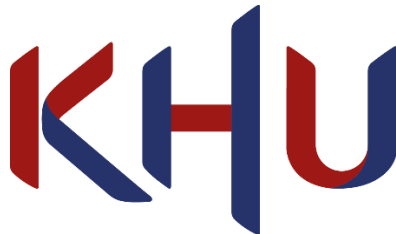


2020 년 1 학기 캡스톤 디자인 1 최종 결과 보고서

실내 문서 전달 자율주행 카트 개발을 위한  
객체 인지 및 충돌 방지 모듈 개발

An Implementation of Object Recognition and  
Collision Avoidance Modules for Indoor Self-driving Cart



컴퓨터 공학과 이재빈 김명현

지도교수: 허의남

# INDEX

|   |  |
|---|--|
| <b>1. 서론</b>  |  |
| 1.1 연구배경  |  |
| 1.2 연구 목표   |  |
| <b>2. 관련 연구</b>   |  |
| 2.1. SLAM(Simultaneous Localization And Mapping) 기반 자율주행 카트 |  |
| 2.2. 클라우드 연계 자율주행 맵 시스템 기술 동향                               |  |
| 2.3. 위치 추정, 충돌 회피, 동작 계획이 융합된 이동 로봇의 자율 주행 기술               |  |
| 2.4. SVHN 숫자 인식 연구(Multi-digit Number Recognition)          |  |
| 2.5. 관련 연구 분석   |  |
| <b>3. 프로젝트 내용</b>   |  |
| 3.1. 주행 시나리오 및 모듈 별 기능                                      |  |
| 3.1.1. 구현 환경  |  |
| 3.1.2. 시나리오   |  |
| 3.1.2.1. 명패 인식 (Door Plate Number Detection) 시나리오           |  |
| 3.1.2.2. 사람 인지 (Human Object Recognition) 시나리오              |  |
| 3.2. 요구사항 및 해결 방안   |  |
| 3.2.1. 인식 시간  |  |
| 3.2.2. 객체 인지  |  |
| 3.2.3. 명패 (Door Plate Number Tag) 정보 인식                     |  |
| 3.2.4. 명패 번호 (Door Plate Number) 인식용 데이터셋 활용                |  |
| 3.3. 시스템 설계   |  |
| 3.3.1. UML Diagram 을 통한 시스템 모델링                             |  |
| 3.3.1.1. Activity Diagram                                   |  |
| 3.3.1.2. Sequence Diagram                                   |  |
| 3.4 구현 과정   |  |

|   |  |
|---|--|
| 3.4.1. 개발 환경 .....  |  |
| 3.4.2. Door Plate Crop 모듈 개발 .....                                      |  |
| 3.4.3. Noise Decision Model 구축 .....                                    |  |
| 3.4.4. Number Recognition Model 구축 .....                                |  |
| 3.4.5. Door Plate Recognition 모듈 통합 .....                               |  |
| 3.4.6. Human Recognition 모듈 개발 .....                                    |  |
| 4. 프로젝트 성능 개선 .....   |  |
| 4.1. 명패 크롭 이슈[Door Plate Crop Issue].....                               |  |
| 4.2. 모델 예측도 이슈[Model Accuracy Issue - Number Recognition] .....         |  |
| 4.3. 크롭-예측 병목현상 이슈[Bottle Neck Issue Between Crop and Prediction] ..... |  |
| 5. 결론 및 향후 연구 .....   |  |
| 6. 참고 문헌 .....  |  |

## 초 록

최근 머신 러닝 기술이 자동차에 접목 되면서 '자율주행'에 관한 연구가 활발히 이루어지고 있다. 다만 실내의 경우, 환경적 제약조건에 따라 자율주행차의 원활한 주행이 어려운 실정이다. 따라서 본 보고서는 실내문서전달용 자율주행 카트의 원활한 주행을 위해 기계학습 기반의 실시간 객체 인지 및 충돌 방지 모듈 개발을 목표로 한다. 이후, 최종적으로 이를 자율주행차에 적용하여 수집된 실내 정보를 분석한 후 정확한 실내 정보 인지 및 안정적인 실내 주행에 기여한다.

## 1. 서론

### 1.1. 연구배경

5G 이동통신의 도래 및 머신 러닝과 같은 SW 기술발전으로 자율 주행 기술의 점진적 발전이 예상된다. 미국 자동차 학회(SAE)는 자율 주행의 구현 정도에 따라 자율주행 기준 발전단계를 마련하였고, 현대자동차는 2022 년 까지 주행 관련 정보를 판단하고 행동을 결정할 수 있는 레벨 3 단계를 대부분 구축할 것으로 전망하고 있다. 이처럼, 실외 용 Autonomous Vehicle 의 연구가 현재 활발히 연구 개발되고 있으나, 도로 위가 아닌 실내의 경우 또 다른 문제점에 직면하게 된다. 먼저 Guide Line 인 차선이 없다는 점과 더불어, 비좁은 통로가 존재하고 사람들의 왕래가 실내에서는 더욱 잦다는 점을 감안하면 단순히 위치정보만을 활용한 자율주행카트 구현에는 무리가 있다. 따라서 본 연구는 기존 실외 자율주행카트의 개념에 덧붙여 실시간 영상처리, 머신 러닝을 통한 Object Recognition 과 Collision Avoidance Module 을 구현하고 정확한 실내 정보를 인지하여 안정적인 실내 자율주행 카트 개발을 목표로한다.

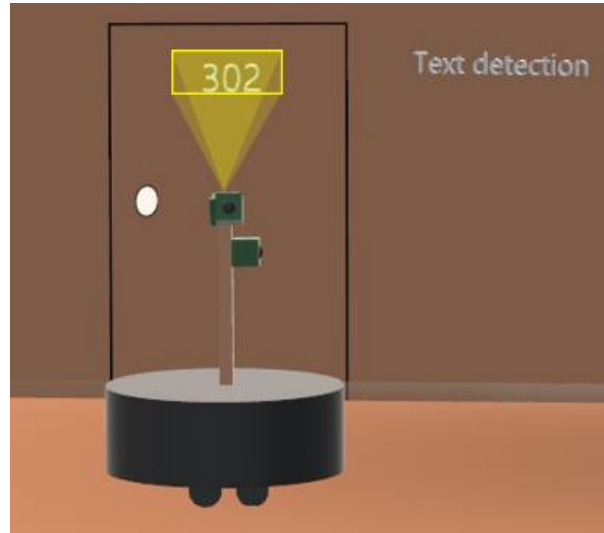
### 1.2. 연구목표

실내 자율주행 카트의 위치정보는 클라우드에 저장되고, 이와 동시에 전면, 측면에 대한 실시간 영상 처리를 통해 보다 정확한 위치정보를 얻을 수 있도록 자율 주행 카트 모듈을 구현한다. 이에 덧붙여, 실내에서의 다양한 텍스트(실내의 각 태그 혹은 문패)의 특징점을 기반으로 위치정보를 특정 한다. 또한, 영상처리 모듈을 통해 실시간 객체(사람) 인식 기능을 구현하여 Human Oriented Driving System 을 구현한다. 구현 목표에 대한 각 모듈별 동작 개념도와 설명은 다음과 같다.

- 명패 인식[Text Recognition]모듈

- 명패인식모듈은 카트의 양 측면에 부착된 카메라를 통해 카트의 정확한 위치를 파악하는 기능을 담당한다. 실내의 특성상, GPS 사용이 어렵다는점, 다양한 전파들의 간섭이 잦은만큼 클라우드에의해 구체적인 위치를 추정하기가 힘들다는 점을 고려하여, 양측면 카메라를 사용하여 카트의 구체적인 위치를 파악하도록 구현하였다.

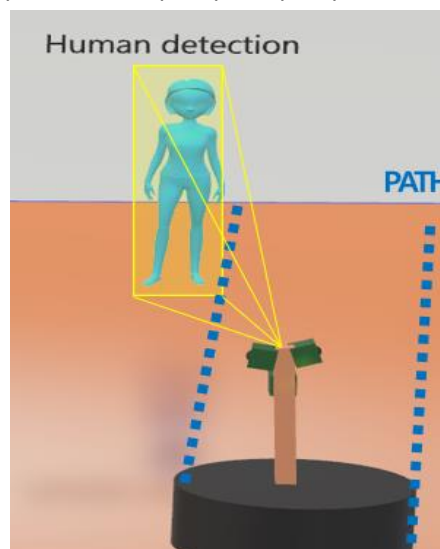
- 구체적인 동작 프로세스는 다음과 같다.
1. 카트의 양측면에 달린 카메라를 통해 명패(Door Plate)를 추출한다.
  2. 사전 학습된 머신러닝 모델을 통해 현재 카트의 위치가 어디인지를 예측한다.
  3. 예측된 값을 바탕으로 제어모듈에 주행정보를 전송한다.



[그림 1 명패인식모듈 개념도]

- 사람 인지[Human Recognition]모듈

- 사람인지모듈은 카트 전방에 부착된 카메라를 통해 사람객체를 인식할 수 있도록 구현된다. 또한 사람을 회피한 예상 경로를 출력하고, 이를 1 차원 형태의 배열로 변환, 제어모듈로 주행정보를 전송한다.
  - 구체적인 동작 프로세스는 다음과 같다.
1. 카트 전방의 카메라를 통해 사람을 인지한다.  
(이때 감지 최소범위는 5m 이상으로, 5m 이내의 사람, 장애물은 LIDAR 센서를 활용하여 탐지함)
  2. 카트 정면기준 화면을 10 등분 하여, 현재 장애물의 위치를 포함한 주행정보를 반환한다.
  3. 장애물을 회피하는 예상 주행경로를 화면에 출력한다.



[그림 2 사람 인지 모듈 개념도]

## 2. 관련 연구

### 2.1. SLAM(Simultaneous Localization And Mapping) 기반 자율주행 카트

초음파, 적외선, 관성 측정 장치인 IMU와 LIDAR, 모터 등을 사용하여 제어 가능한 로봇에 대해 연구가 이루어진 바 있다. 이 원격제어 로봇은 LIDAR 센서를 통해 물체와의 거리를 측정하여 장애물 회피 기능을 구현하였다. 다만 <sup>1</sup>LIDAR 센서가 Object의 반사 펄스를 이용하여 물체의 거리를 측정하는 특성을 고려하면, 여전히 반사율이 높은 환경에서 정확한 거리 인식이 어렵고, 이에 따른 우발적 장애 요소에 대한 회피문제점이 잔존한다.

### 2.2. 클라우드 연계 자율주행 맵 시스템 기술 동향

클라우드와 연계하여 생성되는 자율주행 맵을 구성하면 보다 정밀성 있는 맵을 구성할 수 있다. 또한 도로 교통과 관련된 실시간 갱신 주기 측면에서 더욱 세밀하고 정확한 위치정보를 제공할 수 있다. 그러나 실내에서의 경우 주행 경로 상의 장애물이 있을 가능성이 높고, 좁은 통로의 특성상 많은 우발적 요소가 예상되기 때문에 정적인 지도만을 구축하여 주행하기에는 한계점이 있다.

### 2.3. 위치 추정, 충돌 회피, 동작 계획이 융합된 이동 로봇의 자율 주행 기술

실내 이동로봇의 자율주행 방법으로 지도 생성, 위치 추정, 장애물 회피, 경로 계획에 대한 연구가 이루어진 바 있다. 위치 추정을 위해서 지도 정보를 이용, 센서 데이터를 계산하였고 인공지능을 사용하여 장애물로부터의 척력, 목표위치로의 인력을 구하여 <sup>2</sup>회피 알고리즘을 구현하였다. 그러나 <sup>3</sup>LRF 센서를 바탕으로 장애물을 인식하였고 때문에 실내의 좁은 통로 특성상 센서 길이의 제약으로 인해 장애물 감지 거리가 55cm에 불과하였다.

### 2.4. SVHN 숫자 인식 연구(Multi-digit Number Recognition)

CNN을 이용해 이미지에서 여러 자리의 숫자를 감지하는 <sup>4</sup>연구가 존재하며, 해당 연구에서 사용된 SVHN(Street View House Numbers) <sup>5</sup>데이터셋을 훈련 용도로 사용하고자 한다.

---

<sup>1</sup> 레이저 펄스의 반사 시점까지의 시간을 측정하여 반사체의 위치와 표를 측정하는 센서.

<sup>2</sup> 해당 연구에서는 이동로봇의 위치추정방법으로 MCL알고리즘, 삼각측량, 칼만 필터 등을 사용하였음

<sup>3</sup> LIDAR 센서와 동일한 원리로, 레이저펄스를 이용하여 반사체 간의 거리를 측정하는 센서.

<sup>4</sup> 참고문헌[5] 참조

<sup>5</sup> <http://ufldl.stanford.edu/housenumbers>

## 2.5. 관련 연구 분석

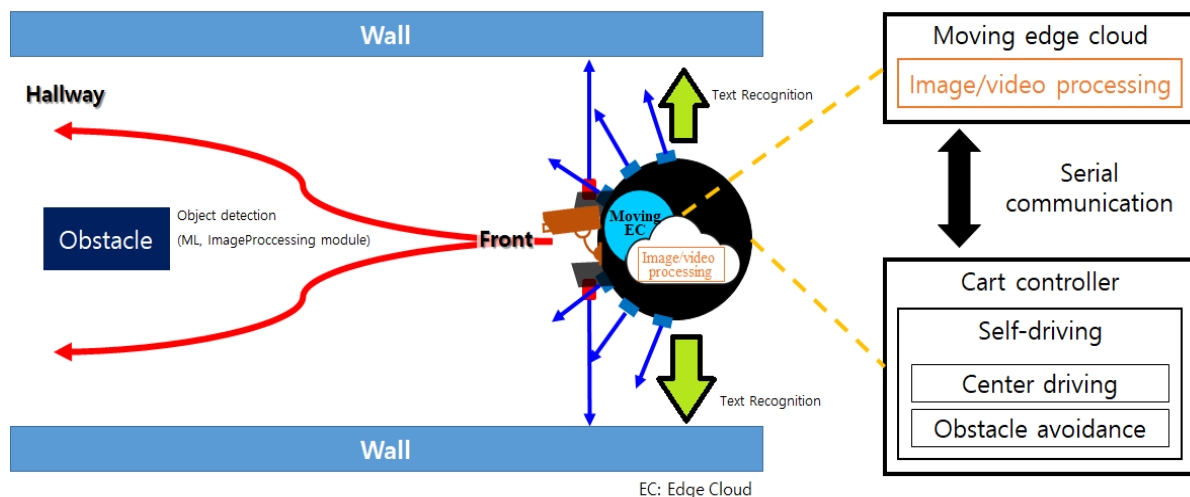
앞서 보듯, 관련 기존 연구는 정적인 맵을 구축하거나, 센서에만 의존한 주행으로 안정성의 한계가 있었다. 따라서 본 연구는 LIDAR 센서를 통한 근거리 감지와 동시에 실시간 영상처리를 통한 원거리 이동 객체(사람) 인지, 실내 특징점(Door Plate Number)인식을 바탕으로 실내 환경에 특화된 보다 효율적인 자율 주행카트를 제시한다.

| 구분  | 연구내용  | 한계점  |
|---|---|--|
| SLAM기반<br>자율주행 카트                                       | IMU, LIDAR, 모터를<br>사용하여 자율주행<br>구현                            | 반사펄스를 이용하는 LIDAR,<br>초음파센서의 특성상,<br>반사율이 높은 환경에서<br>우발적 상황에 대한 한계점<br>잔존                         |
| 클라우드 연계<br>자율주행 맵<br>시스템<br>기술동향                        | 클라우드 기반 지도<br>구축,<br>엣지클라우드 -<br>카트 간<br>상호작용을 통한<br>자율 주행 구현 | 실내의 경우 주행 경로 상에<br>장애물이 있을 가능성과 좁은<br>도로 폭을 고려해 볼 때<br>정적인 지도만을 바탕으로 한<br>자율주행에는 많은 어려움이<br>존재함. |
| 위치 추정,<br>충돌 회피,<br>동작 계획이<br>융합된 이동<br>로봇의 자율<br>주행 기술 | LRF 센서를 바탕으로<br>한 장애물 인식,<br>인공전위계를<br>사용하여<br>회피알고리즘 구현      | 실내의 좁은 통로 특성상<br>LRF센서의 길이가 짧아짐에<br>따라 장애물 감지거리가 약<br>55cm에 불과하여 실질적인<br>자율 주행에는 어려움이 있음.        |





[표 1] 기존 연구 분석

## 3. 프로젝트 내용

### 3.1. 주행 시나리오 및 모듈 별 기능



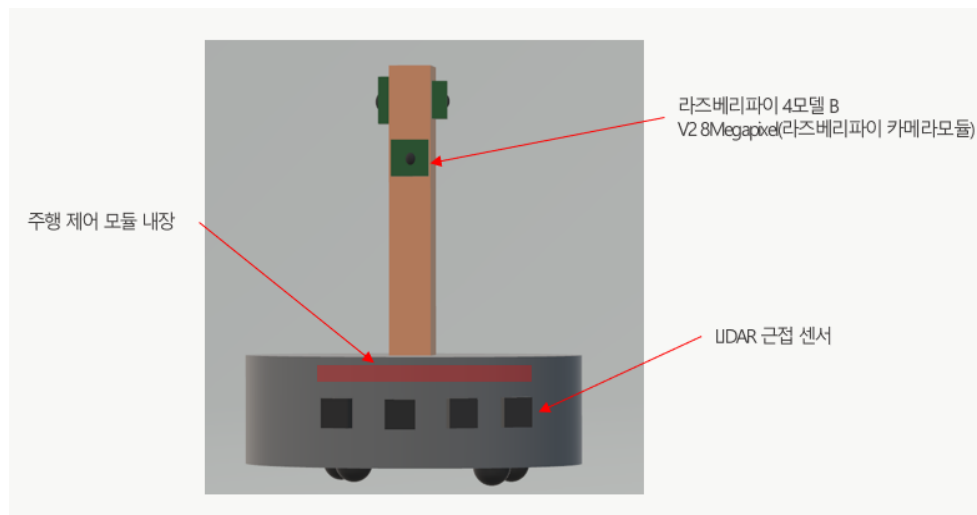
[그림 3] 주행 개념도 (위)

| 구분  | 상세 내용                     | 구분  | 상세 내용                           |
|---|---------------------------|---|---------------------------------|
|  | 추후 카트 예상 이동<br>경로(장애물 회피) |  | 카트 양 측면 Door<br>Plate의 Number인식 |
|  | LIDAR센서를 통한<br>근거리 객체 감지  |  | 카트 전면 부 이동<br>객체 (사람) 인지        |

[표 2] 주행 개념도 기호

### 1) 기기 구성

자율주행 카트에 LIDAR 센서를 전방, 양 측면에 부착하여 근거리에 대한 물체감지가 이루어진다. 또한, 카트의 위치 정보는 각 클라우드의 엣지에 의해 세밀화 된다. 다만 클라우드의 엣지만으로는 카트의 실내 위치를 정확히 추론할 수 없음과 LIDAR 센서의 감지 거리 한계를 고려하여, 다음 두 모듈을 통해 실내의 위치를 정확하게 추론하고 원거리의 물체를 회피할 수 있도록 한다.



[그림 4] 기기 구성도

### 2) 영상 처리 모듈

영상처리 모듈은 카트에 부착된 전면카메라와 양 측면 카메라를 활용한다. Human Recognition 모듈은 전면 카메라의 실시간 영상으로부터 객체를 회피한 예상주행경로를 제어 모듈로 전송한다. 다음으로 양 측면 카메라는 실내의 정보 파악을 위한 Door Plate Number 추출 기능을 위해 동작한다. Number Tag Detection 모듈에 의해 영상의 각 프레임마다 실내의 Door Plate 에 대해 Number Tag 를 크롭하며 이후 크롭된 이미지의 해석을 위해 동일 라즈베리 파이에 로드된 머신 러닝 모델로 해당 이미지를 반환한다.

### 3) 머신 러닝 모듈


실내 환경에 특화된 모델에 의해 크롭 이미지의 노이즈를 필터링한다. 또한 정확한 Number Tag 값을 해석하여 보다 정확한 실내의 위치를 추론한다. 최종적으로, 주행에 관한 정보가 도출되고 해당 정보를 카트의 제어 모듈로 전송, 이를 기반으로 안정적인 실내 주행을 구현한다.



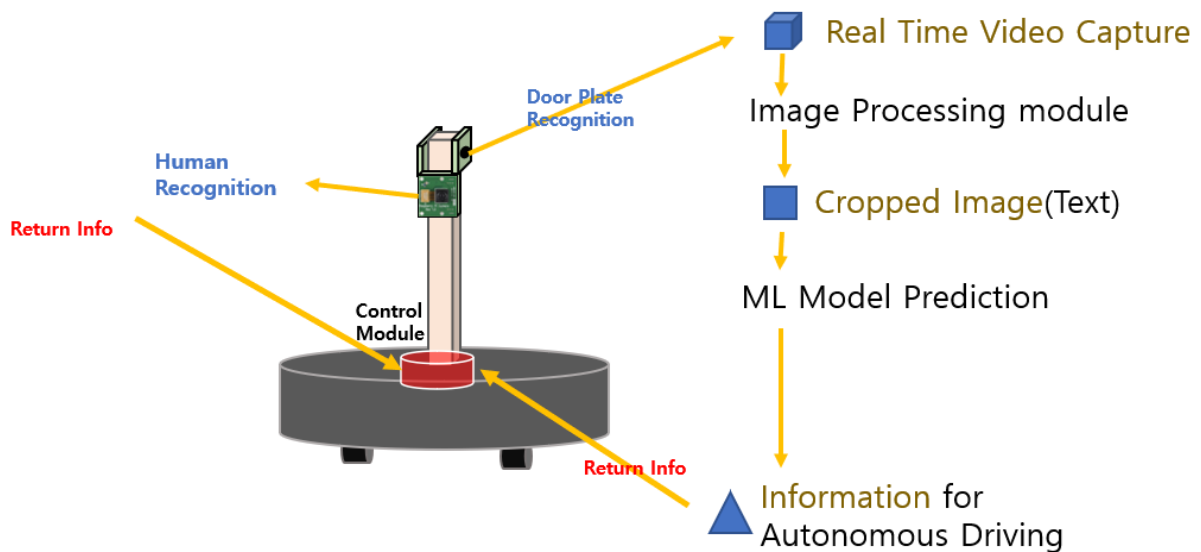
### 3.1.1. 구현 환경

Object Recognition and Collision Avoidance 모듈은 라즈베리 파이 4B 모델에 별도 카메라 모듈을 부착하여 구성 되었으며 라즈베리 파이의 하드웨어 사양은 다음과 같다. 또한, 카트의 동작 범위는 좁은 복도가 있고, 명패(Door Plate)가 있는 실내 환경으로 한정한다.

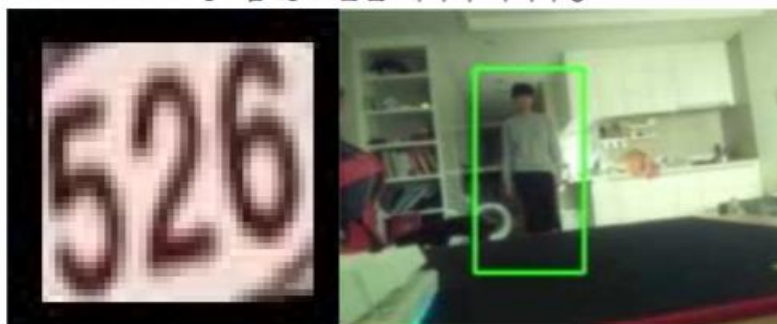
[표 3] 주요 모듈 개발 환경

| 구분  | 구현 환경   |
|---|---|
|  | [라즈베리파이 4B모델]<br>브로드컴 BCM2711, 쿼드코어<br>Cortex-A72 (ARM v8)<br>64-bit SoC @ 1.5GHz<br>4GB LPDDR4-3200 SDRAM<br>카메라모듈 V2 8Megapixel |

### 3.1.2. 시나리오



[그림 5] 모듈 간 데이터 처리 과정



[그림 6, 7] 실내 명패 크롭 예시 (좌) 및 사람 인식 (우)

### 3.1.2.1. 명패 인식 (Door Plate Number Detection) 시나리오

양 측면 카메라에서 전달받은 실시간 RGB 영상에서 명패에 해당하는 이미지를 48\*48 사이즈로 크롭한다. 이때, 명패 크롭의 잡음을 줄이기 위해, Canny Edge 검출법을 사용하여 윤곽선을 검출한다. 또한, 명패의 일반적인 위치를 고려하여 크롭이미지의 좌표값을 통해 보다 면밀히 크롭할 수 있도록 한다. 올바르게 크롭된 이미지는 Grayscale 로 변환되어 잡음인지 정확한 명패인지 판단하는 CNN 모델에 입력된 후, 올바른 명패이미지라고 판단될 경우, 해당 이미지는 Number Detection 모델에 의해 숫자값을 예측한다. 이 과정을 통해 총 6 개의 결과 값을 도출한다. 이 때 6 개의 결과 값은 이미지가 숫자를 포함하는지 여부, 인식된 숫자의 길이, 최대 4 자리의 숫자 값으로, 이 결과 값을 통해 실내 정보 데이터베이스와 비교하여 현재 카트의 정확한 실내 위치를 카트의 제어 모듈로 전송한다.

### 3.1.2.2. 사람 인지 (Human Object Recognition) 시나리오

전방 카메라에서 받은 실시간 영상으로부터 OpenCV 의 Hog 디스크립터를 이용하여 사람 객체를 추출한다. 이를 바탕으로 사전에 학습된 SVM 모델을 이용하여 대상 객체가 사람인지에 대한 예측값을 반환한다. 이후 카트의 예상 동선을 10 개요소를 가진 1 차원배열로 작성하고 카트의 너비를 고려, 사람을 회피한 가장 이상적인 경로 정보를 1 차원 배열 형태로 제어 모듈에게 전송한다.

## 3.2. 요구사항 및 해결 방안

### 3.2.1. 인식 시간

현재 pytesseract 의 OCR 라이브러리는 720\*960 이미지 파일 기준, Text Extraction 의 지연시간은 평균 0.38 초로 측정되었다. 라즈베리 파이의 경우 CPU, GPU 환경이 좋지 않아 많은 Latency 가 예상되므로 기존 OCR 라이브러리 보다는 학습된 머신 러닝 모델을 통하여 예측하는 것이 바람직할 것으로 보인다. 7)현재 라즈베리 파이 환경에서의 실시간 Human Detection 실행시간은 약 0.05 - 0.06 초로 실시간 Object 인식에는 무리가 없다. 다만 Number Tag 추출의 경우 더 높은 해상도와 머신 러닝 모델의 지원이 필요한 부분이므로 학습 모델의 경량화와 프레임 조절 및 정확도 높은 크롭(Crop)이 요구된다.

### 3.2.2. 객체 인지

Object Recognition 의 경우 가장먼저 Human Oriented 로 구현되어야 하며, 사람 중심으로 철저히 구동되어야 한다. 최소 5m 에서 최대 25m 거리에서 사람을 인식할 수 있어야 하며, 우발적인 동작에도 빠르게 반응할 수 있어야 한다.

### 3.2.3. 명패 (Door Plate Number Tag) 정보 인식

Number Detection 의 경우 라즈베리 파이의 카메라모듈 프레임이 초당 20-30 프레임을 감안하여 최소 4-5%이상의 이미지 크롭 정확도를 요구한다. 하지만 이는 최소 범위이므로, 원활한 주행을 위해서는 약 40%이상의 이미지 크롭 정확도를 요구한다. 머신 러닝 모델의 경우 약 80% 이상의 정확도를 유지하여, 실내 환경에서의 Text 를 예측하고 잡음을 걸러낼 수 있어야한다.

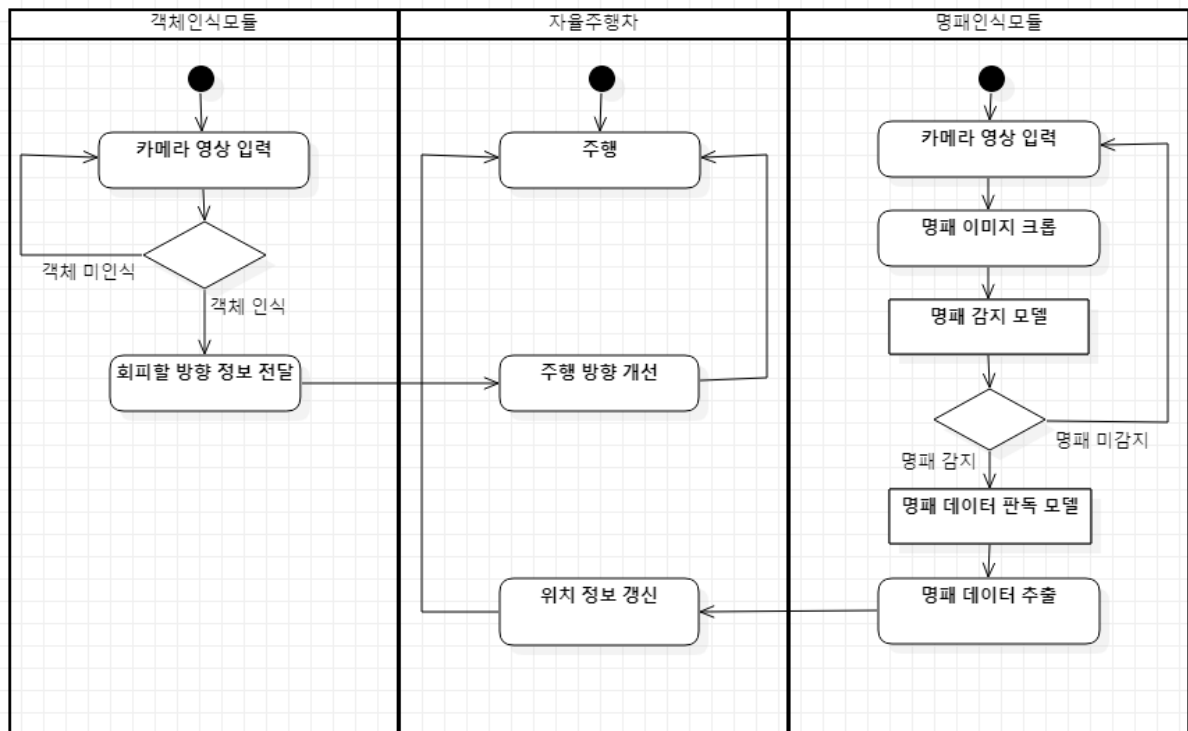
### 3.2.4. 명패 번호 (Door Plate Number) 인식용 데이터셋 활용

Number Detection 의 용도로 사용될 데이터셋은 3~4 자리의 숫자로 이루어진 이미지 집합을 요구한다. 이 경우 기존 연구에서 사용되었던 SVHN 데이터셋만으로는 높은 정확도를 기대하기 어려우므로 기존 SVHN 데이터셋에 실내 환경에 특화된 데이터를 포함하여 데이터셋을 새로 구축할 필요가 있다.

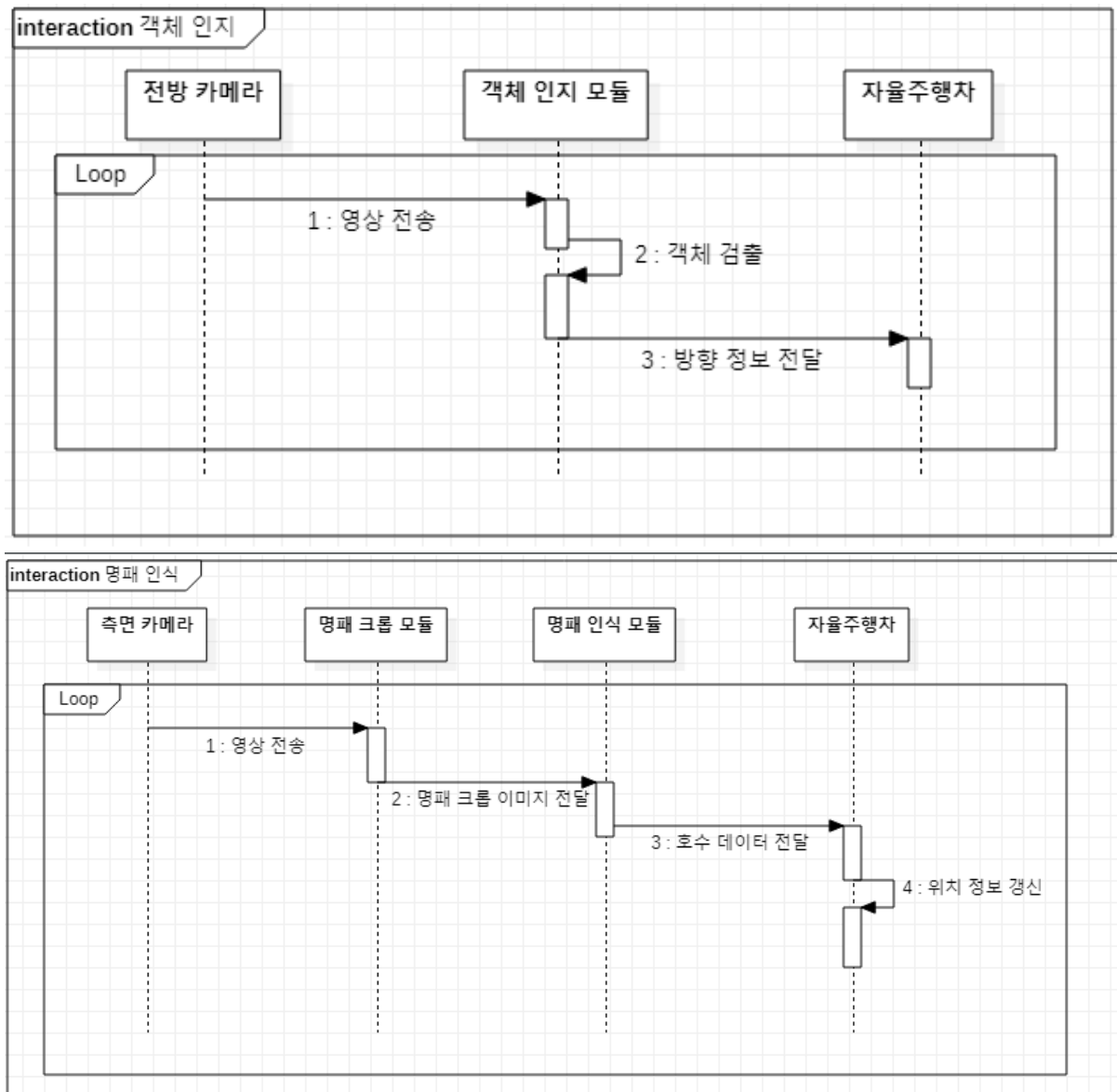
## 3.3. 시스템 설계

### 3.3.1. UML Diagram 을 통한 시스템 모델링

#### 3.3.1.1. Activity Diagram



### 3.3.1.2. Sequence Diagram



## 3.4 구현 과정

### 3.4.1. 개발 환경

구현하려는 명패인식(Door Plate Recognition), 사람 인지(Human Recognition)모듈은 라즈베리 파이의 python3 환경 상에서 개발 및 구축되었으며 이때 사용된 라이브러리 목록은 다음과 같다.

- OpenCV – Python [실시간 컴퓨터 비전을 목적으로 한 프로그래밍 라이브러리]
- Numpy [행렬이나 다차원 배열을 쉽게 처리 할 수 있도록 지원하는 라이브러리]
- Tensorflow[ 다양한 작업에대해 데이터 흐름 프로그래밍을 위한 오픈소스 소프트웨어 라이브러리]
- Keras [오픈소스 신경망 라이브러리]

### 3.4.2. Door Plate Crop 모듈 개발

좁은 복도가 존재하고 양 옆에 명패가 부착되어 있는 실내환경에서 카트의 현재 주행위치를 보다 정확하게 파악하기 위해 우선적으로 명패영역 이미지를 추출하는 것이 우선적으로 필요하였다. 그러나 실내의 반사율이 높은 환경, 라즈베리파이 카메라모듈의 해상도 및 컴퓨팅 성능 제약, 각도에 따른 형태 왜곡 등의 문제로 초창기 명패가 아닌 다른 이미지가 크롭되는 잡음이 상당수 발생하였다. 이에 따라, 잡음을 낮추는 방법으로, Canny Edge 검출법과 평균색 추출 함수를 별도로 작성하여 해당 이미지 크롭 정확도를 개선하였다.



[그림 8] 정확한 명패 크롭이미지(좌)와 잘못된 이미지 크롭 예시(우측 3개 이미지)  
이미지 크롭과정에서 작성된 소스코드와 세부 설명은 다음과 같다.

```
def filter_img(img):  
    #이미지의 RGB 값을 분석하여 찾는 실내 Tag 가 맞는지 판별  
    img = cv2.resize(img, (10,10))  
    first = [0,0,0]  
    for x_loc in range(0, 10):  
        for y_loc in range(0, 10):  
            bgr_value = img[x_loc,y_loc]  
            first=first+bgr_value  
    first[0] = first[0]/100  
    first[1] = first[1]/100  
    first[2] = first[2]/100  
    print(first)  
    blue = first[0]<200 and first[0]>120  
    green = first[1]>120 and first[1]<210  
    red = first[2]>130 and first[2]<230  
    if(blue and green and red):  
        return False  
    else:  
        return True
```

먼저 크롭된 이미지에 대해 평균색상을 추출 및 비교하는 함수이다. 해당 함수는 크롭된 영역의 이미지를 10\*10 픽셀로 resize 하고 이후 해당 이미지에 대해 평균 색상을 BGR 순서로 추출한다. 이후 사전에 조사한 명패의 색상 임계값을 토대로 해당 이미지를 필터링하였다. 해당 프로젝트에서 사용된 실내환경의 색상 임계값은 다음과 같다.

R : 130~230  
G : 120~210  
B : 120~200

```

def bboxes(inp,prevTime):
    #Frame 을 인자로 전달받음
    img = inp
    index = prevTime
    start = time.time()
    curTime = time.time()
    img_final = inp
    img2gray = cv2.cvtColor(inp, cv2.COLOR_BGR2GRAY)
    #GRAY Image 8bit per pixel
    img_canny = cv2.Canny(img2gray, 50, 150)
    contours, _ = cv2.findContours(img_canny, cv2.RETR_CCOMP,
    cv2.CHAIN_APPROX_SIMPLE)
    # get contours
    #RETR_CCOMP : 이미지에서 모든 윤곽선 추출
    #APPROX_SIMPLE : Contour 를 구성하는 끝점만을 저장
    #APPROX_NONE :: Contour 를 구성하는 모든 점 저장
    for contour in contours:
        if w > 50 or h > 35 or w<13:
            continue
        if h / w > 1.0 or w / h > 2.0:
            continue
        if h>40 or w>70:
            continue
        if y>150:
            continue
        cropped = img_final[y :y + h , x : x + w]
        if(filter_img(cropped)):
            #cv2.rectangle(img, (x-10, y-20), (x + w + 10, y +
h+10), (0, 0, 255), 3)
            cropped = img_final[y :y + h , x : x + w]
            cropped = cv2.cvtColor(cropped, cv2.COLOR_BGR2RGB)
            cropped = cv2.resize(cropped, (48,48))
            index = index + 1
        else:
            continue
    sec = curTime - prevTime
    prevTime = curTime
    try:
        fps = 1/(sec)
    except ZeroDivisionError:
        fps = 0
    str1 = ("FPS : {0}".format(int(fps)))
    cv2.putText(img, str1, (0, 40),
    cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.8, (0, 255, 0),1)
    cv2.imshow('captcha_result', img)
    return index

```

이미지를 크롭하는 bboxes 함수의 경우 프레임을 그레이스케일로 변환, 변환한 이미지를 바탕으로 Canny Edge Detection 을 적용하였다.

Canny Edge 검출 알고리즘은 노이즈제거, Gradient 값이 높은 부분을 찾고 최대값을 가진 픽셀만을 검출하여 최종적으로 Thresholding 을 적용하는 다단계 알고리즘으로 구성되어있다. 구체적인 내용은 다음과 같다.

1. 5X5 가우시안 필터를 적용, 노이즈 제거
  2. Gradient 값이 높은 부분 찾기
  3. Gradient 최대값이 아닌 픽셀의 값을 0 으로 만들기
  4. Hyteresis Thresholding
- 3 번의 도출된 엣지가 실제 엣지인지 판단하는 단계로 임계값을 기준으로 엣지 검출

위와 같은 과정으로 도출된 Edge 를 기준으로 윤곽선을 검출하였다. 이때, Open CV 의 Find Contour 함수를 통해 윤곽선을 검출할 수 있도록 구현하였으며 적용한 옵션은 다음과 같다.

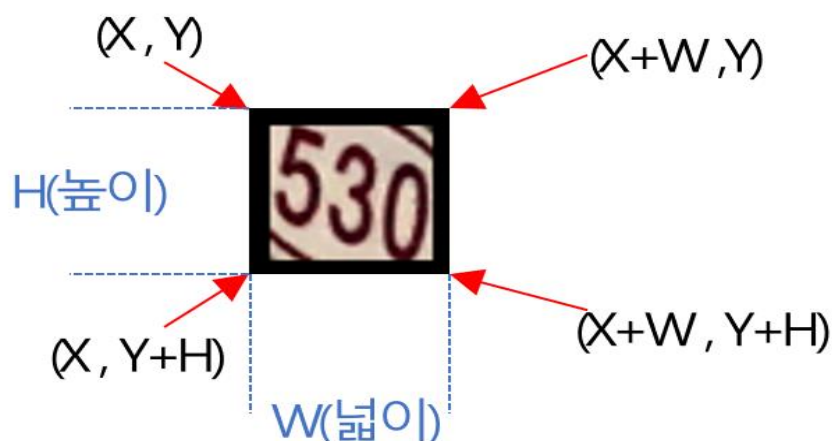
1. Find Contour 의 추출모드 옵션 RETR\_CCOMP 적용

해당 옵션은 이미지에서 모든 윤곽선을 검출하는 옵션인데, 실내 명패의 이미지가 전체 프레임 크기와 비교하여 지나치게 작아 해당 옵션을 통해 면밀한 윤곽선을 검출할 수 있도록 하였다.

2. Contour 근사 방법 APPROX\_SIMPLE 옵션 적용

주어진 Edge 에 대해서 윤곽선의 외곽지점만을 검출하는 옵션이다.

이후, 해당 작업을 통해 윤곽선의 가장 외곽지점 4 점이 검출된 이미지 x,y,w,h 값을 통해 추출된다. 이때 x,y,w,h 값은 아래 그림의 픽셀 위치와 같다.

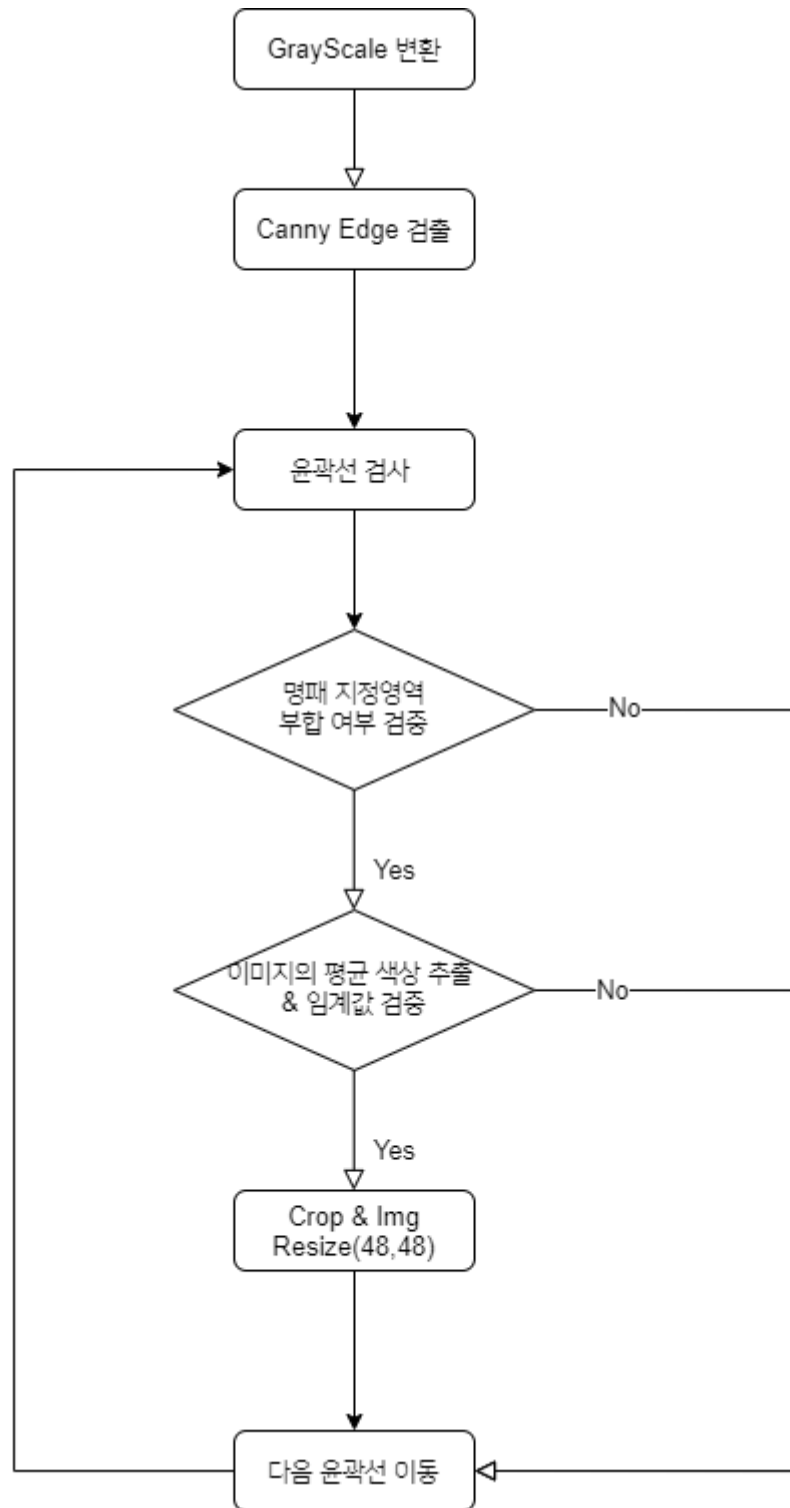


[그림 9] 크롭이미지 좌표

추출된 이미지는 명패의 위치가 지정 영역을 벗어나지 않는지 검증하게 된다. 해당 조건을 부합하게 되면 최종적으로 기존에 작성한 filter\_img 함수에 의해 이미지의 평균 색상이 임계값 기준에 부합하는지 판단하게 되며 해당 조건을 만족하면 해당 이미지는 크롭되어 48\*48 size 로 변환된다.

만약 조건을 만족하지 못했을 경우 for 문 반복구조에 의해 다음 Edge 를 기준으로 해당 알고리즘을 반복하게 된다.

해당 소스코드에 대한 대략적인 플로우차트는 다음과 같다.

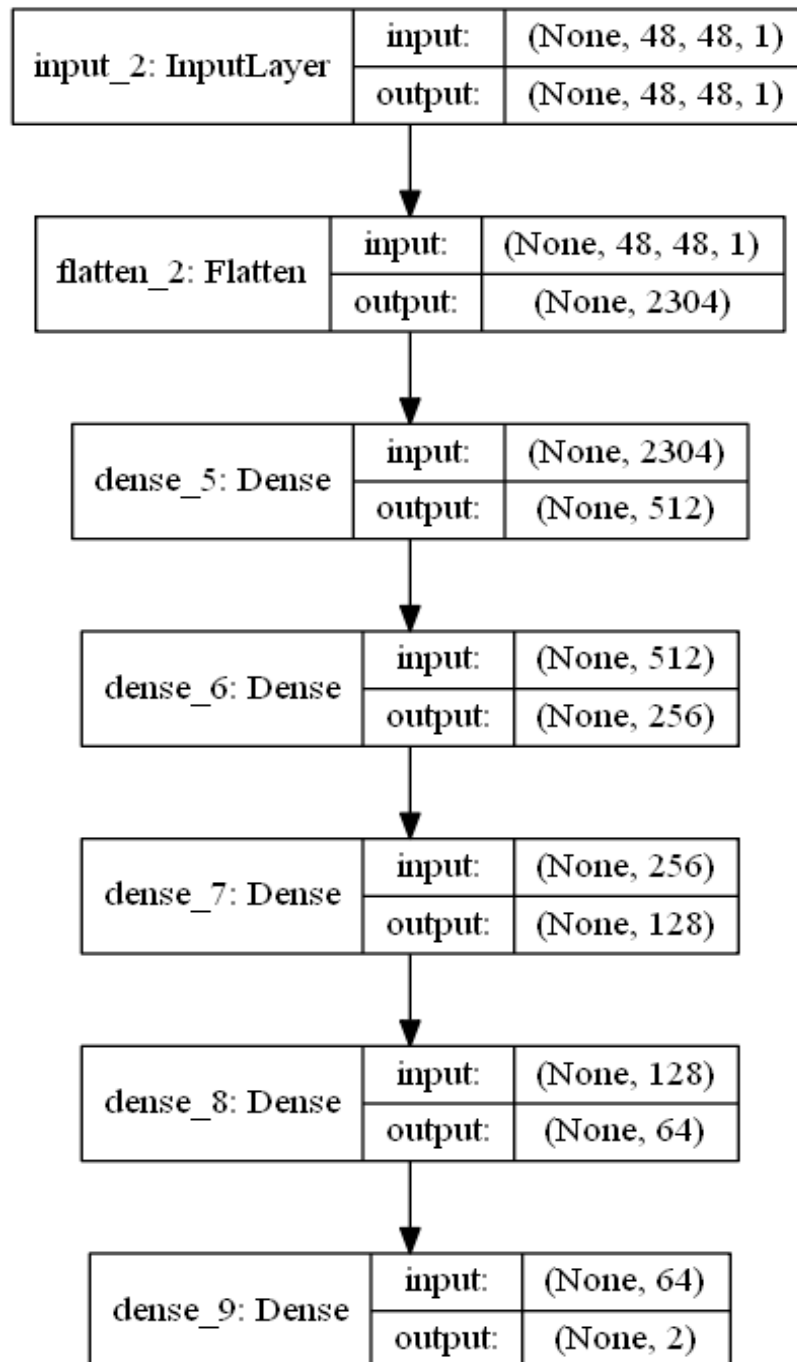




### 3.4.3. Noise Decision Model 구축

Number Recognition Model 은 48\*48 이미지에서 숫자를 3-4 개 감지하는 모델로 복잡하고 많은 파라미터를 가지고 있기 때문에 라즈베리 파이의 하드웨어 한계 상 모든 크롭 이미지를 모델에 넣을 수 없다. 따라서 본 연구에서는 경량화된 노이즈 판별 모델을 추가로 구축, 크롭된 이미지를 한 번 더 걸러내어 숫자가 포함되었다고 강하게 추측되는 이미지만 선별하는 로직을 추가했다. 여기서 Noise Decision Model 이 해당 역할을 하는 경량 모델이다.

Noise Decision Model 로는, 미리 훈련된 VGG-16 모델을 기반으로 제작된 Number Recognition Model 에 비해 훨씬 경량화된 모델이 요구되어 최대한 적은 파라미터 수를 갖게 모델을 구성했다.



[그림 10] Noise Decision Model 구성

```
def designed_model():
    inp = Input(shape=(48, 48, 1))
    x = Flatten()(inp)
    x = Dense(512, activation='relu')(x)
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    x = Dense(64, activation='relu')(x)
    out = Dense(2, activation='softmax')(x)

    model = keras.Model(inputs=inp, outputs=out)
    optim = optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999,
epsilon=None, decay=0.001, amsgrad=True)
    model.compile(loss='categorical_crossentropy', optimizer=optim,
metrics=['accuracy'])
    return model
```

Grayscale 로 미리 처리된 48\*48\*1 이미지 텐서가 입력으로 주어지고, 5 개의 Fully Connected Layer 를 배치했다. 숫자가 포함되었는지 확인하는 것이 목적이기 때문에, CNN 을 이용해 모델을 제작해보기도 했지만 파라미터 수가 줄어든 대신 오히려 복잡성이 증가하여 만족스러운 결과를 얻지 못했다.

출력은 softmax 함수를 통과해 [0, 1]의 확률을 나타낸 배열로, 0 일 확률이 높으면 이미지에 숫자가 들어 있지 않은 것으로 판별, 1 일 확률이 높으면 이미지에 숫자가 들어있는 것으로 판별한다.

optimizer 로 Adam 을 사용하였고, 하이퍼 파라미터는 SVHN 을 이용한 기존 연구의 파라미터와 같은 값을 사용하였다.

### 3.4.4. Number Recognition Model 구축

Number Recognition Model 에는 opencv 를 이용한 크롭 이미지 전략과, Noise Decision Model 을 통과한 이미지만이 입력으로 주어지므로, 숫자가 들어있지 않을 확률은 매우 낮다. 다만, 위의 전략과 모델이 완벽하지 않으므로 본 모델에도 숫자가 들어있는지 판별하는 기능이 포함되어야 한다. 따라서, 본 모델에게 요구되는 기능은 숫자 판별, 숫자의 개수 판별, 1 번째 자리 ~ 4 번째 자리의 숫자 판별로 총 3 개이다.

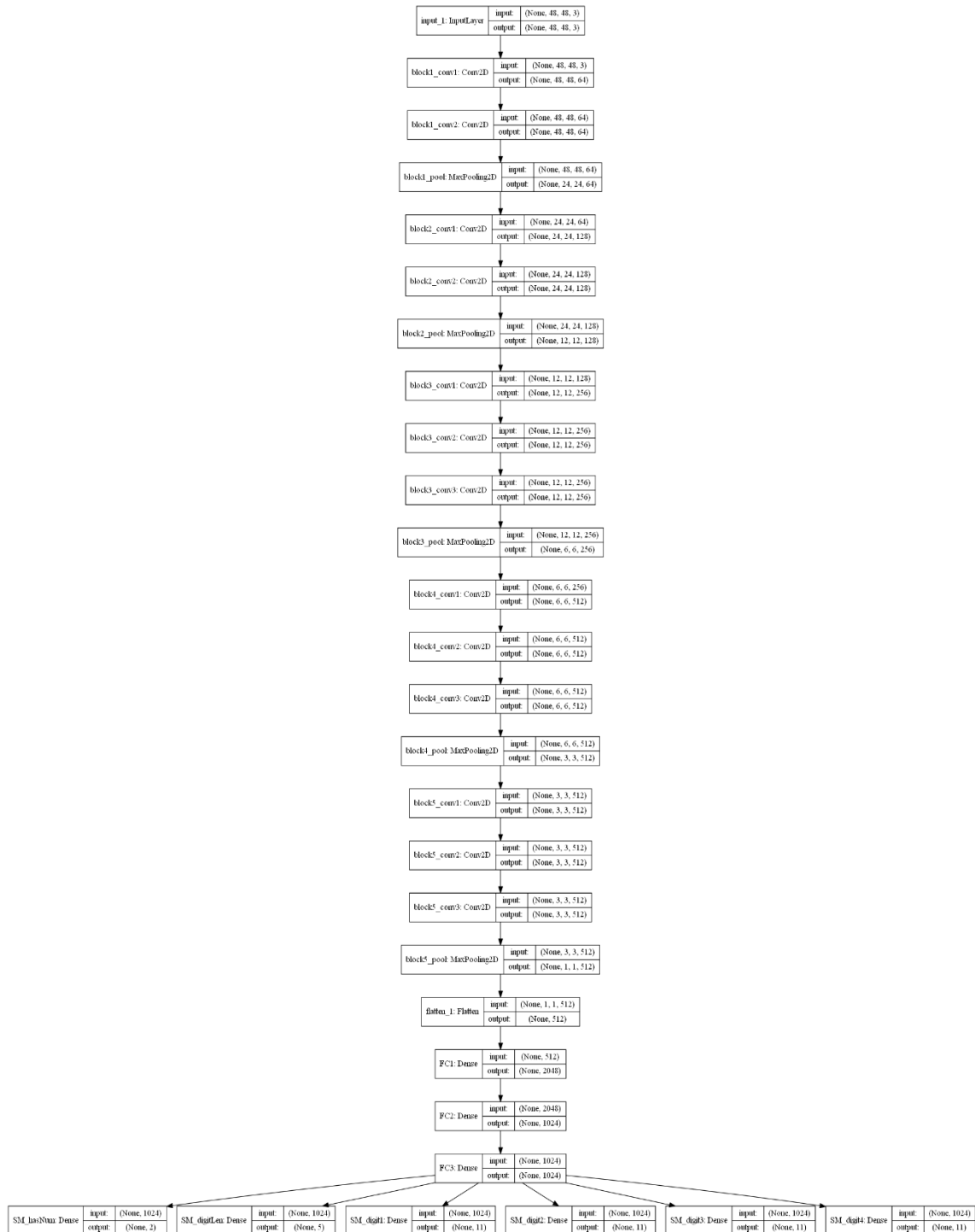
본 모델에는 입력으로 48\*48\*1 Grayscale 이미지가 주어지고, 이를 미리 훈련된 VGG-16 모델에 넣은 후, VGG-16 모델의 출력을 4 개의 Fully Connected Layer 를 거치게 해서 출력을 만든다.

다만, VGG-16 모델은 RGB 이미지를 기준으로 제작되었기에 3 개의 채널이 있어야하므로 48\*48\*1 크롭 이미지를 3 번 중첩하는 방식으로 48\*48\*3 텐서를 구성하여 입력하였다.

VGG-16 모델에서 1\*1\*512 텐서가 출력되는데, 이를 512 사이즈의 텐서로 변환 후, 3 개의 Fully Connected Layer 를 거친다. 이 때, 출력된 1024 사이즈의 텐서를 6 개의 FC Layer 의 입력으로 받아 최종적인 결과가 나온다.

말단 출력층의 Layer 들의 구성은 다음과 같다.

1. [0, 1]의 출력, 숫자 포함 판별 기능.
2. [0, 1, 2, 3, 4]의 출력, 이미지에 포함된 숫자의 개수 판별 기능. 주로 0, 3, 4 의 값이 많이 예측된다.
3. [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]의 출력. 첫번째 자리의 숫자 판별 기능. (10 은 Null 을 의미한다.)
4. [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]의 출력. 두번째 자리의 숫자 판별 기능.
5. [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]의 출력. 세번째 자리의 숫자 판별 기능.
6. [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]의 출력. 네번째 자리의 숫자 판별 기능.

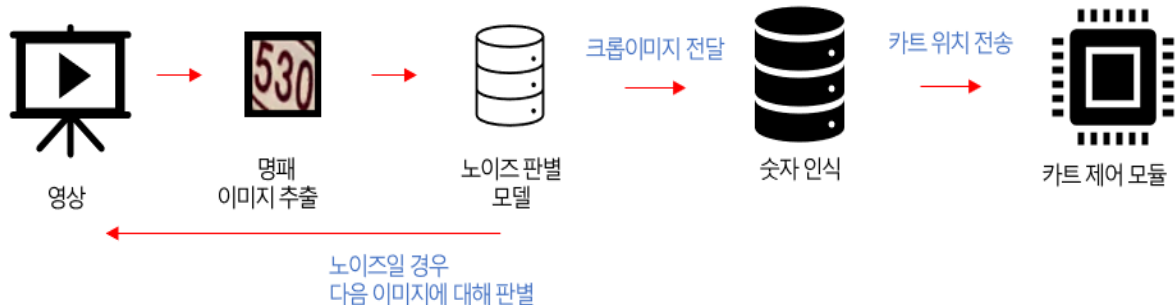


[그림 11] Number Recognition Model 구성

### 3.4.5. Door Plate Recognition 모듈 통합

Door Plate Recognition 모듈은 다음 소스코드를 통해 통합 구현하였다.

모듈의 개념적 진행 내용은 다음과 같다.



[그림 12] Door Plate Recognition 모듈 구성도

소스코드는 다음과 같다. 이전 설명된 모듈과 모델 간의 결합이므로 소스코드에 대한 세부 설명은 생략한다.

```
labeling_module.py
# plaidml
# import plaidml.keras
# plaidml.keras.install_backend()
# packages
from keras.models import load_model
from keras.preprocessing import image
# import queue
import numpy as np
from queue import Full, Empty
from multiprocessing import Process, Queue
class LabelingModule:
    def __init__(self):
        # self.model1 = load_model('svhn_model.h5')
        self.model2 = load_model('svhn_model.h5')
        self.image_queue = Queue(maxsize=3000)
        self.label_queue = Queue(maxsize=10)
        self.signal_queue = Queue()
        self.predict_process = Process(target=_predict, \
            args=(self.model2, self.image_queue, self.label_queue,
self.signal_queue))
    def run(self):
        self.predict_process.start()
    def close(self):
        self.image_queue.close()
        self.label_queue.close()
    def new_tensor(self, tensor):
        try:
            self.image_queue.put(tensor)
        except Full:
            print('[LabelingModule] image_queue is full')
    def new_image(self, filename):
        tensor = self._img_to_tensor(filename)
        try:
```

```

        self.image_queue.put(tensor)
    except Full:
        print('[LabelingModule] image_queue is full')
def _img_to_tensor(self, filename):
    img = image.load_img(filename, target_size=(48, 48))
    img_tensor = image.img_to_array(img)
    img_tensor = np.squeeze(img_tensor)
    img_tensor /= 255.
    img_tensor = img_tensor - img_tensor.mean()
    return img_tensor
def _predict(model, input_queue, output_queue, signal_queue):
    print('predict process started.')
    while True:
        try:
            signal = signal_queue.get_nowait()
            if signal == 'stop':
                break
        except Empty:
            pass

        tensor = input_queue.get(timeout=-1)
        dat = model.predict(np.array([tensor]))
        o1 = np.argmax(dat[0])
        o2 = np.argmax(dat[1])
        o3 = np.argmax(dat[2])
        o4 = np.argmax(dat[3])
        o5 = np.argmax(dat[4])
        o6 = np.argmax(dat[5])
        output = [o1, o2, o3, o4, o5, o6]
        print('[LabelingModule] predict result :', output)

```

Main.py

```

import cv2
import numpy as np
import time
from multiprocessing import Queue
from labeling_module import LabelingModule
fname = "./croppedimg/"
index = 0
prevTime = 0
lm = LabelingModule()
def filter_img(img):
    #이미지의 RGB 값을 분석하여 찾는 실내 Tag 가 맞는지 판별
    img = cv2.resize(img, (10,10))
    first = [0,0,0]
    for x_loc in range(0, 10):
        for y_loc in range(0, 10):
            bgr_value = img[x_loc,y_loc]
            first=first+bgr_value
    first[0] = first[0]/100
    first[1] = first[1]/100
    first[2] = first[2]/100

```

```

blue = first[0]<200 and first[0]>120
green = first[1]>120 and first[1]<210
red = first[2]>130 and first[2]<230
if(blue and green and red):
    return True
else:
    return False
def bboxes(inp,prevTime):
    #Frame 을 인자로 전달받음
    img = inp
    start = time.time()
    curTime = time.time()
    img_final = inp
    img2gray = cv2.cvtColor(inp, cv2.COLOR_BGR2GRAY) #GRAY Image 8bit
per pixel
    img_canny = cv2.Canny(img2gray, 50, 150)
    contours, _ = cv2.findContours(img_canny, cv2.RETR_CCOMP,
cv2.CHAIN_APPROX_SIMPLE)
    for contour in contours:
        [x, y, w, h] = cv2.boundingRect(contour)
        if w > 50 or h > 35 or w<13:
            continue
        if h / w > 1.0 or w / h > 2.0:
            continue
        if h>40 or w>70:
            continue
        if y>150:
            continue
        cropped = img_final[y :y + h , x : x + w]
        # draw rectangle around contour on original image
        if(filter_img(cropped)):
            cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 3)
            cv2.putText(img,"cropped", (x-50,y-10),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1, (0,0,255), 1)
            cropped = img_final[y :y + h , x : x + w]
            cropped = cv2.cvtColor(cropped, cv2.COLOR_BGR2RGB)
            cropped = cv2.resize(cropped, (48,48))
            lm.new_tensor(cropped)
        else:
            continue
    img = cv2.resize(img, (720, 380))
    sec = curTime - prevTime
    prevTime = curTime
    try:
        fps = 1/(sec)
    except ZeroDivisionError:
        fps = 0

    str1 = ("FPS : {}".format(int(fps)))
    cv2.putText(img, str1, (0, 40), cv2.FONT_HERSHEY_COMPLEX_SMALL,
0.8, (0, 255, 0),1)
    cv2.imshow('captcha_result', img)

```

```

    return prevTime
if __name__ == "__main__":
    lm.predict_process.start()
    cap = cv2.VideoCapture(0) #동영상 파일 읽어옴
    while (cap.isOpened()):
        ret, inp = cap.read() #프레임을 읽어옴, 읽어온 프레임을 인자로
        bboxes 전달
        if(ret): #success boolean
            prevTime = bboxes(inp, prevTime)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                print("Terminate Process..")
                break
    cap.release() #파일 닫아줌
    lm.predict_process.join()

```

### 3.4.6. Human Recognition 모듈 개발

사람의 왕래가 잦고, 장애물의 간섭이 빈번한 실내 환경의 특성과 좁은 복도가 존재하는 물리적 환경을 고려하여 Human Recognition 을 통해 카트의 예상 주행 경로를 파악할 수 있도록 설계하였다.

최종적으로 근거리의 경우, LIDAR 센서를 통해 우발적 대처를 가능케 하였으며 원거리(5m 이상)의 경우 사람 객체를 인식하여 회피주행 할 수 있도록 해당 모듈을 개발하였다. Human Recognition 은 OpenCV 의 Hog 디스크립터와 SVM 모델을 기반으로 설계하였으며 예상 경로를 1 차원 배열로 반환하고, 화면상에 실시간으로 예상 경로를 그릴 수 있도록 구현하였다.

해당 모듈의 상세 코드는 다음과 같다.

```

def draw_left_path(img,x,y,w,h):
    start_point = x+w
    horizon = 640
    vertical = 480
    cv2.line(img, (horizon/2-2*w,vertical), (start_point, y+int(h)), (255,0,0),
8) #8 픽셀 선 그리기
    cv2.line(img, (start_point, y+int(h)), (start_point+int(w/5),y+int(h-20)),
(255, 0, 0), 10)
    start_point = x+2*w
    cv2.line(img, (horizon*0.8,vertical), (start_point, y+int(h)), (255,0,0), 8)
#8 픽셀 선 그리기
    return img

def draw_right_path(img, x, y, w, h):
    start_point = x
    horizon = 640
    vertical = 480
    cv2.line(img, (horizon/2+2*w, vertical), (start_point, y+int(h)), (255, 0,
0), 8) # 8 픽셀 선 그리기
    cv2.line(img, (start_point, y+int(h)), (start_point - int(w/5), y+int(h-
20)), (255, 0, 0), 8)
    start_point = x-2*w

```

```

        cv2.line(img, (horizon*0.1, vertical), (start_point, y + int(h)), (255, 0,
0), 8) # 8 픽셀 선 그리기
return img

```

먼저 라즈베리 파이 LCD 해상도(640\*480)에 맞게 해당 경로를 출력해주는 함수를 작성하였다. 각 함수는 인식된 객체가 화면의 왼쪽 혹은 오른쪽에 위치할 때 호출되며, 객체의 좌표를 기준으로 카트의 회피경로를 계산하여 화면에 실시간으로 그려준다.

```

hog = cv2.HOGDescriptor()
hog.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())
weight_list = [0,0,0,0,0,0,0,0,0,0]
prevTime = 0
def check_obstacle(weight_list, xA, xB):
    for i in range(0, len(weight_list)):
        weight_list[i] = 0
    xA = int(xA/32)
    xB = int(xB/32)
    for i in range(xA, xB+1):
        if(weight_list[i]==0):
            weight_list[i] = weight_list[i]-1
    return weight_list
while (True):
    # c 이미지 캡처
    start = time.time()
    curTime = time.time()
    ret, frame = cap.read()
    # 프레임 resize
    frame = cv2.resize(frame, (320, 240))
    # Human Recognition
    # 탐지된 객체에 대해 Bbox 리턴
    boxes, weights = hog.detectMultiScale(frame, winStride=(8, 8))
    detectCount = 0
    boxes = np.array([[x, y, x + w, y + h] for (x, y, w, h) in boxes])
    for (xA, yA, xB, yB) in boxes:
        # bbox 그리기
        w = xB-xA
        h = yB-yA
        cv2.rectangle(frame, (xA, yA), (xB, yB),
            (0, 255, 0), 2)
        cv2.putText(frame, "Detect", (xA - 50, yA - 10), cv2.FONT_HERSHEY_COMPLEX_SMALL,
1, (0, 0, 255), 1)
        detectCount = detectCount+1
        if(detectCount>1):
            print("Waiting...")
        else :
            print(check_obstacle(weight_list,xA, xB)) #현재 카트 전방의 장애물 상황 1 차원
배열 형태로 출력
            if(i%10 == 0):
                cropped = frame[yA:yB,xA:xB]
                i=0
            if(xB < 190 and xA<130): #카트의 좌측에 객체 위치
                print("Left Side Detect.")
                try:
                    frame = draw_left_path(frame, xA,yA,xB-xA,yB-yA)
                except:
                    pass
            elif(xA>130 and xB>190): #카트의 우측에 객체 위치
                print("Right Side Detect")
                try:
                    frame = draw_right_path(frame, xA, yA, xB - xA, yB - yA)
                except:
                    pass
            else:

```



```

try:
    frame = draw_right_path(frame, xA, yA, xB - xA, yB - yA)
except:
    pass
print("Center Side Detect")
print("time :", time.time() - start)
sec = curTime - prevTime
prevTime = curTime
fps = 1/(sec)

```

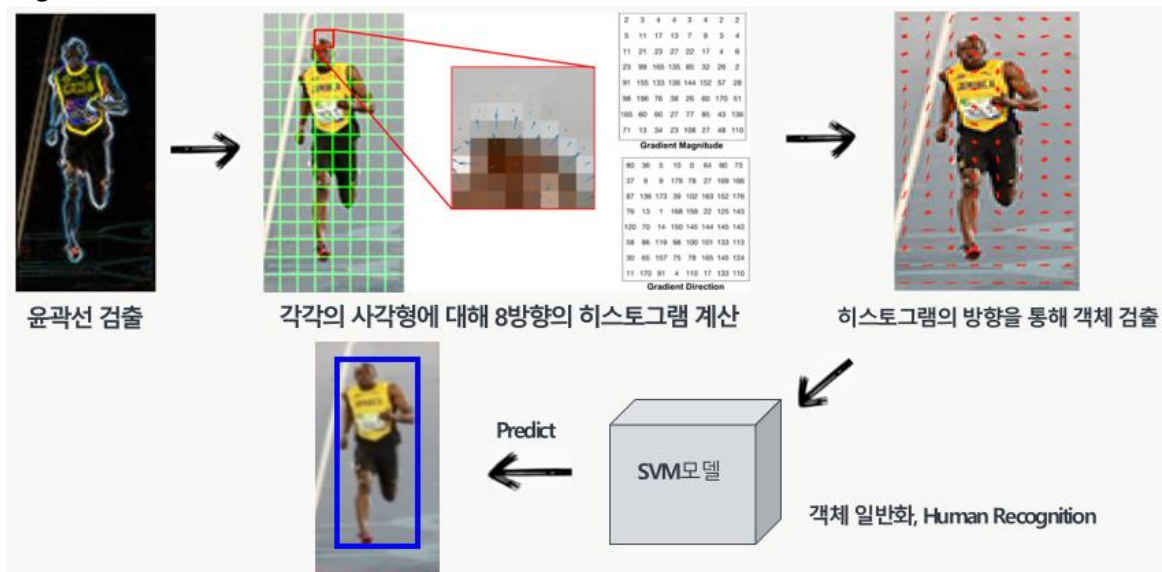
사람 객체의 인지를 위해 Open CV 의 Hog 디스크립터를 사용하였다. 코드 전체의 대략적인 내용은 다음과 같다.

1. 프레임 사이즈 조정 및 그레이 스케일 변환
2. Hog 디스크립터를 사용하여 객체 탐지
3. 탐지된 객체에 대해 bbox(bounding box) Drawing.
4. 회피 주행 경로 출력 및 1 차원 배열 반환

[사람이 여러명이 있고 복도의 공간이 여유롭지 않을 경우 대기]

해당 모듈에서 사용된 Hog 디스크립터는 보행자 검출 목적의 디스크립터로, 대상 객체의 일반화에 매우 적합한 디스크립터이다.

Hog 디스크립터의 동작과정은 다음과 같다.



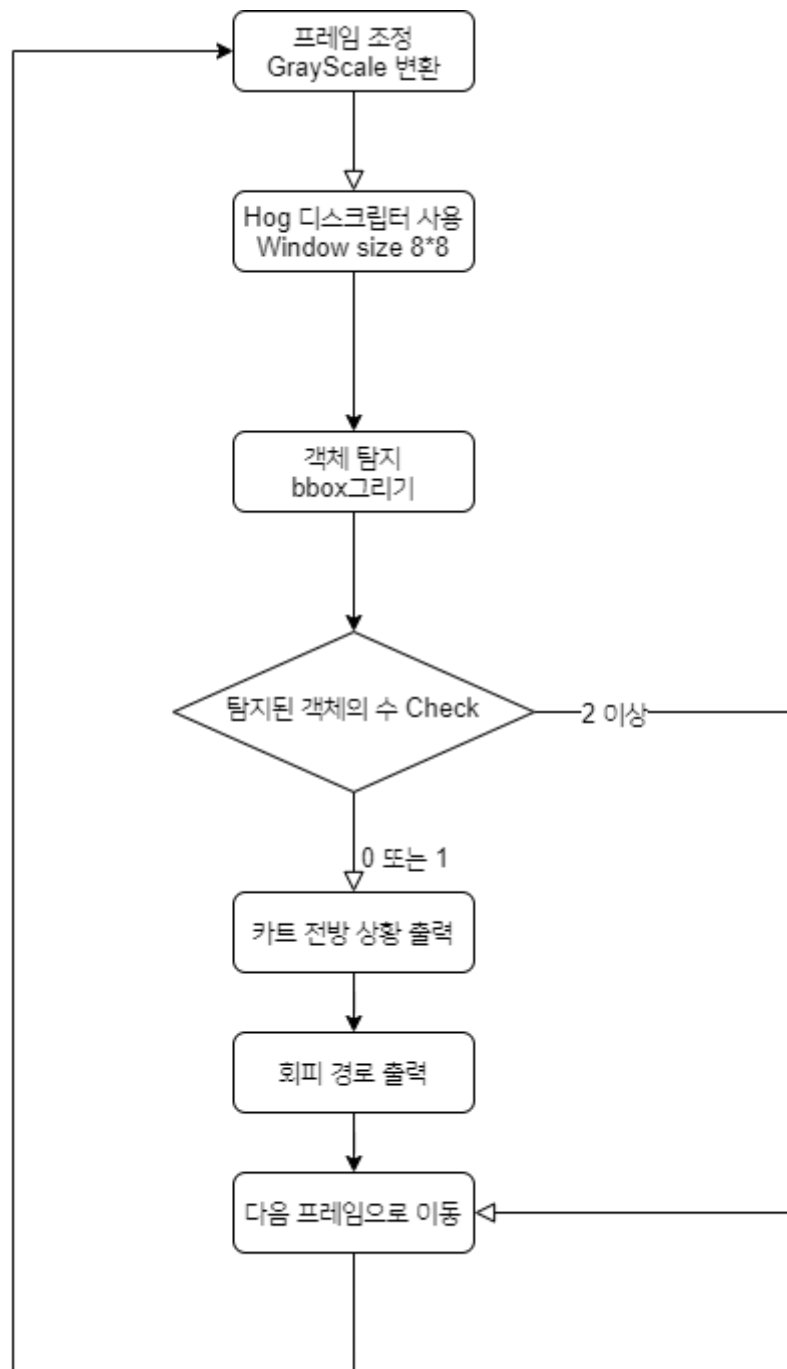
[그림 12] Hog 디스크립터를 통한 사람 객체 검출 과정

Hog 디스크립터는 단순히 윤곽선이 아닌, window 크기에 따른 8 방향의 히스토그램을 계산한다. 객체를 인지하는 순서는 다음과 같다.

1. 영상으로부터 Gradient 를 계산한다.
2. Gradient 를 이용하여 Local Histogram 을 생성한다.
3. Local Histogram 을 이어붙여 1 차원의 Vector(HOG Feature)를 생성한다.
4. 검출된 객체를 학습된 SVM 모델의 입력값으로 반환한다.
5. 사람 객체 검출

위의 과정을 통한 HOG 디스크립터의 경우 조명 변화의 영향을 적게 받으며 대상을 일반화하여 검출할 수 있다는 장점이 있다. 또한, 대상객체의 방향에 구애받지 않고 사람객체를 인지할 수 있어 보다 정확한 예측 및 인지가 가능하다.

해당 코드에 대한 개략적인 플로우차트는 다음과 같다.



## 4. 프로젝트 성능 개선

### 4.1. 명패 크롭 이슈[Door Plate Crop Issue]

개발 초기, 3.2.3 절의 명패 인식 과정에서 실내 조명의 영향과 Door Plate 의 외형이 제각각 상이함에 따라 이미지 크롭의 잡음이 지나치게 발생하여 정확한 실내 정보 파악이 불가능 하였다. 때문에 여러 크롭 전략을 마련하여 100 장의 이미지를 대상으로 <sup>6</sup>Door Plate Detection 실험을 진행하였으며 최종적으로 크롭 이미지 정확도를 84%까지 개선하였다.

#### 4.1.1. 전략 1) Gaussian Blur 적용

Door Plate의 작은 윤곽선이 Blur처리 되어 정확한 크롭이 불가능하였다. 크롭 정확도는 8%에 그쳤으며, 나머지 92%의 이미지가 잡음이었다.

#### 4.1.2. 전략 2) Gaussian Blur 미적용, 크롭 이미지 Y좌표 150 픽셀 이상의 잡음 배제

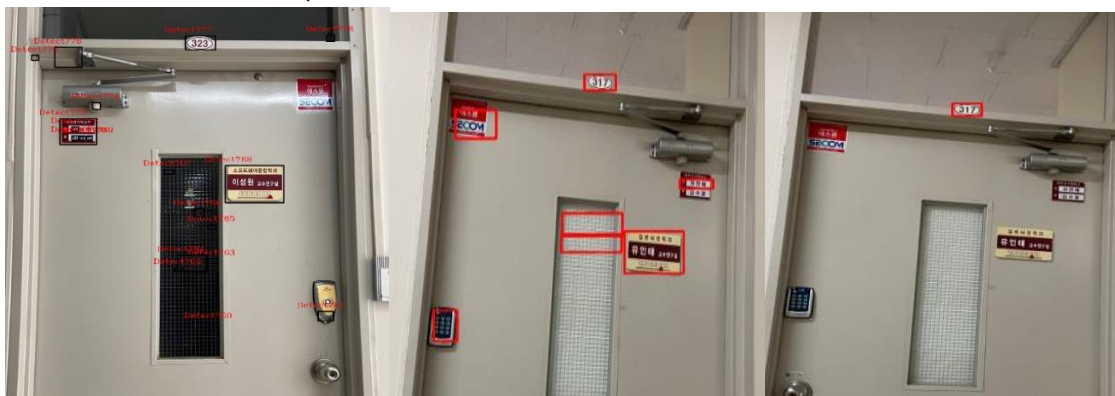
Door Plate 가 영상의 상단에 위치하고 있는 점을 고려하여 특정 영역에서만 Detection 이 이루어지도록 구성하였다. 정확도는 약 42%에 달했으나 여전히 크롭 이미지 중 대부분이 잡음이었다.

#### 4.1.3. 전략 3) 크롭된 이미지의 평균 RGB 값 추출, 범위 만족 여부 확인, Canny Edge Detection Algorithm 적용

기존 크롭 전략 2 번에 덧붙여 크롭된 이미지를 10\*10 픽셀크기의 이미지로 resize 하였다. 이후 이미지의 전체 평균 RGB 색상을 추출하여, 기존 크롭 이미지들의 평균 RGB 색상을 비교, 색 범위 만족여부를 확인하였다. 또한 Canny Edge 검출 알고리즘을 적용하여 빛 반사율에 영향을 많이 받지 않도록 옵션을 주어 전략을 구성하였다. Door Plate Detection 실험 진행 결과 정확도는 약 84%로, 보다 정확한 Door Plate Detection 이 가능하였다.

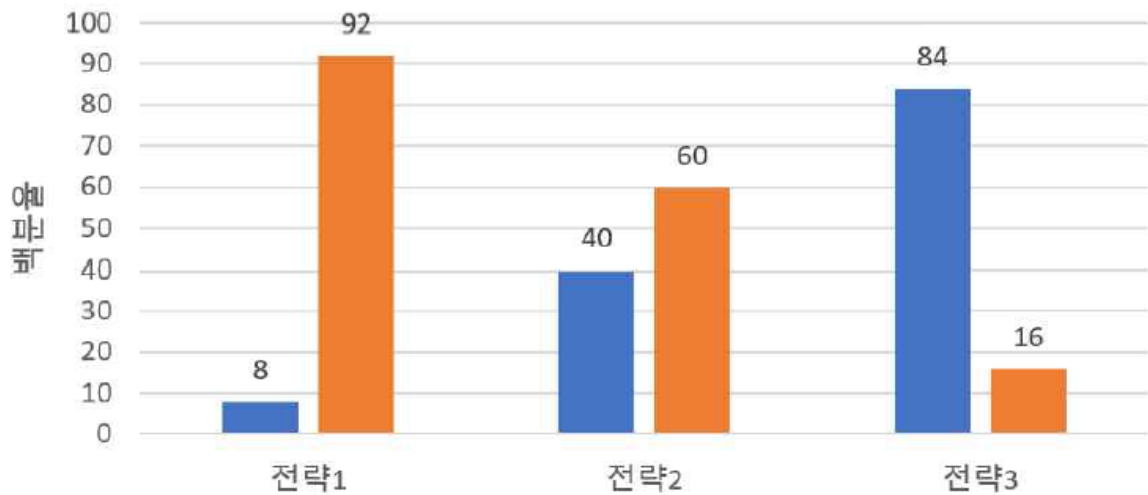
#### 4.1.4. 전략 평가 실험 결과

동일 영상에서 Door Plate를 대상으로 크롭의 성능 테스트를 진행하였음. 정확도는 크롭 이미지 100장 중 올바른 Crop이 이루어진 이미지의 비율로 규정함.



[그림 13] 전략별 이미지 잡음 비율 (왼쪽부터 전략1, 전략2, 전략3)

<sup>6</sup> 동일 영상에서 Door Plate를 대상으로 크롭의 성능 테스트를 진행 하였음. 정확도는 크롭이미지 100장 중 올바른 Crop이 이루어진 이미지의 비율로 규정함.



[그림 14] 전략별 크롭 정확도 측정 결과

## 4.2. 모델 예측도 이슈[Model Accuracy Issue - Number Recognition]

현실적인 실내환경을 고려해보았을 때, 각각의 명패가 훼손된 경우도 적잖이 있어 데이터셋 구축에 어려움이 있었다. 때문에 단순히 SVHN 을 사용하는 것이 아닌, 실내 환경에 특화된 데이터셋을 별도로 구축하여 이를 토대로 모델을 훈련하였으며 이때 각각의 전략은 다음과 같았다.

### 4.2.1. 전략 1) 실내 정보만으로 Data Set 구성, 예측 모델 구축

실내 정보만으로 Data Set 을 구성하여 모델을 훈련하였다. DataSet 은 경희대학교 전자정보대학의 명패를 기준으로 2 만 4 천장으로 구성하였으며 기존 이미지 800 장을 -8~10 도 까지 범위로 각도를 비틀어 데이터셋을 구성하였다. 다만, 층별로 명패가 훼손된 수가 다른 경우와 간혹 명패에 이물질이 묻어있는 경우에 의해 정확한 인식이 어려운 경우가 많았다. 이 경우, 예측 모델의 정확도는 약 20%에 불과하였다.

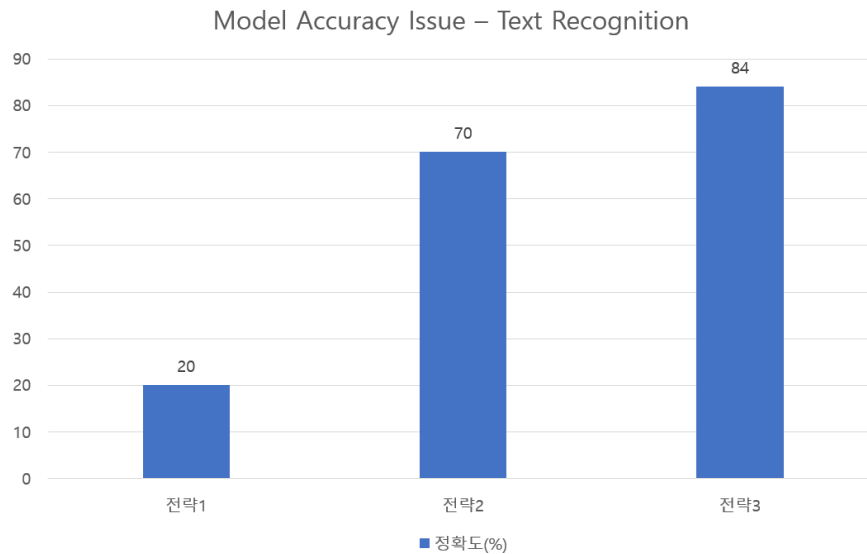
### 4.2.2. 전략 2) SVHN 데이터셋만을 이용한 모델

여러자리의 숫자를 인식하는 연구인 SVHN 연구의 데이터셋을 활용하여 모델을 구축하였다. 이 경우, SVHN 데이터셋 22 만장을 학습에 사용하였으며 모델예측 정확도는 약 70%에 달했다. 다만, 노이즈모델을 별도로 구축하지 않아 노이즈 이미지까지 숫자로 판별해버리는 문제가 발생하여, 실질적으로 이를 카트에 적용하기에는 무리가 있었다.

### 4.2.3. 전략 3) SVHN\_DateSet + 실내 특화 데이터셋

기존에 사용한 데이터셋인 전략 1 의 실내 특화 데이터셋과 더불어 전략 2 의 SVHN 데이터셋을 사용하여 모델을 구축하였다. 또한 Noise 모델을 구축하여 분명한 Door Plate Recognition 을 시도한 결과 모델 정확도는 약 84%에 달했다.

#### 4.2.4 전략 평가 실험 결과



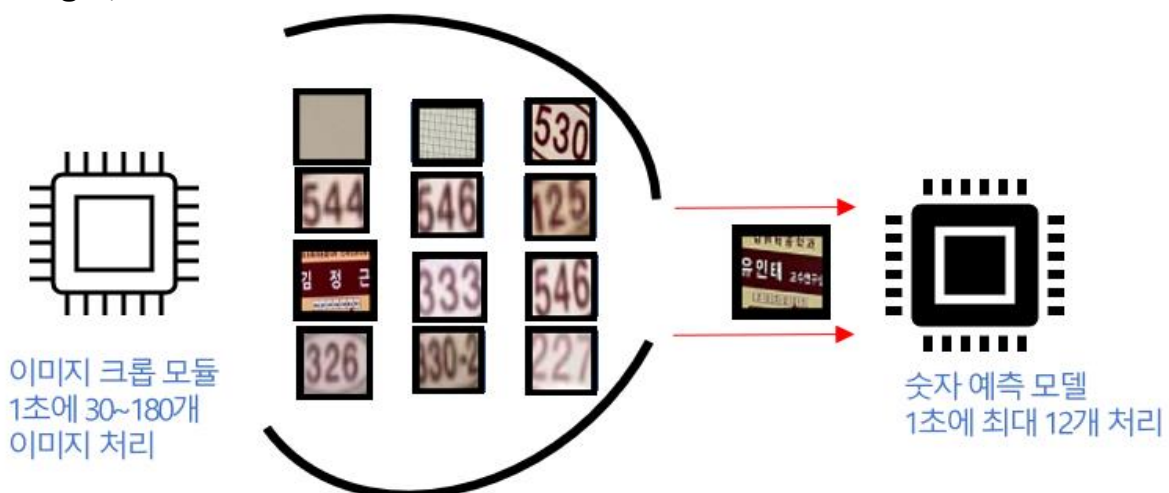
[그림 15] 모델 예측 정확도 전략별 분석

기존 실내 데이터 뿐 아니라, SVHN 의 다양한 숫자 데이터를 통해 모델을 학습시켜 정확도를 약 64%p 개선할 수 있었다. 또한, 프로젝트 후반 잡음 판별모델을 구축하여 Number Recognition 모델의 운용역시 안정적으로 유지할 수 있었다.

#### 4.3. 크롭-예측 병목현상 이슈[Bottle Neck Issue Between Crop and Prediction]

라즈베리파이 4 모델 B 환경 위에서 Door Plate Crop 은 평균적으로 30 프레임 이상 유지된다. 한 프레임 내에서 적게는 1 개에서 많게는 약 6 개까지의 이미지 크롭이 이루어지는데 이는 모델의 입력값으로 주어진다. 개발 초기, 해당 과정에서 병목현상이 발생하여 프로그램이 일정 시간 이후, 강제 중단되는 경우가 빈번하였다. 이에 분석한 문제점은 다음과 같다.

- Crop 프레임 : 평균 30FPS 이상, 한 프레임에 대해 평균 2~3 개 이미지 크롭.
- Predict 지연시간은 약 0.1 초로 1 초당 평균 10 개의 이미지에 대해 예측 가능
- Crop & Predict 의 처리속도 간극에 의해 병목현상 발생. 일정시간 이후 프로그램 강제 종료.



[그림 16] 모듈간 처리 속도 차이에 따른 병목현상

#### 4.3.1 전략 1) 병목현상 해소를 위한 타임스탬프 구현

숫자인식모델이 이미지들에 대해 숫자 예측을 할 경우, Queue 에서 crop 된 이미지를 하나씩 추출하여 예측하도록 구현되었다. 따라서 병목현상을 해소하기 위해, 각 이미지가 넘어갈때마다 타임스탬프를 넘겨주도록 구현하였고 이를 통해 타임스탬프를 확인하고 오래지나 처리하지 못한 이미지는 예측하지 않도록 구현하였다. 다만, 이 경우 올바른 Crop 이미지를 지나칠 수 있기 때문에 오히려 모델의 예측도를 감소하는 역효과를 보였다.

#### 4.3.2 전략 2) 잡음 판별모델 구축을 통한 병목현상 해소

올바른 Crop 이미지를 제대로 판별하기 위해 잡음판별모델을 별도로 구축하였다. 잡음 판별모델은 숫자 이미지를 확인하고 0 또는 1로 숫자인지, 잡음인지를 판별하는 모델로 프레임이 평균 60 프레임까지 나와 충분히 적용가능했다. 이에, 크롭된 이미지를 잡음판별 모델을 통해 사전적으로 숫자를 판별하도록 구현하여 병목현상을 해소하였다

### 5. 결론 및 향후 연구

실내용 자율 주행 카트를 개발함에 있어서, 이미 저장된 지도 정보가 아닌, 상황에 따라 유연한 대응을 하는 자율주행카트를 개발하고, 실내 환경에 최적화된 자율주행 시스템을 연구하였다. 최종적으로 근거리의 경우 센서를 활용한 Object Detection 과, 원거리의 경우 영상처리 및 머신러닝 모델에 의한 이동경로 예측 및 카트 자체 위치 파악을 구현하여 우발적이고 변화하는 환경에도 안정적인 주행을 기대할 수 있다. 또한 영상처리모듈의 경우 개발 초기에 비해 약 80%p의 정확도를 개선하여 안정적 주행을 구현하였다. 향후 연구로는 기기 제어 권한을 부여 및 승인할 수 있는 생체인식 기능과 하드웨어적 구현으로 카트의 전달 기능을 감안하여 보다 최적화된 기기를 구현할 계획이다.

### 6. 참고 문헌

- [1] 김용년, and 서일홍. "바닥 특징점을 사용하는 실내용 정밀 고속 자율 주행 로봇을 위한 싱글보드 컴퓨터 솔루션." 로봇학회논문지 14.4 (2019): 293-300.
- [2] 박세준, 서용호, and 양태규. "위치센서와 레이저거리센서를 이용한 실내용 이동로봇의 SLAM." 한국정보기술학회논문지 8.11 (2010): 61-69.
- [3] 노성우, 고낙용, 김태균, Noh, Sung-Woo, Ko, Nak-Yong, and Kim, Tae-Gyun. "위치 추정, 충돌 회피, 동작 계획이 융합된 이동 로봇의 자율주행 기술 구현." 한국전자통신학회 논문지 = The Journal of the Korea Institute of Electronic Communication Sciences 6.1 (2011): 148-56. Web.
- [4] 최정단, 민경욱, 성경복, 한승준, 이동진, 박상현, 강정규, 조용우, Choi, J.D., Min, K.W., Sung,K.B., Han, S.J., Lee, D.J., Park, S.H., Kang, J.G., and Jo, Y.W. "클라우드 연계 자율주행 맵 시스템 기술동향." 전자통신동향분석 = Electronics and Telecommunications Trends 32.4(2017): 40-47. Web.
- [5] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibraz, Sacha Arnoud, Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks, , Vinay Shet, 2011