Kalman Filter와 Convolutional Neural Network(CNN)을 활용한 RSSI-거리 측정 최적화에 대한 연구

최동헌, 이재협, 백현규, 허창현

아주대학교 수학과

요약

최근 위치를 기반으로 사용자에게 서비스를 제공하는 어플리케이션이 매우 많다. 측위를 위해 중 비콘의 RSSI 를 이용하기도 한다. 그러나 RSSI 측정값을 그대로 사용하기에는 그 값이 매우 불규칙적이고 오차가 크다는 단점이 있다. 따라서 Kalman Filter 를 사용해서 측정된 RSSI 를 보정하고, 이를 Convolutional Neural Network 를 사용하여 학습시켜 정확도를 높여보았다.

Ⅰ. 서 론

2020 년 코로나 19 사태로 인해 외부 활동이 축소됨에 따라, 실내 공간 활동의 중요성이 강조되고 있다. 이로 인해 실내에서 제공되는 실내 위치 확인 시스템(Indoor Positioning System, IPS)이 주목받고 있다. 실내 위치 확인 시스템은 실내에서 단말기의 위치를 측정하는 기술이다. 대표적 위치 서비스인 GPS 는 위성과 통신이 가능한 외부에서 주로 사용된다. 때문에 실내에서는 블루투스, Wi-Fi, RFID 등의 기술을 사용한다.

코로나 19 사태로 세계 경제가 주춤한 가운데, 블루투스 위치 서비스 시장은 최근 몇 년 동안 크게 성장하고 있다. 또한 2025 년경에는 블루투스 기술이 탑재된 기기가 60 억대를 넘어설 것으로 전망하고 있다. 블루투스는 저전력으로 신호를 주고받을 수 있고 작은 범위에서 정밀한 측위가 가능하다는 장점이 있다. 기존의 블루투스 신호를 이용한 실내 위치 측위 방식으로는 FingerPrint, 삼변 측량 등이 있다.

FingerPrint 방식은 특정 위치에서 보내는 블루투스 신호를 저장해 놓고, 신호들을 모아 서버로 요청하면 이미 기록된 신호들을 수집하여 유사도를 계산하는 방식이다. 하지만, 해당 서비스 지역의 실내 지도와 신호 수집이 필요하다는 단점이 있다.

삼변 측량을 이용한 측위는 각각의 블루투스 장비에서 보내는 신호 강도를 이용해서 현위치에서의 거리를 구하고 교차하는 공간의 중앙점을 위치로 결정하는 기술이다. FingerPrint 방식과는 달리 블루투스 장비들의 정확한 위치만 알면 계산을 통해 측위를 할 수 있다는 장점이 있지만, 지형지물, 장비 위치의 변화로 인한 변수가 많이 발생한다.

본 논문에서는 블루투스의 신호인 RSSI 값을 이용하여 스마트 기기와 블루투스 비콘과의 거리를 측정한다. 환경 요인으로 인한 오차가 발생할 수 있으므로, RSSI 값 안정화를 위해 칼만필터 알고리즘과 CNN 을 이용하여 분산된 기본 신호 편차를 보정하는 연구를 진행하였다.

Ⅱ. 비콘을 이용한 측위와 RSSI 안정화 방안

1.RSSI

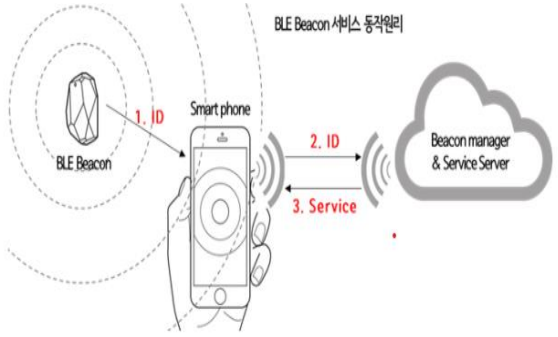
RSSI 란 Received Signal Strenght Indicator 의 약자로, 수신 강도를 나타내는 지표이다. 강도의 단위는 dBm 이다. RSSI 는 보통 -99dBm~ 35dBm 까지의 세기를 송출한다. 숫자가 높을수록 신호의 강도가 강하다가 볼 수 있다. RSSI 값을 통해 거리를 구하는 공식이 있지만, 신호강도는 장애물에 의해 왜곡된 값을 가지게 된다.

$$RSSI = -(n \log_{10} d + A)$$

또한 RSSI 값이 1 만 변해도 거리 오차가 m 단위로 발생하기도한다. 따라서 순수하게 얻은 RSSI 값은 직접적으로 이용하기 어렵다는 단점을 가진다.

2.비콘

비콘은 가까운 거리에 있는 스마트 기기를 인식하여 필요한 데이터를 전송할 수 있는 무선 통신 장치이다. 블루투스 비콘이라고 부르기도 한다. 블루투스 비콘은 RSSI 값을 전송할 수 있기 때문에 비콘과 스마트 기기 사이의 거리를 추정할 수 있다. 블루투스 비콘은 크기가 작아서 설치가 간편하고 저전력으로 사용할 수 있다는 장점을 가진다.



비콘의 동작 원리는 신호 도달 거리 내에 스마트 기기가 들어오면, 비콘은 ID 값을 보낸다. 스마트 기기의 애플리케이션은 비콘으로부터 받은 ID 값을 인식하여 서버로 전달하게 된다. ID 를 가지고 있는 서버는 위치를 확인하게 되고, 해당 위치에 설정된 서비스를 스마트폰으로 전송한다.

3.칼만필터

루돌프 칼만(Rudolf E. Kalman)이 개발한 잡음이 포함되어 있는 역학적 상태를 추적하는 재귀 필터이다. 칼만 필터링은 불규칙 외난을 포함하는 동적 시스템에 적용되는 최적 상태 추정과정이다. 칼만필터는 이산 실시간격마다 측정되는 잡음이 실린 데이터로부터 동적시스템의 미지의 상태변수를 최적으로 추정하기위한 선형, 불편, 최소오차분산의 반복적 알고리즘이다.

칼만필터는 입력과 출력이 하나씩인 아주 간단한 구조이다. 측정값 (z_k)이 입력되면 내부에서 처리한 다음 추정값($\hat{x}_k$)을 출력한다. 내부 계산은 총 네 단계에 걸쳐 이뤄진다.

0. 초깃값 선정

$$\hat{x}_0, P_0$$

1. 추정값과 오차 공분산 예측

$$\hat{x}_{\bar{k}} = A\hat{x}_{k-1}$$

$$P_{\bar{k}} = AP_{k-1}A^T + Q$$

2. 칼만 이득 계산

$$K_k = P_{\bar{k}}H^T\left(HP_{\bar{k}}H^T + R\right)^{-1}$$

3. 추정값 계산

측정값 z_k ⟶ ⟵ 추정값 $\hat{x}_k$

$$\hat{x}_k = \hat{x}_{\bar{k}} + K(z_k - H\hat{x}_{\bar{k}})$$

4. 오차 공분산 계산

$$P_k = P_{\bar{k}} - K_kHP_{\bar{k}}$$

첫 번째 단계는 예측 단계이다. 이 단계에서는 2~4 단계에서 계속 사용하는 두 변수 $\hat{x}_{\bar{k}}$와$P_{\bar{k}}$를 계산한다. 위첨자 '-'는 예측값을 의미한다. 예측 단계의 계산식은 시스템 모델과 밀접하게 관련되어 있다. 2 단계에서는 칼만 이득(K_k)을 계산한다. 변수 $P_{\bar{k}}$는 앞 단계에서 계산한 값을 사용한다. 그리고 H 와 R 은 칼만 필터

알고리즘 밖에서 미리 결정되는 값이다. 3 단계에서는 입력된 측정값으로 추정값을 계산한다. 아직 명확하게 드러나지 않았지만 이 단계의 계산식은 저주파 통과 필터와 관련 있다. 4 단계는 오차 공분산을 구하는 단계이다. 오차 공분산은 추정값이 얼마나 정확한지를 알려주는 척도로 사용된다. 보통 오차 공분산을 검토해서 앞서 계산한 추정값을 믿고 쓸지 아니면 버릴지를 판단한다. A, H, Q, R 네 개의 변수는 칼만 필터를 구현하기 전에 미리 결정해야 한다. 즉 칼만 필터 알고리즘에서 계산하거나 가정하는 값이 아니다. 이 값들은 대상 시스템과 칼만 필터를 사용하는 목적에 따라 설계자가 사전에 확정한다. 칼만 필터의 성능에 중요한 영향을 미치는 값이다. 칼만 필터 알고리즘의 계산 과정은 네 단계로 되어 있지만, 의미를 기준으로 나누면 예측 과정과 추정과정 두 단계로 분류된다.

예측 과정

1 단계가 여기에 해당된다. 직전 추정값($\hat{x}_{k-1}$)과 오차 공분산(P_{k-1})이 입력되면 최종 결과로 예측값($\hat{x}_{\bar{k}}$, $P_{\bar{k}}$)을 내놓는다. 이 값들은 추정 과정에 사용된다. 이 단계에서 사용하는 시스템 모델 변수는 A와 Q 이다.

추정 과정

칼만 필터 알고리즘에서 2 단계, 3 단계, 4 단계가 여기에 속한다. 추정 과정의 결과물은 추정값($\hat{x}_k$)과 오차 공분산(P_k)이다. 입력값으로는 예측 과정의 예측값($\hat{x}_{\bar{k}}$, $P_{\bar{k}}$) 뿐만 아니라, 측정값(z_k)을 전달받아 사용한다. 이 단계에서 사용하는 시스템 모델 변수는 H와 R 이다.

이러한 관점에서 칼만 필터 알고리즘을 정리하면 다음과 같다.

1.시스템 모델(A , Q)을 기초로 다음 시각에 상태와 오차 공분산이 어떤 값이 될지를 예측한다. $\hat{x}_{\bar{k}}$, $P_{\bar{k}}$

2.측정값과 예측값의 차이를 보정해서 새로운 추정값을 계산한다. 이 추정값이 칼만 필터의 최종 결과물이다. $\hat{x}_k$, P_k

3.위의 두 과정을 반복한다.

추정 과정은 3 단계와 4 단계에 해당하는 과정이며 칼만 필터의 최종 결과물인 추정값을 계산해내는 과정이다.

3 단계 추정값을 계산하는 식은 $\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$이며 z_k는 측정값 $\hat{x}_k^-$는 예측값을 의미한다.

앞서 얘기한대로 이과정은 1 차 저주파 통과 필터와 관련이 있다. 식을 정리할 경우 $\hat{x}_k^- = (1 - K_k H)\hat{x}_k^- + K_k z_k$과 같이 나오는데 H를 단위행렬로 가정할 경우 $\hat{x}_k^- = (1 - K_k)\hat{x}_k^- + K_k z_k$는 1 차 저주파 통과 필터의 수식 ($\bar{x}_k = \alpha\bar{x}_{k-1} + (1-\alpha)x_k$)과 유사하는 것을 볼 수 있다.

수식이 유사한만큼 계산과정도 유사하다. 1 차 저주파 필터는 직전 추정값과 측정값에 가중치를 부여한뒤 더하여 추정값을 계산한다. 칼만필터 또한 예측값($\hat{x}_k^-$) 과 측정값(z_k) 에 적절한 가중치를 곱한 다음 두 값을 더하여 최종적인 추정 값을 계산한다.

하지만 칼만 필터가 1 차 저주파 통과 필터와 구별되는 점은 1 차 저주파 필터는 가중치가 상수로 그 값이 변화하지 않는다. 반면에 칼만필터의 가중치인 칼만이득(Kalman gain)은 2 단계의 수식에서 보았듯이 그 값이 알고리즘을 반복함에 따라 K_k 값을 계산하여 가중치를 다시 조정하게 된다.

4 단계 오차 공분산 계산을 하는 단계에서는 다음과 같은 수식($P_k = P_k^- - K_k H P_k^-$)을 통해 오차 공분산을 예측과정으로 넘겨주면 된다.

이과정에서 중요한점은 추정 값이 정확한지 아닌지를 오차 공분산으로 판단할 수 있다는 점이다.

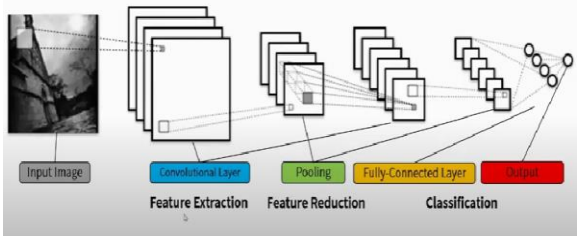
예측과정

예측과정은 알고리즘에서 1 단계에 해당하며 시각이 t_k 에서 t_{k+1}로 바뀔 때 추정 값 $\hat{x}_k$가

어떻게 변하는지를 예측한다. $\hat{x}_{k+1}^{-} = A\hat{x}_k$, $P_{k+1}^{-} = AP_kA^T + Q$ 이와 같은 식으로 예측을 한다. $\hat{x}_k$와 P_k는 3 단계, 4 단계에서 계산한 값이며 A와 Q는 시스템 모델에 정의되어 있는 값이다. 예측 단계의 계산식은 간단하지만 칼만필터의 성능에 큰 영향을 준다. 칼만 필터 추정 값 계산식($\hat{x}_k = \hat{x}_k^{-} + K_k\,(z_k - H\hat{x}_k^{-})$)에서 $H\hat{x}_k^{-}$는 예측 값을 뜻한다 따라서 $z_k - H\hat{x}_k^{-}$는 실제 측정값과 예측한 측정값의 차이, 즉 측정값의 예측 오차를 뜻한다. 이를 통해 칼만필터는 측정값의 예측 오차로 예측 값을 보정하여 최종 추정 값을 계산한다는 걸 알 수 있다. 이처럼 추정 값은 예측 값의 정확성에 따라 그 정확도 가 달라진다. 아무리 칼만 이득을 잘 선정하더라도 시스템 모델의 A와 Q는 실제 시스템이 많이 다르면 예측 값은 부정확하게 되고 추정 값도 부정확하게 된다.

4. CNN (Convolutional Neural Network)

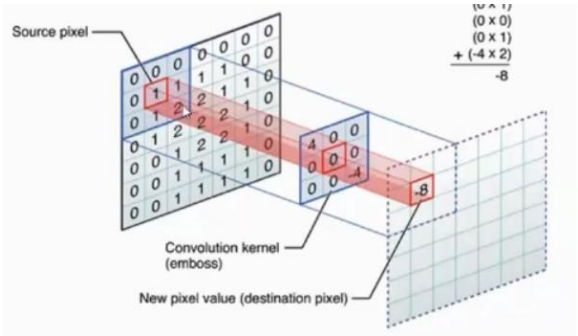
CNN(Convolutional Neural Network)은 DNN 의 부정확한 결과를 해결하기 위해서 개발된 알고리즘이다. 다음 그림은 CNN 의 구조이다.



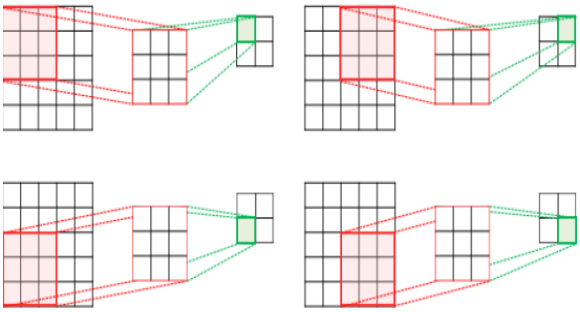
CNN 의 구조는 크게 5 단계로 나눌 수 있다. Input 과 Output 이 기본으로 되는 구조이다. CNN 에서 Input 은 2D 혹은 3D 이미지이다. Input 과 Output 사이에는 크게 3 단계로 분류한다.

첫번째는 입력 데이터로부터 특징을 추출할 수 있는 Convolutional Layer 이다. 두번째는 전 단계에서 만들어진 Feature 들을 Reduction 하는 과정이다. 이는 Feature 들을 Resizing 하여 새로운 layer 을 얻는 과정인 pooling 을 거친다. 세번째는 Feature Extraction 과 Feature Reduction 을 거친 최종 Feature 들을 평탄화 작업을 통해 이를 분류하는데 사용하는 Fully connected layer 이다.

Convolutional layer 은 Convolution 연산을 통해 이미지의 feature 을 추출하는 역할을 한다. Convolution 연산은 kernel 이라 불리는 m × n 크기의 행렬로 이미지와 겹쳐지는 부분을 곱해서 모두 더하는 과정이다. 다음 그림은 Convolution 의 예시이다.

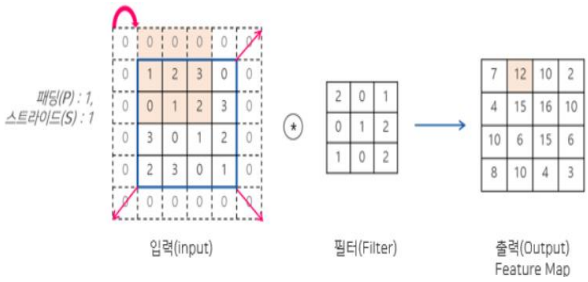


그림과 같이 Input 으로부터 kernel 을 사용하여 통해 나온 결과를 feature map 이라 부른다. Kernel 의 크기와 이동 범위는 사용자가 정할 수 있다. 이때 이동 범위는 stride 라고 한다. 다음 그림은 stride 가 2 일때 5×5 이미지에 3×3 kenel 의 움직임을 보여준다.

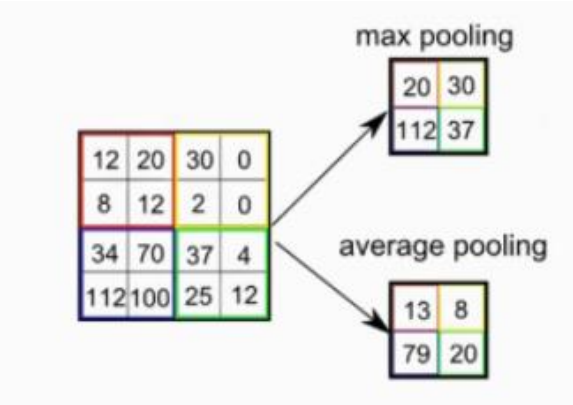


그림과 같이 Convolution 결과로 얻은 feature map 은 Input 이미지보다 크기가 작아진다. feature map 의 크기를 Input 과 동일하게 만들기 위해 padding 을 사용한다. 가장자리에 유의미한 데이터가 존재할 경우 가장자리의 데이터가 다른 데이터들에 비해 활용도가 떨어지는 것을 방지하기 위해 Padding 을 통해 임의의 데이터를 가장자리에 추가하여 가장자리 데이터의 활용도를 높인다. 지정된

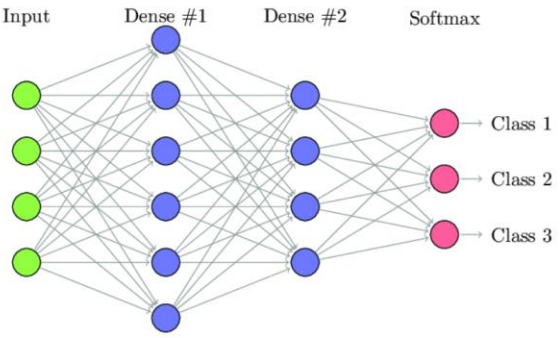
개수의 폭만큼 테두리를 추가하고 주로 값을 0 으로 채우는 zero padding 을 사용한다. 다음 그림은 padding 의 예시이다.



Pooling layer 에서는 featyre map 의 크기를 줄여 훈련시킨 데이터에서만 높은 성능을 보이는 overfitting 을 방지하고 그 특징을 더 잘 인식할 수 있게 한다. pooling 과정에서도 kernel 과 stride 의 개념을 가진다. pooling 연산에서는 일반적으로 max pooling 과 average pooling 이 사용된다. max pooling 은 kernel 과 겹치는 영역 안에서 최대값을 추출하는 방식이다. average pooling 은 평균값을 추출하는 방식이다. 다음 그림은 pooling 의 예시이다.



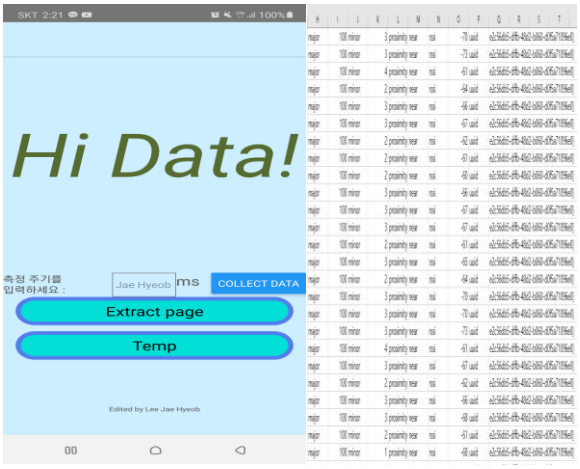
Fully connected layer 은 추출된 결과를 정의된 라벨로 분류하는 데 사용한된다. Dense layer 이라고도 부른다. Flatten 을 통해 흑백 이미지, 2 차원 벡터의 행렬을 1 차원 배열로 평탄화하고 relu 함수로 활성화하고 softmax 함수로 이미지를 분류하는 과정이다.



Ⅲ. 실험 과정 및 결과

1. 실험환경

실험은 실내 주차창에서 진행하였다. 비콘과 스마트 기기와의 거리는 0.5m, 0.8m, 1.2m, 1.6m, 2.0m, 2.4m, 2.8m 로 각각의 거리에서 RSSI 를 수집하였다. 다음 그림은 RSSI 값을 측정할 수 있는 스마트폰 어플리케이션과 이로 얻은 RSSI 측정값이다.



2.RSSI 측정 결과

다음 그림은 각 거리별 이론상 RSSI 값, 어플리케이션으로 측정한 RSSI 의 평균값이다. 데이터는 거리별로 2000 개를 수집하였다.

실제 거리	이론상 RSSI(평균)	실제 RSSI(평균)
0.5m	-57.97	-61.37
0.8m	-62.06	-63.19
1.2m	-65.58	-65.98
1.6m	-68.08	-65.22
2.0m	-70.02	-65.80
2.4m	-72.94	-70.29
2.8m	-75.12	-66.72

실제로 측정한 RSSI 값이 이론상 RSSI 값과 오차가 다양하게 발생한 것을 볼 수 있다. 오차가 0.4dbm 부터 9dbm 까지 발생했다.

3.RSSI 값을 이용한 거리 환산 및 정확도 확인

RSSI 값을 이용하여 거리로 변환하기 위해 다음 공식을 이용하였다.

$$RSSI = -(n \log_{10} d + A)$$

다음 그림은 RSSI 를 통해서 거리를 구하는 함수, 거리를 이용해서 RSSI 를 구하는 함수, 평균과 표준편차를 구하는 함수를 구현한 코드이다.

```
class Beacon:
    x, y, N, M_Power = 0, 0, 0, 0
    mu_RSS, sigma_RSS = 0, 0
    dataset = []

    def __init__(self, x, y, N, M_Power):
        self.x = x
        self.y = y
        self.N = N
        self.M_Power = M_Power

    def RSSI_to_distance(self, RSSI):
        return math.pow(10, ((self.M_Power - RSSI) / (10 * self.N)))

    def Dist_to_RSSI(self, Dist):
        return ((-10) * self.N * math.log(Dist, 10) + self.M_Power)

    def set_mu_sigma_RSS(self, a, b, c):
        temp = []
        temp.append(a)
        temp.append(b)
        temp.append(c)
        self.mu_RSS, self.sigma_RSS = np.mean(temp), np.std(temp)

    def add_RSS(self, RSS):
        self.dataset.append(RSS)
        self.mu_RSS = np.mean(self.dataset)
        self.sigma_RSS = np.std(self.dataset)
```

다음 그림은 공식을 이용하여 측정한 RSSI 값을 거리로 변환한 값과 오차율이다.

실제 거리	RSSI로 측정한 거리	오차율
0.5m	0.73	46%
0.8m	0.91	14%
1.2m	1.25	4%
1.6m	1.15	28%
2.0m	1.23	39%
2.4m	2.06	14%
2.8m	1.36	51%

이론상 거리가 멀어질수록 오차율이 크게 발생해야한다. 하지만 거리가 멀어짐에 따라 오차율이 증가하지는 않았다.

4.실제 거리에 가장 적합한 칼만필터 시스템 모델 설정

칼만필터를 사용하기 위해서는 시스템 모델을 설정해야 한다.

Q,R은 잡음의 특성을 정확히 반영해서 구성하는 게 원칙이지만 여러 오차가 복합적으로 작용하기 때문에 해석적으로 결정하는 데는 한계가 있다. 잡음에 대한 지식을 최대한 활용하되, Q,R을 칼만필터의 설계 인자로 보고 시행착오 과정을 통해 보정하면서 적절한 값을 찾아야한다. 실험 결과, A,H,Q,R이 각각 1, 1, 0, 1000 을 사용했을 때가 실제와 가장 유사했다.

<실제 A,H,Q,R을 실험해본 결과 중 일부>

이론상 RSSI	-65.58	
(Q,R)	측정 RSSI 평균	표준편차
(0,50)	-66.95174	1.293
(0, 100)	-66.9471	1.33
(0, 1000)	**-66.81694**	**0.915**
(1,50)	-67.07816	1.117
(1, 100)	-67.02378	1.157
(1, 1000)	-66.84224	1.711

5. Convolutional Neural Network(CNN)

사전에 측정해둔 Data 들을 학습시켰다. Input 에 해당하는 RSSI 와 output 에 해당하는 거리를 적용하여 filter 의 원소들을 찾아냈다. 그 후, input 과 filter 를 통해서 output 을 유추해내는 실제 적용 과정을 진행했다.

6.설정한 시스템 모델을 기준으로

모델 적용 방식은 다음과 같다.

.

7. 실험결과

이전 오차율에 비해 다소 줄어든 것을 알 수 있다.

<결과 중 일부 : 0.5m 에서 실험해본 결과>

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 c1d (Conv1D)                (None, 206, 1)            5

=================================================================
Total params: 5
Trainable params: 5
Non-trainable params: 0
_________________________________________________________________
Input RSS: -60  -61 -59 -58 -58
47.528821885585785
```

실제 거리	추정 RSSI로 측정한 거리	오차율
0.5m	0.475	5%
0.8m	0.858	7.25%
1.2m	1.245	3.75%
1.6m	1.732	8.25%
2.0m	2.14	7%
2.4m	2.70	12.5%

9. 참고문헌

[1] 김우찬, 이청길, 곽호영, "위치측정을 위한 비콘의 RSSI 안정화", 제주대학교, 2019

[2] 박종형, 박두익, 염철민, 강진수, 원유재, "비콘의 세기를 이용한 실내 위치 추적 정확도 개선에 관한 연구", 충남대학교, 2018

[3] 김지성, 김용갑, "RSSI 측정결과 필터링을 이용한 거리계산 보정 알고리즘에 관한 연구", IBC, 2017

[4] 김지성, 김용갑, 황근창, "비콘의 RSSI 특성을 이용한 실내 위치 추적 시스템에 관한 연구", IIBC, 2017

[5] 김정진, 조성욱, 지영민, "CNN 을 이용한 이미지 분류", 한국정보기술학회, 2017

[6] 최학영, "CNN 기반 결함 검출", 서경대학교 대학원, 2017

[7] 이현수, "이미지 컨텐츠 검색을 위한 CNN 의 구조", 중앙대학교 대학원, 2018

[8] 김성필, 2019, "칼만 필터는 어렵지 않아:with MATLAB Examples", 한빛 아카데미