Edge Computing 을 활용한 무인 마켓

Title	Edge Computing 을 활용한 무인 마켓
Abstract	본 문서는 Edge Computing 의 효율을 분석하기 위한 문서이다. 무인 마켓이라는 시나리오로 Cloud Computing 과 Edge Computing 을 비교 분석하는 연구에 대한 제안서 이다.

제출일 : 2019 년 6 월 25 일
지도교수 : 김 세 화 교수님

정보통신공학과	201601406	박 지 훈
정보통신공학과	201401059	류 형 오
정보통신공학과	201202192	유 정 훈

제목 : Edge Computing 을 활용한 무인 마켓

대 학 : 공 과 대 학
학 과 : 정보통신공학과

학 번 : 201601406
201401059
201202192

성 명 : 박 지 훈
류 형 오
유 정 훈

이 논문을 제출 하오니 승인하여 주십시오.

2019 년 6 월 25 일

성 명 : 박 지 훈 (인)

류 형 오 (인)

유 정 훈 (인)

위 학생의 논문 제출을 승인함.

2019 년 6 월 25 일

지 도 교 수 : 김 세 화 (인)

요 약

최근 마켓 관련 기업들이 각기 다른 기술력을 지닌 무인 마켓들을 시장에 내놓고 있다. 그 중에서 무인 마켓이라는 이름이 가장 걸 맞는 Amazon 사의 Amazon Go를 벤치마킹 하였다. 이때 요구되는 기술인 Object Tracking, Object Detection은 막대한 처리량을 요구하여 각 카메라 디바이스에서 처리하기에는 비용적인 문제가 있고 Cloud Computing을 활용하여 Cloud에서 처리 하기에는 여러 대의 카메라에서 실시간으로 영상 정보(frame)를 전송할 때 발생하는 교통 혼잡(traffic congestion) 문제가 발생한다.

본 논문에서는 Edge Computing 을 활용하여 Edge에서 로컬 네트워크상에 있는 각 디바이스로부터 데이터를 수집하여 처리하는 서비스를 제안한다.

(Edge Computing)

Edge Computing Platform 을 도입하여 갖게 되는 4가지 기능에 초점을 맞춘다. 첫 번째로 단말기 혹은 Cloud 에서 처리 하던 Task 들을 Edge Server로 Offloading 하여 쇼핑을 하던 고객들에게 Cloud Computing 환경보다 빠른 서비스를 제공한다. 두 번째로 물리적으로 분산 되어있는 센싱 장비들의 센싱 정보를 모두 Edge Server에서 수집후에 Sensor Fusion 하여 소비자 행동 분석 시 오차를 줄인 보다 정확한 파악을 가능하게 하였다. 세 번째로 소비자가 마켓 내부로 입장후에 이루어지는 모든 처리들은 마켓 자체 Edge Server 에서 처리 하도록 설계하여 마켓으로 들어오는 네트워크가 끊기거나 Cloud Server가 다운된다 해도 소비자는 여전히 서비스를 제공 받을 수 있는 Masking Outage 환경을 제공한다. 마지막으로 모든 촬영 정보들을 Edge Server 에서만 취급하여 외부로는 절대 내보내지 않고, 얼굴 인식을 위해 촬영되는 정보들은 blurring 처리 후에 30일간 Edge Server 에 보관하여 고객들의 Privacy도 고려하였다.

(Computer Vision)

본 논문에 사용되는 Computer Vision 기술은 소비자가 구매하려는 물건을 파악하기 위한 물체 인식 기술과 어떤 사용자가 구매하려 하는지를 파악하기 위한 얼굴 인식 기술이 있다. 본 논문에 사용된 물체 인식 기술은 실시간 물체 인식으로 가장 유명한 Faster-RCNN, RFCN, SSD 중 본 논문에서 가장 적합하다고 판단한 inception v2 모델을 활용한 Faster-RCNN 알고리즘을 차용하여 이를 Tensorflow 로 구현하였고, 얼굴 인식 기술은 Python에서 제공하는 Face Recognition, dlib 라이브러리를 활용하여 물체에서 얼굴을 먼저 인식한 후 특징들을 추출하여 저장되어 있는 이미지들과 비교하여 얼굴 인식을 할 수 있도록 하였다.

목차

그림 목차

1 서론

1.1 연구의 필요성

Cloud Computing은 시작된 이래 우리의 삶을 엄청나게 변화시켰다. 시스코 인터넷 비즈니스 솔루션 그룹이 예측한 대로 2020년까지 인터넷에 연결된 500억 개의 디바이스들이 있을 것이라고 예측했다.[1] 어떤 IoT Application은 응답 시간이 매우 짧을 수 있고, 어떤 Application은 개인 데이터를 포함할 수 있으며, 또 다른 application은 네트워크에 큰 부하가 될 수 있는 대량의 데이터를 생성할 수 있다. Cloud Computing은 이러한 Application을 지원할 만큼 효율적이지 않다.

위에서 말한 Application들을 효과적으로 사용 할 수 있도록 연구할 필요가 있다. 그것이 바로 Edge Computing이다. Edge Computing이란 데이터 처리 능력을 사용자들이 사용하는 단말장치(Terminal Device)들과 가까운 곳에 Computing Device를 위치 시키는 것이다.[2] Edge Computing의 첫 번째 장점은 바로 낮은 지연시간(latency)이다[3]. 사용자와 가까운 위치에서 서비스를 제공해주니 당연히 서비스의 속도가 빠르다, 라는 것이다. 물론 서비스의 속도가 빠른 이유는 Edge Computing이 기존의 데이터센터보다 성능이 우수해서가 아니라, 사용자에게 지리적으로 가까이 위치한 곳에서 서비스를 제공하기 때문이다. 지금도 인터넷은 충분히 빠르지만, 그 네트워크로 감당하지 못하는 분야가 존재한다. 예를 들어 자율주행 자동차는 수많은 데이터를 실시간으로 처리해주어야 하므로 지연시간(latency)가 짧아야 하고, VR/AR과 같은 분야는 60 ~ 120fps를 만족하지 않으면 어지럼증을 유발할 수 있다는 보고가 있다[4]. 즉, Edge Computing을 통해 이러한 요구 사항을 충족시킬 수 있기 때문에 Edge Computing의 가장 큰 수혜자는 360도 영상 전송 등을 포함하는 미디어 분야, 스마트 팩토리 같은 IoT분야, 자율주행 자동차 분야가 될 수 있을 것이다. 두 번째 장점은 네트워크 트래픽 감소이다.[3] 이는 각종 IoT 디바이스, 예컨대 스마트카에서 생산되는 데이터의 크기가 수십 GB임에도 불구하고, 굳이 데이터를Cloud로 전송할 필요가 있는가, 에 대한 이야기이다. 즉, 수집되는 데이터를 Cloud Server로 전송하지 않고, Edge Computing의 Local Process를 통해 Cloud로 통하는 네트워크 트래픽을 감소시키는 것을 목적으로 한다.

1.2 연구 주제

Edge Computing의 효율을 알아보기 위해 Amazon사의 Amazon Go를 벤치마킹한 무인 마켓을 구현한다. 이때 요구되는 기술인 Face Recognition, Object Detection등 막대한 처리량을 요구하는 기술들을 Edge Device에서 처리하는 것과 Cloud에서 처리하는 두 가지 방식으로 구현하여 서로의 성능을 분석하여 Edge Computing의 효율을 검증하고자 한다.

1.3 연구구성 및 역할분담

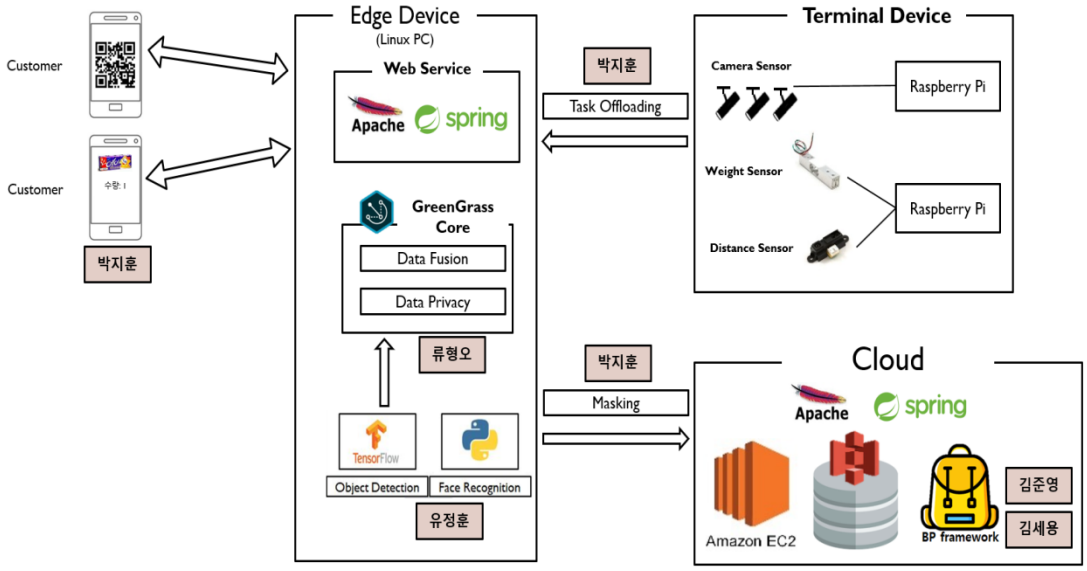


Figure 1 HUFS GO 구성 및 역할 분담

2 배경 기반 기술

2.1 Edge Computing 기술 소개

2.1.1 개념

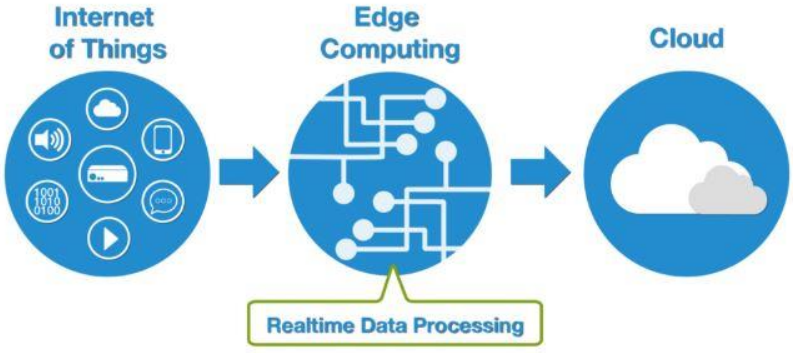


Figure 2 Edge Computing 의 구조 (출처: Byline Network, 2018)

Edge Computing 기술은 데이터 처리능력이 Cloud에 있는 Cloud Computing과는 다르게 데이터 처리 능력을 사용자들이 사용하는 단말 장치(Terminal Device)들과 가까운 곳에 Computing Device를 위치 시키는 것이다. 여기서 착각을 하면 안 되는 부분이 있다. 바로 Edge Computing은 Cloud Computing을 대체하는 기술이 아닌 Cloud Computing과 공존하며 Cloud Computing을 발전 시킨 것이라는 점이다.

2.1.2 장점 및 사용이유

Edge Computing을 사용하는 이유는 크게 4가지로 나눌 수 있다.

첫 번째 장점은 Highly Responsive Cloud Services 이다.

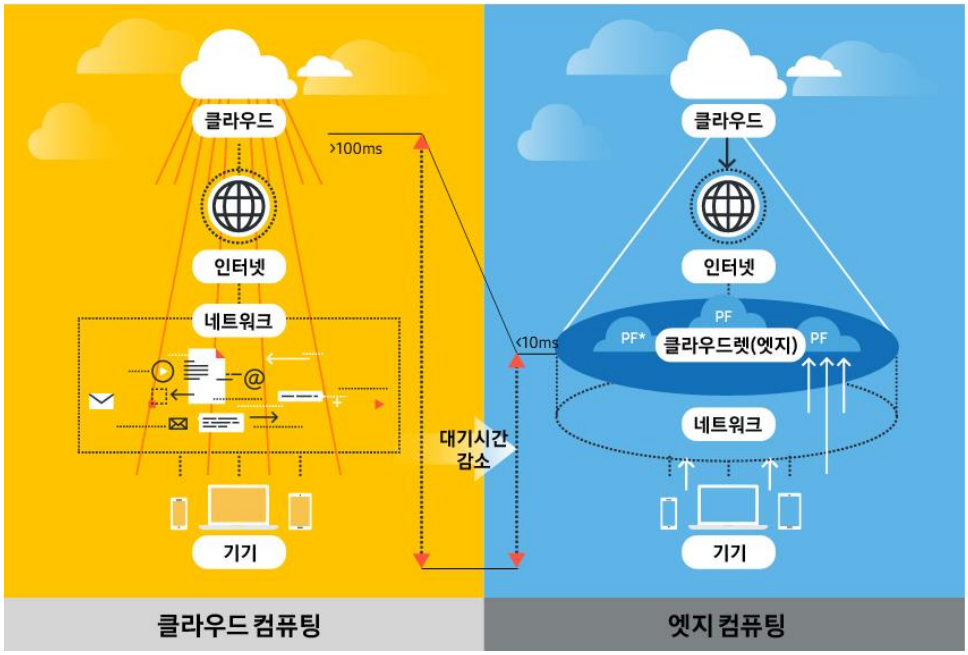


Figure 3 Cloud Computing 과 Edge Computing 구조 (출처: Samsung Newsroom)

Figure 3은 Cloud Computing과 Edge Computing의 구조를 나타내어 주는 것으로 Edge Computing은 사용자의 Terminal Device와 가까운 곳에 Edge Device를 둠으로써 Offloading을 통하여 거리상 멀리 있는 Cloud 보다 데이터 처리 대기시간이 빠른 것을 볼 수 있다. 이러한 장점 덕에 실시간 어플리케이션 응용에 유용하게 쓰일 수 있다.

두 번째 장점은 Scalability Through Edge Analytics 이다. High data rate sensor(Video camera, Senosrs of aircraft, etc ..)들로부터 Edge Device로 Offloading을 통해 sensing 값들을 전달 받아 Fusion 시킴으로써 실시간 위험으로부터 대처할 수도 있고 하나의 Terminal Device에서의 정보로만은 알 수 없는 정보들을 도출해낼 수가 있도록 확장 할 수 있다는 점에서 좋다.

세 번째 장점은 Privacy Policy Enforcement 이다.

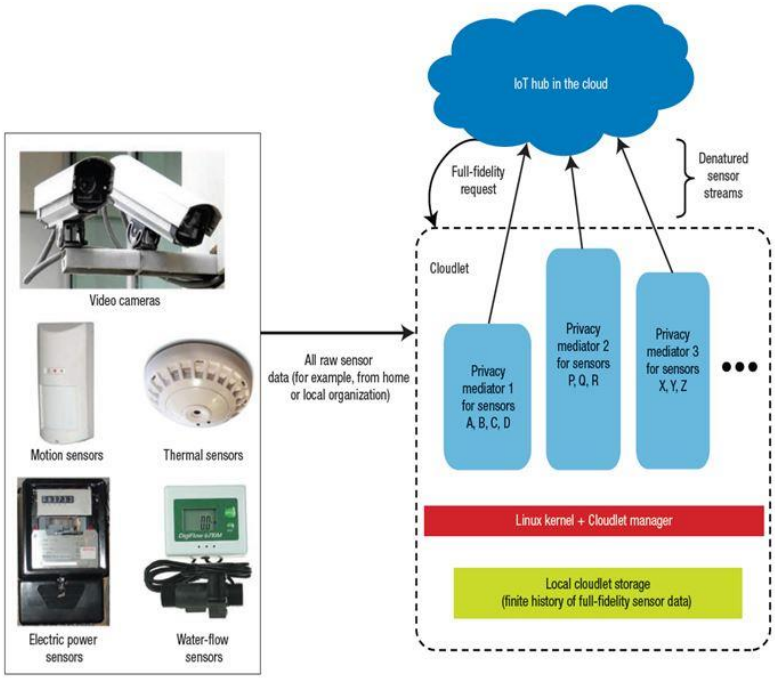


Figure 4 IoT privacy architecture.[5]

Figure 4 는 IoT 보안 구조를 나타내고 있다. Edge 와 연결된 모든 Sensor 로부터 받은 data 들을 Edge 에서 특정시간 동안(몇 시간,일,주,달, etc …)보관해두고 그 시간 동안 Cloud 가 필요로 하는 정보만을 제공하고 시간이 지나가면 Edge 에서도 삭제 함으로써 Cloud 로 모든 Data 를 보내지 않고 Local 에서만 data 를 처리하게 되므로 보안측면에서도 우수 하다고 할 수 있다.

네 번째 장점은 Masking Cloud Outages 즉 Cloud 중단 혹은 네트워크 불량 시에도 시스템을 활용 할 수 있다는 점이다. Cloud Computing 은 Cyber Attack 이나 자연재해 등등 특수한 이유로 Cloud 가 작동 불가상태일 경우에 어떠한 시스템도 정상적으로 작동 할 수 없지만 Edge Computing 은 Cloud 가 작동 불가상태일 경우에도 Edge 자체적으로 처리할 수 있는 능력이 있기 때문에 시스템을 정상적으로 활용할 수 있다는 점에서 유용 하다고 할 수 있다.

2.2 Computer Vision 소개

Computer vision 은 인공지능 (Artificial Intelligence) 의 한 분야로서, 어떤 영상에서 장면이나 특징 (scene or features) 들을 "이해 (Understanding)" 하는 컴퓨터를 프로그램하는 것이 목적이다. 컴퓨터비전의 일반적인 목표는 다음과 같다 :

- 영상에서 물체의 detection, segmentation, location, recognition (예를 들면 인간의 얼굴인식 (Face Recognition))
- 결과의 평가 (예를 들면 segmentation, registration)
- 같은 장면이나 물체에 대한 다른 관점 (view) 의 등록 (registration)
- 연속 영상에서 물체를 추적
- 어떤 장면을 3 차원 모델로 mapping ; 그 모델은 영상화된 장면을 돌아다니는 로봇에 의해 사용된다
- 인간의 자세와 팔다리 움직임을 3 차원으로 추정 (estimation)
- 콘텐츠에 따라 디지털 영상을 탐색 (content-based image retrieval)

이러한 목표들은 패턴인식 (Pattern Recognition), statistical learning, projective geometry, image processing, 그래프 이론 (Graph Theory) 등등에 의해 성취될 수 있다. Cognitive computer vision 은 인지 심리학 (Cognitive Psychology), biological computation 과 밀접하게 관련되어있다.

관련되는 machine vision 과 medical imaging 분야에서, 컴퓨터비전 기술을 사용한 시스템들은 매년 수십억 달러의 시장을 형성하고 있다. 흥미로운 응용분야는 영화와 방송, 즉 camera tracking 또는 matchmoving 으로 visual effects 를 만드는데 흔히 사용된다는 것이다. 컴퓨터비전은 의학, 군사, 보안과 감시 (surveillance), 품질검사 (quality inspection), 로봇, 자동차산업 등등 여러 분야에 응용된다

3 관련 연구

3.1 AWS Greengrass

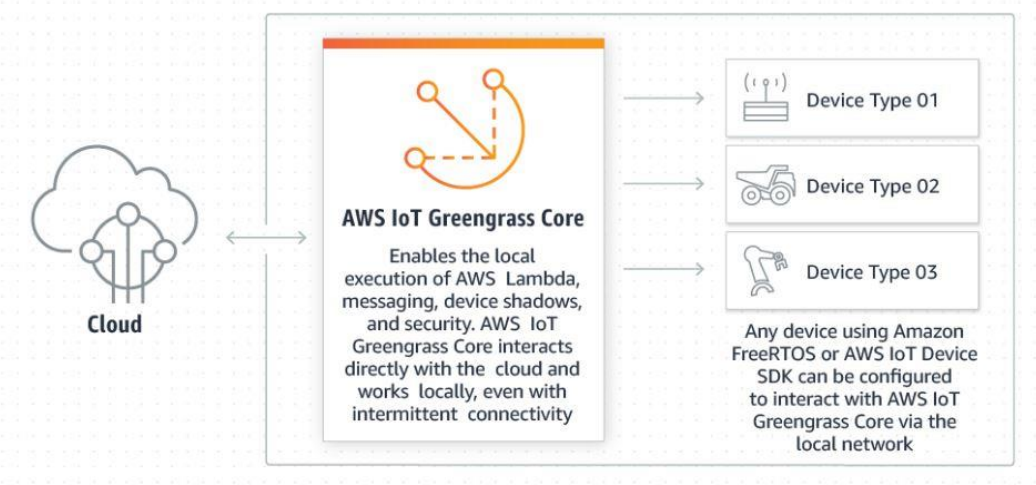


Figure 5 AWS Greengrass architecture(출처: AWS Greengrass 홈페이지)

AWS Greengrass 는 AWS Cloud 기능을 로컬 디바이스로 확장하는 소프트웨어이다. 로컬 디바이스에서 데이터를 수집 분석하고, 로컬 네트워크에서 있는 디바이스들이 서로 안전하게 통신할 수 있도록 하는 역할이다. Figure 5 처럼 AWS Greengrass core 를 AWS Greengrass Cloud 에서 생성하여 Edge Device 로 배포하여 사용된다.

3.1.1 AWS Greengrass core

AWS Greengrass core 는 소프트웨어를 실행하는 AWS IoT 디바이스로, AWS IoT 및 AWS IOT Greengrass 클라우드 서비스와 직접 통신하는데 사용된다.

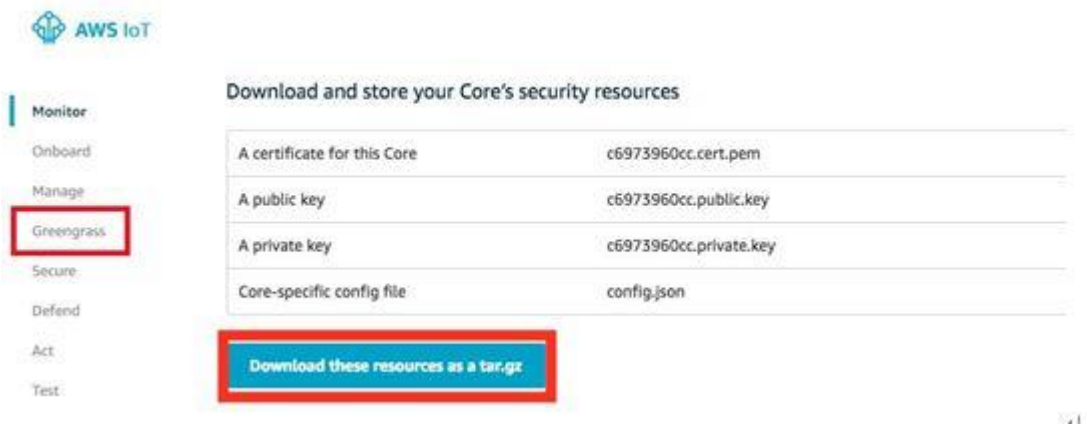


Figure 6 AWS Greengrass core 생성(출처: AWS Greengrass 홈페이지)

Figure 6 처럼 AWS Greengrass Cloud 에 들어가서 Group 을 생성하고 Core 를 생성하게 되면 Core 의 보안 리소스가 생성된다. 이 보안 리소스를 Edge Device 에 저장 시킴으로써 Cloud 와 Edge Device 가 안전하게 연결 할 수 있게 된다.

3.1.2 AWS Greengrass device

AWS Greengrass Device 는 AWS IoT Device SDK 를 사용하여 Core 에 대한 연결 정보를 가져온다.

Figure 7 AWS Greengrass device 생성(출처: AWS Greengrass 홈페이지)

Figure 7 처럼 AWS Greengrass Cloud 에 들어가서 Device 를 생성하고 등록을 하면해당 Device 의 보안 리소스가 생성이 되고 이 보안 리소스를 Edge Device 에 저장 시킴으로써 Edge 에 등록되어있는 보안 리소스를 가지고 있는 Device 만 접근할 수 있도록 해준다.

3.1.3 AWS Greengrass Lambda Function

AWS Lambda Function 은 서버를 프로비저닝 하거나 관리하지 않고도 코드를 실행할 수 있게 해주는 컴퓨팅 서비스이다. AWS Lambda Function 필요 시에만 코드를 실행하며, 하루에 몇 개의 요청에서 초당 수천 개의 요청까지 자동으로 확장이 가능하다.

코딩을 통하여 Lambda Function 을 계속 실행 시킬 수도 있고 필요 시에만 작동 시킬 수도 있다. 필요 시에 사용하려면 MQTT 프로토콜로 Event 를 발생시켜주면 Event 가 발생한 시점에만 실행이 진행된다.

3.1.4 AWS Greengrass architecture Sequence Diagram

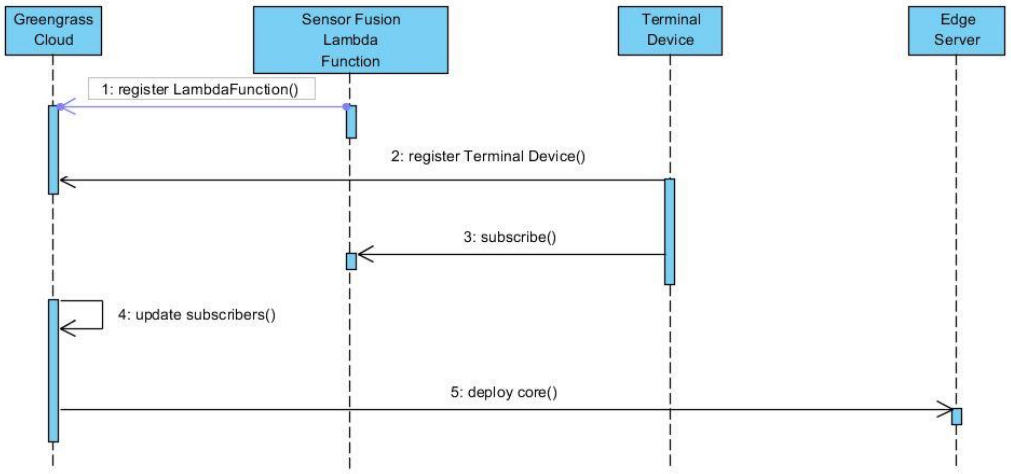


Figure 8 AWS Greengrass architecture Sequence Diagram

Greengrass 설정에 대한 Sequence Diagram 을 나타낸다. 처음에 Terminal Device 에서 실행시킬 lambda 함수를 생성 후 Terminal Device 와 같이 Greengrass core 에 등록해준다. 그 다음 Terminal Device 와 lambda 함수간의 구독을 정의해주고 core 는 구독자를 업데이트 시킨 후 Edge 로 배포를 하여 Terminal Device 와 Edge 가 MQTT 프로토콜을 사용하여 통신 할 수 있게 된다.

3.2 Faster-RCNN

Faster R-CNN 은 Fast R-CNN 을 개선하기 위해 Region Proposal Network (RPN)을 도입한다. RPN 은 region proposal 을 만들기 위한 network 이다. 즉, Faster R-CNN 에서는 외부의 느린 selective search (CPU 로 계산)대신, 내분의 빠른 RPN(GPU 로 계산)을 사용한다. RPN 은 마지막 convolutional layer 다음에 위치하고, 그 뒤에 Fast R-CNN 과 마찬가지로 ROI pooling 과 classifier, bounding box regressor 가 위치한다. RPN 은 또한 image 를 입력받아 사각형 형태의 Object Proposal 과 Objectness Score 를 출력 해주는 역할을 한다. 이는 Fully convolutional network 형태이며, Fast R-CNN 과 convolutional layers 를 공유하게끔 디자인되어 있다. 다음은 주요 특징들이다.

3.2.1 Anchor Box

Anchor box 는 sliding window 의 각 위치에서 Bounding Box 의 후보로 사용되는 상자다. 이는 기존에 두루 사용되던 Image/Feature pyramids 와 Multiple-scaled sliding window 와 다음과 같은 차이를 보인다.

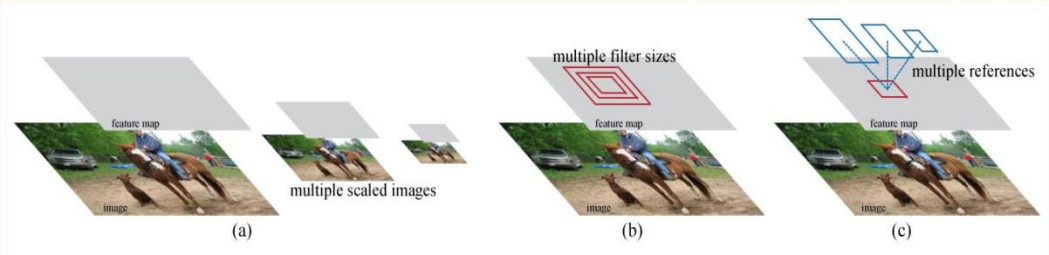


Figure 9 Different schemes for addressing multiple scales and sizes

직관적으로 설명하자면, 동일한 크기의 sliding window 를 이동시키며 window 의 위치를 중심으로 사전에 정의된 다양한 비율/크기의 anchor box 들을 적용하여 feature 를 추출하는 것이다. 이는 image/feature pyramids 처럼 image 크기를 조정할 필요가 없으며, multiple-scaled sliding window 처럼 filter 크기를 변경할 필요도 없으므로 계산효율이 높은 방식이라 할 수 있다. 논문에서는 3 가지 크기와 3 가지 비율의, 총 9 개의 anchor box 들을 사용하였다.

Labeling to each anchor (Object? or Backgound?)

특정 anchor 에 positive label 이 할당되는 데에는 다음과 같은 기준이 있다.

1. 가장 높은 Intersection-over-Union(IoU)을 가지고 있는 anchor.

2. IoU > 0.7 을 만족하는 anchor.

2 번 기준만으로는 아주 드물게 Object 를 잡아내지 못하는 경우가 있어서 후에 1 번 기준이 추가되었다.

반면, IoU 가 0.3 보다 낮은 anchor 에 대해선 non-positive anchor 로 간주한다

3.2.2 Computer Process

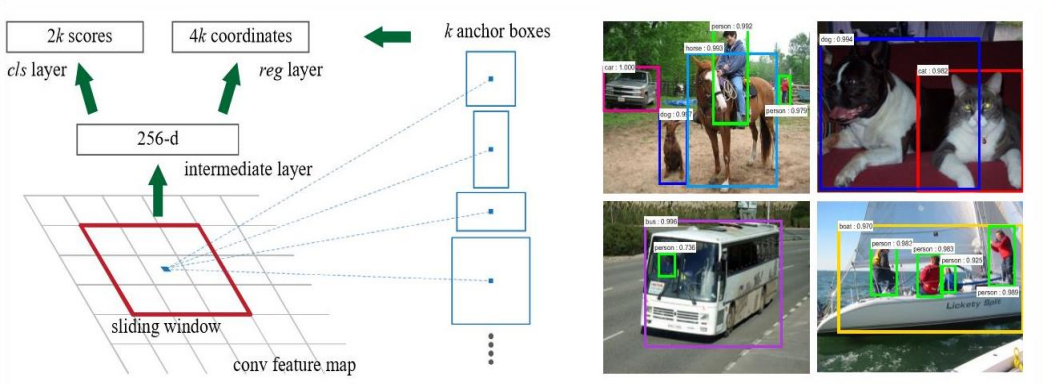


Figure 10 LEFT: Region Proposal Network(RPN). Right: Example detec tions using RPN proposals on PASCAL VOC 2007 test.

1. Shared CNN 에서 convolutional feature map(14X14X512 for VGG)을 입력받는다. 여기서는 Shared CNN 으로 VGG 가 사용되었다고 가정한다. (Figure3 는 ZF Net 의 예시 - 256d)

2. Intermediate Layer: 3X3 filter with 1 stride and 1 padding 을 512 개 적용하여 14X14X512 의 아웃풋을 얻는다.

3. Output layer

- cls layer: 1X1 filter with 1 stride and 0 padding 을 9*2(=18)개 적용하여 14X14X9X2 의 이웃풋을 얻는다. 여기서 filter 의 개수는, anchor box 의 개수(9 개) * score 의 개수(2 개: object? / non-object?)로 결정된다.

- reg layer: 1X1 filter with 1 stride and 0 padding 을 9*4(=36)개 적용하여 14X14X9X4 의 아웃풋을 얻는다. 여기서 filter 의 개수는, anchor box 의 개수(9 개) * 각 box 의 좌표 표시를 위한 데이터의 개수(4 개: dx, dy, w, h)로 결정된다.

  주목할 점은, output layer 에서 사용되는 파라미터의 개수다. VGG-16 을 기준으로 했을때 약 2.8 X 10^4 개의 파라미터를 갖게 되는데(512 X (4+2) X 9), 다른 모델들의 output layer 파라미터 개수 -가령, GoogleNet in MultiBox 의 경우 약 6.1 X 10^6- 보다 훨씬 적은 것을 알 수 있다. 이를 통해 small dataset 에 대한 overfitting 의 위험도가 상대적으로 낮으리라 예상할 수 있다.

3.2.3 Loss Function

Loss Function 은 아래와 같다.

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*). \quad (1)$$

- pi: Predicted probability of anchor
- pi*: Ground-truth label (1: anchor is positive, 0: anchor is negative)
- lambda: Balancing parameter. Ncls 와 Nreg 차이로 발생하는 불균형을 방지하기 위해 사용된다. cls 에 대한 mini-batch 의 크기가 256(=Ncls)이고, 이미지 내부에서 사용된 모든 anchor 의 location 이 약 2,400(=Nreg)라 하면 lamda 값은 10 정도로 설정한다.
- ti: Predicted Bounding box
- ti*: Ground-truth box

Faster R-CNN 에서 Feature map 을 추출하는 과정에서 Google 에서 제공하는 inception v2 모델을 사용해 Feature map 을 추출했다.

4 Market Scenario 설계 Diagram 및 한계점

4.1 Market Scenario

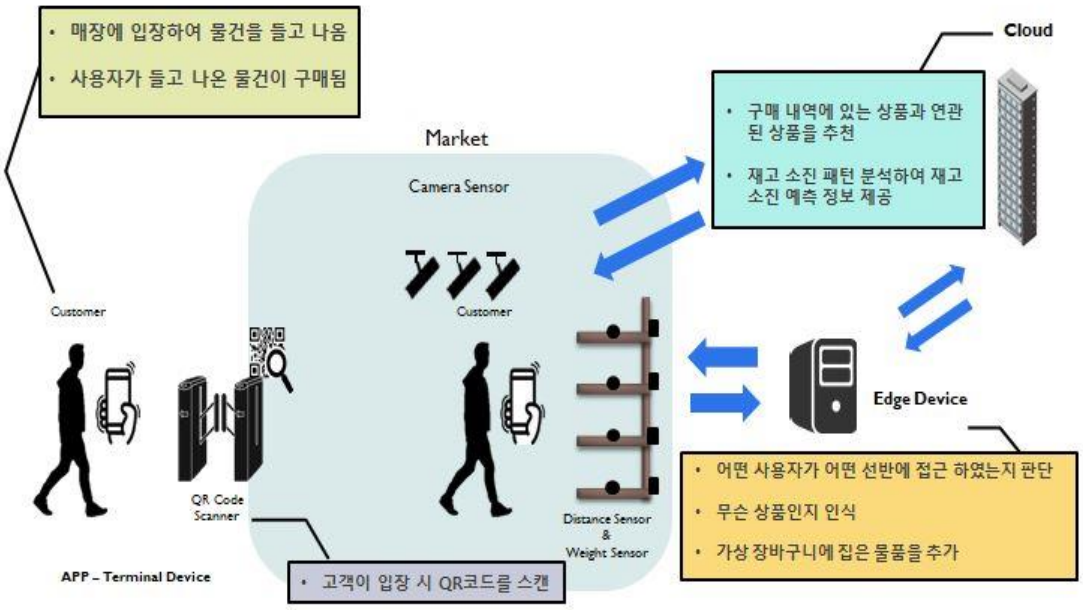


Figure 11 HUFS GO Market Scenario

4.2 Deployment Diagram

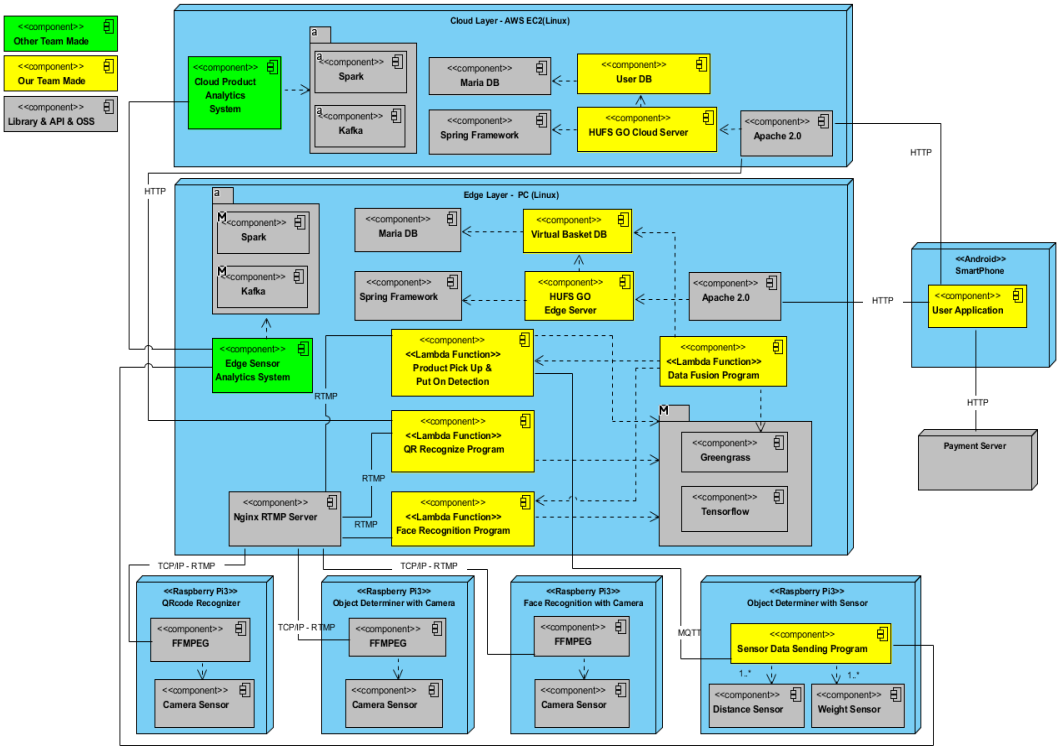


Figure 12 HUFS GO Deployment Diagram

4.3 Activity Diagram for Register

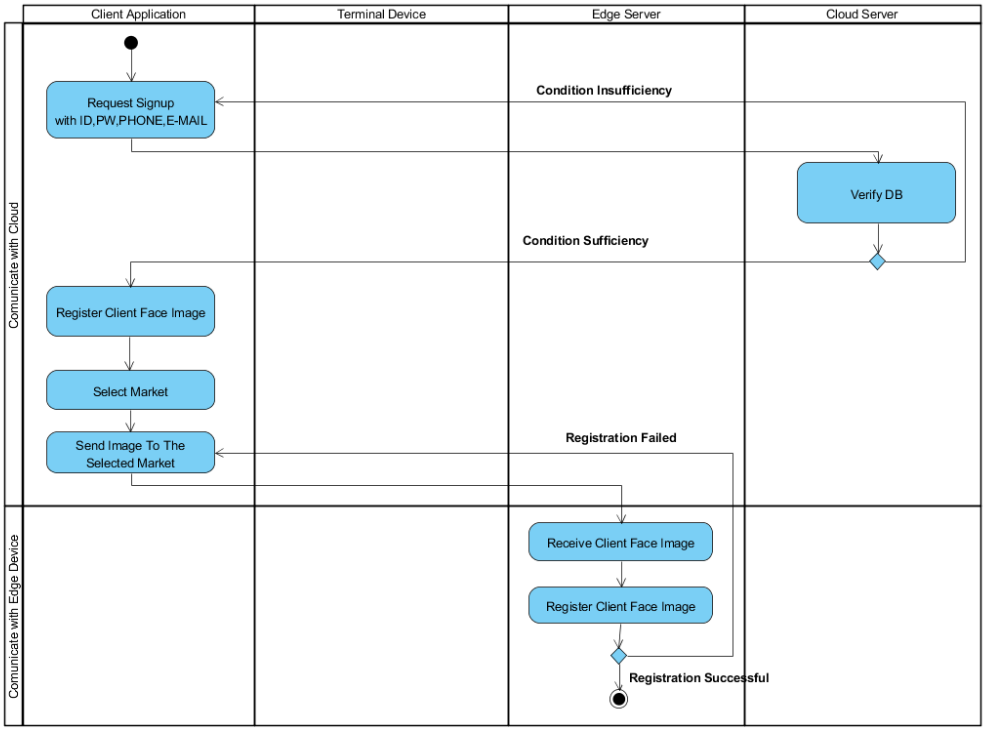


Figure 13 HUFS GO Activity Diagram for Register

4.4 Activity Diagram for Entrance

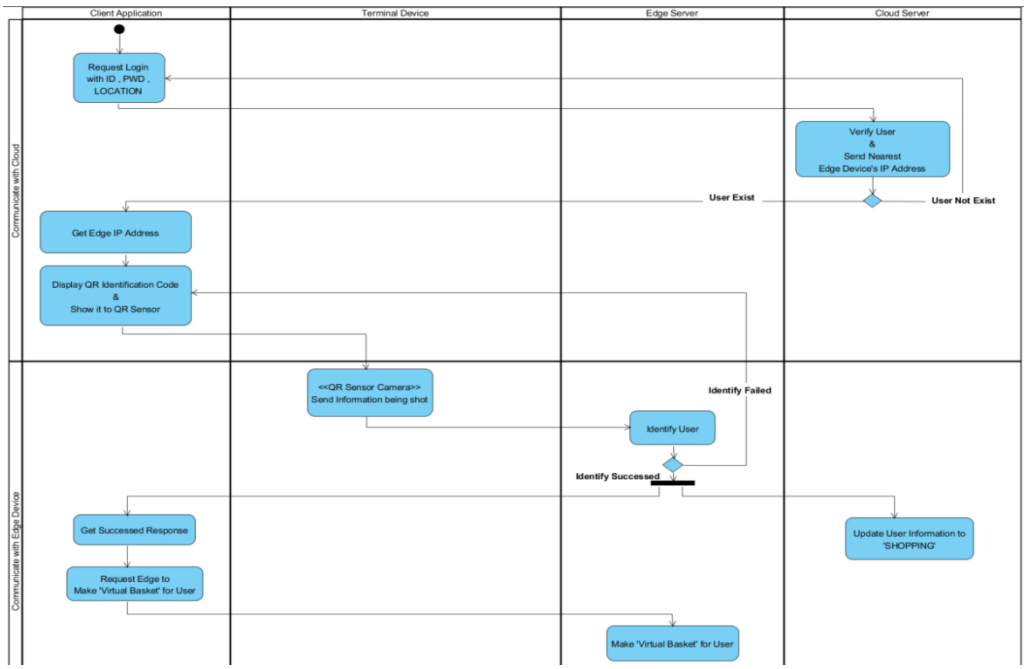


Figure 14 HUFS GO Activity Diagram for Entrance

4.5 Activity Diagram for Shopping

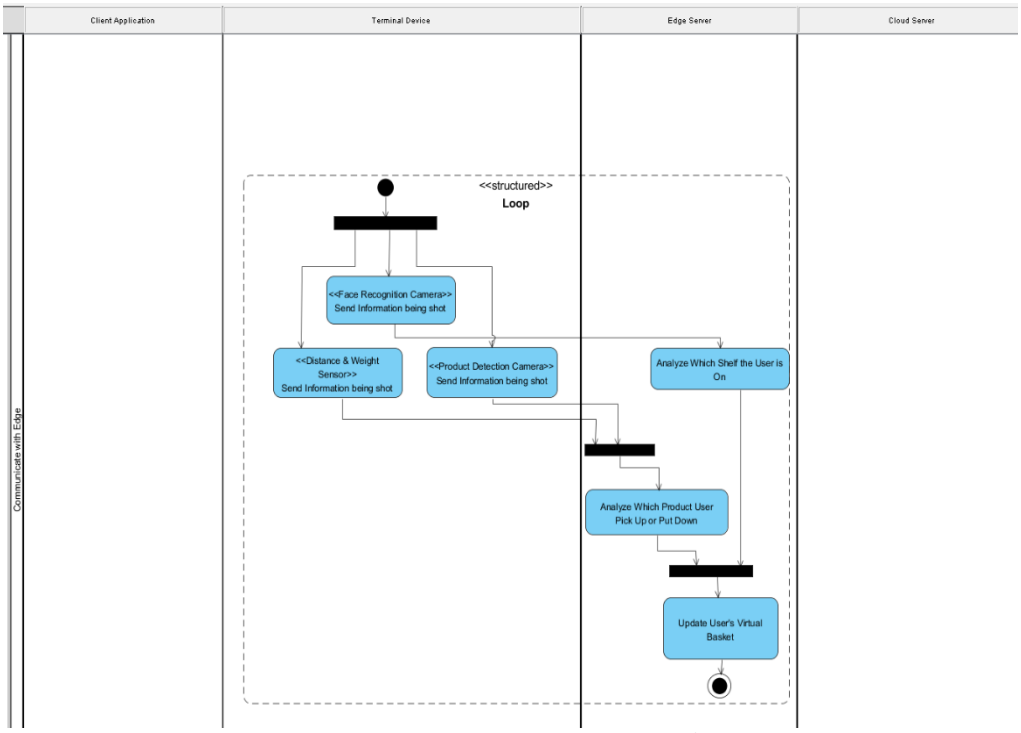


Figure 15 HUFS GO Activity Diagram for Shopping

4.6 Activity Diagram for Purchase

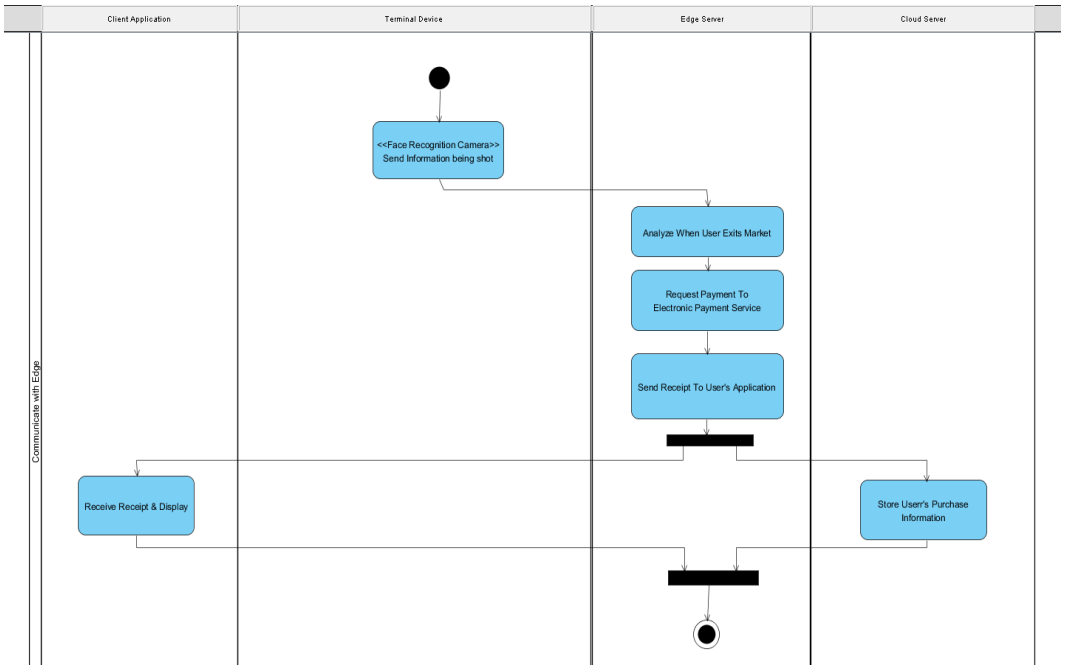


Figure 16 HUFS GO Activity Diagram for Purchase

4.7 전제조건

HUFS GO 는 이미 회원가입을 통하여 고객이 원하는 마켓 Edge 에 고객들의 얼굴 사진이 들어가 있다고 가정한다.

4.8 제약사항

- 한 매대당 한 사람만 있어야 한다.
- 상품은 제자리에서 집고 놓아야 한다.
- 상품을 1 개만 집고 놓아야 한다.
- 고객이 얼굴인식에 방해되는 물품을 착용하고 있으면 안 된다.

5 Offloading 및 Masking Outage

5.1 Offloading

Offloading 이란 Terminal Device 에서 하기에는 무거운 계산을 서버로 전달하여 서버에서 계산을 한 후 결과만을 Terminal Device 에게 알려주는 기술이다.

이 Offloading 에는 Terminal Device 에서 특정정보만 보내는 Partial Offloading 과 모든 정보를 서버로 보내는 Full Offloading 이 존재한다. 우리는 이중 Full Offloading 을 채택하였다. 그 이유는 다음과 같다. 본 연구에서는 매대당 존재하는 거리 센서, 무게 센서를 측정해주는 Terminal Device, 해당 물건이 무엇인지 판단하는 Object Detection Camera 센서 Terminal Device, 물건을 집는 고객이 누군지 판단하는 Face Recognition 센서 Terminal Device 이렇게 총 3 개의 Terminal Device 가 존재한다. 이 3 개의 Terminal Device 로부터 Offloading 을 진행하게 되는데 Partial Offloading 방법을 채택하게 되면 3 개의 Terminal Device 끼리 정보를 공유하여 특정 정보는 Edge Server 로 Offloading 하고 Terminal Device 에서 처리할 수 있는 계산은 Terminal Device 에서 처리하도록 하는 알고리즘을 짜주면 된다. 그렇지만 우리는 3 개의 Terminal Device 끼리 정보를 공유하는 것이 어렵다고 판단하였다. 그러하여 본 연구에서는 Full Offloading 방식을 채택하였다.

Terminal Device 인 라즈베리파이에서 촬영한 영상을 RTMP 기술을 활용하여 Edge Server 로 보내주게 되고 Edge Server 는 라즈베리파이에서 받은 영상을 바탕으로 무거운 계산인 Machine Learning 을 하여 얼굴인식, 물체인식, QR Code 인식 등을 진행하게 된다.

5.2 Masking Outage

Masking 이란 Edge Computing 에서 일컫기를 Cloud Server 가 breaks down 되었을 때에도 Edge 에서는 평소에 진행하던 작업을 계속 진행하다가 다시 연결이 되었을 때

breaks down 되었을 동안 저장해 두었던 데이터들을 Cloud Server 에 전송해주는 것을 말한다.

본 연구의 초기 단계에는 고객이 쇼핑하는 장바구니 DB 를 Cloud Server 에 위치 시켜놓고 마켓으로 들어오는 네트워크가 끊기거나 Cloud 와의 네트워크가 끊겼을 때 마켓에서 쇼핑하고 있는 고객은 쇼핑을 진행하면 자신의 장바구니에는 업데이트는 안되지만 쇼핑 정보들을 Edge Server 에서 가지고 있다가 네트워크가 다시 연결 되었을 때 쇼핑정보를 Cloud 로 보냄으로써 Masking Outage 기술을 구현 하였다. 그렇지만 이러한 방식은 네트워크가 다시 연결 될 때까지 기다려야 진행이 된다는 단점이 있었다.

그래서 HUFS GO 마켓은 입장한 후로부터는 모든 통신은 Edge Server 와 진행이 되도록 고객이 쇼핑하는 장바구니 DB 를 Edge Server 에 위치 시켰다. 때문에 마켓으로 들어오는 네트워크나 Cloud Server 가 breaks down 되어도 입장한 고객은 평소와 다름없이 쇼핑을 계속 진행할 수 있고 결제까지도 가능하다. 그렇지만 새롭게 입장은 불가능하다. 이러한 고객들의 결제정보는 Edge Server 에서 가지고 있다가 Cloud Server 에 전송해줌으로써 Masking Outage 기술을 구현하였다.

5.3 Privacy

HUFS GO 마켓에서는 Object Detection, Face Recognition 기술을 구현하기 위해 Camera 센서를 사용하여 마켓 운영시간 동안 영상을 저장하게 되는데 이때 모든 영상 정보는 Cloud Server 로는 전송하지 않고 Edge Server 자체에서만 30 일간 저장해 두었다가 기간이 지나면 삭제 된다. 추가적으로 Face Recognition 기술을 구현하기 위한 Camera 센서에서 영상을 저장을 할 때는 고객이 누군지 판별을 하면서 고객의 얼굴을 Blurring 을 하여 저장을 함으로써 고객들의 Privacy 를 지켜 주었다.

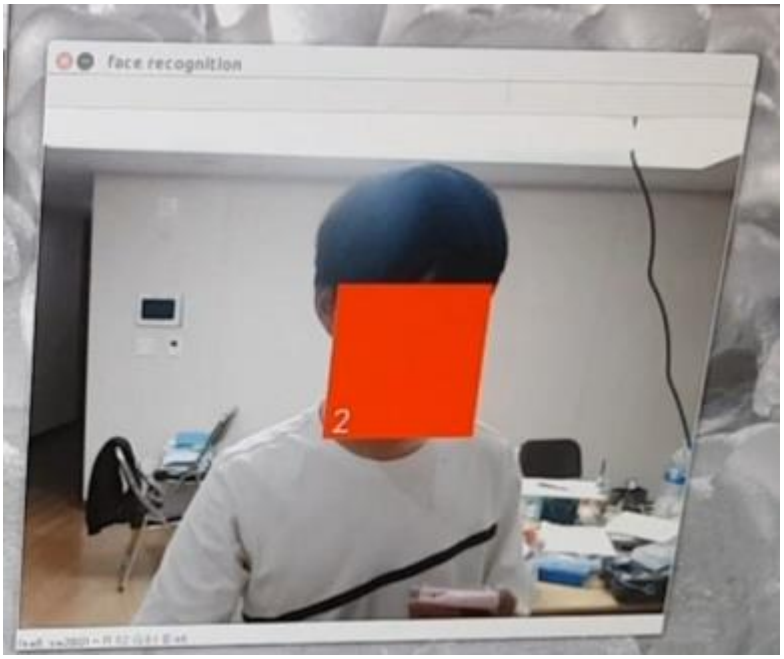


Figure 17 Image Blurring

6 Sensor Fusion

6.1 Sensor Fusion

Sensor Fusion 이란 여러 가지 Sensor 들로부터 정보를 받아 종합하여 하나의 결과를 도출시키는 것으로써 여러 가지 Sensor 들의 값을 조합하여 하나의 정확한 결과값을 도출한다.

HUFS GO 에서는 Camera Sensor, Distance Sensor, Weight Sensor 를 사용하여 Fusion 하여 결론을 도출하였다. Weight, Distance, Camera Sensor 중 2 가지 이상의 Sensor 가 동일한 물건이라고 판단하였을 때 결론을 도출하였다. 마지막으로 고객의 얼굴을 인식하고 있는 Camera 센서와 Sensor Fusion 으로 도출된 물품을 Processing 하여 상품을 집은 고객 어플 장바구니에 업데이트 시킨다.

6.2 Exception Handling

일반적인 경우 고객은 매대의 상품을 맨 앞부터 집어가겠지만 예외상황으로 뒤의 상품을 집어가거나 놓았을 경우도 Object Detection 중인 Camera Sensor 와 Weight Sensor 의 Sensor Fusion 을 통하여 해결 하였다.

6.3 Our Smooth Filter

Weight, Distance Sensor 값들이 불안정하여 가끔 비정상적으로 튈 때를 대비하여 Threshold(임계치)값을 지정해두고 최근 N 개의 평균을 계속 계산하여 최근 N 개의 평균이 Threshold 값을 넘으면 값이 튀어 비정상적으로 큰 값이 나왔다고 판단을 하여 그 값을 무시하고 계속 Sensing 을 진행하게 함으로써 Weight, Distance Sensor 의 불안정한 값을 잡아 주었다.

if $\left|\frac{\left(\sum_{i=0}^{N-1} L[i]\right)}{N}\right| \leq T$ ⟹ 정상적인 Sensor값 으로 판단

else ⟹ 비정상적으로 Sensor값으로 판단하여 판단 무시

(L = [x1,x2.... xn] 센서값들이 저장되는 1xN 행렬
T = Threshold)

6.4 Sequence Diagram

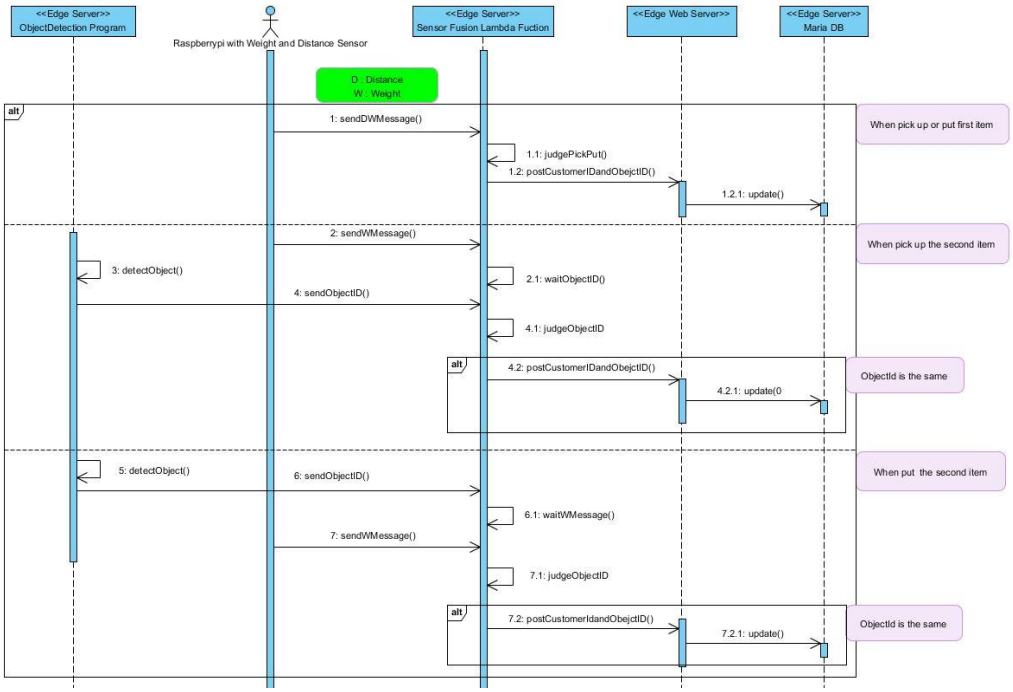


Figure 18 Sensor Fusion Sequence Diagram

7 Face Recognition 및 Object Detection

7.1 Face Recognition

Face Recognition 은 디지털 이미지를 통해 각 사람을 자동으로 식별하는 컴퓨터 지원 응용 프로그램이다.

본 논문에서는 Python 에서 제공하는 Face Recognition 라이브러리를 사용하여 진행하였다. 먼저 HOG 알고리즘을 사용하여 사진을 encoding 한 후 얼굴에 존재하는 68 개의 landmark 포인트를 생성한다. 특징들을 측정하는 방법을 학습한 모델을 통해 128 개의 측정값을 저장하여 과거에 측정해 놓은 얼굴에 대해, 이 얼굴의 측정값에 가장 가까운 사람이 누구인지 판단한다.

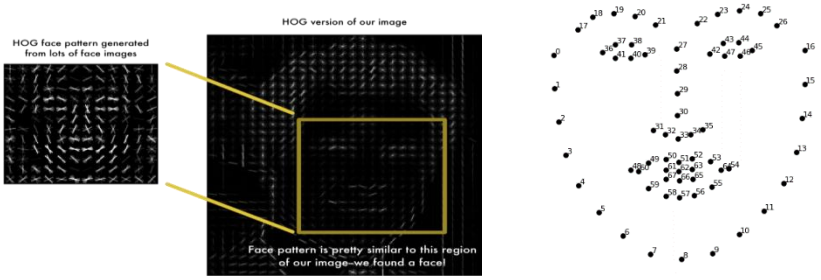


Figure 19 HOG 알고리즘 및 landmark 알고리즘

Figure 20 Face Recognition 진행 과정

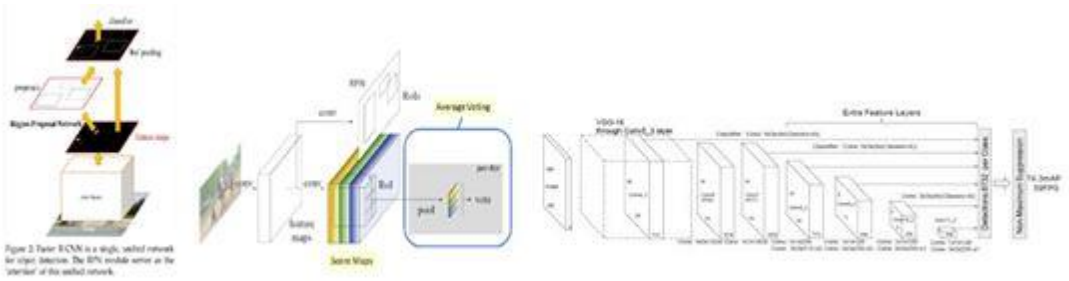
7.2 Object Detection

객체 검출에서 딥러닝 알고리즘 개발은 활발히 진행중이다. 그 중 본 연구를 시작한 시점에서 가장 많이 사용되고 있는 세 가지의 객체 분석 알고리즘인 Faster R-CNN, RFCN, Single shot Multibox Detector 을 비교 분석하였다.

먼저 Figure. 1(a)에서 Faster R-CNN 은 CNN 기반의 구조로서, Convolution layer 를 통해 이미지의 특징을 Feature map 으로 나타내고, 이 Feature map 에서 특징이 검출되는 부분에 Region Proposal Network 를 진행하여 최종적으로 객체를 검출한다.

RFCN 역시 CNN 기반의 구조로서 Faster R-CNN 과 달리 Convolution layer 에서 RPN 과 ROI (관심 영역)을 통해 Feature map 을 나타내고 이 특징을 바탕으로 Score Maps 를 생성 최종적으로 객체를 검출한다.

Single Shot Multibox Detector 또한 CNN 구조 알고리즘으로 구성되어 있다. 입력 이미지에서 Convolution layer 를 거쳐 각 층에서 나온 Feature map 을 사용하여, 여러 개의 Feature map 에서 다양한 크기와 모양의 객체를 검출할 수 있게 한다.



(a) Faster R-CNN (b) RFCN (c)Single Shot Multibox

Figure 21 딥러닝의 다양한 객체 검출 알고리즘

7.3 딥러닝 네트워크 모델

다양한 딥러닝 객체 검출 알고리즘 중에 프로젝트에 맞는 알고리즘을 선택하기 위해 Table. 16 에서의 정확도와 속도 비교를 참고하였다. 또한, 실시간으로 정확한 객체 탐지를 위해선 3 가지 모델을 테스트 하여 결정하기로 하였다.

Model	Map	Fps	Batch size	Input resolution
FasterR-CNN_inception v2	73.2	3.7	1	1280 × 720
RFCN_Resnet101	76.6	4.9	1	1280 × 720
SSD_Mobilenet v1	75.1	15.	1	1280 × 720

Figure 22 Face Recognition Faster R-CNN, RFCN, SSD 의 성능 비교

테스트 결과

테스트는 OS: [Linux] Ubuntu 16.04, GPU: RTX2070, RAM: 16GB, 230 장의 image(504 × 378)와 약 20000 steps 의 Learning 을 기준으로 했으며, Table. 1 의 수치와 실제 web cam 을 통한 테스트와 차이를 보였다. Fps 는 SSD > Faster R-CNN > RFCN 순이 이었으며, Detecting 률은 RFCN >= Faster R-CNN > SSD 순 이였지만 RFCN 으로 시연 하기에는 많은 GPU 를 할당 해야 되기 때문에 Faster R-CNN 이 적합하다고 판단 했다.

8 결과

8.1 비교환경

(a) ~ (b): Single Camera Environment

(a): Face Recognition

- 사용자가 얼굴을 카메라에 비추고 응답을 받기까지의 시간

(b): QR Code Recognition

- 사용자가 QR Code를 카메라에 비추고 응답을 받기까지의 시간

(c): Sensor Fusion

- 사용자가 물건을 집고(사용자 식별 없이)집었다는 사실을 확인 하기까지의 시간

(d): Entire Service (Visualize Virtual Basket Update)

- 사용자가 물건을 집고 이를 확인 하기까지의 시간

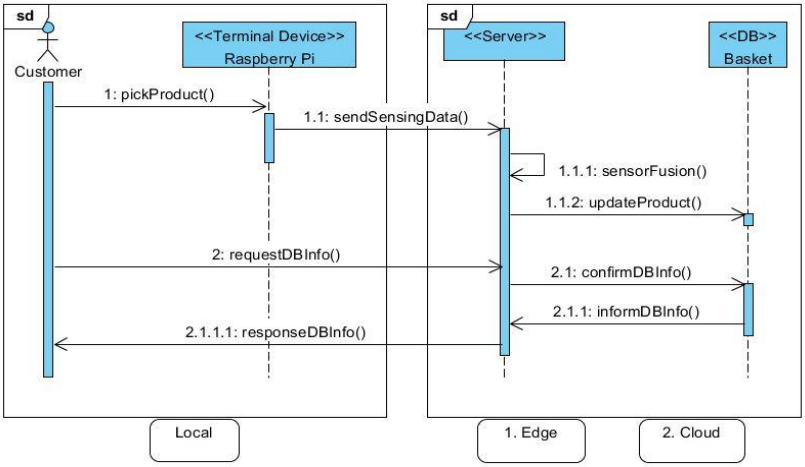


Figure 23 비교 환경 Sequence Diagram

	1. Edge	2. Cloud
RAM	16.3 [GiB]	1 [GiB]
CPU core	6 [개]	1 [개]
CPU cache size	12.2 [MB]	8 [MB]
OS	Linux 16.04	Linux 18.04
Nvidia GPU RAM	8 [GB]	X
프로세스 기본 주파수	3.2 [GHz]	2.4 [GHz]

Figure 24 비교 컴퓨터 성능 비교

실험에 사용된 두 컴퓨터의 성능은 Figure 24 과 같다.

[단위: msec]

기술	1. Edge	2. Cloud
Face Recognition	2.1	120
QR code	14.4	90
Sensor	0.1	6

Figure 25 컴퓨터별 Average Processing Time

각 Process 별로 약 1000 회씩 실행시킨 후 평균값을 계산한 평균 Processing Time 은 Figure 25 과 같다.

Terminal Device 에서 Server 로 Data 가 전송되는 시간을 Tc1, Tc2 라고 하고 수신 받은 Data 로 Server 에서 처리하는 시간을 Tp 로 하였다.

즉 총 Service Time(측정시간) T = Tc1 + Tc2 + Tp = 2 * Tc1 + Tp 이므로 결과적으로 Tc = (측정시간 - Tp) / 2 로 결론을 도출하였다. 그 결과는 아래의 그래프들과 같다.

8.2 실험결과

본 논문에서 제안하는 방법인 4 가지 프로세스들에 대해 응답 시간(Tc)을 CDF (Cumulative Distribution Function)으로 나타내었다.

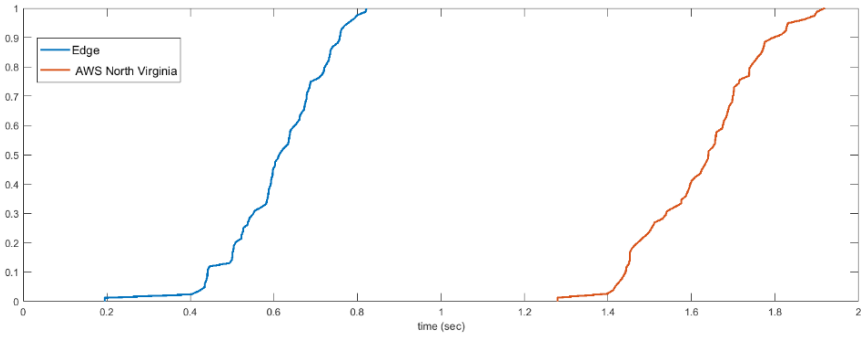


Figure 26 Face Recognition 비교 결과

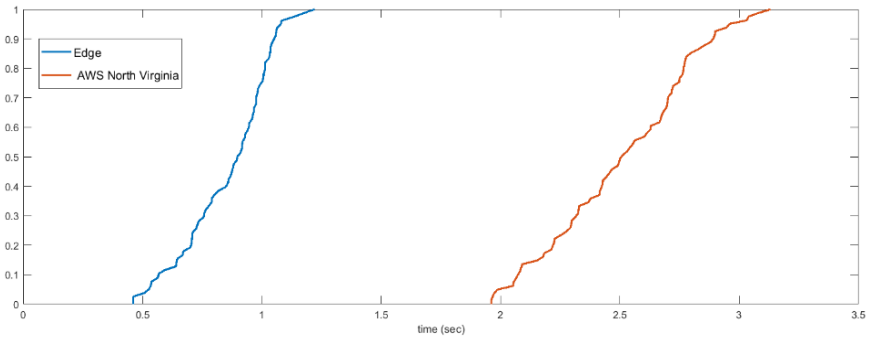


Figure 27 QR Code 비교 결과

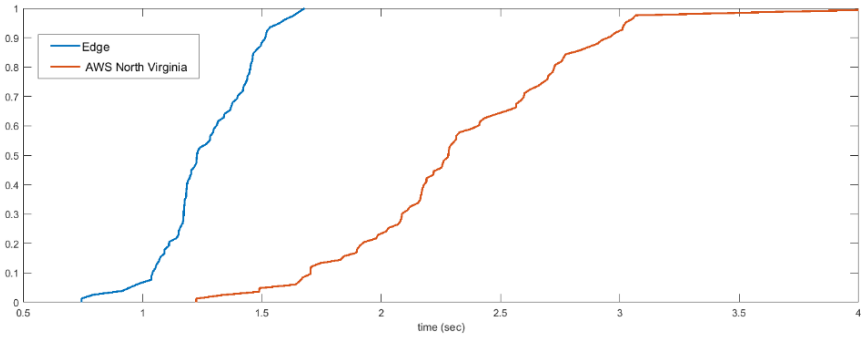


Figure 28 Sensor 비교 결과

이미지 처리 프로세스인 Face Recognition 과 QR Code Detection 에서는 Figure 26, Figure 27 에서 보는것과 같이 RTMP 프로토콜을 활용하여 신뢰도 높은 통신으로 영상 전송이 되어 Maximum Response Time 과 Minimum Response Time 의 차이가 크지 않지만, 8 할 의 프로세스들은 Face Recognition 와 QR Code Detection 에서 각각 0.72 초, 1.73 초와 1.1 초, 2.8 초 이내로 처리가 완료된다. 즉 본 논문에서 제안하는 방법인 Edge Computing 환경에서 Cloud Computing 환경에서 보다 2.4~2.5 배정도 빠르다는 것을 알 수 있다.

Sensor Fusion 에서는 Figure 28 에서 보는것과 같이 Maximum Response Time 과 Minimum Response Time 의 차이가 두 환경에서 확연히 다른 것을 확인할 수 있다. 이는 위의 두 프로세스와 다르게 각 센서별로 신뢰도 낮은 스트리밍이 다중으로 들어오게 되어 지연시간이 센서별로 누적되는 상황이 발생하는 프로세스이다. 그러므로 본 논문에서 제안하는 방법을 활용하면 이러한 오차를 확연하게 줄이게 되어 Maximum Response Time 과 Minimum Response Time 의 차이를 1/3 이상으로 축소 시킬 수 있다.

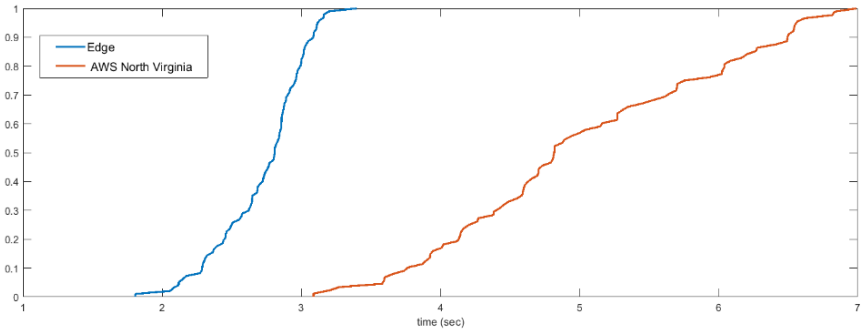


Figure 29 Total Service 비교 결과

Figure 29 는 위 세가지 프로세스들을 합친 실제 사용자가 물건을 집·놓은 상황에서의 측정값으로 앞서 설명한 두가지 장점을 나타내고 있다. 즉, 연구에서 제안하는 방법을 사용한 경우 전체적인 응답시간을 단축 시킬 수 있을 뿐만 아니라, 응답시간의 오차율을 대폭 감소시키게 되어 무인 마켓을 사용하는데 있어서 사용자가 불편함을 느낄 확률을 감소시키는 효과를 가져온다.

9 향후 발전 방향

9.1 Tracking

현재 HUFS GO 는 매대에 있는 Camera Sensor 로 얼굴인식을 통하여 상품을 집은 고객을 판단하고 있다. 이는 너무 제한적이다 향후로는 여러 개의 Camera Sensor 들을 천장에 부착하여 고객이 입장한 순간부터 계속 Tracking 함으로써 상품을 집은 고객을 판단할 수 있도록 발전 시킬 예정이다.

9.2 Service

현재 HUFS GO 는 상품을 1 개만 집고 놓아야 하고 상품을 다른 매대에 올려놓았을 경우 안 된다는 제약 사항이 있다. 이러한 제약 사항을 여러 개의 Sensor 들을 추가하여 더욱더 정밀한 Sensor Fusion 을 하여 발전 시킬 예정이다.

10 결론

본 연구에서는 Edge Computing 의 효율을 알아보기 위해 Amazon 사의 Amazon Go 를 벤치마킹한 무인 마켓을 구현하였고 Cloud Computing 과 비교실험을 진행하였다.

실험 결과 Face Recognition, QR-Code, Sensor 그리고 Total Service 모든 부분에서 Edge Computing 이 Cloud Computing 보다 속도 면에서 좋다는 것을 확인 할 수 있었다. 따라서 Edge Computing 의 효율이 좋다는 결론을 도출 할 수 있었다.

비교 실험을 하였던 Cloud 는 AWS 에서 제공하는 EC2 Linux 프리티어 버전을 활용하여서 Edge 에 비해 성능이 좋지 않은 것으로 테스트하여 실험을 성공적으로 완료 하였다. Edge 와 비슷한 성능을 가진 Cloud 와 비교를 하였어도 거리에 따른 Delay 가 존재 하기 때문에 성공적으로 완료 할 수 있었을 것이다.

본 프로젝트에서 진행하였던 시연 동영상을 Youtube 에 등록하였고 참고할 수 있도록 본 문서에 해당 링크를 첨부하도록 한다.

시연 동영상: https://www.youtube.com/watch?v=XOmQtkOHeqk
GitHub URL: https://github.com/rihjeo/Computervision,
https://github.com/rihjeo/SensorFusion

11 참고문헌

[1] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu : Edge Computing: Vision and Challenges. IEEE Internet of Things Journal (Volume: 3, Issue: 5, Oct. 2016)

[2] Hesham El-Sayed, Sharmi Sankar, Mukesh Prasad, Deepak Puthal, Akshansh Gupta, Manoranjan: Edge of Things: The Big Picture on the Integration of Edge, IoT and the Cloud in a Distributed Computing Environment. IEEE Access (Volume: 6)

[3] Edge Computing의 장점

https://m.blog.naver.com/PostView.nhn?blogId=alice_k106&logNo=221300292775&proxyReferer=https%3A%2F%2Fwww.google.com%2F

[4] Bastug, Ejder, et al. "Toward interconnected virtual reality: Opportunities, challenges, and enablers." IEEE Communications Magazine 55.6 (2017): 110-117.

[5] Mahadev Satyanarayanan"The Emergence of Edge Computing" IEEE Computer (Volume: 50, Issue: 1, Jan. 2017)

[6]R. Girshick, J. Donahue, T.Darrel, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation”, https://arxiv.org/abs/1311.2524

[7]R. Girshick, “Fast R-CNN”, https://arxiv.org/abs/1504.08083

[8]Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, “Faster R-CNN”, https://arxiv.org/abs/1506.01497

[9]py-faster-rcnn (Github Issues), “Why does it need a reshape layer in RPN’s cls layer?”, https://github.com/rbgirshick/py-faster-rcnn/issues/292

[10]Faster R-CNN (Caffe + Python)

https://github.com/rbgirshick/py-faster-rcnn

[11]Faster R-CNN (Caffe + Matlab)

https://github.com/ShaoqingRen/faster_rcnn

[12] Going Deeper with Convolutions

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich

[13] Rethinking the Inception Architecture for Computer Vision

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna

[14] Face Recognition

https://github.com/ageitgey/face_recognition