shared_ptr< GenericMotor >` &motor)  
 모터의 위치를 수정합니다.
- void `MaxonDisable` ()  
 Maxon 모터를 비활성화합니다.

**Private 속성**

- [SystemState](#) & [systemState](#)
- [CanManager](#) & [canManager](#)
- `std::map< std::string, std::shared_ptr< GenericMotor > > & motors`
- [TMotorCommandParser](#) [tmotorecmd](#)
- [MaxonCommandParser](#) [maxoncmd](#)
- Sensor [sensor](#)

**4.4.1 상세한 설명**

모터의 홈 위치 설정 및 관리를 담당하는 클래스입니다.

이 클래스는 모터의 홈 위치를 설정하고, 모터와 관련된 다양한 작업을 관리합니다.

[HomeManager.hpp](#) 파일의 39 번째 라인에서 정의되었습니다.

**4.4.2 생성자 & 소멸자 문서화****4.4.2.1 HomeManager()**

```
HomeManager::HomeManager (
    SystemState & systemStateRef,
    CanManager & canManagerRef,
    std::map< std::string, std::shared_ptr< GenericMotor > > & motorsRef )
```

[HomeManager](#) 클래스의 생성자입니다.

매개변수

<a href="#">systemStateRef</a>	시스템 상태에 대한 참조입니다.
<a href="#">canManagerRef</a>	CAN 통신 관리자에 대한 참조입니다.
<a href="#">motorsRef</a>	모터 객체에 대한 참조를 매핑합니다.

[HomeManager.cpp](#) 파일의 3 번째 라인에서 정의되었습니다.

```
00006 : systemState(systemStateRef), canManager(canManagerRef), motors(motorsRef)
00007 {
00008 }
```

**4.4.3 멤버 함수 문서화****4.4.3.1 displayHomingStatus()**

```
void HomeManager::displayHomingStatus ( ) [private]
```

모터의 홈 위치 설정 상태를 표시합니다.

[HomeManager.cpp](#) 파일의 405 번째 라인에서 정의되었습니다.

```
00406 {
```

```

00407     std::cout << "Homing Status of Motors:\n";
00408     for (const auto &motor_pair : motors)
00409     {
00410         std::cout << motor_pair.first << ": "
00411                 << (motor_pair.second->isHomed ? "Homed" : "Not Homed") << std::endl;
00412     }
00413 }

```

#### 4.4.3.2 FixMotorPosition()

```

void HomeManager::FixMotorPosition (
    std::shared_ptr< GenericMotor > & motor ) [private]

```

모터의 위치를 수정합니다.

매개변수

motor	모터 객체에 대한 참조입니다.
-------	------------------

[HomeManager.cpp](#) 파일의 597 번째 라인에서 정의되었습니다.

```

00598 {
00599     struct can_frame frame;
00600
00601     canManager.checkConnection(motor);
00602
00603     if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00604     {
00605         tMotorCmd.parseSendCommand(*tMotor, &frame, motor->nodeId, 8, motor->currentPos, 0, 450, 1,
00606     );
00607         if (canManager.sendAndRecv(motor, frame))
00608         {
00609             std::cout << "Position fixed for motor [" << motor->nodeId << "]." << std::endl;
00610         }
00611         else
00612         {
00613             std::cerr << "Failed to fix position for motor [" << motor->nodeId << "]." << std::endl;
00614         }
00615     }
00616     else if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00617     {
00618         maxonCmd.getTargetPosition(*maxonMotor, &frame, motor->currentPos);
00619         if (canManager.sendAndRecv(motor, frame))
00620         {
00621             std::cout << "Position fixed for motor [" << motor->nodeId << "]." << std::endl;
00622         }
00623         else
00624         {
00625             std::cerr << "Failed to fix position for motor [" << motor->nodeId << "]." << std::endl;
00626         }
00627     }
}

```

#### 4.4.3.3 HomeTMotor()

```

void HomeManager::HomeTMotor (
    vector< std::shared_ptr< GenericMotor > > & motors,
    vector< std::string > & motorNames ) [private]

```

T모터의 홈 위치를 설정하는 작업을 수행합니다.

매개변수

motors	모터 객체 목록입니다.
motorNames	모터 이름 목록입니다.

HomeManager.cpp 파일의 106 번째 라인에서 정의되었습니다.

```
00107 { // arm2 모터는 -30도, 나머지 모터는 +90도에 센서 위치함.
00108     struct can_frame frameToProcess;
00109     vector<shared_ptr<TMotor>> tMotors;
00110     vector<int> sensorsBit;
00111
00112     // 속도 제어 - 센서 방향으로 이동
00113     for (long unsigned int i = 0; i < motorNames.size(); i++)
00114     {
00115         cout << "« Homing for " << motorNames[i] << " »\n";
00116         tMotors.push_back(dynamic_pointer_cast<TMotor>(motors[i]));
00117
00118         double initialDirection;
00119         if (motorNames[i] == "L_arm2" || motorNames[i] == "R_arm2")
00120             initialDirection = (-0.2) * motors[i]->cwDir;
00121         else
00122             initialDirection = 0.2 * motors[i]->cwDir;
00123
00124         double additionalTorque = 0.0;
00125         if (motorNames[i] == "L_arm2" || motorNames[i] == "R_arm2")
00126             additionalTorque = motors[i]->cwDir * (-3.0);
00127         else if (motorNames[i] == "L_arm3" || motorNames[i] == "R_arm3")
00128             additionalTorque = motors[i]->cwDir * (2.1);
00129
00130         sensorsBit.push_back(tMotors[i]->sensorBit);
00131
00132         tMotorcmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8, 0,
00133             initialDirection, 0, 4.5, additionalTorque);
00134         canManager.sendAndRecv(motors[i], frameToProcess);
00135     }
00136     vector<float> midpoints = MoveTMotorToSensorLocation(motors, motorNames, sensorsBit);
00137
00138     vector<double> directions, degrees;
00139     for (long unsigned int i = 0; i < motorNames.size(); i++)
00140     {
00141         if (motorNames[i] == "L_arm2" || motorNames[i] == "R_arm2")
00142         {
00143             degrees.push_back(-30.0);
00144             midpoints[i] = midpoints[i] * (-1);
00145         }
00146         else
00147         {
00148             degrees.push_back(90.0);
00149         }
00150         directions.push_back(-motors[i]->cwDir);
00151     }
00152
00153     RotateTMotor(motors, motorNames, directions, degrees, midpoints);
00154
00155     cout << "-----moved 90 degree (Anti clock wise) -----
00156     \n";
00157     for (long unsigned int i = 0; i < motors.size(); i++)
00158     {
00159         // 모터를 멈추는 신호를 보냄
00160         tMotorcmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8, 0, 0, 0, 5, 0);
00161         if (canManager.sendAndRecv(motors[i], frameToProcess))
00162             cout << "Set " << motorNames[i] << " speed Zero.\n";
00163
00164         canManager.setSocketsTimeout(2, 0);
00165         // 현재 position을 0으로 인식하는 명령을 보냄
00166         tMotorcmd.getZero(*tMotors[i], &frameToProcess);
00167         if (canManager.sendAndRecv(motors[i], frameToProcess))
00168             cout << "Set Zero.\n";
00169         if (canManager.checkConnection(motors[i]))
00170             cout << motorNames[i] << " Position : " << motors[i]->currentPos;
00171
00172         degrees[i] = 0.0;
00173         directions[i] = motors[i]->cwDir;
00174         midpoints[i] = 0.0;
00175         if (motorNames[i] == "L_arm1" || motorNames[i] == "R_arm1")
00176         {
00177             degrees[i] = 90.0;
00178         }
00179         /*if (motorNames[i] == "L_arm2" || motorNames[i] == "R_arm2"){
00180             degrees[i] = -30.0;
00181         }*/
00182         if (motorNames[i] == "L_arm3" || motorNames[i] == "R_arm3")
00183         {
00184             degrees[i] = 90.0;
00185         }
00186     }
00187
00188     RotateTMotor(motors, motorNames, directions, degrees, midpoints);
00189 }
```

#### 4.4.3.4 mainLoop()

```
void HomeManager::mainLoop ( )
```

모터 홈 위치 설정의 메인 루프를 실행합니다.

HomeManager.cpp 파일의 10 번째 라인에서 정의되었습니다.

```
00011 {
00012     while (systemState.main == Main::Homing)
00013     {
00014         displayHomingStatus();
00015
00016         std::string motorName;
00017         std::cout << "Enter the name of the motor to home, or 'all' to home all motors: ";
00018         std::cin >> motorName;
00019
00020         if (motorName == "all") // 차례행로 동시실행
00021         {
00022             // 우선순위가 높은 순서대로 먼저 홈
00023             vector<vector<string>> Priority = {"L_arm1", "R_arm1"}, {"L_arm2", "R_arm2"}, {"L_arm3",
00024 "R_arm3"};};
00025             for (auto &PmotorNames : Priority)
00026             {
00027                 vector<shared_ptr<GenericMotor>> Pmotors;
00028                 vector<string> Pnames;
00029                 for (const auto &pmotorName : PmotorNames)
00030                 {
00031                     if (motors.find(pmotorName) != motors.end() && !motors[pmotorName]->isHomed)
00032                     {
00033                         Pmotors.push_back(motors[pmotorName]);
00034                         Pnames.push_back(pmotorName);
00035                     }
00036                 }
00037                 if (!Pmotors.empty())
00038                     SetTmotorHome(Pmotors, Pnames);
00039                 Pmotors.clear();
00040                 Pnames.clear();
00041             }
00042             vector<string> PmotorNames = {"L_wrist", "R_wrist", "maxonForTest"};
00043             vector<shared_ptr<GenericMotor>> Pmotors;
00044             for (const auto &pmotorName : PmotorNames)
00045             {
00046                 if (motors.find(pmotorName) != motors.end() && !motors[pmotorName]->isHomed)
00047                     Pmotors.push_back(motors[pmotorName]);
00048             }
00049             if (!Pmotors.empty())
00050                 SetMaxonHome(Pmotors);
00051         }
00052     else if (motors.find(motorName) != motors.end() && !motors[motorName]->isHomed)
00053     { // 원하는 하나의 모터 실행
00054         vector<shared_ptr<GenericMotor>> Pmotor;
00055         vector<string> Pnames;
00056         // 타입에 따라 적절한 캐스팅과 초기화 수행
00057         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motors[motorName]))
00058         {
00059             Pmotor.push_back(motors[motorName]);
00060             Pnames.push_back(motorName);
00061             SetTmotorHome(Pmotor, Pnames);
00062         }
00063     else if (std::shared_ptr<MaxonMotor> maxonMotor =
00064 std::dynamic_pointer_cast<MaxonMotor>(motors[motorName]))
00065     {
00066         Pmotor.push_back(motors[motorName]);
00067         SetMaxonHome(Pmotor);
00068     }
00069     else
00070     {
00071         std::cout << "Motor not found or already homed: " << motorName << std::endl;
00072     }
00073     UpdateHomingStatus();
00074 }
00075 }
```

#### 4.4.3.5 MaxonDisable()

```
void HomeManager::MaxonDisable ( ) [private]
```



Maxon 모터를 비활성화합니다.

HomeManager.cpp 파일의 524 번째 라인에서 정의되었습니다.

```
00525 {
00526     struct can_frame frame;
00527
00528     canManager.setSocketsTimeout(0, 50000);
00529
00530     for (auto &motorPair : motors)
00531     {
00532         std::string name = motorPair.first;
00533         auto &motor = motorPair.second;
00534
00535         if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00536         {
00537             maxoncmd.getQuickStop(*maxonMotor, &frame);
00538             canManager.txFrame(motor, frame);
00539
00540             maxoncmd.getSync(&frame);
00541             canManager.txFrame(motor, frame);
00542             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00543             {
00544                 while (!motor->recieveBuffer.empty())
00545                 {
00546                     frame = motor->recieveBuffer.front();
00547                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00548                     {
00549                         break;
00550                     }
00551                 }
00552                 motor->recieveBuffer.pop();
00553             }
00554         }
00555         else
00556         {
00557             std::cerr << "Failed to exit for motor [" << name << "]." << std::endl;
00558         }
00559     }
00560 }
00561 }
```

#### 4.4.3.6 MaxonEnable()

```
void HomeManager::MaxonEnable ( ) [private]
```

Maxon 모터를 활성화합니다.

HomeManager.cpp 파일의 438 번째 라인에서 정의되었습니다.

```
00439 {
00440     struct can_frame frame;
00441     canManager.setSocketsTimeout(2, 0);
00442
00443     int maxonMotorCount = 0;
00444     for (const auto &motor_pair : motors)
00445     {
00446         // 각 요소가 MaxonMotor 타입인지 확인
00447         if (std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00448         {
00449             maxonMotorCount++;
00450         }
00451     }
00452
00453     // 제어 모드 설정
00454     for (const auto &motorPair : motors)
00455     {
00456         std::string name = motorPair.first;
00457         std::shared_ptr<GenericMotor> motor = motorPair.second;
00458         if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00459         {
00460
00461             maxoncmd.getOperational(*maxonMotor, &frame);
00462             canManager.txFrame(motor, frame);
00463
00464             maxoncmd.getEnable(*maxonMotor, &frame);
00465             canManager.txFrame(motor, frame);
00466
00467             maxoncmd.getSync(&frame);
00468             canManager.txFrame(motor, frame);
00469         }
00470     }
00471 }
```

```

00470         if (canManager.recvToBuff(motor, canManager.maxonCnt))
00471         {
00472             while (!motor->recieveBuffer.empty())
00473             {
00474                 frame = motor->recieveBuffer.front();
00475                 if (frame.can_id == maxonMotor->rxPdoIds[0])
00476                 {
00477                     std::cout << "Maxon Enabled \n";
00478                 }
00479                 motor->recieveBuffer.pop();
00480             }
00481         }
00482
00483         maxoncmd.getQuickStop(*maxonMotor, &frame);
00484         canManager.txFrame(motor, frame);
00485
00486         maxoncmd.getSync(&frame);
00487         canManager.txFrame(motor, frame);
00488
00489         if (canManager.recvToBuff(motor, canManager.maxonCnt))
00490         {
00491             while (!motor->recieveBuffer.empty())
00492             {
00493                 frame = motor->recieveBuffer.front();
00494                 if (frame.can_id == maxonMotor->rxPdoIds[0])
00495                 {
00496                     std::cout << "Maxon Quick Stopped\n";
00497                 }
00498                 motor->recieveBuffer.pop();
00499             }
00500         }
00501
00502         maxoncmd.getEnable(*maxonMotor, &frame);
00503         canManager.txFrame(motor, frame);
00504
00505         maxoncmd.getSync(&frame);
00506         canManager.txFrame(motor, frame);
00507
00508         if (canManager.recvToBuff(motor, canManager.maxonCnt))
00509         {
00510             while (!motor->recieveBuffer.empty())
00511             {
00512                 frame = motor->recieveBuffer.front();
00513                 if (frame.can_id == maxonMotor->rxPdoIds[0])
00514                 {
00515                     std::cout << "Maxon Enabled \n";
00516                 }
00517                 motor->recieveBuffer.pop();
00518             }
00519         }
00520     }
00521 }
00522 };

```

#### 4.4.3.7 MoveTMotorToSensorLocation()

```

vector< float > HomeManager::MoveTMotorToSensorLocation (
    vector< std::shared_ptr< GenericMotor > > & motors,
    vector< std::string > & motorNames,
    vector< int > & sensorsBit ) [private]

```

T모터를 센서 위치로 이동시킵니다.

매개변수

motors	모터 객체 목록입니다.
motorNames	모터 이름 목록입니다.
sensorsBit	센서 비트 값 목록입니다.

반환값

vector<float> 각 모터의 중간 위치를 나타내는 벡터입니다.

HomeManager.cpp 파일의 191 번째 라인에서 정의되었습니다.

```

00192 {
00193     struct can_frame frameToProcess;
00194     vector<shared_ptr<TMotor>> tMotors;
00195     vector<float> firstPosition, secondPosition, positionDifference;
00196     vector<bool> firstSensorTriggered, TriggeredDone;
00197
00198     for (long unsigned int i = 0; i < sensorsBit.size(); i++)
00199     {
00200         tMotors.push_back(dynamic_pointer_cast<TMotor>(motors[i]));
00201         firstPosition.push_back(0.0f);
00202         secondPosition.push_back(0.0f);
00203         firstSensorTriggered.push_back(false);
00204         TriggeredDone.push_back(false);
00205
00206         cout << "Moving " << motorNames[i] << " to sensor location.\n";
00207     }
00208
00209     while (true)
00210     {
00211         // 모든 모터 센싱 완료 시 break
00212         bool done = true;
00213         for (long unsigned int i = 0; i < sensorsBit.size(); i++)
00214         {
00215             if (!TriggeredDone[i])
00216                 done = false;
00217         }
00218         if (done)
00219             break;
00220
00221         for (long unsigned int i = 0; i < sensorsBit.size(); i++)
00222         {
00223             if (!TriggeredDone[i])
00224             {
00225                 bool sensorTriggered = ((sensor.ReadVal() >> sensorsBit[i]) & 1) != 0;
00226
00227                 if (!firstSensorTriggered[i] && sensorTriggered)
00228                 {
00229                     // 첫 번째 센서 인식
00230                     firstSensorTriggered[i] = true;
00231                     canManager.checkConnection(motors[i]);
00232                     firstPosition[i] = motors[i]->currentPos;
00233                     std::cout << motorNames[i] << " first sensor position: " << firstPosition[i] << endl;
00234                 }
00235                 else if (firstSensorTriggered[i] && !sensorTriggered)
00236                 {
00237                     // 센서 인식 해제
00238                     canManager.checkConnection(motors[i]);
00239                     secondPosition[i] = motors[i]->currentPos;
00240                     std::cout << motorNames[i] << " second sensor position: " << secondPosition[i] <<
endl;
00241
00242                     TriggeredDone[i] = true;
00243                 }
00244             }
00245             else
00246             {
00247                 tMotorcmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8,
secondPosition[i], 0, motors[i]->Kp, 2.5, 0);
00248                 canManager.sendAndRecv(motors[i], frameToProcess);
00249             }
00250         }
00251     }
00252
00253     for (long unsigned int i = 0; i < sensorsBit.size(); i++)
00254     {
00255         positionDifference.push_back(abs((secondPosition[i] - firstPosition[i]) / 2.0f));
00256         cout << motorNames[i] << " midpoint position: " << positionDifference[i] << endl;
00257     }
00258
00259     return positionDifference;
00260 }

```

#### 4.4.3.8 PromptUserForHoming()

```

bool HomeManager::PromptUserForHoming (
    const std::string & motorName ) [private]

```

사용자에게 홈 위치 설정을 위한 확인을 요청합니다.

매개변수

motorName	모터의 이름입니다.
-----------	------------

반환값

bool 사용자가 홈 위치 설정을 진행할지 여부를 반환합니다.

[HomeManager.cpp](#) 파일의 77 번째 라인에서 정의되었습니다.

```
00078 {
00079     char userResponse;
00080     std::cout << "Would you like to start homing mode for motor [" << motorName << "]? (y/n): ";
00081     std::cin >> userResponse;
00082     return userResponse == 'y';
00083 }
```

#### 4.4.3.9 RotateTMotor()

```
void HomeManager::RotateTMotor (
    vector< std::shared_ptr< GenericMotor > > & motors,
    vector< std::string > & motorNames,
    vector< double > & directions,
    vector< double > & degrees,
    vector< float > & midpoints ) [private]
```

T모터를 회전시킵니다.

매개변수

motors	모터 객체 목록입니다.
motorNames	모터 이름 목록입니다.
directions	회전 방향 목록입니다.
degrees	회전 각도 목록입니다.
midpoints	모터의 중간 위치 목록입니다.

[HomeManager.cpp](#) 파일의 262 번째 라인에서 정의되었습니다.

```
00263 {
00264
00265     struct can_frame frameToProcess;
00266     vector<shared_ptr<TMotor>> tMotors;
00267     vector<double> targetRadians;
00268     for (long unsigned int i = 0; i < motorNames.size(); i++)
00269     {
00270         if (degrees[i] == 0.0)
00271             return;
00272         tMotors.push_back(dynamic_pointer_cast<TMotor>(motors[i]));
00273         targetRadians.push_back((degrees[i] * M_PI / 180.0 + midpoints[i]) * directions[i]);
00274     }
00275
00276     chrono::system_clock::time_point startTime = std::chrono::system_clock::now();
00277     int totalSteps = 4000 / 5; // 4초 동안 이동 - 5ms 간격으로 나누기
00278     for (int step = 1; step <= totalSteps; ++step)
00279     {
00280         while (1)
00281         {
00282             chrono::system_clock::time_point currentTime = std::chrono::system_clock::now();
00283             if (chrono::duration_cast<chrono::microseconds>(currentTime - startTime).count() > 5000)
00284                 break;
00285         }
00286
00287         startTime = std::chrono::system_clock::now();
00288     }
```

```

00289         // 5ms마다 목표 위치 계산 및 프레임 전송
00290         for (long unsigned int i = 0; i < motorNames.size(); i++)
00291         {
00292             double targetPosition = targetRadians[i] * (static_cast<double>(step) / totalSteps) +
motors[i]->currentPos;
00293             tMotorCmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8,
targetPosition, 0, motors[i]->Kp, motors[i]->Kd, 0);
00294             canManager.sendAndRecv(motors[i], frameToProcess);
00295         }
00296     }
00297
00298     totalSteps = 500 / 5;
00299     for (int step = 1; step <= totalSteps; ++step)
00300     {
00301         while (1)
00302         {
00303             chrono::system_clock::time_point currentTime = std::chrono::system_clock::now();
00304             if (chrono::duration_cast<chrono::microseconds>(currentTime - startTime).count() > 5000)
00305                 break;
00306         }
00307
00308         startTime = std::chrono::system_clock::now();
00309
00310         // 5ms마다 목표 위치 계산 및 프레임 전송
00311         for (long unsigned int i = 0; i < motorNames.size(); i++)
00312         {
00313             double targetPosition = targetRadians[i] + motors[i]->currentPos;
00314             tMotorCmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8,
targetPosition, 0, motors[i]->Kp, motors[i]->Kd, 0);
00315             canManager.sendAndRecv(motors[i], frameToProcess);
00316         }
00317     }
00318
00319     for (auto &motor : motors)
00320     {
00321         canManager.checkConnection(motor);
00322     }
00323 }

```

#### 4.4.3.10 SetMaxonHome()

```

void HomeManager::SetMaxonHome (
    vector< std::shared_ptr< GenericMotor > > & motors ) [private]

```

Maxon 모터를 홈 위치로 설정합니다.

매개변수

motors	모터 객체 목록입니다.
--------	--------------

[HomeManager.cpp](#) 파일의 325 번째 라인에서 정의되었습니다.

```

00326 {
00327
00328     setMaxonMode("HMM");
00329     MaxonEnable();
00330     struct can_frame frame;
00331
00332     canManager.clearReadBuffers();
00333     canManager.setSocketsTimeout(2, 0);
00334     vector<shared_ptr<MaxonMotor> maxonMotors;
00335     for (long unsigned int i = 0; i < motors.size(); i++)
00336     {
00337         maxonMotors.push_back(dynamic_pointer_cast<MaxonMotor>(motors[i]));
00338
00339         // Start to Move by homing method (일단은 PDO)
00340
00341         maxonCmd.getStartHoming(*maxonMotors[i], &frame);
00342         canManager.txFrame(motors[i], frame);
00343         usleep(50000);
00344     }
00345
00346     maxonCmd.getSync(&frame);
00347     canManager.txFrame(motors[0], frame);
00348     if (canManager.recvToBuff(motors[0], canManager.maxonCnt))
00349     {

```

```

00350         while (!motors[0]->recieveBuffer.empty())
00351         {
00352             frame = motors[0]->recieveBuffer.front();
00353             for (long unsigned int i = 0; i < motors.size(); i++)
00354             {
00355                 if (frame.can_id == maxonMotors[i]->rxPdoIds[0])
00356                 {
00357                     cout << "\nMaxon Homing Start!!\n";
00358                 }
00359             }
00360             motors[0]->recieveBuffer.pop();
00361         }
00362     }
00363
00364     sleep(5);
00365     // 홈 위치에 도달할 때까지 반복
00366     bool done = false;
00367     while (!done)
00368     {
00369         done = true;
00370         for (auto &motor : motors)
00371         {
00372             if (!motor->isHomed)
00373                 done = false;
00374         }
00375
00376         maxoncmd.getSync(&frame);
00377         canManager.txFrame(motors[0], frame);
00378         if (canManager.recvToBuff(motors[0], canManager.maxonCnt))
00379         {
00380             while (!motors[0]->recieveBuffer.empty())
00381             {
00382                 frame = motors[0]->recieveBuffer.front();
00383                 for (long unsigned int i = 0; i < motors.size(); i++)
00384                 {
00385                     if (frame.can_id == maxonMotors[i]->rxPdoIds[0])
00386                     {
00387                         if (frame.data[1] & 0x80) // 비트 15 확인
00388                         {
00389                             motors[i]->isHomed = true; // MaxonMotor 객체의 isHomed 속성을 true로 설정
00390                             // 'this'를 사용하여 멤버 함수 호출
00391                         }
00392                     }
00393                 }
00394                 motors[0]->recieveBuffer.pop();
00395             }
00396         }
00397         canManager.clearReadBuffers();
00398
00399         sleep(1); // 100ms 대기
00400     }
00401     setMaxonMode("CSP");
00402     MaxonDisable();
00403 }

```

#### 4.4.3.11 setMaxonMode()

```

void HomeManager::setMaxonMode (
    std::string targetMode ) [private]

```

Maxon 모터의 작동 모드를 설정합니다.

매개변수

targetMode	목표 모드입니다.
------------	-----------

HomeManager.cpp 파일의 563 번째 라인에서 정의되었습니다.

```

00564 {
00565     struct can_frame frame;
00566     canManager.setSocketsTimeout(0, 10000);
00567     for (const auto &motorPair : motors)
00568     {
00569         std::string name = motorPair.first;
00570         std::shared_ptr<GenericMotor> motor = motorPair.second;
00571         if (std::shared_ptr<MaxonMotor> maxonMotor =
            std::dynamic_pointer_cast<MaxonMotor>(motorPair.second))

```

```

00572     {
00573         if (targetMode == "CSV")
00574         {
00575             maxoncmd.getCsvMode(*maxonMotor, &frame);
00576             canManager.sendAndRecv(motor, frame);
00577         }
00578         else if (targetMode == "CST")
00579         {
00580             maxoncmd.getCSTMode(*maxonMotor, &frame);
00581             canManager.sendAndRecv(motor, frame);
00582         }
00583         else if (targetMode == "HMM")
00584         {
00585             maxoncmd.getHomeMode(*maxonMotor, &frame);
00586             canManager.sendAndRecv(motor, frame);
00587         }
00588         else if (targetMode == "CSP")
00589         {
00590             maxoncmd.getCSPMode(*maxonMotor, &frame);
00591             canManager.sendAndRecv(motor, frame);
00592         }
00593     }
00594 }
00595 }

```

#### 4.4.3.12 SetTmotorHome()

```

void HomeManager::SetTmotorHome (
    vector< std::shared_ptr< GenericMotor > > & motors,
    vector< std::string > & motorNames ) [private]

```

T모터를 홈 위치로 설정합니다.

매개변수

motors	모터 객체 목록입니다.
motorNames	모터 이름 목록입니다.

[HomeManager.cpp](#) 파일의 85 번째 라인에서 정의되었습니다.

```

00086 {
00087     sensor.OpenDeviceUntilSuccess();
00088     canManager.setSocketsTimeout(5, 0);
00089
00090     HomeTMotor(motors, motorNames);
00091     for (auto &motor : motors)
00092     {
00093         motor->isHomed = true; // 홈잉 상태 업데이트
00094         sleep(2);
00095         FixMotorPosition(motor);
00096     }
00097
00098     for (auto &motorname : motorNames)
00099     {
00100         cout << "-- Homing completed for " << motorname << " --\n\n";
00101     }
00102
00103     sensor.closeDevice();
00104 }

```

#### 4.4.3.13 UpdateHomingStatus()

```

void HomeManager::UpdateHomingStatus ( ) [private]

```

모터의 홈 위치 설정 상태를 업데이트합니다.

[HomeManager.cpp](#) 파일의 415 번째 라인에서 정의되었습니다.

```

00416 {

```

```

00417     bool allMotorsHomed = true;
00418     for (const auto &motor_pair : motors)
00419     {
00420         if (!motor_pair.second->isHomed)
00421         {
00422             allMotorsHomed = false;
00423             break;
00424         }
00425     }
00426
00427     if (allMotorsHomed)
00428     {
00429         systemState.homeMode = HomeMode::HomeDone;
00430         systemState.main = Main::Ideal;
00431     }
00432     else
00433     {
00434         systemState.homeMode = HomeMode::NotHome;
00435     }
00436 }

```

#### 4.4.4 멤버 데이터 문서화

##### 4.4.4.1 canManager

```
CanManager& HomeManager::canManager [private]
```

[HomeManager.hpp](#) 파일의 59 번째 라인에서 정의되었습니다.

##### 4.4.4.2 maxoncmd

```
MaxonCommandParser HomeManager::maxoncmd [private]
```

[HomeManager.hpp](#) 파일의 63 번째 라인에서 정의되었습니다.

##### 4.4.4.3 motors

```
std::map<std::string, std::shared_ptr<GenericMotor> >& HomeManager::motors [private]
```

[HomeManager.hpp](#) 파일의 60 번째 라인에서 정의되었습니다.

##### 4.4.4.4 sensor

```
Sensor HomeManager::sensor [private]
```

[HomeManager.hpp](#) 파일의 64 번째 라인에서 정의되었습니다.

##### 4.4.4.5 systemState

```
SystemState& HomeManager::systemState [private]
```

[HomeManager.hpp](#) 파일의 58 번째 라인에서 정의되었습니다.



## 4.4.4.6 tmotorcmd

TMotorCommandParser HomeManager::tmotorcmd [private]

HomeManager.hpp 파일의 62 번째 라인에서 정의되었습니다.

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- include/managers/HomeManager.hpp
- src/HomeManager.cpp

## 4.5 MaxonCommandParser 클래스 참조

Maxon 모터 명령어를 파싱하는 클래스입니다.

```
#include <CommandParser.hpp>
```

## Public 멤버 함수

- std::tuple< int, float, float > [parseRecieveCommand](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getCheck](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getStop](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getQuickStop](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getOperational](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getEnable](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getSync](#) (struct can\_frame \*frame)
- void [getCSPMode](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getTorqueOffset](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getPosOffset](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getTargetPosition](#) (MaxonMotor &motor, struct can\_frame \*frame, float p\_des\_radians)
- void [getHomeMode](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getFlowingErrorWindow](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getHomeoffsetDistance](#) (MaxonMotor &motor, struct can\_frame \*frame, int degree)
- void [getHomePosition](#) (MaxonMotor &motor, struct can\_frame \*frame, int degree)
- void [getHomingMethodL](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getHomingMethodR](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getStartHoming](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getCurrentThreshold](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getCSVMode](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getVelOffset](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getTargetVelocity](#) (MaxonMotor &motor, struct can\_frame \*frame, int targetVelocity)
- void [getCSTMode](#) (MaxonMotor &motor, struct can\_frame \*frame)
- void [getTargetTorque](#) (MaxonMotor &motor, struct can\_frame \*frame, int targetTorque)

## 4.5.1 상세한 설명

Maxon 모터 명령어를 파싱하는 클래스입니다.

Maxon 모터에 대한 명령어 송수신 및 해석을 담당합니다. 이 클래스는 Maxon 모터 명령 구성 및 응답 파싱을 위한 메서드를 제공합니다.

CommandParser.hpp 파일의 75 번째 라인에서 정의되었습니다.

## 4.5.2 멤버 함수 문서화

### 4.5.2.1 getCheck()

```
void MaxonCommandParser::getCheck (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 261 번째 라인에서 정의되었습니다.

```
00262 {
00263
00264     frame->can_id = motor.canSendId;
00265     frame->can_dlc = 8;
00266     frame->data[0] = 0x00;
00267     frame->data[1] = 0x00;
00268     frame->data[2] = 0x00;
00269     frame->data[3] = 0x00;
00270     frame->data[4] = 0x00;
00271     frame->data[5] = 0x00;
00272     frame->data[6] = 0x00;
00273     frame->data[7] = 0x00;
00274 }
```

### 4.5.2.2 getCSPMode()

```
void MaxonCommandParser::getCSPMode (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 339 번째 라인에서 정의되었습니다.

```
00340 {
00341     frame->can_id = motor.canSendId;
00342     frame->can_dlc = 8;
00343     frame->data[0] = 0x22;
00344     frame->data[1] = 0x60;
00345     frame->data[2] = 0x60;
00346     frame->data[3] = 0x00;
00347     frame->data[4] = 0x08;
00348     frame->data[5] = 0x00;
00349     frame->data[6] = 0x00;
00350     frame->data[7] = 0x00;
00351 }
```

### 4.5.2.3 getCSTMode()

```
void MaxonCommandParser::getCSTMode (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 585 번째 라인에서 정의되었습니다.

```
00586 {
00587     frame->can_id = motor.canSendId;
00588     frame->can_dlc = 8;
00589     frame->data[0] = 0x22;
00590     frame->data[1] = 0x60;
00591     frame->data[2] = 0x60;
00592     frame->data[3] = 0x00;
00593     frame->data[4] = 0x0A;
00594     frame->data[5] = 0x00;
00595     frame->data[6] = 0x00;
00596     frame->data[7] = 0x00;
00597 }
```

## 4.5.2.4 getCSVMode()

```
void MaxonCommandParser::getCSVMode (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 534 번째 라인에서 정의되었습니다.

```
00535 {
00536     frame->can_id = motor.canSendId;
00537     frame->can_dlc = 8;
00538     frame->data[0] = 0x22;
00539     frame->data[1] = 0x60;
00540     frame->data[2] = 0x60;
00541     frame->data[3] = 0x00;
00542     frame->data[4] = 0x09;
00543     frame->data[5] = 0x00;
00544     frame->data[6] = 0x00;
00545     frame->data[7] = 0x00;
00546 }
```

## 4.5.2.5 getCurrentThreshold()

```
void MaxonCommandParser::getCurrentThreshold (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 517 번째 라인에서 정의되었습니다.

```
00518 {
00519     // 1000 = 3E8
00520     // 500 = 01F4
00521     frame->can_id = motor.canSendId;
00522     frame->can_dlc = 8;
00523     frame->data[0] = 0x23;
00524     frame->data[1] = 0xB2;
00525     frame->data[2] = 0x30;
00526     frame->data[3] = 0x00;
00527     frame->data[4] = 0xF4;
00528     frame->data[5] = 0x01;
00529     frame->data[6] = 0x00;
00530     frame->data[7] = 0x00;
00531 }
```

## 4.5.2.6 getEnable()

```
void MaxonCommandParser::getEnable (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 318 번째 라인에서 정의되었습니다.

```
00319 {
00320     frame->can_id = motor.txPdoIds[0];
00321     frame->can_dlc = 8;
00322     frame->data[0] = 0x0F;
00323     frame->data[1] = 0x00;
00324     frame->data[2] = 0x00;
00325     frame->data[3] = 0x00;
00326     frame->data[4] = 0x00;
00327     frame->data[5] = 0x00;
00328     frame->data[6] = 0x00;
00329     frame->data[7] = 0x00;
00330 }
```

#### 4.5.2.7 getFlowingErrorWindow()

```
void MaxonCommandParser::getFlowingErrorWindow (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 422 번째 라인에서 정의되었습니다.

```
00423 {
00424     frame->can_id = motor.canSendId;
00425     frame->can_dlc = 8;
00426     frame->data[0] = 0x22;
00427     frame->data[1] = 0x65;
00428     frame->data[2] = 0x60;
00429     frame->data[3] = 0x00;
00430     frame->data[4] = 0x00;
00431     frame->data[5] = 0x00;
00432     frame->data[6] = 0x00;
00433     frame->data[7] = 0x00;
00434 }
```

#### 4.5.2.8 getHomeMode()

```
void MaxonCommandParser::getHomeMode (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 408 번째 라인에서 정의되었습니다.

```
00409 {
00410     frame->can_id = motor.canSendId;
00411     frame->can_dlc = 8;
00412     frame->data[0] = 0x22;
00413     frame->data[1] = 0x60;
00414     frame->data[2] = 0x60;
00415     frame->data[3] = 0x00;
00416     frame->data[4] = 0x06;
00417     frame->data[5] = 0x00;
00418     frame->data[6] = 0x00;
00419     frame->data[7] = 0x00;
00420 }
```

#### 4.5.2.9 getHomeoffsetDistance()

```
void MaxonCommandParser::getHomeoffsetDistance (
    MaxonMotor & motor,
    struct can_frame * frame,
    int degree )
```

CommandParser.cpp 파일의 436 번째 라인에서 정의되었습니다.

```
00437 {
00438     // 1도당 값
00439     float value_per_degree = 398.22;
00440
00441     // 입력된 각도에 대한 값을 계산
00442     int value = static_cast<int>(degree * value_per_degree);
00443
00444     frame->can_id = motor.canSendId;
00445     frame->can_dlc = 8;
00446     frame->data[0] = 0x22;
00447     frame->data[1] = 0xB1;
00448     frame->data[2] = 0x30;
00449     frame->data[3] = 0x00;
00450     // 계산된 값의 리틀 엔디언 형식으로 분할하여 할당
00451     frame->data[4] = value & 0xFF; // 하위 바이트
00452     frame->data[5] = (value >> 8) & 0xFF; // 상위 바이트
00453     frame->data[6] = 0x00;
00454     frame->data[7] = 0x00;
00455 }
```

## 4.5.2.10 getHomePosition()

```
void MaxonCommandParser::getHomePosition (
    MaxonMotor & motor,
    struct can_frame * frame,
    int degree )
```

CommandParser.cpp 파일의 457 번째 라인에서 정의되었습니다.

```
00458 {
00459
00460     float value_per_degree = 398.22;
00461     int value = static_cast<int>(degree * value_per_degree);
00462
00463     frame->can_id = motor.canSendId;
00464     frame->can_dlc = 8;
00465     frame->data[0] = 0x22;
00466     frame->data[1] = 0xB0;
00467     frame->data[2] = 0x30;
00468     frame->data[3] = 0x00;
00469     frame->data[4] = value & 0xFF; // 하위 바이트
00470     frame->data[5] = (value >> 8) & 0xFF; // 상위 바이트
00471     frame->data[6] = 0x00;
00472     frame->data[7] = 0x00;
00473 }
```

## 4.5.2.11 getHomingMethodL()

```
void MaxonCommandParser::getHomingMethodL (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 475 번째 라인에서 정의되었습니다.

```
00476 {
00477     frame->can_id = motor.canSendId;
00478     frame->can_dlc = 8;
00479     frame->data[0] = 0x22;
00480     frame->data[1] = 0x98;
00481     frame->data[2] = 0x60;
00482     frame->data[3] = 0x00;
00483     frame->data[4] = 0xFD;
00484     frame->data[5] = 0xFF;
00485     frame->data[6] = 0xFF;
00486     frame->data[7] = 0xFF;
00487 }
```

## 4.5.2.12 getHomingMethodR()

```
void MaxonCommandParser::getHomingMethodR (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 489 번째 라인에서 정의되었습니다.

```
00490 {
00491     frame->can_id = motor.canSendId;
00492     frame->can_dlc = 8;
00493     frame->data[0] = 0x22;
00494     frame->data[1] = 0x98;
00495     frame->data[2] = 0x60;
00496     frame->data[3] = 0x00;
00497     frame->data[4] = 0xFC;
00498     frame->data[5] = 0xFF;
00499     frame->data[6] = 0xFF;
00500     frame->data[7] = 0xFF;
00501 }
```

#### 4.5.2.13 getOperational()

```
void MaxonCommandParser::getOperational (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 304 번째 라인에서 정의되었습니다.

```
00305 {
00306     frame->can_id = 0x00;
00307     frame->can_dlc = 8;
00308     frame->data[0] = 0x01;
00309     frame->data[1] = motor.nodeId;
00310     frame->data[2] = 0x00;
00311     frame->data[3] = 0x00;
00312     frame->data[4] = 0x00;
00313     frame->data[5] = 0x00;
00314     frame->data[6] = 0x00;
00315     frame->data[7] = 0x00;
00316 }
```

#### 4.5.2.14 getPosOffset()

```
void MaxonCommandParser::getPosOffset (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 367 번째 라인에서 정의되었습니다.

```
00368 {
00369     frame->can_id = motor.canSendId;
00370     frame->can_dlc = 8;
00371     frame->data[0] = 0x22;
00372     frame->data[1] = 0xB0;
00373     frame->data[2] = 0x60;
00374     frame->data[3] = 0x00;
00375     frame->data[4] = 0x00;
00376     frame->data[5] = 0x00;
00377     frame->data[6] = 0x00;
00378     frame->data[7] = 0x00;
00379 }
```

#### 4.5.2.15 getQuickStop()

```
void MaxonCommandParser::getQuickStop (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 290 번째 라인에서 정의되었습니다.

```
00291 {
00292     frame->can_id = motor.txPdoIds[0];
00293     frame->can_dlc = 8;
00294     frame->data[0] = 0x06;
00295     frame->data[1] = 0x00;
00296     frame->data[2] = 0x00;
00297     frame->data[3] = 0x00;
00298     frame->data[4] = 0x00;
00299     frame->data[5] = 0x00;
00300     frame->data[6] = 0x00;
00301     frame->data[7] = 0x00;
00302 }
```

#### 4.5.2.16 getStartHoming()

```
void MaxonCommandParser::getStartHoming (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 503 번째 라인에서 정의되었습니다.

```
00504 {
00505     frame->can_id = motor.txPdoIds[0];
00506     frame->can_dlc = 8;
00507     frame->data[0] = 0x1F;
00508     frame->data[1] = 0x00;
00509     frame->data[2] = 0x00;
00510     frame->data[3] = 0x00;
00511     frame->data[4] = 0x00;
00512     frame->data[5] = 0x00;
00513     frame->data[6] = 0x00;
00514     frame->data[7] = 0x00;
00515 }
```

#### 4.5.2.17 getStop()

```
void MaxonCommandParser::getStop (
    MaxonMotor & motor,
    struct can_frame * frame )
```

CommandParser.cpp 파일의 276 번째 라인에서 정의되었습니다.

```
00277 {
00278     frame->can_id = 0x00;
00279     frame->can_dlc = 8;
00280     frame->data[0] = 0x02;
00281     frame->data[1] = motor.nodeId;
00282     frame->data[2] = 0x00;
00283     frame->data[3] = 0x00;
00284     frame->data[4] = 0x00;
00285     frame->data[5] = 0x00;
00286     frame->data[6] = 0x00;
00287     frame->data[7] = 0x00;
00288 }
```

#### 4.5.2.18 getSync()

```
void MaxonCommandParser::getSync (
    struct can_frame * frame )
```

CommandParser.cpp 파일의 332 번째 라인에서 정의되었습니다.

```
00333 {
00334     frame->can_id = 0x80;
00335     frame->can_dlc = 0;
00336 }
```

#### 4.5.2.19 getTargetPosition()

```
void MaxonCommandParser::getTargetPosition (
    MaxonMotor & motor,
    struct can_frame * frame,
    float p_des_radians )
```

pack ints into the can buffer ///

CommandParser.cpp 파일의 381 번째 라인에서 정의되었습니다.

```
00382 {
```

```

00383 // 라디안 값을 인코더 값으로 변환
00384 float p_des_degrees = p_des_radians * (180.0f / M_PI); // 라디안을 도로 변환
00385 int p_des_enc = static_cast<int>(p_des_degrees * (35.0f * 4096.0f) / 360.0f); // 도를 인코더 값으로 변
환
00386
00387 unsigned char posByte0 = p_des_enc & 0xFF; // 하위 8비트
00388 unsigned char posByte1 = (p_des_enc >> 8) & 0xFF; // 다음 8비트
00389 unsigned char posByte2 = (p_des_enc >> 16) & 0xFF; // 다음 8비트
00390 unsigned char posByte3 = (p_des_enc >> 24) & 0xFF; // 최상위 8비트
00391
00392 // Set CAN frame id and data length code
00393 frame->can_id = motor.txPdoIds[1];
00394 frame->can_dlc = 4;
00395
00396 frame->data[0] = posByte0;
00397 frame->data[1] = posByte1;
00398 frame->data[2] = posByte2;
00399 frame->data[3] = posByte3;
00400 frame->data[4] = 0x00;
00401 frame->data[5] = 0x00;
00402 frame->data[6] = 0x00;
00403 frame->data[7] = 0x00;
00404
00405 }

```

#### 4.5.2.20 getTargetTorque()

```

void MaxonCommandParser::getTargetTorque (
    MaxonMotor & motor,
    struct can_frame * frame,
    int targetTorque )

```

pack ints into the can buffer ///

CommandParser.cpp 파일의 599 번째 라인에서 정의되었습니다.

```

00600 {
00601     unsigned char torByte0 = targetTorque & 0xFF; // 하위 8비트
00602     unsigned char torByte1 = (targetTorque >> 8) & 0xFF; // 다음 8비트
00603
00604     // Set CAN frame id and data length code
00605     frame->can_id = motor.txPdoIds[3];
00606     frame->can_dlc = 2;
00607
00608     frame->data[0] = torByte0;
00609     frame->data[1] = torByte1;
00610     frame->data[2] = 0x00;
00611     frame->data[3] = 0x00;
00612     frame->data[4] = 0x00;
00613     frame->data[5] = 0x00;
00614     frame->data[6] = 0x00;
00615     frame->data[7] = 0x00;
00616
00617 }

```

#### 4.5.2.21 getTargetVelocity()

```

void MaxonCommandParser::getTargetVelocity (
    MaxonMotor & motor,
    struct can_frame * frame,
    int targetVelocity )

```

pack ints into the can buffer ///

CommandParser.cpp 파일의 562 번째 라인에서 정의되었습니다.

```

00563 {
00564     unsigned char velByte0 = targetVelocity & 0xFF; // 하위 8비트
00565     unsigned char velByte1 = (targetVelocity >> 8) & 0xFF; // 다음 8비트
00566     unsigned char velByte2 = (targetVelocity >> 16) & 0xFF; // 다음 8비트
00567     unsigned char velByte3 = (targetVelocity >> 24) & 0xFF; // 최상위 8비트
00568
00569     // Set CAN frame id and data length code
00570     frame->can_id = motor.txPdoIds[2];
00571     frame->can_dlc = 4;

```



```

00572
00574     frame->data[0] = velByte0;
00575     frame->data[1] = velByte1;
00576     frame->data[2] = velByte2;
00577     frame->data[3] = velByte3;
00578     frame->data[4] = 0x00;
00579     frame->data[5] = 0x00;
00580     frame->data[6] = 0x00;
00581     frame->data[7] = 0x00;
00582 }

```

#### 4.5.2.22 getTorqueOffset()

```

void MaxonCommandParser::getTorqueOffset (
    MaxonMotor & motor,
    struct can_frame * frame )

```

CommandParser.cpp 파일의 353 번째 라인에서 정의되었습니다.

```

00354 {
00355     frame->can_id = motor.canSendId;
00356     frame->can_dlc = 8;
00357     frame->data[0] = 0x22;
00358     frame->data[1] = 0xB2;
00359     frame->data[2] = 0x60;
00360     frame->data[3] = 0x00;
00361     frame->data[4] = 0x00;
00362     frame->data[5] = 0x00;
00363     frame->data[6] = 0x00;
00364     frame->data[7] = 0x00;
00365 }

```

#### 4.5.2.23 getVelOffset()

```

void MaxonCommandParser::getVelOffset (
    MaxonMotor & motor,
    struct can_frame * frame )

```

CommandParser.cpp 파일의 548 번째 라인에서 정의되었습니다.

```

00549 {
00550     frame->can_id = motor.canSendId;
00551     frame->can_dlc = 8;
00552     frame->data[0] = 0x22;
00553     frame->data[1] = 0xB1;
00554     frame->data[2] = 0x60;
00555     frame->data[3] = 0x00;
00556     frame->data[4] = 0x00;
00557     frame->data[5] = 0x00;
00558     frame->data[6] = 0x00;
00559     frame->data[7] = 0x00;
00560 }

```

#### 4.5.2.24 parseRecieveCommand()

```

std::tuple< int, float, float > MaxonCommandParser::parseRecieveCommand (
    MaxonMotor & motor,
    struct can_frame * frame )

```

CommandParser.cpp 파일의 230 번째 라인에서 정의되었습니다.

```

00231 {
00232     int id = frame->can_id;
00233
00234     int32_t currentPosition = 0;
00235     currentPosition |= static_cast<uint8_t>(frame->data[2]);
00236     currentPosition |= static_cast<uint8_t>(frame->data[3]) << 8;
00237     currentPosition |= static_cast<uint8_t>(frame->data[4]) << 16;
00238     currentPosition |= static_cast<uint8_t>(frame->data[5]) << 24;
00239 }

```

```

00240     int16_t torqueActualValue = 0;
00241     torqueActualValue |= static_cast<uint8_t>(frame->data[6]);
00242     torqueActualValue |= static_cast<uint8_t>(frame->data[7]) << 8;
00243
00244     // Motor rated torque 값을 N·m 단위로 변환 (mNm -> N·m)
00245     const float motorRatedTorquemNm = 31.052; //
00246
00247     // 실제 토크 값을 N·m 단위로 계산
00248     // Torque actual value는 천분의 일 단위이므로, 실제 토크 값은 (torqueActualValue / 1000) * motorRatedTorqueNm
00249     float currentTorqueNm = (static_cast<float>(torqueActualValue) / 1000.0f) * motorRatedTorquemNm;
00250
00251     float currentPositionDegrees = (static_cast<float>(currentPosition) / (35.0f * 4096.0f)) * 360.0f;
00252     float currentPositionRadians = currentPositionDegrees * (M_PI / 180.0f);
00253
00254     motor.currentPos = currentPositionRadians;
00255     motor.currentTor = currentTorqueNm;
00256
00257     return std::make_tuple(id, currentPositionRadians, currentTorqueNm);
00258 }

```

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- include/motors/CommandParser.hpp
- src/CommandParser.cpp

## 4.6 MaxonMotor 클래스 참조

Maxon 모터를 위한 클래스입니다. GenericMotor를 상속받습니다.

```
#include <Motor.hpp>
```

### Public 멤버 함수

- [MaxonMotor](#) (uint32\_t nodeId)
- void [clearSendBuffer](#) ()  
송신 버퍼를 클리어합니다.
- void [clearReceiveBuffer](#) ()  
수신 버퍼를 클리어합니다.

### Public 속성

- uint32\_t [canSendId](#)  
CAN 송신 ID.
- uint32\_t [canReceiveId](#)  
CAN 수신 ID.
- uint32\_t [txPdoIds](#) [4]  
TX PDO 식별자 배열.
- uint32\_t [rxPdoIds](#) [4]  
RX PDO 식별자 배열.
- uint32\_t [nodeId](#)  
모터의 노드 ID.
- std::string [interFaceName](#)  
모터가 연결된 인터페이스의 이름.
- float [desPos](#)
- float [desVel](#)
- float [desTor](#)

- 목표 위치, 속도, 토크.
  - float `currentPos`
  - float `currentVel`
  - float `currentTor`
- 현재 위치, 속도, 토크.
  - float `cwDir`
- 시계 방향 회전을 나타내는 방향 값.
  - bool `isHomed`
  - bool `isConected`
- 홈 위치에 있는지, 연결되어 있는지의 상태.
  - float `rMin`
  - float `rMax`
- 회전 범위의 최소, 최대 값.
  - int `socket`
- 모터가 연결된 소켓의 식별자.
  - int `Kp`
- 비례 제어 게인.
  - double `Kd`
- 미분 제어 게인.
  - `std::queue< can_frame >` `sendBuffer`
- 송신 버퍼.
  - `std::queue< can_frame >` `recieveBuffer`
- 수신 버퍼.

### 4.6.1 상세한 설명

Maxon 모터를 위한 클래스입니다. `GenericMotor`를 상속받습니다.

Maxon 모터 특화된 기능과 속성을 정의합니다.

`Motor.hpp` 파일의 70 번째 라인에서 정의되었습니다.

## 4.6.2 생성자 & 소멸자 문서화

### 4.6.2.1 MaxonMotor()

```
MaxonMotor::MaxonMotor (
    uint32_t nodeId )
```

`Motor.cpp` 파일의 42 번째 라인에서 정의되었습니다.

```
00043 : GenericMotor(nodeId, interFaceName)
00044 {
00045     // canId 값 설정
00046     canSendId = 0x600 + nodeId;
00047     canReceiveId = 0x580 + nodeId;
00048 }
```

### 4.6.3 멤버 함수 문서화

#### 4.6.3.1 clearReceiveBuffer()

```
void GenericMotor::clearReceiveBuffer ( ) [inherited]
```

수신 버퍼를 클리어합니다.

[Motor.cpp](#) 파일의 16 번째 라인에서 정의되었습니다.

```
00017 {
00018     while (!recieveBuffer.empty())
00019     {
00020         recieveBuffer.pop();
00021     }
00022 }
```

#### 4.6.3.2 clearSendBuffer()

```
void GenericMotor::clearSendBuffer ( ) [inherited]
```

송신 버퍼를 클리어합니다.

[Motor.cpp](#) 파일의 8 번째 라인에서 정의되었습니다.

```
00009 {
00010     while (!sendBuffer.empty())
00011     {
00012         sendBuffer.pop();
00013     }
00014 }
```

### 4.6.4 멤버 데이터 문서화

#### 4.6.4.1 canReceiveId

```
uint32_t MaxonMotor::canReceiveId
```

CAN 수신 ID.

[Motor.hpp](#) 파일의 76 번째 라인에서 정의되었습니다.

#### 4.6.4.2 canSendId

```
uint32_t MaxonMotor::canSendId
```

CAN 송신 ID.

[Motor.hpp](#) 파일의 75 번째 라인에서 정의되었습니다.

#### 4.6.4.3 currentPos

```
float GenericMotor::currentPos [inherited]
```

[Motor.hpp](#) 파일의 30 번째 라인에서 정의되었습니다.

#### 4.6.4.4 currentTor

```
float GenericMotor::currentTor [inherited]
```

현재 위치, 속도, 토크.

[Motor.hpp](#) 파일의 30 번째 라인에서 정의되었습니다.

#### 4.6.4.5 currentVel

```
float GenericMotor::currentVel [inherited]
```

[Motor.hpp](#) 파일의 30 번째 라인에서 정의되었습니다.

#### 4.6.4.6 cwDir

```
float GenericMotor::cwDir [inherited]
```

시계 방향 회전을 나타내는 방향 값.

[Motor.hpp](#) 파일의 31 번째 라인에서 정의되었습니다.

#### 4.6.4.7 desPos

```
float GenericMotor::desPos [inherited]
```

[Motor.hpp](#) 파일의 29 번째 라인에서 정의되었습니다.

#### 4.6.4.8 desTor

```
float GenericMotor::desTor [inherited]
```

목표 위치, 속도, 토크.

[Motor.hpp](#) 파일의 29 번째 라인에서 정의되었습니다.

#### 4.6.4.9 desVel

```
float GenericMotor::desVel [inherited]
```

[Motor.hpp](#) 파일의 29 번째 라인에서 정의되었습니다.

#### 4.6.4.10 interFaceName

```
std::string GenericMotor::interFaceName [inherited]
```

모터가 연결된 인터페이스의 이름.

[Motor.hpp](#) 파일의 28 번째 라인에서 정의되었습니다.

#### 4.6.4.11 isConnected

```
bool GenericMotor::isConected [inherited]
```

홈 위치에 있는지, 연결되어 있는지의 상태.

[Motor.hpp](#) 파일의 32 번째 라인에서 정의되었습니다.

#### 4.6.4.12 isHomed

```
bool GenericMotor::isHomed [inherited]
```

[Motor.hpp](#) 파일의 32 번째 라인에서 정의되었습니다.

#### 4.6.4.13 Kd

```
double GenericMotor::Kd [inherited]
```

미분 제어 게인.

[Motor.hpp](#) 파일의 36 번째 라인에서 정의되었습니다.

#### 4.6.4.14 Kp

```
int GenericMotor::Kp [inherited]
```

비례 제어 게인.

[Motor.hpp](#) 파일의 35 번째 라인에서 정의되었습니다.

#### 4.6.4.15 nodeId

```
uint32_t GenericMotor::nodeId [inherited]
```

모터의 노드 ID.

[Motor.hpp](#) 파일의 27 번째 라인에서 정의되었습니다.

#### 4.6.4.16 recieveBuffer

```
std::queue<can_frame> GenericMotor::recieveBuffer [inherited]
```

수신 버퍼.

[Motor.hpp](#) 파일의 38 번째 라인에서 정의되었습니다.

#### 4.6.4.17 rMax

```
float GenericMotor::rMax [inherited]
```

회전 범위의 최소, 최대 값.

[Motor.hpp](#) 파일의 33 번째 라인에서 정의되었습니다.

#### 4.6.4.18 rMin

```
float GenericMotor::rMin [inherited]
```

[Motor.hpp](#) 파일의 33 번째 라인에서 정의되었습니다.

#### 4.6.4.19 rxPdoIds

```
uint32_t MaxonMotor::rxPdoIds[4]
```

RX PDO 식별자 배열.

[Motor.hpp](#) 파일의 79 번째 라인에서 정의되었습니다.

#### 4.6.4.20 sendBuffer

```
std::queue<can_frame> GenericMotor::sendBuffer [inherited]
```

송신 버퍼.

[Motor.hpp](#) 파일의 37 번째 라인에서 정의되었습니다.

#### 4.6.4.21 socket

```
int GenericMotor::socket [inherited]
```

모터가 연결된 소켓의 식별자.

[Motor.hpp](#) 파일의 34 번째 라인에서 정의되었습니다.

#### 4.6.4.22 txPdoIds

```
uint32_t MaxonMotor::txPdoIds[4]
```

TX PDO 식별자 배열.

[Motor.hpp](#) 파일의 78 번째 라인에서 정의되었습니다.

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- include/motors/Motor.hpp
- src/Motor.cpp

## 4.7 PathManager 클래스 참조

### Public 멤버 함수

- `PathManager` (`SystemState` &systemStateRef, `CanManager` &canManagerRef, `std::map< std::string, std::shared_ptr< GenericMotor > >` &motorsRef)
- `vector< double > ffun` ()
- `void ApplyDir` ()
- `void GetDrumPositoin` ()
- `void GetMusicSheet` ()
- `void PathLoopTask` ()
- `void GetArr` (`vector< double >` &arr)
- `void TestArr` (`double t`, `int cycles`, `int type`, `int LnR`)

### Public 속성

- `int total` = 0
- `int line` = 0
- `vector< vector< double > > p`
- `vector< vector< double > > v`
- `vector< double > standby` = {0,  $M\_PI / 2$ ,  $M\_PI / 2$ ,  $M\_PI / 4$ ,  $M\_PI / 2.4$ ,  $M\_PI / 4$ ,  $M\_PI / 2.4$ , 0, 0}
- `vector< double > backarr` = {0,  $M\_PI / 2$ ,  $M\_PI / 2$ , 0, 0, 0, 0,  $M\_PI / 3$ ,  $M\_PI / 3$ }

### Private 멤버 함수

- `vector< double > connect` (`vector< double >` &Q1, `vector< double >` &Q2, `int k`, `int n`)
- `void iconnect` (`vector< double >` &P0, `vector< double >` &P1, `vector< double >` &P2, `vector< double >` &V0, `double t1`, `double t2`, `double t`)
- `vector< double > IKfun` (`vector< double >` &P1, `vector< double >` &P2)
- `void getDrummingPosAndAng` ()
- `void getMotorPos` ()
- `void getQ1AndQ2` ()
- `void getQ3AndQ4` ()
- `void Motors_sendBuffer` ()

### Private 속성

- `TMotorCommandParser` TParser
- `MaxonCommandParser` MParser
- `SystemState` & systemState
- `CanManager` & canManager
- `std::map< std::string, std::shared_ptr< GenericMotor > >` & motors
- `vector< double > c_MotorAngle` = {0, 0, 0, 0, 0, 0, 0, 0, 0}
- `vector< vector< double > > right_inst`
- `vector< vector< double > > left_inst`
- `int n_inst` = 10
- `double bpm` = 10
- `vector< double > time_arr`
- `vector< vector< int > > RA`
- `vector< vector< int > > LA`



- `vector< int > RF`
- `vector< int > LF`
- `double p_R = 0`
- `double p_L = 0`
- `double c_R = 0`
- `double c_L = 0`
- `double r_wrist = 0.0`
- `double l_wrist = 0.0`
- `vector< double > P1 = {0.3, 0.94344, 1.16582}`
- `vector< double > P2 = {-0.3, 0.94344, 1.16582}`
- `vector< double > R = {0.363, 0.793, 0.363, 0.793}`
- `double s = 0.600`
- `double z0 = 1.026`
- `vector< double > Q1`
- `vector< double > Q2`
- `vector< double > Q3`
- `vector< double > Q4`
- `double ElbowAngle_ready = M_PI / 36`
- `double ElbowAngle_hit = M_PI / 18`
- `double WristAngle_ready = M_PI / 4`
- `double WristAngle_hit = M_PI / 2`
- `vector< double > wrist = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0}`
- `map< std::string, int > motor_mapping`
- `map< int, int > motor_dir`

### 4.7.1 상세한 설명

`PathManager.hpp` 파일의 34 번째 라인에서 정의되었습니다.

### 4.7.2 생성자 & 소멸자 문서화

#### 4.7.2.1 PathManager()

```
PathManager::PathManager (
    SystemState & systemStateRef,
    CanManager & canManagerRef,
    std::map< std::string, std::shared_ptr< GenericMotor > > & motorsRef )
```

`PathManager.cpp` 파일의 3 번째 라인에서 정의되었습니다.

```
00006 : systemState(systemStateRef), canManager(canManagerRef), motors(motorsRef)
00007 {
00008 }
```

### 4.7.3 멤버 함수 문서화

#### 4.7.3.1 ApplyDir()

```
void PathManager::ApplyDir ( )
```

`PathManager.cpp` 파일의 47 번째 라인에서 정의되었습니다.

```
00048 { // CW / CCW에 따른 방향 적용
00049     for (auto &entry : motors)
00050     {
00051         shared_ptr<GenericMotor> motor = entry.second;
00052         standby[motor_mapping[entry.first]] *= motor->cwDir;
00053         backarr[motor_mapping[entry.first]] *= motor->cwDir;
00054         motor_dir[motor_mapping[entry.first]] = motor->cwDir;
00055     }
00056 }
```

#### 4.7.3.2 connect()

```
vector< double > PathManager::connect (
    vector< double > & Q1,
    vector< double > & Q2,
    int k,
    int n ) [private]
```

PathManager.cpp 파일의 58 번째 라인에서 정의되었습니다.

```
00059 {
00060     vector<double> Qi;
00061     std::vector<double> A, B;
00062
00063     // Compute A and Bk
00064     for (long unsigned int i = 0; i < Q1.size(); ++i)
00065     {
00066         A.push_back(0.5 * (Q1[i] - Q2[i]));
00067         B.push_back(0.5 * (Q1[i] + Q2[i]));
00068     }
00069
00070     // Compute Qi using the provided formula
00071     for (long unsigned int i = 0; i < Q1.size(); ++i)
00072     {
00073         double val = A[i] * cos(M_PI * k / n) + B[i];
00074         Qi.push_back(val);
00075     }
00076
00077     return Qi;
00078 }
```

#### 4.7.3.3 fkfun()

```
vector< double > PathManager::fkfun ( )
```

PathManager.cpp 파일의 170 번째 라인에서 정의되었습니다.

```
00171 {
00172     vector<double> P;
00173     vector<double> theta(7);
00174     for (auto &motorPair : motors)
00175     {
00176         auto &name = motorPair.first;
00177         auto &motor = motorPair.second;
00178         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00179         {
00180             theta[motor_mapping[name]] = tMotor->currentPos * tMotor->cwDir;
00181         }
00182     }
00183     double r1 = R[0], r2 = R[1], l1 = R[2], l2 = R[3];
00184     double r, l;
00185     r = r1 * sin(theta[3]) + r2 * sin(theta[3] + theta[4]);
00186     l = l1 * sin(theta[5]) + l2 * sin(theta[5] + theta[6]);
00187
00188     P.push_back(0.5 * s * cos(theta[0]) + r * cos(theta[0] + theta[1]));
00189     P.push_back(0.5 * s * sin(theta[0]) + r * sin(theta[0] + theta[1]));
00190     P.push_back(z0 - r1 * cos(theta[3]) - r2 * cos(theta[3] + theta[4]));
00191     P.push_back(0.5 * s * cos(theta[0] + M_PI) + l * cos(theta[0] + theta[2]));
00192     P.push_back(0.5 * s * sin(theta[0] + M_PI) + l * sin(theta[0] + theta[2]));
00193     P.push_back(z0 - l1 * cos(theta[5]) - l2 * cos(theta[5] + theta[6]));
00194
00195     return P;
00196 }
```

#### 4.7.3.4 GetArr()

```
void PathManager::GetArr (
    vector< double > & arr )
```

PathManager.cpp 파일의 648 번째 라인에서 정의되었습니다.

```
00649 {
00650     cout << "Get Array...\n";
```

```

00651     struct can_frame frame;
00652
00653     vector<double> Qi;
00654     vector<vector<double>> q_setting;
00655
00656     getMotorPos();
00657
00658     int n = 800;
00659     for (int k = 0; k < n; ++k)
00660     {
00661         // Make GetBack Array
00662         Qi = connect(c_MotorAngle, arr, k, n);
00663         q_setting.push_back(Qi);
00664
00665         // Send to Buffer
00666         for (auto &entry : motors)
00667         {
00668             if (std::shared_ptr<TMotor> motor = std::dynamic_pointer_cast<TMotor>(entry.second))
00669             {
00670                 float p_des = Qi[motor_mapping[entry.first]];
00671                 TParser.parseSendCommand(*motor, &frame, motor->nodeId, 8, p_des, 0, motor->Kp,
00672                 motor->Kd, 0.0);
00673                 entry.second->sendBuffer.push(frame);
00674             }
00675             else if (std::shared_ptr<MaxonMotor> motor =
00676             std::dynamic_pointer_cast<MaxonMotor>(entry.second))
00677             {
00678                 float p_des = Qi[motor_mapping[entry.first]];
00679                 MParser.getTargetPosition(*motor, &frame, p_des);
00680                 entry.second->sendBuffer.push(frame);
00681             }
00682         }
00683         c_MotorAngle = Qi;
00684     }

```

#### 4.7.3.5 getDrummingPosAndAng()

void PathManager::getDrummingPosAndAng ( ) [private]

PathManager.cpp 파일의 331 번째 라인에서 정의되었습니다.

```

00332 {
00333     for (int j = 0; j < n_inst; ++j)
00334     { // 약기에 맞는 오/원 손목 위치 및 손목 각도
00335         if (RA[line][j] != 0)
00336         {
00337             P1 = right_inst[j];
00338             r_wrist = wrist[j];
00339             c_R = 1;
00340         }
00341         if (LA[line][j] != 0)
00342         {
00343             P2 = left_inst[j];
00344             l_wrist = wrist[j];
00345             c_L = 1;
00346         }
00347     }
00348 }

```

#### 4.7.3.6 GetDrumPositoin()

void PathManager::GetDrumPositoin ( )

PathManager.cpp 파일의 468 번째 라인에서 정의되었습니다.

```

00469 {
00470     getMotorPos();
00471
00472     ifstream inputFile("../include/managers/rT.txt");
00473
00474     if (!inputFile.is_open())
00475     {
00476         cerr << "Failed to open the file."
00477         << "\n";
00478     }
00479 }

```

```

00480 // Read data into a 2D vector
00481 vector<vector<double>> inst_xyz(6, vector<double>(8, 0));
00482
00483 for (int i = 0; i < 6; ++i)
00484 {
00485     for (int j = 0; j < 8; ++j)
00486     {
00487         inputFile » inst_xyz[i][j];
00488         if (i == 1 || i == 4)
00489             inst_xyz[i][j] = inst_xyz[i][j] * 1.0;
00490     }
00491 }
00492
00493 // Extract the desired elements
00494 vector<double> right_B = {0, 0, 0};
00495 vector<double> right_S;
00496 vector<double> right_FT;
00497 vector<double> right_MT;
00498 vector<double> right_HT;
00499 vector<double> right_HH;
00500 vector<double> right_R;
00501 vector<double> right_RC;
00502 vector<double> right_LC;
00503
00504 for (int i = 0; i < 3; ++i)
00505 {
00506     right_S.push_back(inst_xyz[i][0]);
00507     right_FT.push_back(inst_xyz[i][1]);
00508     right_MT.push_back(inst_xyz[i][2]);
00509     right_HT.push_back(inst_xyz[i][3]);
00510     right_HH.push_back(inst_xyz[i][4]);
00511     right_R.push_back(inst_xyz[i][5]);
00512     right_RC.push_back(inst_xyz[i][6]);
00513     right_LC.push_back(inst_xyz[i][7]);
00514 }
00515
00516 vector<double> left_B = {0, 0, 0};
00517 vector<double> left_S;
00518 vector<double> left_FT;
00519 vector<double> left_MT;
00520 vector<double> left_HT;
00521 vector<double> left_HH;
00522 vector<double> left_R;
00523 vector<double> left_RC;
00524 vector<double> left_LC;
00525
00526 for (int i = 3; i < 6; ++i)
00527 {
00528     left_S.push_back(inst_xyz[i][0]);
00529     left_FT.push_back(inst_xyz[i][1]);
00530     left_MT.push_back(inst_xyz[i][2]);
00531     left_HT.push_back(inst_xyz[i][3]);
00532     left_HH.push_back(inst_xyz[i][4]);
00533     left_R.push_back(inst_xyz[i][5]);
00534     left_RC.push_back(inst_xyz[i][6]);
00535     left_LC.push_back(inst_xyz[i][7]);
00536 }
00537
00538 // Combine the elements into right_inst and left_inst
00539 right_inst = {right_B, right_RC, right_R, right_S, right_HH, right_HH, right_FT, right_MT,
right_LC, right_HT};
00540 left_inst = {left_B, left_RC, left_R, left_S, left_HH, left_HH, left_FT, left_MT, left_LC,
left_HT};
00541 }

```

#### 4.7.3.7 getMotorPos()

```
void PathManager::getMotorPos ( ) [private]
```

PathManager.cpp 파일의 80 번째 라인에서 정의되었습니다.

```

00081 {
00082     // 각 모터의 현재위치 값 불러오기 ** CheckMotorPosition 이후에 해야함(변수값을 불러오기만 해서 갱신 필요)
00083     for (auto &entry : motors)
00084     {
00085         c_MotorAngle[motor_mapping[entry.first]] = entry.second->currentPos;
00086         // 각 모터의 현재 위치 출력
00087         cout << "Motor " << entry.first << " current position: " << entry.second->currentPos << "\n";
00088     }
00089 }

```

## 4.7.3.8 GetMusicSheet()

```
void PathManager::GetMusicSheet ( )
```

PathManager.cpp 파일의 543 번째 라인에서 정의되었습니다.

```
00544 {
00545     map<string, int> instrument_mapping = {
00546         {"0", 10}, {"1", 3}, {"2", 6}, {"3", 7}, {"4", 9}, {"5", 4}, {"6", 2}, {"7", 1}, {"8", 8},
00547         {"11", 3}, {"51", 3}, {"61", 3}, {"71", 3}, {"81", 3}, {"91", 3}};
00548
00549     string score_path = "../include/managers/codeConfession.txt";
00550
00551     ifstream file(score_path);
00552     if (!file.is_open())
00553         cerr << "Error opening file." << endl;
00554
00555     string line;
00556     int lineIndex = 0;
00557     while (getline(file, line))
00558     {
00559         istringstream iss(line);
00560         string item;
00561         vector<string> columns;
00562         while (getline(iss, item, '\t'))
00563         {
00564             item = trimWhitespace(item);
00565             columns.push_back(item);
00566         }
00567
00568         if (lineIndex == 0)
00569         { // 첫번째 행엔 bpm에 대한 정보
00570             bpm = stod(columns[0].substr(4));
00571             cout << "bpm = " << bpm << "\n";
00572         }
00573         else
00574         {
00575             vector<int> inst_arr_R(10, 0), inst_arr_L(10, 0);
00576             time_arr.push_back(stod(columns[1]) * 100 / bpm);
00577
00578             if (columns[2] != "0")
00579                 inst_arr_R[instrument_mapping[columns[2]]] = 1;
00580             if (columns[3] != "0")
00581                 inst_arr_L[instrument_mapping[columns[3]]] = 1;
00582
00583             RF.push_back(stoi(columns[6]) == 1 ? 1 : 0);
00584             LF.push_back(stoi(columns[7]) == 2 ? 1 : 0);
00585
00586             RA.push_back(inst_arr_R);
00587             LA.push_back(inst_arr_L);
00588         }
00589         lineIndex++;
00590     }
00591
00592     file.close();
00593
00594     total = RF.size();
00595 }
00596 }
```

## 4.7.3.9 getQ1AndQ2()

```
void PathManager::getQ1AndQ2 ( ) [private]
```

PathManager.cpp 파일의 350 번째 라인에서 정의되었습니다.

```
00351 {
00352     if (c_R == 0 && c_L == 0)
00353     { // 왼손 & 오른손 안침
00354         Q1 = c_MotorAngle;
00355         if (p_R == 1)
00356         {
00357             Q1[4] = Q1[4] + ElbowAngle_ready * motor_dir[4];
00358             Q1[7] = Q1[7] + WristAngle_ready * motor_dir[7];
00359         }
00360         if (p_L == 1)
00361         {
00362             Q1[6] = Q1[6] + ElbowAngle_ready * motor_dir[6];
00363             Q1[8] = Q1[8] + WristAngle_ready * motor_dir[8];
00364         }
00365         Q2 = Q1;
00366     }
```

```

00366     }
00367     else
00368     {
00369         Q1 = IKfun(P1, P2);
00370         Q1.push_back(r_wrist);
00371         Q1.push_back(l_wrist);
00372         Q2 = Q1;
00373         if (c_R != 0 && c_L != 0)
00374         { // 왼손 & 오른손 침
00375             Q1[4] = Q1[4] + ElbowAngle_hit * motor_dir[4];
00376             Q1[6] = Q1[6] + ElbowAngle_hit * motor_dir[6];
00377             Q1[7] = Q1[7] + WristAngle_hit * motor_dir[7];
00378             Q1[8] = Q1[8] + WristAngle_hit * motor_dir[8];
00379         }
00380         else if (c_L != 0)
00381         { // 왼손만 침
00382             Q1[4] = Q1[4] + ElbowAngle_ready * motor_dir[4];
00383             Q2[4] = Q2[4] + ElbowAngle_ready * motor_dir[4];
00384             Q1[6] = Q1[6] + ElbowAngle_hit * motor_dir[6];
00385             Q1[7] = Q1[7] + WristAngle_ready * motor_dir[7];
00386             Q2[7] = Q2[7] + WristAngle_ready * motor_dir[7];
00387             Q1[8] = Q1[8] + WristAngle_hit * motor_dir[8];
00388         }
00389         else if (c_R != 0)
00390         { // 오른손만 침
00391             Q1[4] = Q1[4] + ElbowAngle_hit * motor_dir[4];
00392             Q1[6] = Q1[6] + ElbowAngle_ready * motor_dir[6];
00393             Q2[6] = Q2[6] + ElbowAngle_ready * motor_dir[6];
00394             Q1[7] = Q1[7] + WristAngle_hit * motor_dir[7];
00395             Q1[8] = Q1[8] + WristAngle_ready * motor_dir[8];
00396             Q2[8] = Q2[8] + WristAngle_ready * motor_dir[8];
00397         }
00398         // waist & Arm1 & Arm2는 Q1 ~ Q2 동안 계속 이동
00399         Q1[0] = (Q2[0] + c_MotorAngle[0]) / 2.0;
00400         Q1[1] = (Q2[1] + c_MotorAngle[1]) / 2.0;
00401         Q1[2] = (Q2[2] + c_MotorAngle[2]) / 2.0;
00402         Q1[3] = (Q2[3] + c_MotorAngle[3]) / 2.0;
00403         Q1[5] = (Q2[5] + c_MotorAngle[5]) / 2.0;
00404     }
00405 }

```

#### 4.7.3.10 getQ3AndQ4()

void PathManager::getQ3AndQ4 ( ) [private]

PathManager.cpp 파일의 407 번째 라인에서 정의되었습니다.

```

00408 {
00409     if (c_R == 0 && c_L == 0)
00410     { // 왼손 & 오른손 안침
00411         Q3 = Q2;
00412         if (p_R == 1)
00413         {
00414             Q3[4] = Q3[4] + ElbowAngle_ready * motor_dir[4];
00415             Q3[7] = Q3[7] + WristAngle_ready * motor_dir[7];
00416         }
00417         if (p_L == 1)
00418         {
00419             Q3[6] = Q3[6] + ElbowAngle_ready * motor_dir[6];
00420             Q3[8] = Q3[8] + WristAngle_ready * motor_dir[8];
00421         }
00422         Q4 = Q3;
00423     }
00424     else
00425     {
00426         Q3 = IKfun(P1, P2);
00427         Q3.push_back(r_wrist);
00428         Q3.push_back(l_wrist);
00429         Q4 = Q3;
00430         if (c_R != 0 && c_L != 0)
00431         { // 왼손 & 오른손 침
00432             Q3[4] = Q3[4] + ElbowAngle_hit * motor_dir[4];
00433             Q3[6] = Q3[6] + ElbowAngle_hit * motor_dir[6];
00434             Q3[7] = Q3[7] + WristAngle_hit * motor_dir[7];
00435             Q3[8] = Q3[8] + WristAngle_hit * motor_dir[8];
00436         }
00437         else if (c_L != 0)
00438         { // 왼손만 침
00439             Q3[4] = Q3[4] + ElbowAngle_ready * motor_dir[4];
00440             Q4[4] = Q4[4] + ElbowAngle_ready * motor_dir[4];
00441             Q3[6] = Q3[6] + ElbowAngle_hit * motor_dir[6];
00442             Q3[7] = Q3[7] + WristAngle_ready * motor_dir[7];

```

```

00443         Q4[7] = Q4[7] + WristAngle_ready * motor_dir[7];
00444         Q3[8] = Q3[8] + WristAngle_hit * motor_dir[8];
00445     }
00446     else if (c_R != 0)
00447     { // 오른손만 침
00448         Q3[4] = Q3[4] + ElbowAngle_hit * motor_dir[4];
00449         Q3[6] = Q3[6] + ElbowAngle_ready * motor_dir[6];
00450         Q4[6] = Q4[6] + ElbowAngle_ready * motor_dir[6];
00451         Q3[7] = Q3[7] + WristAngle_hit * motor_dir[7];
00452         Q3[8] = Q3[8] + WristAngle_ready * motor_dir[8];
00453         Q4[8] = Q4[8] + WristAngle_ready * motor_dir[8];
00454     }
00455     // waist & Arm1 & Arm2는 Q3 ~ Q4 동안 계속 이동
00456     Q3[0] = (Q4[0] + Q2[0]) / 2.0;
00457     Q3[1] = (Q4[1] + Q2[1]) / 2.0;
00458     Q3[2] = (Q4[2] + Q2[2]) / 2.0;
00459     Q3[3] = (Q4[3] + Q2[3]) / 2.0;
00460     Q3[5] = (Q4[5] + Q2[5]) / 2.0;
00461 }
00462 }

```

#### 4.7.3.11 iconnect()

```

void PathManager::iconnect (
    vector< double > & P0,
    vector< double > & P1,
    vector< double > & P2,
    vector< double > & V0,
    double t1,
    double t2,
    double t ) [private]

```

PathManager.cpp 파일의 123 번째 라인에서 정의되었습니다.

```

00124 {
00125     vector<double> V1;
00126     vector<double> p_out;
00127     vector<double> v_out;
00128     for (size_t i = 0; i < P0.size(); ++i)
00129     {
00130         if ((P1[i] - P0[i]) / (P2[i] - P1[i]) > 0)
00131             V1.push_back((P2[i] - P0[i]) / t2);
00132         else
00133             V1.push_back(0);
00134
00135         double f = P0[i];
00136         double d = 0;
00137         double e = V0[i];
00138
00139         double M[3][3] = {
00140             {20.0 * pow(t1, 2), 12.0 * t1, 6.0},
00141             {5.0 * pow(t1, 4), 4.0 * pow(t1, 3)},
00142             {3.0 * pow(t1, 2), 3.0 * pow(t1, 2)}},
00143         double ANS[3] = {0, V1[i] - V0[i], P1[i] - P0[i] - V0[i] * t1};
00144
00145         double invM[3][3];
00146         inverseMatrix(M, invM);
00147         // Multiply the inverse of M with ANS
00148         double tem[3];
00149         for (size_t j = 0; j < 3; ++j)
00150         {
00151             tem[j] = 0;
00152             for (size_t k = 0; k < 3; ++k)
00153             {
00154                 tem[j] += invM[j][k] * ANS[k];
00155             }
00156         }
00157
00158         double a = tem[0];
00159         double b = tem[1];
00160         double c = tem[2];
00161
00162         p_out.push_back(a * pow(t, 5) + b * pow(t, 4) + c * pow(t, 3) + d * pow(t, 2) + e * t + f);
00163         v_out.push_back(5 * a * pow(t, 4) + 4 * b * pow(t, 3) + 3 * c * pow(t, 2) + 3 * d * t + e);
00164     }
00165
00166     p.push_back(p_out);
00167     v.push_back(v_out);
00168 }

```

## 4.7.3.12 IKfun()

```
vector< double > PathManager::IKfun (
    vector< double > & P1,
    vector< double > & P2 ) [private]
```

PathManager.cpp 파일의 198 번째 라인에서 정의되었습니다.

```
00199 {
00200     // 드럼위치의 중점 각도
00201     double direction = 0.0 * M_PI; //-M_PI / 3.0;
00202
00203     // 몸통과 팔이 부딪히지 않을 각도 => 36deg
00204     double differ = M_PI / 5.0;
00205
00206     vector<double> Qf(7);
00207
00208     double X1 = P1[0], Y1 = P1[1], z1 = P1[2];
00209     double X2 = P2[0], Y2 = P2[1], z2 = P2[2];
00210     double r1 = R[0], r2 = R[1], r3 = R[2], r4 = R[3];
00211
00212     vector<double> the3(1801);
00213     for (int i = 0; i < 1801; i++)
00214     { // 오른팔 들어올리는 각도 범위 : -90deg ~ 90deg
00215         the3[i] = -M_PI / 2 + (M_PI * i) / 1800;
00216     }
00217
00218     double zeta = z0 - z2;
00219
00220     double det_the0, det_the1, det_the2, det_the4, det_the5, det_the6;
00221     double the0_f, the0, the1, the2, the34, the4, the5, the6;
00222     double r, L, Lp, T;
00223     double sol;
00224     double alpha;
00225     bool first = true;
00226
00227     for (long unsigned int i = 0; i < the3.size(); i++)
00228     {
00229         det_the4 = (z0 - z1 - r1 * cos(the3[i])) / r2;
00230
00231         if (det_the4 < 1 && det_the4 > -1)
00232         {
00233             the34 = acos((z0 - z1 - r1 * cos(the3[i])) / r2);
00234             the4 = the34 - the3[i];
00235             if (the4 > 0 && the4 < M_PI * 0.75)
00236             { // 오른팔꿈치 각도 범위 : 0 ~ 135deg
00237                 r = r1 * sin(the3[i]) + r2 * sin(the34);
00238
00239                 det_the1 = (X1 * X1 + Y1 * Y1 - r * r - s * s / 4) / (s * r);
00240                 if (det_the1 < 1 && det_the1 > -1)
00241                 {
00242                     the1 = acos(det_the1);
00243                     if (the1 > 0 && the1 < (M_PI - differ))
00244                     { // 오른팔 돌리는 각도 범위 : 0 ~ 150deg
00245                         alpha = asin(X1 / sqrt(X1 * X1 + Y1 * Y1));
00246                         det_the0 = (s / 4 + (X1 * X1 + Y1 * Y1 - r * r) / s) / sqrt(X1 * X1 + Y1 *
00247 Y1);
00248
00249                         if (det_the0 < 1 && det_the0 > -1)
00250                         {
00251                             the0 = asin(det_the0) - alpha;
00252
00253                             L = sqrt(pow(X2 - 0.5 * s * cos(the0 + M_PI), 2) +
00254                                     pow(Y2 - 0.5 * s * sin(the0 + M_PI), 2));
00255                             det_the2 = (X2 + 0.5 * s * cos(the0)) / L;
00256
00257                             if (det_the2 < 1 && det_the2 > -1)
00258                             {
00259                                 the2 = acos(det_the2) - the0;
00260                                 if (the2 > differ && the2 < M_PI)
00261                                 { // 왼팔 돌리는 각도 범위 : 30deg ~ 180deg
00262                                     Lp = sqrt(L * L + zeta * zeta);
00263                                     det_the6 = (Lp * Lp - r3 * r3 - r4 * r4) / (2 * r3 * r4);
00264                                     if (det_the6 < 1 && det_the6 > -1)
00265                                     {
00266                                         the6 = acos(det_the6);
00267                                         if (the6 > 0 && the6 < M_PI * 0.75)
00268                                         { // 왼팔꿈치 각도 범위 : 0 ~ 135deg
00269                                             T = (zeta * zeta + L * L + r3 * r3 - r4 * r4) / (r3 * 2);
00270                                             det_the5 = L * L + zeta * zeta - T * T;
00271
00272                                             if (det_the5 > 0)
00273                                             {
00274                                                 sol = T * L - zeta * sqrt(L * L + zeta * zeta - T *
00275 T);
00276                                             }
00277                                         }
00278                                     }
00279                                 }
00280                             }
00281                         }
00282                     }
00283                 }
00284             }
00285         }
00286     }
00287 }
```



```

00273         sol /= (L * L + zeta * zeta);
00274         the5 = asin(sol);
00275         if (the5 > -M_PI / 4 && the5 < M_PI / 2)
00276         { // 왼팔 들어올리는 각도 범위 : -45deg ~ 90deg
00277
00278             if (first || abs(the0 - direction) < abs(the0_f -
direction))
00279             {
00280                 the0_f = the0;
00281                 Qf[0] = the0;
00282                 Qf[1] = the1;
00283                 Qf[2] = the2;
00284                 Qf[3] = the3[i];
00285                 Qf[4] = the4;
00286                 Qf[5] = the5;
00287                 Qf[6] = the6;
00288
00289                 first = false;
00290             }
00291         }
00292     }
00293 }
00294 }
00295 }
00296 }
00297 }
00298 }
00299 }
00300 }
00301 }
00302 }
00303
00304 if(first){
00305     std::cout << "IKfun Not Solved!!\n";
00306     systemState.main = Main::Pause;
00307 }
00308
00309 for (auto &entry : motors)
00310 {
00311     if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(entry.second))
00312     {
00313         Qf[motor_mapping[entry.first]] *= tMotor->cwDir;
00314     }
00315 }
00316
00317 return Qf;
00318 }

```

#### 4.7.3.13 Motors\_sendBuffer()

```
void PathManager::Motors_sendBuffer ( ) [private]
```

PathManager.cpp 파일의 14 번째 라인에서 정의되었습니다.

```

00015 {
00016     struct can_frame frame;
00017
00018     vector<double> Pi;
00019     vector<double> Vi;
00020
00021     Pi = p.back();
00022     Vi = v.back();
00023
00024     for (auto &entry : motors)
00025     {
00026         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(entry.second))
00027         {
00028             float p_des = Pi[motor_mapping[entry.first]];
00029             float v_des = Vi[motor_mapping[entry.first]];
00030
00031             TParser.parseSendCommand(*tMotor, &frame, tMotor->nodeId, 8, p_des, v_des, tMotor->Kp,
tMotor->Kd,
00032             0.0);
00033             entry.second->sendBuffer.push(frame);
00034         }
00035         else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(entry.second))
00036         {
00037             float p_des = Pi[motor_mapping[entry.first]];
00038             MParser.getTargetPosition(*maxonMotor, &frame, p_des);
00039             entry.second->sendBuffer.push(frame);
00040         }
00041     }
}

```

#### 4.7.3.14 PathLoopTask()

```
void PathManager::PathLoopTask ( )
```

[PathManager.cpp](#) 파일의 598 번째 라인에서 정의되었습니다.

```
00599 {
00600     // 연주 처음 시작할 때 Q1, Q2 계산
00601     if (line == 0)
00602     {
00603         c_R = 0;
00604         c_L = 0;
00605
00606         getDrummingPosAndAng();
00607         getQ1AndQ2();
00608
00609         p_R = c_R;
00610         p_L = c_L;
00611
00612         line++;
00613
00614         p.push_back(c_MotorAngle);
00615         v.push_back({0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0});
00616     }
00617
00618     c_R = 0;
00619     c_L = 0;
00620
00621     getDrummingPosAndAng();
00622     getQ3AndQ4();
00623
00624     p_R = c_R;
00625     p_L = c_L;
00626
00627     double t1 = time_arr[line - 1];
00628     double t2 = time_arr[line];
00629     double t = 0.005;
00630     int n = round((t1 / 2) / t);
00631     vector<double> V0 = v.back();
00632     for (int i = 0; i < n; i++)
00633     {
00634         iconnect(c_MotorAngle, Q1, Q2, V0, t1 / 2, t1, t * i);
00635         Motors_sendBuffer();
00636     }
00637     V0 = v.back();
00638     for (int i = 0; i < n; i++)
00639     {
00640         iconnect(Q1, Q2, Q3, V0, t1 / 2, (t1 + t2) / 2, t * i);
00641         Motors_sendBuffer();
00642     }
00643     c_MotorAngle = p.back();
00644     Q1 = Q3;
00645     Q2 = Q4;
00646 }
```

### 4.7.4 멤버 데이터 문서화

#### 4.7.4.1 backarr

```
vector<double> PathManager::backarr = {0, M_PI / 2, M_PI / 2, 0, 0, 0, 0, M_PI / 3, M_PI / 3}
```

[PathManager.hpp](#) 파일의 61 번째 라인에서 정의되었습니다.

```
00061 {0, M_PI / 2, M_PI / 2, 0, 0, 0, 0, M_PI / 3, M_PI / 3};
```

#### 4.7.4.2 bpm

```
double PathManager::bpm = 10 [private]
```

[PathManager.hpp](#) 파일의 77 번째 라인에서 정의되었습니다.

#### 4.7.4.3 c\_L

```
double PathManager::c_L = 0 [private]
```

[PathManager.hpp](#) 파일의 85 번째 라인에서 정의되었습니다.

#### 4.7.4.4 c\_MotorAngle

```
vector<double> PathManager::c_MotorAngle = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} [private]
```

[PathManager.hpp](#) 파일의 72 번째 라인에서 정의되었습니다.

```
00072 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

#### 4.7.4.5 c\_R

```
double PathManager::c_R = 0 [private]
```

[PathManager.hpp](#) 파일의 84 번째 라인에서 정의되었습니다.

#### 4.7.4.6 canManager

```
CanManager& PathManager::canManager [private]
```

[PathManager.hpp](#) 파일의 68 번째 라인에서 정의되었습니다.

#### 4.7.4.7 ElbowAngle\_hit

```
double PathManager::ElbowAngle_hit = M_PI / 18 [private]
```

[PathManager.hpp](#) 파일의 109 번째 라인에서 정의되었습니다.

#### 4.7.4.8 ElbowAngle\_ready

```
double PathManager::ElbowAngle_ready = M_PI / 36 [private]
```

[PathManager.hpp](#) 파일의 108 번째 라인에서 정의되었습니다.

#### 4.7.4.9 l\_wrist

```
double PathManager::l_wrist = 0.0 [private]
```

[PathManager.hpp](#) 파일의 88 번째 라인에서 정의되었습니다.

#### 4.7.4.10 LA

```
vector<vector<int> > PathManager::LA [private]
```

[PathManager.hpp](#) 파일의 79 번째 라인에서 정의되었습니다.

#### 4.7.4.11 left\_inst

```
vector<vector<double> > PathManager::left_inst [private]
```

[PathManager.hpp](#) 파일의 74 번째 라인에서 정의되었습니다.

#### 4.7.4.12 LF

```
vector<int> PathManager::LF [private]
```

[PathManager.hpp](#) 파일의 80 번째 라인에서 정의되었습니다.

#### 4.7.4.13 line

```
int PathManager::line = 0
```

[PathManager.hpp](#) 파일의 52 번째 라인에서 정의되었습니다.

#### 4.7.4.14 motor\_dir

```
map<int, int> PathManager::motor_dir [private]
```

#### 초기값:

```
= {
    {0, 1},
    {1, 1},
    {2, 1},
    {3, 1},
    {4, 1},
    {5, 1},
    {6, 1},
    {7, 1},
    {8, 1}}
```

[PathManager.hpp](#) 파일의 120 번째 라인에서 정의되었습니다.

```
00120                                     { // 1 : CW , -1 : CCW
00121         {0, 1},
00122         {1, 1},
00123         {2, 1},
00124         {3, 1},
00125         {4, 1},
00126         {5, 1},
00127         {6, 1},
00128         {7, 1},
00129         {8, 1}};
```

#### 4.7.4.15 motor\_mapping

```
map<std::string, int> PathManager::motor_mapping [private]
```

##### 초기값:

```
= {
    {"waist", 0}, {"R_arm1", 1}, {"L_arm1", 2}, {"R_arm2", 3}, {"R_arm3", 4}, {"L_arm2", 5}, {"L_arm3",
    6}, {"R_wrist", 7}, {"L_wrist", 8}}
```

[PathManager.hpp](#) 파일의 117 번째 라인에서 정의되었습니다.

```
00117         {"waist", 0}, {"R_arm1", 1}, {"L_arm1", 2}, {"R_arm2", 3}, {"R_arm3", 4}, {"L_arm2", 5},
00118         {"L_arm3", 6}, {"R_wrist", 7}, {"L_wrist", 8}};
```

#### 4.7.4.16 motors

```
std::map<std::string, std::shared_ptr<GenericMotor> > & PathManager::motors [private]
```

[PathManager.hpp](#) 파일의 69 번째 라인에서 정의되었습니다.

#### 4.7.4.17 MParser

```
MaxonCommandParser PathManager::MParser [private]
```

[PathManager.hpp](#) 파일의 65 번째 라인에서 정의되었습니다.

#### 4.7.4.18 n\_inst

```
int PathManager::n_inst = 10 [private]
```

[PathManager.hpp](#) 파일의 76 번째 라인에서 정의되었습니다.

#### 4.7.4.19 p

```
vector<vector<double> > PathManager::p
```

[PathManager.hpp](#) 파일의 55 번째 라인에서 정의되었습니다.

#### 4.7.4.20 P1

```
vector<double> PathManager::P1 = {0.3, 0.94344, 1.16582} [private]
```

[PathManager.hpp](#) 파일의 91 번째 라인에서 정의되었습니다.

```
00091 {0.3, 0.94344, 1.16582}; // RightArm Standby xyz
```

#### 4.7.4.21 P2

```
vector<double> PathManager::P2 = {-0.3, 0.94344, 1.16582} [private]
```

[PathManager.hpp](#) 파일의 92 번째 라인에서 정의되었습니다.

```
00092 {-0.3, 0.94344, 1.16582}; // LeftArm Standby xyz
```

#### 4.7.4.22 p\_L

```
double PathManager::p_L = 0 [private]
```

[PathManager.hpp](#) 파일의 83 번째 라인에서 정의되었습니다.

#### 4.7.4.23 p\_R

```
double PathManager::p_R = 0 [private]
```

[PathManager.hpp](#) 파일의 82 번째 라인에서 정의되었습니다.

#### 4.7.4.24 Q1

```
vector<double> PathManager::Q1 [private]
```

[PathManager.hpp](#) 파일의 106 번째 라인에서 정의되었습니다.

#### 4.7.4.25 Q2

```
vector<double> PathManager::Q2 [private]
```

[PathManager.hpp](#) 파일의 106 번째 라인에서 정의되었습니다.

#### 4.7.4.26 Q3

```
vector<double> PathManager::Q3 [private]
```

[PathManager.hpp](#) 파일의 106 번째 라인에서 정의되었습니다.

#### 4.7.4.27 Q4

```
vector<double> PathManager::Q4 [private]
```

[PathManager.hpp](#) 파일의 106 번째 라인에서 정의되었습니다.

#### 4.7.4.28 R

```
vector<double> PathManager::R = {0.363, 0.793, 0.363, 0.793} [private]
```

[PathManager.hpp](#) 파일의 93 번째 라인에서 정의되었습니다.

```
00093 {0.363, 0.793, 0.363, 0.793}; // 오른팔 상완, 오른팔 하완+스틱, 왼팔 상완, 왼팔 하완+스틱
```

#### 4.7.4.29 r\_wrist

```
double PathManager::r_wrist = 0.0 [private]
```

[PathManager.hpp](#) 파일의 87 번째 라인에서 정의되었습니다.

#### 4.7.4.30 RA

```
vector<vector<int> > PathManager::RA [private]
```

[PathManager.hpp](#) 파일의 79 번째 라인에서 정의되었습니다.

#### 4.7.4.31 RF

```
vector<int> PathManager::RF [private]
```

[PathManager.hpp](#) 파일의 80 번째 라인에서 정의되었습니다.

#### 4.7.4.32 right\_inst

```
vector<vector<double> > PathManager::right_inst [private]
```

[PathManager.hpp](#) 파일의 73 번째 라인에서 정의되었습니다.

#### 4.7.4.33 s

```
double PathManager::s = 0.600 [private]
```

[PathManager.hpp](#) 파일의 94 번째 라인에서 정의되었습니다.

#### 4.7.4.34 standby

```
vector<double> PathManager::standby = {0, M_PI / 2, M_PI / 2, M_PI / 4, M_PI / 2.4, M_PI / 4,  
M_PI / 2.4, 0, 0}
```

[PathManager.hpp](#) 파일의 60 번째 라인에서 정의되었습니다.

```
00060 {0, M_PI / 2, M_PI / 2, M_PI / 4, M_PI / 2.4, M_PI / 4, M_PI / 2.4, 0, 0};
```

#### 4.7.4.35 systemState

```
SystemState& PathManager::systemState [private]
```

[PathManager.hpp](#) 파일의 67 번째 라인에서 정의되었습니다.

**4.7.4.36 time\_arr**

```
vector<double> PathManager::time_arr [private]
```

[PathManager.hpp](#) 파일의 78 번째 라인에서 정의되었습니다.

**4.7.4.37 total**

```
int PathManager::total = 0
```

[PathManager.hpp](#) 파일의 51 번째 라인에서 정의되었습니다.

**4.7.4.38 TParser**

```
TMotorCommandParser PathManager::TParser [private]
```

[PathManager.hpp](#) 파일의 64 번째 라인에서 정의되었습니다.

**4.7.4.39 v**

```
vector<vector<double> > PathManager::v
```

[PathManager.hpp](#) 파일의 56 번째 라인에서 정의되었습니다.

**4.7.4.40 wrist**

```
vector<double> PathManager::wrist = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0} [private]
```

[PathManager.hpp](#) 파일의 114 번째 라인에서 정의되었습니다.  
00114 {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};

**4.7.4.41 WristAngle\_hit**

```
double PathManager::WristAngle_hit = M_PI / 2 [private]
```

[PathManager.hpp](#) 파일의 111 번째 라인에서 정의되었습니다.

**4.7.4.42 WristAngle\_ready**

```
double PathManager::WristAngle_ready = M_PI / 4 [private]
```

[PathManager.hpp](#) 파일의 110 번째 라인에서 정의되었습니다.



## 4.7.4.43 z0

```
double PathManager::z0 = 1.026 [private]
```

[PathManager.hpp](#) 파일의 95 번째 라인에서 정의되었습니다.

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- [include/managers/PathManager.hpp](#)
- [src/PathManager.cpp](#)

## 4.8 SystemState 구조체 참조

시스템의 전반적인 상태를 관리합니다.

```
#include <SystemState.hpp>
```

## Public 멤버 함수

- [SystemState\(\)](#)  
SystemState의 기본 생성자.

## Public 속성

- `std::atomic< Main >` [main](#)  
시스템의 주 상태.
- `std::atomic< HomeMode >` [homeMode](#)  
홈 모드의 상태.

## 4.8.1 상세한 설명

시스템의 전반적인 상태를 관리합니다.

이 구조체는 시스템의 주 상태(Main)와 홈 모드 상태(HomeMode)를 관리합니다. 각 상태는 `std::atomic`을 사용하여 멀티스레딩 환경에서 안전하게 접근됩니다.

[SystemState.hpp](#) 파일의 41 번째 라인에서 정의되었습니다.

## 4.8.2 생성자 &amp; 소멸자 문서화

## 4.8.2.1 SystemState()

```
SystemState::SystemState ( ) [inline]
```

SystemState의 기본 생성자.

시스템을 시작 상태(SystemInit)와 홈 모드를 미완료 상태(NotHome)로 초기화합니다.

[SystemState.hpp](#) 파일의 51 번째 라인에서 정의되었습니다.

```
00051 : main(Main::SystemInit),
00052 : homeMode(HomeMode::NotHome) {}
```

### 4.8.3 멤버 데이터 문서화

#### 4.8.3.1 homeMode

```
std::atomic<HomeMode> SystemState::homeMode
```

홈 모드의 상태.

[SystemState.hpp](#) 파일의 44 번째 라인에서 정의되었습니다.

#### 4.8.3.2 main

```
std::atomic<Main> SystemState::main
```

시스템의 주 상태.

[SystemState.hpp](#) 파일의 43 번째 라인에서 정의되었습니다.

이 구조체에 대한 문서화 페이지는 다음의 파일로부터 생성되었습니다.:

- `include/tasks/SystemState.hpp`

## 4.9 TestManager 클래스 참조

모터의 성능 테스트 및 파라미터 튜닝을 위한 클래스입니다.

```
#include <TestManager.hpp>
```

### Public 멤버 함수

- [TestManager](#) ([SystemState](#) &systemStateRef, [CanManager](#) &canManagerRef, std::map< std::string, std::shared\_ptr< [GenericMotor](#) > > &motorsRef)  
[TestManager](#) 클래스의 생성자.
- void [mainLoop](#) ()  
 테스트 매니저의 메인 루프를 실행하는 함수입니다. 주요 테스트 루틴을 실행하고 결과를 분석합니다.
- void [multiTestLoop](#) ()  
 다중 테스트 루프를 실행하는 함수입니다. 여러 테스트 케이스를 동시에 실행하여 모터의 동작을 평가합니다.
- void [TestArr](#) (double t, int cycles, int type, int LnR, double amp[])  
 테스트 배열을 생성하고 실행하는 함수입니다.

**Private 멤버 함수**

- void `move` ()  
모터를 이동시키는 함수입니다. 지정된 명령에 따라 모터를 이동시킵니다.
- void `mkArr` (vector< string > &motorName, int time, int cycles, int LnR, double amp)  
테스트 배열을 생성하는 함수입니다.
- void `SendLoop` ()  
생성된 배열을 모터에 전송하는 루프를 실행하는 함수입니다. 설정된 파라미터에 따라 모터에 명령을 전송합니다.
- void `parse_and_save_to_csv` (const std::string &csv\_file\_name)  
테스트 결과를 CSV 파일로 파싱하고 저장하는 함수입니다.
- void `FixMotorPosition` (std::shared\_ptr< `GenericMotor` > &motor)  
단일 모터의 위치를 고정하는 함수입니다.
- void `TuningTmotor` (float kp, float kd, float sine\_t, const std::string selectedMotor, int cycles, float peakAngle, int pathType)  
T 모터의 파라미터를 튜닝하는 함수입니다.
- void `TuningLoopTask` ()  
튜닝 루프 작업을 실행하는 함수입니다. 사용자에게 입력을 받아 모터의 튜닝 과정을 관리합니다.
- void `InitializeParameters` (const std::string selectedMotor, float &kp, float &kd, float &peakAngle, int &pathType, int &controlType, int &des\_vel, int &des\_tff, int &direction)  
선택된 모터와 테스트 파라미터를 초기화하는 함수입니다.
- void `TuningMaxonCSP` (float sine\_t, const std::string selectedMotor, int cycles, float peakAngle, int pathType)  
Maxon 모터의 CSP 모드를 튜닝하는 함수입니다.
- void `TuningMaxonCSV` (const std::string selectedMotor, int des\_vel, int direction)  
Maxon 모터의 CSV 모드를 튜닝하는 함수입니다.
- void `TuningMaxonCST` (const std::string selectedMotor, int des\_tff, int direction)  
Maxon 모터의 CST 모드를 튜닝하는 함수입니다.
- void `setMaxonMode` (std::string targetMode)  
Maxon 모터의 작동 모드를 설정하는 함수입니다.
- int `kbhit` ()  
키보드 입력이 있는지 확인하는 함수입니다. 사용자로부터의 입력을 비동기적으로 확인하기 위해 사용됩니다.
- void `TestStickLoop` ()  
스틱 모드 테스트 루프를 실행하는 함수입니다. 스틱 모드의 성능을 테스트하기 위해 사용됩니다.
- void `TestStick` (const std::string selectedMotor, int des\_tff, float tffThreshold, float posThreshold, int backTorqueUnit)  
특정 모터에 대한 스틱 모드 테스트를 실행하는 함수입니다.
- bool `dct_fun` (float positions[], float vel\_th)  
위치와 속도 임계값을 기반으로 DCT(Discrete Cosine Transform) 함수를 실행하는 함수입니다.

**Private 속성**

- `SystemState` & `systemState`  
시스템의 현재 상태입니다.
- `CanManager` & `canManager`  
CAN 통신을 통한 모터 제어를 담당합니다.
- std::map< std::string, std::shared\_ptr< `GenericMotor` > > & `motors`  
연결된 모터들의 정보입니다.
- `TMotorCommandParser` `tmotorecmd`  
T 모터 명령어 파서입니다.
- `MaxonCommandParser` `maxoncmd`  
Maxon 모터 명령어 파서입니다.
- vector< string > `InputData`  
테스트 입력 데이터입니다.

### 4.9.1 상세한 설명

모터의 성능 테스트 및 파라미터 튜닝을 위한 클래스입니다.

`TestManager` 클래스는 다양한 테스트 시나리오를 실행하여 모터의 성능을 평가하고, 최적의 운영 파라미터를 결정하기 위한 메서드를 제공합니다.

`TestManager.hpp` 파일의 39 번째 라인에서 정의되었습니다.

### 4.9.2 생성자 & 소멸자 문서화

#### 4.9.2.1 TestManager()

```
TestManager::TestManager (
    SystemState & systemStateRef,
    CanManager & canManagerRef,
    std::map< std::string, std::shared_ptr< GenericMotor > > & motorsRef )
```

`TestManager` 클래스의 생성자.

매개변수

systemStateRef	시스템 상태에 대한 참조입니다.
canManagerRef	CAN 통신을 관리하는 <code>CanManager</code> 클래스의 참조입니다.
motorsRef	연결된 모터들의 정보를 담고 있는 맵입니다.

`TestManager.cpp` 파일의 5 번째 라인에서 정의되었습니다.

```
00006 : systemState(systemStateRef), canManager(canManagerRef), motors(motorsRef)
00007 {
00008 }
```

### 4.9.3 멤버 함수 문서화

#### 4.9.3.1 dct\_fun()

```
bool TestManager::dct_fun (
    float positions[],
    float vel_th ) [private]
```

위치와 속도 임계값을 기반으로 DCT(Discrete Cosine Transform) 함수를 실행하는 함수입니다.

매개변수

positions	모터 위치 데이터 배열입니다.
vel_th	속도 임계값입니다. 위치와 속도 데이터를 분석하여 모터의 동작 품질을 평가합니다.

## 반환값

분석 결과에 따라 true 또는 false를 반환합니다.

TestManager.cpp 파일의 1731 번째 라인에서 정의되었습니다.

```
01732 {
01733     // 포지션 배열에서 각각의 값을 추출합니다.
01734     float the_k = positions[3]; // 가장 최신 값
01735     float the_k_1 = positions[2];
01736     float the_k_2 = positions[1];
01737     float the_k_3 = positions[0]; // 가장 오래된 값
01738
01739     float ang_k = (the_k + the_k_1) / 2;
01740     float ang_k_1 = (the_k_1 + the_k_2) / 2;
01741     float ang_k_2 = (the_k_2 + the_k_3) / 2;
01742     float vel_k = ang_k - ang_k_1;
01743     float vel_k_1 = ang_k_1 - ang_k_2;
01744
01745     if (vel_k > vel_k_1 && vel_k > vel_th && ang_k < 0.1 * M_PI)
01746         return true;
01747     else if (ang_k < -0.25 * M_PI)
01748         return true;
01749     else
01750         return false;
01751 }
```

## 4.9.3.2 FixMotorPosition()

```
void TestManager::FixMotorPosition (
    std::shared_ptr< GenericMotor > & motor ) [private]
```

단일 모터의 위치를 고정하는 함수입니다.

## 매개변수

motor	모터 객체의 공유 포인터입니다. 모터를 고정된 위치에 정확하게 유지하기 위해 사용됩니다.
-------	---

TestManager.cpp 파일의 631 번째 라인에서 정의되었습니다.

```
00632 {
00633     struct can_frame frame;
00634
00635     canManager.checkConnection(motor);
00636
00637     if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00638     {
00639         tmotorcmd.parseSendCommand(*tMotor, &frame, motor->nodeId, 8, motor->currentPos, 0, 250, 1,
00640 );
00641         if (canManager.sendAndRecv(motor, frame))
00642         {
00643             std::cout << "Position fixed for motor [" << motor->nodeId << "]." << std::endl;
00644         }
00645         else
00646         {
00647             std::cerr << "Failed to fix position for motor [" << motor->nodeId << "]." << std::endl;
00648         }
00649     }
00650     else if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00651     {
00652         maxoncmd.getTargetPosition(*maxonMotor, &frame, motor->currentPos);
00653         if (canManager.sendAndRecv(motor, frame))
00654         {
00655             std::cout << "Position fixed for motor [" << motor->nodeId << "]." << std::endl;
00656         }
00657         else
00658         {
00659             std::cerr << "Failed to fix position for motor [" << motor->nodeId << "]." << std::endl;
00660         }
00661     }
```

#### 4.9.3.3 InitializeParameters()

```
void TestManager::InitializeParameters (
    const std::string selectedMotor,
    float & kp,
    float & kd,
    float & peakAngle,
    int & pathType,
    int & controlType,
    int & des_vel,
    int & des_tff,
    int & direction ) [private]
```

선택된 모터와 테스트 파라미터를 초기화하는 함수입니다.

매개변수

selectedMotor	선택된 모터의 이름입니다.
kp	비례 제어 계수입니다.
kd	미분 제어 계수입니다.
peakAngle	최대 회전 각도입니다.
pathType	경로 유형입니다.
controlType	제어 유형입니다.
des_vel	목표 속도입니다.
des_tff	목표 토크 피드포워드 값입니다.
direction	회전 방향입니다. 튜닝 과정에서 사용될 파라미터를 사용자로부터 받아 설정합니다.

TestManager.cpp 파일의 1151 번째 라인에서 정의되었습니다.

```
01152 {
01153     if (selectedMotor == "waist")
01154     {
01155         kp = 200.0;
01156         kd = 1.0;
01157         peakAngle = 30;
01158         pathType = 2;
01159         des_vel = 0;
01160         des_tff = 0;
01161     }
01162     else if (selectedMotor == "R_arm1" || selectedMotor == "L_arm1" ||
01163             selectedMotor == "R_arm2" || selectedMotor == "R_arm3" ||
01164             selectedMotor == "L_arm2" || selectedMotor == "L_arm3")
01165     {
01166         kp = 50.0; // 예시 값, 실제 필요한 값으로 조정
01167         kd = 1.0; // 예시 값, 실제 필요한 값으로 조정
01168         peakAngle = 90;
01169         pathType = 1;
01170         des_vel = 0;
01171         des_tff = 0;
01172     }
01173     else if (selectedMotor == "L_wrist" || selectedMotor == "R_wrist" || selectedMotor ==
01174             "maxonForTest")
01175     {
01176         peakAngle = 90;
01177         pathType = 1;
01178         controlType = 1;
01179         direction = 1;
01180         des_vel = 0;
01181         des_tff = 0;
01182     }
01183     else if (selectedMotor == "maxonForTest")
01184     {
01185         peakAngle = 90;
01186         pathType = 1;
01187         controlType = 3;
01188         direction = -1;
01189         des_vel = 0;
01190         des_tff = 500;
01191     }
01192 }
```

```
01190     }
01191 }
```

#### 4.9.3.4 kbhit()

```
int TestManager::kbhit ( ) [private]
```

키보드 입력이 있는지 확인하는 함수입니다. 사용자로부터의 입력을 비동기적으로 확인하기 위해 사용됩니다.

반환값

키보드 입력이 있으면 1, 없으면 0을 반환합니다.

TestManager.cpp 파일의 1481 번째 라인에서 정의되었습니다.

```
01482 {
01483     struct termios oldt, newt;
01484     int ch;
01485     int oldf;
01486
01487     tcgetattr(STDIN_FILENO, &oldt);
01488     newt = oldt;
01489     newt.c_lflag &= ~(ICANON | ECHO);
01490     tcsetattr(STDIN_FILENO, TCSANOW, &newt);
01491     oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
01492     fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);
01493
01494     ch = getchar();
01495
01496     tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
01497     fcntl(STDIN_FILENO, F_SETFL, oldf);
01498
01499     if (ch != EOF)
01500     {
01501         ungetc(ch, stdin);
01502         return 1;
01503     }
01504
01505     return 0;
01506 }
```

#### 4.9.3.5 mainLoop()

```
void TestManager::mainLoop ( )
```

테스트 매니저의 메인 루프를 실행하는 함수입니다. 주요 테스트 루틴을 실행하고 결과를 분석합니다.

TestManager.cpp 파일의 10 번째 라인에서 정의되었습니다.

```
00011 {
00012     int choice;
00013     canManager.checkAllMotors();
00014     setMaxonMode("CSP");
00015     while (systemState.main == Main::Tune)
00016     {
00017         // 사용자에게 선택지 제공
00018         std::cout << "1: MultiMode\n2: SingleMode\n3: StickMode\n4: Exit\n";
00019         std::cout << "Select mode (1-4): ";
00020         std::cin >> choice;
00021
00022         // 선택에 따라 testMode 설정
00023         switch (choice)
00024         {
00025             case 1:
00026                 multiTestLoop();
00027                 break;
00028             case 2:
00029                 TuningLoopTask();
00030                 break;
00031             case 3:
00032                 TestStickLoop();
00033         }
00034     }
00035 }
```

```

00033         break;
00034     case 4:
00035         systemState.main = Main::Ideal;
00036         break;
00037     default:
00038         std::cout << "Invalid choice. Please try again.\n";
00039         continue;
00040     }
00041 }
00042 }

```

#### 4.9.3.6 mkArr()

```

void TestManager::mkArr (
    vector< string > & motorName,
    int time,
    int cycles,
    int LnR,
    double amp ) [private]

```

테스트 배열을 생성하는 함수입니다.

매개변수

motorName	모터의 이름입니다.
time	시간 주기입니다.
cycles	반복 횟수입니다.
LnR	왼쪽 또는 오른쪽 모터를 선택합니다.
amp	진폭입니다.

[TestManager.cpp](#) 파일의 48 번째 라인에서 정의되었습니다.

```

00049 {
00050     struct can_frame frame;
00051
00052     int Kp_fixed = 450;
00053     double Kd_fixed = 4.5;
00054     map<string, bool> TestMotor;
00055     if (LnR == 0) // 양쪽 다 고정
00056     {
00057         for (auto &motorname : motorName)
00058         {
00059             TestMotor[motorname] = false;
00060         }
00061     }
00062     else if (LnR == 1) // 오른쪽만 Test
00063     {
00064         for (auto &motorname : motorName)
00065         {
00066             if (motorname[0] == 'L')
00067                 TestMotor[motorname] = false;
00068             else
00069                 TestMotor[motorname] = true;
00070         }
00071     }
00072     else if (LnR == 2) // 왼쪽만 Test
00073     {
00074         for (auto &motorname : motorName)
00075         {
00076             if (motorname[0] == 'R')
00077                 TestMotor[motorname] = false;
00078             else
00079                 TestMotor[motorname] = true;
00080         }
00081     }
00082     else if (LnR == 3) // 양쪽 다 Test
00083     {
00084         for (auto &motorname : motorName)
00085         {
00086             TestMotor[motorname] = true;
00087         }
00088     }
00089 }

```



```

00088     }
00089
00090     amp = amp / 180.0 * M_PI; // Degree -> Radian 변경
00091     for (const auto &motorname : motorName)
00092     {
00093         if (motors.find(motorname) != motors.end())
00094         {
00095             std::cout << motorname << " ";
00096             if (TestMotor[motorname])
00097             { // Test 하는 모터
00098                 std::cout << "Move\n";
00099                 InputData[0] += motorname + ",";
00100                 if (std::shared_ptr<TMotor> tMotor =
std::dynamic_pointer_cast<TMotor>(motors[motorname]))
00101                 {
00102                     int kp = tMotor->Kp;
00103                     double kd = tMotor->Kd;
00104
00105                     for (int c = 0; c < cycles; c++)
00106                     {
00107                         for (int i = 0; i < time; i++)
00108                         {
00109                             float val = tMotor->currentPos + (1.0 - cos(2.0 * M_PI * i / time)) / 2 *
amp * tMotor->cwDir;
00110                             tMotor->sendBuffer.push(frame);
00111                             InputData[time * c + i + 1] += to_string(val) + ",";
00112                         }
00113                     }
00114                 }
00115             }
00116             else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[motorname]))
00117             {
00118                 for (int c = 0; c < cycles; c++)
00119                 {
00120                     for (int i = 0; i < time; i++)
00121                     {
00122                         float val = maxonMotor->currentPos + (1.0 - cos(2.0 * M_PI * i / time)) /
2 * amp * maxonMotor->cwDir;
00123                         maxonMotor->sendBuffer.push(frame);
00124                         InputData[time * c + i + 1] += to_string(val) + ",";
00125                     }
00126                 }
00127             }
00128         }
00129     }
00130     else
00131     { // Fixed 하는 모터
00132         std::cout << "Fixed\n";
00133         InputData[0] += motorname + ",";
00134         if (std::shared_ptr<TMotor> tMotor =
std::dynamic_pointer_cast<TMotor>(motors[motorname]))
00135         {
00136             for (int c = 0; c < cycles; c++)
00137             {
00138                 for (int i = 0; i < time; i++)
00139                 {
00140                     float val = tMotor->currentPos;
00141                     tMotor->sendBuffer.push(frame);
00142                     InputData[time * c + i + 1] += to_string(val) + ",";
00143                 }
00144             }
00145         }
00146     }
00147     else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[motorname]))
00148     {
00149         for (int c = 0; c < cycles; c++)
00150         {
00151             for (int i = 0; i < time; i++)
00152             {
00153                 float val = maxonMotor->currentPos;
00154                 maxonMotor->sendBuffer.push(frame);
00155                 InputData[time * c + i + 1] += to_string(val) + ",";
00156             }
00157         }
00158     }
00159 }
00160 }
00161 }
00162 }
00163 }

```

#### 4.9.3.7 move()

```
void TestManager::move ( ) [private]
```

모터를 이동시키는 함수입니다. 지정된 명령에 따라 모터를 이동시킵니다.

#### 4.9.3.8 multiTestLoop()

```
void TestManager::multiTestLoop ( )
```

다중 테스트 루프를 실행하는 함수입니다. 여러 테스트 케이스를 동시에 실행하여 모터의 동작을 평가합니다.

[TestManager.cpp](#) 파일의 227 번째 라인에서 정의되었습니다.

```
00228 {
00229     string userInput;
00230     vector<double> c_deg;
00231     double t = 4.0;
00232     int cycles = 1;
00233     int type = 0b00001;
00234     int LnR = 1;
00235     double amplitude[5] = {30.0, 30.0, 30.0, 30.0, 30.0};
00236
00237     while (systemState.main == Main::Tune)
00238     {
00239         int result = system("clear");
00240         if (result != 0)
00241         {
00242             cerr << "Error during clear screen" << std::endl;
00243         }
00244
00245         string typeDescription;
00246         if ((type | 0b11110) == 0b11111)
00247         {
00248             typeDescription += "Arm3 Turn, ";
00249         }
00250         if ((type | 0b11101) == 0b11111)
00251         {
00252             typeDescription += "Arm2 Turn, ";
00253         }
00254         if ((type | 0b11011) == 0b11111)
00255         {
00256             typeDescription += "Arm1 Turn, ";
00257         }
00258         if ((type | 0b10111) == 0b11111)
00259         {
00260             typeDescription += "Waist Turn, ";
00261         }
00262         if ((type | 0b01111) == 0b11111)
00263         {
00264             typeDescription += "Wrist Turn, ";
00265         }
00266
00267         std::string LeftAndRight;
00268         if (LnR == 1)
00269         {
00270             LeftAndRight = "<< Right move >>\n";
00271         }
00272         else if (LnR == 2)
00273         {
00274             LeftAndRight = "<< Left move >>\n";
00275         }
00276         else if (LnR == 3)
00277         {
00278             LeftAndRight = "<< Left and Right move >>\n";
00279         }
00280
00281         std::cout <<
00282             "-----\n";
00283         std::cout << "< Current Position >\n";
00284         for (auto &motor : motors)
00285         {
00286             c_deg.push_back(motor.second->currentPos * motor.second->cwDir / M_PI * 180);
00287             std::cout << motor.first << " : " << motor.second->currentPos * motor.second->cwDir / M_PI *
180 << "deg\n";
00288         }
00289         std::cout << "\n"
```

```

00289         « LeftAndRight;
00290     std::cout << "Type : " << typeDescription << "\n";
00291     std::cout << "Period : " << t << "\n";
00292     std::cout << "Cycles : " << cycles << "\n";
00293     std::cout << "\nAmplitude :\n";
00294     std::cout << "1) Arm3 = " << amplitude[0] << "\n";
00295     std::cout << "2) Arm2 = " << amplitude[1] << "\n";
00296     std::cout << "3) Arm1 = " << amplitude[2] << "\n";
00297     std::cout << "4) Waist = " << amplitude[3] << "\n";
00298     std::cout << "5) Wrist = " << amplitude[4] << "\n";
00299     std::cout <<
    "-----\n";

00300
00301     std::cout << "[Commands]\n";
00302     std::cout << "[d] : Left and Right | [t] : Type | [p] : Period | [c] : Cycles\n"
00303         << "[a] : Amplitude | [kp] : Kp | [kd] : Kd | [m] : move | [r] : run | [e] : Exit\n";
00304     std::cout << "Enter Command: ";
00305     std::cin >> userInput;
00306
00307     if (userInput[0] == 'e')
00308     {
00309         systemState.main = Main::Ideal;
00310         break;
00311     }
00312     else if (userInput[0] == 'd')
00313     {
00314         std::cout << "\nEnter Desired Direction\n";
00315         std::cout << "1: Right Move\n";
00316         std::cout << "2: Left Move\n";
00317         std::cout << "3: Left and Right Move\n";
00318         std::cout << "\nEnter Path Type (1 or 2 or 3): ";
00319         std::cin >> LnR;
00320     }
00321     else if (userInput[0] == 't')
00322     {
00323         int num;
00324         std::cout << "\nEnter Desired Type\n";
00325         std::cout << "1: Arm3 Turn ON/OFF\n";
00326         std::cout << "2: Arm2 Turn ON/OFF\n";
00327         std::cout << "3: Arm1 Turn ON/OFF\n";
00328         std::cout << "4: Waist Turn ON/OFF\n";
00329         std::cout << "5: Wrist Turn ON/OFF\n";
00330         std::cout << "\nEnter Path Type (1 or 2 or 3 or 4 or 5): ";
00331         std::cin >> num;
00332
00333         if (num == 1)
00334         {
00335             type = type ^ 0b00001;
00336         }
00337         else if (num == 2)
00338         {
00339             type = type ^ 0b00010;
00340         }
00341         else if (num == 3)
00342         {
00343             type = type ^ 0b00100;
00344         }
00345         else if (num == 4)
00346         {
00347             type = type ^ 0b01000;
00348         }
00349         else if (num == 5)
00350         {
00351             type = type ^ 0b10000;
00352         }
00353     }
00354     else if (userInput[0] == 'p')
00355     {
00356         std::cout << "\nEnter Desired Period : ";
00357         std::cin >> t;
00358     }
00359     else if (userInput[0] == 'c')
00360     {
00361         std::cout << "\nEnter Desired Cycles : ";
00362         std::cin >> cycles;
00363     }
00364     else if (userInput[0] == 'a')
00365     {
00366         int input;
00367         std::cout << "\n[Select Motor]\n";
00368         std::cout << "1: Arm3\n";
00369         std::cout << "2: Arm2\n";
00370         std::cout << "3: Arm1\n";
00371         std::cout << "4: Waist\n";
00372         std::cout << "5: Wrist\n";
00373         std::cout << "\nEnter Desired Motor : ";
00374         std::cin >> input;

```

```

00375
00376         std::cout << "\nEnter Desired Amplitude(degree) : ";
00377         std::cin >> amplitude[input - 1];
00378     }
00379     else if (userInput == "kp")
00380     {
00381         char input;
00382         int kp;
00383         std::cout << "\n[Select Motor]\n";
00384         std::cout << "1: Arm3\n";
00385         std::cout << "2: Arm2\n";
00386         std::cout << "3: Arm1\n";
00387         std::cout << "4: Waist\n";
00388         std::cout << "\nEnter Desired Motor : ";
00389         std::cin >> input;
00390
00391         if (input == '1')
00392         {
00393             std::cout << "Arm3's Kp : " << motors["R_arm3"]->Kp << "\n";
00394             std::cout << "Enter Arm3's Desired Kp : ";
00395             std::cin >> kp;
00396             motors["R_arm3"]->Kp = kp;
00397             // motors["L_arm3"]->Kp = kp;
00398         }
00399         else if (input == '2')
00400         {
00401             std::cout << "Arm2's Kp : " << motors["R_arm2"]->Kp << "\n";
00402             std::cout << "Enter Arm2's Desired Kp : ";
00403             std::cin >> kp;
00404             motors["R_arm2"]->Kp = kp;
00405             motors["L_arm2"]->Kp = kp;
00406         }
00407         else if (input == '3')
00408         {
00409             std::cout << "Arm1's Kp : " << motors["R_arm1"]->Kp << "\n";
00410             std::cout << "Enter Arm1's Desired Kp : ";
00411             std::cin >> kp;
00412             motors["R_arm1"]->Kp = kp;
00413             motors["L_arm1"]->Kp = kp;
00414         }
00415         else if (input == '4')
00416         {
00417             std::cout << "Waist's Kp : " << motors["waist"]->Kp << "\n";
00418             std::cout << "Enter Waist's Desired Kp : ";
00419             std::cin >> kp;
00420             motors["waist"]->Kp = kp;
00421         }
00422     }
00423     else if (userInput == "kd")
00424     {
00425         char input;
00426         int kd;
00427         std::cout << "\n[Select Motor]\n";
00428         std::cout << "1: Arm3\n";
00429         std::cout << "2: Arm2\n";
00430         std::cout << "3: Arm1\n";
00431         std::cout << "4: Waist\n";
00432         std::cout << "\nEnter Desired Motor : ";
00433         std::cin >> input;
00434
00435         if (input == '1')
00436         {
00437             std::cout << "Arm3's Kd : " << motors["R_arm3"]->Kd << "\n";
00438             std::cout << "Enter Arm3's Desired Kd : ";
00439             std::cin >> kd;
00440             motors["R_arm3"]->Kd = kd;
00441             // motors["L_arm3"]->Kd = kd;
00442         }
00443         else if (input == '2')
00444         {
00445             std::cout << "Arm2's Kd : " << motors["R_arm2"]->Kd << "\n";
00446             std::cout << "Enter Arm2's Desired Kd : ";
00447             std::cin >> kd;
00448             motors["R_arm2"]->Kd = kd;
00449             motors["L_arm2"]->Kd = kd;
00450         }
00451         else if (input == '3')
00452         {
00453             std::cout << "Arm1's Kd : " << motors["R_arm1"]->Kd << "\n";
00454             std::cout << "Enter Arm1's Desired Kd : ";
00455             std::cin >> kd;
00456             motors["R_arm1"]->Kd = kd;
00457             motors["L_arm1"]->Kd = kd;
00458         }
00459         else if (input == '4')
00460         {
00461             std::cout << "Waist's Kd : " << motors["waist"]->Kd << "\n";

```

```

00462         std::cout << "Enter Waist's Desired Kd : ";
00463         std::cin >> kd;
00464         motors["waist"]->Kd = kd;
00465     }
00466 } /*
00467 else if (userInput[0] == 'm')
00468 {
00469     while (true)
00470     {
00471         string input;
00472         double deg;
00473         cout << "\n[Move to]\n";
00474         int i = 0;
00475         for (auto &motor : motors)
00476         {
00477             cout << i + 1 << " - " << motor.first << " : " << c_deg[i] << "deg\n";
00478             i++;
00479         }
00480         cout << "m - Move\n";
00481         cout << "e - Exit\n";
00482         cout << "\nEnter Desired Option : ";
00483         cin >> input;
00484
00485         if (input[0] == 'e')
00486         {
00487             break;
00488         }
00489         else if (input[0] == 'm')
00490         {
00491             // 움직이는 함수 작성
00492             // 나중에 이동할 위치 값 : c_deg * motor.second->cwDir / 180 * M_PI
00493             break;
00494         }
00495         else
00496         {
00497             cout << "\nEnter Desired Degree : ";
00498             cin >> deg;
00499             c_deg[stoi(input) - 1] = deg;
00500         }
00501     }
00502 }*/
00503 else if (userInput[0] == 'r')
00504 {
00505     TestArr(t, cycles, type, LnR, amplitude);
00506 }
00507 }
00508 }

```

#### 4.9.3.9 parse\_and\_save\_to\_csv()

```

void TestManager::parse_and_save_to_csv (
    const std::string & csv_file_name ) [private]

```

테스트 결과를 CSV 파일로 파싱하고 저장하는 함수입니다.

매개변수

csv_file_name ↵	저장할 CSV 파일의 이름입니다.
-----------------	--------------------

[TestManager.cpp](#) 파일의 572 번째 라인에서 정의되었습니다.

```

00573 {
00574     // CSV 파일 열기. 파일이 없으면 새로 생성됩니다.
00575     std::ofstream ofs(csv_file_name, std::ios::app);
00576     if (!ofs.is_open())
00577     {
00578         std::cerr << "Failed to open or create the CSV file: " << csv_file_name << std::endl;
00579         return;
00580     }
00581
00582     // 파일이 새로 생성되었으면 CSV 헤더를 추가
00583     ofs.seekp(0, std::ios::end);
00584     if (ofs.tellp() == 0)
00585     {
00586         ofs << "CAN_ID,p_act,tff_des,tff_act\n";

```

```

00587     }
00588
00589     // 각 모터에 대한 처리
00590     for (const auto &pair : motors)
00591     {
00592         auto &motor = pair.second;
00593         if (!motor->recieveBuffer.empty())
00594         {
00595             can_frame frame = motor->recieveBuffer.front();
00596             motor->recieveBuffer.pop();
00597
00598             int id = motor->nodeId;
00599             float position, speed, torque;
00600
00601             // TMotor 또는 MaxonMotor에 따른 데이터 파싱 및 출력
00602             if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00603             {
00604                 std::tuple<int, float, float, float> parsedData =
00605                 tmotorcmd.parseRecieveCommand(*tMotor, &frame);
00606                 position = std::get<1>(parsedData);
00607                 speed = std::get<2>(parsedData);
00608                 torque = std::get<3>(parsedData);
00609             }
00610             else if (std::shared_ptr<MaxonMotor> maxonMotor =
00611             std::dynamic_pointer_cast<MaxonMotor>(motor))
00612             {
00613                 std::tuple<int, float, float> parsedData = maxoncmd.parseRecieveCommand(*maxonMotor,
00614                 &frame);
00615                 position = std::get<1>(parsedData);
00616                 torque = std::get<2>(parsedData);
00617                 speed = 0.0;
00618             }
00619             // 데이터 CSV 파일에 쓰기
00620             ofs << "0x" << std::hex << std::setw(4) << std::setfill('0') << id << ", "
00621             << std::dec << position << ", " << speed << ", " << torque << "\n";
00622         }
00623     }
00624     ofs.close();
00625     std::cout << "연주 txt_OutData 파일이 생성되었습니다." << csv_file_name << std::endl;

```

#### 4.9.3.10 SendLoop()

```
void TestManager::SendLoop ( ) [private]
```

생성된 배열을 모터에 전송하는 루프를 실행하는 함수입니다. 설정된 파라미터에 따라 모터에 명령을 전송합니다.

TestManager.cpp 파일의 165 번째 라인에서 정의되었습니다.

```

00166 {
00167     std::cout << "Settig...\n";
00168     struct can_frame frameToProcess;
00169     std::string maxonCanInterface;
00170     std::shared_ptr<GenericMotor> virtualMaxonMotor;
00171
00172     int maxonMotorCount = 0;
00173     for (const auto &motor_pair : motors)
00174     {
00175         // 각 요소가 MaxonMotor 타입인지 확인
00176         if (std::shared_ptr<MaxonMotor> maxonMotor =
00177         std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00178         {
00179             maxonMotorCount++;
00180             maxonCanInterface = maxonMotor->interFaceName;
00181             virtualMaxonMotor = motor_pair.second;
00182         }
00183     }
00184     chrono::system_clock::time_point external = std::chrono::system_clock::now();
00185
00186     bool allBuffersEmpty;
00187     do
00188     {
00189         allBuffersEmpty = true;
00190         for (const auto &motor_pair : motors)
00191         {
00192             if (!motor_pair.second->sendBuffer.empty())
00193             {
00194                 allBuffersEmpty = false;

```

```

00194         break;
00195     }
00196 }
00197
00198 if (!allBuffersEmpty)
00199 {
00200     chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00201     chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
external);
00202
00203     if (elapsed_time.count() >= 5000) // 5ms
00204     {
00205         external = std::chrono::system_clock::now();
00206
00207         for (auto &motor_pair : motors)
00208         {
00209             shared_ptr<GenericMotor> motor = motor_pair.second;
00210             canManager.sendFromBuff(motor);
00211         }
00212
00213         if (maxonMotorCount != 0)
00214         {
00215             maxoncmd.getSync(&frameToProcess);
00216             canManager.txFrame(virtualMaxonMotor, frameToProcess);
00217         }
00218
00219         // canManager.readFramesFromAllSockets();
00220         // canManager.distributeFramesToMotors();
00221     }
00222 }
00223 } while (!allBuffersEmpty);
00224 canManager.clearReadBuffers();
00225 }

```

#### 4.9.3.11 setMaxonMode()

```

void TestManager::setMaxonMode (
    std::string targetMode ) [private]

```

Maxon 모터의 작동 모드를 설정하는 함수입니다.

매개변수

targetMode	설정할 모드의 이름입니다. 모터의 작동 모드를 변경하기 위해 사용됩니다.
------------	--

TestManager.cpp 파일의 1447 번째 라인에서 정의되었습니다.

```

1448 {
1449     struct can_frame frame;
1450     canManager.setSocketsTimeout(0, 10000);
1451     for (const auto &motorPair : motors)
1452     {
1453         std::string name = motorPair.first;
1454         std::shared_ptr<GenericMotor> motor = motorPair.second;
1455         if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motorPair.second))
1456         {
1457             if (targetMode == "CSV")
1458             {
1459                 maxoncmd.getCSVMode(*maxonMotor, &frame);
1460                 canManager.sendAndRecv(motor, frame);
1461             }
1462             else if (targetMode == "CST")
1463             {
1464                 maxoncmd.getCSTMode(*maxonMotor, &frame);
1465                 canManager.sendAndRecv(motor, frame);
1466             }
1467             else if (targetMode == "HMM")
1468             {
1469                 maxoncmd.getHomeMode(*maxonMotor, &frame);
1470                 canManager.sendAndRecv(motor, frame);
1471             }
1472             else if (targetMode == "CSP")
1473             {
1474                 maxoncmd.getCSPMode(*maxonMotor, &frame);
1475                 canManager.sendAndRecv(motor, frame);

```

```

01476         }
01477     }
01478 }
01479 }

```

#### 4.9.3.12 TestArr()

```

void TestManager::TestArr (
    double t,
    int cycles,
    int type,
    int LnR,
    double amp[ ] )

```

테스트 배열을 생성하고 실행하는 함수입니다.

매개변수

t	시간 주기입니다.
cycles	반복 횟수입니다.
type	테스트 유형입니다.
LnR	왼쪽 또는 오른쪽 모터를 선택합니다.
amp	진폭 배열입니다.

[TestManager.cpp](#) 파일의 510 번째 라인에서 정의되었습니다.

```

00511 {
00512     std::cout << "Test Start!!\n";
00513
00514     int time = t / 0.005;
00515     std::vector<std::string> SmotorName;
00516     InputData.clear();
00517     InputData.resize(time * cycles + 1);
00518
00519     SmotorName = {"waist"};
00520     if ((type | 0b10111) == 0b11111) // Turn Waist
00521         mkArr(SmotorName, time, cycles, LnR, amp[3]);
00522     else
00523         mkArr(SmotorName, time, cycles, 0, 0);
00524
00525     SmotorName = {"R_arm1", "L_arm1"};
00526     if ((type | 0b11011) == 0b11111) // Turn Arm1
00527         mkArr(SmotorName, time, cycles, LnR, amp[2]);
00528     else
00529         mkArr(SmotorName, time, cycles, 0, 0);
00530
00531     SmotorName = {"R_arm2", "L_arm2"};
00532     if ((type | 0b11101) == 0b11111) // Turn Arm2
00533         mkArr(SmotorName, time, cycles, LnR, amp[1]);
00534     else
00535         mkArr(SmotorName, time, cycles, 0, 0);
00536
00537     SmotorName = {"R_arm3", "L_arm3"};
00538     if ((type | 0b11110) == 0b11111) // Turn Arm3
00539         mkArr(SmotorName, time, cycles, LnR, amp[0]);
00540     else
00541         mkArr(SmotorName, time, cycles, 0, 0);
00542
00543     SmotorName = {"R_wrist", "L_wrist", "maxonForTest"};
00544     if ((type | 0b01111) == 0b11111) // Turn Wrist
00545         mkArr(SmotorName, time, cycles, LnR, amp[4]);
00546     else
00547         mkArr(SmotorName, time, cycles, 0, 0);
00548
00549     // TXT 파일 열기
00550     string FileNameIn = "../../READ/test_in.txt";
00551     ofstream csvFileIn(FileNameIn);
00552     if (!csvFileIn.is_open())
00553     {
00554         std::cerr << "Error opening TXT file." << std::endl;
00555     }

```



```

00556
00557 // TXT 파일 입력
00558 for (auto &data : InputData)
00559 {
00560     csvFileIn << data << "\n";
00561 }
00562
00563 // CSV 파일 닫기
00564 csvFileIn.close();
00565 std::cout << "연주 txt_InData 파일이 생성되었습니다." << FileNameIn << std::endl;
00566
00567 SendLoop();
00568
00569 parse_and_save_to_csv("../..//READ/test_out.txt");
00570 }

```

#### 4.9.3.13 TestStick()

```

void TestManager::TestStick (
    const std::string selectedMotor,
    int des_tff,
    float tffThreshold,
    float posThreshold,
    int backTorqueUnit ) [private]

```

특정 모터에 대한 스틱 모드 테스트를 실행하는 함수입니다.

매개변수

selectedMotor	선택된 모터의 이름입니다.
des_tff	목표 토크 피드포워드 값입니다.
tffThreshold	토크 피드포워드 임계값입니다.
posThreshold	위치 임계값입니다.
backTorque← Unit	역토크 단위입니다. 모터의 스틱 모드 성능을 평가하기 위해 사용됩니다.

TestManager.cpp 파일의 1598 번째 라인에서 정의되었습니다.

```

01599 {
01600
01601     canManager.setSocketsTimeout(0, 50000);
01602     std::string FileName1 = "../..//READ/" + selectedMotor + "_cst_in.txt";
01603
01604     std::ofstream csvFileIn(FileName1);
01605
01606     if (!csvFileIn.is_open())
01607     {
01608         std::cerr << "Error opening CSV file." << std::endl;
01609     }
01610
01611     // Input File
01612     csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
01613     std::string FileName2 = "../..//READ/" + selectedMotor + "_cst_out.txt";
01614     std::ofstream csvFileOut(FileName2);
01615
01616     if (!csvFileOut.is_open())
01617     {
01618         std::cerr << "Error opening CSV file." << std::endl;
01619     }
01620     csvFileOut << "CAN_ID,pos_act,tff_act\n"; // CSV 헤더
01621
01622     struct can_frame frame;
01623
01624     float p_act, tff_act;
01625
01626     std::shared_ptr<MaxonMotor> maxonMotor =
01627         std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]);
01628     for (int i = 0; i < (int)maxonMotor->nodeId - 1; i++)
01629     {

```

```

01630         csvFileIn << "0,";
01631     }
01632     csvFileIn << std::dec << des_tff << ",";
01633     for (int i = 0; i < (9 - (int)maxonMotor->nodeId); i++)
01634     {
01635         csvFileIn << "0,";
01636     }
01637     bool reachedDrum = false;
01638     bool motorFixed = false;
01639
01640     chrono::system_clock::time_point external = std::chrono::system_clock::now();
01641     bool motorModeSet = false;
01642
01643     float positionValues[4] = {0}; // 포지션 값 저장을 위한 정적 배열
01644     int posIndex = 0;              // 현재 포지션 값 인덱스
01645
01646     while (1)
01647     {
01648
01649         if (!motorModeSet)
01650         {
01651             maxoncmd.getCSTMode(*maxonMotor, &frame);
01652             canManager.sendAndRecv(motors[selectedMotor], frame);
01653             motorModeSet = true; // 모터 모드 설정 완료
01654         }
01655         if (motorFixed)
01656         {
01657             break;
01658         }
01659
01660         chrono::system_clock::time_point internal = std::chrono::system_clock::now();
01661         chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
external);
01662         if (elapsed_time.count() >= 5000)
01663         {
01664
01665             maxoncmd.getTargetTorque(*maxonMotor, &frame, des_tff);
01666             canManager.txFrame(motors[selectedMotor], frame);
01667
01668             maxoncmd.getSync(&frame);
01669             canManager.txFrame(motors[selectedMotor], frame);
01670
01671             if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))
01672             {
01673                 while (!motors[selectedMotor]->recieveBuffer.empty())
01674                 {
01675                     frame = motors[selectedMotor]->recieveBuffer.front();
01676                     if (frame.can_id == maxonMotor->rxPdoIds[0])
01677                     {
01678                         std::tuple<int, float, float> result =
maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
01679
01680                         p_act = std::get<1>(result);
01681                         tff_act = std::get<2>(result);
01682                         csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') <<
maxonMotor->nodeId;
01683                         csvFileOut << ',' << std::dec << p_act << "," << tff_act << '\n';
01684
01685                         positionValues[posIndex % 4] = p_act;
01686                         posIndex++;
01687
01688                         if (!reachedDrum && dct_fun(positionValues, 0))
01689                         {
01690                             des_tff = backTorqueUnit;
01691                             reachedDrum = true;
01692                         }
01693
01694                         // 특정 각도에 도달했는지 확인하는 조건
01695                         if (p_act > posThreshold && reachedDrum)
01696                         {
01697                             maxoncmd.getCSPMode(*maxonMotor, &frame);
01698                             canManager.sendAndRecv(motors[selectedMotor], frame);
01699
01700                             maxoncmd.getTargetPosition(*maxonMotor, &frame, p_act);
01701                             canManager.txFrame(motors[selectedMotor], frame);
01702                             maxoncmd.getSync(&frame);
01703                             canManager.txFrame(motors[selectedMotor], frame);
01704                             if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))
01705                             {
01706                                 while (!motors[selectedMotor]->recieveBuffer.empty())
01707                                 {
01708                                     frame = motors[selectedMotor]->recieveBuffer.front();
01709                                     if (frame.can_id == maxonMotor->rxPdoIds[0])
01710                                     {
01711                                         motorFixed = true;
01712                                     }
01713                                     motors[selectedMotor]->recieveBuffer.pop();

```

```

01714         }
01715     }
01716 }
01717     }
01718     if (!motors[selectedMotor]->recieveBuffer.empty())
01719     {
01720         motors[selectedMotor]->recieveBuffer.pop();
01721     }
01722 }
01723 }
01724 }
01725 }
01726
01727 csvFileIn.close();
01728 csvFileOut.close();
01729 }

```

#### 4.9.3.14 TestStickLoop()

```
void TestManager::TestStickLoop ( ) [private]
```

스틱 모드 테스트 루프를 실행하는 함수입니다. 스틱 모드의 성능을 테스트하기 위해 사용됩니다.

TestManager.cpp 파일의 1512 번째 라인에서 정의되었습니다.

```

01513 {
01514     std::string userInput;
01515     std::string selectedMotor = "maxonForTest";
01516     float des_tff = 0;
01517     float posThreshold = 1.57; // 위치 임계값 초기화
01518     float tffThreshold = 18;   // 토크 임계값 초기화
01519     int backTorqueUnit = 150;
01520     for (auto motor_pair : motors)
01521     {
01522         FixMotorPosition(motor_pair.second);
01523     }
01524     while (true)
01525     {
01526         int result = system("clear");
01527         if (result != 0)
01528         {
01529             std::cerr << "Error during clear screen" << std::endl;
01530         }
01531         std::cout << "===== Tuning Menu =====\n";
01532         std::cout << "Available Motors for Stick Mode:\n";
01533         for (const auto &motor_pair : motors)
01534         {
01535             if (motor_pair.first == "maxonForTest")
01536                 std::cout << " - " << motor_pair.first << "\n";
01537         }
01538         bool isMaxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]) != nullptr;
01539         if (!isMaxonMotor)
01540             break;
01541         std::cout << "-----\n";
01542         std::cout << "Selected Motor: " << selectedMotor << "\n";
01543         std::cout << "Des Torque: " << des_tff * 31.052 / 1000 << " [mNm]\n";
01544         std::cout << "Torque Threshold: " << tffThreshold << " [mNm]\n"; // 현재 토크 임계값 출력
01545         std::cout << "Position Threshold: " << posThreshold << " [rad]\n";
01546         std::cout << "Back Torque: " << backTorqueUnit * 31.052 / 1000 << " [mNm]\n";
01547         std::cout << "\nCommands:\n";
01548         std::cout << "[a]: des_tff | [b]: Direction | [c]: Back Torque\n";
01549         std::cout << "[d]: Set Torque Threshold [e]: Set Position Threshold\n";
01550         std::cout << "[f]: Run | [g]: Exit\n";
01551         std::cout << "=====\n";
01552         std::cout << "Enter Command: ";
01553         std::cin >> userInput;
01554         std::transform(userInput.begin(), userInput.end(), userInput.begin(), ::tolower);
01555         if (userInput[0] == 'g')
01556         {
01557             break;
01558         }
01559         else if (userInput == "c")

```

```

01569     {
01570         std::cout << "Enter Desired [Back] Torque In Unit: ";
01571         std::cout << "100 [unit] = 3.1052 [mNm]\n";
01572         std::cin >> backTorqueUnit;
01573     }
01574     else if (userInput == "a" && isMaxonMotor)
01575     {
01576         std::cout << "Enter Desired Torque In Unit: ";
01577         std::cout << "-100 [unit] = -3.1052 [mNm]\n";
01578         std::cin >> des_tff;
01579     }
01580     else if (userInput == "d" && isMaxonMotor)
01581     {
01582         std::cout << "Enter Desired Torque Threshold: ";
01583         std::cout << "-100 [unit] = -3.1052 [mNm]\n";
01584         std::cin >> tffThreshold;
01585     }
01586     else if (userInput == "e" && isMaxonMotor)
01587     {
01588         std::cout << "Enter Desired Position Threshold: ";
01589         std::cin >> posThreshold;
01590     }
01591     else if (userInput[0] == 'f' && isMaxonMotor)
01592     {
01593         TestStick(selectedMotor, des_tff, tffThreshold, posThreshold, backTorqueUnit);
01594     }
01595 }
01596 }

```

#### 4.9.3.15 TuningLoopTask()

```
void TestManager::TuningLoopTask ( ) [private]
```

튜닝 루프 작업을 실행하는 함수입니다. 사용자에게 입력을 받아 모터의 튜닝 과정을 관리합니다.

[TestManager.cpp](#) 파일의 897 번째 라인에서 정의되었습니다.

```

00898 {
00899     for (auto motor_pair : motors)
00900     {
00901         FixMotorPosition(motor_pair.second);
00902     }
00903
00904     std::string userInput, selectedMotor, fileName;
00905     float kp, kd, peakAngle;
00906     float sine_t = 4.0;
00907     int cycles = 2, pathType, controlType, des_vel, des_tff, direction;
00908
00909     selectedMotor = motors.begin()->first;
00910
00911     InitializeParameters(selectedMotor, kp, kd, peakAngle, pathType, controlType, des_vel, des_tff,
direction);
00912     while (true)
00913     {
00914         int result = system("clear");
00915         if (result != 0)
00916         {
00917             std::cerr << "Error during clear screen" << std::endl;
00918         }
00919
00920         std::string pathTypeDescription;
00921         if (pathType == 1)
00922         {
00923             pathTypeDescription = "1: 1 - cos (0 -> peak -> 0)";
00924         }
00925         else if (pathType == 2)
00926         {
00927             pathTypeDescription = "2: 1 - cos & cos - 1 (0 -> peak -> 0 -> -peak -> 0)";
00928         }
00929
00930         std::string controlTypeDescription;
00931         if (controlType == 1)
00932         {
00933             controlTypeDescription = "CSP";
00934
00935             setMaxonMode("CSP");
00936         }
00937         else if (controlType == 2)
00938         {
00939             controlTypeDescription = "CSV";
00940

```

```

00941         setMaxonMode("CSV");
00942     }
00943     else if (controlType == 3)
00944     {
00945         controlTypeDescription = "CST";
00946
00947         setMaxonMode("CST");
00948     }
00949     else if (controlType == 4)
00950     {
00951         controlTypeDescription = "Drum Test";
00952     }
00953
00954     std::string directionDescription;
00955     if (direction == 1)
00956     {
00957         directionDescription = "CW";
00958     }
00959     else if (direction == 2)
00960     {
00961         directionDescription = "CCW";
00962     }
00963
00964     std::cout << "===== Tuning Menu =====\n";
00965     std::cout << "Available Motors:\n";
00966
00967     for (const auto &motor_pair : motors)
00968     {
00969         std::cout << " - " << motor_pair.first << "\n";
00970     }
00971     bool isMaxonMotor;
00972     if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]))
00973     {
00974         isMaxonMotor = true;
00975     }
00976     else
00977     {
00978         isMaxonMotor = false;
00979     }
00980
00981     std::cout << "-----\n";
00982     std::cout << "Selected Motor: " << selectedMotor << "\n";
00983     if (!isMaxonMotor)
00984     {
00985         std::cout << "Kp: " << kp << " | Kd: " << kd << "\n";
00986     }
00987     else
00988     {
00989         std::cout << "Control Type : " << controlTypeDescription;
00990         std::cout << " | Vel [rpm]: " << des_vel << " | Des Torque: " << des_ttf * 31.052 / 1000 <<
"[mNm]"
00991         << "\n";
00992     };
00993     std::cout << "Sine Period: " << sine_t << " | Cycles: " << cycles << " | Hz: " << 1 / sine_t << "\n";
00994     std::cout << "Peak Angle: " << peakAngle << " | Path Type [Pos]: " << pathTypeDescription << "\n";
00995     std::cout << "Direction: " << directionDescription << "\n";
00996     std::cout << "\nCommands:\n";
00997     if (!isMaxonMotor)
00998     {
00999         std::cout << "[a]: kp | [b]: kd |";
01000     }
01001     else
01002     {
01003         std::cout << "[a]: des_vel | [b]: des_ttf | [c]: Control |";
01004     }
01005     std::cout << "[d]: Select Motor | [e]: Peak | [f]: Path\n";
01006     std::cout << "[g]: Period | [h]: Cycles | [i]: Run \n";
01007     if (isMaxonMotor)
01008     {
01009         std::cout << "[k]: Direction | ";
01010     }
01011     std::cout << "[j]: Exit\n";
01012     std::cout << "===== \n";
01013     std::cout << "Enter Command: ";
01014     std::cin >> userInput;
01015     std::transform(userInput.begin(), userInput.end(), userInput.begin(), ::tolower);
01016
01017     if (userInput[0] == 'j')
01018     {
01019         break;
01020     }
01021     else if (userInput[0] == 'd')
01022     {
01023         while (true)
01024         {
01025             std::cout << "Enter the name of the motor to tune: ";

```

```

01026         std::cin >> selectedMotor;
01027         if (motors.find(selectedMotor) != motors.end())
01028         {
01029             InitializeParameters(selectedMotor, kp, kd, peakAngle, pathType, controlType,
des_vel, des_tff, direction);
01030             break;
01031         }
01032         else
01033         {
01034             std::cout << "Invalid motor name. Please enter a valid motor name.\n";
01035         }
01036     }
01037 }
01038 else if (userInput == "k" && isMaxonMotor)
01039 {
01040     std::cout << "Enter Desired Direction: ";
01041     std::cout << "1: Clock Wise\n";
01042     std::cout << "2: Counter Clock Wise\n";
01043     std::cout << "Enter Path Type (1 or 2): ";
01044     std::cin >> direction;
01045
01046     if (direction != 1 && direction != 2)
01047     {
01048         std::cout << "Invalid direction type. Please enter 1 or 2.\n";
01049         pathType = 1;
01050     }
01051
01052     std::cin >> direction;
01053 }
01054 else if (userInput == "a" && !isMaxonMotor)
01055 {
01056     std::cout << "Enter Desired Kp: ";
01057     std::cin >> kp;
01058 }
01059 else if (userInput == "b" && !isMaxonMotor)
01060 {
01061     std::cout << "Enter Desired Kd: ";
01062     std::cin >> kd;
01063 }
01064 else if (userInput == "a" && isMaxonMotor)
01065 {
01066     std::cout << "Enter Desired Velocity: ";
01067     std::cin >> des_vel;
01068 }
01069 else if (userInput == "b" && isMaxonMotor)
01070 {
01071     std::cout << "Enter Desired Torque: ";
01072     std::cout << "-100 [unit] = -3.1052 [mNm]\n";
01073     std::cin >> des_tff;
01074 }
01075 else if (userInput == "e")
01076 {
01077     std::cout << "Enter Desired Peak Angle: ";
01078     std::cin >> peakAngle;
01079 }
01080 else if (userInput == "f")
01081 {
01082     std::cout << "Select Path Type:\n";
01083     std::cout << "1: 1 - cos (0 -> peak -> 0)\n";
01084     std::cout << "2: 1 - cos & cos - 1 (0 -> peak -> 0 -> -peak -> 0)\n";
01085     std::cout << "Enter Path Type (1 or 2): ";
01086     std::cin >> pathType;
01087
01088     if (pathType != 1 && pathType != 2)
01089     {
01090         std::cout << "Invalid path type. Please enter 1 or 2.\n";
01091         pathType = 1;
01092     }
01093 }
01094 else if (userInput[0] == 'g')
01095 {
01096     std::cout << "Enter Desired Sine Period: ";
01097     std::cin >> sine_t;
01098 }
01099 else if (userInput[0] == 'h')
01100 {
01101     std::cout << "Enter Desired Cycles: ";
01102     std::cin >> cycles;
01103 }
01104 else if (userInput == "c" && isMaxonMotor)
01105 {
01106     std::cout << "Select Control Type:\n";
01107     std::cout << "1: Cyclic Synchronous Position Mode (CSP)\n";
01108     std::cout << "2: Cyclic Synchronous Velocity Mode (CSV)\n";
01109     std::cout << "3: Cyclic Synchronous Torque Mode (CST)\n";
01110     std::cout << "4: Maxon Drum Test (CSP)\n";
01111     std::cout << "Enter Path Type (1 or 2 or 3 or 4): ";

```

```

01112         std::cin » controlType;
01113
01114         if (controlType != 1 && controlType != 2 && controlType != 3 && controlType != 4)
01115         {
01116             std::cout << "Invalid path type. Please enter 1 or 2 or 3 or 4.\n";
01117             controlType = 1;
01118         }
01119     }
01120     else if (userInput[0] == 'i')
01121     {
01122         if (!isMaxonMotor) // Tmotor일 경우
01123         {
01124             TuningTmotor(kp, kd, sine_t, selectedMotor, cycles, peakAngle, pathType);
01125         }
01126         else // MaxonMotor일 경우
01127         {
01128             if (controlType == 1)
01129             {
01130                 TuningMaxonCSP(sine_t, selectedMotor, cycles, peakAngle, pathType);
01131             }
01132             else if (controlType == 2)
01133             {
01134                 TuningMaxonCSV(selectedMotor, des_vel, direction);
01135             }
01136             else if (controlType == 3)
01137             {
01138                 TuningMaxonCST(selectedMotor, des_tff, direction);
01139             }
01140             else if (controlType == 4)
01141             {
01142                 //
01143             }
01144         }
01145     }
01146 }
01147 }
01148 }
01149 }

```

#### 4.9.3.16 TuningMaxonCSP()

```

void TestManager::TuningMaxonCSP (
    float sine_t,
    const std::string selectedMotor,
    int cycles,
    float peakAngle,
    int pathType ) [private]

```

Maxon 모터의 CSP 모드를 튜닝하는 함수입니다.

매개변수

sine_t	사인 파형의 주기입니다.
selectedMotor	선택된 모터의 이름입니다.
cycles	실행할 사이클 수입니다.
peakAngle	최대 회전 각도입니다.
pathType	경로 유형입니다. Maxon 모터의 위치 제어 성능을 최적화하기 위해 사용됩니다.

[TestManager.cpp](#) 파일의 778 번째 라인에서 정의되었습니다.

```

00779 {
00780     canManager.setSocketsTimeout(0, 50000);
00781     std::string FileName1 = ".././READ/" + selectedMotor + "_in.txt";
00782
00783     std::ofstream csvFileIn(FileName1);
00784
00785     if (!csvFileIn.is_open())
00786     {
00787         std::cerr << "Error opening CSV file." << std::endl;
00788     }
00789 }

```

```

00790
00791 // 헤더 추가
00792 csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
00793
00794 // CSV 파일명 설정
00795 std::string FileName2 = ".././READ/" + selectedMotor + "_out.txt";
00796
00797 // CSV 파일 열기
00798 std::ofstream csvFileOut(FileName2);
00799
00800 if (!csvFileOut.is_open())
00801 {
00802     std::cerr << "Error opening CSV file." << std::endl;
00803 }
00804 csvFileOut << "CAN_ID,p_act,v_act,tff_act\n"; // CSV 헤더
00805
00806 struct can_frame frame;
00807
00808 float peakRadian = peakAngle * M_PI / 180.0; // 피크 각도를 라디안으로 변환
00809 float amplitude = peakRadian;
00810
00811 float sample_time = 0.005;
00812 int max_samples = static_cast<int>(sine_t / sample_time);
00813 float p_des = 0;
00814 float p_act;
00815 // float tff_des = 0, v_des = 0;
00816 float tff_act;
00817
00818 std::shared_ptr<MaxonMotor> maxonMotor =
00819 std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]);
00820 for (int cycle = 0; cycle < cycles; cycle++)
00821 {
00822     for (int i = 0; i < max_samples; i++)
00823     {
00824         float time = i * sample_time;
00825
00826         for (int i = 0; i < (int)maxonMotor->nodeId - 1; i++)
00827         {
00828             csvFileIn << "0,";
00829         }
00830         csvFileIn << std::dec << p_des << ",";
00831         for (int i = 0; i < (9 - (int)maxonMotor->nodeId); i++)
00832         {
00833             csvFileIn << "0,";
00834         }
00835         float local_time = std::fmod(time, sine_t);
00836         if (pathType == 1) // 1-cos 경로
00837         {
00838             p_des = amplitude * (1 - cosf(2 * M_PI * local_time / sine_t)) / 2;
00839         }
00840         else if (pathType == 2) // 1-cos 및 -1+cos 결합 경로
00841         {
00842             if (local_time < sine_t / 2)
00843                 p_des = amplitude * (1 - cosf(4 * M_PI * local_time / sine_t)) / 2;
00844             else
00845                 p_des = amplitude * (-1 + cosf(4 * M_PI * (local_time - sine_t / 2) / sine_t)) /
00846 2;
00847         }
00848         maxoncmd.getTargetPosition(*maxonMotor, &frame, p_des);
00849         csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') << maxonMotor->nodeId;
00850
00851         chrono::system_clock::time_point external = std::chrono::system_clock::now();
00852         while (1)
00853         {
00854             chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00855             chrono::microseconds elapsed_time =
00856             chrono::duration_cast<chrono::microseconds>(internal - external);
00857             if (elapsed_time.count() >= 5000)
00858             {
00859                 ssize_t bytesWritten = write(canManager.sockets.at(maxonMotor->interFaceName),
00860 &frame, sizeof(struct can_frame));
00861                 if (bytesWritten == -1)
00862                 {
00863                     std::cerr << "Failed to write to socket for interface: " <<
00864 maxonMotor->interFaceName << std::endl;
00865                     std::cerr << "Error: " << strerror(errno) << " (errno: " << errno << ") " <<
00866 std::endl;
00867                 }
00868                 canManager.txFrame(motors[selectedMotor], frame);
00869                 maxoncmd.getSync(&frame);
00870                 canManager.txFrame(motors[selectedMotor], frame);
00871                 if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))

```



```

00871         {
00872             while (!motors[selectedMotor]->recieveBuffer.empty())
00873             {
00874                 frame = motors[selectedMotor]->recieveBuffer.front();
00875                 if (frame.can_id == maxonMotor->rxPdoIds[0])
00876                 {
00877                     std::tuple<int, float, float> result =
maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
00878
00879                     p_act = std::get<1>(result);
00880                     tff_act = std::get<2>(result);
00881                     // tff_des = kp * (p_des - p_act) + kd * (v_des - v_act);
00882                     csvFileOut << ',' << std::dec << p_act << ", " << tff_act << '\n';
00883                     break;
00884                 }
00885                 motors[selectedMotor]->recieveBuffer.pop();
00886             }
00887         }
00888     }
00889 }
00890     csvFileIn << "\n";
00891 }
00892 }
00893 csvFileIn.close();
00894 csvFileOut.close();
00895 }

```

#### 4.9.3.17 TuningMaxonCST()

```

void TestManager::TuningMaxonCST (
    const std::string selectedMotor,
    int des_tff,
    int direction ) [private]

```

Maxon 모터의 CST 모드를 튜닝하는 함수입니다.

매개변수

selectedMotor	선택된 모터의 이름입니다.
des_tff	목표 토크 피드포워드 값입니다.
direction	회전 방향입니다. 토크 제어 성능을 평가하고 최적화하기 위해 사용됩니다.

[TestManager.cpp](#) 파일의 1303 번째 라인에서 정의되었습니다.

```

01304 {
01305     if (direction == 1)
01306         des_tff *= 1;
01307     else
01308         des_tff *= -1;
01309
01310     canManager.setSocketsTimeout(0, 50000);
01311     std::string FileName1 = "../READ/" + selectedMotor + "_cst_in.txt";
01312
01313     std::ofstream csvFileIn(FileName1);
01314
01315     if (!csvFileIn.is_open())
01316     {
01317         std::cerr << "Error opening CSV file." << std::endl;
01318     }
01319
01320     // 헤더 추가
01321     csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
01322
01323     // CSV 파일명 설정
01324     std::string FileName2 = "../READ/" + selectedMotor + "_cst_out.txt";
01325
01326     // CSV 파일 열기
01327     std::ofstream csvFileOut(FileName2);
01328
01329     if (!csvFileOut.is_open())
01330     {
01331         std::cerr << "Error opening CSV file." << std::endl;
01332     }
01333     csvFileOut << "CAN_ID,pos_act,tff_act\n"; // CSV 헤더

```

```

01334
01335     struct can_frame frame;
01336
01337     float p_act, tff_act;
01338     char input = 'a';
01339     std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]);
01340
01341     for (int i = 0; i < (int)maxonMotor->nodeId - 1; i++)
01342     {
01343         csvFileIn << "0,";
01344     }
01345     csvFileIn << std::dec << des_tff << ",";
01346     for (int i = 0; i < (9 - (int)maxonMotor->nodeId); i++)
01347     {
01348         csvFileIn << "0,";
01349     }
01350     bool reachedDrum = false;
01351     bool waitForEInput = false;
01352     chrono::system_clock::time_point external = std::chrono::system_clock::now();
01353     while (1)
01354     {
01355
01356         if (kbhit())
01357         {
01358             input = getchar();
01359             if (input == 'e' && waitForEInput) // waitForEInput이 true일 때만 'e' 입력 처리
01360             {
01361                 maxoncmd.getCSTMode(*maxonMotor, &frame);
01362                 canManager.sendAndRecv(motors[selectedMotor], frame);
01363                 break;
01364             }
01365         }
01366
01367         if (waitForEInput)
01368         {
01369             // 'e' 입력을 기다리는 동안 다른 작업을 하지 않음
01370             continue;
01371         }
01372
01373         chrono::system_clock::time_point internal = std::chrono::system_clock::now();
01374         chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
external);
01375         if (elapsed_time.count() >= 5000)
01376         {
01377
01378             maxoncmd.getTargetTorque(*maxonMotor, &frame, des_tff);
01379             canManager.txFrame(motors[selectedMotor], frame);
01380
01381             maxoncmd.getSync(&frame);
01382             canManager.txFrame(motors[selectedMotor], frame);
01383
01384             if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))
01385             {
01386                 while (!motors[selectedMotor]->recieveBuffer.empty())
01387                 {
01388                     frame = motors[selectedMotor]->recieveBuffer.front();
01389                     if (frame.can_id == maxonMotor->rxPdoIds[0])
01390                     {
01391                         std::tuple<int, float, float> result =
maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
01392
01393                         p_act = std::get<1>(result);
01394                         tff_act = std::get<2>(result);
01395                         // tff_des = kp * (p_des - p_act) + kd * (v_des - v_act);
01396                         csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') <<
maxonMotor->nodeId;
01397
01398                         csvFileOut << ',' << std::dec << p_act << "," << tff_act << '\n';
01399
01400                         // 임계 토크 값을 체크하고, 조건을 충족하면 반대 방향으로 토크 주기
01401                         if (abs(tff_act) > 18)
01402                         {
01403                             des_tff = 100;
01404                             reachedDrum = true;
01405                         }
01406
01407                         // 특정 각도에 도달했는지 확인하는 조건
01408                         if (p_act > -0.5 && reachedDrum)
01409                         {
01410                             maxoncmd.getCSPMode(*maxonMotor, &frame);
01411                             canManager.sendAndRecv(motors[selectedMotor], frame);
01412
01413                             canManager.checkConnection(motors[selectedMotor]);
01414                             maxoncmd.getTargetPosition(*maxonMotor, &frame,
motors[selectedMotor]->currentPos);
01415                             canManager.txFrame(motors[selectedMotor], frame);
01416                             maxoncmd.getSync(&frame);

```

```

01416         canManager.txFrame(motors[selectedMotor], frame);
01417         if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))
01418         {
01419             while (!motors[selectedMotor]->recieveBuffer.empty())
01420             {
01421                 frame = motors[selectedMotor]->recieveBuffer.front();
01422                 if (frame.can_id == maxonMotor->rxPdoIds[0])
01423                 {
01424                     std::cout << "This is My stick!! \n";
01425                 }
01426                 motors[selectedMotor]->recieveBuffer.pop();
01427             }
01428         }
01429
01430         waitForEInput = true; // 'e' 입력 대기 상태로 전환
01431         std::cout << "Waiting for 'e' input...\n";
01432     }
01433 }
01434 if (!motors[selectedMotor]->recieveBuffer.empty())
01435 {
01436     motors[selectedMotor]->recieveBuffer.pop();
01437 }
01438 }
01439 }
01440 }
01441 }
01442
01443 csvFileIn.close();
01444 csvFileOut.close();
01445 }

```

#### 4.9.3.18 TuningMaxonCSV()

```

void TestManager::TuningMaxonCSV (
    const std::string selectedMotor,
    int des_vel,
    int direction ) [private]

```

Maxon 모터의 CSV 모드를 튜닝하는 함수입니다.

매개변수

selectedMotor	선택된 모터의 이름입니다.
des_vel	목표 속도입니다.
direction	회전 방향입니다. 속도 제어 성능을 평가하고 최적화하기 위해 사용됩니다.

[TestManager.cpp](#) 파일의 1193 번째 라인에서 정의되었습니다.

```

01194 {
01195     des_vel = des_vel * 35;
01196
01197     if (direction == 1)
01198         des_vel *= 1;
01199     else
01200         des_vel *= -1;
01201
01202     canManager.setSocketsTimeout(0, 50000);
01203     std::string FileName1 = "../READ/" + selectedMotor + "_csv_in.txt";
01204
01205     std::ofstream csvFileIn(FileName1);
01206
01207     if (!csvFileIn.is_open())
01208     {
01209         std::cerr << "Error opening CSV file." << std::endl;
01210     }
01211
01212     // 헤더 추가
01213     csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
01214
01215     // CSV 파일명 설정
01216     std::string FileName2 = "../READ/" + selectedMotor + "_csv_out.txt";
01217
01218     // CSV 파일 열기
01219     std::ofstream csvFileOut(FileName2);

```

```

01220
01221     if (!csvFileOut.is_open())
01222     {
01223         std::cerr << "Error opening CSV file." << std::endl;
01224     }
01225     csvFileOut << "CAN_ID,pos_act,tff_act\n"; // CSV 헤더
01226
01227     struct can_frame frame;
01228
01229     float p_act, tff_act;
01230     char input = 'a';
01231     std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]);
01232
01233     for (int i = 0; i < (int)maxonMotor->nodeId - 1; i++)
01234     {
01235         csvFileIn << "0,";
01236     }
01237     csvFileIn << std::dec << des_vel << ",";
01238     for (int i = 0; i < (9 - (int)maxonMotor->nodeId); i++)
01239     {
01240         csvFileIn << "0,";
01241     }
01242
01243     maxoncmd.getTargetVelocity(*maxonMotor, &frame, des_vel);
01244
01245     chrono::system_clock::time_point external = std::chrono::system_clock::now();
01246     while (1)
01247     {
01248
01249         if (kbhit())
01250         {
01251             input = getchar();
01252         }
01253
01254         if (input == 'q')
01255             continue;
01256         else if (input == 'e')
01257             break;
01258
01259         chrono::system_clock::time_point internal = std::chrono::system_clock::now();
01260         chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
external);
01261         if (elapsed_time.count() >= 5000)
01262         {
01263
01264             ssize_t bytesWritten = write(canManager.sockets.at(maxonMotor->interFaceName), &frame,
sizeof(struct can_frame));
01265             if (bytesWritten == -1)
01266             {
01267                 std::cerr << "Failed to write to socket for interface: " << maxonMotor->interFaceName <<
std::endl;
01268                 std::cerr << "Error: " << strerror(errno) << " (errno: " << errno << ")" << std::endl;
01269             }
01270
01271             maxoncmd.getSync(&frame);
01272             bytesWritten = write(canManager.sockets.at(maxonMotor->interFaceName), &frame,
sizeof(struct can_frame));
01273             if (bytesWritten == -1)
01274             {
01275                 std::cerr << "Failed to write to socket for interface: " << maxonMotor->interFaceName <<
std::endl;
01276                 std::cerr << "Error: " << strerror(errno) << " (errno: " << errno << ")" << std::endl;
01277             }
01278             ssize_t bytesRead = read(canManager.sockets.at(maxonMotor->interFaceName), &frame,
sizeof(struct can_frame));
01279             if (bytesRead == -1)
01280             {
01281                 std::cerr << "Failed to read from socket for interface: " << maxonMotor->interFaceName <<
std::endl;
01282                 return;
01283             }
01284             else
01285             {
01286
01287                 std::tuple<int, float, float> result = maxoncmd.parseRecieveCommand(*maxonMotor,
&frame);
01288
01289                 p_act = std::get<1>(result);
01290                 tff_act = std::get<2>(result);
01291                 // tff_des = kp * (p_des - p_act) + kd * (v_des - v_act);
01292                 csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') << maxonMotor->nodeId;
01293                 csvFileOut << ',' << std::dec << p_act << "," << tff_act << '\n';
01294             }
01295         }
01296     }
01297 }

```

```

01298
01299     csvFileIn.close();
01300     csvFileOut.close();
01301 }

```

#### 4.9.3.19 TuningTmotor()

```

void TestManager::TuningTmotor (
    float kp,
    float kd,
    float sine_t,
    const std::string selectedMotor,
    int cycles,
    float peakAngle,
    int pathType ) [private]

```

T 모터의 파라미터를 튜닝하는 함수입니다.

매개변수

kp	비례 제어 계수입니다.
kd	미분 제어 계수입니다.
sine_t	사인 파형의 주기입니다.
selectedMotor	선택된 모터의 이름입니다.
cycles	실행할 사이클 수입니다.
peakAngle	최대 회전 각도입니다.
pathType	경로 유형입니다. T 모터의 응답성과 정확도를 개선하기 위해 사용됩니다.

[TestManager.cpp](#) 파일의 663 번째 라인에서 정의되었습니다.

```

00664 {
00665     canManager.setSocketsTimeout(0, 50000);
00666
00667     std::stringstream ss;
00668     ss << std::fixed << std::setprecision(2); // 소수점 둘째 자리까지만
00669     ss << "kp_" << kp << "_kd_" << kd << "_Hz_" << 1 / sine_t;
00670     std::string parameter = ss.str();
00671
00672     // CSV 파일명 설정
00673     std::string FileName1 = "../READ/" + parameter + "_in.txt";
00674
00675     // CSV 파일 열기
00676     std::ofstream csvFileIn(FileName1);
00677
00678     if (!csvFileIn.is_open())
00679     {
00680         std::cerr << "Error opening CSV file." << std::endl;
00681     }
00682
00683     // 헤더 추가
00684     csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
00685
00686     // CSV 파일명 설정
00687     std::string FileName2 = "../READ/" + parameter + "_out.txt";
00688
00689     // CSV 파일 열기
00690     std::ofstream csvFileOut(FileName2);
00691
00692     if (!csvFileOut.is_open())
00693     {
00694         std::cerr << "Error opening CSV file." << std::endl;
00695     }
00696
00697     // 헤더 추가
00698     csvFileOut << "CAN_ID,p_act,v_act,tff_act\n"; // CSV 헤더
00699
00700     struct can_frame frame;
00701

```

```

00702     float peakRadian = peakAngle * M_PI / 180.0; // 피크 각도를 라디안으로 변환
00703     float amplitude = peakRadian;
00704
00705     float sample_time = 0.005;
00706     int max_samples = static_cast<int>(sine_t / sample_time);
00707     float v_des = 0, p_des = 0;
00708     float tff_des = 0;
00709     float p_act, v_act, tff_act;
00710     std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motors[selectedMotor]);
00711     chrono::system_clock::time_point external = std::chrono::system_clock::now();
00712     for (int cycle = 0; cycle < cycles; cycle++)
00713     {
00714         for (int i = 0; i < max_samples; i++)
00715         {
00716             float time = i * sample_time;
00717
00718             if ((int)tMotor->nodeId == 7)
00719             {
00720                 csvFileIn << std::dec << p_des << "0,0,0,0,0,0,0,0";
00721             }
00722             else
00723             {
00724                 for (int i = 0; i < (int)tMotor->nodeId; i++)
00725                 {
00726                     csvFileIn << "0,";
00727                 }
00728                 csvFileIn << std::dec << p_des << ",";
00729                 for (int i = 0; i < (8 - (int)tMotor->nodeId); i++)
00730                 {
00731                     csvFileIn << "0,";
00732                 }
00733             }
00734
00735             float local_time = std::fmod(time, sine_t);
00736             if (pathType == 1) // 1-cos 경로
00737             {
00738                 p_des = amplitude * (1 - cosf(2 * M_PI * local_time / sine_t)) / 2;
00739             }
00740             else if (pathType == 2) // 1-cos 및 -1+cos 결합 경로
00741             {
00742                 if (local_time < sine_t / 2)
00743                     p_des = amplitude * (1 - cosf(4 * M_PI * local_time / sine_t)) / 2;
00744                 else
00745                     p_des = amplitude * (-1 + cosf(4 * M_PI * (local_time - sine_t / 2) / sine_t)) /
00746                 2;
00747             }
00748             tmotorcmd.parseSendCommand(*tMotor, &frame, tMotor->nodeId, 8, p_des, v_des, kp, kd,
00749             tff_des);
00750             csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') << tMotor->nodeId;
00751             while (1)
00752             {
00753                 chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00754                 chrono::microseconds elapsed_time =
00755                 chrono::duration_cast<chrono::microseconds>(internal - external);
00756                 if (elapsed_time.count() >= 5000)
00757                 {
00758                     external = std::chrono::system_clock::now();
00759                     if (canManager.sendAndRecv(motors[selectedMotor], frame))
00760                     {
00761                         std::tuple<int, float, float, float> result =
00762                         tmotorcmd.parseRecieveCommand(*tMotor, &frame);
00763                         p_act = std::get<1>(result);
00764                         v_act = std::get<2>(result);
00765                         tff_act = std::get<3>(result);
00766                         // tff_des = kp * (p_des - p_act) + kd * (v_des - v_act);
00767                         csvFileOut << ',' << std::dec << p_act << ',' << v_act << ',' << tff_act << '\n';
00768                         break;
00769                     }
00770                 }
00771                 csvFileIn << "\n";
00772             }
00773         }
00774         csvFileIn.close();
00775         csvFileOut.close();
00776     }

```

## 4.9.4 멤버 데이터 문서화

### 4.9.4.1 canManager

```
CanManager& TestManager::canManager [private]
```

CAN 통신을 통한 모터 제어를 담당합니다.

[TestManager.hpp](#) 파일의 74 번째 라인에서 정의되었습니다.

### 4.9.4.2 InputData

```
vector<string> TestManager::InputData [private]
```

테스트 입력 데이터입니다.

[TestManager.hpp](#) 파일의 80 번째 라인에서 정의되었습니다.

### 4.9.4.3 maxoncmd

```
MaxonCommandParser TestManager::maxoncmd [private]
```

Maxon 모터 명령어 파서입니다.

[TestManager.hpp](#) 파일의 78 번째 라인에서 정의되었습니다.

### 4.9.4.4 motors

```
std::map<std::string, std::shared_ptr<GenericMotor> >& TestManager::motors [private]
```

연결된 모터들의 정보입니다.

[TestManager.hpp](#) 파일의 75 번째 라인에서 정의되었습니다.

### 4.9.4.5 systemState

```
SystemState& TestManager::systemState [private]
```

시스템의 현재 상태입니다.

[TestManager.hpp](#) 파일의 73 번째 라인에서 정의되었습니다.

#### 4.9.4.6 tmotorcmd

`TMotorCommandParser` `TestManager::tmotorcmd` [private]

T 모터 명령어 파서입니다.

`TestManager.hpp` 파일의 77 번째 라인에서 정의되었습니다.

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- `include/managers/TestManager.hpp`
- `src/TestManager.cpp`

## 4.10 TMotor 클래스 참조

TMotor를 위한 클래스입니다. `GenericMotor`를 상속받습니다.

```
#include <Motor.hpp>
```

### Public 멤버 함수

- `TMotor` (`uint32_t nodeId`, `const std::string &motorType`)
- `void clearSendBuffer ()`  
송신 버퍼를 클리어합니다.
- `void clearReceiveBuffer ()`  
수신 버퍼를 클리어합니다.

### Public 속성

- `std::string motorType`  
모터의 타입.
- `int sensorBit`  
센서 비트.
- `uint32_t nodeId`  
모터의 노드 ID.
- `std::string interFaceName`  
모터가 연결된 인터페이스의 이름.
- `float desPos`
- `float desVel`
- `float desTor`  
목표 위치, 속도, 토크.
- `float currentPos`
- `float currentVel`
- `float currentTor`  
현재 위치, 속도, 토크.
- `float cwDir`  
시계 방향 회전을 나타내는 방향 값.
- `bool isHomed`
- `bool isConnected`



- 홈 위치에 있는지, 연결되어 있는지의 상태.
- float `rMin`  
회전 범위의 최소, 최대 값.
- int `socket`  
모터가 연결된 소켓의 식별자.
- int `Kp`  
비례 제어 게인.
- double `Kd`  
미분 제어 게인.
- `std::queue< can_frame >` `sendBuffer`  
송신 버퍼.
- `std::queue< can_frame >` `recieveBuffer`  
수신 버퍼.

### 4.10.1 상세한 설명

TMotor를 위한 클래스입니다. GenericMotor를 상속받습니다.

TMotor 특화된 기능과 속성을 정의합니다.

Motor.hpp 파일의 53 번째 라인에서 정의되었습니다.

### 4.10.2 생성자 & 소멸자 문서화

#### 4.10.2.1 TMotor()

```
TMotor::TMotor (
    uint32_t nodeId,
    const std::string & motorType )
```

Motor.cpp 파일의 28 번째 라인에서 정의되었습니다.

```
00029 : GenericMotor(nodeId, interFaceName), motorType(motorType)
00030 {
00031 }
```

### 4.10.3 멤버 함수 문서화

#### 4.10.3.1 clearReceiveBuffer()

```
void GenericMotor::clearReceiveBuffer ( ) [inherited]
```

수신 버퍼를 클리어합니다.

Motor.cpp 파일의 16 번째 라인에서 정의되었습니다.

```
00017 {
00018     while (!recieveBuffer.empty())
00019     {
00020         recieveBuffer.pop();
00021     }
00022 }
```

#### 4.10.3.2 clearSendBuffer()

```
void GenericMotor::clearSendBuffer ( ) [inherited]
```

송신 버퍼를 클리어합니다.

[Motor.cpp](#) 파일의 8 번째 라인에서 정의되었습니다.

```
00009 {
00010     while (!sendBuffer.empty())
00011     {
00012         sendBuffer.pop();
00013     }
00014 }
```

### 4.10.4 멤버 데이터 문서화

#### 4.10.4.1 currentPos

```
float GenericMotor::currentPos [inherited]
```

[Motor.hpp](#) 파일의 30 번째 라인에서 정의되었습니다.

#### 4.10.4.2 currentTor

```
float GenericMotor::currentTor [inherited]
```

현재 위치, 속도, 토크.

[Motor.hpp](#) 파일의 30 번째 라인에서 정의되었습니다.

#### 4.10.4.3 currentVel

```
float GenericMotor::currentVel [inherited]
```

[Motor.hpp](#) 파일의 30 번째 라인에서 정의되었습니다.

#### 4.10.4.4 cwDir

```
float GenericMotor::cwDir [inherited]
```

시계 방향 회전을 나타내는 방향 값.

[Motor.hpp](#) 파일의 31 번째 라인에서 정의되었습니다.

#### 4.10.4.5 desPos

```
float GenericMotor::desPos [inherited]
```

[Motor.hpp](#) 파일의 29 번째 라인에서 정의되었습니다.

#### 4.10.4.6 desTor

```
float GenericMotor::desTor [inherited]
```

목표 위치, 속도, 토크.

[Motor.hpp](#) 파일의 29 번째 라인에서 정의되었습니다.

#### 4.10.4.7 desVel

```
float GenericMotor::desVel [inherited]
```

[Motor.hpp](#) 파일의 29 번째 라인에서 정의되었습니다.

#### 4.10.4.8 interFaceName

```
std::string GenericMotor::interFaceName [inherited]
```

모터가 연결된 인터페이스의 이름.

[Motor.hpp](#) 파일의 28 번째 라인에서 정의되었습니다.

#### 4.10.4.9 isConnected

```
bool GenericMotor::isConected [inherited]
```

홈 위치에 있는지, 연결되어 있는지의 상태.

[Motor.hpp](#) 파일의 32 번째 라인에서 정의되었습니다.

#### 4.10.4.10 isHomed

```
bool GenericMotor::isHomed [inherited]
```

[Motor.hpp](#) 파일의 32 번째 라인에서 정의되었습니다.

#### 4.10.4.11 Kd

```
double GenericMotor::Kd [inherited]
```

미분 제어 게인.

[Motor.hpp](#) 파일의 36 번째 라인에서 정의되었습니다.

#### 4.10.4.12 Kp

```
int GenericMotor::Kp [inherited]
```

비례 제어 게인.

[Motor.hpp](#) 파일의 35 번째 라인에서 정의되었습니다.

#### 4.10.4.13 motorType

```
std::string TMotor::motorType
```

모터의 타입.

[Motor.hpp](#) 파일의 57 번째 라인에서 정의되었습니다.

#### 4.10.4.14 nodeId

```
uint32_t GenericMotor::nodeId [inherited]
```

모터의 노드 ID.

[Motor.hpp](#) 파일의 27 번째 라인에서 정의되었습니다.

#### 4.10.4.15 recieveBuffer

```
std::queue<can_frame> GenericMotor::recieveBuffer [inherited]
```

수신 버퍼.

[Motor.hpp](#) 파일의 38 번째 라인에서 정의되었습니다.

#### 4.10.4.16 rMax

```
float GenericMotor::rMax [inherited]
```

회전 범위의 최소, 최대 값.

[Motor.hpp](#) 파일의 33 번째 라인에서 정의되었습니다.

#### 4.10.4.17 rMin

```
float GenericMotor::rMin [inherited]
```

[Motor.hpp](#) 파일의 33 번째 라인에서 정의되었습니다.

## 4.10.4.18 sendBuffer

```
std::queue<can_frame> GenericMotor::sendBuffer [inherited]
```

송신 버퍼.

[Motor.hpp](#) 파일의 37 번째 라인에서 정의되었습니다.

## 4.10.4.19 sensorBit

```
int TMotor::sensorBit
```

센서 비트.

[Motor.hpp](#) 파일의 59 번째 라인에서 정의되었습니다.

## 4.10.4.20 socket

```
int GenericMotor::socket [inherited]
```

모터가 연결된 소켓의 식별자.

[Motor.hpp](#) 파일의 34 번째 라인에서 정의되었습니다.

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- include/motors/Motor.hpp
- src/Motor.cpp

## 4.11 TMotorCommandParser 클래스 참조

[TMotor](#) 명령어를 파싱하는 클래스입니다.

```
#include <CommandParser.hpp>
```

## Public 멤버 함수

- void [parseSendCommand](#) (TMotor &motor, struct can\_frame \*frame, int canId, int dlc, float p←\_des, float v←\_des, float kp, float kd, float t←\_ff)  
TMotor로 송신할 명령을 파싱합니다.
- std::tuple< int, float, float, float > [parseRecieveCommand](#) (TMotor &motor, struct can\_frame \*frame)  
TMotor로부터 수신된 명령을 파싱합니다.
- void [getCheck](#) (TMotor &motor, struct can\_frame \*frame)
- void [getControlMode](#) (TMotor &motor, struct can\_frame \*frame)
- void [getExit](#) (TMotor &motor, struct can\_frame \*frame)
- void [getZero](#) (TMotor &motor, struct can\_frame \*frame)
- void [getQuickStop](#) (TMotor &motor, struct can\_frame \*frame)

**Public 속성**

- float `GLOBAL_P_MIN` = -12.5  
최소 위치 제한.
- float `GLOBAL_P_MAX` = 12.5  
최대 위치 제한.
- float `GLOBAL_KP_MIN` = 0  
최소 비례 게인 제한.
- float `GLOBAL_KP_MAX` = 500  
최대 비례 게인 제한.
- float `GLOBAL_KD_MIN` = 0  
최소 미분 게인 제한.
- float `GLOBAL_KD_MAX` = 5  
최대 미분 게인 제한.
- float `GLOBAL_V_MIN`
- float `GLOBAL_V_MAX`
- float `GLOBAL_T_MIN`
- float `GLOBAL_T_MAX`  
속도 및 토크 제한.

**Private 멤버 함수**

- int `floatToUint` (float x, float x\_min, float x\_max, unsigned int bits)
- float `uintToFloat` (int x\_int, float x\_min, float x\_max, int bits)
- void `setMotorLimits` (`TMotor` &motor)

**4.11.1 상세한 설명**

`TMotor` 명령어를 파싱하는 클래스입니다.

`TMotor`에 대한 명령어 송수신 및 해석을 담당합니다. 이 클래스는 `TMotor` 명령 구성 및 응답 파싱을 위한 메서드를 제공합니다.

`CommandParser.hpp` 파일의 19 번째 라인에서 정의되었습니다.

**4.11.2 멤버 함수 문서화****4.11.2.1 floatToUint()**

```
int TMotorCommandParser::floatToUint (
    float x,
    float x_min,
    float x_max,
    unsigned int bits ) [private]
```

`CommandParser.cpp` 파일의 124 번째 라인에서 정의되었습니다.

```
00125 {
00126     float span = x_max - x_min;
00127     if (x < x_min)
00128         x = x_min;
00129     else if (x > x_max)
00130         x = x_max;
00131     return (int)((x - x_min) * ((float)((1 << bits) / span)));
00132 };
```

## 4.11.2.2 getCheck()

```
void TMotorCommandParser::getCheck (
    TMotor & motor,
    struct can_frame * frame )
```

pack ints into the can buffer ///

CommandParser.cpp 파일의 141 번째 라인에서 정의되었습니다.

```
00142 {
00143     // Set CAN frame id and data length code
00144     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00145     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00146
00148     frame->data[0] = 0x80;
00149     frame->data[1] = 0x00;
00150     frame->data[2] = 0x80;
00151     frame->data[3] = 0x00;
00152     frame->data[4] = 0x00;
00153     frame->data[5] = 0x00;
00154     frame->data[6] = 0x08;
00155     frame->data[7] = 0x00;
00156 }
```

## 4.11.2.3 getControlMode()

```
void TMotorCommandParser::getControlMode (
    TMotor & motor,
    struct can_frame * frame )
```

pack ints into the can buffer ///

CommandParser.cpp 파일의 158 번째 라인에서 정의되었습니다.

```
00159 {
00160     // Set CAN frame id and data length code
00161     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00162     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00163
00165     frame->data[0] = 0xFF;
00166     frame->data[1] = 0xFF;
00167     frame->data[2] = 0xFF;
00168     frame->data[3] = 0xFF;
00169     frame->data[4] = 0xFF;
00170     frame->data[5] = 0xFF;
00171     frame->data[6] = 0xFF;
00172     frame->data[7] = 0xFC;
00173 }
```

## 4.11.2.4 getExit()

```
void TMotorCommandParser::getExit (
    TMotor & motor,
    struct can_frame * frame )
```

pack ints into the can buffer ///

CommandParser.cpp 파일의 175 번째 라인에서 정의되었습니다.

```
00176 {
00177     // Set CAN frame id and data length code
00178     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00179     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00180
00182     frame->data[0] = 0xFF;
00183     frame->data[1] = 0xFF;
00184     frame->data[2] = 0xFF;
00185     frame->data[3] = 0xFF;
00186     frame->data[4] = 0xFF;
00187     frame->data[5] = 0xFF;
00188     frame->data[6] = 0xFF;
00189     frame->data[7] = 0xFD;
00190 }
```

#### 4.11.2.5 getQuickStop()

```
void TMotorCommandParser::getQuickStop (
    TMotor & motor,
    struct can_frame * frame )
```

pack ints into the can buffer ///

CommandParser.cpp 파일의 209 번째 라인에서 정의되었습니다.

```
00210 {
00211     // Set CAN frame id and data length code
00212     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00213     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00214
00216     frame->data[0] = 0x80;
00217     frame->data[1] = 0x00;
00218     frame->data[2] = 0x80;
00219     frame->data[3] = 0x00;
00220     frame->data[4] = 0x00;
00221     frame->data[5] = 0x00;
00222     frame->data[6] = 0x08;
00223     frame->data[7] = 0x00;
00224 }
```

#### 4.11.2.6 getZero()

```
void TMotorCommandParser::getZero (
    TMotor & motor,
    struct can_frame * frame )
```

pack ints into the can buffer ///

CommandParser.cpp 파일의 192 번째 라인에서 정의되었습니다.

```
00193 {
00194     // Set CAN frame id and data length code
00195     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00196     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00197
00199     frame->data[0] = 0xFF;
00200     frame->data[1] = 0xFF;
00201     frame->data[2] = 0xFF;
00202     frame->data[3] = 0xFF;
00203     frame->data[4] = 0xFF;
00204     frame->data[5] = 0xFF;
00205     frame->data[6] = 0xFF;
00206     frame->data[7] = 0xFE;
00207 }
```

#### 4.11.2.7 parseRecieveCommand()

```
std::tuple< int, float, float, float > TMotorCommandParser::parseRecieveCommand (
    TMotor & motor,
    struct can_frame * frame )
```

TMotor로부터 수신된 명령을 파싱합니다.

매개변수

motor	명령을 수신한 TMotor 객체.
frame	수신한 CAN 프레임.



반환값

명령 코드, 위치, 속도, 토크의 튜플.

unpack ints from can buffer ///

convert ints to floats ///

CommandParser.cpp 파일의 101 번째 라인에서 정의되었습니다.

```
00102 {
00103     int id;
00104     float position, speed, torque;
00105     setMotorLimits(motor);
00106     id = frame->data[0];
00107     int p_int = (frame->data[1] << 8) | frame->data[2];
00108     int v_int = (frame->data[3] << 4) | (frame->data[4] >> 4);
00109     int i_int = ((frame->data[4] & 0xF) << 8) | frame->data[5];
00110
00111     position = uintToFloat(p_int, GLOBAL_P_MIN, GLOBAL_P_MAX, 16);
00112     speed = uintToFloat(v_int, GLOBAL_V_MIN, GLOBAL_V_MAX, 12);
00113     torque = uintToFloat(i_int, GLOBAL_T_MIN, GLOBAL_T_MAX, 12);
00114
00115     motor.currentPos = position;
00116     motor.currentVel = speed;
00117     motor.currentTor = torque;
00118
00119     return std::make_tuple(id, position, speed, torque);
00120 }
00121
00122 }
```

#### 4.11.2.8 parseSendCommand()

```
void TMotorCommandParser::parseSendCommand (
    TMotor & motor,
    struct can_frame * frame,
    int canId,
    int dlc,
    float p_des,
    float v_des,
    float kp,
    float kd,
    float t_ff )
```

TMotor로 송신할 명령을 파싱합니다.

매개변수

motor	명령을 송신할 TMotor 객체.
frame	송신할 CAN 프레임.
canId	CAN ID.
dlc	데이터 길이 코드.
p_des	목표 위치.
v_des	목표 속도.
kp	비례 게인.
kd	미분 게인.
t_ff	피드포워드 토크.

pack ints into the can buffer ///

CommandParser.cpp 파일의 7 번째 라인에서 정의되었습니다.

```

00008 {
00009     // 모터 타입에 따른 제한값 설정
00010     setMotorLimits(motor);
00011
00012     // 기존 변수를 계산
00013     p_des = fminf(fmaxf(GLOBAL_P_MIN, p_des), GLOBAL_P_MAX);
00014     v_des = fminf(fmaxf(GLOBAL_V_MIN, v_des), GLOBAL_V_MAX);
00015     kp = fminf(fmaxf(GLOBAL_KP_MIN, kp), GLOBAL_KP_MAX);
00016     kd = fminf(fmaxf(GLOBAL_KD_MIN, kd), GLOBAL_KD_MAX);
00017     t_ff = fminf(fmaxf(GLOBAL_T_MIN, t_ff), GLOBAL_T_MAX);
00018
00019     motor.desPos = p_des;
00020     motor.desVel = v_des;
00021     motor.desTor = t_ff;
00022
00023     // 계산된 변수를 이용하여 unsigned int로 변환
00024     int p_int = floatToUint(p_des, GLOBAL_P_MIN, GLOBAL_P_MAX, 16);
00025     int v_int = floatToUint(v_des, GLOBAL_V_MIN, GLOBAL_V_MAX, 12);
00026     int kp_int = floatToUint(kp, GLOBAL_KP_MIN, GLOBAL_KP_MAX, 12);
00027     int kd_int = floatToUint(kd, GLOBAL_KD_MIN, GLOBAL_KD_MAX, 12);
00028     int t_int = floatToUint(t_ff, GLOBAL_T_MIN, GLOBAL_T_MAX, 12);
00029     // Set CAN frame id and data length code
00030     frame->can_id = canId & CAN_SFF_MASK; // Replace YOUR_CAN_ID with the appropriate id
00031     frame->can_dlc = dlc; // Data Length Code is set to maximum allowed length
00032
00033     frame->data[0] = p_int >> 8; // Position 8 higher
00034     frame->data[1] = p_int & 0xFF; // Position 8 lower
00035     frame->data[2] = v_int >> 4; // Speed 8 higher
00036     frame->data[3] = ((v_int & 0xF) << 4) | (kp_int >> 8); // Speed 4 bit lower KP 4bit higher
00037     frame->data[4] = kp_int & 0xFF; // KP 8 bit lower
00038     frame->data[5] = kd_int >> 4; // Kd 8 bit higher
00039     frame->data[6] = ((kd_int & 0xF) << 4) | (t_int >> 8); // KP 4 bit lower torque 4 bit higher
00040     frame->data[7] = t_int & 0xFF; // torque 4 bit lower
00041
00042 }

```

#### 4.11.2.9 setMotorLimits()

```

void TMotorCommandParser::setMotorLimits (
    TMotor & motor ) [private]

```

CommandParser.cpp 파일의 44 번째 라인에서 정의되었습니다.

```

00045 {
00046     if (motor.motorType == "AK10_9")
00047     {
00048         GLOBAL_V_MIN = -50;
00049         GLOBAL_V_MAX = 50;
00050         GLOBAL_T_MIN = -65;
00051         GLOBAL_T_MAX = 65;
00052     }
00053     else if (motor.motorType == "AK70_10")
00054     {
00055         GLOBAL_V_MIN = -50;
00056         GLOBAL_V_MAX = 50;
00057         GLOBAL_T_MIN = -25;
00058         GLOBAL_T_MAX = 25;
00059     }
00060     else if (motor.motorType == "AK60_6")
00061     {
00062         GLOBAL_V_MIN = -45;
00063         GLOBAL_V_MAX = 45;
00064         GLOBAL_T_MIN = -15;
00065         GLOBAL_T_MAX = 15;
00066     }
00067     else if (motor.motorType == "AK80_6")
00068     {
00069         GLOBAL_V_MIN = -76;
00070         GLOBAL_V_MAX = 76;
00071         GLOBAL_T_MIN = -12;
00072         GLOBAL_T_MAX = 12;
00073     }
00074     else if (motor.motorType == "AK80_9")
00075     {
00076         GLOBAL_V_MIN = -50;
00077         GLOBAL_V_MAX = 50;
00078         GLOBAL_T_MIN = -18;
00079         GLOBAL_T_MAX = 18;
00080     }
00081     else if (motor.motorType == "AK80_80" || motor.motorType == "AK80_64")
00082     {
00083         GLOBAL_V_MIN = -8;
00084         GLOBAL_V_MAX = 8;

```

```

00085         GLOBAL_T_MIN = -144;
00086         GLOBAL_T_MAX = 144;
00087     }
00088     else if (motor.motorType == "AK80_8")
00089     {
00090         GLOBAL_V_MIN = -37.5;
00091         GLOBAL_V_MAX = 37.5;
00092         GLOBAL_T_MIN = -32;
00093         GLOBAL_T_MAX = 32;
00094     }
00095     else
00096     {
00097         std::cout << "Error: Invalid motor motorType entered!" << std::endl;
00098     }
00099 }

```

#### 4.11.2.10 uintToFloat()

```

float TMotorCommandParser::uintToFloat (
    int x_int,
    float x_min,
    float x_max,
    int bits ) [private]

```

CommandParser.cpp 파일의 134 번째 라인에서 정의되었습니다.

```

00135 {
00136     float span = x_max - x_min;
00137     float offset = x_min;
00138     return ((float)x_int) * span / ((float)((1 << bits) - 1)) + offset;
00139 }

```

### 4.11.3 멤버 데이터 문서화

#### 4.11.3.1 GLOBAL\_KD\_MAX

```
float TMotorCommandParser::GLOBAL_KD_MAX = 5
```

최대 미분 게인 제한.

CommandParser.hpp 파일의 27 번째 라인에서 정의되었습니다.

#### 4.11.3.2 GLOBAL\_KD\_MIN

```
float TMotorCommandParser::GLOBAL_KD_MIN = 0
```

최소 미분 게인 제한.

CommandParser.hpp 파일의 26 번째 라인에서 정의되었습니다.

#### 4.11.3.3 GLOBAL\_KP\_MAX

```
float TMotorCommandParser::GLOBAL_KP_MAX = 500
```

최대 비례 게인 제한.

CommandParser.hpp 파일의 25 번째 라인에서 정의되었습니다.

#### 4.11.3.4 GLOBAL\_KP\_MIN

```
float TMotorCommandParser::GLOBAL_KP_MIN = 0
```

최소 비례 게인 제한.

[CommandParser.hpp](#) 파일의 24 번째 라인에서 정의되었습니다.

#### 4.11.3.5 GLOBAL\_P\_MAX

```
float TMotorCommandParser::GLOBAL_P_MAX = 12.5
```

최대 위치 제한.

[CommandParser.hpp](#) 파일의 23 번째 라인에서 정의되었습니다.

#### 4.11.3.6 GLOBAL\_P\_MIN

```
float TMotorCommandParser::GLOBAL_P_MIN = -12.5
```

최소 위치 제한.

[CommandParser.hpp](#) 파일의 22 번째 라인에서 정의되었습니다.

#### 4.11.3.7 GLOBAL\_T\_MAX

```
float TMotorCommandParser::GLOBAL_T_MAX
```

속도 및 토크 제한.

[CommandParser.hpp](#) 파일의 28 번째 라인에서 정의되었습니다.

#### 4.11.3.8 GLOBAL\_T\_MIN

```
float TMotorCommandParser::GLOBAL_T_MIN
```

[CommandParser.hpp](#) 파일의 28 번째 라인에서 정의되었습니다.

#### 4.11.3.9 GLOBAL\_V\_MAX

```
float TMotorCommandParser::GLOBAL_V_MAX
```

[CommandParser.hpp](#) 파일의 28 번째 라인에서 정의되었습니다.

#### 4.11.3.10 GLOBAL\_V\_MIN

```
float TMotorCommandParser::GLOBAL_V_MIN
```

[CommandParser.hpp](#) 파일의 28 번째 라인에서 정의되었습니다.

이 클래스에 대한 문서화 페이지는 다음의 파일들로부터 생성되었습니다.:

- `include/motors/CommandParser.hpp`
- `src/CommandParser.cpp`

## Chapter 5

# 파일 문서화

### 5.1 CanManager.hpp

```
00001 #ifndef CAN_SOCKET_UTILS_H
00002 #define CAN_SOCKET_UTILS_H
00003
00004 #include <linux/can.h>
00005 #include <math.h>
00006 #include <stdio.h>
00007 #include <termios.h>
00008 #include <fcntl.h>
00009 #include <unistd.h>
00010 #include <sys/socket.h>
00011 #include <time.h>
00012 #include <stdlib.h>
00013 #include <string.h>
00014 #include <linux/if.h>
00015 #include <sys/ioctl.h>
00016 #include <bits/types.h>
00017 #include <linux/can/raw.h>
00018 #include <sys/time.h>
00019 #include <vector>
00020 #include <string>
00021 #include <sstream>
00022 #include <iostream>
00023 #include <map>
00024 #include <queue>
00025 #include <memory>
00026 #include "Motor.hpp"
00027 #include "CommandParser.hpp"
00028
00029 using namespace std;
00030
00031 class CanManager
00032 {
00033 public:
00034     static const int ERR_SOCKET_CREATE_FAILURE = -1;
00035     static const int ERR_SOCKET_CONFIGURE_FAILURE = -2;
00036
00037     CanManager(std::map<std::string, std::shared_ptr<GenericMotor> &motorsRef);
00038     ~CanManager();
00039
00040     void initializeCAN();
00041
00042     void restartCanPorts();
00043
00044     void setSocketsTimeout(int sec, int usec);
00045
00046     void checkCanPortsStatus();
00047
00048     void setMotorsSocket();
00049
00050     bool checkConnection(std::shared_ptr<GenericMotor> motor);
00051
00052     bool checkAllMotors();
00053
00054     bool sendAndRecv(std::shared_ptr<GenericMotor> &motor, struct can_frame &frame);
00055
00056     bool sendFromBuff(std::shared_ptr<GenericMotor> &motor);
00057 }
```

```

00109
00116     bool recvToBuff(std::shared_ptr<GenericMotor> &motor, int readCount);
00117
00124     bool txFrame(std::shared_ptr<GenericMotor> &motor, struct can_frame &frame);
00125
00132     bool rxFrame(std::shared_ptr<GenericMotor> &motor, struct can_frame &frame);
00133
00137     void readFramesFromAllSockets();
00138
00142     void distributeFramesToMotors();
00143
00147     void clearReadBuffers();
00148
00149     std::map<std::string, int> sockets;
00150     std::map<std::string, bool> isConnected;
00151     int maxonCnt=0;
00152
00153 private:
00154     std::vector<std::string> ifnames;
00155     std::map<std::string, std::shared_ptr<GenericMotor> > &motors;
00156
00157     TMotorCommandParser tmotorcmd;
00158     MaxonCommandParser maxoncmd;
00159
00160     std::map<int, std::vector<can_frame> > tempFrames;
00161
00162     bool getCanPortStatus(const char *port);
00163     void activateCanPort(const char *port);
00164     void list_and_activate_available_can_ports();
00165     void deactivateCanPort(const char *port);
00166
00167     int createSocket(const std::string &ifname);
00168     int setSocketTimeout(int socket, int sec, int usec);
00169
00170     void clearCanBuffer(int canSocket);
00171 };
00172
00173 #endif // CAN_SOCKET_UTILS_H

```

## 5.2 HomeManager.hpp

```

00001 #pragma once
00002
00003 #include <stdio.h>
00004 #include "../include/managers/CanManager.hpp"
00005 #include "../include/motors/CommandParser.hpp"
00006 #include "../include/motors/Motor.hpp"
00007 #include "../include/tasks/SystemState.hpp"
00008 #include "../include/usbio/Sensor.hpp"
00009 #include <map>
00010 #include <memory>
00011 #include <string>
00012 #include <functional>
00013 #include <queue>
00014 #include <algorithm>
00015 #include <thread>
00016 #include <cerrno>
00017 #include <cstring>
00018 #include <sstream>
00019 #include <iomanip>
00020 #include <filesystem>
00021 #include <iostream>
00022 #include <vector>
00023 #include <limits>
00024 #include <ctime>
00025 #include <fstream>
00026 #include <atomic>
00027 #include <cmath>
00028 #include <chrono>
00029 #include <set>
00030
00031 using namespace std;
00032
00039 class HomeManager
00040 {
00041 public:
00048     HomeManager(SystemState &systemStateRef,
00049                 CanManager &canManagerRef,
00050                 std::map<std::string, std::shared_ptr<GenericMotor> > &motorsRef);
00051
00055     void mainLoop();
00056
00057 private:

```

```

00058     SystemState &systemState;
00059     CanManager &canManager;
00060     std::map<std::string, std::shared_ptr<GenericMotor>> &motors;
00061
00062     TMotorCommandParser tMotorcmd;
00063     MaxonCommandParser maxoncmd;
00064     Sensor sensor;
00065
00066     // 홈
00070     void displayHomingStatus();
00071
00075     void UpdateHomingStatus();
00076
00077     /* Tmotor */
00083     void SetTMotorHome(vector<std::shared_ptr<GenericMotor>> &motors, vector<std::string> &motorNames);
00084
00090     void HomeTMotor(vector<std::shared_ptr<GenericMotor>> &motors, vector<std::string> &motorNames);
00091
00099     vector<float> MoveTMotorToSensorLocation(vector<std::shared_ptr<GenericMotor>> &motors,
vector<std::string> &motorNames, vector<int> &sensorsBit);
00100
00109     void RotateTMotor(vector<std::shared_ptr<GenericMotor>> &motors, vector<std::string> &motorNames,
vector<double> &directions, vector<double> &degrees, vector<float> &midpoints);
00110
00116     bool PromptUserForHoming(const std::string &motorName);
00117
00118     /* Maxon */
00123     void SetMaxonHome(vector<std::shared_ptr<GenericMotor>> &motors);
00124
00129     void setMaxonMode(std::string targetMode);
00130
00134     void MaxonEnable();
00135
00140     void FixMotorPosition(std::shared_ptr<GenericMotor> &motor);
00141
00145     void MaxonDisable();
00146 };

```

## 5.3 PathManager.hpp

```

00001 #pragma once
00002
00003 #include <stdio.h>
00004 #include "../include/managers/CanManager.hpp"
00005 #include "../include/motors/CommandParser.hpp"
00006 #include "../include/motors/Motor.hpp"
00007
00008 #include "../include/tasks/SystemState.hpp"
00009
00010 #include <map>
00011 #include <memory>
00012 #include <string>
00013 #include <functional>
00014 #include <queue>
00015 #include <algorithm>
00016 #include <thread>
00017 #include <cerrno> // errno
00018 #include <cstring> // strerror
00019 #include <sstream>
00020 #include <iomanip>
00021 #include <filesystem>
00022 #include <iostream>
00023 #include <vector>
00024 #include <limits>
00025 #include <ctime>
00026 #include <fstream>
00027 #include <atomic>
00028 #include <cmath>
00029 #include <chrono>
00030 #include <set>
00031
00032 using namespace std;
00033
00034 class PathManager
00035 {
00036
00037 public:
00038     PathManager(SystemState &systemStateRef,
00039                 CanManager &canManagerRef,
00040                 std::map<std::string, std::shared_ptr<GenericMotor>> &motorsRef);
00041
00042     vector<double> fkfun();
00043     void ApplyDir();

```

```

00044 void GetDrumPositoin();
00045 void GetMusicSheet();
00046 void PathLoopTask();
00047 void GetArr(vector<double> &arr);
00048 void TestArr(double t, int cycles, int type, int LnR);
00049
00050 // 실시간X, 전체 악보 처리 후 연주
00051 int total = 0;
00052 int line = 0;
00053
00054 // 악보에 따른 position & velocity 값 저장 (5ms 단위)
00055 vector<vector<double>> p;
00056 vector<vector<double>> v;
00057
00058 // Ready Array : waist, R_arm1, L_arm1, R_arm2, R_arm3, L_arm2, L_arm3, R_wrist, L_wrist
00059 // : 0, 90, 90, 45, 75, 45, 75, 0, 0
00060 vector<double> standby = {0, M_PI / 2, M_PI / 2, M_PI / 4, M_PI / 2.4, M_PI / 4, M_PI / 2.4, 0,
0};
00061 vector<double> backarr = {0, M_PI / 2, M_PI / 2, 0, 0, 0, 0, M_PI / 3, M_PI / 3};
00062
00063 private:
00064 TMotorCommandParser TParser;
00065 MaxonCommandParser MParser;
00066
00067 SystemState &systemState;
00068 CanManager &canManager;
00069 std::map<std::string, std::shared_ptr<GenericMotor>> &motors;
00070
00071 // Functions for DrumRobot PathGenerating
00072 vector<double> c_MotorAngle = {0, 0, 0, 0, 0, 0, 0, 0, 0};
00073 vector<vector<double>> right_inst;
00074 vector<vector<double>> left_inst;
00075
00076 int n_inst = 10;
00077 double bpm = 10;
00078 vector<double> time_arr;
00079 vector<vector<int>> RA, LA;
00080 vector<int> RF, LF;
00081
00082 double p_R = 0; // 오른손 이전 악기 유무
00083 double p_L = 0; // 왼손 이전 악기 유무
00084 double c_R = 0; // 오른손 현재 악기 유무
00085 double c_L = 0; // 왼손 현재 악기 유무
00086
00087 double r_wrist = 0.0;
00088 double l_wrist = 0.0;
00089
00090 // 실측값
00091 vector<double> P1 = {0.3, 0.94344, 1.16582}; // RightArm Standby xyz
00092 vector<double> P2 = {-0.3, 0.94344, 1.16582}; // LeftArm Standby xyz
00093 vector<double> R = {0.363, 0.793, 0.363, 0.793}; // 오른팔 상완, 오른팔 하완+스틱, 왼팔 상완, 왼팔 하완+
스틱
00094 double s = 0.600;
00095 double z0 = 1.026;
00096
00097 /*// 임의값
00098 vector<double> P1 = {0.3, -1.0897, 0.0714}; // RightArm Standby xyz
00099 vector<double> P2 = {-0.3, -1.0897, 0.0714}; // LeftArm Standby xyz
00100 vector<double> R = {0.500, 0.850, 0.500, 0.850}; // 오른팔 상완, 오른팔 하완+스틱, 왼팔 상완, 왼팔 하완+스
틱
00101 double s = 0.600;
00102 double z0 = 0.000;
00103 */
00104
00105 // Q1, Q3 : 악기를 연주하기 전 들어올린 상태 / Q2 : 이번에 치는 악기 위치 / Q4 : 다음에 치는 악기 위치
00106 vector<double> Q1, Q2, Q3, Q4;
00107
00108 double ElbowAngle_ready = M_PI / 36; // -0.5일 때 들어올리는 팔꿈치 각도 : 5deg
00109 double ElbowAngle_hit = M_PI / 18; // -1일 때 들어올리는 팔꿈치 각도 : 10deg
00110 double WristAngle_ready = M_PI / 4; // -0.5일 때 들어올리는 손목 각도 : 45deg
00111 double WristAngle_hit = M_PI / 2; // -1일 때 들어올리는 손목 각도 : 90deg
00112
00113 // 각 악기별 치는 손목 각도
00114 vector<double> wrist = {0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0};
00115
00116 // 각 관절에 해당하는 열
00117 map<std::string, int> motor_mapping = {
00118 {"waist", 0}, {"R_arm1", 1}, {"L_arm1", 2}, {"R_arm2", 3}, {"R_arm3", 4}, {"L_arm2", 5},
{"L_arm3", 6}, {"R_wrist", 7}, {"L_wrist", 8}};
00119 // 각 열에 해당하는 관절방향
00120 map<int, int> motor_dir = { // 1 : CW, -1 : CCW
00121 {0, 1},
00122 {1, 1},
00123 {2, 1},
00124 {3, 1},
00125 {4, 1},
00126 {5, 1},

```



```

00127         {6, 1},
00128         {7, 1},
00129         {8, 1}};
00130
00131     vector<double> connect(vector<double> &Q1, vector<double> &Q2, int k, int n);
00132     void iconnect(vector<double> &P0, vector<double> &P1, vector<double> &P2, vector<double> &V0,
double t1, double t2, double t);
00133     vector<double> IKfun(vector<double> &P1, vector<double> &P2);
00134     void getDrummingPosAndAng();
00135     void getMotorPos();
00136     void getQ1AndQ2();
00137     void getQ3AndQ4();
00138     void Motors_sendBuffer();
00139 };

```

## 5.4 TestManager.hpp

```

00001 #pragma once
00002
00003 #include <stdio.h>
00004 #include "../include/managers/CanManager.hpp"
00005 #include "../include/motors/CommandParser.hpp"
00006 #include "../include/motors/Motor.hpp"
00007 #include "../include/tasks/SystemState.hpp"
00008 #include <map>
00009 #include <memory>
00010 #include <string>
00011 #include <functional>
00012 #include <queue>
00013 #include <algorithm>
00014 #include <thread>
00015 #include <cerrno> // errno
00016 #include <cstring> // strerror
00017 #include <sstream>
00018 #include <iomanip>
00019 #include <filesystem>
00020 #include <iostream>
00021 #include <vector>
00022 #include <limits>
00023 #include <ctime>
00024 #include <fstream>
00025 #include <atomic>
00026 #include <cmath>
00027 #include <chrono>
00028 #include <set>
00029
00030 using namespace std;
00031
00039 class TestManager
00040 {
00041 public:
00048     TestManager(SystemState &systemStateRef, CanManager &canManagerRef, std::map<std::string,
std::shared_ptr<GenericMotor> &motorsRef);
00049
00054     void mainLoop();
00055
00060     void multiTestLoop();
00061
00070     void TestArr(double t, int cycles, int type, int LnR, double amp[]);
00071
00072 private:
00073     SystemState &systemState;
00074     CanManager &canManager;
00075     std::map<std::string, std::shared_ptr<GenericMotor> &motors;
00076
00077     TMotorCommandParser tmotorcmd;
00078     MaxonCommandParser maxoncmd;
00079
00080     vector<string> InputData;
00081
00086     void move();
00087
00096     void mkArr(vector<string> &motorName, int time, int cycles, int LnR, double amp);
00097
00102     void SendLoop();
00103
00108     void parse_and_save_to_csv(const std::string &csv_file_name);
00109
00110     // Single Mode Test
00116     void FixMotorPosition(std::shared_ptr<GenericMotor> &motor);
00117
00129     void TuningTmotor(float kp, float kd, float sine_t, const std::string selectedMotor, int cycles,
float peakAngle, int pathType);

```

```

00130
00135     void TuningLoopTask();
00136
00150     void InitializeParameters(const std::string selectedMotor, float &kp, float &kd, float &peakAngle,
int &pathType, int &controlType, int &des_vel, int &des_tff, int &direction);
00151
00161     void TuningMaxonCSP(float sine_t, const std::string selectedMotor, int cycles, float peakAngle,
int pathType);
00162
00170     void TuningMaxonCSV(const std::string selectedMotor, int des_vel, int direction);
00171
00179     void TuningMaxonCST(const std::string selectedMotor, int des_tff, int direction);
00180
00186     void setMaxonMode(std::string targetMode);
00187
00193     int kbhit();
00194
00195     // Stick Mode Test
00200     void TestStickLoop();
00201
00211     void TestStick(const std::string selectedMotor, int des_tff, float tffThreshold, float
posThreshold, int backTorqueUnit);
00212
00220     bool dct_fun(float positions[], float vel_th);
00221
00222 };

```

## 5.5 CommandParser.hpp

```

00001 #ifndef COMMANDPARSER_H
00002 #define COMMANDPARSER_H
00003
00004 #include "Motor.hpp"
00005 #include <linux/can.h>
00006 #include <cmath>
00007 #include <tuple>
00008 #include <iostream>
00009
00010 using namespace std;
00011
00019 class TMotorCommandParser
00020 {
00021 public:
00022     float GLOBAL_P_MIN = -12.5;
00023     float GLOBAL_P_MAX = 12.5;
00024     float GLOBAL_KP_MIN = 0;
00025     float GLOBAL_KP_MAX = 500;
00026     float GLOBAL_KD_MIN = 0;
00027     float GLOBAL_KD_MAX = 5;
00028     float GLOBAL_V_MIN, GLOBAL_V_MAX, GLOBAL_T_MIN, GLOBAL_T_MAX;
00029
00042     void parseSendCommand(TMotor &motor, struct can_frame *frame, int canId, int dlc, float p_des,
float v_des, float kp, float kd, float t_ff);
00043
00050     std::tuple<int, float, float, float> parseRecieveCommand(TMotor &motor, struct can_frame *frame);
00051
00052     // 이하 메서드들은 TMotor의 특정 상태 또는 모드를 요청하는 명령을 구성합니다.
00053     // 각 메서드에 대한 자세한 설명은 생략하나, 실제 코드에는 해당 명령의 목적과 사용 방법에 대해 설명을 추가해야 합니다.
00054     void getCheck(TMotor &motor, struct can_frame *frame);
00055     void getControlMode(TMotor &motor, struct can_frame *frame);
00056     void getExit(TMotor &motor, struct can_frame *frame);
00057     void getZero(TMotor &motor, struct can_frame *frame);
00058     void getQuickStop(TMotor &motor, struct can_frame *frame);
00059
00060 private:
00061     // 내부적으로 사용되는 유틸리티 메서드들입니다. 이러한 메서드들은 클래스의 공개 인터페이스의 일부가 아니며, 주로 데이터 변
환에 사용됩니다.
00062     int floatToUInt(float x, float x_min, float x_max, unsigned int bits);
00063     float uintToFloat(int x_int, float x_min, float x_max, int bits);
00064
00065     void setMotorLimits(TMotor &motor);
00066 };
00067
00075 class MaxonCommandParser
00076 {
00077 public:
00078     // Maxon 모터로부터 수신된 명령을 파싱하는 메서드입니다. 반환된 튜플은 명령 코드, 위치, 속도를 포함합니다.
00079     std::tuple<int, float, float> parseRecieveCommand(MaxonMotor &motor, struct can_frame *frame);
00080
00081     // 이하 메서드들은 Maxon 모터의 특정 상태 또는 모드를 요청하는 명령을 구성합니다.
00082     // 각 메서드에 대한 자세한 설명은 생략하나, 실제 코드에는 해당 명령의 목적과 사용 방법에 대해 설명을 추가해야 합니다.
00083     void getCheck(MaxonMotor &motor, struct can_frame *frame);
00084     void getStop(MaxonMotor &motor, struct can_frame *frame);

```

```

00085 void getQuickStop(MaxonMotor &motor, struct can_frame *frame);
00086 void getOperational(MaxonMotor &motor, struct can_frame *frame);
00087 void getEnable(MaxonMotor &motor, struct can_frame *frame);
00088 void getSync(struct can_frame *frame);
00089
00090 // CSP 모드 관련 명령 구성 메서드.
00091 void getCSPMode(MaxonMotor &motor, struct can_frame *frame);
00092 void getTorqueOffset(MaxonMotor &motor, struct can_frame *frame);
00093 void getPosOffset(MaxonMotor &motor, struct can_frame *frame);
00094 void getTargetPosition(MaxonMotor &motor, struct can_frame *frame, float p_des_radians);
00095
00096 // HMM 모드 관련 명령 구성 메서드.
00097 void getHomeMode(MaxonMotor &motor, struct can_frame *frame);
00098 void getFlowingErrorWindow(MaxonMotor &motor, struct can_frame *frame);
00099 void getHomeoffsetDistance(MaxonMotor &motor, struct can_frame *frame, int degree);
00100 void getHomePosition(MaxonMotor &motor, struct can_frame *frame, int degree);
00101 void getHomingMethodL(MaxonMotor &motor, struct can_frame *frame);
00102 void getHomingMethodR(MaxonMotor &motor, struct can_frame *frame);
00103 void getStartHoming(MaxonMotor &motor, struct can_frame *frame);
00104 void getCurrentThreshold(MaxonMotor &motor, struct can_frame *frame);
00105
00106 // CSV 모드 관련 명령 구성 메서드.
00107 void getCSVMode(MaxonMotor &motor, struct can_frame *frame);
00108 void getVelOffset(MaxonMotor &motor, struct can_frame *frame);
00109 void getTargetVelocity(MaxonMotor &motor, struct can_frame *frame, int targetVelocity);
00110
00111 // CST 모드 관련 명령 구성 메서드.
00112 void getCSTMode(MaxonMotor &motor, struct can_frame *frame);
00113 void getTargetTorque(MaxonMotor &motor, struct can_frame *frame, int targetTorque);
00114 };
00115
00116 #endif // COMMANDPARSER_H

```

## 5.6 Motor.hpp

```

00001 #ifndef MOTOR_H
00002 #define MOTOR_H
00003
00004 #include <stdio.h>
00005 #include <stdlib.h>
00006 #include <string.h>
00007 #include <float.h>
00008 #include <stdlib.h>
00009 #include <array>
00010 #include <stdint>
00011 #include <string>
00012 #include <vector>
00013 #include <queue>
00014 #include <linux/can/raw.h>
00015 #include <iostream>
00016 using namespace std;
00017
00024 class GenericMotor
00025 {
00026 public:
00027     uint32_t nodeId;
00028     std::string interFaceName;
00029     float desPos, desVel, desTor;
00030     float currentPos, currentVel, currentTor;
00031     float cwDir;
00032     bool isHomed, isConnected;
00033     float rMin, rMax;
00034     int socket;
00035     int Kp;
00036     double Kd;
00037     std::queue<can_frame> sendBuffer;
00038     std::queue<can_frame> receiveBuffer;
00039
00040     GenericMotor(uint32_t nodeId, const std::string &interFaceName);
00041     virtual ~GenericMotor() = default;
00042
00043     void clearSendBuffer();
00044     void clearReceiveBuffer();
00045 };
00046
00053 class TMotor : public GenericMotor
00054 {
00055 public:
00056     TMotor(uint32_t nodeId, const std::string &motorType);
00057     std::string motorType;
00058
00059     int sensorBit;
00060

```

```

00061 private:
00062 };
00063
00070 class MaxonMotor : public GenericMotor
00071 {
00072 public:
00073     MaxonMotor(uint32_t nodeId);
00074
00075     uint32_t canSendId;
00076     uint32_t canReceiveId;
00077
00078     uint32_t txPdoIds[4];
00079     uint32_t rxPdoIds[4];
00080 };
00081
00082 #endif // MOTOR_H

```

## 5.7 DrumRobot.hpp

```

00001 #pragma once
00002
00003 #include <stdio.h>
00004 #include <map>
00005 #include <memory>
00006 #include <string>
00007 #include <functional>
00008 #include <queue>
00009 #include <algorithm>
00010 #include <thread>
00011 #include <cerrno> // errno
00012 #include <cstring> // strerror
00013 #include <sstream>
00014 #include <iomanip>
00015 #include <filesystem>
00016 #include <iostream>
00017 #include <vector>
00018 #include <limits>
00019 #include <ctime>
00020 #include <fstream>
00021 #include <atomic>
00022 #include <cmath>
00023 #include <chrono>
00024 #include <set>
00025
00026 #include "SystemState.hpp"
00027 #include "../include/usbio/Sensor.hpp"
00028 #include "../include/managers/CanManager.hpp"
00029 #include "../include/managers/PathManager.hpp"
00030 #include "../include/motors/CommandParser.hpp"
00031 #include "../include/motors/Motor.hpp"
00032 #include "../include/managers/TestManager.hpp"
00033 #include "../include/managers/HomeManager.hpp"
00034
00035 using namespace std;
00036
00043 class DrumRobot
00044 {
00045 public:
00055     DrumRobot(SystemState &systemStateRef,
00056               CanManager &canManagerRef,
00057               PathManager &pathManagerRef,
00058               HomeManager &homeManagerRef,
00059               TestManager &testManagerRef,
00060               std::map<std::string, std::shared_ptr<GenericMotor>> &motorsRef);
00061
00062     void stateMachine();
00063     void sendLoopForThread();
00064     void recvLoopForThread();
00065
00066 private:
00067     SystemState &systemState;
00068     CanManager &canManager;
00069     PathManager &pathManager;
00070     HomeManager &homeManager;
00071     TestManager &testManager;
00072     std::map<std::string, std::shared_ptr<GenericMotor>> &motors;
00073
00074     TMotorCommandParser tmotorcmd;
00075     MaxonCommandParser maxoncmd;
00076     Sensor sensor;
00077
00078     // State Utility 메소드들
00079     void displayAvailableCommands() const;

```

```

00080     bool processInput(const std::string &input);
00081     void idealStateRoutine();
00082     void checkUserInput();
00083     void printCurrentPositions();
00084     int kbhit();
00085
00086     bool isReady;
00087     bool isBack;
00088
00089     // System Initialize 메소드들
00090     void initializeMotors();
00091     void initializeCanManager();
00092     void DeactivateControlTask();
00093     void motorSettingCmd();
00094     void setMaxonMode(std::string targetMode);
00095     void MaxonEnable();
00096     void MaxonDisable();
00097
00098     // Send Thread Loop 메소드들
00099     void SendLoop();
00100     void save_to_txt_inputData(const string &csv_file_name);
00101     void SendReadyLoop();
00102     int writeFailCount;
00103     void initializePathManager();
00104     void clearMotorsSendBuffer();
00105
00106     // Receive Thread Loop 메소드들
00107     const int NUM_FRAMES;
00108     const int TIME_THRESHOLD_MS;
00109
00110     void RecieveLoop();
00111     void parse_and_save_to_csv(const std::string &csv_file_name);
00112 };

```

## 5.8 SystemState.hpp

```

00001 #pragma once
00002
00003 #include <atomic>
00004
00009 enum class Main
00010 {
00011     SystemInit,
00012     Ideal,
00013     Homing,
00014     Tune,
00015     Perform,
00016     Check,
00017     Shutdown,
00018     Ready,
00019     Back,
00020     Pause
00021 };
00022
00027 enum class HomeMode
00028 {
00029     NotHome,
00030     HomeDone,
00031     HomeError
00032 };
00033
00041 struct SystemState
00042 {
00043     std::atomic<Main> main;
00044     std::atomic<HomeMode> homeMode;
00045
00051     SystemState() : main(Main::SystemInit),
00052                   homeMode(HomeMode::NotHome) {}
00053 };

```

## 5.9 CanManager.cpp

```

00001 #include "../include/managers/CanManager.hpp"
00002 CanManager::CanManager(std::map<std::string, std::shared_ptr<GenericMotor>> &motorsRef)
00003     : motors(motorsRef)
00004 {
00005 }
00006

```

```

00007 CanManager::~CanManager()
00008 {
00009     // 모든 소켓 닫기
00010     for (const auto &socketPair : sockets)
00011     {
00012         if (socketPair.second >= 0)
00013         {
00014             close(socketPair.second);
00015         }
00016     }
00017     sockets.clear();
00018 }
00019
00020 /*                                     Settign Functions [Public]                                     */
00021
00022 void CanManager::initializeCAN()
00023 {
00024     list_and_activate_available_can_ports();
00025     for (const auto &ifname : this->ifnames)
00026     {
00027         std::cout << "Processing interface: " << ifname << std::endl;
00028         int hsocket = createSocket(ifname);
00029         if (hsocket < 0)
00030         {
00031             std::cerr << "Socket creation error for interface: " << ifname << std::endl;
00032             exit(EXIT_FAILURE);
00033         }
00034         sockets[ifname] = hsocket;
00035         isConnected[ifname] = true;
00036         std::cout << "Socket created for " << ifname << ": " << hsocket << std::endl;
00037     }
00038 }
00039
00040 void CanManager::restartCanPorts()
00041 {
00042     // 먼저 모든 포트를 down 시킵니다.
00043     for (const auto &port : ifnames)
00044     {
00045         deactivateCanPort(port.c_str());
00046
00047         int socket_fd = sockets[port];
00048         if (socket_fd >= 0)
00049         {
00050             close(socket_fd); // 기존 소켓을 닫습니다.
00051             sockets[port] = -1; // 소켓 디스크립터 값을 초기화합니다.
00052         }
00053     }
00054
00055     // 각 포트에 대해 새로운 소켓을 생성하고 디스크립터를 업데이트합니다.
00056     for (const auto &port : ifnames)
00057     {
00058         usleep(100000); // 100ms 대기
00059         activateCanPort(port.c_str());
00060
00061         int new_socket_fd = createSocket(port);
00062         if (new_socket_fd < 0)
00063         {
00064             // 새로운 소켓 생성에 실패한 경우 처리
00065             fprintf(stderr, "Failed to create a new socket for port: %s\n", port.c_str());
00066         }
00067         else
00068         {
00069             sockets[port] = new_socket_fd; // 소켓 디스크립터 값을 업데이트합니다.
00070         }
00071     }
00072 }
00073
00074 void CanManager::setSocketsTimeout(int sec, int usec)
00075 {
00076     for (const auto &socketPair : sockets)
00077     {
00078         int socket_fd = socketPair.second;
00079         if (setSocketTimeout(socket_fd, sec, usec) != 0)
00080         {
00081             std::cerr << "Failed to set socket timeout for " << socketPair.first << std::endl;
00082         }
00083     }
00084 }
00085
00086 void CanManager::checkCanPortsStatus()
00087 {
00088     for (const auto &ifname : this->ifnames)
00089     {
00090
00091
00092
00093
00094
00095

```

```

00096         isConnected[ifname] = getCanPortStatus(ifname.c_str());
00097
00098         if (!isConnected[ifname])
00099         {
00100             std::cout << "Port " << ifname << " is NOT CONNECTED" << std::endl;
00101         }
00102     }
00103
00104     // 모든 포트가 연결된 경우 1, 아니면 0 반환
00105 }
00106
00107 /*                                     Settign Functions [Private]                                     */
00108
00109 bool CanManager::getCanPortStatus(const char *port)
00110 {
00111     char command[50];
00112     snprintf(command, sizeof(command), "ip link show %s", port);
00113
00114     FILE *fp = popen(command, "r");
00115     if (fp == NULL)
00116     {
00117         perror("Error opening pipe");
00118         return false;
00119     }
00120
00121     char output[1024];
00122     while (fgets(output, sizeof(output) - 1, fp) != NULL)
00123     {
00124         if (strstr(output, "DOWN") || strstr(output, "does not exist"))
00125         {
00126             pclose(fp);
00127             return false;
00128         }
00129         else if (strstr(output, "UP"))
00130         {
00131             pclose(fp);
00132             return true;
00133         }
00134     }
00135
00136     perror("fgets failed");
00137     printf("Errno: %d\n", errno); // errno 값을 출력
00138     pclose(fp);
00139     return false;
00140 }
00141
00142 int CanManager::createSocket(const std::string &ifname)
00143 {
00144     int result;
00145     struct sockaddr_can addr;
00146     struct ifreq ifr;
00147
00148     int localSocket = socket(PF_CAN, SOCK_RAW, CAN_RAW); // 지역 변수로 소켓 생성
00149     if (localSocket < 0)
00150     {
00151         return ERR_SOCKET_CREATE_FAILURE;
00152     }
00153
00154     memset(&ifr, 0, sizeof(struct ifreq));
00155     memset(&addr, 0, sizeof(struct sockaddr_can));
00156
00157     strcpy(ifr.ifr_name, ifname.c_str());
00158     result = ioctl(localSocket, SIOCGIFINDEX, &ifr);
00159     if (result < 0)
00160     {
00161         close(localSocket);
00162         return ERR_SOCKET_CREATE_FAILURE;
00163     }
00164
00165     addr.can_ifindex = ifr.ifr_ifindex;
00166     addr.can_family = AF_CAN;
00167
00168     if (bind(localSocket, (struct sockaddr *)&addr, sizeof(addr)) < 0)
00169     {
00170         close(localSocket);
00171         return ERR_SOCKET_CREATE_FAILURE;
00172     }
00173
00174     return localSocket; // 생성된 소켓 디스크립터 반환
00175 }
00176
00177 void CanManager::activateCanPort(const char *port)
00178 {
00179     char command1[100], command2[100];
00180     snprintf(command1, sizeof(command1), "sudo ip link set %s type can bitrate 1000000 sample-point 0.850", port);
00181     snprintf(command2, sizeof(command2), "sudo ip link set %s up", port);
00182 }

```

```

00184
00185     int ret1 = system(command1);
00186     int ret2 = system(command2);
00187
00188     if (ret1 != 0 || ret2 != 0)
00189     {
00190         fprintf(stderr, "Failed to activate port: %s\n", port);
00191         exit(1); // 또는 다른 에러 처리
00192     }
00193 }
00194
00195 void CanManager::list_and_activate_available_can_ports()
00196 {
00197     int portCount = 0; // CAN 포트 수를 세기 위한 변수
00198
00199     FILE *fp = popen("ip link show | grep can", "r");
00200     if (fp == nullptr)
00201     {
00202         perror("No available CAN port");
00203         exit(1);
00204     }
00205
00206     char output[1024];
00207     while (fgets(output, sizeof(output) - 1, fp) != nullptr)
00208     {
00209         std::string line(output);
00210         std::istringstream iss(line);
00211         std::string skip, port;
00212         iss » skip » port;
00213
00214         // 콜론 제거
00215         if (!port.empty() && port.back() == ':')
00216         {
00217             port.pop_back();
00218         }
00219
00220         // 포트 이름이 유효한지 확인
00221         if (!port.empty() && port.find("can") == 0)
00222         {
00223             portCount++;
00224             if (!getCanPortStatus(port.c_str()))
00225             {
00226                 printf("%s is DOWN, activating...\n", port.c_str());
00227                 activateCanPort(port.c_str());
00228             }
00229             else
00230             {
00231                 printf("%s is already UP\n", port.c_str());
00232             }
00233
00234             this->ifnames.push_back(port); // 포트 이름을 ifnames 벡터에 추가
00235         }
00236     }
00237
00238     if (feof(fp) == 0)
00239     {
00240         perror("fgets failed");
00241         printf("Errno: %d\n", errno);
00242     }
00243
00244     pclose(fp);
00245
00246     if (portCount == 0)
00247     {
00248         printf("No CAN port found. Exiting...\n");
00249         exit(1);
00250     }
00251 }
00252
00253 void CanManager::deactivateCanPort(const char *port)
00254 {
00255     char command[100];
00256     snprintf(command, sizeof(command), "sudo ip link set %s down", port);
00257     int ret = system(command);
00258     if (ret != 0)
00259     {
00260         fprintf(stderr, "Failed to down port: %s\n", port);
00261     }
00262 }
00263
00264 void CanManager::clearReadBuffers()
00265 {
00266     for (const auto &socketPair : sockets)
00267     {
00268         int socket_fd = socketPair.second;
00269         clearCanBuffer(socket_fd);
00270     }

```



```

00271 }
00272
00273 void CanManager::clearCanBuffer(int canSocket)
00274 {
00275     struct can_frame frame;
00276     fd_set readSet;
00277     struct timeval timeout;
00278
00279     // 수신 대기 시간 설정
00280     timeout.tv_sec = 0;
00281     timeout.tv_usec = 0; // 즉시 반환
00282
00283     while (true)
00284     {
00285         FD_ZERO(&readSet);
00286         FD_SET(canSocket, &readSet);
00287
00288         // 소켓에서 읽을 데이터가 있는지 확인
00289         int selectRes = select(canSocket + 1, &readSet, NULL, NULL, &timeout);
00290
00291         if (selectRes > 0)
00292         {
00293             // 수신 버퍼에서 데이터 읽기
00294             ssize_t nbytes = read(canSocket, &frame, sizeof(struct can_frame));
00295
00296             if (nbytes <= 0)
00297             {
00298                 // 읽기 실패하거나 더 이상 읽을 데이터가 없음
00299                 break;
00300             }
00301         }
00302         else
00303         {
00304             // 읽을 데이터가 없음
00305             break;
00306         }
00307     }
00308 }
00309
00310 int CanManager::setSocketTimeout(int socket, int sec, int usec)
00311 {
00312     struct timeval timeout;
00313     timeout.tv_sec = sec;
00314     timeout.tv_usec = usec;
00315
00316     if (setsockopt(socket, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout, sizeof(timeout)) < 0)
00317     {
00318         perror("setsockopt failed");
00319         return -1;
00320     }
00321
00322     return 0;
00323 }
00324
00326 /*                      Utility Functions                      */
00328
00329 bool CanManager::txFrame(std::shared_ptr<GenericMotor> &motor, struct can_frame &frame)
00330 {
00331     if (write(motor->socket, &frame, sizeof(frame)) != sizeof(frame))
00332     {
00333         // perror("CAN write error");
00334         return false;
00335     }
00336     return true;
00337 }
00338
00339 bool CanManager::rxFrame(std::shared_ptr<GenericMotor> &motor, struct can_frame &frame)
00340 {
00341     if (read(motor->socket, &frame, sizeof(frame)) != sizeof(frame))
00342     {
00343         // perror("CAN read error");
00344         return false;
00345     }
00346     return true;
00347 }
00348
00349
00350 bool CanManager::sendAndRecv(std::shared_ptr<GenericMotor> &motor, struct can_frame &frame)
00351 {
00352     if (!txFrame(motor, frame) || !rxFrame(motor, frame))
00353     {
00354         // perror("Send and receive error");
00355         return false;
00356     }
00357     return true;
00358 }
00359

```

```

00360 bool CanManager::sendFromBuff(std::shared_ptr<GenericMotor> &motor)
00361 {
00362     if (!motor->sendBuffer.empty())
00363     {
00364         struct can_frame frame = motor->sendBuffer.front();
00365         motor->sendBuffer.pop();
00366         return txFrame(motor, frame);
00367     }
00368     return false;
00369 }
00370
00371 bool CanManager::recvToBuff(std::shared_ptr<GenericMotor> &motor, int readCount)
00372 {
00373     struct can_frame frame;
00374     for (int i = 0; i < readCount; i++)
00375     {
00376         if (rxFrame(motor, frame))
00377         {
00378             motor->recieveBuffer.push(frame);
00379         }
00380         else
00381         {
00382             return false;
00383         }
00384     }
00385     return true;
00386 }
00387
00388 void CanManager::setMotorsSocket()
00389 {
00390     struct can_frame frame;
00391     setSocketsTimeout(0, 10000);
00392
00393     // 모든 소켓에 대해 각 모터에 명령을 보내고 응답을 확인
00394     for (const auto &socketPair : sockets)
00395     {
00396         int socket_fd = socketPair.second;
00397
00398         for (auto &motor_pair : motors)
00399         {
00400             auto &motor = motor_pair.second;
00401             clearReadBuffers();
00402
00403             // TMotor 및 MaxonMotor에 대해 적절한 명령 설정
00404             if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00405             {
00406                 tMotorcmd.getCheck(*tMotor, &frame);
00407             }
00408             else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor))
00409             {
00410                 maxoncmd.getCheck(*maxonMotor, &frame);
00411             }
00412             usleep(10000);
00413             // 모터의 현재 소켓을 임시 소켓으로 설정
00414             int original_socket = motor->socket;
00415             motor->socket = socket_fd;
00416             usleep(10000);
00417             // 소켓에 CAN 프레임 보내고 응답 확인
00418             if (sendAndRecv(motor, frame))
00419             {
00420                 motor->isConected = true;
00421             }
00422             else
00423             {
00424                 motor->socket = original_socket;
00425             }
00426         }
00427     }
00428
00429     // 모든 소켓에 대한 검사가 완료된 후, 모터 연결 상태 확인 및 삭제
00430     for (auto it = motors.begin(); it != motors.end(); )
00431     {
00432         std::string name = it->first;
00433         std::shared_ptr<GenericMotor> motor = it->second;
00434         if (motor->isConected)
00435         {
00436             std::cerr << "-----> Motor [" << name << "] is Connected." << std::endl;
00437             ++it;
00438         }
00439         else
00440         {
00441             std::cerr << "Motor [" << name << "] Not Connected." << std::endl;
00442             it = motors.erase(it);
00443         }
00444     }
00445 }

```

```

00446     for (auto &motor_pair : motors)
00447     {
00448         if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00449         {
00450             maxonCnt++;
00451         }
00452     }
00453 }
00454
00455 void CanManager::readFramesFromAllSockets()
00456 {
00457     struct can_frame frame;
00458     for (const auto &socketPair : sockets)
00459     {
00460         int socket_fd = socketPair.second;
00461         while (read(socket_fd, &frame, sizeof(frame)) == sizeof(frame))
00462         {
00463             tempFrames[socket_fd].push_back(frame);
00464         }
00465     }
00466 }
00467
00468 void CanManager::distributeFramesToMotors()
00469 {
00470     for (auto &motor_pair : motors)
00471     {
00472         auto &motor = motor_pair.second;
00473
00474         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00475         {
00476             // TMotor 처리
00477             for (auto &frame : tempFrames[motor->socket])
00478             {
00479                 if (frame.data[0] == tMotor->nodeId)
00480                 {
00481                     std::tuple<int, float, float, float> parsedData =
tmotorcmd.parseRecieveCommand(*tMotor, &frame);
00482                     tMotor->currentPos = std::get<1>(parsedData);
00483                     tMotor->currentVel = std::get<2>(parsedData);
00484                     tMotor->currentTor = std::get<3>(parsedData);
00485                     tMotor->recieveBuffer.push(frame);
00486                 }
00487             }
00488         }
00489         else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor))
00490         {
00491             // MaxonMotor 처리
00492             for (auto &frame : tempFrames[motor->socket])
00493             {
00494                 if (frame.can_id == maxonMotor->txPdoIds[0])
00495                 {
00496                     std::tuple<int, float, float> parsedData =
maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
00497                     maxonMotor->currentPos = std::get<1>(parsedData);
00498                     maxonMotor->currentTor = std::get<2>(parsedData);
00499                     maxonMotor->recieveBuffer.push(frame);
00500                 }
00501             }
00502         }
00503     }
00504     tempFrames.clear(); // 프레임 분배 후 임시 배열 비우기
00505 }
00506
00507 bool CanManager::checkConnection(std::shared_ptr<GenericMotor> motor)
00508 {
00509     struct can_frame frame;
00510     setSocketsTimeout(0, 5000 /*5ms*/);
00511     clearReadBuffers();
00512
00513     if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00514     {
00515         tmotorcmd.getControlMode(*tMotor, &frame);
00516         if (sendAndRecv(motor, frame))
00517         {
00518             std::tuple<int, float, float, float> parsedData = tmotorcmd.parseRecieveCommand(*tMotor,
&frame);
00519             motor->currentPos = std::get<1>(parsedData);
00520             motor->currentVel = std::get<2>(parsedData);
00521             motor->currentTor = std::get<3>(parsedData);
00522             motor->isConected = true;
00523         }
00524         else
00525         {
00526             return false;
00527         }
00528     }

```

```

00528     }
00529     else if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00530     {
00531         maxoncmd.getSync(&frame);
00532         txFrame(motor, frame);
00533         motor->clearReceiveBuffer();
00534         if (recvToBuff(motor, maxonCnt))
00535         {
00536             while (!motor->recieveBuffer.empty())
00537             {
00538                 frame = motor->recieveBuffer.front();
00539                 if (frame.can_id == maxonMotor->rxPdoIds[0])
00540                 {
00541                     std::tuple<int, float, float> parsedData =
maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
00542                     motor->currentPos = std::get<1>(parsedData);
00543                     motor->currentTor = std::get<2>(parsedData);
00544                     motor->isConected = true;
00545                 }
00546                 motor->recieveBuffer.pop();
00547             }
00548         }
00549         else
00550         {
00551             return false;
00552         }
00553     }
00554     return true;
00555 }
00556
00557 bool CanManager::checkAllMotors()
00558 {
00559     bool allMotorsChecked = true;
00560     for (auto &motorPair : motors)
00561     {
00562         std::string name = motorPair.first;
00563         auto &motor = motorPair.second;
00564
00565         if (!checkConnection(motor))
00566         {
00567             allMotorsChecked = false;
00568         }
00569     }
00570     return allMotorsChecked;
00571 }

```

## 5.10 CommandParser.cpp

```

00001 #include "../include/motors/CommandParser.hpp"
00002
00003 /*
00004                                     Tmotor Parser definition
00005 */
00006
00007 void TMotorCommandParser::parseSendCommand(TMotor &motor, struct can_frame *frame, int canId, int dlc,
float p_des, float v_des, float kp, float kd, float t_ff)
00008 {
00009     // 모터 타입에 따른 제한값 설정
00010     setMotorLimits(motor);
00011
00012     // 기존 변수를 계산
00013     p_des = fminf(fmaxf(GLOBAL_P_MIN, p_des), GLOBAL_P_MAX);
00014     v_des = fminf(fmaxf(GLOBAL_V_MIN, v_des), GLOBAL_V_MAX);
00015     kp = fminf(fmaxf(GLOBAL_KP_MIN, kp), GLOBAL_KP_MAX);
00016     kd = fminf(fmaxf(GLOBAL_KD_MIN, kd), GLOBAL_KD_MAX);
00017     t_ff = fminf(fmaxf(GLOBAL_T_MIN, t_ff), GLOBAL_T_MAX);
00018
00019     motor.desPos = p_des;
00020     motor.desVel = v_des;
00021     motor.desTor = t_ff;
00022
00023     // 계산된 변수를 이용하여 unsigned int로 변환
00024     int p_int = floatToUInt(p_des, GLOBAL_P_MIN, GLOBAL_P_MAX, 16);
00025     int v_int = floatToUInt(v_des, GLOBAL_V_MIN, GLOBAL_V_MAX, 12);
00026     int kp_int = floatToUInt(kp, GLOBAL_KP_MIN, GLOBAL_KP_MAX, 12);
00027     int kd_int = floatToUInt(kd, GLOBAL_KD_MIN, GLOBAL_KD_MAX, 12);
00028     int t_int = floatToUInt(t_ff, GLOBAL_T_MIN, GLOBAL_T_MAX, 12);
00029     // Set CAN frame id and data length code
00030     frame->can_id = canId & CAN_SFF_MASK; // Replace YOUR_CAN_ID with the appropriate id
00031     frame->can_dlc = dlc; // Data Length Code is set to maximum allowed length
00032
00033     frame->data[0] = p_int >> 8; // Position 8 higher
00034     frame->data[1] = p_int & 0xFF; // Position 8 lower
00035     frame->data[2] = v_int >> 4; // Speed 8 higher

```

```

00037     frame->data[3] = ((v_int & 0xF) << 4) | (kp_int >> 8); // Speed 4 bit lower KP 4bit higher
00038     frame->data[4] = kp_int & 0xFF; // KP 8 bit lower
00039     frame->data[5] = kd_int >> 4; // Kd 8 bit higher
00040     frame->data[6] = ((kd_int & 0xF) << 4) | (t_int >> 8); // KP 4 bit lower torque 4 bit higher
00041     frame->data[7] = t_int & 0xFF; // torque 4 bit lower
00042 }
00043
00044 void TMotorCommandParser::setMotorLimits(TMotor &motor)
00045 {
00046     if (motor.motorType == "AK10_9")
00047     {
00048         GLOBAL_V_MIN = -50;
00049         GLOBAL_V_MAX = 50;
00050         GLOBAL_T_MIN = -65;
00051         GLOBAL_T_MAX = 65;
00052     }
00053     else if (motor.motorType == "AK70_10")
00054     {
00055         GLOBAL_V_MIN = -50;
00056         GLOBAL_V_MAX = 50;
00057         GLOBAL_T_MIN = -25;
00058         GLOBAL_T_MAX = 25;
00059     }
00060     else if (motor.motorType == "AK60_6")
00061     {
00062         GLOBAL_V_MIN = -45;
00063         GLOBAL_V_MAX = 45;
00064         GLOBAL_T_MIN = -15;
00065         GLOBAL_T_MAX = 15;
00066     }
00067     else if (motor.motorType == "AK80_6")
00068     {
00069         GLOBAL_V_MIN = -76;
00070         GLOBAL_V_MAX = 76;
00071         GLOBAL_T_MIN = -12;
00072         GLOBAL_T_MAX = 12;
00073     }
00074     else if (motor.motorType == "AK80_9")
00075     {
00076         GLOBAL_V_MIN = -50;
00077         GLOBAL_V_MAX = 50;
00078         GLOBAL_T_MIN = -18;
00079         GLOBAL_T_MAX = 18;
00080     }
00081     else if (motor.motorType == "AK80_80" || motor.motorType == "AK80_64")
00082     {
00083         GLOBAL_V_MIN = -8;
00084         GLOBAL_V_MAX = 8;
00085         GLOBAL_T_MIN = -144;
00086         GLOBAL_T_MAX = 144;
00087     }
00088     else if (motor.motorType == "AK80_8")
00089     {
00090         GLOBAL_V_MIN = -37.5;
00091         GLOBAL_V_MAX = 37.5;
00092         GLOBAL_T_MIN = -32;
00093         GLOBAL_T_MAX = 32;
00094     }
00095     else
00096     {
00097         std::cout << "Error: Invalid motor motorType entered!" << std::endl;
00098     }
00099 }
00100
00101 std::tuple<int, float, float, float> TMotorCommandParser::parseRecieveCommand(TMotor &motor, struct
can_frame *frame)
00102 {
00103     int id;
00104     float position, speed, torque;
00105     setMotorLimits(motor);
00106     id = frame->data[0];
00107     int p_int = (frame->data[1] << 8) | frame->data[2];
00108     int v_int = (frame->data[3] << 4) | (frame->data[4] >> 4);
00109     int i_int = ((frame->data[4] & 0xF) << 8) | frame->data[5];
00110
00111     position = uintToFloat(p_int, GLOBAL_P_MIN, GLOBAL_P_MAX, 16);
00112     speed = uintToFloat(v_int, GLOBAL_V_MIN, GLOBAL_V_MAX, 12);
00113     torque = uintToFloat(i_int, GLOBAL_T_MIN, GLOBAL_T_MAX, 12);
00114
00115     motor.currentPos = position;
00116     motor.currentVel = speed;
00117     motor.currentTor = torque;
00118
00119     return std::make_tuple(id, position, speed, torque);
00120 }
00121
00122 int TMotorCommandParser::floatToUint(float x, float x_min, float x_max, unsigned int bits)

```

```

00125 {
00126     float span = x_max - x_min;
00127     if (x < x_min)
00128         x = x_min;
00129     else if (x > x_max)
00130         x = x_max;
00131     return (int)((x - x_min) * ((float)((1 << bits) / span)));
00132 };
00133
00134 float TMotorCommandParser::uintToFloat(int x_int, float x_min, float x_max, int bits)
00135 {
00136     float span = x_max - x_min;
00137     float offset = x_min;
00138     return ((float)x_int) * span / ((float)((1 << bits) - 1)) + offset;
00139 }
00140
00141 void TMotorCommandParser::getCheck(TMotor &motor, struct can_frame *frame)
00142 {
00143     // Set CAN frame id and data length code
00144     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00145     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00146
00147     frame->data[0] = 0x80;
00148     frame->data[1] = 0x00;
00149     frame->data[2] = 0x80;
00150     frame->data[3] = 0x00;
00151     frame->data[4] = 0x00;
00152     frame->data[5] = 0x00;
00153     frame->data[6] = 0x08;
00154     frame->data[7] = 0x00;
00155 }
00156
00157
00158 void TMotorCommandParser::getControlMode(TMotor &motor, struct can_frame *frame)
00159 {
00160     // Set CAN frame id and data length code
00161     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00162     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00163
00164     frame->data[0] = 0xFF;
00165     frame->data[1] = 0xFF;
00166     frame->data[2] = 0xFF;
00167     frame->data[3] = 0xFF;
00168     frame->data[4] = 0xFF;
00169     frame->data[5] = 0xFF;
00170     frame->data[6] = 0xFF;
00171     frame->data[7] = 0xFC;
00172 }
00173
00174
00175 void TMotorCommandParser::getExit(TMotor &motor, struct can_frame *frame)
00176 {
00177     // Set CAN frame id and data length code
00178     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00179     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00180
00181     frame->data[0] = 0xFF;
00182     frame->data[1] = 0xFF;
00183     frame->data[2] = 0xFF;
00184     frame->data[3] = 0xFF;
00185     frame->data[4] = 0xFF;
00186     frame->data[5] = 0xFF;
00187     frame->data[6] = 0xFF;
00188     frame->data[7] = 0xFD;
00189 }
00190
00191
00192 void TMotorCommandParser::getZero(TMotor &motor, struct can_frame *frame)
00193 {
00194     // Set CAN frame id and data length code
00195     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00196     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00197
00198     frame->data[0] = 0xFF;
00199     frame->data[1] = 0xFF;
00200     frame->data[2] = 0xFF;
00201     frame->data[3] = 0xFF;
00202     frame->data[4] = 0xFF;
00203     frame->data[5] = 0xFF;
00204     frame->data[6] = 0xFF;
00205     frame->data[7] = 0xFE;
00206 }
00207
00208
00209 void TMotorCommandParser::getQuickStop(TMotor &motor, struct can_frame *frame)
00210 {
00211     // Set CAN frame id and data length code
00212     frame->can_id = motor.nodeId; // Replace YOUR_CAN_ID with the appropriate id
00213     frame->can_dlc = 8;           // Data Length Code is set to maximum allowed length
00214
00215     frame->data[0] = 0x80;

```

```

00217     frame->data[1] = 0x00;
00218     frame->data[2] = 0x80;
00219     frame->data[3] = 0x00;
00220     frame->data[4] = 0x00;
00221     frame->data[5] = 0x00;
00222     frame->data[6] = 0x08;
00223     frame->data[7] = 0x00;
00224 }
00225
00227 /*                                     Maxon Parser definition
    */
00229
00230 std::tuple<int, float, float> MaxonCommandParser::parseRecieveCommand(MaxonMotor &motor, struct
    can_frame *frame)
00231 {
00232     int id = frame->can_id;
00233
00234     int32_t currentPosition = 0;
00235     currentPosition |= static_cast<uint8_t>(frame->data[2]);
00236     currentPosition |= static_cast<uint8_t>(frame->data[3]) << 8;
00237     currentPosition |= static_cast<uint8_t>(frame->data[4]) << 16;
00238     currentPosition |= static_cast<uint8_t>(frame->data[5]) << 24;
00239
00240     int16_t torqueActualValue = 0;
00241     torqueActualValue |= static_cast<uint8_t>(frame->data[6]);
00242     torqueActualValue |= static_cast<uint8_t>(frame->data[7]) << 8;
00243
00244     // Motor rated torque 값을 N·m 단위로 변환 (mNm -> N·m)
00245     const float motorRatedTorquemNm = 31.052; //
00246
00247     // 실제 토크 값을 N·m 단위로 계산
00248     // Torque actual value는 천분의 일 단위이므로, 실제 토크 값은 (torqueActualValue / 1000) * motorRatedTorqueNm
00249     float currentTorqueNm = (static_cast<float>(torqueActualValue) / 1000.0f) * motorRatedTorquemNm;
00250
00251     float currentPositionDegrees = (static_cast<float>(currentPosition) / (35.0f * 4096.0f)) * 360.0f;
00252     float currentPositionRadians = currentPositionDegrees * (M_PI / 180.0f);
00253
00254     motor.currentPos = currentPositionRadians;
00255     motor.currentTor = currentTorqueNm;
00256
00257     return std::make_tuple(id, currentPositionRadians, currentTorqueNm);
00258 }
00259
00260 // System
00261 void MaxonCommandParser::getCheck(MaxonMotor &motor, struct can_frame *frame)
00262 {
00263
00264     frame->can_id = motor.canSendId;
00265     frame->can_dlc = 8;
00266     frame->data[0] = 0x00;
00267     frame->data[1] = 0x00;
00268     frame->data[2] = 0x00;
00269     frame->data[3] = 0x00;
00270     frame->data[4] = 0x00;
00271     frame->data[5] = 0x00;
00272     frame->data[6] = 0x00;
00273     frame->data[7] = 0x00;
00274 }
00275
00276 void MaxonCommandParser::getStop(MaxonMotor &motor, struct can_frame *frame)
00277 {
00278     frame->can_id = 0x00;
00279     frame->can_dlc = 8;
00280     frame->data[0] = 0x02;
00281     frame->data[1] = motor.nodeId;
00282     frame->data[2] = 0x00;
00283     frame->data[3] = 0x00;
00284     frame->data[4] = 0x00;
00285     frame->data[5] = 0x00;
00286     frame->data[6] = 0x00;
00287     frame->data[7] = 0x00;
00288 }
00289
00290 void MaxonCommandParser::getQuickStop(MaxonMotor &motor, struct can_frame *frame)
00291 {
00292     frame->can_id = motor.txPdoIds[0];
00293     frame->can_dlc = 8;
00294     frame->data[0] = 0x06;
00295     frame->data[1] = 0x00;
00296     frame->data[2] = 0x00;
00297     frame->data[3] = 0x00;
00298     frame->data[4] = 0x00;
00299     frame->data[5] = 0x00;
00300     frame->data[6] = 0x00;
00301     frame->data[7] = 0x00;
00302 }
00303

```

```

00304 void MaxonCommandParser::getOperational(MaxonMotor &motor, struct can_frame *frame)
00305 {
00306     frame->can_id = 0x00;
00307     frame->can_dlc = 8;
00308     frame->data[0] = 0x01;
00309     frame->data[1] = motor.nodeId;
00310     frame->data[2] = 0x00;
00311     frame->data[3] = 0x00;
00312     frame->data[4] = 0x00;
00313     frame->data[5] = 0x00;
00314     frame->data[6] = 0x00;
00315     frame->data[7] = 0x00;
00316 }
00317
00318 void MaxonCommandParser::getEnable(MaxonMotor &motor, struct can_frame *frame)
00319 {
00320     frame->can_id = motor.txPdoIds[0];
00321     frame->can_dlc = 8;
00322     frame->data[0] = 0x0F;
00323     frame->data[1] = 0x00;
00324     frame->data[2] = 0x00;
00325     frame->data[3] = 0x00;
00326     frame->data[4] = 0x00;
00327     frame->data[5] = 0x00;
00328     frame->data[6] = 0x00;
00329     frame->data[7] = 0x00;
00330 }
00331
00332 void MaxonCommandParser::getSync(struct can_frame *frame)
00333 {
00334     frame->can_id = 0x80;
00335     frame->can_dlc = 0;
00336 }
00337
00338 // CSP
00339 void MaxonCommandParser::getCSPMode(MaxonMotor &motor, struct can_frame *frame)
00340 {
00341     frame->can_id = motor.canSendId;
00342     frame->can_dlc = 8;
00343     frame->data[0] = 0x22;
00344     frame->data[1] = 0x60;
00345     frame->data[2] = 0x60;
00346     frame->data[3] = 0x00;
00347     frame->data[4] = 0x08;
00348     frame->data[5] = 0x00;
00349     frame->data[6] = 0x00;
00350     frame->data[7] = 0x00;
00351 }
00352
00353 void MaxonCommandParser::getTorqueOffset(MaxonMotor &motor, struct can_frame *frame)
00354 {
00355     frame->can_id = motor.canSendId;
00356     frame->can_dlc = 8;
00357     frame->data[0] = 0x22;
00358     frame->data[1] = 0xB2;
00359     frame->data[2] = 0x60;
00360     frame->data[3] = 0x00;
00361     frame->data[4] = 0x00;
00362     frame->data[5] = 0x00;
00363     frame->data[6] = 0x00;
00364     frame->data[7] = 0x00;
00365 }
00366
00367 void MaxonCommandParser::getPosOffset(MaxonMotor &motor, struct can_frame *frame)
00368 {
00369     frame->can_id = motor.canSendId;
00370     frame->can_dlc = 8;
00371     frame->data[0] = 0x22;
00372     frame->data[1] = 0xB0;
00373     frame->data[2] = 0x60;
00374     frame->data[3] = 0x00;
00375     frame->data[4] = 0x00;
00376     frame->data[5] = 0x00;
00377     frame->data[6] = 0x00;
00378     frame->data[7] = 0x00;
00379 }
00380
00381 void MaxonCommandParser::getTargetPosition(MaxonMotor &motor, struct can_frame *frame, float
    p_des_radians)
00382 {
00383     // 라디안 값을 인코더 값으로 변환
00384     float p_des_degrees = p_des_radians * (180.0f / M_PI); // 라디안을 도로 변환
00385     int p_des_enc = static_cast<int>(p_des_degrees * (35.0f * 4096.0f) / 360.0f); // 도를 인코더 값으로 변
    환
00386
00387     unsigned char posByte0 = p_des_enc & 0xFF; // 하위 8비트
00388     unsigned char posByte1 = (p_des_enc >> 8) & 0xFF; // 다음 8비트

```



```

00389     unsigned char posByte2 = (p_des_enc » 16) & 0xFF; // 다음 8비트
00390     unsigned char posByte3 = (p_des_enc » 24) & 0xFF; // 최상위 8비트
00391
00392     // Set CAN frame id and data length code
00393     frame->can_id = motor.txPdoIds[1];
00394     frame->can_dlc = 4;
00395
00397     frame->data[0] = posByte0;
00398     frame->data[1] = posByte1;
00399     frame->data[2] = posByte2;
00400     frame->data[3] = posByte3;
00401     frame->data[4] = 0x00;
00402     frame->data[5] = 0x00;
00403     frame->data[6] = 0x00;
00404     frame->data[7] = 0x00;
00405 }
00406
00407 // HMM
00408 void MaxonCommandParser::getHomeMode(MaxonMotor &motor, struct can_frame *frame)
00409 {
00410     frame->can_id = motor.canSendId;
00411     frame->can_dlc = 8;
00412     frame->data[0] = 0x22;
00413     frame->data[1] = 0x60;
00414     frame->data[2] = 0x60;
00415     frame->data[3] = 0x00;
00416     frame->data[4] = 0x06;
00417     frame->data[5] = 0x00;
00418     frame->data[6] = 0x00;
00419     frame->data[7] = 0x00;
00420 }
00421
00422 void MaxonCommandParser::getFlowingErrorWindow(MaxonMotor &motor, struct can_frame *frame)
00423 {
00424     frame->can_id = motor.canSendId;
00425     frame->can_dlc = 8;
00426     frame->data[0] = 0x22;
00427     frame->data[1] = 0x65;
00428     frame->data[2] = 0x60;
00429     frame->data[3] = 0x00;
00430     frame->data[4] = 0x00;
00431     frame->data[5] = 0x00;
00432     frame->data[6] = 0x00;
00433     frame->data[7] = 0x00;
00434 }
00435
00436 void MaxonCommandParser::getHomeoffsetDistance(MaxonMotor &motor, struct can_frame *frame, int degree)
00437 {
00438     // 1도당 값
00439     float value_per_degree = 398.22;
00440
00441     // 입력된 각도에 대한 값을 계산
00442     int value = static_cast<int>(degree * value_per_degree);
00443
00444     frame->can_id = motor.canSendId;
00445     frame->can_dlc = 8;
00446     frame->data[0] = 0x22;
00447     frame->data[1] = 0xB1;
00448     frame->data[2] = 0x30;
00449     frame->data[3] = 0x00;
00450     // 계산된 값의 리틀 엔디언 형식으로 분할하여 할당
00451     frame->data[4] = value & 0xFF; // 하위 바이트
00452     frame->data[5] = (value » 8) & 0xFF; // 상위 바이트
00453     frame->data[6] = 0x00;
00454     frame->data[7] = 0x00;
00455 }
00456
00457 void MaxonCommandParser::getHomePosition(MaxonMotor &motor, struct can_frame *frame, int degree)
00458 {
00459     float value_per_degree = 398.22;
00460     int value = static_cast<int>(degree * value_per_degree);
00461
00462     frame->can_id = motor.canSendId;
00463     frame->can_dlc = 8;
00464     frame->data[0] = 0x22;
00465     frame->data[1] = 0xB0;
00466     frame->data[2] = 0x30;
00467     frame->data[3] = 0x00;
00468     frame->data[4] = value & 0xFF; // 하위 바이트
00469     frame->data[5] = (value » 8) & 0xFF; // 상위 바이트
00470     frame->data[6] = 0x00;
00471     frame->data[7] = 0x00;
00472 }
00473 }
00474
00475 void MaxonCommandParser::getHomingMethodL(MaxonMotor &motor, struct can_frame *frame)
00476 {

```

```

00477     frame->can_id = motor.canSendId;
00478     frame->can_dlc = 8;
00479     frame->data[0] = 0x22;
00480     frame->data[1] = 0x98;
00481     frame->data[2] = 0x60;
00482     frame->data[3] = 0x00;
00483     frame->data[4] = 0xFD;
00484     frame->data[5] = 0xFF;
00485     frame->data[6] = 0xFF;
00486     frame->data[7] = 0xFF;
00487 }
00488
00489 void MaxonCommandParser::getHomingMethodR(MaxonMotor &motor, struct can_frame *frame)
00490 {
00491     frame->can_id = motor.canSendId;
00492     frame->can_dlc = 8;
00493     frame->data[0] = 0x22;
00494     frame->data[1] = 0x98;
00495     frame->data[2] = 0x60;
00496     frame->data[3] = 0x00;
00497     frame->data[4] = 0xFC;
00498     frame->data[5] = 0xFF;
00499     frame->data[6] = 0xFF;
00500     frame->data[7] = 0xFF;
00501 }
00502
00503 void MaxonCommandParser::getStartHoming(MaxonMotor &motor, struct can_frame *frame)
00504 {
00505     frame->can_id = motor.txPdoIds[0];
00506     frame->can_dlc = 8;
00507     frame->data[0] = 0x1F;
00508     frame->data[1] = 0x00;
00509     frame->data[2] = 0x00;
00510     frame->data[3] = 0x00;
00511     frame->data[4] = 0x00;
00512     frame->data[5] = 0x00;
00513     frame->data[6] = 0x00;
00514     frame->data[7] = 0x00;
00515 }
00516
00517 void MaxonCommandParser::getCurrentThreshold(MaxonMotor &motor, struct can_frame *frame)
00518 {
00519     // 1000 = 3E8
00520     // 500 = 01F4
00521     frame->can_id = motor.canSendId;
00522     frame->can_dlc = 8;
00523     frame->data[0] = 0x23;
00524     frame->data[1] = 0xB2;
00525     frame->data[2] = 0x30;
00526     frame->data[3] = 0x00;
00527     frame->data[4] = 0xF4;
00528     frame->data[5] = 0x01;
00529     frame->data[6] = 0x00;
00530     frame->data[7] = 0x00;
00531 }
00532
00533 // CSV
00534 void MaxonCommandParser::getCSVMode(MaxonMotor &motor, struct can_frame *frame)
00535 {
00536     frame->can_id = motor.canSendId;
00537     frame->can_dlc = 8;
00538     frame->data[0] = 0x22;
00539     frame->data[1] = 0x60;
00540     frame->data[2] = 0x60;
00541     frame->data[3] = 0x00;
00542     frame->data[4] = 0x09;
00543     frame->data[5] = 0x00;
00544     frame->data[6] = 0x00;
00545     frame->data[7] = 0x00;
00546 }
00547
00548 void MaxonCommandParser::getVelOffset(MaxonMotor &motor, struct can_frame *frame)
00549 {
00550     frame->can_id = motor.canSendId;
00551     frame->can_dlc = 8;
00552     frame->data[0] = 0x22;
00553     frame->data[1] = 0xB1;
00554     frame->data[2] = 0x60;
00555     frame->data[3] = 0x00;
00556     frame->data[4] = 0x00;
00557     frame->data[5] = 0x00;
00558     frame->data[6] = 0x00;
00559     frame->data[7] = 0x00;
00560 }
00561
00562 void MaxonCommandParser::getTargetVelocity(MaxonMotor &motor, struct can_frame *frame, int
    targetVelocity)

```

```

00563 {
00564     unsigned char velByte0 = targetVelocity & 0xFF; // 하위 8비트
00565     unsigned char velByte1 = (targetVelocity >> 8) & 0xFF; // 다음 8비트
00566     unsigned char velByte2 = (targetVelocity >> 16) & 0xFF; // 다음 8비트
00567     unsigned char velByte3 = (targetVelocity >> 24) & 0xFF; // 최상위 8비트
00568
00569     // Set CAN frame id and data length code
00570     frame->can_id = motor.txPdoIds[2];
00571     frame->can_dlc = 4;
00572
00573     frame->data[0] = velByte0;
00574     frame->data[1] = velByte1;
00575     frame->data[2] = velByte2;
00576     frame->data[3] = velByte3;
00577     frame->data[4] = 0x00;
00578     frame->data[5] = 0x00;
00579     frame->data[6] = 0x00;
00580     frame->data[7] = 0x00;
00581 }
00582 }
00583
00584 // CST
00585 void MaxonCommandParser::getCSTMode(MaxonMotor &motor, struct can_frame *frame)
00586 {
00587     frame->can_id = motor.canSendId;
00588     frame->can_dlc = 8;
00589     frame->data[0] = 0x22;
00590     frame->data[1] = 0x60;
00591     frame->data[2] = 0x60;
00592     frame->data[3] = 0x00;
00593     frame->data[4] = 0x0A;
00594     frame->data[5] = 0x00;
00595     frame->data[6] = 0x00;
00596     frame->data[7] = 0x00;
00597 }
00598
00599 void MaxonCommandParser::getTargetTorque(MaxonMotor &motor, struct can_frame *frame, int targetTorque)
00600 {
00601     unsigned char torByte0 = targetTorque & 0xFF; // 하위 8비트
00602     unsigned char torByte1 = (targetTorque >> 8) & 0xFF; // 다음 8비트
00603
00604     // Set CAN frame id and data length code
00605     frame->can_id = motor.txPdoIds[3];
00606     frame->can_dlc = 2;
00607
00608     frame->data[0] = torByte0;
00609     frame->data[1] = torByte1;
00610     frame->data[2] = 0x00;
00611     frame->data[3] = 0x00;
00612     frame->data[4] = 0x00;
00613     frame->data[5] = 0x00;
00614     frame->data[6] = 0x00;
00615     frame->data[7] = 0x00;
00616 }
00617 }

```

## 5.11 DrumRobot.cpp

```

00001 #include "../include/tasks/DrumRobot.hpp"
00002
00003 // DrumRobot 클래스의 생성자
00004 DrumRobot::DrumRobot(SystemState &systemStateRef,
00005                      CanManager &canManagerRef,
00006                      PathManager &pathManagerRef,
00007                      HomeManager &homeManagerRef,
00008                      TestManager &testManagerRef,
00009                      std::map<std::string, std::shared_ptr<GenericMotor>> &motorsRef)
00010 : systemState(systemStateRef),
00011   canManager(canManagerRef),
00012   pathManager(pathManagerRef),
00013   homeManager(homeManagerRef),
00014   testManager(testManagerRef),
00015   motors(motorsRef)
00016 {
00017 }
00018
00019 /*                      SYSTEM LOOPS                      */
00020
00021 void DrumRobot::stateMachine()
00022 {
00023     while (systemState.main != Main::Shutdown)
00024     {
00025         switch (systemState.main.load())
00026         {
00027         }
00028     }
00029 }

```

```

00030     case Main::SystemInit:
00031         initializeMotors();
00032         initializecanManager();
00033         motorSettingCmd();
00034         std::cout << "System Initialize Complete [ Press Enter ]\n";
00035         getchar();
00036         systemState.main = Main::Ideal;
00037         break;
00038
00039     case Main::Ideal:
00040         idealStateRoutine();
00041         break;
00042
00043     case Main::Homing:
00044         homeManager.mainLoop();
00045         break;
00046
00047     case Main::Perform:
00048         checkUserInput();
00049         break;
00050
00051     case Main::Check:
00052         canManager.checkAllMotors();
00053         printCurrentPositions();
00054         systemState.main = Main::Ideal;
00055         break;
00056
00057     case Main::Tune:
00058         testManager.mainLoop();
00059         break;
00060
00061     case Main::Shutdown:
00062         break;
00063
00064     case Main::Ready:
00065         if (!isReady)
00066         {
00067             if (canManager.checkAllMotors())
00068             {
00069                 MaxonEnable();
00070                 setMaxonMode("CSP");
00071                 cout << "Get Ready...\n";
00072                 clearMotorsSendBuffer();
00073                 pathManager.GetArr(pathManager.standby);
00074                 SendReadyLoop();
00075                 isReady = true;
00076             }
00077         }
00078         else
00079             idealStateRoutine();
00080         break;
00081
00082     case Main::Back:
00083         if (!isBack)
00084         {
00085             if (canManager.checkAllMotors())
00086             {
00087                 cout << "Get Back...\n";
00088                 clearMotorsSendBuffer();
00089                 pathManager.GetArr(pathManager.backarr);
00090                 SendReadyLoop();
00091                 isBack = true;
00092             }
00093         }
00094         else
00095         {
00096             systemState.main = Main::Shutdown;
00097             DeactivateControlTask();
00098         }
00099         break;
00100     case Main::Pause:
00101         checkUserInput();
00102         break;
00103     }
00104 }
00105 }
00106
00107 void DrumRobot::sendLoopForThread()
00108 {
00109     initializePathManager();
00110     while (systemState.main != Main::Shutdown)
00111     {
00112         usleep(50000);
00113         if (systemState.main == Main::Perform)
00114         {
00115             if (canManager.checkAllMotors())
00116             {

```

```

00117         SendLoop();
00118     }
00119 }
00120 }
00121 }
00122
00123 void DrumRobot::recvLoopForThread()
00124 {
00125
00126     while (systemState.main != Main::Shutdown)
00127     {
00128         usleep(50000);
00129         if (systemState.main == Main::Ideal)
00130         {
00131             canManager.checkCanPortsStatus();
00132             canManager.checkAllMotors();
00133             sleep(3);
00134         }
00135         else if (systemState.main == Main::Perform)
00136         {
00137             canManager.clearReadBuffers();
00138             RecieveLoop();
00139         }
00140     }
00141 }
00142 /* STATE UTILITY */
00143
00144 void DrumRobot::displayAvailableCommands() const
00145 {
00146     std::cout << "Available Commands:\n";
00147
00148     if (systemState.main == Main::Ideal)
00149     {
00150         if (systemState.homeMode == HomeMode::NotHome)
00151         {
00152             std::cout << "- h : Start Homing Mode\n";
00153             std::cout << "- x : Make home state by user\n";
00154         }
00155         else if (systemState.homeMode == HomeMode::HomeDone)
00156         {
00157             std::cout << "- r : Move to Ready Position\n";
00158             std::cout << "- t : Start tuning\n";
00159         }
00160     }
00161     else if (systemState.main == Main::Ready)
00162     {
00163         std::cout << "- p : Start Perform\n";
00164         std::cout << "- t : Start tuning\n";
00165     }
00166     std::cout << "- s : Shut down the system\n";
00167     std::cout << "- c : Check Motors position\n";
00168 }
00169
00170 bool DrumRobot::processInput(const std::string &input)
00171 {
00172     if (systemState.main == Main::Ideal)
00173     {
00174         if (input == "h" && systemState.homeMode == HomeMode::NotHome)
00175         {
00176             systemState.main = Main::Homing;
00177             return true;
00178         }
00179         else if (input == "t" && systemState.homeMode == HomeMode::HomeDone)
00180         {
00181             systemState.main = Main::Tune;
00182             return true;
00183         }
00184         else if (input == "r" && systemState.homeMode == HomeMode::HomeDone)
00185         {
00186             systemState.main = Main::Ready;
00187             return true;
00188         }
00189         else if (input == "x" && systemState.homeMode == HomeMode::NotHome)
00190         {
00191             systemState.homeMode = HomeMode::HomeDone;
00192             return true;
00193         }
00194         else if (input == "c")
00195         {
00196             systemState.main = Main::Check;
00197             return true;
00198         }
00199         else if (input == "s")
00200         {
00201             if (systemState.homeMode == HomeMode::NotHome)
00202                 systemState.main = Main::Shutdown;
00203             else if (systemState.homeMode == HomeMode::HomeDone)

```

```

00206         systemState.main = Main::Back;
00207         return true;
00208     }
00209 }
00210 else if (systemState.main == Main::Ready)
00211 {
00212     if (input == "p")
00213     {
00214         systemState.main = Main::Perform;
00215         return true;
00216     }
00217     else if (input == "s")
00218     {
00219         systemState.main = Main::Back;
00220         return true;
00221     }
00222     else if (input == "t")
00223     {
00224         systemState.main = Main::Tune;
00225         return true;
00226     }
00227     else if (input == "c")
00228     {
00229         systemState.main = Main::Check;
00230         return true;
00231     }
00232 }
00233 return false;
00235 }
00236
00237 void DrumRobot::idealStateRoutine()
00238 {
00239     int ret = system("clear");
00240     if (ret == -1)
00241         cout << "system clear error" << endl;
00242
00243     displayAvailableCommands();
00244
00245     std::string input;
00246     std::cout << "Enter command: ";
00247     std::getline(std::cin, input);
00248
00249     if (!processInput(input))
00250         std::cout << "Invalid command or not allowed in current state!\n";
00251
00252     usleep(2000);
00253 }
00254
00255 void DrumRobot::checkUserInput()
00256 {
00257     if (kbhit())
00258     {
00259         char input = getchar();
00260         if (input == 'q')
00261             systemState.main = Main::Pause;
00262         else if (input == 'e')
00263         {
00264             isReady = false;
00265             systemState.main = Main::Ready;
00266             pathManager.line = 0;
00267         }
00268         else if (input == 'r')
00269             systemState.main = Main::Perform;
00270     }
00271     usleep(500000);
00272 }
00273
00274 int DrumRobot::kbhit()
00275 {
00276     struct termios oldt, newt;
00277     int ch;
00278     int oldf;
00279
00280     tcgetattr(STDIN_FILENO, &oldt);
00281     newt = oldt;
00282     newt.c_lflag &= ~(ICANON | ECHO);
00283     tcsetattr(STDIN_FILENO, TCSANOW, &newt);
00284     oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
00285     fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);
00286
00287     ch = getchar();
00288
00289     tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
00290     fcntl(STDIN_FILENO, F_SETFL, oldf);
00291
00292     if (ch != EOF)

```

```

00293     {
00294         ungetc(ch, stdin);
00295         return 1;
00296     }
00297
00298     return 0;
00299 }
00301 /*                      SYSTEM                      */
00303
00304 void DrumRobot::initializeMotors()
00305 {
00306     motors["waist"] = make_shared<TMotor>(0x007, "AK10_9");
00307     motors["R_arm1"] = make_shared<TMotor>(0x001, "AK70_10");
00308     motors["L_arm1"] = make_shared<TMotor>(0x002, "AK70_10");
00309     motors["R_arm2"] = make_shared<TMotor>(0x003, "AK70_10");
00310     motors["R_arm3"] = make_shared<TMotor>(0x004, "AK70_10");
00311     motors["L_arm2"] = make_shared<TMotor>(0x005, "AK70_10");
00312     motors["L_arm3"] = make_shared<TMotor>(0x006, "AK70_10");
00313     motors["L_wrist"] = make_shared<MaxonMotor>(0x009);
00314     motors["R_wrist"] = make_shared<MaxonMotor>(0x008);
00315     motors["maxonForTest"] = make_shared<MaxonMotor>(0x00A);
00316
00317     for (auto &motor_pair : motors)
00318     {
00319         auto &motor = motor_pair.second;
00320
00321         // 타입에 따라 적절한 캐스팅과 초기화 수행
00322         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00323         {
00324             // 각 모터 이름에 따른 멤버 변수 설정
00325             if (motor_pair.first == "waist")
00326             {
00327                 tMotor->cwDir = 1.0f;
00328                 tMotor->rMin = -M_PI / 2.0f; // -90deg
00329                 tMotor->rMax = M_PI / 2.0f; // 90deg
00330                 tMotor->Kp = 400;
00331                 tMotor->Kd = 3.5;
00332                 tMotor->isHomed = true;
00333                 tMotor->interFaceName = "can0";
00334             }
00335             else if (motor_pair.first == "R_arm1")
00336             {
00337                 tMotor->cwDir = -1.0f;
00338                 tMotor->sensorBit = 3;
00339                 tMotor->rMin = -M_PI; // -180deg
00340                 tMotor->rMax = 0.0f; // 0deg
00341                 tMotor->Kp = 200;
00342                 tMotor->Kd = 2.5;
00343                 tMotor->isHomed = false;
00344                 tMotor->interFaceName = "can1";
00345             }
00346             else if (motor_pair.first == "L_arm1")
00347             {
00348                 tMotor->cwDir = 1.0f;
00349                 tMotor->sensorBit = 0;
00350                 tMotor->rMin = 0.0f; // 0deg
00351                 tMotor->rMax = M_PI; // 180deg
00352                 tMotor->Kp = 200;
00353                 tMotor->Kd = 2.5;
00354                 tMotor->isHomed = false;
00355                 tMotor->interFaceName = "can0";
00356             }
00357             else if (motor_pair.first == "R_arm2")
00358             {
00359                 tMotor->cwDir = 1.0f;
00360                 tMotor->sensorBit = 4;
00361                 tMotor->rMin = -M_PI / 4.0f; // -45deg
00362                 tMotor->rMax = M_PI / 2.0f; // 90deg
00363                 tMotor->Kp = 350;
00364                 tMotor->Kd = 3.5;
00365                 tMotor->isHomed = false;
00366                 tMotor->interFaceName = "can1";
00367             }
00368             else if (motor_pair.first == "R_arm3")
00369             {
00370                 tMotor->cwDir = -1.0f;
00371                 tMotor->sensorBit = 5;
00372                 tMotor->rMin = -M_PI * 0.75f; // -135deg
00373                 tMotor->rMax = 0.0f; // 0deg
00374                 tMotor->Kp = 250;
00375                 tMotor->Kd = 3.5;
00376                 tMotor->isHomed = false;
00377                 tMotor->interFaceName = "can1";
00378             }
00379             else if (motor_pair.first == "L_arm2")
00380             {
00381                 tMotor->cwDir = -1.0f;

```

```

00382         tMotor->sensorBit = 1;
00383         tMotor->rMin = -M_PI / 2.0f; // -90deg
00384         tMotor->rMax = M_PI / 4.0f; // 45deg
00385         tMotor->Kp = 350;
00386         tMotor->Kd = 3.5;
00387         tMotor->isHomed = false;
00388         tMotor->interFaceName = "can0";
00389     }
00390     else if (motor_pair.first == "L_arm3")
00391     {
00392         tMotor->cwDir = -1.0f;
00393         tMotor->sensorBit = 2;
00394         tMotor->rMin = -M_PI * 0.75f; // -135deg
00395         tMotor->rMax = 0.0f; // 0deg
00396         tMotor->Kp = 250;
00397         tMotor->Kd = 3.5;
00398         tMotor->isHomed = false;
00399         tMotor->interFaceName = "can0";
00400     }
00401 }
00402 else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor))
00403 {
00404     // 각 모터 이름에 따른 멤버 변수 설정
00405     if (motor_pair.first == "L_wrist")
00406     {
00407         maxonMotor->cwDir = -1.0f;
00408         maxonMotor->rMin = -M_PI * 0.75f; // -120deg
00409         maxonMotor->rMax = M_PI / 2.0f; // 90deg
00410         maxonMotor->isHomed = false;
00411         maxonMotor->txPdoIds[0] = 0x209; // Controlword
00412         maxonMotor->txPdoIds[1] = 0x309; // TargetPosition
00413         maxonMotor->txPdoIds[2] = 0x409; // TargetVelocity
00414         maxonMotor->txPdoIds[3] = 0x509; // TargetTorque
00415         maxonMotor->rxPdoIds[0] = 0x189; // Statusword, ActualPosition, ActualTorque
00416         maxonMotor->interFaceName = "can2";
00417     }
00418     else if (motor_pair.first == "R_wrist")
00419     {
00420         maxonMotor->cwDir = 1.0f;
00421         maxonMotor->rMin = 0.0f; // 0deg
00422         maxonMotor->rMax = M_PI; // 180deg
00423         maxonMotor->isHomed = false;
00424         maxonMotor->txPdoIds[0] = 0x208; // Controlword
00425         maxonMotor->txPdoIds[1] = 0x308; // TargetPosition
00426         maxonMotor->txPdoIds[2] = 0x408; // TargetVelocity
00427         maxonMotor->txPdoIds[3] = 0x508; // TargetTorque
00428         maxonMotor->rxPdoIds[0] = 0x188; // Statusword, ActualPosition, ActualTorque
00429         maxonMotor->interFaceName = "can2";
00430     }
00431     else if (motor_pair.first == "maxonForTest")
00432     {
00433         maxonMotor->cwDir = 1.0f;
00434         maxonMotor->rMin = 0.0f; // 0deg
00435         maxonMotor->rMax = M_PI; // 180deg
00436         maxonMotor->isHomed = false;
00437         maxonMotor->txPdoIds[0] = 0x20A; // Controlword
00438         maxonMotor->txPdoIds[1] = 0x30A; // TargetPosition
00439         maxonMotor->txPdoIds[2] = 0x40A; // TargetVelocity
00440         maxonMotor->txPdoIds[3] = 0x50A; // TargetTorque
00441         maxonMotor->rxPdoIds[0] = 0x18A; // Statusword, ActualPosition, ActualTorque
00442     }
00443 }
00444 }
00445 };
00446
00447 void DrumRobot::initializecanManager()
00448 {
00449     canManager.initializeCAN();
00450     canManager.checkCanPortsStatus();
00451     canManager.setMotorsSocket();
00452 }
00453
00454 void DrumRobot::DeactivateControlTask()
00455 {
00456     struct can_frame frame;
00457
00458     canManager.setSocketsTimeout(0, 50000);
00459
00460     for (auto &motorPair : motors)
00461     {
00462         std::string name = motorPair.first;
00463         auto &motor = motorPair.second;
00464
00465         // 타입에 따라 적절한 캐스팅과 초기화 수행
00466         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00467         {

```



```

00468         tMotorCmd.getCheck(*tMotor, &frame);
00469         canManager.sendAndRecv(motor, frame);
00470
00471         tMotorCmd.getExit(*tMotor, &frame);
00472         if (canManager.sendAndRecv(motor, frame))
00473             std::cout << "Exiting for motor [" << name << "]" << std::endl;
00474         else
00475             std::cerr << "Failed to exit control mode for motor [" << name << "]." << std::endl;
00476     }
00477     else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor))
00478     {
00479         maxonCmd.getQuickStop(*maxonMotor, &frame);
00480         canManager.txFrame(motor, frame);
00481
00482         maxonCmd.getSync(&frame);
00483         canManager.txFrame(motor, frame);
00484         if (canManager.recvToBuff(motor, canManager.maxonCnt))
00485         {
00486             while (!motor->recieveBuffer.empty())
00487             {
00488                 frame = motor->recieveBuffer.front();
00489                 if (frame.can_id == maxonMotor->rxPdoIds[0])
00490                 {
00491                     std::cout << "Exiting for motor [" << name << "]" << std::endl;
00492                     break;
00493                 }
00494                 motor->recieveBuffer.pop();
00495             }
00496         }
00497         else
00498             std::cerr << "Failed to exit for motor [" << name << "]." << std::endl;
00499     }
00500 }
00501 }
00502
00503 void DrumRobot::printCurrentPositions()
00504 {
00505     for (auto &motorPair : motors)
00506     {
00507         std::string name = motorPair.first;
00508         auto &motor = motorPair.second;
00509         std::cout << "[" << std::hex << motor->nodeId << std::dec << "] ";
00510         std::cout << name << " : " << motor->currentPos << endl;
00511     }
00512
00513     vector<double> P(6);
00514     P = pathManager.fkfun();
00515
00516     cout << "Right Hand Position : { " << P[0] << " , " << P[1] << " , " << P[2] << " }\n";
00517     cout << "Left Hand Position : { " << P[3] << " , " << P[4] << " , " << P[5] << " }\n";
00518     printf("Print Enter to Go Home\n");
00519     getchar();
00520 }
00521
00522 void DrumRobot::setMaxonMode(std::string targetMode)
00523 {
00524     struct can_frame frame;
00525     canManager.setSocketsTimeout(0, 10000);
00526     for (const auto &motorPair : motors)
00527     {
00528         std::string name = motorPair.first;
00529         std::shared_ptr<GenericMotor> motor = motorPair.second;
00530         if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motorPair.second))
00531         {
00532             if (targetMode == "CSV")
00533             {
00534                 maxonCmd.getCSVMode(*maxonMotor, &frame);
00535                 canManager.sendAndRecv(motor, frame);
00536             }
00537             else if (targetMode == "CST")
00538             {
00539                 maxonCmd.getCSTMode(*maxonMotor, &frame);
00540                 canManager.sendAndRecv(motor, frame);
00541             }
00542             else if (targetMode == "HMM")
00543             {
00544                 maxonCmd.getHomeMode(*maxonMotor, &frame);
00545                 canManager.sendAndRecv(motor, frame);
00546             }
00547             else if (targetMode == "CSP")
00548             {
00549                 maxonCmd.getCSPMode(*maxonMotor, &frame);
00550                 canManager.sendAndRecv(motor, frame);
00551             }
00552         }
00553     }
00554 }

```

```

00553     }
00554 }
00555
00556 void DrumRobot::motorSettingCmd()
00557 {
00558     struct can_frame frame;
00559     canManager.setSocketsTimeout(2, 0);
00560     for (const auto &motorPair : motors)
00561     {
00562         std::string name = motorPair.first;
00563         std::shared_ptr<GenericMotor> motor = motorPair.second;
00564         if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motorPair.second))
00565         {
00566
00567             // CSP Settings
00568             maxoncmd.getCSVMode(*maxonMotor, &frame);
00569             canManager.sendAndRecv(motor, frame);
00570
00571             maxoncmd.getPosOffset(*maxonMotor, &frame);
00572             canManager.sendAndRecv(motor, frame);
00573
00574             maxoncmd.getTorqueOffset(*maxonMotor, &frame);
00575             canManager.sendAndRecv(motor, frame);
00576
00577             // CSV Settings
00578             maxoncmd.getCSVMode(*maxonMotor, &frame);
00579             canManager.sendAndRecv(motor, frame);
00580
00581             maxoncmd.getVelOffset(*maxonMotor, &frame);
00582             canManager.sendAndRecv(motor, frame);
00583
00584             // CST Settings
00585             maxoncmd.getCSTMode(*maxonMotor, &frame);
00586             canManager.sendAndRecv(motor, frame);
00587
00588             maxoncmd.getTorqueOffset(*maxonMotor, &frame);
00589             canManager.sendAndRecv(motor, frame);
00590
00591             // HMM Settings
00592             maxoncmd.getHomeMode(*maxonMotor, &frame);
00593             canManager.sendAndRecv(motor, frame);
00594
00595             if (name == "L_wrist")
00596             {
00597                 maxoncmd.getHomingMethodL(*maxonMotor, &frame);
00598                 canManager.sendAndRecv(motor, frame);
00599
00600                 maxoncmd.getHomeoffsetDistance(*maxonMotor, &frame, 0);
00601                 canManager.sendAndRecv(motor, frame);
00602
00603                 maxoncmd.getHomePosition(*maxonMotor, &frame, -90);
00604                 canManager.sendAndRecv(motor, frame);
00605             }
00606             else if (name == "R_wrist")
00607             {
00608                 maxoncmd.getHomingMethodR(*maxonMotor, &frame);
00609                 canManager.sendAndRecv(motor, frame);
00610
00611                 maxoncmd.getHomeoffsetDistance(*maxonMotor, &frame, 0);
00612                 canManager.sendAndRecv(motor, frame);
00613
00614                 maxoncmd.getHomePosition(*maxonMotor, &frame, -90);
00615                 canManager.sendAndRecv(motor, frame);
00616             }
00617             else if (name == "maxonForTest")
00618             {
00619                 maxoncmd.getHomingMethodL(*maxonMotor, &frame);
00620                 canManager.sendAndRecv(motor, frame);
00621
00622                 maxoncmd.getHomeoffsetDistance(*maxonMotor, &frame, 20);
00623                 canManager.sendAndRecv(motor, frame);
00624
00625                 maxoncmd.getHomePosition(*maxonMotor, &frame, 90);
00626                 canManager.sendAndRecv(motor, frame);
00627             }
00628
00629             maxoncmd.getCurrentThreshold(*maxonMotor, &frame);
00630             canManager.sendAndRecv(motor, frame);
00631         }
00632         else if (std::shared_ptr<TMotor> tmotor = std::dynamic_pointer_cast<TMotor>(motorPair.second))
00633         {
00634             if (name == "waist")
00635             {
00636                 tmotorcmd.getZero(*tmotor, &frame);
00637                 canManager.sendAndRecv(motor, frame);
00638             }

```

```

00639         usleep(5000);
00640         tMotorCmd.getControlMode(*tmotor, &frame);
00641         canManager.sendAndRecv(motor, frame);
00642     }
00643 }
00644 }
00645
00646 void DrumRobot::MaxonEnable()
00647 {
00648     struct can_frame frame;
00649     canManager.setSocketsTimeout(2, 0);
00650
00651     int maxonMotorCount = 0;
00652     for (const auto &motor_pair : motors)
00653     {
00654         // 각 요소가 MaxonMotor 타입인지 확인
00655         if (std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00656             maxonMotorCount++;
00657     }
00658
00659     // 제어 모드 설정
00660     for (const auto &motorPair : motors)
00661     {
00662         std::string name = motorPair.first;
00663         std::shared_ptr<GenericMotor> motor = motorPair.second;
00664         if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00665         {
00666             maxoncmd.getOperational(*maxonMotor, &frame);
00667             canManager.txFrame(motor, frame);
00668
00669             maxoncmd.getEnable(*maxonMotor, &frame);
00670             canManager.txFrame(motor, frame);
00671
00672             maxoncmd.getSync(&frame);
00673             canManager.txFrame(motor, frame);
00674
00675             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00676             {
00677                 while (!motor->recieveBuffer.empty())
00678                 {
00679                     frame = motor->recieveBuffer.front();
00680                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00681                         std::cout << "Maxon Enabled \n";
00682                     motor->recieveBuffer.pop();
00683                 }
00684             }
00685
00686             maxoncmd.getQuickStop(*maxonMotor, &frame);
00687             canManager.txFrame(motor, frame);
00688
00689             maxoncmd.getSync(&frame);
00690             canManager.txFrame(motor, frame);
00691
00692             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00693             {
00694                 while (!motor->recieveBuffer.empty())
00695                 {
00696                     frame = motor->recieveBuffer.front();
00697                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00698                         std::cout << "Maxon Quick Stopped\n";
00699                     motor->recieveBuffer.pop();
00700                 }
00701             }
00702
00703             maxoncmd.getEnable(*maxonMotor, &frame);
00704             canManager.txFrame(motor, frame);
00705
00706             maxoncmd.getSync(&frame);
00707             canManager.txFrame(motor, frame);
00708
00709             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00710             {
00711                 while (!motor->recieveBuffer.empty())
00712                 {
00713                     frame = motor->recieveBuffer.front();
00714                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00715                         std::cout << "Maxon Enabled \n";
00716                     motor->recieveBuffer.pop();
00717                 }
00718             }
00719         }
00720     }
00721 }
00722 };
00723
00724 void DrumRobot::MaxonDisable()
00725 {

```

```

00726     struct can_frame frame;
00727
00728     canManager.setSocketsTimeout(0, 50000);
00729
00730     for (auto &motorPair : motors)
00731     {
00732         std::string name = motorPair.first;
00733         auto &motor = motorPair.second;
00734
00735         if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00736         {
00737             maxoncmd.getQuickStop(*maxonMotor, &frame);
00738             canManager.txFrame(motor, frame);
00739
00740             maxoncmd.getSync(&frame);
00741             canManager.txFrame(motor, frame);
00742             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00743             {
00744                 while (!motor->recieveBuffer.empty())
00745                 {
00746                     frame = motor->recieveBuffer.front();
00747                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00748                         break;
00749                     motor->recieveBuffer.pop();
00750                 }
00751             }
00752             else
00753                 std::cerr << "Failed to exit for motor [" << name << "]." << std::endl;
00754         }
00755     }
00756 }
00757 /*                                     Send Thread Loop                                     */
00760
00761 void DrumRobot::SendLoop()
00762 {
00763     struct can_frame frameToProcess;
00764     std::string maxonCanInterface;
00765     std::shared_ptr<GenericMotor> virtualMaxonMotor;
00766
00767     int maxonMotorCount = 0;
00768     for (const auto &motor_pair : motors)
00769     {
00770         // 각 요소가 MaxonMotor 타입인지 확인
00771         if (std::shared_ptr<MaxonMotor> maxonMotor =
00772             std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00773         {
00774             maxonMotorCount++;
00775             maxonCanInterface = maxonMotor->interFaceName;
00776             virtualMaxonMotor = motor_pair.second;
00777         }
00778     }
00779     chrono::system_clock::time_point external = std::chrono::system_clock::now();
00780
00781     while (systemState.main == Main::Perform || systemState.main == Main::Pause)
00782     {
00783         if (systemState.main == Main::Pause)
00784             continue;
00785
00786         bool isAnyBufferLessThanTen = false;
00787         for (const auto &motor_pair : motors)
00788         {
00789             if (motor_pair.second->sendBuffer.size() < 10)
00790             {
00791                 isAnyBufferLessThanTen = true;
00792                 break;
00793             }
00794         }
00795         if (isAnyBufferLessThanTen)
00796         {
00797             if (pathManager.line < pathManager.total)
00798             {
00799                 std::cout << "line : " << pathManager.line << ", total : " << pathManager.total << "\n";
00800                 pathManager.PathLoopTask();
00801                 pathManager.line++;
00802             }
00803             else if (pathManager.line == pathManager.total)
00804             {
00805                 std::cout << "Perform Done\n";
00806                 systemState.main = Main::Ready;
00807                 pathManager.line = 0;
00808             }
00809         }
00810
00811         bool allBuffersEmpty = true;
00812         for (const auto &motor_pair : motors)
00813         {

```

```

00814         if (!motor_pair.second->sendBuffer.empty())
00815         {
00816             allBuffersEmpty = false;
00817             break;
00818         }
00819     }
00820
00821     // 모든 모터의 sendBuffer가 비었을 때 성능 종료 로직 실행
00822     if (allBuffersEmpty)
00823     {
00824         std::cout << "Performance is Over\n";
00825         systemState.main = Main::Ideal;
00826     }
00827
00828     chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00829     chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
external);
00830
00831     if (elapsed_time.count() >= 5000) // 5ms
00832     {
00833         external = std::chrono::system_clock::now();
00834
00835         for (auto &motor_pair : motors)
00836         {
00837             shared_ptr<GenericMotor> motor = motor_pair.second;
00838             canManager.sendFromBuff(motor);
00839         }
00840
00841         if (maxonMotorCount != 0)
00842         {
00843             maxoncmd.getSync(&frameToProcess);
00844             canManager.txFrame(virtualMaxonMotor, frameToProcess);
00845         }
00846     }
00847 }
00848
00849 // CSV 파일명 설정
00850 std::string csvFileName = "../..\\READ\\DrumData_in.txt";
00851
00852 // input 파일 저장
00853 save_to_txt_inputData(csvFileName);
00854 }
00855
00856 void DrumRobot::save_to_txt_inputData(const string &csv_file_name)
00857 {
00858     // CSV 파일 열기
00859     std::ofstream csvFile(csv_file_name);
00860
00861     if (!csvFile.is_open())
00862         std::cerr << "Error opening CSV file." << std::endl;
00863
00864     // 헤더 추가
00865     csvFile << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
00866
00867     // 2차원 벡터의 데이터를 CSV 파일로 쓰기
00868     for (const auto &row : pathManager.p)
00869     {
00870         for (const double cell : row)
00871         {
00872             csvFile << std::fixed << std::setprecision(5) << cell;
00873             if (&cell != &row.back())
00874                 csvFile << ","; // 쉼표로 셀 구분
00875         }
00876         csvFile << "\n"; // 다음 행으로 이동
00877     }
00878
00879     // CSV 파일 닫기
00880     csvFile.close();
00881
00882     std::cout << "연주 DrumData_in 파일이 생성되었습니다." << csv_file_name << std::endl;
00883
00884     std::cout << "SendLoop terminated\n";
00885 }
00886
00887 void DrumRobot::SendReadyLoop()
00888 {
00889     cout << "Settig...\n";
00890     struct can_frame frameToProcess;
00891     std::string maxonCanInterface;
00892     std::shared_ptr<GenericMotor> virtualMaxonMotor;
00893
00894     int maxonMotorCount = 0;
00895     for (const auto &motor_pair : motors)
00896     {
00897         // 각 요소가 MaxonMotor 타입인지 확인
00898         if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))

```

```

00899     {
00900         maxonMotorCount++;
00901         maxonCanInterface = maxonMotor->interFaceName;
00902         virtualMaxonMotor = motor_pair.second;
00903     }
00904 }
00905 chrono::system_clock::time_point external = std::chrono::system_clock::now();
00906
00907 bool allBuffersEmpty;
00908 do
00909 {
00910     allBuffersEmpty = true;
00911     for (const auto &motor_pair : motors)
00912     {
00913         if (!motor_pair.second->sendBuffer.empty())
00914         {
00915             allBuffersEmpty = false;
00916             break;
00917         }
00918     }
00919
00920     if (!allBuffersEmpty)
00921     {
00922         chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00923         chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
external);
00924
00925         if (elapsed_time.count() >= 5000) // 5ms
00926         {
00927             external = std::chrono::system_clock::now();
00928
00929             for (auto &motor_pair : motors)
00930             {
00931                 shared_ptr<GenericMotor> motor = motor_pair.second;
00932                 canManager.sendFromBuff(motor);
00933             }
00934
00935             if (maxonMotorCount != 0)
00936             {
00937                 maxoncmd.getSync(&frameToProcess);
00938                 canManager.txFrame(virtualMaxonMotor, frameToProcess);
00939             }
00940         }
00941     }
00942 } while (!allBuffersEmpty);
00943 canManager.clearReadBuffers();
00944 }
00945
00946 void DrumRobot::initializePathManager()
00947 {
00948     pathManager.ApplyDir();
00949     pathManager.GetDrumPositoin();
00950     pathManager.GetMusicSheet();
00951 }
00952
00953 void DrumRobot::clearMotorsSendBuffer()
00954 {
00955     for (auto motor_pair : motors)
00956         motor_pair.second->clearSendBuffer();
00957 }
00958
00959 /*
00960                                     Recive Thread Loop
00961 */
00962
00963 void DrumRobot::RecieveLoop()
00964 {
00965     chrono::system_clock::time_point external = std::chrono::system_clock::now();
00966
00967     canManager.setSocketsTimeout(0, 50000);
00968     canManager.clearReadBuffers();
00969
00970     sensor.connect();
00971     if (!sensor.connected)
00972         cout << "Sensor initialization failed. Skipping sensor related logic." << endl;
00973
00974     while (systemState.main == Main::Perform || systemState.main == Main::Pause)
00975     {
00976         if (systemState.main == Main::Pause)
00977             continue;
00978
00979         /*if (sensor.connected && (sensor.ReadVal() & 1) != 0)
00980         {
00981             cout << "Motors at Sensor Location please check!!!\n";
00982             systemState.runMode = RunMode::Pause;
00983         }*/
00984
00985         chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00986         chrono::milliseconds elapsed_time = chrono::duration_cast<chrono::milliseconds>(internal -

```

```

        external);
00987         if (elapsed_time.count() >= TIME_THRESHOLD_MS)
00988         {
00989             external = std::chrono::system_clock::now();
00990             canManager.readFramesFromAllSockets();
00991             canManager.distributeFramesToMotors();
00992         }
00993     }
00994
00995     parse_and_save_to_csv("../..//READ/DrumData_out.txt");
00996 }
00997
00998 void DrumRobot::parse_and_save_to_csv(const std::string &csv_file_name)
00999 {
01000     // CSV 파일 열기. 파일이 없으면 새로 생성됩니다.
01001     std::ofstream ofs(csv_file_name, std::ios::app);
01002     if (!ofs.is_open())
01003     {
01004         std::cerr << "Failed to open or create the CSV file: " << csv_file_name << std::endl;
01005         return;
01006     }
01007
01008     // 파일이 새로 생성되었으면 CSV 헤더를 추가
01009     ofs.seekp(0, std::ios::end);
01010     if (ofs.tellp() == 0)
01011         ofs << "CAN_ID,p_act,tff_des,tff_act\n";
01012
01013     // 각 모터에 대한 처리
01014     for (const auto &pair : motors)
01015     {
01016         auto &motor = pair.second;
01017         if (!motor->recieveBuffer.empty())
01018         {
01019             can_frame frame = motor->recieveBuffer.front();
01020             motor->recieveBuffer.pop();
01021
01022             int id = motor->nodeId;
01023             float position, speed, torque;
01024
01025             // TMotor 또는 MaxonMotor에 따른 데이터 파싱 및 출력
01026             if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
01027             {
01028                 std::tuple<int, float, float, float> parsedData =
01029                 tmotorecmd.parseRecieveCommand(*tMotor, &frame);
01030                 position = std::get<1>(parsedData);
01031                 speed = std::get<2>(parsedData);
01032                 torque = std::get<3>(parsedData);
01033             }
01034             else if (std::shared_ptr<MaxonMotor> maxonMotor =
01035                 std::dynamic_pointer_cast<MaxonMotor>(motor))
01036             {
01037                 std::tuple<int, float, float> parsedData = maxoncmd.parseRecieveCommand(*maxonMotor,
01038                 &frame);
01039                 position = std::get<1>(parsedData);
01040                 torque = std::get<2>(parsedData);
01041                 speed = 0.0;
01042             }
01043
01044             // 데이터 CSV 파일에 쓰기
01045             ofs << "0x" << std::hex << id << std::setw(4) << std::setfill('0') << id << ", "
01046             << std::dec << position << ", " << speed << ", " << torque << "\n";
01047         }
01048     }
01049     ofs.close();
01050     std::cout << "연주 txt_OutData 파일이 생성되었습니다." << csv_file_name << std::endl;
01051 }

```

## 5.12 HomeManager.cpp

```

00001 #include "../include/managers/HomeManager.hpp"
00002
00003 HomeManager::HomeManager(SystemState &systemStateRef,
00004                           CanManager &canManagerRef,
00005                           std::map<std::string, std::shared_ptr<GenericMotor>> &motorsRef)
00006 : systemState(systemStateRef), canManager(canManagerRef), motors(motorsRef)
00007 {
00008 }
00009
00010 void HomeManager::mainLoop()
00011 {
00012     while (systemState.main == Main::Homing)
00013     {

```

```

00014         displayHomingStatus();
00015
00016         std::string motorName;
00017         std::cout << "Enter the name of the motor to home, or 'all' to home all motors: ";
00018         std::cin >> motorName;
00019
00020         if (motorName == "all") // 차례행로 동시실행
00021         {
00022             // 우선순위가 높은 순서대로 먼저 홈
00023             vector<vector<string>> Priority = {{ "L_arm1", "R_arm1"}, {"L_arm2", "R_arm2"}, {"L_arm3",
"R_arm3"} };
00024             for (auto &PmotorNames : Priority)
00025             {
00026                 vector<shared_ptr<GenericMotor>> Pmotors;
00027                 vector<string> Pnames;
00028                 for (const auto &pmotorName : PmotorNames)
00029                 {
00030                     if (motors.find(pmotorName) != motors.end() && !motors[pmotorName]->isHomed)
00031                     {
00032                         Pmotors.push_back(motors[pmotorName]);
00033                         Pnames.push_back(pmotorName);
00034                     }
00035                 }
00036                 if (!Pmotors.empty())
00037                     SetTmotorHome(Pmotors, Pnames);
00038                 Pmotors.clear();
00039                 Pnames.clear();
00040             }
00041
00042             vector<string> PmotorNames = {"L_wrist", "R_wrist", "maxonForTest"};
00043             vector<shared_ptr<GenericMotor>> Pmotors;
00044             for (const auto &pmotorName : PmotorNames)
00045             {
00046                 if (motors.find(pmotorName) != motors.end() && !motors[pmotorName]->isHomed)
00047                     Pmotors.push_back(motors[pmotorName]);
00048             }
00049             if (!Pmotors.empty())
00050                 SetMaxonHome(Pmotors);
00051         }
00052         else if (motors.find(motorName) != motors.end() && !motors[motorName]->isHomed)
00053         { // 원하는 하나의 모터 실행
00054             vector<shared_ptr<GenericMotor>> Pmotor;
00055             vector<string> Pnames;
00056             // 타입에 따라 적절한 캐스팅과 초기화 수행
00057             if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motors[motorName]))
00058             {
00059                 Pmotor.push_back(motors[motorName]);
00060                 Pnames.push_back(motorName);
00061                 SetTmotorHome(Pmotor, Pnames);
00062             }
00063             else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[motorName]))
00064             {
00065                 Pmotor.push_back(motors[motorName]);
00066                 SetMaxonHome(Pmotor);
00067             }
00068         }
00069         else
00070         {
00071             std::cout << "Motor not found or already homed: " << motorName << std::endl;
00072         }
00073         UpdateHomingStatus();
00074     }
00075 }
00076
00077 bool HomeManager::PromptUserForHoming(const std::string &motorName)
00078 {
00079     char userResponse;
00080     std::cout << "Would you like to start homing mode for motor [" << motorName << "]? (y/n): ";
00081     std::cin >> userResponse;
00082     return userResponse == 'y';
00083 }
00084
00085 void HomeManager::SetTmotorHome(vector<std::shared_ptr<GenericMotor>> &motors, vector<std::string>
&motorNames)
00086 {
00087     sensor.OpenDeviceUntilSuccess();
00088     canManager.setSocketsTimeout(5, 0);
00089
00090     HomeTMotor(motors, motorNames);
00091     for (auto &motor : motors)
00092     {
00093         motor->isHomed = true; // 홈임 상태 업데이트
00094         sleep(2);
00095         FixMotorPosition(motor);
00096     }
00097 }

```



```

00098     for (auto &motorname : motorNames)
00099     {
00100         cout << "-- Homing completed for " << motorname << " --\n\n";
00101     }
00102
00103     sensor.closeDevice();
00104 }
00105
00106 void HomeManager::HomeTMotor(vector<std::shared_ptr<GenericMotor>> &motors, vector<std::string>
&motorNames)
00107 { // arm2 모터는 -30도, 나머지 모터는 +90도에 센서 위치함.
00108     struct can_frame frameToProcess;
00109     vector<shared_ptr<TMotor>> tMotors;
00110     vector<int> sensorsBit;
00111
00112     // 속도 제어 - 센서 방향으로 이동
00113     for (long unsigned int i = 0; i < motorNames.size(); i++)
00114     {
00115         cout << "< Homing for " << motorNames[i] << " >\n";
00116         tMotors.push_back(dynamic_pointer_cast<TMotor>(motors[i]));
00117
00118         double initialDirection;
00119         if (motorNames[i] == "L_arm2" || motorNames[i] == "R_arm2")
00120             initialDirection = (-0.2) * motors[i]->cwDir;
00121         else
00122             initialDirection = 0.2 * motors[i]->cwDir;
00123
00124         double additionalTorque = 0.0;
00125         if (motorNames[i] == "L_arm2" || motorNames[i] == "R_arm2")
00126             additionalTorque = motors[i]->cwDir * (-3.0);
00127         else if (motorNames[i] == "L_arm3" || motorNames[i] == "R_arm3")
00128             additionalTorque = motors[i]->cwDir * (2.1);
00129
00130         sensorsBit.push_back(tMotors[i]->sensorBit);
00131
00132         tMotorcmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8, 0,
initialDirection, 0, 4.5, additionalTorque);
00133         canManager.sendAndRecv(motors[i], frameToProcess);
00134     }
00135
00136     vector<float> midpoints = MoveTMotorToSensorLocation(motors, motorNames, sensorsBit);
00137
00138     vector<double> directions, degrees;
00139     for (long unsigned int i = 0; i < motorNames.size(); i++)
00140     {
00141         if (motorNames[i] == "L_arm2" || motorNames[i] == "R_arm2")
00142         {
00143             degrees.push_back(-30.0);
00144             midpoints[i] = midpoints[i] * (-1);
00145         }
00146         else
00147         {
00148             degrees.push_back(90.0);
00149         }
00150         directions.push_back(-motors[i]->cwDir);
00151     }
00152
00153     RotateTMotor(motors, motorNames, directions, degrees, midpoints);
00154
00155     cout << "-----moved 90 degree (Anti clock wise) -----
\n";
00156
00157     for (long unsigned int i = 0; i < motors.size(); i++)
00158     {
00159         // 모터를 멈추는 신호를 보냄
00160         tMotorcmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8, 0, 0, 0, 5, 0);
00161         if (canManager.sendAndRecv(motors[i], frameToProcess))
00162             cout << "Set " << motorNames[i] << " speed Zero.\n";
00163
00164         canManager.setSocketsTimeout(2, 0);
00165         // 현재 position을 0으로 인식하는 명령을 보냄
00166         tMotorcmd.getZero(*tMotors[i], &frameToProcess);
00167         if (canManager.sendAndRecv(motors[i], frameToProcess))
00168             cout << "Set Zero.\n";
00169         if (canManager.checkConnection(motors[i]))
00170             cout << motorNames[i] << " Position : " << motors[i]->currentPos;
00171
00172         degrees[i] = 0.0;
00173         directions[i] = motors[i]->cwDir;
00174         midpoints[i] = 0.0;
00175         if (motorNames[i] == "L_arm1" || motorNames[i] == "R_arm1")
00176         {
00177             degrees[i] = 90.0;
00178         }
00179         /*if (motorNames[i] == "L_arm2" || motorNames[i] == "R_arm2"){
00180             degrees[i] = -30.0;
00181         }*/

```

```

00182         if (motorNames[i] == "L_arm3" || motorNames[i] == "R_arm3")
00183         {
00184             degrees[i] = 90.0;
00185         }
00186     }
00187
00188     RotateTMotor(motors, motorNames, directions, degrees, midpoints);
00189 }
00190
00191 vector<float> HomeManager::MoveTMotorToSensorLocation(vector<std::shared_ptr<GenericMotor>> &motors,
vector<std::string> &motorNames, vector<int> &sensorsBit)
00192 {
00193     struct can_frame frameToProcess;
00194     vector<shared_ptr<TMotor>> tMotors;
00195     vector<float> firstPosition, secondPosition, positionDifference;
00196     vector<bool> firstSensorTriggered, TriggeredDone;
00197
00198     for (long unsigned int i = 0; i < sensorsBit.size(); i++)
00199     {
00200         tMotors.push_back(dynamic_pointer_cast<TMotor>(motors[i]));
00201         firstPosition.push_back(0.0f);
00202         secondPosition.push_back(0.0f);
00203         firstSensorTriggered.push_back(false);
00204         TriggeredDone.push_back(false);
00205
00206         cout << "Moving " << motorNames[i] << " to sensor location.\n";
00207     }
00208
00209     while (true)
00210     {
00211         // 모든 모터 센싱 완료 시 break
00212         bool done = true;
00213         for (long unsigned int i = 0; i < sensorsBit.size(); i++)
00214         {
00215             if (!TriggeredDone[i])
00216                 done = false;
00217         }
00218         if (done)
00219             break;
00220
00221         for (long unsigned int i = 0; i < sensorsBit.size(); i++)
00222         {
00223             if (!TriggeredDone[i])
00224             {
00225                 bool sensorTriggered = ((sensor.ReadVal() >> sensorsBit[i]) & 1) != 0;
00226
00227                 if (!firstSensorTriggered[i] && sensorTriggered)
00228                 {
00229                     // 첫 번째 센서 인식
00230                     firstSensorTriggered[i] = true;
00231                     canManager.checkConnection(motors[i]);
00232                     firstPosition[i] = motors[i]->currentPos;
00233                     std::cout << motorNames[i] << " first sensor position: " << firstPosition[i] << endl;
00234                 }
00235                 else if (firstSensorTriggered[i] && !sensorTriggered)
00236                 {
00237                     // 센서 인식 해제
00238                     canManager.checkConnection(motors[i]);
00239                     secondPosition[i] = motors[i]->currentPos;
00240                     std::cout << motorNames[i] << " second sensor position: " << secondPosition[i] <<
endl;
00241
00242                     TriggeredDone[i] = true;
00243                 }
00244             }
00245             else
00246             {
00247                 tMotorcmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8,
secondPosition[i], 0, motors[i]->Kp, 2.5, 0);
00248                 canManager.sendAndRecv(motors[i], frameToProcess);
00249             }
00250         }
00251     }
00252
00253     for (long unsigned int i = 0; i < sensorsBit.size(); i++)
00254     {
00255         positionDifference.push_back(abs((secondPosition[i] - firstPosition[i]) / 2.0f));
00256         cout << motorNames[i] << " midpoint position: " << positionDifference[i] << endl;
00257     }
00258
00259     return positionDifference;
00260 }
00261
00262 void HomeManager::RotateTMotor(vector<std::shared_ptr<GenericMotor>> &motors, vector<std::string>
&motorNames, vector<double> &directions, vector<double> &degrees, vector<float> &midpoints)
00263 {
00264

```

```

00265     struct can_frame frameToProcess;
00266     vector<shared_ptr<TMotor>> tMotors;
00267     vector<double> targetRadians;
00268     for (long unsigned int i = 0; i < motorNames.size(); i++)
00269     {
00270         if (degrees[i] == 0.0)
00271             return;
00272         tMotors.push_back(dynamic_pointer_cast<TMotor>(motors[i]));
00273         targetRadians.push_back((degrees[i] * M_PI / 180.0 + midpoints[i]) * directions[i]);
00274     }
00275
00276     chrono::system_clock::time_point startTime = std::chrono::system_clock::now();
00277     int totalSteps = 4000 / 5; // 4초 동안 이동 - 5ms 간격으로 나누기
00278     for (int step = 1; step <= totalSteps; ++step)
00279     {
00280         while (1)
00281         {
00282             chrono::system_clock::time_point currentTime = std::chrono::system_clock::now();
00283             if (chrono::duration_cast<chrono::microseconds>(currentTime - startTime).count() > 5000)
00284                 break;
00285         }
00286
00287         startTime = std::chrono::system_clock::now();
00288
00289         // 5ms마다 목표 위치 계산 및 프레임 전송
00290         for (long unsigned int i = 0; i < motorNames.size(); i++)
00291         {
00292             double targetPosition = targetRadians[i] * (static_cast<double>(step) / totalSteps) +
00293             motors[i]->currentPos;
00294             tmotorcmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8,
00295             targetPosition, 0, motors[i]->Kp, motors[i]->Kd, 0);
00296             canManager.sendAndRecv(motors[i], frameToProcess);
00297         }
00298         totalSteps = 500 / 5;
00299         for (int step = 1; step <= totalSteps; ++step)
00300         {
00301             while (1)
00302             {
00303                 chrono::system_clock::time_point currentTime = std::chrono::system_clock::now();
00304                 if (chrono::duration_cast<chrono::microseconds>(currentTime - startTime).count() > 5000)
00305                     break;
00306             }
00307
00308             startTime = std::chrono::system_clock::now();
00309
00310             // 5ms마다 목표 위치 계산 및 프레임 전송
00311             for (long unsigned int i = 0; i < motorNames.size(); i++)
00312             {
00313                 double targetPosition = targetRadians[i] + motors[i]->currentPos;
00314                 tmotorcmd.parseSendCommand(*tMotors[i], &frameToProcess, motors[i]->nodeId, 8,
00315                 targetPosition, 0, motors[i]->Kp, motors[i]->Kd, 0);
00316                 canManager.sendAndRecv(motors[i], frameToProcess);
00317             }
00318
00319             for (auto &motor : motors)
00320             {
00321                 canManager.checkConnection(motor);
00322             }
00323         }
00324
00325     void HomeManager::SetMaxonHome(vector<std::shared_ptr<GenericMotor>> &motors)
00326     {
00327         setMaxonMode("HMM");
00328         MaxonEnable();
00329         struct can_frame frame;
00330
00331         canManager.clearReadBuffers();
00332         canManager.setSocketsTimeout(2, 0);
00333         vector<shared_ptr<MaxonMotor>> maxonMotors;
00334         for (long unsigned int i = 0; i < motors.size(); i++)
00335         {
00336             maxonMotors.push_back(dynamic_pointer_cast<MaxonMotor>(motors[i]));
00337
00338             // Start to Move by homing method (일단은 PDO)
00339
00340             maxoncmd.getStartHoming(*maxonMotors[i], &frame);
00341             canManager.txFrame(motors[i], frame);
00342             usleep(50000);
00343         }
00344
00345         maxoncmd.getSync(&frame);
00346         canManager.txFrame(motors[0], frame);
00347         if (canManager.recvToBuff(motors[0], canManager.maxonCnt))

```

```

00349     {
00350         while (!motors[0]->recieveBuffer.empty())
00351         {
00352             frame = motors[0]->recieveBuffer.front();
00353             for (long unsigned int i = 0; i < motors.size(); i++)
00354             {
00355                 if (frame.can_id == maxonMotors[i]->rxPdoIds[0])
00356                 {
00357                     cout << "\nMaxon Homing Start!!\n";
00358                 }
00359             }
00360             motors[0]->recieveBuffer.pop();
00361         }
00362     }
00363
00364     sleep(5);
00365     // 홈 위치에 도달할 때까지 반복
00366     bool done = false;
00367     while (!done)
00368     {
00369         done = true;
00370         for (auto &motor : motors)
00371         {
00372             if (!motor->isHomed)
00373                 done = false;
00374         }
00375
00376         maxoncmd.getSync(&frame);
00377         canManager.txFrame(motors[0], frame);
00378         if (canManager.recvToBuff(motors[0], canManager.maxonCnt))
00379         {
00380             while (!motors[0]->recieveBuffer.empty())
00381             {
00382                 frame = motors[0]->recieveBuffer.front();
00383                 for (long unsigned int i = 0; i < motors.size(); i++)
00384                 {
00385                     if (frame.can_id == maxonMotors[i]->rxPdoIds[0])
00386                     {
00387                         if (frame.data[1] & 0x80) // 비트 15 확인
00388                         {
00389                             motors[i]->isHomed = true; // MaxonMotor 객체의 isHomed 속성을 true로 설정
00390                             // 'this'를 사용하여 멤버 함수 호출
00391                         }
00392                     }
00393                 }
00394                 motors[0]->recieveBuffer.pop();
00395             }
00396         }
00397         canManager.clearReadBuffers();
00398
00399         sleep(1); // 100ms 대기
00400     }
00401     setMaxonMode("CSP");
00402     MaxonDisable();
00403 }
00404
00405 void HomeManager::displayHomingStatus()
00406 {
00407     std::cout << "Homing Status of Motors:\n";
00408     for (const auto &motor_pair : motors)
00409     {
00410         std::cout << motor_pair.first << ": "
00411             << (motor_pair.second->isHomed ? "Homed" : "Not Homed") << std::endl;
00412     }
00413 }
00414
00415 void HomeManager::UpdateHomingStatus()
00416 {
00417     bool allMotorsHomed = true;
00418     for (const auto &motor_pair : motors)
00419     {
00420         if (!motor_pair.second->isHomed)
00421         {
00422             allMotorsHomed = false;
00423             break;
00424         }
00425     }
00426
00427     if (allMotorsHomed)
00428     {
00429         systemState.homeMode = HomeMode::HomeDone;
00430         systemState.main = Main::Ideal;
00431     }
00432     else
00433     {
00434         systemState.homeMode = HomeMode::NotHome;
00435     }

```

```

00436 }
00437
00438 void HomeManager::MaxonEnable()
00439 {
00440     struct can_frame frame;
00441     canManager.setSocketsTimeout(2, 0);
00442
00443     int maxonMotorCount = 0;
00444     for (const auto &motor_pair : motors)
00445     {
00446         // 각 요소가 MaxonMotor 타입인지 확인
00447         if (std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00448         {
00449             maxonMotorCount++;
00450         }
00451     }
00452
00453     // 제어 모드 설정
00454     for (const auto &motorPair : motors)
00455     {
00456         std::string name = motorPair.first;
00457         std::shared_ptr<GenericMotor> motor = motorPair.second;
00458         if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00459         {
00460
00461             maxoncmd.getOperational(*maxonMotor, &frame);
00462             canManager.txFrame(motor, frame);
00463
00464             maxoncmd.getEnable(*maxonMotor, &frame);
00465             canManager.txFrame(motor, frame);
00466
00467             maxoncmd.getSync(&frame);
00468             canManager.txFrame(motor, frame);
00469
00470             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00471             {
00472                 while (!motor->recieveBuffer.empty())
00473                 {
00474                     frame = motor->recieveBuffer.front();
00475                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00476                     {
00477                         std::cout << "Maxon Enabled \n";
00478                     }
00479                     motor->recieveBuffer.pop();
00480                 }
00481             }
00482
00483             maxoncmd.getQuickStop(*maxonMotor, &frame);
00484             canManager.txFrame(motor, frame);
00485
00486             maxoncmd.getSync(&frame);
00487             canManager.txFrame(motor, frame);
00488
00489             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00490             {
00491                 while (!motor->recieveBuffer.empty())
00492                 {
00493                     frame = motor->recieveBuffer.front();
00494                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00495                     {
00496                         std::cout << "Maxon Quick Stopped\n";
00497                     }
00498                     motor->recieveBuffer.pop();
00499                 }
00500             }
00501
00502             maxoncmd.getEnable(*maxonMotor, &frame);
00503             canManager.txFrame(motor, frame);
00504
00505             maxoncmd.getSync(&frame);
00506             canManager.txFrame(motor, frame);
00507
00508             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00509             {
00510                 while (!motor->recieveBuffer.empty())
00511                 {
00512                     frame = motor->recieveBuffer.front();
00513                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00514                     {
00515                         std::cout << "Maxon Enabled \n";
00516                     }
00517                     motor->recieveBuffer.pop();
00518                 }
00519             }
00520         }
00521     }
00522 };

```

```

00523
00524 void HomeManager::MaxonDisable()
00525 {
00526     struct can_frame frame;
00527
00528     canManager.setSocketsTimeout(0, 50000);
00529
00530     for (auto &motorPair : motors)
00531     {
00532         std::string name = motorPair.first;
00533         auto &motor = motorPair.second;
00534
00535         if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00536         {
00537             maxoncmd.getQuickStop(*maxonMotor, &frame);
00538             canManager.txFrame(motor, frame);
00539
00540             maxoncmd.getSync(&frame);
00541             canManager.txFrame(motor, frame);
00542             if (canManager.recvToBuff(motor, canManager.maxonCnt))
00543             {
00544                 while (!motor->recieveBuffer.empty())
00545                 {
00546                     frame = motor->recieveBuffer.front();
00547                     if (frame.can_id == maxonMotor->rxPdoIds[0])
00548                     {
00549
00550                         break;
00551                     }
00552                     motor->recieveBuffer.pop();
00553                 }
00554             }
00555             else
00556             {
00557                 std::cerr << "Failed to exit for motor [" << name << "]." << std::endl;
00558             }
00559         }
00560     }
00561 }
00562
00563 void HomeManager::setMaxonMode(std::string targetMode)
00564 {
00565     struct can_frame frame;
00566     canManager.setSocketsTimeout(0, 10000);
00567     for (const auto &motorPair : motors)
00568     {
00569         std::string name = motorPair.first;
00570         std::shared_ptr<GenericMotor> motor = motorPair.second;
00571         if (std::shared_ptr<MaxonMotor> maxonMotor =
00572             std::dynamic_pointer_cast<MaxonMotor>(motorPair.second))
00573         {
00574             if (targetMode == "CSV")
00575             {
00576                 maxoncmd.getCSVMode(*maxonMotor, &frame);
00577                 canManager.sendAndRecv(motor, frame);
00578             }
00579             else if (targetMode == "CST")
00580             {
00581                 maxoncmd.getCSTMode(*maxonMotor, &frame);
00582                 canManager.sendAndRecv(motor, frame);
00583             }
00584             else if (targetMode == "HMM")
00585             {
00586                 maxoncmd.getHomeMode(*maxonMotor, &frame);
00587                 canManager.sendAndRecv(motor, frame);
00588             }
00589             else if (targetMode == "CSP")
00590             {
00591                 maxoncmd.getCSPMode(*maxonMotor, &frame);
00592                 canManager.sendAndRecv(motor, frame);
00593             }
00594         }
00595     }
00596
00597 void HomeManager::FixMotorPosition(std::shared_ptr<GenericMotor> &motor)
00598 {
00599     struct can_frame frame;
00600
00601     canManager.checkConnection(motor);
00602
00603     if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00604     {
00605         tMotorcmd.parseSendCommand(*tMotor, &frame, motor->nodeId, 8, motor->currentPos, 0, 450, 1,
00606         0);
00607         if (canManager.sendAndRecv(motor, frame))
00608         {

```

```

00608         std::cout << "Position fixed for motor [" << motor->nodeId << "]." << std::endl;
00609     }
00610     else
00611     {
00612         std::cerr << "Failed to fix position for motor [" << motor->nodeId << "]." << std::endl;
00613     }
00614 }
00615 else if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00616 {
00617     maxoncmd.getTargetPosition(*maxonMotor, &frame, motor->currentPos);
00618     if (canManager.sendAndRecv(motor, frame))
00619     {
00620         std::cout << "Position fixed for motor [" << motor->nodeId << "]." << std::endl;
00621     }
00622     else
00623     {
00624         std::cerr << "Failed to fix position for motor [" << motor->nodeId << "]." << std::endl;
00625     }
00626 }
00627 }

```

## 5.13 main.cpp

```

00001 #include <thread>
00002 #include <vector>
00003 #include <iostream>
00004 #include <algorithm>
00005 #include <cctype>
00006 #include <memory>
00007 #include <map>
00008 #include <atomic>
00009
00010 #include "../include/motors/Motor.hpp"
00011 #include "../include/managers/PathManager.hpp"
00012 #include "../include/managers/CanManager.hpp"
00013 #include "../include/managers/TestManager.hpp"
00014 #include "../include/managers/HomeManager.hpp"
00015 #include "../include/tasks/DrumRobot.hpp"
00016 #include "../include/tasks/SystemState.hpp"
00017
00018
00019 using namespace std;
00020
00021 // 스레드 우선순위 설정 함수
00022 bool setThreadPriority(std::thread &th, int priority, int policy = SCHED_FIFO)
00023 {
00024     sched_param sch_params;
00025     sch_params.sched_priority = priority;
00026     if (pthread_setschedparam(th.native_handle(), policy, &sch_params))
00027     {
00028         std::cerr << "Failed to set Thread scheduling : " << std::strerror(errno) << std::endl;
00029         return false;
00030     }
00031     return true;
00032 }
00033
00034 int main(int argc, char *argv[])
00035 {
00036     // Create Share Resource
00037     SystemState systemState;
00038     std::map<std::string, std::shared_ptr<GenericMotor>> motors;
00039
00040     CanManager canManager(motors);
00041     PathManager pathManager(systemState, canManager, motors);
00042     TestManager testManager(systemState, canManager, motors);
00043     HomeManager homeManager(systemState, canManager, motors);
00044
00045     DrumRobot drumRobot(systemState, canManager, pathManager, homeManager, testManager, motors);
00046
00047     // Create Threads
00048     std::thread stateThread(&DrumRobot::stateMachine, &drumRobot);
00049     std::thread sendThread(&DrumRobot::sendLoopForThread, &drumRobot);
00050     std::thread receiveThread(&DrumRobot::recvLoopForThread, &drumRobot);
00051
00052     // Threads Priority Settings
00053     if (!setThreadPriority(sendThread, 3))
00054     {
00055         std::cerr << "Error setting priority for sendCanFrame" << std::endl;
00056         return -1;
00057     }
00058     if (!setThreadPriority(receiveThread, 2))
00059     {
00060

```

```

00061         std::cerr << "Error setting priority for receiveCanFrame" << std::endl;
00062         return -1;
00063     }
00064     if (!setThreadPriority(stateThread, 1))
00065     {
00066         std::cerr << "Error setting priority for stateMachine" << std::endl;
00067         return -1;
00068     }
00069
00070     // Wait Threads
00071     stateThread.join();
00072     sendThread.join();
00073     receiveThread.join();
00074 }

```

## 5.14 Motor.cpp

```

00001
00002 #include "../include/motors/Motor.hpp" // Include header file
00003
00004 // GenericMotor
00005
00006 void GenericMotor::clearSendBuffer()
00007 {
00008     while (!sendBuffer.empty())
00009     {
00010         sendBuffer.pop();
00011     }
00012 }
00013
00014 void GenericMotor::clearReceiveBuffer()
00015 {
00016     while (!recieveBuffer.empty())
00017     {
00018         recieveBuffer.pop();
00019     }
00020 }
00021
00022 // TMotor
00023
00024 TMotor::TMotor(uint32_t nodeId, const std::string &motorType)
00025 : GenericMotor(nodeId, interFaceName), motorType(motorType)
00026 {
00027 }
00028
00029 /*void TMotor::addTMotorData(float position, float velocity, float kp, float kd, float torqueOffset)
00030 {
00031     sendBuffer.push(TMotorData(position, velocity, kp, kd, torqueOffset));
00032 }*/
00033
00034 // maxonMotor
00035
00036 MaxonMotor::MaxonMotor(uint32_t nodeId)
00037 : GenericMotor(nodeId, interFaceName)
00038 {
00039     // canId 값 설정
00040     canSendId = 0x600 + nodeId;
00041     canReceiveId = 0x580 + nodeId;
00042 }
00043
00044 /*void MaxonMotor::addMaxonMotorData(int position){
00045     sendBuffer.push(MaxonMotorData(position));
00046 }*/
00047
00048

```

## 5.15 PathManager.cpp

```

00001 #include "../include/managers/PathManager.hpp" // 적절한 경로로 변경하세요.
00002
00003 PathManager::PathManager(SystemState &systemStateRef,
00004                           CanManager &canManagerRef,
00005                           std::map<std::string, std::shared_ptr<GenericMotor>> &motorsRef)
00006 : systemState(systemStateRef), canManager(canManagerRef), motors(motorsRef)
00007 {
00008 }
00009
00010 /*
00011     SEND BUFFER TO MOTOR
00012 */
00013
00014 void PathManager::Motors_sendBuffer()

```



```

00015 {
00016     struct can_frame frame;
00017
00018     vector<double> Pi;
00019     vector<double> Vi;
00020
00021     Pi = p.back();
00022     Vi = v.back();
00023
00024     for (auto &entry : motors)
00025     {
00026         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(entry.second))
00027         {
00028             float p_des = Pi[motor_mapping[entry.first]];
00029             float v_des = Vi[motor_mapping[entry.first]];
00030
00031             TParser.parseSendCommand(*tMotor, &frame, tMotor->nodeId, 8, p_des, v_des, tMotor->Kp,
00032 tMotor->Kd, 0.0);
00033             entry.second->sendBuffer.push(frame);
00034         }
00035         else if (std::shared_ptr<MaxonMotor> maxonMotor =
00036 std::dynamic_pointer_cast<MaxonMotor>(entry.second))
00037         {
00038             float p_des = Pi[motor_mapping[entry.first]];
00039             MParser.getTargetPosition(*maxonMotor, &frame, p_des);
00040             entry.second->sendBuffer.push(frame);
00041         }
00042     }
00043 }
00044 /*                      SYSTEM FUNCTION                      */
00045
00046 void PathManager::ApplyDir()
00047 { // CW / CCW에 따른 방향 적용
00048     for (auto &entry : motors)
00049     {
00050         shared_ptr<GenericMotor> motor = entry.second;
00051         standby[motor_mapping[entry.first]] *= motor->cwDir;
00052         backarr[motor_mapping[entry.first]] *= motor->cwDir;
00053         motor_dir[motor_mapping[entry.first]] = motor->cwDir;
00054     }
00055 }
00056
00057 vector<double> PathManager::connect(vector<double> &Q1, vector<double> &Q2, int k, int n)
00058 {
00059     vector<double> Qi;
00060     std::vector<double> A, B;
00061
00062     // Compute A and Bk
00063     for (long unsigned int i = 0; i < Q1.size(); ++i)
00064     {
00065         A.push_back(0.5 * (Q1[i] - Q2[i]));
00066         B.push_back(0.5 * (Q1[i] + Q2[i]));
00067     }
00068
00069     // Compute Qi using the provided formula
00070     for (long unsigned int i = 0; i < Q1.size(); ++i)
00071     {
00072         double val = A[i] * cos(M_PI * k / n) + B[i];
00073         Qi.push_back(val);
00074     }
00075
00076     return Qi;
00077 }
00078
00079 void PathManager::getMotorPos()
00080 {
00081     // 각 모터의 현재위치 값 불러오기 ** CheckMotorPosition 이후에 해야함(변수값을 불러오기만 해서 갱신 필요)
00082     for (auto &entry : motors)
00083     {
00084         c_MotorAngle[motor_mapping[entry.first]] = entry.second->currentPos;
00085         // 각 모터의 현재 위치 출력
00086         cout << "Motor " << entry.first << " current position: " << entry.second->currentPos << "\n";
00087     }
00088 }
00089
00090 double determinant(double mat[3][3])
00091 { // 행렬의 determinant 계산 함수
00092     return mat[0][0] * (mat[1][1] * mat[2][2] - mat[2][1] * mat[1][2]) -
00093         mat[0][1] * (mat[1][0] * mat[2][2] - mat[2][0] * mat[1][2]) +
00094         mat[0][2] * (mat[1][0] * mat[2][1] - mat[2][0] * mat[1][1]);
00095 }
00096
00097 void inverseMatrix(double mat[3][3], double inv[3][3])
00098 { // 역행렬 계산 함수
00099     double det = determinant(mat);
00100 }
00101

```

```

00102     if (det == 0)
00103     {
00104         std::cerr << "역행렬이 존재하지 않습니다." << std::endl;
00105         return;
00106     }
00107
00108     double invDet = 1.0 / det;
00109
00110     inv[0][0] = (mat[1][1] * mat[2][2] - mat[2][1] * mat[1][2]) * invDet;
00111     inv[0][1] = (mat[0][2] * mat[2][1] - mat[0][1] * mat[2][2]) * invDet;
00112     inv[0][2] = (mat[0][1] * mat[1][2] - mat[0][2] * mat[1][1]) * invDet;
00113
00114     inv[1][0] = (mat[1][2] * mat[2][0] - mat[1][0] * mat[2][2]) * invDet;
00115     inv[1][1] = (mat[0][0] * mat[2][2] - mat[0][2] * mat[2][0]) * invDet;
00116     inv[1][2] = (mat[1][0] * mat[0][2] - mat[0][0] * mat[1][2]) * invDet;
00117
00118     inv[2][0] = (mat[1][0] * mat[2][1] - mat[2][0] * mat[1][1]) * invDet;
00119     inv[2][1] = (mat[2][0] * mat[0][1] - mat[0][0] * mat[2][1]) * invDet;
00120     inv[2][2] = (mat[0][0] * mat[1][1] - mat[1][0] * mat[0][1]) * invDet;
00121 }
00122
00123 void PathManager::connect(vector<double> &P0, vector<double> &P1, vector<double> &P2, vector<double>
&V0, double t1, double t2, double t)
00124 {
00125     vector<double> V1;
00126     vector<double> p_out;
00127     vector<double> v_out;
00128     for (size_t i = 0; i < P0.size(); ++i)
00129     {
00130         if ((P1[i] - P0[i]) / (P2[i] - P1[i]) > 0)
00131             V1.push_back((P2[i] - P0[i]) / t2);
00132         else
00133             V1.push_back(0);
00134
00135         double f = P0[i];
00136         double d = 0;
00137         double e = V0[i];
00138
00139         double M[3][3] = {
00140             {20.0 * pow(t1, 2), 12.0 * t1, 6.0},
00141             {5.0 * pow(t1, 4), 4.0 * pow(t1, 3), 3.0 * pow(t1, 2)},
00142             {pow(t1, 5), pow(t1, 4), pow(t1, 3)}};
00143         double ANS[3] = {0, V1[i] - V0[i], P1[i] - P0[i] - V0[i] * t1};
00144
00145         double invM[3][3];
00146         inverseMatrix(M, invM);
00147         // Multiply the inverse of T with ANS
00148         double tem[3];
00149         for (size_t j = 0; j < 3; ++j)
00150         {
00151             tem[j] = 0;
00152             for (size_t k = 0; k < 3; ++k)
00153             {
00154                 tem[j] += invM[j][k] * ANS[k];
00155             }
00156         }
00157
00158         double a = tem[0];
00159         double b = tem[1];
00160         double c = tem[2];
00161
00162         p_out.push_back(a * pow(t, 5) + b * pow(t, 4) + c * pow(t, 3) + d * pow(t, 2) + e * t + f);
00163         v_out.push_back(5 * a * pow(t, 4) + 4 * b * pow(t, 3) + 3 * c * pow(t, 2) + 3 * d * t + e);
00164     }
00165
00166     p.push_back(p_out);
00167     v.push_back(v_out);
00168 }
00169
00170 vector<double> PathManager::fkfun()
00171 {
00172     vector<double> P;
00173     vector<double> theta(7);
00174     for (auto &motorPair : motors)
00175     {
00176         auto &name = motorPair.first;
00177         auto &motor = motorPair.second;
00178         if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00179         {
00180             theta[motor_mapping[name]] = tMotor->currentPos * tMotor->cwDir;
00181         }
00182     }
00183     double r1 = R[0], r2 = R[1], l1 = R[2], l2 = R[3];
00184     double r, l;
00185     r = r1 * sin(theta[3]) + r2 * sin(theta[3] + theta[4]);
00186     l = l1 * sin(theta[5]) + l2 * sin(theta[5] + theta[6]);
00187

```

```

00188     P.push_back(0.5 * s * cos(theta[0]) + r * cos(theta[0] + theta[1]));
00189     P.push_back(0.5 * s * sin(theta[0]) + r * sin(theta[0] + theta[1]));
00190     P.push_back(z0 - r1 * cos(theta[3]) - r2 * cos(theta[3] + theta[4]));
00191     P.push_back(0.5 * s * cos(theta[0] + M_PI) + l * cos(theta[0] + theta[2]));
00192     P.push_back(0.5 * s * sin(theta[0] + M_PI) + l * sin(theta[0] + theta[2]));
00193     P.push_back(z0 - l1 * cos(theta[5]) - l2 * cos(theta[5] + theta[6]));
00194
00195     return P;
00196 }
00197
00198 vector<double> PathManager::IKfun(vector<double> &P1, vector<double> &P2)
00199 {
00200     // 드럼위치의 중점 각도
00201     double direction = 0.0 * M_PI; //-M_PI / 3.0;
00202
00203     // 몸통과 팔이 부딪히지 않을 각도 => 36deg
00204     double differ = M_PI / 5.0;
00205
00206     vector<double> Qf(7);
00207
00208     double X1 = P1[0], Y1 = P1[1], z1 = P1[2];
00209     double X2 = P2[0], Y2 = P2[1], z2 = P2[2];
00210     double r1 = R[0], r2 = R[1], r3 = R[2], r4 = R[3];
00211
00212     vector<double> the3(1801);
00213     for (int i = 0; i < 1801; i++)
00214     { // 오른팔 들어올리는 각도 범위 :-90deg ~ 90deg
00215         the3[i] = -M_PI / 2 + (M_PI * i) / 1800;
00216     }
00217
00218     double zeta = z0 - z2;
00219
00220     double det_the0, det_the1, det_the2, det_the4, det_the5, det_the6;
00221     double the0_f, the0, the1, the2, the34, the4, the5, the6;
00222     double r, L, Lp, T;
00223     double sol;
00224     double alpha;
00225     bool first = true;
00226
00227     for (long unsigned int i = 0; i < the3.size(); i++)
00228     {
00229         det_the4 = (z0 - z1 - r1 * cos(the3[i])) / r2;
00230
00231         if (det_the4 < 1 && det_the4 > -1)
00232         {
00233             the34 = acos((z0 - z1 - r1 * cos(the3[i])) / r2);
00234             the4 = the34 - the3[i];
00235             if (the4 > 0 && the4 < M_PI * 0.75)
00236             { // 오른팔꿈치 각도 범위 : 0 ~ 135deg
00237                 r = r1 * sin(the3[i]) + r2 * sin(the34);
00238
00239                 det_the1 = (X1 * X1 + Y1 * Y1 - r * r - s * s / 4) / (s * r);
00240                 if (det_the1 < 1 && det_the1 > -1)
00241                 {
00242                     the1 = acos(det_the1);
00243                     if (the1 > 0 && the1 < (M_PI - differ))
00244                     { // 오른팔 돌리는 각도 범위 : 0 ~ 150deg
00245                         alpha = asin(X1 / sqrt(X1 * X1 + Y1 * Y1));
00246                         det_the0 = (s / 4 + (X1 * X1 + Y1 * Y1 - r * r) / s) / sqrt(X1 * X1 + Y1 *
00247 Y1);
00248
00249                         if (det_the0 < 1 && det_the0 > -1)
00250                         {
00251                             the0 = asin(det_the0) - alpha;
00252
00253                             L = sqrt(pow(X2 - 0.5 * s * cos(the0 + M_PI), 2) +
00254                                     pow(Y2 - 0.5 * s * sin(the0 + M_PI), 2));
00255                             det_the2 = (X2 + 0.5 * s * cos(the0)) / L;
00256
00257                             if (det_the2 < 1 && det_the2 > -1)
00258                             {
00259                                 the2 = acos(det_the2) - the0;
00260                                 if (the2 > differ && the2 < M_PI)
00261                                 { // 왼팔 돌리는 각도 범위 : 30deg ~ 180deg
00262                                     Lp = sqrt(L * L + zeta * zeta);
00263                                     det_the6 = (Lp * Lp - r3 * r3 - r4 * r4) / (2 * r3 * r4);
00264                                     if (det_the6 < 1 && det_the6 > -1)
00265                                     {
00266                                         the6 = acos(det_the6);
00267                                         if (the6 > 0 && the6 < M_PI * 0.75)
00268                                         { // 왼팔꿈치 각도 범위 : 0 ~ 135deg
00269                                             T = (zeta * zeta + L * L + r3 * r3 - r4 * r4) / (r3 * 2);
00270                                             det_the5 = L * L + zeta * zeta - T * T;
00271
00272                                             if (det_the5 > 0)
00273                                             {
00274                                                 sol = T * L - zeta * sqrt(L * L + zeta * zeta - T *
00275 T);

```

```

00273         sol /= (L * L + zeta * zeta);
00274         the5 = asin(sol);
00275         if (the5 > -M_PI / 4 && the5 < M_PI / 2)
00276         { // 왼팔 들어올리는 각도 범위 : -45deg ~ 90deg
00277
00278             if (first || abs(the0 - direction) < abs(the0_f -
direction))
00279             {
00280                 the0_f = the0;
00281                 Qf[0] = the0;
00282                 Qf[1] = the1;
00283                 Qf[2] = the2;
00284                 Qf[3] = the3[i];
00285                 Qf[4] = the4;
00286                 Qf[5] = the5;
00287                 Qf[6] = the6;
00288
00289                 first = false;
00290             }
00291         }
00292     }
00293 }
00294 }
00295 }
00296 }
00297 }
00298 }
00299 }
00300 }
00301 }
00302 }
00303
00304 if(first){
00305     std::cout << "IKfun Not Solved!!\n";
00306     systemState.main = Main::Pause;
00307 }
00308
00309 for (auto &entry : motors)
00310 {
00311     if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(entry.second))
00312     {
00313         Qf[motor_mapping[entry.first]] *= tMotor->cwDir;
00314     }
00315 }
00316
00317 return Qf;
00318 }
00319
00320 string trimWhitespace(const std::string &str)
00321 {
00322     size_t first = str.find_first_not_of(" \t");
00323     if (std::string::npos == first)
00324     {
00325         return str;
00326     }
00327     size_t last = str.find_last_not_of(" \t");
00328     return str.substr(first, (last - first + 1));
00329 }
00330
00331 void PathManager::getDrummingPosAndAng()
00332 {
00333     for (int j = 0; j < n_inst; ++j)
00334     { // 약기에 맞는 오/왼 손목 위치 및 손목 각도
00335         if (RA[line][j] != 0)
00336         {
00337             P1 = right_inst[j];
00338             r_wrist = wrist[j];
00339             c_R = 1;
00340         }
00341         if (LA[line][j] != 0)
00342         {
00343             P2 = left_inst[j];
00344             l_wrist = wrist[j];
00345             c_L = 1;
00346         }
00347     }
00348 }
00349
00350 void PathManager::getQ1AndQ2()
00351 {
00352     if (c_R == 0 && c_L == 0)
00353     { // 왼손 & 오른손 안침
00354         Q1 = c_MotorAngle;
00355         if (p_R == 1)
00356         {
00357             Q1[4] = Q1[4] + ElbowAngle_ready * motor_dir[4];
00358             Q1[7] = Q1[7] + WristAngle_ready * motor_dir[7];

```

```

00359     }
00360     if (p_L == 1)
00361     {
00362         Q1[6] = Q1[6] + ElbowAngle_ready * motor_dir[6];
00363         Q1[8] = Q1[8] + WristAngle_ready * motor_dir[8];
00364     }
00365     Q2 = Q1;
00366 }
00367 else
00368 {
00369     Q1 = IKfun(P1, P2);
00370     Q1.push_back(r_wrist);
00371     Q1.push_back(l_wrist);
00372     Q2 = Q1;
00373     if (c_R != 0 && c_L != 0)
00374     { // 왼손 & 오른손 침
00375         Q1[4] = Q1[4] + ElbowAngle_hit * motor_dir[4];
00376         Q1[6] = Q1[6] + ElbowAngle_hit * motor_dir[6];
00377         Q1[7] = Q1[7] + WristAngle_hit * motor_dir[7];
00378         Q1[8] = Q1[8] + WristAngle_hit * motor_dir[8];
00379     }
00380     else if (c_L != 0)
00381     { // 왼손만 침
00382         Q1[4] = Q1[4] + ElbowAngle_ready * motor_dir[4];
00383         Q2[4] = Q2[4] + ElbowAngle_ready * motor_dir[4];
00384         Q1[6] = Q1[6] + ElbowAngle_hit * motor_dir[6];
00385         Q1[7] = Q1[7] + WristAngle_ready * motor_dir[7];
00386         Q2[7] = Q2[7] + WristAngle_ready * motor_dir[7];
00387         Q1[8] = Q1[8] + WristAngle_hit * motor_dir[8];
00388     }
00389     else if (c_R != 0)
00390     { // 오른손만 침
00391         Q1[4] = Q1[4] + ElbowAngle_hit * motor_dir[4];
00392         Q1[6] = Q1[6] + ElbowAngle_ready * motor_dir[6];
00393         Q2[6] = Q2[6] + ElbowAngle_ready * motor_dir[6];
00394         Q1[7] = Q1[7] + WristAngle_hit * motor_dir[7];
00395         Q1[8] = Q1[8] + WristAngle_ready * motor_dir[8];
00396         Q2[8] = Q2[8] + WristAngle_ready * motor_dir[8];
00397     }
00398     // waist & Arm1 & Arm2는 Q1 ~ Q2 동안 계속 이동
00399     Q1[0] = (Q2[0] + c_MotorAngle[0]) / 2.0;
00400     Q1[1] = (Q2[1] + c_MotorAngle[1]) / 2.0;
00401     Q1[2] = (Q2[2] + c_MotorAngle[2]) / 2.0;
00402     Q1[3] = (Q2[3] + c_MotorAngle[3]) / 2.0;
00403     Q1[5] = (Q2[5] + c_MotorAngle[5]) / 2.0;
00404 }
00405 }
00406
00407 void PathManager::getQ3AndQ4()
00408 {
00409     if (c_R == 0 && c_L == 0)
00410     { // 왼손 & 오른손 안침
00411         Q3 = Q2;
00412         if (p_R == 1)
00413         {
00414             Q3[4] = Q3[4] + ElbowAngle_ready * motor_dir[4];
00415             Q3[7] = Q3[7] + WristAngle_ready * motor_dir[7];
00416         }
00417         if (p_L == 1)
00418         {
00419             Q3[6] = Q3[6] + ElbowAngle_ready * motor_dir[6];
00420             Q3[8] = Q3[8] + WristAngle_ready * motor_dir[8];
00421         }
00422         Q4 = Q3;
00423     }
00424     else
00425     {
00426         Q3 = IKfun(P1, P2);
00427         Q3.push_back(r_wrist);
00428         Q3.push_back(l_wrist);
00429         Q4 = Q3;
00430         if (c_R != 0 && c_L != 0)
00431         { // 왼손 & 오른손 침
00432             Q3[4] = Q3[4] + ElbowAngle_hit * motor_dir[4];
00433             Q3[6] = Q3[6] + ElbowAngle_hit * motor_dir[6];
00434             Q3[7] = Q3[7] + WristAngle_hit * motor_dir[7];
00435             Q3[8] = Q3[8] + WristAngle_hit * motor_dir[8];
00436         }
00437         else if (c_L != 0)
00438         { // 왼손만 침
00439             Q3[4] = Q3[4] + ElbowAngle_ready * motor_dir[4];
00440             Q4[4] = Q4[4] + ElbowAngle_ready * motor_dir[4];
00441             Q3[6] = Q3[6] + ElbowAngle_hit * motor_dir[6];
00442             Q3[7] = Q3[7] + WristAngle_ready * motor_dir[7];
00443             Q4[7] = Q4[7] + WristAngle_ready * motor_dir[7];
00444             Q3[8] = Q3[8] + WristAngle_hit * motor_dir[8];
00445         }
00446     }

```

```

00446         else if (c_R != 0)
00447         { // 오른손만 침
00448             Q3[4] = Q3[4] + ElbowAngle_hit * motor_dir[4];
00449             Q3[6] = Q3[6] + ElbowAngle_ready * motor_dir[6];
00450             Q4[6] = Q4[6] + ElbowAngle_ready * motor_dir[6];
00451             Q3[7] = Q3[7] + WristAngle_hit * motor_dir[7];
00452             Q3[8] = Q3[8] + WristAngle_ready * motor_dir[8];
00453             Q4[8] = Q4[8] + WristAngle_ready * motor_dir[8];
00454         }
00455         // waist & Arm1 & Arm2는 Q3 ~ Q4 동안 계속 이동
00456         Q3[0] = (Q4[0] + Q2[0]) / 2.0;
00457         Q3[1] = (Q4[1] + Q2[1]) / 2.0;
00458         Q3[2] = (Q4[2] + Q2[2]) / 2.0;
00459         Q3[3] = (Q4[3] + Q2[3]) / 2.0;
00460         Q3[5] = (Q4[5] + Q2[5]) / 2.0;
00461     }
00462 }
00463
00465 /*                                     MAKE PATH                                     */
00467
00468 void PathManager::GetDrumPositoin()
00469 {
00470     getMotorPos();
00471
00472     ifstream inputFile("../include/managers/rT.txt");
00473
00474     if (!inputFile.is_open())
00475     {
00476         cerr << "Failed to open the file."
00477              << "\n";
00478     }
00479
00480     // Read data into a 2D vector
00481     vector<vector<double>> inst_xyz(6, vector<double>(8, 0));
00482
00483     for (int i = 0; i < 6; ++i)
00484     {
00485         for (int j = 0; j < 8; ++j)
00486         {
00487             inputFile >> inst_xyz[i][j];
00488             if (i == 1 || i == 4)
00489                 inst_xyz[i][j] = inst_xyz[i][j] * 1.0;
00490         }
00491     }
00492
00493     // Extract the desired elements
00494     vector<double> right_B = {0, 0, 0};
00495     vector<double> right_S;
00496     vector<double> right_FT;
00497     vector<double> right_MT;
00498     vector<double> right_HT;
00499     vector<double> right_HH;
00500     vector<double> right_R;
00501     vector<double> right_RC;
00502     vector<double> right_LC;
00503
00504     for (int i = 0; i < 3; ++i)
00505     {
00506         right_S.push_back(inst_xyz[i][0]);
00507         right_FT.push_back(inst_xyz[i][1]);
00508         right_MT.push_back(inst_xyz[i][2]);
00509         right_HT.push_back(inst_xyz[i][3]);
00510         right_HH.push_back(inst_xyz[i][4]);
00511         right_R.push_back(inst_xyz[i][5]);
00512         right_RC.push_back(inst_xyz[i][6]);
00513         right_LC.push_back(inst_xyz[i][7]);
00514     }
00515
00516     vector<double> left_B = {0, 0, 0};
00517     vector<double> left_S;
00518     vector<double> left_FT;
00519     vector<double> left_MT;
00520     vector<double> left_HT;
00521     vector<double> left_HH;
00522     vector<double> left_R;
00523     vector<double> left_RC;
00524     vector<double> left_LC;
00525
00526     for (int i = 3; i < 6; ++i)
00527     {
00528         left_S.push_back(inst_xyz[i][0]);
00529         left_FT.push_back(inst_xyz[i][1]);
00530         left_MT.push_back(inst_xyz[i][2]);
00531         left_HT.push_back(inst_xyz[i][3]);
00532         left_HH.push_back(inst_xyz[i][4]);
00533         left_R.push_back(inst_xyz[i][5]);
00534         left_RC.push_back(inst_xyz[i][6]);

```

```

00535         left_LC.push_back(inst_xyz[i][7]);
00536     }
00537
00538     // Combine the elements into right_inst and left_inst
00539     right_inst = {right_B, right_RC, right_R, right_S, right_HH, right_HH, right_FT, right_MT,
00540 right_LC, right_HT};
00541     left_inst = {left_B, left_RC, left_R, left_S, left_HH, left_HH, left_FT, left_MT, left_LC,
00542 left_HT};
00543 }
00544 void PathManager::GetMusicSheet()
00545 {
00546     map<string, int> instrument_mapping = {
00547         {"0", 10}, {"1", 3}, {"2", 6}, {"3", 7}, {"4", 9}, {"5", 4}, {"6", 2}, {"7", 1}, {"8", 8},
00548 {"11", 3}, {"51", 3}, {"61", 3}, {"71", 3}, {"81", 3}, {"91", 3}};
00549
00550     string score_path = "../include/managers/codeConfession.txt";
00551     ifstream file(score_path);
00552     if (!file.is_open())
00553         cerr << "Error opening file." << endl;
00554
00555     string line;
00556     int lineIndex = 0;
00557     while (getline(file, line))
00558     {
00559         istringstream iss(line);
00560         string item;
00561         vector<string> columns;
00562         while (getline(iss, item, '\t'))
00563         {
00564             item = trimWhitespace(item);
00565             columns.push_back(item);
00566         }
00567
00568         if (lineIndex == 0)
00569         { // 첫번째 행엔 bpm에 대한 정보
00570             bpm = stod(columns[0].substr(4));
00571             cout << "bpm = " << bpm << "\n";
00572         }
00573         else
00574         {
00575             vector<int> inst_arr_R(10, 0), inst_arr_L(10, 0);
00576             time_arr.push_back(stod(columns[1]) * 100 / bpm);
00577
00578             if (columns[2] != "0")
00579                 inst_arr_R[instrument_mapping[columns[2]]] = 1;
00580             if (columns[3] != "0")
00581                 inst_arr_L[instrument_mapping[columns[3]]] = 1;
00582
00583             RF.push_back(stoi(columns[6]) == 1 ? 1 : 0);
00584             LF.push_back(stoi(columns[7]) == 2 ? 1 : 0);
00585
00586             RA.push_back(inst_arr_R);
00587             LA.push_back(inst_arr_L);
00588         }
00589
00590         lineIndex++;
00591     }
00592     file.close();
00593
00594     total = RF.size();
00595 }
00596
00597 void PathManager::PathLoopTask()
00598 {
00599     // 연주 처음 시작할 때 Q1, Q2 계산
00600     if (line == 0)
00601     {
00602         c_R = 0;
00603         c_L = 0;
00604
00605         getDrummingPosAndAng();
00606         getQ1AndQ2();
00607
00608         p_R = c_R;
00609         p_L = c_L;
00610
00611         line++;
00612
00613         p.push_back(c_MotorAngle);
00614         v.push_back({0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0});
00615     }
00616
00617     c_R = 0;
00618     c_L = 0;

```

```

00620
00621     getDrummingPosAndAng();
00622     getQ3AndQ4();
00623
00624     p_R = c_R;
00625     p_L = c_L;
00626
00627     double t1 = time_arr[line - 1];
00628     double t2 = time_arr[line];
00629     double t = 0.005;
00630     int n = round((t1 / 2) / t);
00631     vector<double> V0 = v.back();
00632     for (int i = 0; i < n; i++)
00633     {
00634         icoonnect(c_MotorAngle, Q1, Q2, V0, t1 / 2, t1, t * i);
00635         Motors_sendBuffer();
00636     }
00637     V0 = v.back();
00638     for (int i = 0; i < n; i++)
00639     {
00640         icoonnect(Q1, Q2, Q3, V0, t1 / 2, (t1 + t2) / 2, t * i);
00641         Motors_sendBuffer();
00642     }
00643     c_MotorAngle = p.back();
00644     Q1 = Q3;
00645     Q2 = Q4;
00646 }
00647
00648 void PathManager::GetArr(vector<double> &arr)
00649 {
00650     cout << "Get Array...\n";
00651     struct can_frame frame;
00652
00653     vector<double> Qi;
00654     vector<vector<double>> q_setting;
00655
00656     getMotorPos();
00657
00658     int n = 800;
00659     for (int k = 0; k < n; ++k)
00660     {
00661         // Make GetBack Array
00662         Qi = connect(c_MotorAngle, arr, k, n);
00663         q_setting.push_back(Qi);
00664
00665         // Send to Buffer
00666         for (auto &entry : motors)
00667         {
00668             if (std::shared_ptr<TMotor> motor = std::dynamic_pointer_cast<TMotor>(entry.second))
00669             {
00670                 float p_des = Qi[motor_mapping[entry.first]];
00671                 TParser.parseSendCommand(*motor, &frame, motor->nodeId, 8, p_des, 0, motor->Kp,
00672 motor->Kd, 0.0);
00673                 entry.second->sendBuffer.push(frame);
00674             }
00675             else if (std::shared_ptr<MaxonMotor> motor =
00676 std::dynamic_pointer_cast<MaxonMotor>(entry.second))
00677             {
00678                 float p_des = Qi[motor_mapping[entry.first]];
00679                 MParser.getTargetPosition(*motor, &frame, p_des);
00680                 entry.second->sendBuffer.push(frame);
00681             }
00682         }
00683     }
00684     c_MotorAngle = Qi;
00685 }

```

## 5.16 Sensor.cpp

```

00001 #include "../include/usbio/SenSor.hpp"
00002
00003 Sensor::Sensor()
00004 {
00005 }
00006
00007 Sensor::~Sensor()
00008 {
00009 }
00010
00011 DWORD Sensor::ReadVal()
00012 {
00013

```



```

00014
00015     USBIO_DI_ReadValue(DevNum, &DIValue);
00016
00017     return DIValue;
00018 }
00019
00020 bool Sensor::OpenDeviceUntilSuccess()
00021 {
00022     printf("USB I/O Library Version : %s\n", USBIO_GetLibraryVersion());
00023
00024     while (true)
00025     {
00026         res = USBIO_OpenDevice(DeviceID, BoardID, &DevNum);
00027         if (res == 0)
00028         {
00029             printf("Device opened successfully.\n");
00030             return true;
00031         }
00032         else
00033         {
00034             printf("Open Device failed! Error : 0x%x. Retrying...\n", res);
00035             usleep(100000); // 잠시 대기 후 재시도
00036         }
00037     }
00038
00039     printf("Demo usbio_di DevNum = %d\n", DevNum);
00040
00041     USBIO_ModuleName(DevNum, module_name);
00042
00043     USBIO_GetDITotal(DevNum, &total_di);
00044     printf("%s DI number: %d\n\n", module_name, total_di);
00045 }
00046
00047 void Sensor::closeDevice()
00048 {
00049     res = USBIO_CloseDevice(DevNum);
00050
00051     if (res)
00052     {
00053         printf("close %s with Board ID %d failed! Error : %d\n", module_name, BoardID, res);
00054     }
00055 }
00056
00057 void Sensor::connect()
00058 {
00059     printf("USB I/O Library Version : %s\n", USBIO_GetLibraryVersion());
00060
00061     res = USBIO_OpenDevice(DeviceID, BoardID, &DevNum);
00062     if (res == 0)
00063     {
00064         printf("Device opened successfully.\n");
00065         connected = true;
00066     }
00067     else
00068     {
00069         printf("Open Device failed! Error : 0x%x. Retrying...\n", res);
00070         connected = false;
00071     }
00072
00073     printf("Demo usbio_di DevNum = %d\n", DevNum);
00074
00075     USBIO_ModuleName(DevNum, module_name);
00076
00077     USBIO_GetDITotal(DevNum, &total_di);
00078     printf("%s DI number: %d\n\n", module_name, total_di);
00079 }

```

## 5.17 TestManager.cpp

```

00001 #include "../include/managers/TestManager.hpp" // 적절한 경로로 변경하세요.
00002
00003 using namespace std;
00004
00005 TestManager::TestManager(SystemState &systemStateRef, CanManager &canManagerRef, std::map<std::string,
00006     std::shared_ptr<GenericMotor> &motorsRef)
00007     : systemState(systemStateRef), canManager(canManagerRef), motors(motorsRef)
00008 {
00009 }
00010 void TestManager::mainLoop()
00011 {
00012     int choice;
00013     canManager.checkAllMotors();

```

```

00014     setMaxonMode("CSP");
00015     while (systemState.main == Main::Tune)
00016     {
00017         // 사용자에게 선택지 제공
00018         std::cout << "1: MultiMode\n2: SingleMode\n3: StickMode\n4: Exit\n";
00019         std::cout << "Select mode (1-4): ";
00020         std::cin >> choice;
00021
00022         // 선택에 따라 testMode 설정
00023         switch (choice)
00024         {
00025             case 1:
00026                 multiTestLoop();
00027                 break;
00028             case 2:
00029                 TuningLoopTask();
00030                 break;
00031             case 3:
00032                 TestStickLoop();
00033                 break;
00034             case 4:
00035                 systemState.main = Main::Ideal;
00036                 break;
00037             default:
00038                 std::cout << "Invalid choice. Please try again.\n";
00039                 continue;
00040         }
00041     }
00042 }
00043
00044 /*                                     Multi Test Mode                                     */
00045
00046 void TestManager::mkArr(vector<string> &motorName, int time, int cycles, int LnR, double amp)
00047 {
00048     struct can_frame frame;
00049
00050     int Kp_fixed = 450;
00051     double Kd_fixed = 4.5;
00052     map<string, bool> TestMotor;
00053     if (LnR == 0) // 양쪽 다 고정
00054     {
00055         for (auto &motorname : motorName)
00056         {
00057             TestMotor[motorname] = false;
00058         }
00059     }
00060     else if (LnR == 1) // 오른쪽만 Test
00061     {
00062         for (auto &motorname : motorName)
00063         {
00064             if (motorname[0] == 'L')
00065                 TestMotor[motorname] = false;
00066             else
00067                 TestMotor[motorname] = true;
00068         }
00069     }
00070     else if (LnR == 2) // 왼쪽만 Test
00071     {
00072         for (auto &motorname : motorName)
00073         {
00074             if (motorname[0] == 'R')
00075                 TestMotor[motorname] = false;
00076             else
00077                 TestMotor[motorname] = true;
00078         }
00079     }
00080     else if (LnR == 3) // 양쪽 다 Test
00081     {
00082         for (auto &motorname : motorName)
00083         {
00084             TestMotor[motorname] = true;
00085         }
00086     }
00087
00088     amp = amp / 180.0 * M_PI; // Degree -> Radian 변경
00089     for (const auto &motorname : motorName)
00090     {
00091         if (motors.find(motorname) != motors.end())
00092         {
00093             std::cout << motorname << " ";
00094             if (TestMotor[motorname])
00095             { // Test 하는 모터
00096                 std::cout << "Move\n";
00097                 InputData[0] += motorname + ", ";
00098                 if (std::shared_ptr<TMotor> tMotor =
00099                     std::dynamic_pointer_cast<TMotor>(motors[motorname]))
00100                 {
00101

```

```

00102         int kp = tMotor->Kp;
00103         double kd = tMotor->Kd;
00104
00105         for (int c = 0; c < cycles; c++)
00106         {
00107             for (int i = 0; i < time; i++)
00108             {
00109                 float val = tMotor->currentPos + (1.0 - cos(2.0 * M_PI * i / time)) / 2 *
amp * tMotor->cwDir;
00110                 tMotor->sendBuffer.push(frame);
00111                 InputData[time * c + i + 1] += to_string(val) + ",";
00112             }
00113         }
00114     }
00115 }
00116 else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[motorname]))
00117 {
00118     for (int c = 0; c < cycles; c++)
00119     {
00120         for (int i = 0; i < time; i++)
00121         {
00122             float val = maxonMotor->currentPos + (1.0 - cos(2.0 * M_PI * i / time)) /
2 * amp * maxonMotor->cwDir;
00123             maxoncmd.getTargetPosition(*maxonMotor, &frame, val);
00124             maxonMotor->sendBuffer.push(frame);
00125             InputData[time * c + i + 1] += to_string(val) + ",";
00126         }
00127     }
00128 }
00129 }
00130 else
00131 { // Fixed 하는 모터
00132     std::cout << "Fixed\n";
00133     InputData[0] += motorname + ",";
00134     if (std::shared_ptr<TMotor> tMotor =
std::dynamic_pointer_cast<TMotor>(motors[motorname]))
00135     {
00136         for (int c = 0; c < cycles; c++)
00137         {
00138             for (int i = 0; i < time; i++)
00139             {
00140                 float val = tMotor->currentPos;
00141                 tMotor->sendBuffer.push(frame);
00142                 InputData[time * c + i + 1] += to_string(val) + ",";
00143             }
00144         }
00145     }
00146 }
00147 else if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[motorname]))
00148 {
00149     for (int c = 0; c < cycles; c++)
00150     {
00151         for (int i = 0; i < time; i++)
00152         {
00153             float val = maxonMotor->currentPos;
00154             maxoncmd.getTargetPosition(*maxonMotor, &frame, val);
00155             maxonMotor->sendBuffer.push(frame);
00156             InputData[time * c + i + 1] += to_string(val) + ",";
00157         }
00158     }
00159 }
00160 }
00161 }
00162 }
00163 }
00164 }
00165 void TestManager::SendLoop()
00166 {
00167     std::cout << "Settig...\n";
00168     struct can_frame frameToProcess;
00169     std::string maxonCanInterface;
00170     std::shared_ptr<GenericMotor> virtualMaxonMotor;
00171
00172     int maxonMotorCount = 0;
00173     for (const auto &motor_pair : motors)
00174     {
00175         // 각 요소가 MaxonMotor 타입인지 확인
00176         if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motor_pair.second))
00177         {
00178             maxonMotorCount++;
00179             maxonCanInterface = maxonMotor->interFaceName;
00180             virtualMaxonMotor = motor_pair.second;

```

```

00181     }
00182 }
00183 chrono::system_clock::time_point external = std::chrono::system_clock::now();
00184
00185 bool allBuffersEmpty;
00186 do
00187 {
00188     allBuffersEmpty = true;
00189     for (const auto &motor_pair : motors)
00190     {
00191         if (!motor_pair.second->sendBuffer.empty())
00192         {
00193             allBuffersEmpty = false;
00194             break;
00195         }
00196     }
00197
00198     if (!allBuffersEmpty)
00199     {
00200         chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00201         chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
external);
00202
00203         if (elapsed_time.count() >= 5000) // 5ms
00204         {
00205             external = std::chrono::system_clock::now();
00206
00207             for (auto &motor_pair : motors)
00208             {
00209                 shared_ptr<GenericMotor> motor = motor_pair.second;
00210                 canManager.sendFromBuff(motor);
00211             }
00212
00213             if (maxonMotorCount != 0)
00214             {
00215                 maxoncmd.getSync(&frameToProcess);
00216                 canManager.txFrame(virtualMaxonMotor, frameToProcess);
00217             }
00218
00219             // canManager.readFramesFromAllSockets();
00220             // canManager.distributeFramesToMotors();
00221         }
00222     }
00223 } while (!allBuffersEmpty);
00224 canManager.clearReadBuffers();
00225 }
00226
00227 void TestManager::multiTestLoop()
00228 {
00229     string userInput;
00230     vector<double> c_deg;
00231     double t = 4.0;
00232     int cycles = 1;
00233     int type = 0b00001;
00234     int LnR = 1;
00235     double amplitude[5] = {30.0, 30.0, 30.0, 30.0, 30.0};
00236
00237     while (systemState.main == Main::Tune)
00238     {
00239         int result = system("clear");
00240         if (result != 0)
00241         {
00242             cerr << "Error during clear screen" << std::endl;
00243         }
00244
00245         string typeDescription;
00246         if ((type | 0b11110) == 0b11111)
00247         {
00248             typeDescription += "Arm3 Turn, ";
00249         }
00250         if ((type | 0b11101) == 0b11111)
00251         {
00252             typeDescription += "Arm2 Turn, ";
00253         }
00254         if ((type | 0b11011) == 0b11111)
00255         {
00256             typeDescription += "Arm1 Turn, ";
00257         }
00258         if ((type | 0b10111) == 0b11111)
00259         {
00260             typeDescription += "Waist Turn, ";
00261         }
00262         if ((type | 0b01111) == 0b11111)
00263         {
00264             typeDescription += "Wrist Turn, ";
00265         }
00266

```

```

00267         std::string LeftAndRight;
00268         if (LnR == 1)
00269         {
00270             LeftAndRight = "« Right move »\n";
00271         }
00272         else if (LnR == 2)
00273         {
00274             LeftAndRight = "« Left move »\n";
00275         }
00276         else if (LnR == 3)
00277         {
00278             LeftAndRight = "« Left and Right move »\n";
00279         }
00280
00281         std::cout <<
00282         "-----\n";
00283         std::cout << "< Current Position >\n";
00284         for (auto &motor : motors)
00285         {
00286             c_deg.push_back(motor.second->currentPos * motor.second->cwDir / M_PI * 180);
00287             std::cout << motor.first << " : " << motor.second->currentPos * motor.second->cwDir / M_PI *
180 << "deg\n";
00288         }
00289         std::cout << "\n"
00290         << LeftAndRight;
00291         std::cout << "Type : " << typeDescription << "\n";
00292         std::cout << "Period : " << t << "\n";
00293         std::cout << "Cycles : " << cycles << "\n";
00294         std::cout << "\nAmplitude :\n";
00295         std::cout << "1) Arm3 = " << amplitude[0] << "\n";
00296         std::cout << "2) Arm2 = " << amplitude[1] << "\n";
00297         std::cout << "3) Arm1 = " << amplitude[2] << "\n";
00298         std::cout << "4) Waist = " << amplitude[3] << "\n";
00299         std::cout << "5) Wrist = " << amplitude[4] << "\n";
00300         std::cout << "-----\n";
00301         std::cout << "[Commands]\n";
00302         std::cout << "[d] : Left and Right | [t] : Type | [p] : Period | [c] : Cycles\n"
00303         << "[a] : Amplitude | [kp] : Kp | [kd] : Kd | [m] : move | [r] : run | [e] : Exit\n";
00304         std::cout << "Enter Command: ";
00305         std::cin >> userInput;
00306
00307         if (userInput[0] == 'e')
00308         {
00309             systemState.main = Main::Ideal;
00310             break;
00311         }
00312         else if (userInput[0] == 'd')
00313         {
00314             std::cout << "\nEnter Desired Direction\n";
00315             std::cout << "1: Right Move\n";
00316             std::cout << "2: Left Move\n";
00317             std::cout << "3: Left and Right Move\n";
00318             std::cout << "\nEnter Path Type (1 or 2 or 3): ";
00319             std::cin >> LnR;
00320         }
00321         else if (userInput[0] == 't')
00322         {
00323             int num;
00324             std::cout << "\nEnter Desired Type\n";
00325             std::cout << "1: Arm3 Turn ON/OFF\n";
00326             std::cout << "2: Arm2 Turn ON/OFF\n";
00327             std::cout << "3: Arm1 Turn ON/OFF\n";
00328             std::cout << "4: Waist Turn ON/OFF\n";
00329             std::cout << "5: Wrist Turn ON/OFF\n";
00330             std::cout << "\nEnter Path Type (1 or 2 or 3 or 4 or 5): ";
00331             std::cin >> num;
00332
00333             if (num == 1)
00334             {
00335                 type = type ^ 0b000001;
00336             }
00337             else if (num == 2)
00338             {
00339                 type = type ^ 0b000010;
00340             }
00341             else if (num == 3)
00342             {
00343                 type = type ^ 0b000100;
00344             }
00345             else if (num == 4)
00346             {
00347                 type = type ^ 0b01000;
00348             }
00349             else if (num == 5)
00350             {

```

```

00351         type = type ^ 0b10000;
00352     }
00353 }
00354 else if (userInput[0] == 'p')
00355 {
00356     std::cout << "\nEnter Desired Period : ";
00357     std::cin >> t;
00358 }
00359 else if (userInput[0] == 'c')
00360 {
00361     std::cout << "\nEnter Desired Cycles : ";
00362     std::cin >> cycles;
00363 }
00364 else if (userInput[0] == 'a')
00365 {
00366     int input;
00367     std::cout << "\n[Select Motor]\n";
00368     std::cout << "1: Arm3\n";
00369     std::cout << "2: Arm2\n";
00370     std::cout << "3: Arm1\n";
00371     std::cout << "4: Waist\n";
00372     std::cout << "5: Wrist\n";
00373     std::cout << "\nEnter Desired Motor : ";
00374     std::cin >> input;
00375
00376     std::cout << "\nEnter Desired Amplitude(degree) : ";
00377     std::cin >> amplitude[input - 1];
00378 }
00379 else if (userInput == "kp")
00380 {
00381     char input;
00382     int kp;
00383     std::cout << "\n[Select Motor]\n";
00384     std::cout << "1: Arm3\n";
00385     std::cout << "2: Arm2\n";
00386     std::cout << "3: Arm1\n";
00387     std::cout << "4: Waist\n";
00388     std::cout << "\nEnter Desired Motor : ";
00389     std::cin >> input;
00390
00391     if (input == '1')
00392     {
00393         std::cout << "Arm3's Kp : " << motors["R_arm3"]->Kp << "\n";
00394         std::cout << "Enter Arm3's Desired Kp : ";
00395         std::cin >> kp;
00396         motors["R_arm3"]->Kp = kp;
00397         // motors["L_arm3"]->Kp = kp;
00398     }
00399     else if (input == '2')
00400     {
00401         std::cout << "Arm2's Kp : " << motors["R_arm2"]->Kp << "\n";
00402         std::cout << "Enter Arm2's Desired Kp : ";
00403         std::cin >> kp;
00404         motors["R_arm2"]->Kp = kp;
00405         motors["L_arm2"]->Kp = kp;
00406     }
00407     else if (input == '3')
00408     {
00409         std::cout << "Arm1's Kp : " << motors["R_arm1"]->Kp << "\n";
00410         std::cout << "Enter Arm1's Desired Kp : ";
00411         std::cin >> kp;
00412         motors["R_arm1"]->Kp = kp;
00413         motors["L_arm1"]->Kp = kp;
00414     }
00415     else if (input == '4')
00416     {
00417         std::cout << "Waist's Kp : " << motors["waist"]->Kp << "\n";
00418         std::cout << "Enter Waist's Desired Kp : ";
00419         std::cin >> kp;
00420         motors["waist"]->Kp = kp;
00421     }
00422 }
00423 else if (userInput == "kd")
00424 {
00425     char input;
00426     int kd;
00427     std::cout << "\n[Select Motor]\n";
00428     std::cout << "1: Arm3\n";
00429     std::cout << "2: Arm2\n";
00430     std::cout << "3: Arm1\n";
00431     std::cout << "4: Waist\n";
00432     std::cout << "\nEnter Desired Motor : ";
00433     std::cin >> input;
00434
00435     if (input == '1')
00436     {
00437         std::cout << "Arm3's Kd : " << motors["R_arm3"]->Kd << "\n";

```

```

00438         std::cout << "Enter Arm3's Desired Kd : ";
00439         std::cin >> kd;
00440         motors["R_arm3"]->Kd = kd;
00441         // motors["L_arm3"]->Kd = kd;
00442     }
00443     else if (input == '2')
00444     {
00445         std::cout << "Arm2's Kd : " << motors["R_arm2"]->Kd << "\n";
00446         std::cout << "Enter Arm2's Desired Kd : ";
00447         std::cin >> kd;
00448         motors["R_arm2"]->Kd = kd;
00449         motors["L_arm2"]->Kd = kd;
00450     }
00451     else if (input == '3')
00452     {
00453         std::cout << "Arm1's Kd : " << motors["R_arm1"]->Kd << "\n";
00454         std::cout << "Enter Arm1's Desired Kd : ";
00455         std::cin >> kd;
00456         motors["R_arm1"]->Kd = kd;
00457         motors["L_arm1"]->Kd = kd;
00458     }
00459     else if (input == '4')
00460     {
00461         std::cout << "Waist's Kd : " << motors["waist"]->Kd << "\n";
00462         std::cout << "Enter Waist's Desired Kd : ";
00463         std::cin >> kd;
00464         motors["waist"]->Kd = kd;
00465     }
00466     /*
00467     else if (userInput[0] == 'm')
00468     {
00469         while (true)
00470         {
00471             string input;
00472             double deg;
00473             cout << "\n[Move to]\n";
00474             int i = 0;
00475             for (auto &motor : motors)
00476             {
00477                 cout << i + 1 << " - " << motor.first << " : " << c_deg[i] << "deg\n";
00478                 i++;
00479             }
00480             cout << "m - Move\n";
00481             cout << "e - Exit\n";
00482             cout << "\nEnter Desired Option : ";
00483             cin >> input;
00484
00485             if (input[0] == 'e')
00486             {
00487                 break;
00488             }
00489             else if (input[0] == 'm')
00490             {
00491                 // 움직이는 함수 작성
00492                 // 나중에 이동할 위치 값 : c_deg * motor.second->cwDir / 180 * M_PI
00493                 break;
00494             }
00495             else
00496             {
00497                 cout << "\nEnter Desired Degree : ";
00498                 cin >> deg;
00499                 c_deg[stoi(input) - 1] = deg;
00500             }
00501         }
00502     }*/
00503     else if (userInput[0] == 'r')
00504     {
00505         TestArr(t, cycles, type, LnR, amplitude);
00506     }
00507 }
00508 }
00509
00510 void TestManager::TestArr(double t, int cycles, int type, int LnR, double amp[])
00511 {
00512     std::cout << "Test Start!!\n";
00513
00514     int time = t / 0.005;
00515     std::vector<std::string> SmotorName;
00516     InputData.clear();
00517     InputData.resize(time * cycles + 1);
00518
00519     SmotorName = {"waist"};
00520     if ((type | 0b10111) == 0b11111) // Turn Waist
00521         mkArr(SmotorName, time, cycles, LnR, amp[3]);
00522     else
00523         mkArr(SmotorName, time, cycles, 0, 0);
00524 }

```

```

00525     SmotorName = {"R_arm1", "L_arm1"};
00526     if ((type | 0b11011) == 0b11111) // Turn Arm1
00527         mkArr(SmotorName, time, cycles, LnR, amp[2]);
00528     else
00529         mkArr(SmotorName, time, cycles, 0, 0);
00530
00531     SmotorName = {"R_arm2", "L_arm2"};
00532     if ((type | 0b11101) == 0b11111) // Turn Arm2
00533         mkArr(SmotorName, time, cycles, LnR, amp[1]);
00534     else
00535         mkArr(SmotorName, time, cycles, 0, 0);
00536
00537     SmotorName = {"R_arm3", "L_arm3"};
00538     if ((type | 0b11110) == 0b11111) // Turn Arm3
00539         mkArr(SmotorName, time, cycles, LnR, amp[0]);
00540     else
00541         mkArr(SmotorName, time, cycles, 0, 0);
00542
00543     SmotorName = {"R_wrist", "L_wrist", "maxonForTest"};
00544     if ((type | 0b01111) == 0b11111) // Turn Wrist
00545         mkArr(SmotorName, time, cycles, LnR, amp[4]);
00546     else
00547         mkArr(SmotorName, time, cycles, 0, 0);
00548
00549     // TXT 파일 열기
00550     string FileNameIn = "../../READ/test_in.txt";
00551     ofstream csvFileIn(FileNameIn);
00552     if (!csvFileIn.is_open())
00553     {
00554         std::cerr << "Error opening TXT file." << std::endl;
00555     }
00556
00557     // TXT 파일 입력
00558     for (auto &data : InputData)
00559     {
00560         csvFileIn << data << "\n";
00561     }
00562
00563     // CSV 파일 닫기
00564     csvFileIn.close();
00565     std::cout << "연주 txt_InData 파일이 생성되었습니다." << FileNameIn << std::endl;
00566
00567     SendLoop();
00568
00569     parse_and_save_to_csv("../../READ/test_out.txt");
00570 }
00571
00572 void TestManager::parse_and_save_to_csv(const std::string &csv_file_name)
00573 {
00574     // CSV 파일 열기. 파일이 없으면 새로 생성됩니다.
00575     std::ofstream ofs(csv_file_name, std::ios::app);
00576     if (!ofs.is_open())
00577     {
00578         std::cerr << "Failed to open or create the CSV file: " << csv_file_name << std::endl;
00579         return;
00580     }
00581
00582     // 파일이 새로 생성되었으면 CSV 헤더를 추가
00583     ofs.seekp(0, std::ios::end);
00584     if (ofs.tellp() == 0)
00585     {
00586         ofs << "CAN_ID,p_act,tff_des,tff_act\n";
00587     }
00588
00589     // 각 모터에 대한 처리
00590     for (const auto &pair : motors)
00591     {
00592         auto &motor = pair.second;
00593         if (!motor->recieveBuffer.empty())
00594         {
00595             can_frame frame = motor->recieveBuffer.front();
00596             motor->recieveBuffer.pop();
00597
00598             int id = motor->nodeId;
00599             float position, speed, torque;
00600
00601             // TMotor 또는 MaxonMotor에 따른 데이터 파싱 및 출력
00602             if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00603             {
00604                 std::tuple<int, float, float, float> parsedData =
00605                 tmotorcmd.parseRecieveCommand(*tMotor, &frame);
00606                 position = std::get<1>(parsedData);
00607                 speed = std::get<2>(parsedData);
00608                 torque = std::get<3>(parsedData);
00609             }
00610             else if (std::shared_ptr<MaxonMotor> maxonMotor =
00611                 std::dynamic_pointer_cast<MaxonMotor>(motor))

```



```

00610         {
00611             std::tuple<int, float, float> parsedData = maxoncmd.parseRecieveCommand(*maxonMotor,
&frame);
00612             position = std::get<1>(parsedData);
00613             torque = std::get<2>(parsedData);
00614             speed = 0.0;
00615         }
00616
00617         // 데이터 CSV 파일에 쓰기
00618         ofs << "0x" << std::hex << std::setw(4) << std::setfill('0') << id << ", "
00619             << std::dec << position << ", " << speed << ", " << torque << "\n";
00620     }
00621 }
00622
00623 ofs.close();
00624 std::cout << "연주 txt_OutData 파일이 생성되었습니다." << csv_file_name << std::endl;
00625 }
00626
00627 /*                                     Single Test Mode                                     */
00628
00629 void TestManager::FixMotorPosition(std::shared_ptr<GenericMotor> &motor)
00630 {
00631     struct can_frame frame;
00632     canManager.checkConnection(motor);
00633     if (std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motor))
00634     {
00635         tMotorcmd.parseSendCommand(*tMotor, &frame, motor->nodeId, 8, motor->currentPos, 0, 250, 1,
0);
00636         if (canManager.sendAndRecv(motor, frame))
00637         {
00638             std::cout << "Position fixed for motor [" << motor->nodeId << "]" << std::endl;
00639         }
00640         else
00641         {
00642             std::cerr << "Failed to fix position for motor [" << motor->nodeId << "]" << std::endl;
00643         }
00644     }
00645     else if (std::shared_ptr<MaxonMotor> maxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motor))
00646     {
00647         maxoncmd.getTargetPosition(*maxonMotor, &frame, motor->currentPos);
00648         if (canManager.sendAndRecv(motor, frame))
00649         {
00650             std::cout << "Position fixed for motor [" << motor->nodeId << "]" << std::endl;
00651         }
00652         else
00653         {
00654             std::cerr << "Failed to fix position for motor [" << motor->nodeId << "]" << std::endl;
00655         }
00656     }
00657 }
00658
00659 void TestManager::TuningTmotor(float kp, float kd, float sine_t, const std::string selectedMotor, int
cycles, float peakAngle, int pathType)
00660 {
00661     canManager.setSocketsTimeout(0, 50000);
00662
00663     std::stringstream ss;
00664     ss << std::fixed << std::setprecision(2); // 소수점 둘째 자리까지만
00665     ss << "kp_" << kp << "_kd_" << kd << "_Hz_" << 1 / sine_t;
00666     std::string parameter = ss.str();
00667
00668     // CSV 파일명 설정
00669     std::string FileName1 = "../READ/" + parameter + "_in.txt";
00670
00671     // CSV 파일 열기
00672     std::ofstream csvFileIn(FileName1);
00673
00674     if (!csvFileIn.is_open())
00675     {
00676         std::cerr << "Error opening CSV file." << std::endl;
00677     }
00678
00679     // 헤더 추가
00680     csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
00681
00682     // CSV 파일명 설정
00683     std::string FileName2 = "../READ/" + parameter + "_out.txt";
00684
00685     // CSV 파일 열기
00686     std::ofstream csvFileOut(FileName2);
00687
00688     if (!csvFileOut.is_open())
00689     {
00690         std::cerr << "Error opening CSV file." << std::endl;
00691     }
00692 }

```

```

00696
00697 // 헤더 추가
00698 csvFileOut << "CAN_ID,p_act,v_act,tff_act\n"; // CSV 헤더
00699
00700 struct can_frame frame;
00701
00702 float peakRadian = peakAngle * M_PI / 180.0; // 피크 각도를 라디안으로 변환
00703 float amplitude = peakRadian;
00704
00705 float sample_time = 0.005;
00706 int max_samples = static_cast<int>(sine_t / sample_time);
00707 float v_des = 0, p_des = 0;
00708 float tff_des = 0;
00709 float p_act, v_act, tff_act;
00710 std::shared_ptr<TMotor> tMotor = std::dynamic_pointer_cast<TMotor>(motors[selectedMotor]);
00711 chrono::system_clock::time_point external = std::chrono::system_clock::now();
00712 for (int cycle = 0; cycle < cycles; cycle++)
00713 {
00714     for (int i = 0; i < max_samples; i++)
00715     {
00716         float time = i * sample_time;
00717
00718         if ((int)tMotor->nodeId == 7)
00719         {
00720             csvFileIn << std::dec << p_des << "0,0,0,0,0,0,0,0";
00721         }
00722         else
00723         {
00724             for (int i = 0; i < (int)tMotor->nodeId; i++)
00725             {
00726                 csvFileIn << "0,";
00727             }
00728             csvFileIn << std::dec << p_des << ",";
00729             for (int i = 0; i < (8 - (int)tMotor->nodeId); i++)
00730             {
00731                 csvFileIn << "0,";
00732             }
00733         }
00734
00735         float local_time = std::fmod(time, sine_t);
00736         if (pathType == 1) // 1-cos 경로
00737         {
00738             p_des = amplitude * (1 - cosf(2 * M_PI * local_time / sine_t)) / 2;
00739         }
00740         else if (pathType == 2) // 1-cos 및 -1+cos 결합 경로
00741         {
00742             if (local_time < sine_t / 2)
00743                 p_des = amplitude * (1 - cosf(4 * M_PI * local_time / sine_t)) / 2;
00744             else
00745                 p_des = amplitude * (-1 + cosf(4 * M_PI * (local_time - sine_t / 2) / sine_t)) /
00746 2;
00747         }
00748         tMotorCmd.parseSendCommand(*tMotor, &frame, tMotor->nodeId, 8, p_des, v_des, kp, kd,
00749 tff_des);
00750         csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') << tMotor->nodeId;
00751         while (1)
00752         {
00753             chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00754             chrono::microseconds elapsed_time =
00755 chrono::duration_cast<chrono::microseconds>(internal - external);
00756             if (elapsed_time.count() >= 5000)
00757             {
00758                 external = std::chrono::system_clock::now();
00759                 if (canManager.sendAndRecv(motors[selectedMotor], frame))
00760                 {
00761                     std::tuple<int, float, float, float> result =
00762 tMotorCmd.parseReceiveCommand(*tMotor, &frame);
00763
00764                     p_act = std::get<1>(result);
00765                     v_act = std::get<2>(result);
00766                     tff_act = std::get<3>(result);
00767                     // tff_des = kp * (p_des - p_act) + kd * (v_des - v_act);
00768                     csvFileOut << ',' << std::dec << p_act << ',' << v_act << ',' << tff_act << '\n';
00769                     break;
00770                 }
00771             }
00772             csvFileIn << "\n";
00773         }
00774     }
00775     csvFileIn.close();
00776     csvFileOut.close();
00777 }
00778 void TestManager::TuningMaxonCSP(float sine_t, const std::string selectedMotor, int cycles, float

```

```

    peakAngle, int pathType)
00779 {
00780
00781     canManager.setSocketsTimeout(0, 50000);
00782     std::string FileName1 = ".././READ/" + selectedMotor + "_in.txt";
00783
00784     std::ofstream csvFileIn(FileName1);
00785
00786     if (!csvFileIn.is_open())
00787     {
00788         std::cerr << "Error opening CSV file." << std::endl;
00789     }
00790
00791     // 헤더 추가
00792     csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
00793
00794     // CSV 파일명 설정
00795     std::string FileName2 = ".././READ/" + selectedMotor + "_out.txt";
00796
00797     // CSV 파일 열기
00798     std::ofstream csvFileOut(FileName2);
00799
00800     if (!csvFileOut.is_open())
00801     {
00802         std::cerr << "Error opening CSV file." << std::endl;
00803     }
00804     csvFileOut << "CAN_ID,p_act,v_act,tff_act\n"; // CSV 헤더
00805
00806     struct can_frame frame;
00807
00808     float peakRadian = peakAngle * M_PI / 180.0; // 피크 각도를 라디안으로 변환
00809     float amplitude = peakRadian;
00810
00811     float sample_time = 0.005;
00812     int max_samples = static_cast<int>(sine_t / sample_time);
00813     float p_des = 0;
00814     float p_act;
00815     // float tff_des = 0, v_des = 0;
00816     float tff_act;
00817
00818     std::shared_ptr<MaxonMotor> maxonMotor =
00819     std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]);
00820     for (int cycle = 0; cycle < cycles; cycle++)
00821     {
00822         for (int i = 0; i < max_samples; i++)
00823         {
00824             float time = i * sample_time;
00825
00826             for (int i = 0; i < (int)maxonMotor->nodeId - 1; i++)
00827             {
00828                 csvFileIn << "0,";
00829             }
00830             csvFileIn << std::dec << p_des << ",";
00831             for (int i = 0; i < (9 - (int)maxonMotor->nodeId); i++)
00832             {
00833                 csvFileIn << "0,";
00834             }
00835             float local_time = std::fmod(time, sine_t);
00836             if (pathType == 1) // 1-cos 경로
00837             {
00838                 p_des = amplitude * (1 - cosf(2 * M_PI * local_time / sine_t)) / 2;
00839             }
00840             else if (pathType == 2) // 1-cos 및 -1+cos 결합 경로
00841             {
00842                 if (local_time < sine_t / 2)
00843                     p_des = amplitude * (1 - cosf(4 * M_PI * local_time / sine_t)) / 2;
00844                 else
00845                     p_des = amplitude * (-1 + cosf(4 * M_PI * (local_time - sine_t / 2) / sine_t)) /
00846                 2;
00847             }
00848             maxoncmd.getTargetPosition(*maxonMotor, &frame, p_des);
00849             csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') << maxonMotor->nodeId;
00850
00851             chrono::system_clock::time_point external = std::chrono::system_clock::now();
00852             while (1)
00853             {
00854                 chrono::system_clock::time_point internal = std::chrono::system_clock::now();
00855                 chrono::microseconds elapsed_time =
00856                 chrono::duration_cast<chrono::microseconds>(internal - external);
00857                 if (elapsed_time.count() >= 5000)
00858                 {
00859                     ssize_t bytesWritten = write(canManager.sockets.at(maxonMotor->interFaceName),
00860                     &frame, sizeof(struct can_frame));
00861                     if (bytesWritten == -1)

```

```

00861         {
00862             std::cerr << "Failed to write to socket for interface: " <<
maxonMotor->interFaceName << std::endl;
00863             std::cerr << "Error: " << strerror(errno) << " (errno: " << errno << ")" <<
std::endl;
00864         }
00865         canManager.txFrame(motors[selectedMotor], frame);
00866
00867         maxoncmd.getSync(&frame);
00868         canManager.txFrame(motors[selectedMotor], frame);
00869
00870         if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))
00871         {
00872             while (!motors[selectedMotor]->recieveBuffer.empty())
00873             {
00874                 frame = motors[selectedMotor]->recieveBuffer.front();
00875                 if (frame.can_id == maxonMotor->rxPdoIds[0])
00876                 {
00877                     std::tuple<int, float, float> result =
maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
00878
00879                     p_act = std::get<1>(result);
00880                     tff_act = std::get<2>(result);
00881                     // tff_des = kp * (p_des - p_act) + kd * (v_des - v_act);
00882                     csvFileOut << ',' << std::dec << p_act << ", " << tff_act << '\n';
00883                     break;
00884                 }
00885                 motors[selectedMotor]->recieveBuffer.pop();
00886             }
00887         }
00888     }
00889 }
00890 csvFileIn << "\n";
00891 }
00892 }
00893 csvFileIn.close();
00894 csvFileOut.close();
00895 }
00896
00897 void TestManager::TuningLoopTask()
00898 {
00899     for (auto motor_pair : motors)
00900     {
00901         FixMotorPosition(motor_pair.second);
00902     }
00903
00904     std::string userInput, selectedMotor, fileName;
00905     float kp, kd, peakAngle;
00906     float sine_t = 4.0;
00907     int cycles = 2, pathType, controlType, des_vel, des_tff, direction;
00908
00909     selectedMotor = motors.begin()->first;
00910
00911     InitializeParameters(selectedMotor, kp, kd, peakAngle, pathType, controlType, des_vel, des_tff,
direction);
00912     while (true)
00913     {
00914         int result = system("clear");
00915         if (result != 0)
00916         {
00917             std::cerr << "Error during clear screen" << std::endl;
00918         }
00919
00920         std::string pathTypeDescription;
00921         if (pathType == 1)
00922         {
00923             pathTypeDescription = "1: 1 - cos (0 -> peak -> 0)";
00924         }
00925         else if (pathType == 2)
00926         {
00927             pathTypeDescription = "2: 1 - cos & cos - 1 (0 -> peak -> 0 -> -peak -> 0)";
00928         }
00929
00930         std::string controlTypeDescription;
00931         if (controlType == 1)
00932         {
00933             controlTypeDescription = "CSP";
00934             setMaxonMode("CSP");
00935         }
00936         else if (controlType == 2)
00937         {
00938             controlTypeDescription = "CSV";
00939             setMaxonMode("CSV");
00940         }
00941         else if (controlType == 3)

```

```

00944     {
00945         controlTypeDescription = "CST";
00946
00947         setMaxonMode("CST");
00948     }
00949     else if (controlType == 4)
00950     {
00951         controlTypeDescription = "Drum Test";
00952     }
00953
00954     std::string directionDescription;
00955     if (direction == 1)
00956     {
00957         directionDescription = "CW";
00958     }
00959     else if (direction == 2)
00960     {
00961         directionDescription = "CCW";
00962     }
00963
00964     std::cout << "===== Tuning Menu =====\n";
00965     std::cout << "Available Motors:\n";
00966
00967     for (const auto &motor_pair : motors)
00968     {
00969         std::cout << " - " << motor_pair.first << "\n";
00970     }
00971     bool isMaxonMotor;
00972     if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]))
00973     {
00974         isMaxonMotor = true;
00975     }
00976     else
00977     {
00978         isMaxonMotor = false;
00979     }
00980
00981     std::cout << "-----\n";
00982     std::cout << "Selected Motor: " << selectedMotor << "\n";
00983     if (!isMaxonMotor)
00984     {
00985         std::cout << "Kp: " << kp << " | Kd: " << kd << "\n";
00986     }
00987     else
00988     {
00989         std::cout << "Control Type : " << controlTypeDescription;
00990         std::cout << " | Vel [rpm]: " << des_vel << " | Des Torque: " << des_tff * 31.052 / 1000 <<
" [mNm]"
00991         << "\n";
00992     };
00993     std::cout << "Sine Period: " << sine_t << " | Cycles: " << cycles << " | Hz: " << 1 / sine_t << "\n";
00994     std::cout << "Peak Angle: " << peakAngle << " | Path Type [Pos]: " << pathTypeDescription << "\n";
00995     std::cout << "Direction: " << directionDescription << "\n";
00996     std::cout << "\nCommands:\n";
00997     if (!isMaxonMotor)
00998     {
00999         std::cout << "[a]: kp | [b]: kd |";
01000     }
01001     else
01002     {
01003         std::cout << "[a]: des_vel | [b]: des_tff | [c]: Control |";
01004     }
01005     std::cout << "[d]: Select Motor | [e]: Peak | [f]: Path\n";
01006     std::cout << "[g]: Period | [h]: Cycles | [i]: Run \n";
01007     if (isMaxonMotor)
01008     {
01009         std::cout << "[k]: Direction | ";
01010     }
01011     std::cout << "[j]: Exit\n";
01012     std::cout << "===== \n";
01013     std::cout << "Enter Command: ";
01014     std::cin >> userInput;
01015     std::transform(userInput.begin(), userInput.end(), userInput.begin(), ::tolower);
01016
01017     if (userInput[0] == 'j')
01018     {
01019         break;
01020     }
01021     else if (userInput[0] == 'd')
01022     {
01023         while (true)
01024         {
01025             std::cout << "Enter the name of the motor to tune: ";
01026             std::cin >> selectedMotor;
01027             if (motors.find(selectedMotor) != motors.end())
01028             {

```

```

01029         InitializeParameters(selectedMotor, kp, kd, peakAngle, pathType, controlType,
des_vel, des_tff, direction);
01030         break;
01031     }
01032     else
01033     {
01034         std::cout << "Invalid motor name. Please enter a valid motor name.\n";
01035     }
01036 }
01037 }
01038 else if (userInput == "k" && isMaxonMotor)
01039 {
01040     std::cout << "Enter Desired Direction: ";
01041     std::cout << "1: Clock Wise\n";
01042     std::cout << "2: Counter Clock Wise\n";
01043     std::cout << "Enter Path Type (1 or 2): ";
01044     std::cin >> direction;
01045
01046     if (direction != 1 && direction != 2)
01047     {
01048         std::cout << "Invalid direction type. Please enter 1 or 2.\n";
01049         pathType = 1;
01050     }
01051
01052     std::cin >> direction;
01053 }
01054 else if (userInput == "a" && !isMaxonMotor)
01055 {
01056     std::cout << "Enter Desired Kp: ";
01057     std::cin >> kp;
01058 }
01059 else if (userInput == "b" && !isMaxonMotor)
01060 {
01061     std::cout << "Enter Desired Kd: ";
01062     std::cin >> kd;
01063 }
01064 else if (userInput == "a" && isMaxonMotor)
01065 {
01066     std::cout << "Enter Desired Velocity: ";
01067     std::cin >> des_vel;
01068 }
01069 else if (userInput == "b" && isMaxonMotor)
01070 {
01071     std::cout << "Enter Desired Torque: ";
01072     std::cout << "-100 [unit] = -3.1052 [mNm]\n";
01073     std::cin >> des_tff;
01074 }
01075 else if (userInput == "e")
01076 {
01077     std::cout << "Enter Desired Peak Angle: ";
01078     std::cin >> peakAngle;
01079 }
01080 else if (userInput == "f")
01081 {
01082     std::cout << "Select Path Type:\n";
01083     std::cout << "1: 1 - cos (0 -> peak -> 0)\n";
01084     std::cout << "2: 1 - cos & cos - 1 (0 -> peak -> 0 -> -peak -> 0)\n";
01085     std::cout << "Enter Path Type (1 or 2): ";
01086     std::cin >> pathType;
01087
01088     if (pathType != 1 && pathType != 2)
01089     {
01090         std::cout << "Invalid path type. Please enter 1 or 2.\n";
01091         pathType = 1;
01092     }
01093 }
01094 else if (userInput[0] == 'g')
01095 {
01096     std::cout << "Enter Desired Sine Period: ";
01097     std::cin >> sine_t;
01098 }
01099 else if (userInput[0] == 'h')
01100 {
01101     std::cout << "Enter Desired Cycles: ";
01102     std::cin >> cycles;
01103 }
01104 else if (userInput == "c" && isMaxonMotor)
01105 {
01106     std::cout << "Select Control Type:\n";
01107     std::cout << "1: Cyclic Synchronous Position Mode (CSP)\n";
01108     std::cout << "2: Cyclic Synchronous Velocity Mode (CSV)\n";
01109     std::cout << "3: Cyclic Synchronous Torque Mode (CST)\n";
01110     std::cout << "4: Maxon Drum Test (CSP)\n";
01111     std::cout << "Enter Path Type (1 or 2 or 3 or 4): ";
01112     std::cin >> controlType;
01113
01114     if (controlType != 1 && controlType != 2 && controlType != 3 && controlType != 4)

```

```

01115         {
01116             std::cout << "Invalid path type. Please enter 1 or 2 or 3 or 4.\n";
01117             controlType = 1;
01118         }
01119     }
01120     else if (userInput[0] == 'i')
01121     {
01122         if (!isMaxonMotor) // Tmotor일 경우
01123         {
01124             TuningTmotor(kp, kd, sine_t, selectedMotor, cycles, peakAngle, pathType);
01125         }
01126         else // MaxonMotor일 경우
01127         {
01128             if (controlType == 1)
01129             {
01130                 TuningMaxonCSP(sine_t, selectedMotor, cycles, peakAngle, pathType);
01131             }
01132             else if (controlType == 2)
01133             {
01134                 TuningMaxonCSV(selectedMotor, des_vel, direction);
01135             }
01136             else if (controlType == 3)
01137             {
01138                 TuningMaxonCST(selectedMotor, des_tff, direction);
01139             }
01140             else if (controlType == 4)
01141             {
01142                 //
01143             }
01144         }
01145     }
01146 }
01147 }
01148 }
01149 }
01150
01151 void TestManager::InitializeParameters(const std::string selectedMotor, float &kp, float &kd, float
&peakAngle, int &pathType, int &controlType, int &des_vel, int &des_tff, int &direction)
01152 {
01153     if (selectedMotor == "waist")
01154     {
01155         kp = 200.0;
01156         kd = 1.0;
01157         peakAngle = 30;
01158         pathType = 2;
01159         des_vel = 0;
01160         des_tff = 0;
01161     }
01162     else if (selectedMotor == "R_arm1" || selectedMotor == "L_arm1" ||
selectedMotor == "R_arm2" || selectedMotor == "R_arm3" ||
selectedMotor == "L_arm2" || selectedMotor == "L_arm3")
01163     {
01164         kp = 50.0; // 예시 값, 실제 필요한 값으로 조정
01165         kd = 1.0; // 예시 값, 실제 필요한 값으로 조정
01166         peakAngle = 90;
01167         pathType = 1;
01168         des_vel = 0;
01169         des_tff = 0;
01170     }
01171     else if (selectedMotor == "L_wrist" || selectedMotor == "R_wrist" || selectedMotor ==
"maxonForTest")
01172     {
01173         peakAngle = 90;
01174         pathType = 1;
01175         controlType = 1;
01176         direction = 1;
01177         des_vel = 0;
01178         des_tff = 0;
01179     }
01180     else if (selectedMotor == "maxonForTest")
01181     {
01182         peakAngle = 90;
01183         pathType = 1;
01184         controlType = 3;
01185         direction = -1;
01186         des_vel = 0;
01187         des_tff = 500;
01188     }
01189 }
01190 }
01191 }
01192
01193 void TestManager::TuningMaxonCSV(const std::string selectedMotor, int des_vel, int direction)
01194 {
01195     des_vel = des_vel * 35;
01196     if (direction == 1)
01197         des_vel *= 1;
01198     else

```

```

01200         des_vel *= -1;
01201
01202         canManager.setSocketsTimeout(0, 50000);
01203         std::string FileName1 = ".././READ/" + selectedMotor + "_csv_in.txt";
01204
01205         std::ofstream csvFileIn(FileName1);
01206
01207         if (!csvFileIn.is_open())
01208         {
01209             std::cerr << "Error opening CSV file." << std::endl;
01210         }
01211
01212         // 헤더 추가
01213         csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
01214
01215         // CSV 파일명 설정
01216         std::string FileName2 = ".././READ/" + selectedMotor + "_csv_out.txt";
01217
01218         // CSV 파일 열기
01219         std::ofstream csvFileOut(FileName2);
01220
01221         if (!csvFileOut.is_open())
01222         {
01223             std::cerr << "Error opening CSV file." << std::endl;
01224         }
01225         csvFileOut << "CAN_ID,pos_act,tff_act\n"; // CSV 헤더
01226
01227         struct can_frame frame;
01228
01229         float p_act, tff_act;
01230         char input = 'a';
01231         std::shared_ptr<MaxonMotor> maxonMotor =
01232         std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]);
01233
01234         for (int i = 0; i < (int)maxonMotor->nodeId - 1; i++)
01235         {
01236             csvFileIn << "0,";
01237
01238             csvFileIn << std::dec << des_vel << ",";
01239             for (int i = 0; i < (9 - (int)maxonMotor->nodeId); i++)
01240             {
01241                 csvFileIn << "0,";
01242             }
01243
01244             maxoncmd.getTargetVelocity(*maxonMotor, &frame, des_vel);
01245
01246             chrono::system_clock::time_point external = std::chrono::system_clock::now();
01247             while (1)
01248             {
01249                 if (kbhit())
01250                 {
01251                     input = getchar();
01252                 }
01253
01254                 if (input == 'q')
01255                     continue;
01256                 else if (input == 'e')
01257                     break;
01258
01259                 chrono::system_clock::time_point internal = std::chrono::system_clock::now();
01260                 chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
01261 external);
01262                 if (elapsed_time.count() >= 5000)
01263                 {
01264                     ssize_t bytesWritten = write(canManager.sockets.at(maxonMotor->interFaceName), &frame,
01265 sizeof(struct can_frame));
01266                     if (bytesWritten == -1)
01267                     {
01268                         std::cerr << "Failed to write to socket for interface: " << maxonMotor->interFaceName <<
01269 std::endl;
01270                         std::cerr << "Error: " << strerror(errno) << " (errno: " << errno << ")" << std::endl;
01271                     }
01272                     maxoncmd.getSync(&frame);
01273                     bytesWritten = write(canManager.sockets.at(maxonMotor->interFaceName), &frame,
01274 sizeof(struct can_frame));
01275                     if (bytesWritten == -1)
01276                     {
01277                         std::cerr << "Failed to write to socket for interface: " << maxonMotor->interFaceName <<
01278 std::endl;
01279                         std::cerr << "Error: " << strerror(errno) << " (errno: " << errno << ")" << std::endl;
01280                     }
01281                     ssize_t bytesRead = read(canManager.sockets.at(maxonMotor->interFaceName), &frame,
01282 sizeof(struct can_frame));
01283
01284

```



```

01280         if (bytesRead == -1)
01281         {
01282             std::cerr << "Failed to read from socket for interface: " << maxonMotor->interFaceName <<
std::endl;
01283             return;
01284         }
01285         else
01286         {
01287
01288             std::tuple<int, float, float> result = maxoncmd.parseRecieveCommand(*maxonMotor,
&frame);
01289
01290             p_act = std::get<1>(result);
01291             tff_act = std::get<2>(result);
01292             // tff_des = kp * (p_des - p_act) + kd * (v_des - v_act);
01293             csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') << maxonMotor->nodeId;
01294             csvFileOut << ',' << std::dec << p_act << "," << tff_act << '\n';
01295         }
01296     }
01297 }
01298
01299 csvFileIn.close();
01300 csvFileOut.close();
01301 }
01302
01303 void TestManager::TuningMaxonCST(const std::string selectedMotor, int des_tff, int direction)
01304 {
01305     if (direction == 1)
01306         des_tff *= 1;
01307     else
01308         des_tff *= -1;
01309
01310     canManager.setSocketsTimeout(0, 50000);
01311     std::string FileName1 = "../READ/" + selectedMotor + "_cst_in.txt";
01312
01313     std::ofstream csvFileIn(FileName1);
01314
01315     if (!csvFileIn.is_open())
01316     {
01317         std::cerr << "Error opening CSV file." << std::endl;
01318     }
01319
01320     // 헤더 추가
01321     csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
01322
01323     // CSV 파일명 설정
01324     std::string FileName2 = "../READ/" + selectedMotor + "_cst_out.txt";
01325
01326     // CSV 파일 열기
01327     std::ofstream csvFileOut(FileName2);
01328
01329     if (!csvFileOut.is_open())
01330     {
01331         std::cerr << "Error opening CSV file." << std::endl;
01332     }
01333     csvFileOut << "CAN_ID,pos_act,tff_act\n"; // CSV 헤더
01334
01335     struct can_frame frame;
01336
01337     float p_act, tff_act;
01338     char input = 'a';
01339     std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]);
01340
01341     for (int i = 0; i < (int)maxonMotor->nodeId - 1; i++)
01342     {
01343         csvFileIn << "0,";
01344     }
01345     csvFileIn << std::dec << des_tff << ",";
01346     for (int i = 0; i < (9 - (int)maxonMotor->nodeId); i++)
01347     {
01348         csvFileIn << "0,";
01349     }
01350     bool reachedDrum = false;
01351     bool waitForEInput = false;
01352     chrono::system_clock::time_point external = std::chrono::system_clock::now();
01353     while (1)
01354     {
01355
01356         if (kbhit())
01357         {
01358             input = getchar();
01359             if (input == 'e' && waitForEInput) // waitForEInput이 true일 때만 'e' 입력 처리
01360             {
01361                 maxoncmd.getCSTMode(*maxonMotor, &frame);
01362                 canManager.sendAndRecv(motors[selectedMotor], frame);
01363                 break;

```

```

01364     }
01365 }
01366
01367 if (waitForEInput)
01368 {
01369     // 'e' 입력을 기다리는 동안 다른 작업을 하지 않음
01370     continue;
01371 }
01372
01373 chrono::system_clock::time_point internal = std::chrono::system_clock::now();
01374 chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
external);
01375 if (elapsed_time.count() >= 5000)
01376 {
01377
01378     maxoncmd.getTargetTorque(*maxonMotor, &frame, des_tff);
01379     canManager.txFrame(motors[selectedMotor], frame);
01380
01381     maxoncmd.getSync(&frame);
01382     canManager.txFrame(motors[selectedMotor], frame);
01383
01384     if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))
01385     {
01386         while (!motors[selectedMotor]->recieveBuffer.empty())
01387         {
01388             frame = motors[selectedMotor]->recieveBuffer.front();
01389             if (frame.can_id == maxonMotor->rxPdoIds[0])
01390             {
01391                 std::tuple<int, float, float> result =
maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
01392
01393                 p_act = std::get<1>(result);
01394                 tff_act = std::get<2>(result);
01395                 // tff_des = kp * (p_des - p_act) + kd * (v_des - v_act);
01396                 csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') <<
maxonMotor->nodeId;
01397                 csvFileOut << ',' << std::dec << p_act << "," << tff_act << '\n';
01398
01399                 // 임계 토크 값을 체크하고, 조건을 충족하면 반대 방향으로 토크 주기
01400                 if (abs(tff_act) > 18)
01401                 {
01402                     des_tff = 100;
01403                     reachedDrum = true;
01404                 }
01405
01406                 // 특정 각도에 도달했는지 확인하는 조건
01407                 if (p_act > -0.5 && reachedDrum)
01408                 {
01409                     maxoncmd.getCSPMode(*maxonMotor, &frame);
01410                     canManager.sendAndRecv(motors[selectedMotor], frame);
01411
01412                     canManager.checkConnection(motors[selectedMotor]);
01413                     maxoncmd.getTargetPosition(*maxonMotor, &frame,
motors[selectedMotor]->currentPos);
01414                     canManager.txFrame(motors[selectedMotor], frame);
01415                     maxoncmd.getSync(&frame);
01416                     canManager.txFrame(motors[selectedMotor], frame);
01417                     if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))
01418                     {
01419                         while (!motors[selectedMotor]->recieveBuffer.empty())
01420                         {
01421                             frame = motors[selectedMotor]->recieveBuffer.front();
01422                             if (frame.can_id == maxonMotor->rxPdoIds[0])
01423                             {
01424                                 std::cout << "This is My stick!! \n";
01425                             }
01426                             motors[selectedMotor]->recieveBuffer.pop();
01427                         }
01428                     }
01429
01430                     waitForEInput = true; // 'e' 입력 대기 상태로 전환
01431                     std::cout << "Waiting for 'e' input...\n";
01432                 }
01433             }
01434             if (!motors[selectedMotor]->recieveBuffer.empty())
01435             {
01436                 motors[selectedMotor]->recieveBuffer.pop();
01437             }
01438         }
01439     }
01440 }
01441 }
01442
01443 csvFileIn.close();
01444 csvFileOut.close();
01445 }
01446

```

```

01447 void TestManager::setMaxonMode(std::string targetMode)
01448 {
01449     struct can_frame frame;
01450     canManager.setSocketsTimeout(0, 10000);
01451     for (const auto &motorPair : motors)
01452     {
01453         std::string name = motorPair.first;
01454         std::shared_ptr<GenericMotor> motor = motorPair.second;
01455         if (std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motorPair.second))
01456         {
01457             if (targetMode == "CSV")
01458             {
01459                 maxoncmd.getCSVMode(*maxonMotor, &frame);
01460                 canManager.sendAndRecv(motor, frame);
01461             }
01462             else if (targetMode == "CST")
01463             {
01464                 maxoncmd.getCSTMode(*maxonMotor, &frame);
01465                 canManager.sendAndRecv(motor, frame);
01466             }
01467             else if (targetMode == "HMM")
01468             {
01469                 maxoncmd.getHomeMode(*maxonMotor, &frame);
01470                 canManager.sendAndRecv(motor, frame);
01471             }
01472             else if (targetMode == "CSP")
01473             {
01474                 maxoncmd.getCSPMode(*maxonMotor, &frame);
01475                 canManager.sendAndRecv(motor, frame);
01476             }
01477         }
01478     }
01479 }
01480
01481 int TestManager::kbhit()
01482 {
01483     struct termios oldt, newt;
01484     int ch;
01485     int oldf;
01486
01487     tcgetattr(STDIN_FILENO, &oldt);
01488     newt = oldt;
01489     newt.c_lflag &= ~(ICANON | ECHO);
01490     tcsetattr(STDIN_FILENO, TCSANOW, &newt);
01491     oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
01492     fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);
01493
01494     ch = getchar();
01495
01496     tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
01497     fcntl(STDIN_FILENO, F_SETFL, oldf);
01498
01499     if (ch != EOF)
01500     {
01501         ungetc(ch, stdin);
01502         return 1;
01503     }
01504
01505     return 0;
01506 }
01507
01508 /*                                     Stick Test Mode                                     */
01511
01512 void TestManager::TestStickLoop()
01513 {
01514
01515     std::string userInput;
01516     std::string selectedMotor = "maxonForTest";
01517     float des_tff = 0;
01518     float posThreshold = 1.57; // 위치 임계값 초기화
01519     float tffThreshold = 18;   // 토크 임계값 초기화
01520     int backTorqueUnit = 150;
01521     for (auto motor_pair : motors)
01522     {
01523         FixMotorPosition(motor_pair.second);
01524     }
01525
01526     while (true)
01527     {
01528
01529         int result = system("clear");
01530         if (result != 0)
01531         {
01532             std::cerr << "Error during clear screen" << std::endl;
01533         }
01534

```

```

01535     std::cout << "===== Tuning Menu =====\n";
01536     std::cout << "Available Motors for Stick Mode:\n";
01537     for (const auto &motor_pair : motors)
01538     {
01539         if (motor_pair.first == "maxonForTest")
01540             std::cout << " - " << motor_pair.first << "\n";
01541     }
01542
01543     bool isMaxonMotor = std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]) != nullptr;
01544     if (!isMaxonMotor)
01545         break;
01546
01547     std::cout << "-----\n";
01548     std::cout << "Selected Motor: " << selectedMotor << "\n";
01549
01550     std::cout << "Des Torque: " << des_tff * 31.052 / 1000 << " [mNm]\n";
01551     std::cout << "Torque Threshold: " << tffThreshold << " [mNm]\n"; // 현재 토크 임계값 출력
01552     std::cout << "Position Threshold: " << posThreshold << " [rad]\n";
01553     std::cout << "Back Torque: " << backTorqueUnit * 31.052 / 1000 << " [mNm]\n";
01554     std::cout << "\nCommands:\n";
01555     std::cout << "[a]: des_tff | [b]: Direction | [c]: Back Torque\n";
01556     std::cout << "[d]: Set Torque Threshold [e]: Set Position Threshold \n";
01557     std::cout << "[f]: Run | [g]: Exit\n";
01558     std::cout << "=====\n";
01559     std::cout << "Enter Command: ";
01560     std::cin >> userInput;
01561     std::transform(userInput.begin(), userInput.end(), userInput.begin(), ::tolower);
01562
01563     if (userInput[0] == 'g')
01564     {
01565         break;
01566     }
01567
01568     else if (userInput == "c")
01569     {
01570         std::cout << "Enter Desired [Back] Torque In Unit: ";
01571         std::cout << "100 [unit] = 3.1052 [mNm]\n";
01572         std::cin >> backTorqueUnit;
01573     }
01574     else if (userInput == "a" && isMaxonMotor)
01575     {
01576         std::cout << "Enter Desired Torque In Unit: ";
01577         std::cout << "-100 [unit] = -3.1052 [mNm]\n";
01578         std::cin >> des_tff;
01579     }
01580     else if (userInput == "d" && isMaxonMotor)
01581     {
01582         std::cout << "Enter Desired Torque Threshold: ";
01583         std::cout << "-100 [unit] = -3.1052 [mNm]\n";
01584         std::cin >> tffThreshold;
01585     }
01586     else if (userInput == "e" && isMaxonMotor)
01587     {
01588         std::cout << "Enter Desired Position Threshold: ";
01589         std::cin >> posThreshold;
01590     }
01591     else if (userInput[0] == 'f' && isMaxonMotor)
01592     {
01593         TestStick(selectedMotor, des_tff, tffThreshold, posThreshold, backTorqueUnit);
01594     }
01595 }
01596 }
01597
01598 void TestManager::TestStick(const std::string selectedMotor, int des_tff, float tffThreshold, float
posThreshold, int backTorqueUnit)
01599 {
01600
01601     canManager.setSocketsTimeout(0, 50000);
01602     std::string FileName1 = "../READ/" + selectedMotor + "_cst_in.txt";
01603
01604     std::ofstream csvFileIn(FileName1);
01605
01606     if (!csvFileIn.is_open())
01607     {
01608         std::cerr << "Error opening CSV file." << std::endl;
01609     }
01610
01611     // Input File
01612     csvFileIn << "0x007,0x001,0x002,0x003,0x004,0x005,0x006,0x008,0x009\n";
01613     std::string FileName2 = "../READ/" + selectedMotor + "_cst_out.txt";
01614     std::ofstream csvFileOut(FileName2);
01615
01616     if (!csvFileOut.is_open())
01617     {
01618         std::cerr << "Error opening CSV file." << std::endl;
01619     }
01620     csvFileOut << "CAN_ID,pos_act,tff_act\n"; // CSV 헤더

```

```

01621
01622     struct can_frame frame;
01623
01624     float p_act, tff_act;
01625
01626     std::shared_ptr<MaxonMotor> maxonMotor =
std::dynamic_pointer_cast<MaxonMotor>(motors[selectedMotor]);
01627
01628     for (int i = 0; i < (int)maxonMotor->nodeId - 1; i++)
01629     {
01630         csvFileIn << "0,";
01631     }
01632     csvFileIn << std::dec << des_tff << ",";
01633     for (int i = 0; i < (9 - (int)maxonMotor->nodeId); i++)
01634     {
01635         csvFileIn << "0,";
01636     }
01637     bool reachedDrum = false;
01638     bool motorFixed = false;
01639
01640     chrono::system_clock::time_point external = std::chrono::system_clock::now();
01641     bool motorModeSet = false;
01642
01643     float positionValues[4] = {0}; // 포지션 값 저장을 위한 정적 배열
01644     int posIndex = 0;             // 현재 포지션 값 인덱스
01645
01646     while (1)
01647     {
01648
01649         if (!motorModeSet)
01650         {
01651             maxoncmd.getCSTMode(*maxonMotor, &frame);
01652             canManager.sendAndRecv(motors[selectedMotor], frame);
01653             motorModeSet = true; // 모터 모드 설정 완료
01654         }
01655         if (motorFixed)
01656         {
01657             break;
01658         }
01659
01660         chrono::system_clock::time_point internal = std::chrono::system_clock::now();
01661         chrono::microseconds elapsed_time = chrono::duration_cast<chrono::microseconds>(internal -
external);
01662         if (elapsed_time.count() >= 5000)
01663         {
01664
01665             maxoncmd.getTargetTorque(*maxonMotor, &frame, des_tff);
01666             canManager.txFrame(motors[selectedMotor], frame);
01667
01668             maxoncmd.getSync(&frame);
01669             canManager.txFrame(motors[selectedMotor], frame);
01670
01671             if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))
01672             {
01673                 while (!motors[selectedMotor]->recieveBuffer.empty())
01674                 {
01675                     frame = motors[selectedMotor]->recieveBuffer.front();
01676                     if (frame.can_id == maxonMotor->rxPdoIds[0])
01677                     {
01678                         std::tuple<int, float, float> result =
maxoncmd.parseRecieveCommand(*maxonMotor, &frame);
01679
01680                         p_act = std::get<1>(result);
01681                         tff_act = std::get<2>(result);
01682                         csvFileOut << "0x" << std::hex << std::setw(4) << std::setfill('0') <<
maxonMotor->nodeId;
01683                         csvFileOut << ',' << std::dec << p_act << "," << tff_act << '\n';
01684
01685                         positionValues[posIndex % 4] = p_act;
01686                         posIndex++;
01687
01688                         if (!reachedDrum && dct_fun(positionValues, 0))
01689                         {
01690                             des_tff = backTorqueUnit;
01691                             reachedDrum = true;
01692                         }
01693
01694                         // 특정 각도에 도달했는지 확인하는 조건
01695                         if (p_act > posThreshold && reachedDrum)
01696                         {
01697                             maxoncmd.getCSPMode(*maxonMotor, &frame);
01698                             canManager.sendAndRecv(motors[selectedMotor], frame);
01699
01700                             maxoncmd.getTargetPosition(*maxonMotor, &frame, p_act);
01701                             canManager.txFrame(motors[selectedMotor], frame);
01702                             maxoncmd.getSync(&frame);
01703                             canManager.txFrame(motors[selectedMotor], frame);

```

```

01704         if (canManager.recvToBuff(motors[selectedMotor], canManager.maxonCnt))
01705         {
01706             while (!motors[selectedMotor]->recieveBuffer.empty())
01707             {
01708                 frame = motors[selectedMotor]->recieveBuffer.front();
01709                 if (frame.can_id == maxonMotor->rxPdoIds[0])
01710                 {
01711                     motorFixed = true;
01712                 }
01713                 motors[selectedMotor]->recieveBuffer.pop();
01714             }
01715         }
01716     }
01717 }
01718 if (!motors[selectedMotor]->recieveBuffer.empty())
01719 {
01720     motors[selectedMotor]->recieveBuffer.pop();
01721 }
01722 }
01723 }
01724 }
01725 }
01726
01727 csvFileIn.close();
01728 csvFileOut.close();
01729 }
01730
01731 bool TestManager::dct_fun(float positions[], float vel_th)
01732 {
01733     // 포지션 배열에서 각각의 값을 추출합니다.
01734     float the_k = positions[3]; // 가장 최신 값
01735     float the_k_1 = positions[2];
01736     float the_k_2 = positions[1];
01737     float the_k_3 = positions[0]; // 가장 오래된 값
01738
01739     float ang_k = (the_k + the_k_1) / 2;
01740     float ang_k_1 = (the_k_1 + the_k_2) / 2;
01741     float ang_k_2 = (the_k_2 + the_k_3) / 2;
01742     float vel_k = ang_k - ang_k_1;
01743     float vel_k_1 = ang_k_1 - ang_k_2;
01744
01745     if (vel_k > vel_k_1 && vel_k > vel_th && ang_k < 0.1 * M_PI)
01746         return true;
01747     else if (ang_k < -0.25 * M_PI)
01748         return true;
01749     else
01750         return false;
01751 }

```

# Index

- ~CanManager
  - CanManager, 9
- activateCanPort
  - CanManager, 10
- ApplyDir
  - PathManager, 77
- backarr
  - PathManager, 86
- bpm
  - PathManager, 86
- c\_L
  - PathManager, 86
- c\_MotorAngle
  - PathManager, 87
- c\_R
  - PathManager, 87
- CanManager, 7
  - ~CanManager, 9
  - activateCanPort, 10
  - CanManager, 9
  - checkAllMotors, 10
  - checkCanPortsStatus, 10
  - checkConnection, 11
  - clearCanBuffer, 12
  - clearReadBuffers, 12
  - createSocket, 12
  - deactivateCanPort, 13
  - distributeFramesToMotors, 13
  - ERR\_SOCKET\_CONFIGURE\_FAILURE, 21
  - ERR\_SOCKET\_CREATE\_FAILURE, 21
  - getCanPortStatus, 14
  - ifnames, 21
  - initializeCAN, 14
  - isConnected, 21
  - list\_and\_activate\_available\_can\_ports, 15
  - maxoncmd, 21
  - maxonCnt, 21
  - motors, 22
  - readFramesFromAllSockets, 16
  - recvToBuff, 16
  - restartCanPorts, 16
  - rxFrame, 17
  - sendAndRecv, 17
  - sendFromBuff, 18
  - setMotorsSocket, 18
  - setSocketsTimeout, 19
  - setSocketTimeout, 20
  - sockets, 22
  - tempFrames, 22
  - tmotorcmd, 22
  - txFrame, 20
- canManager
  - DrumRobot, 40
  - HomeManager, 60
  - PathManager, 87
  - TestManager, 123
- canReceiveId
  - MaxonMotor, 72
- canSendId
  - MaxonMotor, 72
- checkAllMotors
  - CanManager, 10
- checkCanPortsStatus
  - CanManager, 10
- checkConnection
  - CanManager, 11
- checkUserInput
  - DrumRobot, 25
- clearCanBuffer
  - CanManager, 12
- clearMotorsSendBuffer
  - DrumRobot, 25
- clearReadBuffers
  - CanManager, 12
- clearReceiveBuffer
  - GenericMotor, 44
  - MaxonMotor, 72
  - TMotor, 125
- clearSendBuffer
  - GenericMotor, 44
  - MaxonMotor, 72
  - TMotor, 125
- connect
  - PathManager, 77
- createSocket
  - CanManager, 12
- currentPos
  - GenericMotor, 45
  - MaxonMotor, 72
  - TMotor, 126
- currentTor
  - GenericMotor, 45
  - MaxonMotor, 72
  - TMotor, 126
- currentVel

- GenericMotor, 45
- MaxonMotor, 73
- TMotor, 126
- cwdir
  - GenericMotor, 45
  - MaxonMotor, 73
  - TMotor, 126
- dct\_fun
  - TestManager, 96
- deactivateCanPort
  - CanManager, 13
- DeactivateControlTask
  - DrumRobot, 26
- desPos
  - GenericMotor, 45
  - MaxonMotor, 73
  - TMotor, 126
- desTor
  - GenericMotor, 45
  - MaxonMotor, 73
  - TMotor, 126
- desVel
  - GenericMotor, 45
  - MaxonMotor, 73
  - TMotor, 127
- displayAvailableCommands
  - DrumRobot, 26
- displayHomingStatus
  - HomeManager, 49
- distributeFramesToMotors
  - CanManager, 13
- DrumRobot, 23
  - canManager, 40
  - checkUserInput, 25
  - clearMotorsSendBuffer, 25
  - DeactivateControlTask, 26
  - displayAvailableCommands, 26
  - DrumRobot, 25
  - homeManager, 40
  - idealStateRoutine, 27
  - initializecanManager, 27
  - initializeMotors, 27
  - initializePathManager, 29
  - isBack, 41
  - isReady, 41
  - kbhit, 29
  - maxoncmd, 41
  - MaxonDisable, 30
  - MaxonEnable, 30
  - motors, 41
  - motorSettingCmd, 31
  - NUM\_FRAMES, 41
  - parse\_and\_save\_to\_csv, 33
  - pathManager, 41
  - printCurrentPositions, 33
  - processInput, 34
  - RecieveLoop, 35
  - recvLoopForThread, 35
  - save\_to\_txt\_inputData, 36
  - SendLoop, 36
  - sendLoopForThread, 37
  - SendReadyLoop, 38
  - sensor, 42
  - setMaxonMode, 39
  - stateMachine, 39
  - systemState, 42
  - testManager, 42
  - TIME\_THRESHOLD\_MS, 42
  - tmotorcmd, 42
  - writeFailCount, 42
- ElbowAngle\_hit
  - PathManager, 87
- ElbowAngle\_ready
  - PathManager, 87
- ERR\_SOCKET\_CONFIGURE\_FAILURE
  - CanManager, 21
- ERR\_SOCKET\_CREATE\_FAILURE
  - CanManager, 21
- FixMotorPosition
  - HomeManager, 50
  - TestManager, 97
- fkfun
  - PathManager, 78
- floatToUint
  - TMotorCommandParser, 130
- GenericMotor, 43
  - clearReceiveBuffer, 44
  - clearSendBuffer, 44
  - currentPos, 45
  - currentTor, 45
  - currentVel, 45
  - cwdir, 45
  - desPos, 45
  - desTor, 45
  - desVel, 45
  - interFaceName, 46
  - isConected, 46
  - isHomed, 46
  - Kd, 46
  - Kp, 46
  - nodeId, 46
  - recieveBuffer, 47
  - rMax, 47
  - rMin, 47
  - sendBuffer, 47
  - socket, 47
- GetArr
  - PathManager, 78
- getCanPortStatus
  - CanManager, 14
- getCheck
  - MaxonCommandParser, 62
  - TMotorCommandParser, 130
- getControlMode



- TMotorCommandParser, 131
- getCSPMode
  - MaxonCommandParser, 62
- getCSTMode
  - MaxonCommandParser, 62
- getCSVMode
  - MaxonCommandParser, 62
- getCurrentThreshold
  - MaxonCommandParser, 63
- getDrummingPosAndAng
  - PathManager, 79
- GetDrumPositoin
  - PathManager, 79
- getEnable
  - MaxonCommandParser, 63
- getExit
  - TMotorCommandParser, 131
- getFlowingErrorWindow
  - MaxonCommandParser, 63
- getHomeMode
  - MaxonCommandParser, 64
- getHomeoffsetDistance
  - MaxonCommandParser, 64
- getHomePosition
  - MaxonCommandParser, 64
- getHomingMethodL
  - MaxonCommandParser, 65
- getHomingMethodR
  - MaxonCommandParser, 65
- getMotorPos
  - PathManager, 80
- GetMusicSheet
  - PathManager, 80
- getOperational
  - MaxonCommandParser, 65
- getPosOffset
  - MaxonCommandParser, 66
- getQ1AndQ2
  - PathManager, 81
- getQ3AndQ4
  - PathManager, 82
- getQuickStop
  - MaxonCommandParser, 66
  - TMotorCommandParser, 131
- getStartHoming
  - MaxonCommandParser, 66
- getStop
  - MaxonCommandParser, 67
- getSync
  - MaxonCommandParser, 67
- getTargetPosition
  - MaxonCommandParser, 67
- getTargetTorque
  - MaxonCommandParser, 68
- getTargetVelocity
  - MaxonCommandParser, 68
- getTorqueOffset
  - MaxonCommandParser, 69

- getVelOffset
  - MaxonCommandParser, 69
- getZero
  - TMotorCommandParser, 132
- GLOBAL\_KD\_MAX
  - TMotorCommandParser, 135
- GLOBAL\_KD\_MIN
  - TMotorCommandParser, 135
- GLOBAL\_KP\_MAX
  - TMotorCommandParser, 135
- GLOBAL\_KP\_MIN
  - TMotorCommandParser, 135
- GLOBAL\_P\_MAX
  - TMotorCommandParser, 136
- GLOBAL\_P\_MIN
  - TMotorCommandParser, 136
- GLOBAL\_T\_MAX
  - TMotorCommandParser, 136
- GLOBAL\_T\_MIN
  - TMotorCommandParser, 136
- GLOBAL\_V\_MAX
  - TMotorCommandParser, 136
- GLOBAL\_V\_MIN
  - TMotorCommandParser, 136
- HomeManager, 48
  - canManager, 60
  - displayHomingStatus, 49
  - FixMotorPosition, 50
  - HomeManager, 49
  - HomeTMotor, 50
  - mainLoop, 51
  - maxoncmd, 60
  - MaxonDisable, 52
  - MaxonEnable, 53
  - motors, 60
  - MoveTMotorToSensorLocation, 54
  - PromptUserForHoming, 55
  - RotateTMotor, 56
  - sensor, 60
  - SetMaxonHome, 57
  - setMaxonMode, 58
  - SetTmotorHome, 59
  - systemState, 60
  - tmotorcmd, 60
  - UpdateHomingStatus, 59
- homeManager
  - DrumRobot, 40
- homeMode
  - SystemState, 94
- HomeTMotor
  - HomeManager, 50
- iconnect
  - PathManager, 83
- idealStateRoutine
  - DrumRobot, 27
- ifnames
  - CanManager, 21

- IKfun
  - PathManager, 83
- include/managers/CanManager.hpp, 137
- include/managers/HomeManager.hpp, 138
- include/managers/PathManager.hpp, 139
- include/managers/TestManager.hpp, 141
- include/motors/CommandParser.hpp, 142
- include/motors/Motor.hpp, 143
- include/tasks/DrumRobot.hpp, 144
- include/tasks/SystemState.hpp, 145
- initializeCAN
  - CanManager, 14
- initializcanManager
  - DrumRobot, 27
- initializeMotors
  - DrumRobot, 27
- InitializeParameters
  - TestManager, 97
- initializePathManager
  - DrumRobot, 29
- InputData
  - TestManager, 123
- interFaceName
  - GenericMotor, 46
  - MaxonMotor, 73
  - TMotor, 127
- isBack
  - DrumRobot, 41
- isConected
  - GenericMotor, 46
  - MaxonMotor, 73
  - TMotor, 127
- isConnected
  - CanManager, 21
- isHomed
  - GenericMotor, 46
  - MaxonMotor, 74
  - TMotor, 127
- isReady
  - DrumRobot, 41
- kbhit
  - DrumRobot, 29
  - TestManager, 99
- Kd
  - GenericMotor, 46
  - MaxonMotor, 74
  - TMotor, 127
- Kp
  - GenericMotor, 46
  - MaxonMotor, 74
  - TMotor, 127
- l\_wrist
  - PathManager, 87
- LA
  - PathManager, 87
- left\_inst
  - PathManager, 88
- LF
  - PathManager, 88
- line
  - PathManager, 88
- list\_and\_activate\_available\_can\_ports
  - CanManager, 15
- main
  - SystemState, 94
- mainLoop
  - HomeManager, 51
  - TestManager, 99
- maxoncmd
  - CanManager, 21
  - DrumRobot, 41
  - HomeManager, 60
  - TestManager, 123
- maxonCnt
  - CanManager, 21
- MaxonCommandParser, 61
  - getCheck, 62
  - getCSPMode, 62
  - getCSTMode, 62
  - getCSVMode, 62
  - getCurrentThreshold, 63
  - getEnable, 63
  - getFlowingErrorWindow, 63
  - getHomeMode, 64
  - getHomeoffsetDistance, 64
  - getHomePosition, 64
  - getHomingMethodL, 65
  - getHomingMethodR, 65
  - getOperational, 65
  - getPosOffset, 66
  - getQuickStop, 66
  - getStartHoming, 66
  - getStop, 67
  - getSync, 67
  - getTargetPosition, 67
  - getTargetTorque, 68
  - getTargetVelocity, 68
  - getTorqueOffset, 69
  - getVelOffset, 69
  - parseRecieveCommand, 69
- MaxonDisable
  - DrumRobot, 30
  - HomeManager, 52
- MaxonEnable
  - DrumRobot, 30
  - HomeManager, 53
- MaxonMotor, 70
  - canReceiveId, 72
  - canSendId, 72
  - clearReceiveBuffer, 72
  - clearSendBuffer, 72
  - currentPos, 72
  - currentTor, 72
  - currentVel, 73
  - cwDir, 73

- desPos, [73](#)
- desTor, [73](#)
- desVel, [73](#)
- interFaceName, [73](#)
- isConected, [73](#)
- isHomed, [74](#)
- Kd, [74](#)
- Kp, [74](#)
- MaxonMotor, [71](#)
- nodeId, [74](#)
- recieveBuffer, [74](#)
- rMax, [74](#)
- rMin, [75](#)
- rxPdoIds, [75](#)
- sendBuffer, [75](#)
- socket, [75](#)
- txPdoIds, [75](#)
- mkArr
  - TestManager, [100](#)
- motor\_dir
  - PathManager, [88](#)
- motor\_mapping
  - PathManager, [88](#)
- motors
  - CanManager, [22](#)
  - DrumRobot, [41](#)
  - HomeManager, [60](#)
  - PathManager, [89](#)
  - TestManager, [123](#)
- Motors\_sendBuffer
  - PathManager, [85](#)
- motorSettingCmd
  - DrumRobot, [31](#)
- motorType
  - TMotor, [128](#)
- move
  - TestManager, [101](#)
- MoveTMotorToSensorLocation
  - HomeManager, [54](#)
- MParser
  - PathManager, [89](#)
- multiTestLoop
  - TestManager, [102](#)
- n\_inst
  - PathManager, [89](#)
- nodeId
  - GenericMotor, [46](#)
  - MaxonMotor, [74](#)
  - TMotor, [128](#)
- NUM\_FRAMES
  - DrumRobot, [41](#)
- p
  - PathManager, [89](#)
- P1
  - PathManager, [89](#)
- P2
  - PathManager, [89](#)
- p\_L
  - PathManager, [89](#)
- p\_R
  - PathManager, [90](#)
- parse\_and\_save\_to\_csv
  - DrumRobot, [33](#)
  - TestManager, [105](#)
- parseRecieveCommand
  - MaxonCommandParser, [69](#)
  - TMotorCommandParser, [132](#)
- parseSendCommand
  - TMotorCommandParser, [133](#)
- PathLoopTask
  - PathManager, [85](#)
- PathManager, [76](#)
  - ApplyDir, [77](#)
  - backarr, [86](#)
  - bpm, [86](#)
  - c\_L, [86](#)
  - c\_MotorAngle, [87](#)
  - c\_R, [87](#)
  - canManager, [87](#)
  - connect, [77](#)
  - ElbowAngle\_hit, [87](#)
  - ElbowAngle\_ready, [87](#)
  - fkfun, [78](#)
  - GetArr, [78](#)
  - getDrummingPosAndAng, [79](#)
  - GetDrumPositoin, [79](#)
  - getMotorPos, [80](#)
  - GetMusicSheet, [80](#)
  - getQ1AndQ2, [81](#)
  - getQ3AndQ4, [82](#)
  - iconnect, [83](#)
  - IKfun, [83](#)
  - l\_wrist, [87](#)
  - LA, [87](#)
  - left\_inst, [88](#)
  - LF, [88](#)
  - line, [88](#)
  - motor\_dir, [88](#)
  - motor\_mapping, [88](#)
  - motors, [89](#)
  - Motors\_sendBuffer, [85](#)
  - MParser, [89](#)
  - n\_inst, [89](#)
  - p, [89](#)
  - P1, [89](#)
  - P2, [89](#)
  - p\_L, [89](#)
  - p\_R, [90](#)
  - PathLoopTask, [85](#)
  - PathManager, [77](#)
  - Q1, [90](#)
  - Q2, [90](#)
  - Q3, [90](#)
  - Q4, [90](#)
  - R, [90](#)

- r\_wrist, 90
  - RA, 91
  - RF, 91
  - right\_inst, 91
  - s, 91
  - standby, 91
  - systemState, 91
  - time\_arr, 91
  - total, 92
  - TParser, 92
  - v, 92
  - wrist, 92
  - WristAngle\_hit, 92
  - WristAngle\_ready, 92
  - z0, 92
- pathManager
  - DrumRobot, 41
- printCurrentPositions
  - DrumRobot, 33
- processInput
  - DrumRobot, 34
- PromptUserForHoming
  - HomeManager, 55
- Q1
  - PathManager, 90
- Q2
  - PathManager, 90
- Q3
  - PathManager, 90
- Q4
  - PathManager, 90
- R
  - PathManager, 90
- r\_wrist
  - PathManager, 90
- RA
  - PathManager, 91
- readFramesFromAllSockets
  - CanManager, 16
- recieveBuffer
  - GenericMotor, 47
  - MaxonMotor, 74
  - TMotor, 128
- RecieveLoop
  - DrumRobot, 35
- recvLoopForThread
  - DrumRobot, 35
- recvToBuff
  - CanManager, 16
- restartCanPorts
  - CanManager, 16
- RF
  - PathManager, 91
- right\_inst
  - PathManager, 91
- rMax
  - GenericMotor, 47
- MaxonMotor, 74
  - TMotor, 128
- rMin
  - GenericMotor, 47
  - MaxonMotor, 75
  - TMotor, 128
- RotateTMotor
  - HomeManager, 56
- rxFrame
  - CanManager, 17
- rxPdoIds
  - MaxonMotor, 75
- s
  - PathManager, 91
- save\_to\_txt\_inputData
  - DrumRobot, 36
- sendAndRecv
  - CanManager, 17
- sendBuffer
  - GenericMotor, 47
  - MaxonMotor, 75
  - TMotor, 128
- sendFromBuff
  - CanManager, 18
- SendLoop
  - DrumRobot, 36
  - TestManager, 106
- sendLoopForThread
  - DrumRobot, 37
- SendReadyLoop
  - DrumRobot, 38
- sensor
  - DrumRobot, 42
  - HomeManager, 60
- sensorBit
  - TMotor, 129
- SetMaxonHome
  - HomeManager, 57
- setMaxonMode
  - DrumRobot, 39
  - HomeManager, 58
  - TestManager, 107
- setMotorLimits
  - TMotorCommandParser, 134
- setMotorsSocket
  - CanManager, 18
- setSocketsTimeout
  - CanManager, 19
- setSocketTimeout
  - CanManager, 20
- SetTmotorHome
  - HomeManager, 59
- socket
  - GenericMotor, 47
  - MaxonMotor, 75
  - TMotor, 129
- sockets
  - CanManager, 22

- src/CanManager.cpp, 145
- src/CommandParser.cpp, 152
- src/DrumRobot.cpp, 159
- src/HomeManager.cpp, 171
- src/main.cpp, 179
- src/Motor.cpp, 180
- src/PathManager.cpp, 180
- src/Sensor.cpp, 188
- src/TestManager.cpp, 189
- standby
  - PathManager, 91
- stateMachine
  - DrumRobot, 39
- SystemState, 93
  - homeMode, 94
  - main, 94
  - SystemState, 93
- systemState
  - DrumRobot, 42
  - HomeManager, 60
  - PathManager, 91
  - TestManager, 123
- tempFrames
  - CanManager, 22
- TestArr
  - TestManager, 108
- TestManager, 94
  - canManager, 123
  - dct\_fun, 96
  - FixMotorPosition, 97
  - InitializeParameters, 97
  - InputData, 123
  - kbhit, 99
  - mainLoop, 99
  - maxoncmd, 123
  - mkArr, 100
  - motors, 123
  - move, 101
  - multiTestLoop, 102
  - parse\_and\_save\_to\_csv, 105
  - SendLoop, 106
  - setMaxonMode, 107
  - systemState, 123
  - TestArr, 108
  - TestManager, 96
  - TestStick, 109
  - TestStickLoop, 111
  - tmotorcmd, 123
  - TuningLoopTask, 112
  - TuningMaxonCSP, 115
  - TuningMaxonCST, 117
  - TuningMaxonCSV, 119
  - TuningTmotor, 121
- testManager
  - DrumRobot, 42
- TestStick
  - TestManager, 109
- TestStickLoop
  - TestManager, 111
- time\_arr
  - PathManager, 91
- TIME\_THRESHOLD\_MS
  - DrumRobot, 42
- TMotor, 124
  - clearReceiveBuffer, 125
  - clearSendBuffer, 125
  - currentPos, 126
  - currentTor, 126
  - currentVel, 126
  - cwDir, 126
  - desPos, 126
  - desTor, 126
  - desVel, 127
  - interFaceName, 127
  - isConected, 127
  - isHomed, 127
  - Kd, 127
  - Kp, 127
  - motorType, 128
  - nodeId, 128
  - recieveBuffer, 128
  - rMax, 128
  - rMin, 128
  - sendBuffer, 128
  - sensorBit, 129
  - socket, 129
  - TMotor, 125
- tmotorcmd
  - CanManager, 22
  - DrumRobot, 42
  - HomeManager, 60
  - TestManager, 123
- TMotorCommandParser, 129
  - floatToUInt, 130
  - getCheck, 130
  - getControlMode, 131
  - getExit, 131
  - getQuickStop, 131
  - getZero, 132
  - GLOBAL\_KD\_MAX, 135
  - GLOBAL\_KD\_MIN, 135
  - GLOBAL\_KP\_MAX, 135
  - GLOBAL\_KP\_MIN, 135
  - GLOBAL\_P\_MAX, 136
  - GLOBAL\_P\_MIN, 136
  - GLOBAL\_T\_MAX, 136
  - GLOBAL\_T\_MIN, 136
  - GLOBAL\_V\_MAX, 136
  - GLOBAL\_V\_MIN, 136
  - parseRecieveCommand, 132
  - parseSendCommand, 133
  - setMotorLimits, 134
  - uintToFloat, 135
- total
  - PathManager, 92
- TParser

- PathManager, [92](#)
- TuningLoopTask
  - TestManager, [112](#)
- TuningMaxonCSP
  - TestManager, [115](#)
- TuningMaxonCST
  - TestManager, [117](#)
- TuningMaxonCSV
  - TestManager, [119](#)
- TuningTmotor
  - TestManager, [121](#)
- txFrame
  - CanManager, [20](#)
- txPdoIds
  - MaxonMotor, [75](#)
- uintToFloat
  - TMotorCommandParser, [135](#)
- UpdateHomingStatus
  - HomeManager, [59](#)
- v
  - PathManager, [92](#)
- wrist
  - PathManager, [92](#)
- WristAngle\_hit
  - PathManager, [92](#)
- WristAngle\_ready
  - PathManager, [92](#)
- writeFailCount
  - DrumRobot, [42](#)
- z0
  - PathManager, [92](#)