**프레임**

|  |
| --- |
| 1. **파일 위치**    1. ㅁ 2. **클래스명**    1. ㅁ 3. **개요**    1. ㅁ 4. **Public 함수**    1. ㅁ 5. **protected 함수**    1. ㅁ 6. **중요 변수**    1. ㅁ 7. **기타 사항**    1. ㅁ |

|  |
| --- |
| 1. **파일 위치**    1. src\utility\CommonData.h 2. **클래스명**    1. Data 3. **개요**    1. Data를 다루기 위한 추상 클래스. 4. **Public 함수**    1. serialize(char\*\* param) : 순수 추상 함수. 클래스가 가지고 있는 데이터들을 직렬화하여 param에게 반환.    2. deserialize(char\*\* param) : 순수 추상 메소드. param으로 들어온 직렬화된 데이터를 분해하여 클래스 데이터로 저장. 5. **protected 함수**    1. Initialize(char\*\* param, int size) : 직렬화하기 전 param을 생성하고 클래스를 구분할 수 있는 정보를 저장하는 메소드.    2. fill(char\*\* param, int start, int size, void\* src) : param내부에 실질적으로 데이터를 저장하는 메소드. char배열에서 시작 인덱스, 저장하고자 하는 변수의 크기, 저장하고자 하는 변수의 address를 매개변수로 전달하면 param에 데이터를 저장하고 마지막 index를 반환.    3. extract(char\*\* param, int start, int size, void\* dst) : 직렬화된 데이터 param으로부터 데이터를 추출하는 메소드. Param에서 원하는 데이터가 시작하는 지점의 인덱스, 데이터의 크기, 저장하고자 하는 변수의 address를 매개변수로 전달하면 param으로부터 데이터를 추출하고, 추출된 데이터의 마지막 index를 반환. 6. **중요 변수**    1. DataType \_type : 데이터의 형식을 구분할 수 있는 정보를 담고있는 변수. 7. **기타 사항**    1. 자세한 구현 방법은 이미 구현된 하위 클래스의 구현을 참고. |

**비중요 클래스 -> 실무자들이 굳이 읽을 필요 없다.**

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\Common.h 2. **클래스명** 3. **개요**    1. 전 프로그램에 걸친 공용 데이터를 저장한 파일 4. **Public 함수** 5. **protected 함수** 6. **중요 변수**    1. UINT : uint8\_t를 재정의한 변수. 서로간의 호환이 가능    2. DELAYTIME : 아두이노 루프 구문의 간격을 조절할 때 사용    3. BAUDRATE : Serial계열 통신 시 Stream간 데이터 전송 속도(9600, 115200 등) 7. **기타 사항**    1. Const가 붙은 변수는 프로그램 진행 중 결코 변경되지 않는다.    2. Static이 붙은 변수는 프로그램 전체에서 항상 유일하게 존재한다. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\PinManager.h 2. **클래스명**    1. PinManager 3. **개요**    1. 아두이노 보드와 연결된 핀의 정보를 보관    2. 저장되는 핀 번호가 중복 시 중복 여부를 출력 -> 오류 전달로 변경 필요    3. getInstance()함수를 통해서만 접근이 가능    4. 핀 번호만을 통해 on-off, high-low 설정 가능 4. **Public 함수**    1. getInstance() : PinManager를 불러오는 함수. 모든 프로그램 내부에서는 항상 이 함수를 통해 접근해야한다.    2. PinRegister(UINT pinNum) : 아두이노와 연결된 핀 정보를 저장한다. INPUT-OUTPUT은 설정하지 않는다. 디바이스 초기부터 핀의 in-out을 설정할 필요가 없다면 이 함수를 사용하면 된다. PinNum변수에 아두이노 보드와 연결되는 핀 번호를 입력.    3. PinRegister(UINT pinNum, UINT mode) : PinRegister(UINT pinNum)과 동일한 역할을 한다. 핀의 INPUT-OUTPUT을 설정할 수 있다. 디바이스 초기부터 핀의 in-out을 설정해야한다면 이 함수를 사용하면 된다. pinNum변수에 아두이노 보드와 연결되는 핀 번호를 입력. mode에 INPUT-OUTPUT을 입력.    4. setPinHigh(UINT pinNum) : 핀 상태를 HIGH로 설정. pinNum변수에 핀 번호를 입력.    5. setPinLow(UINT pinNum) : 핀 상태를 LOW로 설정. PinNum변수에 핀 번호를 입력. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항**    1. Device클래스 이외의 센서를 연결할 때, PinRegister로 등록하지 않았을 경우 작동하지 않는다. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\streamRecorder.h 2. **클래스명**    1. StreamRecorder 3. **개요**    1. 출력을 담당하는 클래스. Stream의 하위 클래스는 모두 사용 가능(Serial, SoftwareSerial, 블루투스 등). 파일 입출력도 가능할것으로 짐작됨. 다른 Stream을 추가하지 않았다면 Serial을 기본 출력 Stream으로 사용. 4. **Public 함수**    1. getInstance() : StreamRecorder를 불러오는 함수. 모든 프로그램 내부에서는 항상 이 함수를 통해 접근해야 한다.    2. print(String str) : String 문자열을 Stream으로 출력. 출력하고 싶은 문장이 있으면 이 함수를 사용하면 된다.    3. setStream(Stream\* stream) : 기본 Stream인 Serial이외의 다른 Stream을 연결하고자 한다면 이 함수를 사용하면 된다.    4. PrintFromBuffer(String str) : 미구현 함수. 5. **protected 함수**    1. ㅁ 6. **중요 변수**    1. ㅁ 7. **기타 사항**    1. 출력에 관여하는 Stream을 관리하는 함수이기 때문에, Serial에서 블루투스로 통신을 바꾸더라도 이 클래스의 setStream()함수만 사용하면 바로 이용 가능. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\Device.h 2. **클래스명**    1. Device 3. **개요**    1. 아두이노에 연결되는 모든 센서들의 공통부분을 추출한 클래스. 추상적인 개념으로 추출했지만 추상 클래스는 아니다. Device클래스를 상속받아 하위 클래스를 구현해야 핀 정보 관리가 편리하다. 부모 클래스 초기화 시 bundlePin()함수를 사용하면 편리하다. 4. **Public 함수**    1. Device(UINT\* pPinArr, UINT pinCount) : 연결된 핀 번호 배열과 개수를 입력받으면 PinManager에 핀 정보를 등록한다.    2. Device(UINT\* pPinArr, UINT\* pModeArr, UINT pinCount) : 연결된 핀 변호 배열과 핀의 INPUT-OUTPUT정보를 담은 배열, 개수를 입력받으면 PinManager에 핀 정보를 등록한다. 5. **protected 함수**    1. registerPin(UINT\* pPinArr, UINT pinCount) : 실질적으로 핀 정보를 PinManager에 등록하는 함수. Device()초기화에 등록할 수 없는 핀 정보(pin, pinmode의 개수가 맞지 않을 경우)에 사용.    2. registerPin(UINT\* pPinArr, UINT\* pModeArr, UINT pinCount) : 실질적으로 핀 정보를 PinManager에 등록하는 함수. Device()초기화에 등록할 수 없는 핀 정보(pin, pinmode의 개수가 맞지 않을 경우)에 사용.    3. bundlePin(int args, ...) : 가변인자를 사용했기 때문에 사용 시에는 핀번호(bundlePin(7, 8, 9, 10))만 연속으로 입력하면 입력된 정보를 배열로 반환해준다. registerPin()에 배열 정보를 입력하기 위해 사용. args변수에 가변 인자의 개수를 적어준다.    4. bundleMode(int args, ...) : 가변인자를 사용했기 때문에 사용 시에는 핀 모드 번호(bundlePin(INPUT, INPUT, OUTPUT, INPUT))만 연속으로 입력하면 입력된 정보를 배열로 반환해준다. registerPin()에 배열 정보를 입력하기 위해 사용. args변수에 가변 인자의 개수를 적어준다. 6. **중요 변수** 7. **기타 사항**    1. 추상적 개념을 모은 클래스이지만 추상 클래스는 아니다. Device(bundlePin(1, 2, 3), bundlePin(INPUT, OUTPUT, INPUT), 3)의 형식으로 작성 |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\DistanceMeasurement.h 2. **클래스명**    1. DistanceMeasurement 3. **개요**    1. 거리 측정 센서들의 상위 클래스. Device 클래스를 상속. measure(), getDistance()함수만을 사용. 하위 클래스를 구현 시 measure()함수를 구체화시켜야 함. 4. **Public 함수**    1. measure() : 거리 측정 함수. 거리를 측정하기 위한 선행 작업이 모두 여기에 들어가야 한다.    2. getDistance() : measure()로 측정된 거리를 반환하는 함수. 5. **protected 함수**    1. setDistance(int dist) : 하위 클래스에서 측정된 거리를 private로 선언된 \_distance함수에 저장하는 함수. 6. **중요 변수** 7. **기타 사항** |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\AngleMotor.h 2. **클래스명**    1. AngleMotor 3. **개요**    1. 각도 설정이 가능한 모터류의 추상 클래스 4. **Public 함수**    1. SetAngle(int angle) : 추상 메서드. 원하는 각도를 설정할때 사용.    2. addAngle(int angle) : 추상 메서드. 현재 각도에서 각도를 수정할때 사용. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항**    1. 순수 추상 함수만 존재하는 클래스로 실제 사용은 실구현된 하위 클래스를 사용해야 한다. 각도를 설정할 수 있는 하드웨어의 지침을 위한 추상 클래스. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibrary\Camera.h 2. **클래스명**    1. Camera 3. **개요**    1. 일반적인 카메라와 관련된 기능을 추상화 시킨 클래스. 4. **Public 함수**    1. Camera(…, …, int width, int height) : 카메라가 읽어올 수 있는 픽셀의 크기    2. getWidth() : 픽셀의 가로 크기    3. getHeight() : 픽셀의 세로 크기    4. getPixels() : 총 픽셀의 수    5. scan() : 순수 추상 함수. 카메라 하드웨어의 각 픽셀 레지스터로부터 raw 데이터를 읽어온다.    6. read(LinearData\* data) : scan함수로부터 읽어온 raw픽셀 데이터를 LinearData형식으로 반환한다. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항**    1. 중요 메서드들은 순수 추상함수이므로 실 사용시 구현한 하위 클래스를 사용해야 한다. |

**중요클래스**

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\UltraSonicSensor.h 2. **클래스명**    1. UltraSonicSensor 3. **개요**    1. 초음파 센서. DistanceMeasurement클래스를 상속. 4. **Public 함수**    1. UltraSonicSensor(UINT trig, UINT echo) : trig변수는 초음파 센서의 trig핀과 연결된 핀 번호를 입력, echo변수는 초음파 센서의 echo핀과 연결된 번호를 입력. 5. **protected 함수** 6. **중요 변수**    1. \_sensingTime = 10 : 10ms단위로 거리 측정.    2. \_velocity : 10ms단위로 거리를 측정할 경우 58cm/s의 속도로 이동.    3. \_trig : 초음파 센서의 trig핀과 연결된 핀 번호    4. \_echo : 초음파 센서의 echo핀과 연결된 핀 번호 7. **기타 사항** |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\TfLidarSensor.h 2. **클래스명**    1. TfLidarSensor 3. **개요**    1. TF Mini Lidar 센서를 위한 클래스. DistanceMeasurement클래스를 상속받는다.    2. 참고 링크 : <https://learn.sparkfun.com/tutorials/tfmini---micro-lidar-module-hookup-guide?_ga=2.84740177.992535469.1584420037-1081340213.1580364895> 4. **Public 함수**    1. TfLidarSensor(UINT txPin, UINT rxPin) : txPin에는 링크 속 센서 이미지의 tx와 연결된 핀의 번호, rxPin에는 링크 속 센서 이미지의 rx와 연결된 핀 번호 입력. 5. **protected 함수** 6. **중요 변수**    1. Stream\* \_stream은 Lidar센서와 통신하는 SoftwareSerial을 의미하는 스트림이다. 7. **기타 사항**    1. 이 클래스는 TFMini에서 지원해주는 기본 코드를 Wrapping한 클래스이다. 세부 코드의 자세한 내용은 개요의 참고 링크에서 지원하는 코드를 참조. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\MotorDriver.h 2. **클래스명**    1. MotorDriver 3. **개요**    1. Motor Driver사용을 위한 클래스. 모터의 정역제어에 사용. 모터에 가해지는 전압을 조절할 경우에도 사용 가능. 단순히 모터에 동력 전달을 제어할 목적을 위해서라면 Relay을 사용하는 것을 추천.    2. forwardPin과 IN1, backwardPin과 IN2, powerPin과 EN\*와 연결하기로 약속한다.    3. VMS, GND를 외부 전원과 연결. 출력부의 +- 방향 여부는 측정 필요.    4. 참고 링크 : <https://devicemart.blogspot.com/2019/07/l298n.html> 4. **Public 함수**    1. MotorDriver(UINT forwardPin, UINT backwardPin, UINT powerPin) : forwardPin과 IN1, backwardPin과 IN2, powerPin과 EN\*와 연결하기로 약속한다. VMS, GND를 외부 전원과 연결. 파워 초기값은 0.    2. setPower(int power) : power에 입력되는 값이 양수일 경우 정방향으로, 음수일 경우 역방향으로 자동 설정. 초기값 0. 절대값 255이내의 값만 가능하고 초과시 255로 변환. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항**    1. 정방향인지 역방향인지를 나타내는 state관련 함수 미구현. 필요시 추가 예정. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\Relay.h 2. **클래스명**    1. Relay 3. **개요**    1. 외부 전원과 연결된 장치의 on-off를 제어하기 위한 장치. 모터의 정역, 전압 제어를 위해서는 MotorDriver사용을 추천. Relay는 연결에 따라 기본 on-off상태가 결정되므로 상태 변환 시 기본 연결 상태 고려해야 한다. 기본적으로 NO와 연결하기로 약속한다.    2. 참고 링크 : <https://k96-ozon.tistory.com/44>    3. 초기 생성 시 Pull Up – Pull Down 저항 설정으로 인한 코드 수정 고려. 4. **Public 함수**    1. Relay(UINT pin) : +-를 제외한 마지막 핀과 연결한 핀 번호를 입력.    2. on() : NO와 연결 후 호출 시 내부 회로 연결.    3. off() : NO와 연결 후 호출 시 내부 회로 단절. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항**    1. on-off를 위한 state변수 미구현. 필요 시 추가 예정. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibaray\ServoMotor.h 2. **클래스명**    1. ServoMotor 3. **개요**    1. 서보 모터 조작을 위한 클래스. 최대 180도까지 가능하며 절대값 180도를 초과 시 180으로 modulus 연산한 결과값을 각도로 설정한다. 서보 모터에 따라 0값이 가리키는 각이 다르기 때문에 디바이스마다 영점 조절이 필요하다.    2. 연결 정보 참고 링크 : <https://codingrun.com/110> 4. **Public 함수**    1. ServoMotor(UINT pulsePinNum) : PWM과 연결된 핀 번호를 입력한다.    2. setAngle(int angle) : angle값으로 서보 모터의 팔 각도를 설정한다. 180초과 시 modulus연산한 값을 설정한다.    3. addAngle(int addAngle) : 현재 \_angle값에 angle값을 더한 각을 설정한다. 두 각의 합이 180을 초과한다면 modulus연산한 결과값을 설정한다. 5. **protected 함수** 6. **중요 변수**    1. \_angle : 현재 각도를 나타내는 값. 7. **기타 사항**    1. 기존 아두이노에서 제공하는 Servo를 Wrapping하는 클래스. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibrary\amg8833.h 2. **클래스명**    1. amg8833 3. **개요**    1. 열감지 카메라 amg8833을 위한 클래스. Camera클래스를 상속. Adafruit\_AMG88xx클래스를 wrapping하는 클래스. 4. **Public 함수**    1. SetThreshold(int temperature) : 데이터를 출력 시 원하는 온도 이상의 값만을 보기위한 임계값을 설정하는 함수. 이 함수 설정 이후 setThreshold()를 실행하면 threshold이상의 값은 정상적으로, 그 이하의 값은 0으로 처리된다.    2. onThreshold(), offThreshold() : Threshold모드를 설정하는 함수.    3. onTrigger(), offTrigger() : threshold값 이상의 픽셀은 1, 그 이하의 픽셀은 0으로 처리하여 출력하도록 설정. 5. **protected 함수** 6. **중요 변수**    1. Buffer \_buffer : union형태로 구현하여 메모리를 최소화. 7. **기타 사항**    1. 여기서 구현한 열감지 카메라는 온도값을 analog하게 받아오기 때문에 온도차가 심한 부분의 정확한 온도를 측정하기 위해서는 최소한의 관측 시간이 필요하다. 충분한 시간을 주지 않고 scan, read할 경우 온도 데이터가 한 frame씩 밀리는 듯한 현상이 발견된다.    2. 여기서 최소한의 관측 시간은 500ms로 설정.    3. Trigger모드는 Threshold모드를 반드시 설정하지만 Threshold모드는 꼭 Trigger모드일 필요는 없다. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibrary\ElectronicSpeedController.h 2. **클래스명**    1. ElectronicSpeedController 3. **개요**    1. ElectronicSpeedController, 일명 ESC라 불리는 하드웨어를 위한 클래스. 4. **Public 함수**    1. ElectronicSpeedController(int pwmPin) : ESC의 pwm과 연결된 선 번호 입력.    2. setPower(int power) : 0~100% 사이의 원하는 힘의 값 입력. 5. **protected 함수** 6. **중요 변수**    1. Servo \_esc : 이 클래스는 기본적으로 Servo클래스를 이용하여 구현한 클래스. 7. **기타 사항**    1. ESC에는 원래 Calibration이라는 기능이 달려있지만, 본 클래스에서는 그 기능은 구현하지 않는다. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibrary\Encoder.h 2. **클래스명**    1. Encoder 3. **개요**    1. 정밀한 각도를 측정하는 Encoder를 위한 클래스. 4. **Public 함수**    1. Encoder(int pinA, int pinB, int PPR) : A 상 정보를 알려주는 선과 연결된 핀 번호, B상 정보를 알려주는 선과 연결된 핀 번호, 1회전당 pulse값을 매개변수로 전달해야 한다. PPR의 경우 datasheet에 공지한 값의 2배를 입력해야 한다.    2. Encoder(int pinA, int pinB, int PPR) : A 상 정보를 알려주는 선과 연결된 핀 번호, B상 정보를 알려주는 선과 연결된 핀 번호, Z상 정보를 알려주는 선과 연결된 핀 번호, 1회전당 pulse값을 매개변수로 전달해야 한다. PPR의 경우 datasheet에 공지한 값의 2배를 입력해야 한다.    3. getAngle() : 현재 회전한 각도를 0~360도 사이로 알려준다.    4. tick() : 1펄스가 입력될 경우 각도를 계산해주는 메소드.    5. reset() : Z 상에서 데이터가 입력될 경우 각도값을 초기화 해주는 메소드. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항**    1. Encoder클래스의 경우 interrupt방식으로 각도를 계산하도록 하고 있는데, 아두이노에서 interrupt를 발생시키기 위해서는 추가적으로 구현해야 하는 것들이 몇가지 존재한다. 이 구현 과정에 추가적으로 소모될 시간이 매우 길것으로 판단되어 Encoder클래스는 기존의 클래스들과는 다른 방식으로 사용해야 한다.    2. Arduino의 void setup()구문 내에서 attachInterrupt()를 이용하여 tick()이나 reset()함수를 등록하여 사용. 자세한 방법은 링크 참조. <https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/>    3. Ino파일에서 void func() { e->tick(); }형식의 함수를 작성 후, attachInterrupt(pin, func, config) 형식으로 작성해야 작동.    4. 너무 빠른 속도로 회전할 경우 오차가 발생하는 현상이 발견. 고속의 상황에서 사용하지 말 것. |
|  |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibrary\ContinuousServoMotor.h 2. **클래스명**    1. ContinuousServoMotor 3. **개요**    1. 무한 회전 서보모터(360도 서보모터) 사용을 위한 클래스. 4. **Public 함수**    1. ContinuousServoMotor(int pwmPin) : 서보모터의 pwm핀과 연결된 번호.    2. rotate(bool cw) : true일 경우, clock-wise방향으로 회전. False일 경우 ccw한 방향으로 회전.(디버깅이 안된 상태이므로 반대로 작동할 수 있음.)    3. stop() : 회전하던 모터가 정지. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항**    1. 무한 회전 서보모터는 독립적으로는 정밀한 각도 제어가 불가능한 하드웨어. 일정한 시간동안 동일한 전압으로 같은 시간을 작동시켜도 회전하는 각도가 밀림이 발생하는 현상으로 인해 미세하게 다른것을 확인.    2. 정밀한 각도 제어를 위해서는 encoder와 함께 사용하는 것을 추천. |

|  |
| --- |
| 1. **파일 위치**    1. src\utility\CommonData.h 2. **클래스명**    1. CoordinateData 3. **개요**    1. 카르테시안 좌표 데이터를 위한 클래스. 4. **Public 함수**    1. get함수들 : 값을 가져오는 함수.    2. set함수들 : 값을 저장하는 함수. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항** |

|  |
| --- |
| 1. **파일 위치**    1. src\utility\CommonData.h 2. **클래스명**    1. ShpericalData 3. **개요**    1. 구면 좌표계 데이터를 위한 클래스. 4. **Public 함수**    1. get함수들 : 데이터를 가져오는 함수.    2. set함수들 : 데이터를 저장하는 함수. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항** |

|  |
| --- |
| 1. **파일 위치**    1. src\utility\CommonData.h 2. **클래스명**    1. LinearData 3. **개요**    1. 1차원 데이터를 다루기 위한 클래스. 4. **Public 함수**    1. getIndexValue(unsigned int index) : index에 위치한 데이터를 반환.    2. storeData(int\* srcBuffer, unsigned int size) : 저장하기를 원하는 데이터의 주소와 그 크기를 매개변수로 전달.    3. storeData(char\* srcBuffer, unsigned int size) : 저장하기를 원하는 데이터의 주소와 그 크기를 매개변수로 전달. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항**    1. 데이터는 int형 변수로 저장된다. |

|  |
| --- |
| 1. **파일 위치**    1. src\imLibrary\LinearMotor.h 2. **클래스명**    1. LinearMotor 3. **개요**    1. 리니어 모터 클래스 4. **Public 함수**    1. LinearMotor(int pwmPin) : pwm선과 연결된 핀 번호.    2. LinearMotor(int pwmPin, int min, int max) : pwm과 연결된 핀 번호, pulse의 최소와 최대 값. 기본은 1000~2000.    3. setLength(int length) : 원하는 길이의 비율 입력. 0 ~ 100% 사이값. 5. **protected 함수** 6. **중요 변수** 7. **기타 사항**    1. Servo클래스를 Wrapping한 클래스. |