문자 인식 기술을 이용한 실내용 택배 분류 카트 개발

경희대학교 최지우

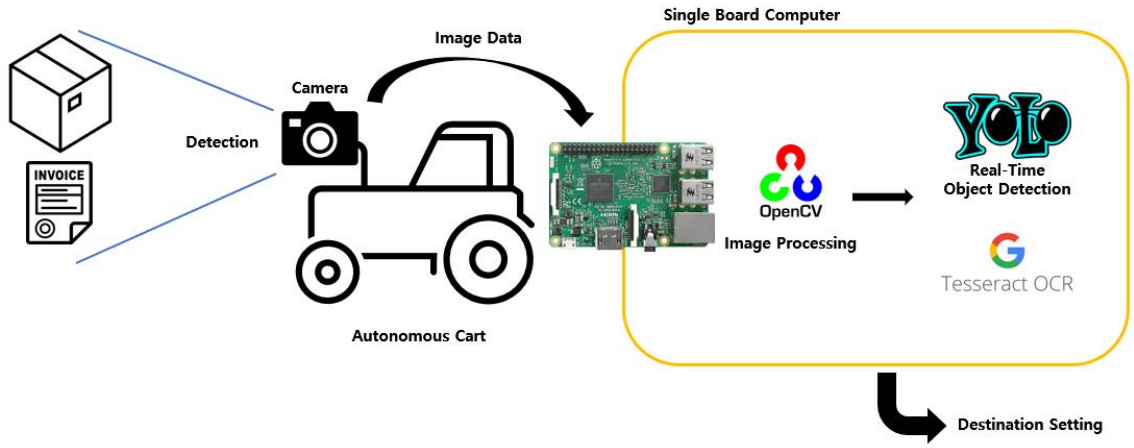
개발 기간	2021.03 ~2021.07
개발 환경	Raspberry Pi, C++, Python, OpenCV, Tiny-YOLO, Colab, Tesseract OCR
개발 인원	2명
기능	1) 택배 상자 객체 인식 2) 운송장 문자 인식 및 특정 데이터 추출 (이름, 주소) 3) 객체와 카트 간 거리 측정 및 카트 이동 / 상자 집기 4) OCR 추출 데이터 기반 자율 주행 목적지 설정

A. Abstract

본 연구는 택배 송장의 글자를 인식하고 필요한 정보를 분류하여 건물 내에서 자율주행 배송작업을 가능하게 한다. 자율주행 카트에 장비된 카메라로부터 촬영되는 실시간 영상에서 Object Detection을 통해 택배와 송장을 감지하며, OCR(Optical Character Recognition, 광학 문자 인식)을 통해 송장의 정보를 인식하여 실내 배송 및 분류에 필요한 정보를 선별한다. 이를 통해 택배 분류 작업 및 건물 내의 배송작업에서의 효율을 높이고, 배송 물량 증가에 따른 택배 종사자들의 부담을 줄일 수 있다.

B. 시스템 설계 및 구현

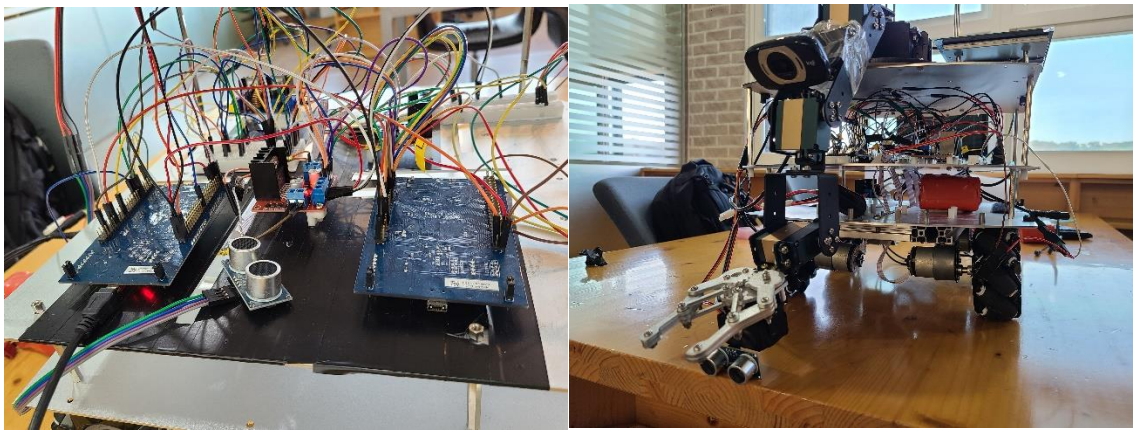
1. System Architecture



자율주행 카트의 동작 개념도이다. 카트 전면에는 카메라와 초음파 센서가 부착되어 있으며, 이는 메인 보드(라즈베리 파이)에 연결된다. 또한 두 개의 STM보드가 메인 보드에 연결되며, 각각 카트의 주행을 위한 모터와 전면부에 장착되는 집게를 작동시키기 위한 알고리즘이 포함된다.

메인 보드에서는 주 알고리즘을 작동시키기 위한 코드가 실행되며, 딥러닝을 통한 Object Detection과 이미지 전처리, OCR 알고리즘이 포함된다. 이미지 신경망 학습에는 Keras 라이브러리와 tiny YOLO를 이용하였으며, OCR을 위한 이미지 전처리에는 OpenCV 라이브러리가 사용되었다.

2. 실내 주행 카트



[사진] 주행 카트의 실제 구현 모습

카트는 Object Detection 및 딥러닝 추론과 OCR을 진행할 메인 코드가 적용된 메인 보드와 바퀴의 모터를 제어하는 보드, 로봇 집게 팔을 제어하는 보드까지 총 3개의 컴퓨팅 보드로 구현되었다. 전면에는 로봇 집게 팔 상단에 카메라가 장착되어 있으며, 실질적으로 물건을 집는 집게에는 초음파 센서를 부착하였다.

3. Object Detection

```
LiveCam = cv2.VideoCapture(0)   # 실시간 영상 입력
YOLO_net = cv2.dnn.readNet('yolov3-tiny_best.weights','yolov3-tiny.cfg')
   # 가중치 및 네트워크 구성 파일 로드
classes = ['box']
layer_names = YOLO_net.getLayerNames()
output_layers = [layer_names[i[0] - 1] for i in
YOLO_net.getUnconnectedOutLayers()]

frame_num = 0
```

```
while LiveCam.isOpened():
    ret, frame = LiveCam.read()   # 실시간 영상 프레임 설정
    if ret is False:
        print("No Video Input")
        break
    if frame_num != 20:
        frame_num += 1
    elif frame_num == 20:   # 입력 프레임 20 번째 마다 작동하도록 설정
        frame_num = 0
         ...
```

실시간으로 촬영되는 영상에서 Object Dection을 진행하기 위한 기본 설정 코드이다. 미리 훈련된 가중치와 네트워크를 구성하는 파일을 OpenCV를 이용하여 로드한다. 영상의 성능 저하 및 이후 코드의 처리가 늦어지는 상황을 대비하여 입력되는 프레임 20회 마다 실행하도록 설정하였다.

```
        h, w, c = frame.shape

        blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0),
True, crop=False)
        YOLO_net.setInput(blob)
        outs = YOLO_net.forward(output_layers)   # Object Detection

        class_ids = []
        confidences = []
        boxes = []

        for out in outs:
            for detection in out:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]

                if confidence > 0.4:   # 인식률이 40% 이상이면 박스로 인식
```

```
                    center_x = int(detection[0] * w)
                    center_y = int(detection[1] * h)
                    dw = int(detection[2] * w)
                    dh = int(detection[3] * h)
                    x = int(center_x - dw / 2)
                    y = int(center_y - dh / 2)
                    boxes.append([x, y, dw, dh])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)

        indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.45, 0.4)
```

입력 프레임을 blob 객체로 변경하여 학습된 모델을 통해 추론하는 부분이다. cv2.dnn_Net.forward 함수를 통해 실질적인 추론이 진행되며, 인식된 객체와 확률을 출력한다. 이 때 인식률이 40% 이상이면 해당 Object를 저장하고 좌표를 저장하도록 설정하였다.

```
if confidences:
            bestscore = confidences.index(max(confidences))
            best_x, best_y, best_w, best_h = boxes[bestscore]

            if best_x > 320 :   # 카메라의 중앙보다 오른쪽에 위치
                ser.write(serial.to_bytes([int('1',16)]))   # 오른쪽 이동

            elif best_x + best_w < 320 :   # 카메라의 중앙보다 왼쪽에 위치
                ser.write(serial.to_bytes([int('2',16)]))   # 왼쪽 이동

            else :   # 카메라의 중앙에 위치
                cv2.imwrite('cap_img.jpg', frame)   # 해당 프레임 저장
                ser.write(serial.to_bytes([int('3',16)]))   # 주행
                break

            cv2.rectangle(frame, (best_x, best_y), (best_x + best_w, best_y
+ best_h), (0, 0, 255), 5)
```

인식한 객체의 좌표를 기준으로 카트의 위치를 이동시키기 위한 통신을 지원하는 부분이다. 감지한 상자들 중에서 가장 큰 인식률을 갖는 객체의 좌표를 저장한 후, 카메라의 중앙점이 인식한 좌표 박스의 범위 내에 있도록 카트를 주행한다. 이 때 카트의 주행 모터를 담당하는 STM보드에 Serial Communication을 통해 움직여야 할 방향을 전달한다. 카트와 인식한 객체의 좌표가 일직선 상에 위치하면, 당시의 프레임을 저장하여 OCR을 진행하게 된다.

4. Image Processing

이전 단계에서 저장된 운송장 이미지를 템플릿에 맞추어 정렬하기 위한 전처리 작업을 나타내는 코드이다.

```
def rotate_image(image, template, maxFeatures=500, keepPercent=0.2,
debug=False):
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray_template = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

    orb = cv2.ORB_create(maxFeatures)
    (kpsA, descsA) = orb.detectAndCompute(gray_image, None)
    (kpsB, descsB) = orb.detectAndCompute(gray_template, None)

    method = cv2.DESCRIPTOR_MATCHER_BRUTEFORCE_HAMMING
    matcher = cv2.DescriptorMatcher_create(method)
    matches = matcher.match(descsA, descsB, None)
```

이미지의 시야각을 회전시켜 템플릿과 일치하도록 수정하는 함수이다. Object Detection 단계에서 촬영된 이미지와 기존에 저장된 템플릿을 입력으로 받아 처리한다.

먼저 이미지와 템플릿에 대하여 ORB 알고리즘을 통해 특징점을 검출한다. ORB 알고리즘은 Oriented FAST and Rotated BRIEF의 약자로, FAST 알고리즘과 BRIEF 알고리즘, 해리스 코너 알고리즘을 결합한 알고리즘이다.

이후 이미지와 템플릿의 특징점과 특징 디스크립터를 저장한 후 비교하여 서로 비슷한 특징을 가진 객체를 매칭시키는 과정이 진행된다. 매칭 과정은 BruteForce 방법을 이용하여 디스크립터를 하나씩 모두 검사한 후 가장 가까운 디스크립터를 찾고 가장 좋은 매칭을 반환하는 방식으로 진행된다.

```
    ptsA = np.zeros((len(matches), 2), dtype=float)
    ptsB = np.zeros((len(matches), 2), dtype=float)
```

```
    for (i, m) in enumerate(matches):
        ptsA[i] = kpsA[m.queryIdx].pt
        ptsB[i] = kpsB[m.trainIdx].pt

    (H, mask) = cv2.findHomography(ptsA, ptsB, method=cv2.RANSAC)
```

이미지의 특징점을 구해 매칭이 완료되면, cv2.findHomography() 함수를 사용하여 호모그래피를 찾아낼 수 있다. 이 때 RANSAC 알고리즘을 이용하여 노이즈를 최소화시키고 투시 변환의 성능을 향상시킨다.

```
aligned = cv2.warpPerspective(image, H, (w, h))

return aligned
```

최종적으로 촬영된 운송장 이미지는 호모그래피와 cv2.warpPerspective 함수를 통해 정렬되어 반환된다.

5. OCR

ocr 함수는 자율주행 카트와 인식한 물체가 일직선상에 위치하였을 때 촬영된 운송장 이미지와, 운송장 템플릿 이미지를 전달받는다.

```
def ocr(image, template) :
    print("[Loading...] OCR Location Setting")

    OCRLocation = namedtuple("OCRLocation", ["id", "bbox",
"filter_keywords"])
    # 문자 판독을 진행할 운송장 템플릿의 박스 좌표
    OCR_Locations = [
        OCRLocation("name", (27, 96, 60, 20), []),
        OCRLocation("address", (27, 115, 276, 21), []),
        OCRLocation("detail_address", (28, 134, 409, 36), []),
    ]

    print("[Loading...] aligning images")
```

```
    aligned = rotate_image.rotate_image(image, template)
```

ocr 함수에서는 먼저 운송장 템플릿에서 문자 판독을 진행할 좌표를 설정하여야 한다. 이 작업은 수동으로 진행되며, 택배사의 송장 템플릿에 따라 차이가 발생할 수 있다. 본 연구에서는 CJ사의 운송장 템플릿이 사용되었다. 배달을 궁극적인 목적으로 두기 때문에, 물품 이름이나 송신자의 정보는 제외하고 최소한의 정보인 수취인 이름과 상세주소만을 추출하도록 설정하였다.

이전 단계에서 서술한 이미지를 정렬시키는 rotate_image 함수는 ocr 함수 내에서 호출되며, 정렬한 이미지는 따로 저장된다.

```
for loc in OCR_Locations:
        (x, y, w, h) = loc.bbox
        roi = aligned[y:y+h, x:x+w]
        cv2.imshow(loc.id, roi)
        cv2.waitKey(0)

        rgb = cv2.cvtColor(roi, cv2.COLOR_BGR2RGB)
        text = pytesseract.image_to_string(rgb, lang='kor')

        for line in text.split("\n"):
            if len(line) == 0:
                continue

            lower = line.lower()
            count = sum([lower.count(x) for x in loc.filter_keywords])

            if count == 0:
                parsingResults.append((loc, line))
```

설정된 각 좌표에 따라 이미지의 문자를 텍스트 데이터로 변환하여 저장하는 부분이다. Tesseract 엔진 및 pytesseract 라이브러리가 사용되었으며, 한글과 숫자만을 인식하도록 설정되었다.

```
    results = {}
```

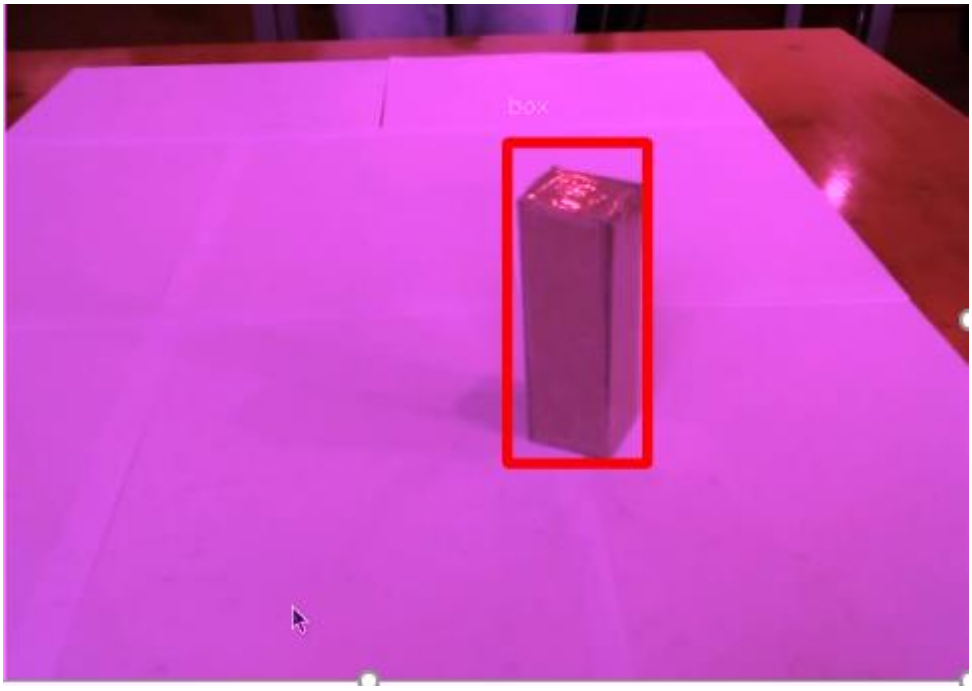
```
    for (loc, line) in parsingResults:
        r = results.get(loc.id, None)

        if r is None:
            results[loc.id] = (line, loc._asdict())

        else:
            (existingText, loc) = r
            text = "{}\n{}".format(existingText, line)

            results[loc["id"]] = (text, loc)
    ...
    return results
```

텍스트 데이터로 변환이 완료되면, 정보를 추출하기 쉽도록 가공되어 반환된다. 반환된 정보는 데이터베이스와 비교하여 최종 목적지를 설정하기 위해 사용된다.

C. 결론 및 기대효과

1. 성능 평가

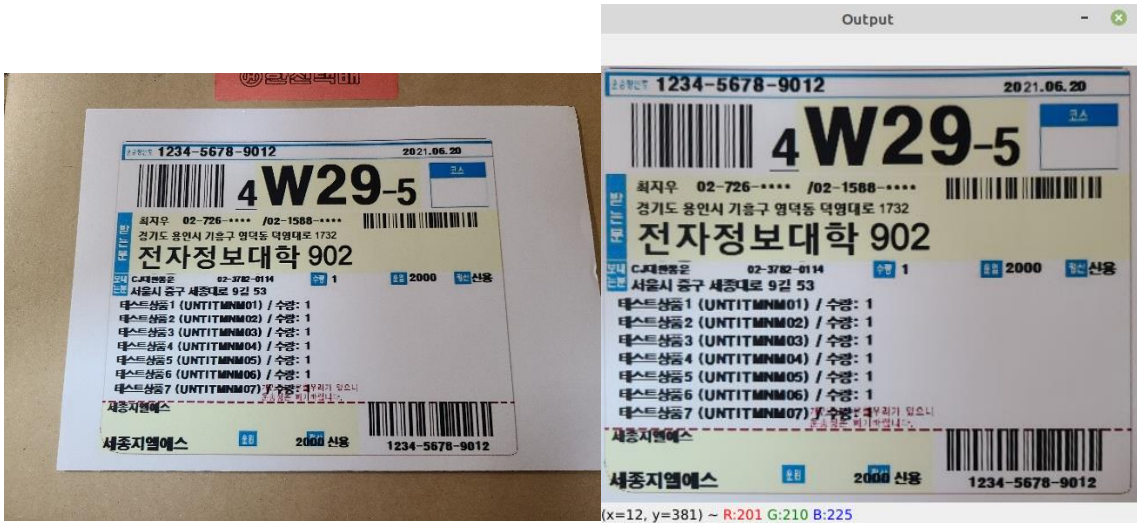
1) Object Detection



[사진] 이미지 내 객체 추론

학습한 모델에 대해서 대부분의 실시간 프레임에서 객체를 검출할 수 있다. 추론은 초당 약 3~4 프레임 정도에서 작동하였으며, 이는 최종적으로 전체 코드의 병목현상을 막기 위해 초당 1프레임마다 객체 추론을 작동하는 것으로 설정하였다. 이에 따라 시리얼 번호를 송신하는 것도 초당 1회 이루어지게 되며, 카트와 카메라가 일직선 상에 위치하도록 조금씩 이동하게 된다.

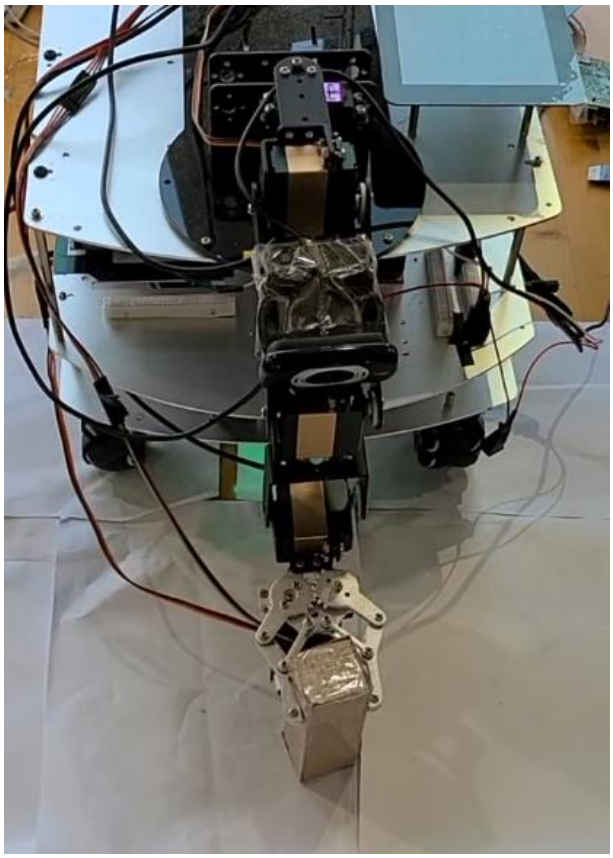
2) 이미지 전처리 및 OCR



[사진] 이미지 전처리 전/후

이미지 전처리과정을 통하여 템플릿과 동일한 크기와 위치를 가지도록 이미지를 정렬시킬 수 있다. 전처리과정 이전의 이미지에서는 OCR이 거의 작동하지 않았지만, 미리 학습된 좌표와 정렬된 이미지를 이용하여 진행하였을 때는 약간의 한글 인식 오류가 존재하지만 90% 이상의 인식률을 보였으며, 데이터베이스에 저장된 이름과 상세주소를 비교하여 최종 목적지를 설정할 수 있을 정도의 정확도를 보인다.

3) 자율 주행 카트



[사진] 감지한 물체를 집는 로봇

Object Detection 과정이 완료되어 직진 신호가 전달되면, 카트는 집게 팔 하단에 장착된 초음파 센서의 감지 거리가 일정 거리 내에 들어올 때까지 직진하여 객체를 잡는다. 이 때 카트의 무게, 무게 중심에 따라 출발 시 방향이 틀어지거나, 제동거리가 늘어나 물체를 밀치는 경우가 발생하기도 한다. 이는 추후 가감속 제어를 통하여 보다 정교한 주행을 할 수 있도록 설정하는 방안을 검토해볼 수 있다.

2. 기대 효과

이러한 실내 자율 주행 카트 및 송장 감지 모델의 개발로 배달원은 건물 내부의 여러 곳을 돌아다니지 않고 지정 지점에 물건을 두면, 자율 주행 카트가 이를 인식하고 판별하여 건물 내 최종 목적지에 배달할 수 있게 한다. 이를 통해 택배 분류 작업 및 건물 내의 배송작업에서의 효율을 높이고, 배송 물량 증가에 따른 택배 종사자들의 부담을 줄일 수 있다.

3. 추후 연구 방향

본 연구에서는 OCR을 진행할 범위를 미리 수동으로 설정한 템플릿을 이용하여 진행하였으나, 이는 향후에 모든 텍스트를 인식하고 정규 표현식 패턴을 기반으로 필드를 자동으로 설계하는 시스템으로 발전시킬 수 있다. 그러한 시스템은 OCR을 진행할 범위를

미리 지정하지 않아도 다양한 종류의 운송장을 인식하여 데이터를 추출할 수 있게 한다. 이 경우 운송장의 전체 이미지에서 OCR을 진행해야 하므로, 추가적인 이미지 전처리가 필수적이다. 예를 들어 운송장에 포함된 불필요한 선들을 제거하기 위해 Prewitt Mask 혹은 Sobel Mask의 사용을 고려할 수 있다. 이러한 추가적인 작업을 통해 OCR의 인식률을 개선할 수 있을 것으로 예상된다.

D. 참고 문헌

[1] Tesseract : https://en.wikipedia.org/wiki/Tesseract_(software)

[2] Naver Cloud Platform Clova Ai : https://clova.ai/ocr

[3] https://www.hankookilbo.com/News/Read/201910041485733553

[4] Amazon Textract : https://aws.amazon.com/ko/textract/

[5] Prof. Suneel K Nagavi, Mahesh S Gothe, Praveen .S. Totiger, "Optical Character Recognition based Auto Navigation of Robot by Reading Signboard", 2015.

[6] Dominic L, Francoise M,JM Valin, Catherine P, "Making a mobile robot read textual Messages", 2003.