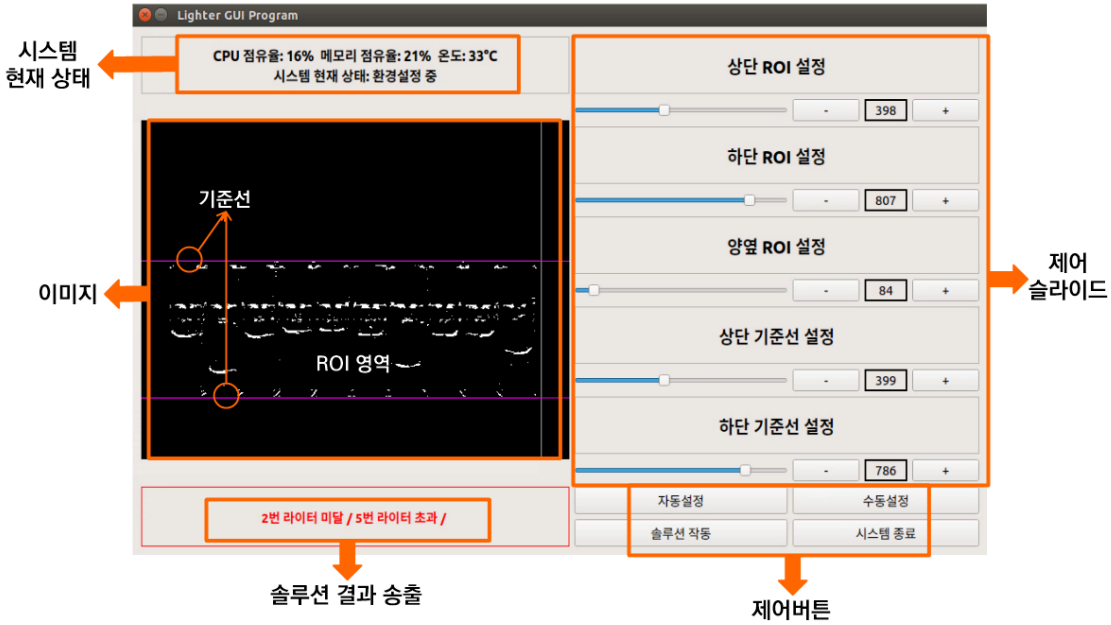
진척상황 보고

1. 가스양

1. GUI 터치 프로그램 제작 완료

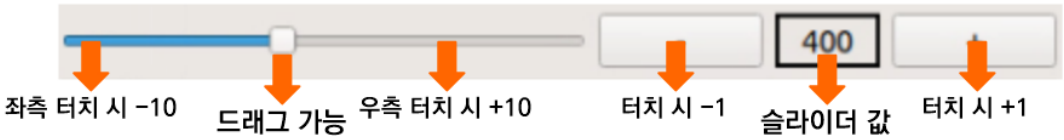
- 라즈베리파이 7인치 LCD 터치스크린과 호환되는 현장 제어용 GUI 프로그램을 제작했습니다.



- 프로그램은 위 사진과 같이 구성돼있습니다.
  - **시스템 현재 상태** : Jetson nano 및 프로그램의 현재 상태를 나타냅니다.
    퍼센티지 표기보다 작업자가 쉽게 이해할 수 있도록 단계별 표기를 사용합니다.
    - **CPU 점유율**: 프로그램이 차지하는 CPU 리소스의 비율을 의미합니다.
      - 높음: 80~100%
        - 10시간 작동해도 해당 수치 나타나지 않음.

        - 만일 '높음' 발견 시 `CSI_Camera` 모듈의 멀티스레딩의 `join()` 문제일 가능성 높음.
    - 정상: 80% 미만
- **메모리 점유율**: 프로그램이 차지하는 메모리의 비율을 의미합니다.
    - 높음: 80~100%
        - 10시간 작동해도 해당 수치 나타나지 않음.
        - 만일 '높음' 발견 시 `CSI_Camera` 모듈의 멀티스레딩의 `join()` 문제일 가능성 높음.
    - 정상: 80% 미만
- **온도**: Jetson nano의 내장 온도 센서 (AO)의 값을 의미합니다.
    - 위험: 90.0 ~ 105.0°C (105°C 이상 시 쓰로틀링으로 인한 작동중지 위험 있음.)
    - 높음: 70.0 ~ 89.9°C
    - 정상: 25.0 ~ 69.9°C
    - 낮음: 24.9°C 미만. (0°C 미만 시 방전으로 인한 작동중지 위험 있음.)
- `프로그램 현재 상태` : 프로그램의 현재 상태를 나타냅니다.
    - 환경설정 중: 우측의 버튼 또는 슬라이더를 조정하는 단계입니다.
    - 작동 중: 외부로부터 계속해서 신호를 받아서 라이터 불량 여부를 판단하는 단계입니다.

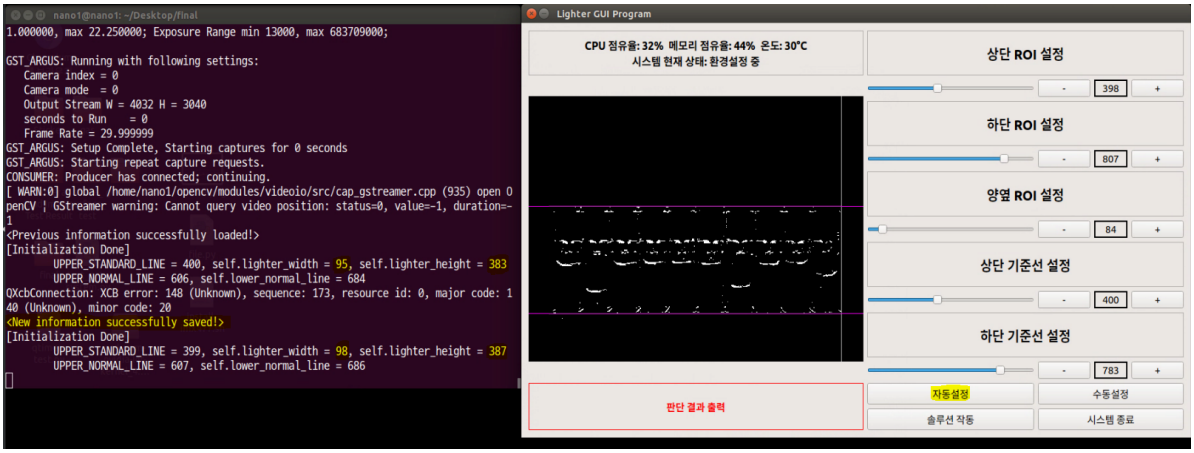
- `이미지` : 사용자가 우측의 제어부분을 눈으로 보면서 제어할 수 있도록 이진화된 카메라 이미지를 보여주는 부분입니다.
- `솔루션 결과` : 매 라이터 세트를 판단한 결과를 송출합니다.
- `제어 슬라이드 및 버튼` : `이미지` 부분의 기준선과 ROI를 수동 또는 자동 설정하거나 시스템을 종료하거나 판단을 수행하는 부분입니다.



- `ROI 설정` : ROI(Region of Interest)를 설정하는 부분입니다. **관심있는 영역** (라이터 연료 탱크 부분)**을 제외한 다른 모든 부분의 픽셀을 0으로 바꿔 검은 화면**으로 바꿔버립니다. 우측의 제어 슬라이드와 버튼을 터치해서 설정할 수 있습니다.
- `기준선 설정` : 라이터의 높이를 측정하기 위해 반드시 상단과 하단의 기준선을 알아야 합니다.
    - **상단 기준선**: 라이터의 인서트 부분
    - **하단 기준선**: 라이터 세트의 플라스틱 트레이 부분
    - 기준선은 두 가지 방법으로 구할 수 있습니다.

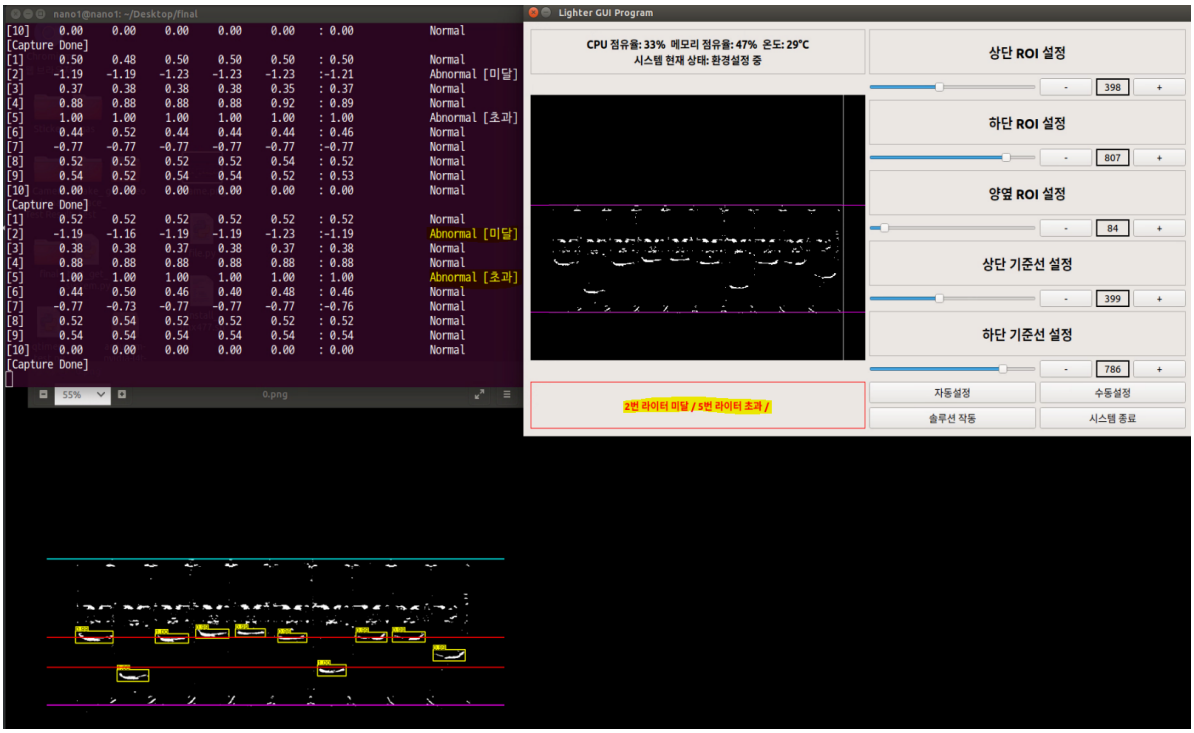
        - **수동설정**: 사용자가 수동으로 설정한 슬라이드의 값을 통해 기준선을 설정합니다.
        - **자동설정**: 내부 알고리즘을 통해 자동으로 두 기준선을 추출합니다.
        - 만일 **자동설정** 결과의 정확도가 떨어진다면 **수동설정**으로 이를 보완합니다.
        - 매 설정 시 ① **상단 기준선**, ② **하단 기준선**, ③ **상단 ROI 선**, ④ **하단 ROI 선**, ⑤ **양옆 ROI 선** 결과를 저장합니다. 저장한 정보는 다음 프로그램 실행 시 불러온 뒤 자동으로 설정합니다.
    - `솔루션 작동` : 디버깅 기능입니다. 외부 신호로 작동하기 전까지는 해당 버튼을 눌러 작동합니다.
    - `시스템 종료` : Jetson nano의 전원을 종료합니다.

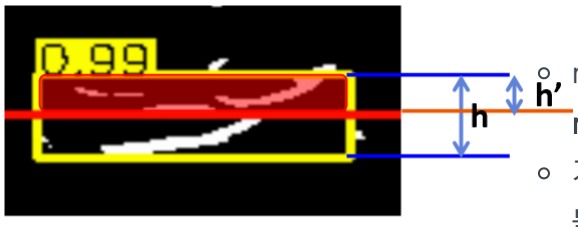
- Jetson nano의 전원이 들어오면 미리 스케쥴러(`Crontab`)에 등록된 쉘스크립트에 의해 프로그램이 실행합니다.
- 프로그램은 실행되면서 이전에 기록해둔 라이터의 정보를 읽어온 뒤 기본 설정을 완료합니다.
    - 좌측에 `<Previous information successfully loaded!>` 라는 문구와 함께 설정값이 출력됩니다.



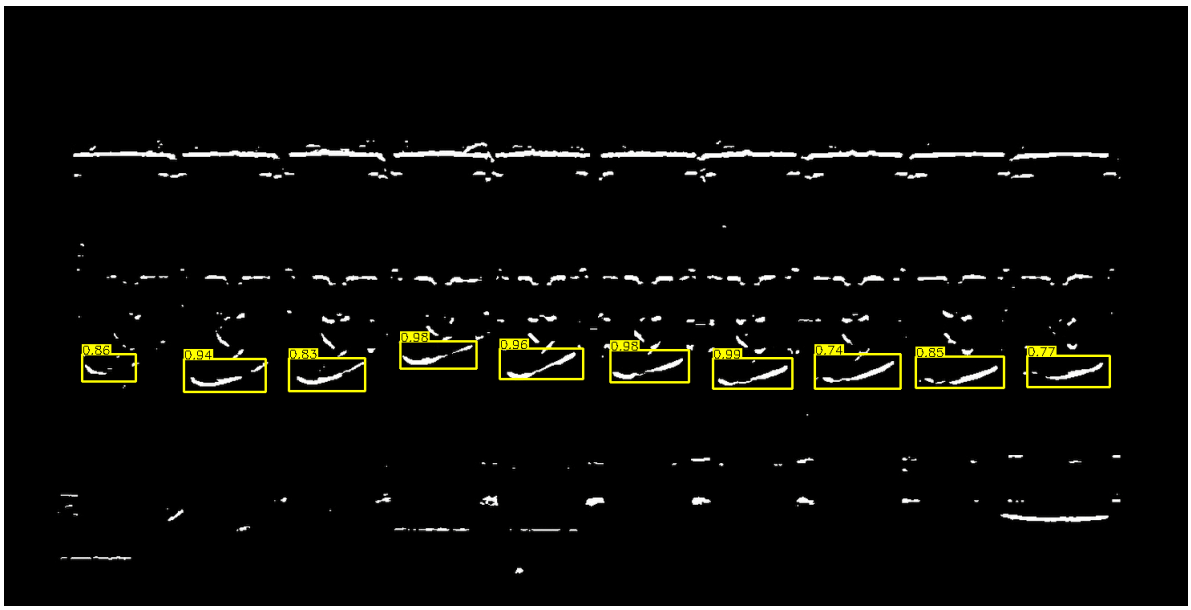
- `자동설정` 버튼을 누르면 라이터의 인서트를 인식한 뒤 상단 기준선과 하단 기준선을 계산합니다.

- 이전 결과와 비교했을 때 약 3, 4픽셀 차이가 생겼지만, 기준선의 두께 내에서 정확히 잡아내고 있습니다.



- **솔루션 작동** 버튼을 누르면 k장(약 5장~10장)의 사진을 찍은 뒤 단일 결과를 반환합니다.
  - 

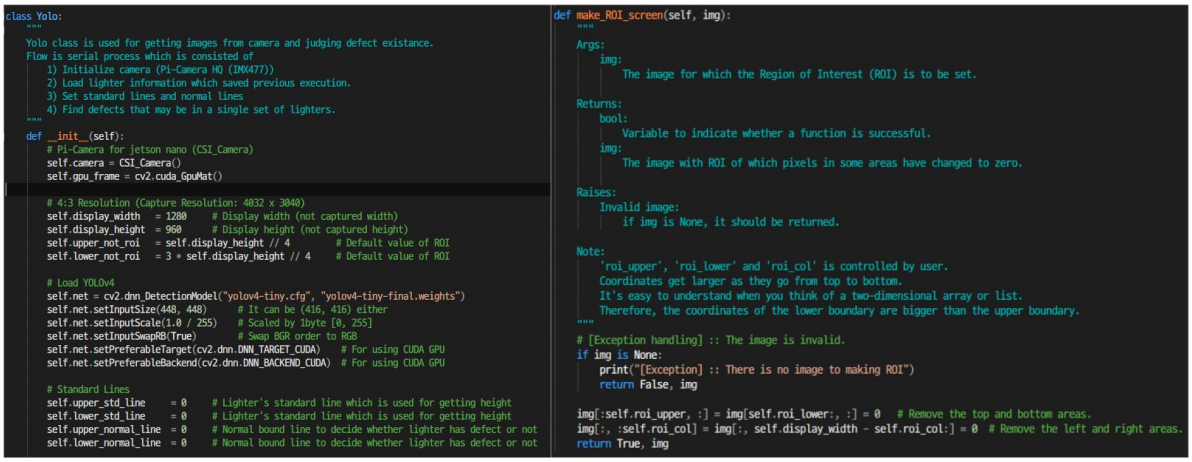
    오류율=h'/h
    n번째 라이터에 대한 k장의 사진.
    k개의 오류율의 평균으로 판단함.
  - n번째 라이터에 대한 k장의 사진을 통해 **n번째 라이터의 오류율**을 계산합니다.
  - 기준치 이상의 오류율을 가진 라이터는 불량품이라고 판단합니다.
  - 위 사진에서는 *2번째 라이터가 미달*, *5번째 라이터가 초과*인 것으로 판단됐습니다.



- 라이터가 흔들리는 상황에서 **인식 정확도**에는 크게 차이가 없으며 **높이에는 차이가 없음**을 확인했습니다.

2. 주석 및 개발 매뉴얼

1. 주석 작성 완료



```
class Yolo:
    """
    Yolo class is used for getting images from camera and judging defect existance.
    Flow is serial process which is consisted of
        1) Initialize camera (Pi-Camera HQ (IMX477))
        2) Load lighter information which saved previous execution.
        3) Set standard lines and normal lines
        4) Find defects that may be in a single set of lighters.
    """
    def __init__(self):
        # Pi-Camera for jetson nano (CSI_Camera)
        self.camera = CSI_Camera()
        self.gpu_frame = cv2.cuda_GpuMat()

        # 4:3 Resolution (Capture Resolution: 4032 x 3040)
        self.display_width   = 1280     # Display width (not captured width)
        self.display_height  = 960      # Display height (not captured height)
        self.upper_not_roi   = self.display_height // 4         # Default value of ROI
        self.lower_not_roi   = 3 * self.display_height // 4     # Default value of ROI

        # Load YOLOv4
        self.net = cv2.dnn_DetectionModel("yolov4-tiny.cfg", "yolov4-tiny-final.weights")
        self.net.setInputSize(448, 448)      # It can be (416, 416) either
        self.net.setInputScale(1.0 / 255)    # Scaled by 1byte [0, 255]
        self.net.setInputSwapRB(True)        # Swap BGR order to RGB
        self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA)    # For using CUDA GPU
        self.net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)  # For using CUDA GPU

        # Standard Lines
        self.upper_std_line     = 0    # Lighter's standard line which is used for getting height
        self.lower_std_line     = 0    # Lighter's standard line which is used for getting height
        self.upper_normal_line  = 0    # Normal bound line to decide whether lighter has defect or not
        self.lower_normal_line  = 0    # Normal bound line to decide whether lighter has defect or not
```

```
def make_ROI_screen(self, img):
    """
    Args:
        img:
            The image for which the Region of Interest (ROI) is to be set.

    Returns:
        bool:
            Variable to indicate whether a function is successful.
        img:
            The image with ROI of which pixels in some areas have changed to zero.

    Raises:
        Invalid image:
            if img is None, it should be returned.

    Note:
        'roi_upper', 'roi_lower' and 'roi_col' is controlled by user.
        Coordinates get larger as they go from top to bottom.
        It's easy to understand when you think of a two-dimensional array or list.
        Therefore, the coordinates of the lower boundary are bigger than the upper boundary.
    """
    # [Exception handling] :: The image is invalid.
    if img is None:
        print("[Exception] :: There is no image to making ROI")
        return False, img

    img[:self.roi_upper, :] = img[self.roi_lower:, :] = 0   # Remove the top and bottom areas.
    img[:, :self.roi_col] = img[:, self.display_width - self.roi_col:] = 0  # Remove the left and right areas.
    return True, img
```

- 가스양 코드는 PEP-0008 코드 규격에 맞춰서 처음보는 사람도 이해하기 쉽도록 작성하려고 노력했습니다.
- 모든 코드에 구글 스타일로 최대한 가독성 좋게 docstring을 작성했습니다.

2. 개발 매뉴얼 최종본 작성

- 개발 매뉴얼 작성 중이며 오늘(11일 목요일) 자정까지 작성 완료한 뒤 내일(12일 금요일) 보고드리겠습니다.
- 개발 매뉴얼을 개별연구 수강생들 OT 자료로 사용할 수 있도록 상세하게 작성하겠습니다.

---

2. 스티커

1. 프로그램 성능 개선

1. 속도 향상

기존 방법

- 기존에는 모든 라이터 세트에서 헤드를 인식한 뒤 바코드를 인식합니다.
- 헤드 인식에 소요되는 시간은 약 80~90ms입니다.
- 하나의 세트 당 촬영 가능한 최대 사진 개수는 약 10장입니다.

변경

- 라이터 세트의 위치와 젯슨 나노 카메라의 위치가 고정돼있음을 근거로 헤드 인식은 최초 1회만 필요하다 판단했습니다.
- 프로그램 실행 시 조정단계에 진입합니다.
    - 조정단계에서는 최초 1회 Yolo 학습 모델을 로드하고 헤드 위치를 판단합니다.
    - 정확한 위치를 판단할 필요가 있으므로 다소 시간을 소요해서 높은 정확도로 판단합니다.
- 헤드 인식에 소요되는 시간이 사라지므로 촬영 가능한 최대 사진 개수가 증가하며 사진 개수가 많을수록 정확도가 높아집니다.

2. 정확도 향상

- 여러 사진에 대해 단일 결과를 출력하는 데 정확한 근거가 필요했습니다.

기존

- 예를 들어 임의의 라이터 세트에 대해 사진 **5장**을 촬영했을 때
-

	1회	2회	3회	4회	5회
불량여부	**정상**	**정상**	**정상**	**정상**	**정상**

    - 모든 시도에서 라이터 세트가 정상인 경우 해당 라이터 세트는 **정상**을 확실히 판단할 수 있습니다.
-

	1회	2회	3회	4회	5회
불량여부	**불량**	**불량**	**불량**	**불량**	**불량**

    - 모든 시도에서 라이터 세트가 불량인 경우 해당 라이터 세트는 **불량**임을 확실히 판단할 수 있습니다.
-

	1회	2회	3회	4회	5회
불량여부	**정상**	**불량**	**정상**	**불량**	**정상**

    - 하지만 일부 시도에서 라이터 세트가 **불량**이라고 판단된 경우
      단일 결과가 **정상**인지 **불량**인지 판단하기 어렵다는 문제가 있습니다.
    - 따라서 특정 라이터 세트에 대한 판단 결과가 정당함을 보이기 위해서는 **확실한 근거**가 필요합니다.

변경 1

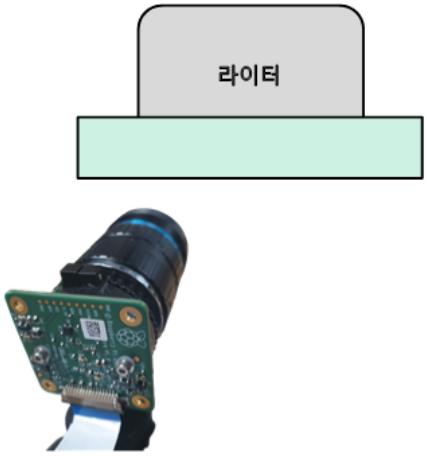
- 탬플릿 이미지 (정상)와 인식한 바코드 사이의 **유사도**를 근거로 불량률을 계산합니다.

-

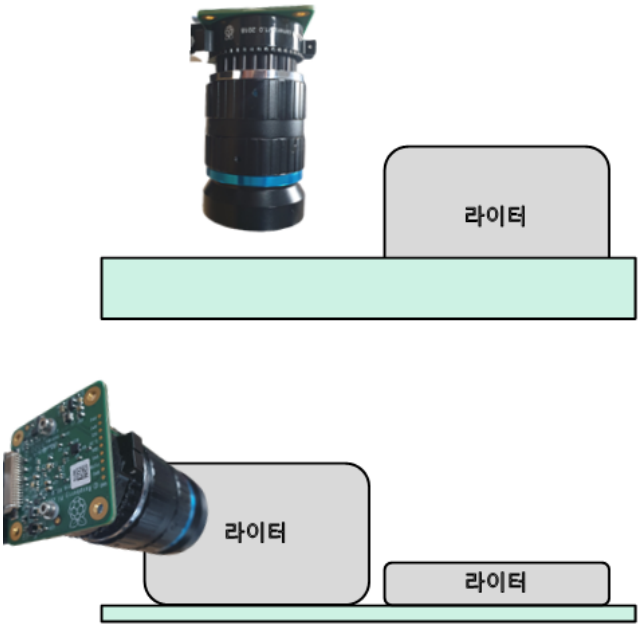
	1회	2회	3회	4회	5회
불량여부	88%	92%	85%	88%	90%

  - 임의의 라이터 세트의 종합 불량률이 $100 - \frac{88+92+85+88+90}{5} = 11.4\%$ 임을 알 수 있습니다.
  - 낮은 불량률이므로 이 라이터 세트는 **정상**이라고 기대할 수 있습니다.

변경 2



- 기존의 스티커 촬영은 위 그림과 같이 이뤄집니다.
- 외부 신호로 작동한다고 하더라도 바코드 스티커의 일부가 가려져서 촬영될 위험이 있습니다.



- 변경된 카메라의 위치에서는 라이터의 일부가 가려질 위험 없이 라이터를 촬영할 수 있어서 정확도가 높아집니다.

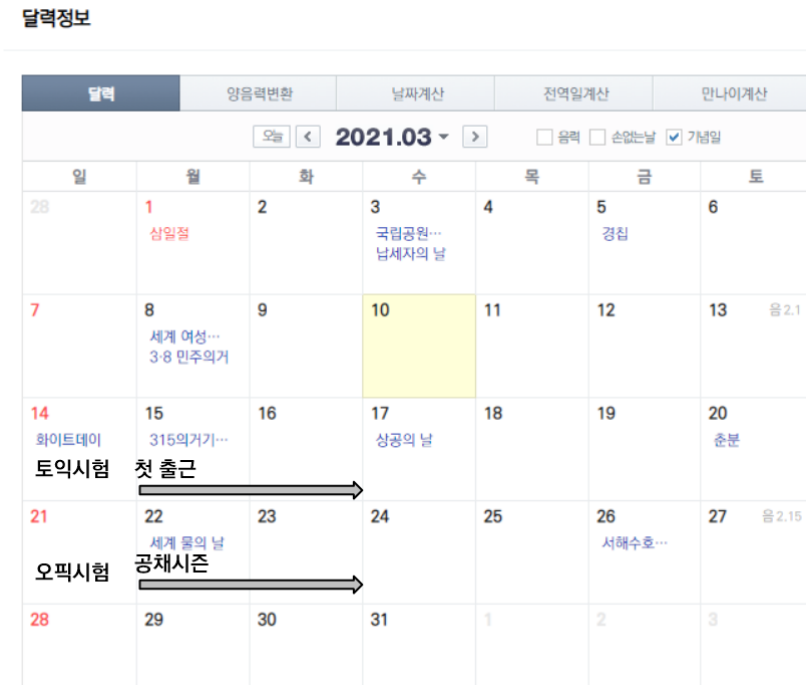
2. 기타

- 스티커 부분의 GUI 도입 여부를 고민하고 있습니다.
- 오늘 (11일 목) 중으로 결정하고 진행하겠습니다.

---

3. 기타

1. 정영준 학생 취업



- 정영준 학생이 3월 15일부로 첫 출근을 하게됐습니다.
- 프로젝트 마무리는 최성준 학생이 맡아서 진행하게 됐습니다.