Human Emergency Recognition System based on Radar sensor using Deep Learning

동국대학교 멀티미디어공학과 2018113627 정경은
지도교수님 : 김지희 교수님

Index

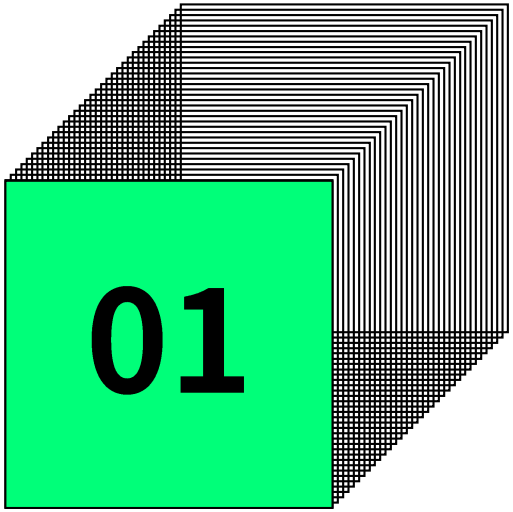
1. 프로젝트 수행 목표

2. 진행 과정

- 사례 조사
- 기획 과정
- 주요 기능 설명
- 사용 하드웨어 구성
- 구현 과정

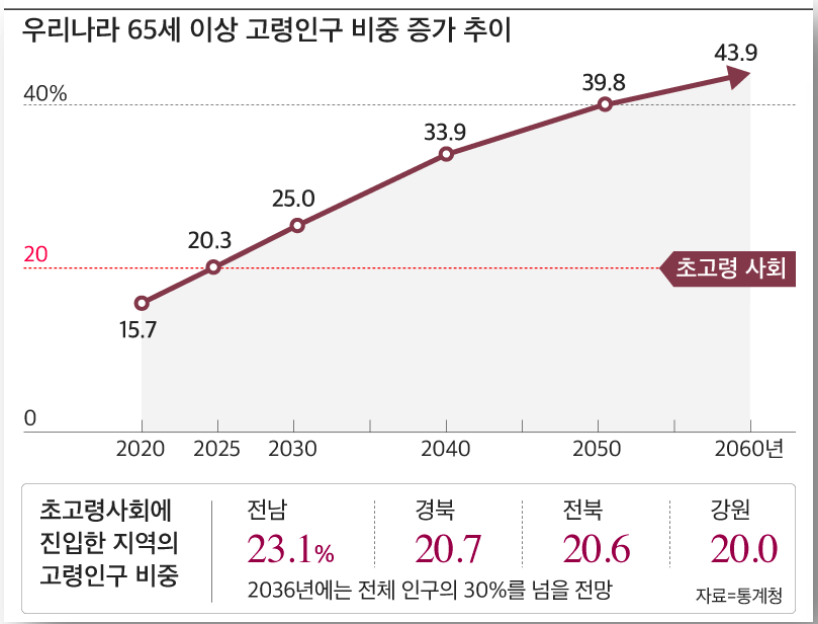
3. 프로젝트 결과

4. 추후 방안 및 일정



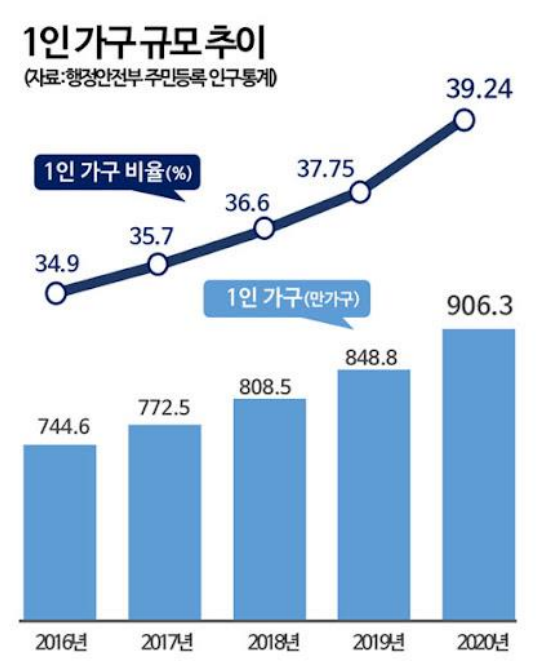
프로젝트 수행 목표

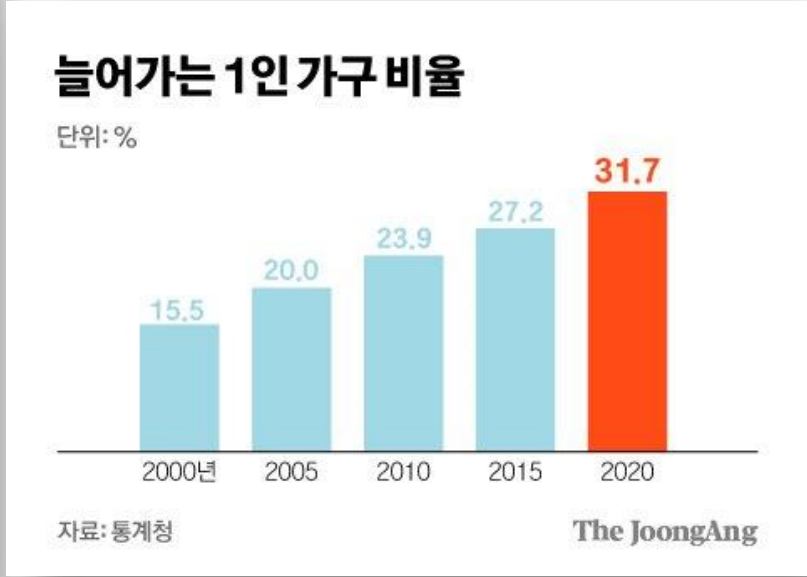
프로젝트 수행 목표 및 필요성



대한민국은 2017년 고령화 사회에서 **고령사회**로 전환 된 이래로,
2020년 7월 기준 노인 인구가 전체 인구 중에서 차지하는 비율이 15.7%에 달한다.
-> 통계청은 **초고령사회**가 2025년(전체 인구 대비 노인 인구 구성비는 20.0%)에 도달할 것으로 전망하고 있다.

프로젝트 수행 목표 및 필요성





19년 기준 전체 가구 10가구 중 3가구(30.2%)가 1인가구이며,
이러한 1인가구 중 절반(51.6%)은 본인이 노후생활비를 마련하는 추세이다.

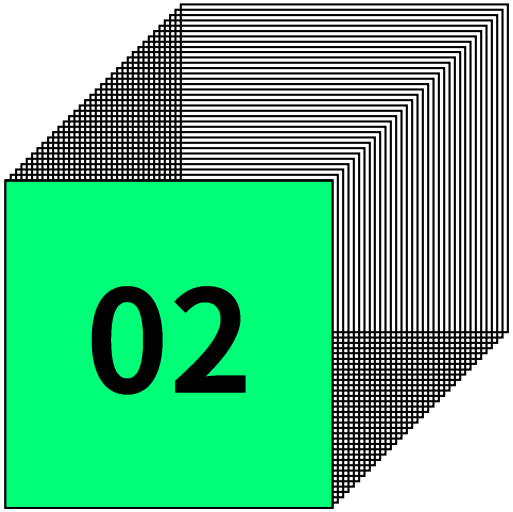
프로젝트 수행 목표 및 필요성

서울시는 코로나19로 인해
복지시설 휴관과 대면서비스 축소로 발생하는
취약계층 노인들의 돌봄 공백을 해소하기 위해 올해까지
‘취약어르신 안전관리 솔루션 사업’ 대상자를 총 1만 가구로 **확대**했다.

노인맞춤돌봄서비스 대상자 중 관계 단절, 우울, 고령 등으로 안전확인이 실시간으로 필요한 노인들을 대상으로
IoT(Internet of Things, 사물인터넷) 기기를 설치해 **일상을 관리**하고 **위급상황 시 대응**하는 서비스를 실시했다.

프로젝트 수행 목표 및 필요성

2020년 이래로 지금까지 지속 되고 있는 **코로나19 팬데믹 하**에서,
이들을 위한 **‘비대면 돌봄’을 최신 디지털 기술**을 제안하고자 한다.

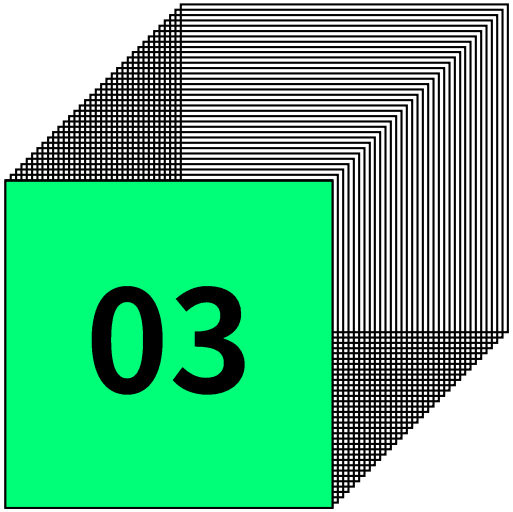


사례 조사

차세대 응급 안전 안심 서비스 댁내장비

보건복지부 보급 '**차세대 응급안전안심서비스 댁내장비**'
응급상황 발생 시 이를 실시간으로 소방서등과 연계해주는
정보통신기술(ICT) 기반 기존 기술

- 화재감지기, 활동량감지기(심박・호흡) 센서 탑재
- 조도・습도・온도감지센서 탑재



기획 과정

기존 시스템의 한계점 및 극복

① 그 대상이 응급안전안심서비스를 희망하는 독거노인 · 중증장애인에 그친다는 점

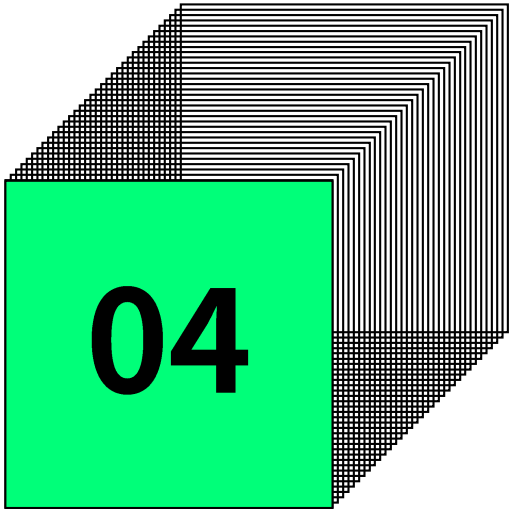
② 활동량 감지기(호흡, 심박)만으로 감지하기에는 한계가 있는 행동

- (ex. 65세 이상 사고 사망원인 2위에 해당하는 '낙상사고') 이 여전히 존재한다는 점

③ **보건복지부 산하 지자체를 통해서만 구매가 가능하다는 점**에서 일반화하기 어렵다는 문제점

④ 번거로운 기술 익힘과, 위급 상황에 한번 더 생각을 해야한다는 어려움

사용 대상을 단순 노인에 그치지 않는
'Radar Based Emergency Recognition System'을 통해
기존기술 대비 사용자로 하여금 **복잡한 작동을 익힐 필요 없이 간편히 행동감지를 수행**할 수 있도록 한다.



주요 기능

Radar Based Emergency Recognition System

제안 아이디어 : **Radar 센서를 기반으로 사람의 행동을 인식해 응급상황을 탐지하는 시스템**

① 빠르고 신속한 응급상황 감지

② 개인의 사생활을 침해하지 않으며, 사용자의 행동을 추정.

③ 응급상황 방지 및 개인 맞춤화 케어

사용자 범위 : 독거노인, 1인 가구 대상

Radar Based Emergency Recognition System

왜 Radar(레이더) 인가?

[스마트 워치를 통한 낙상 감지]

스마트워치를 항상 손목에 차고 있어야 하고,
기기를 지니지 않고 있다면 사용할 수 없다.

[카메라를 통한 낙상 감지]

빛에 세기에 취약하다.
일상생활에서 사용하기엔 사생활 침범의 문제가 있다.

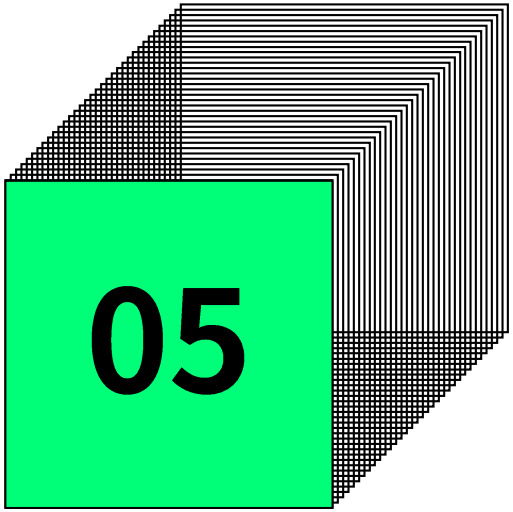
Radar Based Emergency Recognition System

왜 Radar(레이더) 인가?

① 웨어러블 디바이스를 사용할 때와 다르게
별다른 조절 방식을 익힐 필요가 없다.

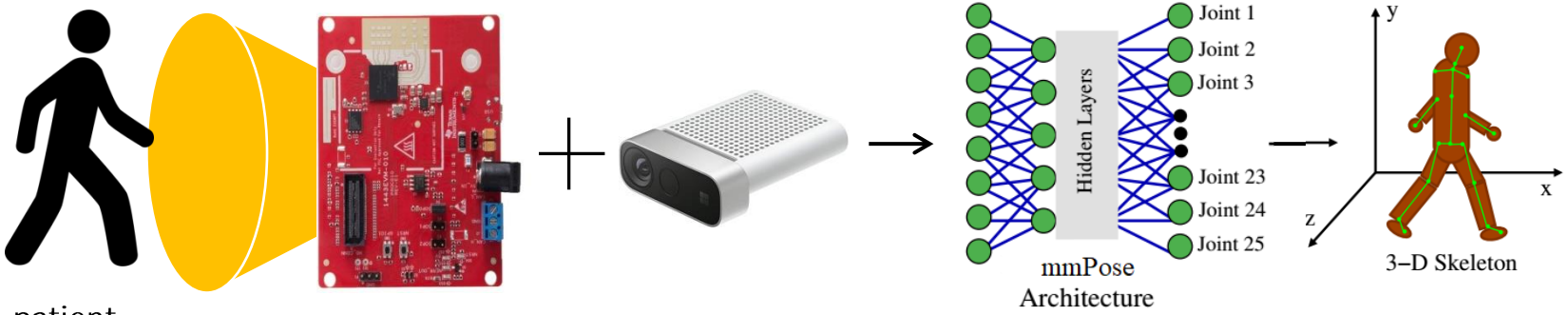
② **컴퓨팅 파워의 강도가 크지 않으며,**
간편하게 사용할 수 있다.

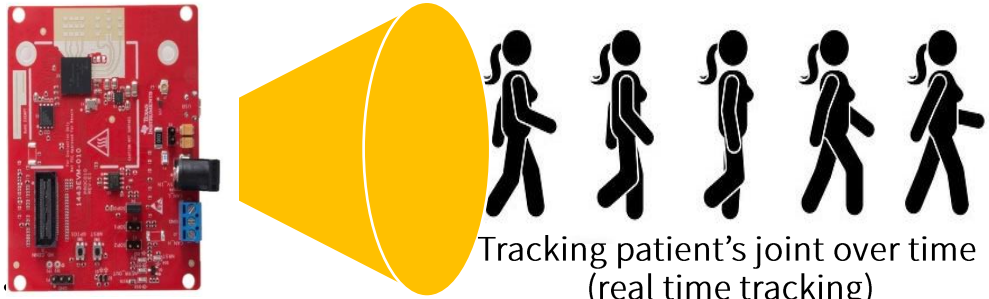
③ **카메라로부터 얻어지는 image를 활용하는**
모델과는 달리 사생활 침해 문제로부터 자유
롭다는 장점



구현 과정

How To ? – 간단 구조





Current Activity

Future Activity

Patient Name
Attending Dr
Condition

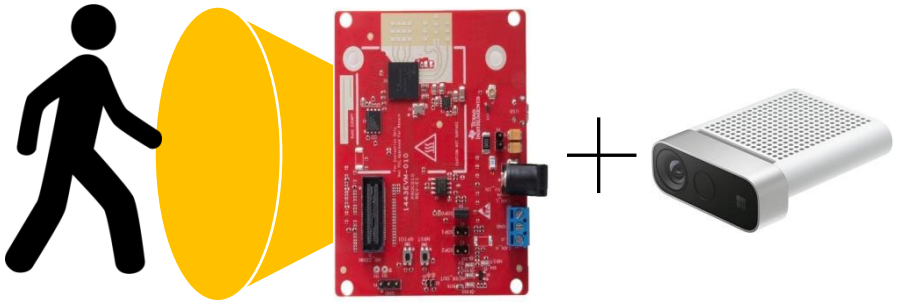
Current Patient Activity

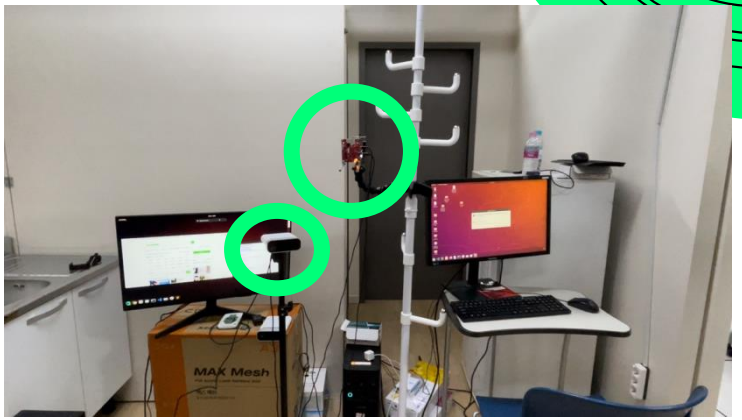
Walking

Future Patient Activity

Walking (88%)

How To ? – 간단 구조 – (1)



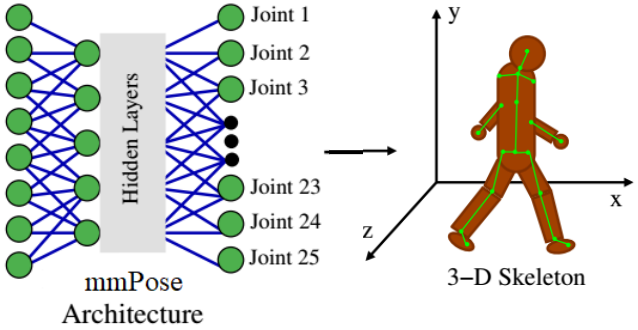


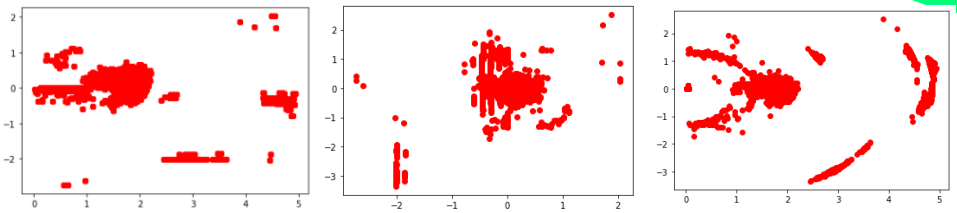
Human action 데이터 수집 – via Radar + KinectV4

대상: 동국대학교 내 학생
장소: 동국대학교 건물 내에 실제 가정 집처럼 부엌, 화장실, 침대, 그리고 매트릭스가 구비되어 있는 실험실에서 진행
[서기', '앉기', '스쾌트', '뛰기', '걷기'],
추후 더 다양한 자세들이 추가 될 예정이며, 기본적으로는 이 5가지 자세가 Base이다.
각 자세들을 취해가며 3D Radar Sensor로부터사용자의 자세에 대한 데이터(point cloud)를 획득한다.

Radar : point cloud / Kinect : 3D skeleton

How To ? – 간단 구조 – (2)



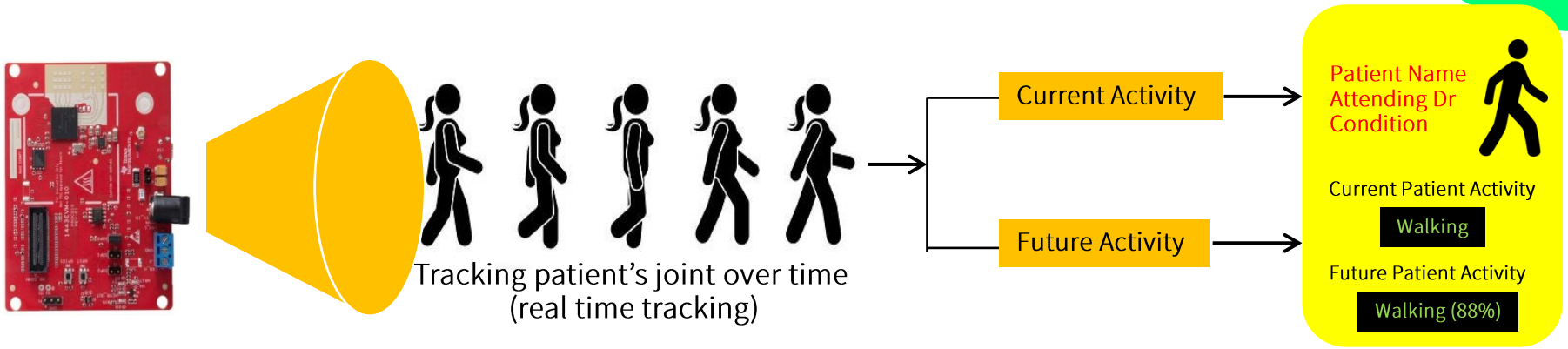


머신러닝 및 딥러닝 기술을 적용시켜 행동 인식 모델을 구축

키넥트(Kinect) 센서로부터 얻은 3D data와 Radar Sensor로부터 얻은 데이터를 가지고 미리 모델을 사전학습(pre-training)시킨다.
이후에 radar data 로 모델을 정밀학습(fine-tuning)시킨다.

Radar : point cloud / Kinect : 3D skeleton
두개의 데이터를 서로 맵핑 시켜, feature값을 획득한 후, 정밀학습을 진행해
Real time 모드에서 구별이 가능한 행동인식모델을 구축한다.

How To ? – 간단 구조 – (3)



Real time (실시간) 행동 인식 및 분석

최종적으로 학습이 완료된 모델은 실시간으로 전송되는 radar data(pcl) 만을 가지고 사람의 행동을 파악

사용하는 하드웨어 – Radar (Ti IWR1443BOOST)

[easy-to-use 77GHz mmWave sensor evaluation board]

- UART 통신
- TI MCU launchpad를 가지고 있음
- 빠른 데이터 전송 처리

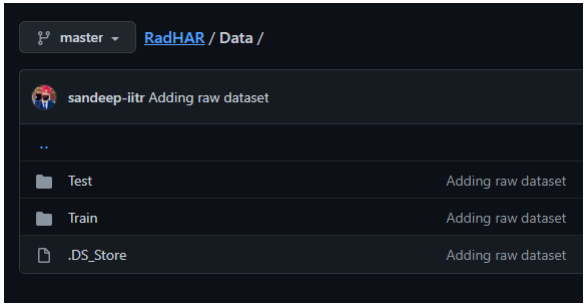
사용하는 하드웨어 – Kinect v4 (microsoft azure)

[kinect V4 - 마이크로소프트사]

애플리케이션을 최적화하는 데 유용한 광시야각/협시야각 옵션이 포함된 1MP 깊이 센서

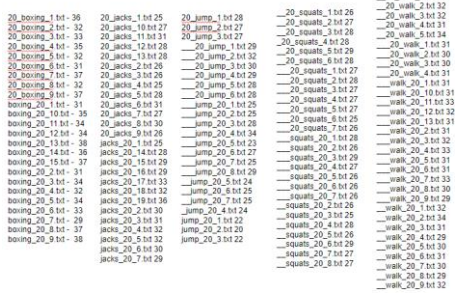
센서 방향 및 공간 추적을 위한 가속도계 및 자이로스코프(IMU)

구현 과정 (0) – 학습 아이디어 도출 및 사례조사1



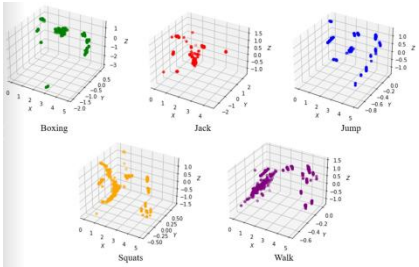
https://github.com/nesl/RadHAR (UCLA : networked & embedded systems lab)

이미 공개되어있는 pcl 데이터 확인



- Human joint 개수 : 25개
- Frame : 600 이 되도록 전처리가 필요하다는것을 확인했다.

< Radar data >



1초 안에 있는 pcl data를 찍어 봤을때의 해당 figure이다.

보통 1초안에 20fps (frame rate)가 나오게 끔 configure값을 조정했다.

< Kinect data >

1 frame 안에 있는 skeleton data figure

구현 과정 (0) – 학습 아이디어 도출 및 사례조사1

전처리 된 radar dataset에 대해 gcn, st-gcn 모델 학습 수행

Table 2. Test accuracy on the RadHAR dataset

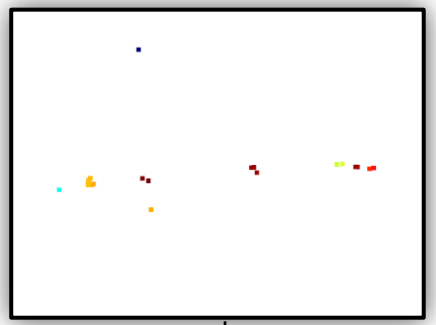
	Accuracy
SVM	63.74
MLP	80.34
Bi-directional LSTM	88.42
Time-distributed CNN + Bi-directional LSTM	90.47
GCN(Graph Convolutional Networks)	46.55
Ours(ST-GCN based)	62.07

Kinetic Data를 Ground truth값으로 배치한 뒤, Radar 데이터를 해당 값에다 group 화 시키는 feature값들을 추출

By Kinect : 32 joint

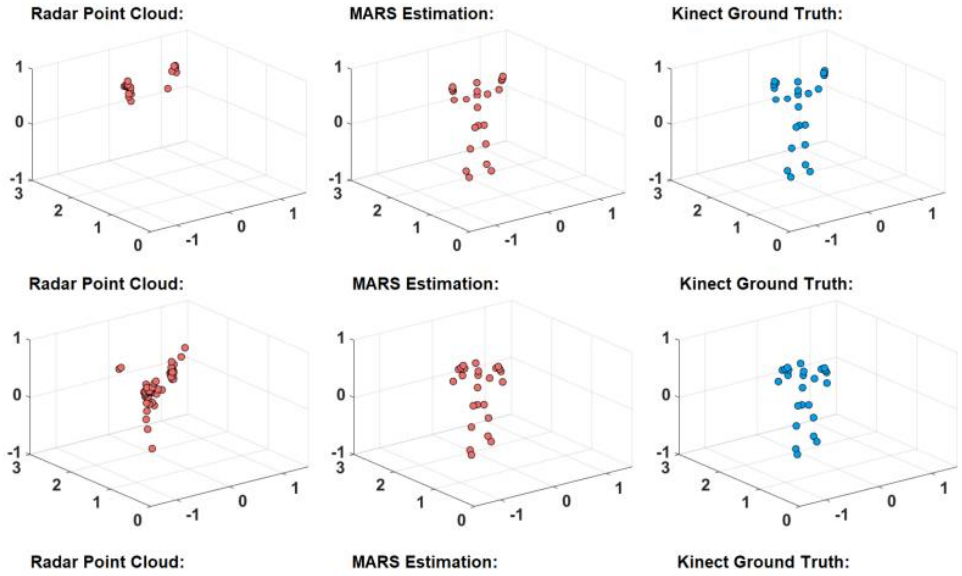
Mapping

By Radar : 20 pcl data



1) mapping 할 point 개수 몇개로 맞출 건인지 ?
2) mapping을 위한 모델은 어떻게 만들 것인지?

구현 과정 (0) – 학습 아이디어 도출 및 사례조사2

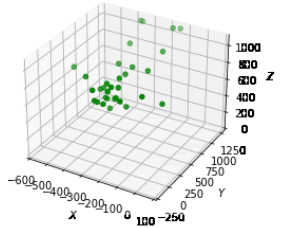


출처 : https://github.com/SizheAn/MARS

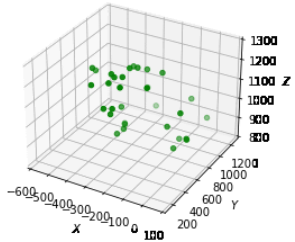
Kinetic Data를 Ground truth값으로 배치한 뒤, Radar 데이터 맵핑

Radar에서 나오게 되는 mmWave Radar값이 포인트 클라우드 전처리를 통하여 5개의 feature 이 존재하는 channel들이 생성됩니다. 이후 kinectv4를 통해 나온 3D 스켈레톤 값에 대한 데이터 정렬과 레이블링을 훈련 데이터로 설정하게 됩니다.

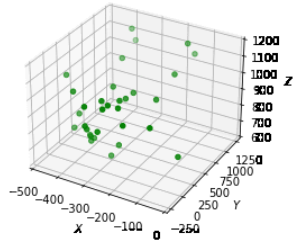
구현 과정 (0) – 키넥트 데이터 only figure plt 시켰을 때



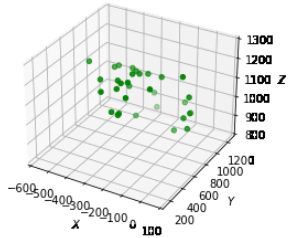
Running



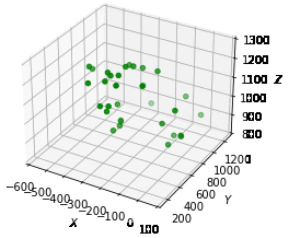
Sitting



Standing

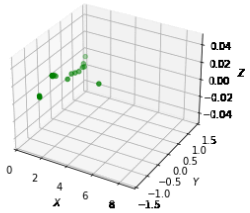


Squat

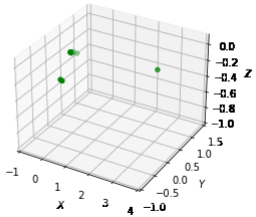


Walking

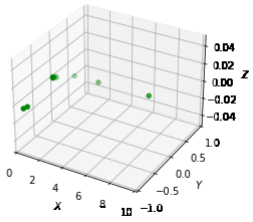
구현 과정 (0) – 레이더 데이터 only figure plt 시켰을 때



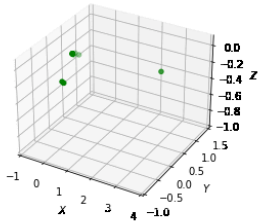
Running



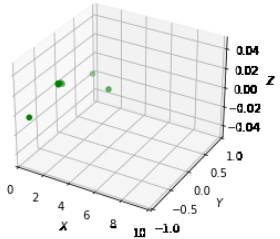
Sitting



Standing

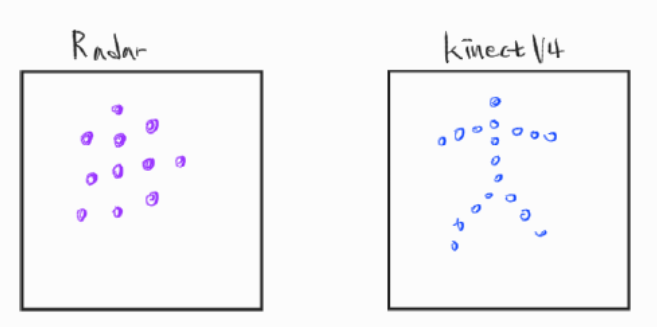


Squat

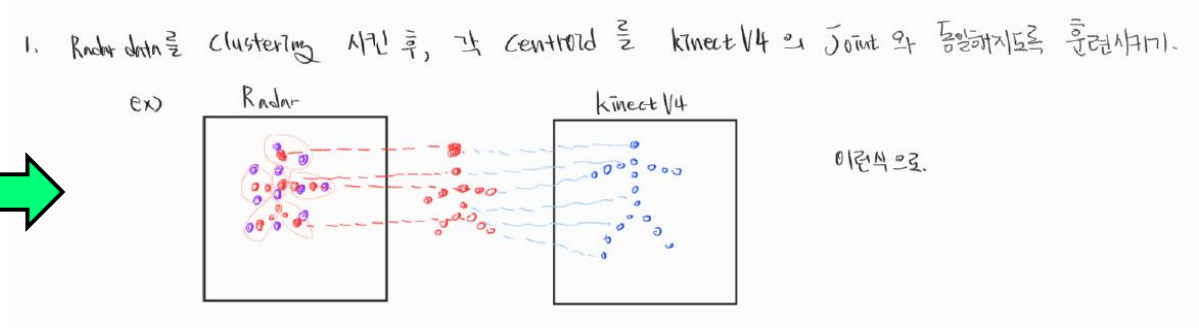


Walking

구현 과정 (0) – 학습 아이디어 도출 및 사례조사 적용



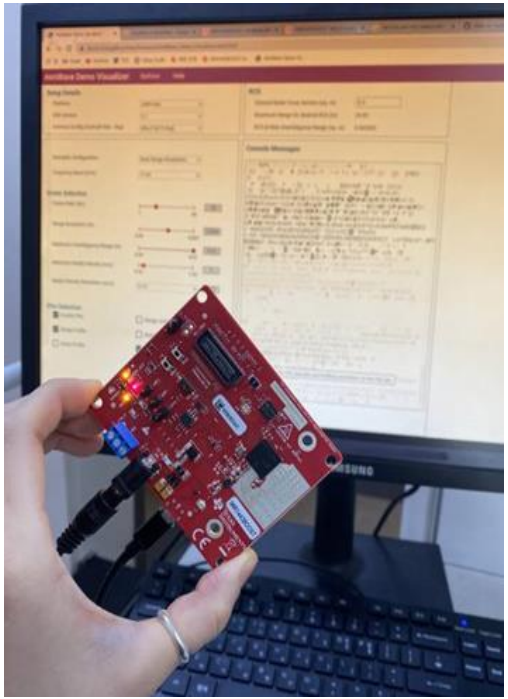
Radar Data와 Kinect data의 맵핑이 꼭 필요하다.

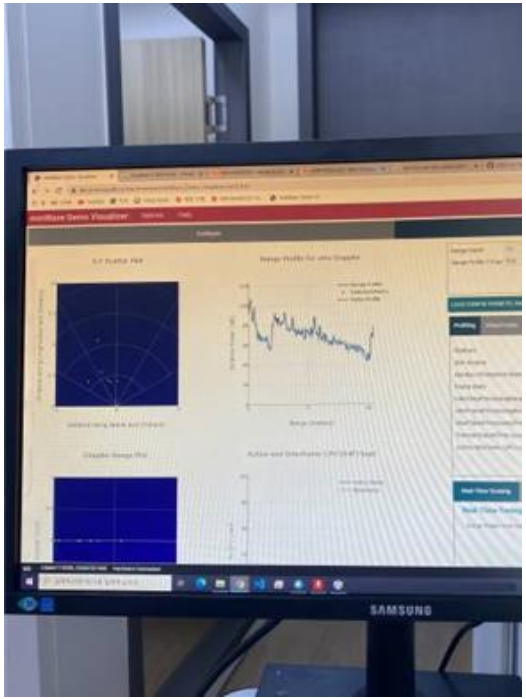


[결과적으로 필요한 데이터 :]

1. 키넥트 데이터
2. 레이더 데이터
3. 충분한 학습을 돌릴 수 있는 서버와 GPU
4. 많은 데이터와 학습량

구현 과정 (1) – 데이터 수집 : 레이더 셋팅



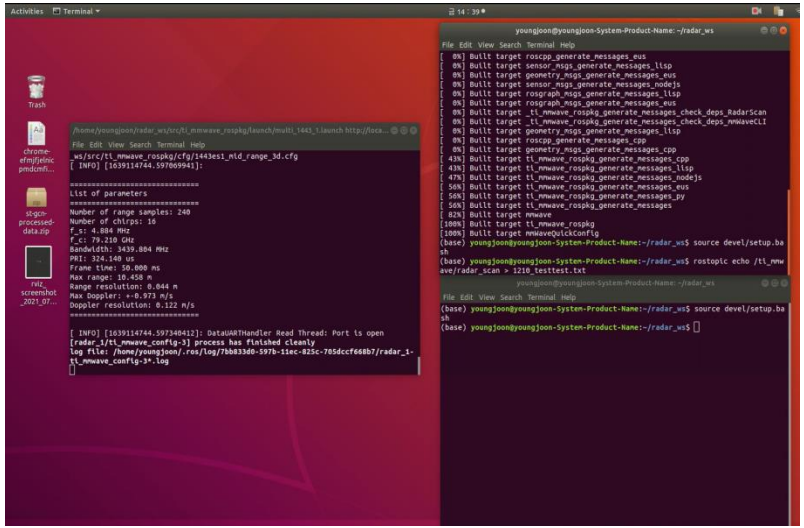


Texas Instrument 제공
API를 사용하기 위해
꽤 많은 단계를 거쳐 레이더 셋팅을 진행한다.

① IWRBOOST1443 모델과 5V 어댑터를 연결한다.
② Windows OS 에서 Ti Cloud, CCS (code composer stdio), CCS (code composer stdio), TI Agent를 설치한다
③ 레이더 셋팅 (데모 프로젝트 코드 빌딩)을 한다.
④ 빌드가 완료되면 디버그 모드(ccs활용)를 통해 EVM을 준비한다. CCS 디버그 펌웨어를 플래시한다.
⑤ Uniflash 툴을 통해 SOP0과 SOP2에 모두 점퍼를 꼽은 뒤, binary image를 업로드 하고 SUCCESS가 뜨게 되면 점퍼를 분리시킨다.

구현 과정 (1) – 데이터 수집 : 레이더 데이터 수집 (ROS)

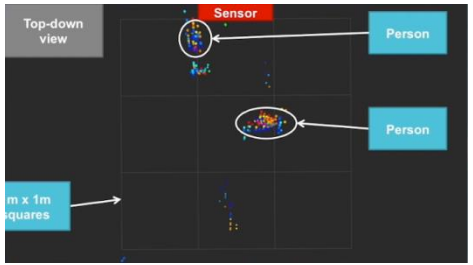
사용 OS : Ubuntu 18.04 / ROS 사용



-> UART 통신을 통해 레이더와 통신을 하며, 간편하게 customized 된 msg파일로 데이터를 받아올 수 있어 사용했다.

```
header:
  seq: 6264
  stamp:
    secs: 1538888235
    nsecs: 712113897
  frame_id: "ti_mmwave"   # Frame ID used for multi-sensor scenarios
point_id: 17              # Point ID of the detecting frame (Every frame starts with 0)
x: 8.650390625            # Point x coordinates in m (front from antenna)
y: 6.92578125             # Point y coordinates in m (left/right from antenna, right positive)
z: 0.0                    # Point z coordinates in m (up/down from antenna, up positive)
range: 11.067276001       # Radar measured range in m
velocity: 0.0             # Radar measured range rate in m/s
doppler_bin: 8            # Doppler bin location of the point (total bins = num of chirps)
bearing: 38.6818885803    # Radar measured angle in degrees (right positive)
intensity: 13.6172780991  # Radar measured intensity in dB
```

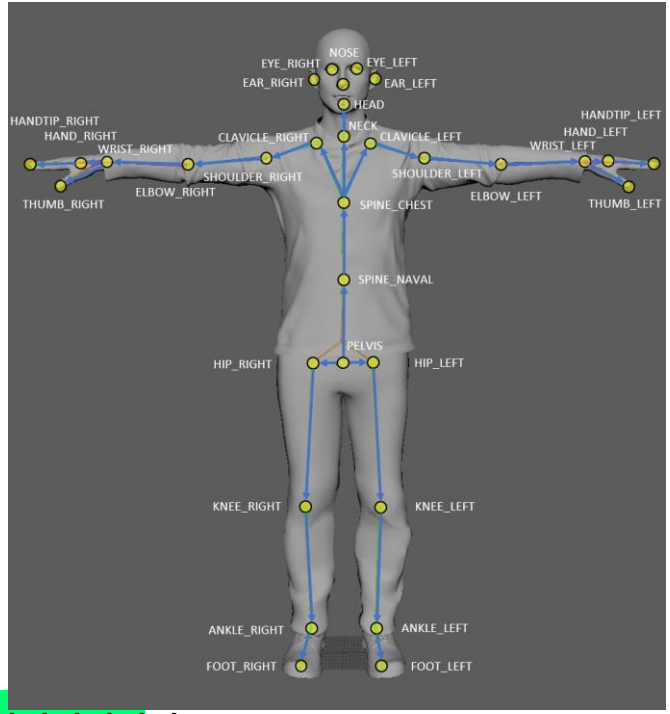
-> 해당 포맷 (시퀀스, timeStamp, frame id, point cloud id, x,y,z, range, velocity, doppler_bin, bearing, intensity 값을 가져올 수 있다.



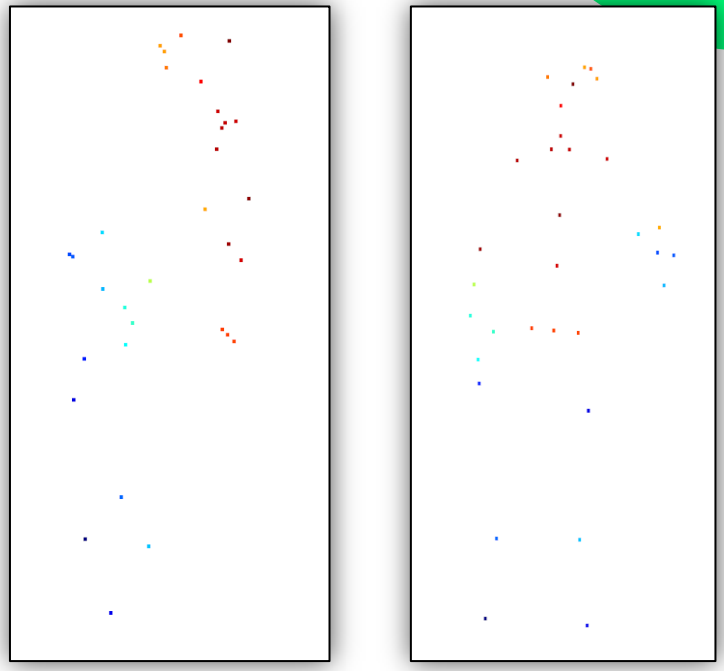
좌측 사진과 동일하게 RVIZ라는 우분투 visualizer에서도 pcl 데이터가 찍히는것을 볼 수 있다.

구현 과정 (1) – 데이터 수집 : 레이더 데이터 수집 (ROS)

사용 OS : Ubuntu 18.04 / ROS 사용

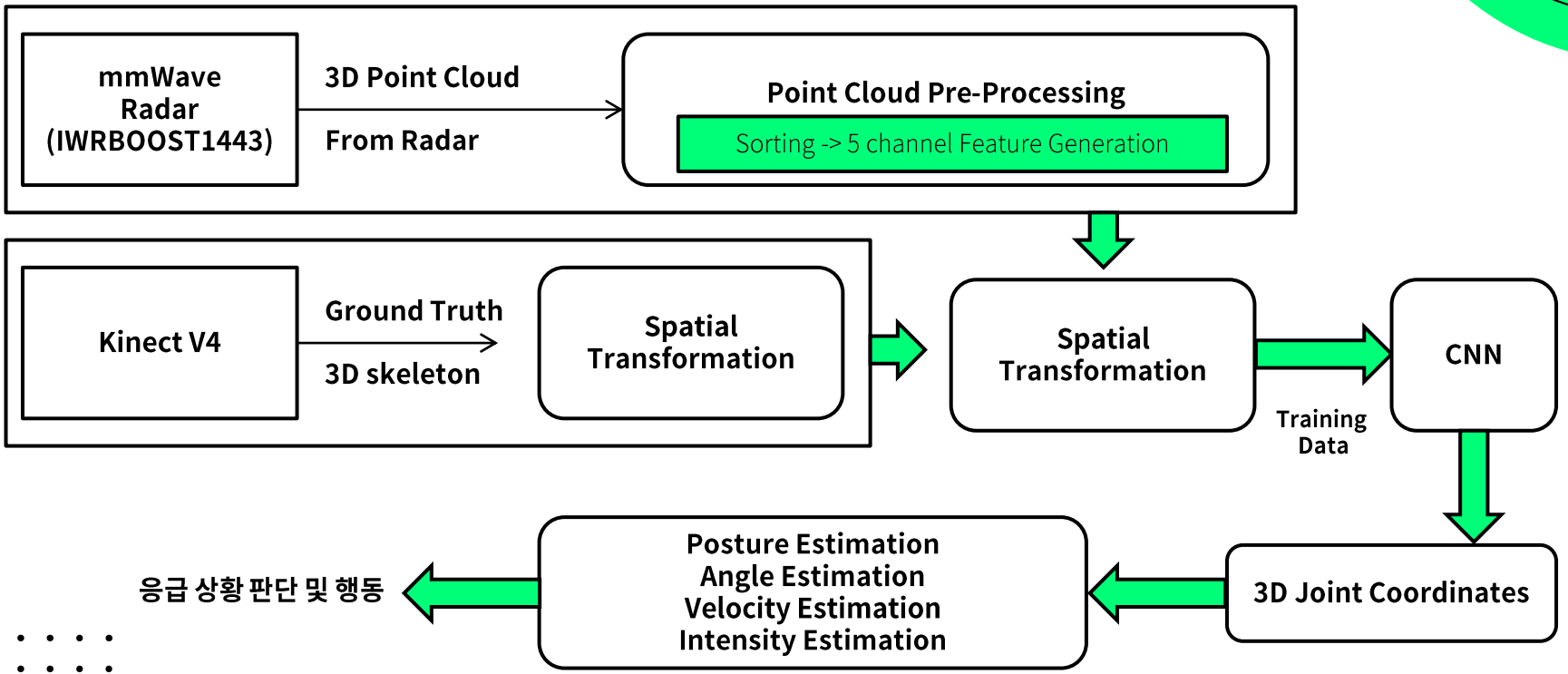



```
ply
format ascii 1.0
element vertex 32
property float x
property float y
property float z
end_header
25.0294 281.949 866.475
33.1233 117.283 922.201
39.4982 -16.6183 959.429
33.6986 -225.728 926.428
67.5222 -192.261 931.107
205.539 -173.159 925.523
381.812 -5.38565 809.579
268.837 -1.7177 613.594
313.515 30.5187 537.519
352.372 110.129 594.633
365.306 35.8833 545.578
1.99979 -190.416 934.669
-121.606 -156.192 937.301
-252.522 84.9035 950.419
-248.897 146.342 729.731
-250.138 206.844 648.697
-222.636 305.515 633.162
-177.864 244.823 659.926
113.872 285.547 864.203
116.158 369.533 482.838
115.138 723.337 601.607
134.677 864.98 487.809
-55.0836 278.704 868.524
-206.04 325.478 511.881
-150.122 696.323 551.588
-171.596 811.649 431.142
30.9027 -301.286 901.279
146.805 -363.672 816.168
128.971 -392.582 858.913
75.4891 -365.547 967.083
103.048 -388.797 815.155
-20.2807 -363.051 836.639
```

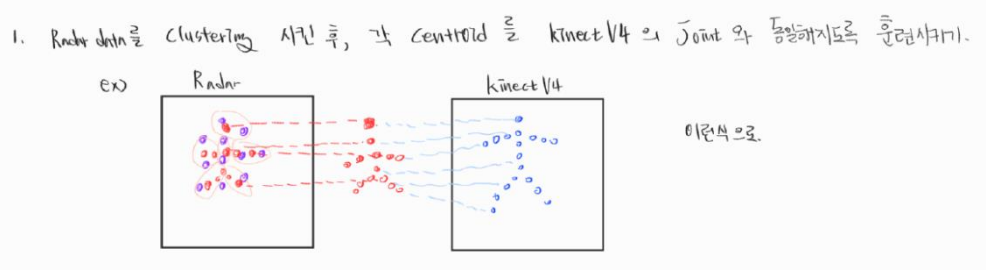


레이더와 비슷한 포맷으로
time stamp, skeleton 의 x,y,z 위치를 알 수 있다
위 사진은 각각 정면과 측면의 skeleton data이다.

구현 과정 (2) – 학습 과정 및 플로우 구축

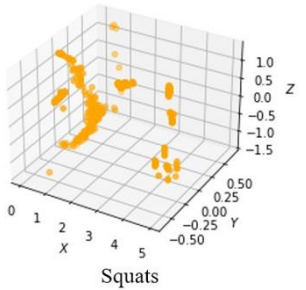


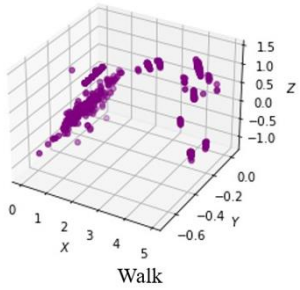
구현 과정 (2) – 학습과정 중 우려했던 점



[우려했던 점]

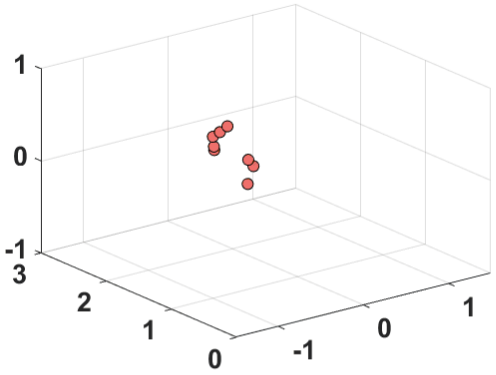
1). radar를 통해 얻은 데이터는 오른쪽 형태와 같다. 이 데이터의 clustering을 가정해 보았을 때, skeleton data(joint) 처럼 나타날 것 같지 않다고 판단 됨.
2). radar data의 수가 clustering 을 해서 25개의 centroid를 잡을 만큼 많이 잡히지 않음.
3) 결국 test할 때 쓰는 data는 radar data 이라서, Kinect data를 concatenate한 training data가 오히려 noise일 수도 있다고 생각함.
실험결과:
Jump 에 해당되는 skeleton data를 radar data와 concate 시켜서 본 결과를 보면 concate 한 게 오히려 안 좋음



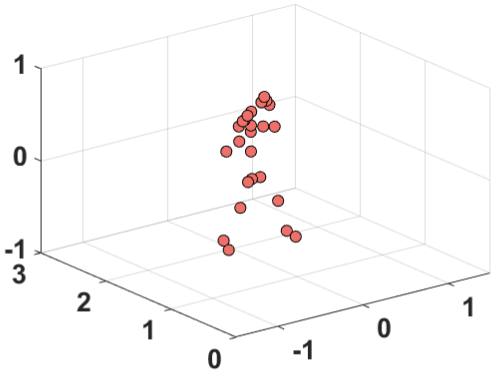


구현 과정 (2) – 목표 figure plt 이미지

Radar Point Cloud:



MARS Estimation:

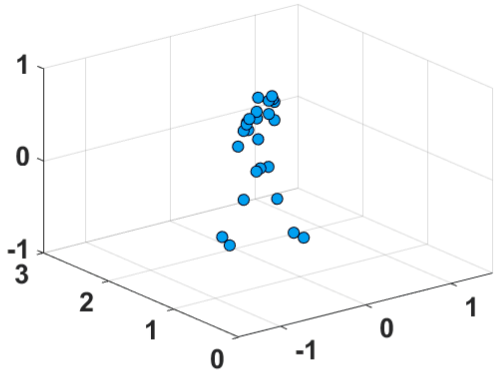


Left elbow: 41° Right elbow: 52°

Left knee: 170° Right knee: 169°

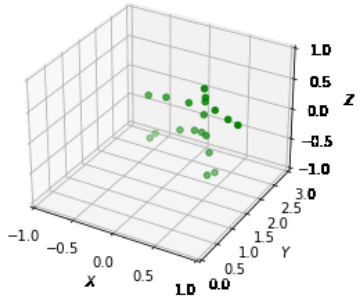
Frame: 4635

Kinect Ground Truth:

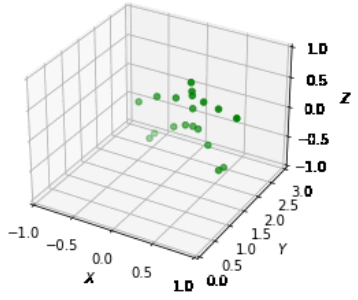


구현 과정 (2) – 학습 과정 및 플로우 구축

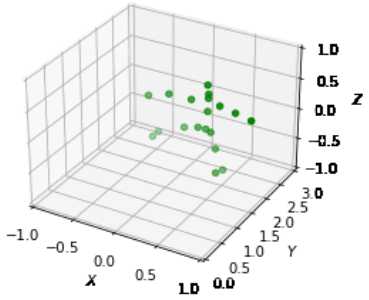
아무런 수정 없이 데이터 plot 시켜봤을 때



Sitting



Squat



Walking

구현 과정 (2) – 변경 된 사항 및 발전 사항

기존 Input 모델 사이즈 수정

기존 참고했던 논문들의 input size는 xyz, intesity, doppler_bin까지 총 5개의 layer로, 8*8*5를 이용하고 있었지만, 현재 레이더상 intensity값이 동일하다는 점, 효율적인 학습을 위해 수정이 필요하다는 점을 통해 8*8*5에서 xyz값만을 사용하는 8*8*3을 사용하기로 수정.

기존 RADHAR 깃허브 데이터를 통해 전처리 일반화 하기

x,y,z 받아와서 64개 row 를 만들되, RadHar의 경우 16~26개 정도밖에 data가 되지 않으므로(64개에 한참 못미침) data 분포를 보고 (예를 들면 가우시안 같은) 데이터가 따르는 분포의 주변값들로 64개의 row가 채워지도록 RadHar데이터 전처리 후, 역시 64x3 의 형태로 완성할 것.

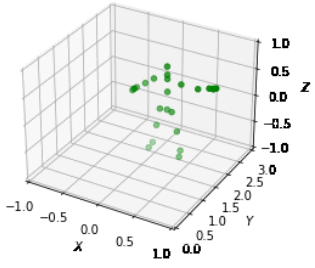
키넥트 데이터와 레이더데이터 개수가 맞지 않는 상황

x,y,z 받아와서 25개 row 를 만들되, RadHar의 경우 16~26개 정도밖에 data가 되지 않으므로(64개에 한참 못미침) 2번째 task의 경우 25row(25개 이상이면 자르고 25개 미만이면 zero padding시킬 것) 25x3 의 형태로 완성할 것

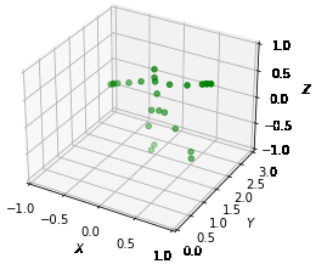
전처리 및 학습이 된 포인트 클라우드들의batch size와 epoch를 바꿔가며 포인트 클라우드의 figure을 plot화 해보기

구현 과정 (2) – 학습 과정 및 플로우 구축

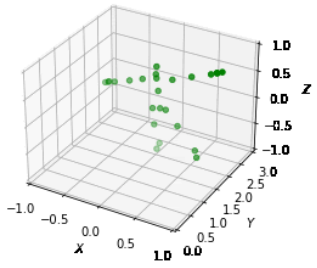
4*4 shape 수정 -> Boxing, Jack, Lying



Boxing



Jumping Jack

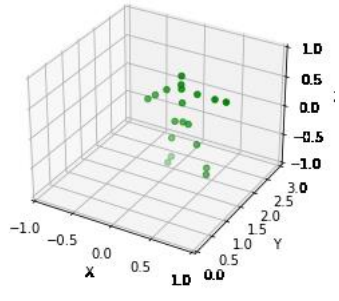


Lying

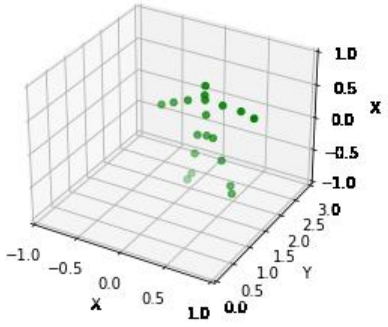
Exp -1. 실험을 통해 알아낸 Best parameter로 mapping test

1) 8x8

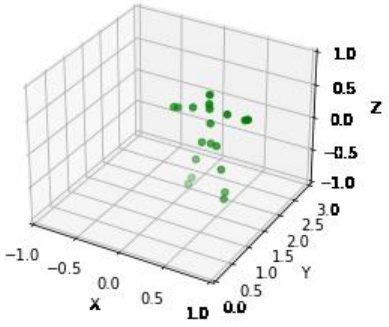
walk



squats

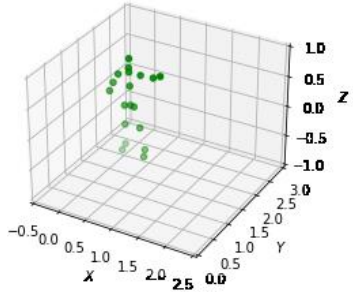


sitting

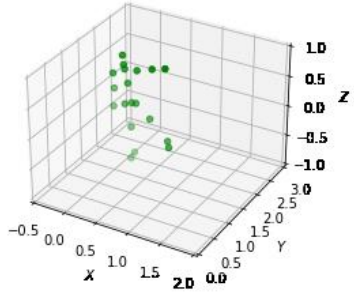


2) 6x6

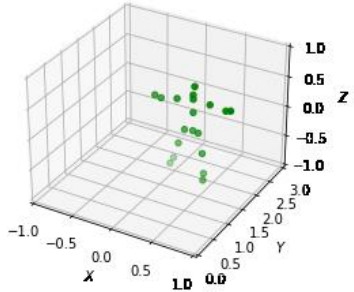
walk



squats

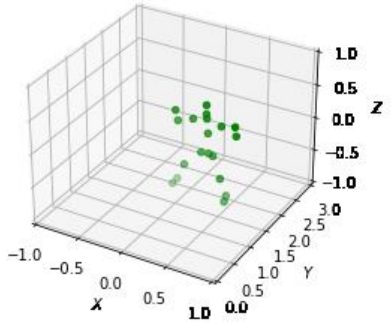


sitting

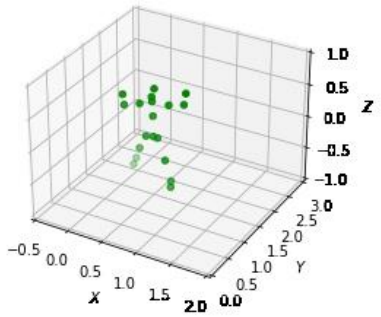


3) 5x5

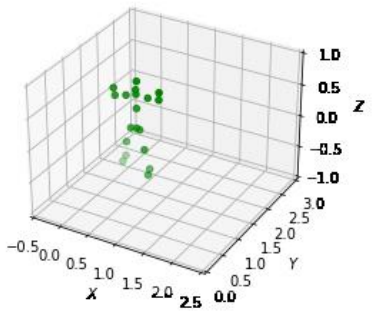
Our sitting



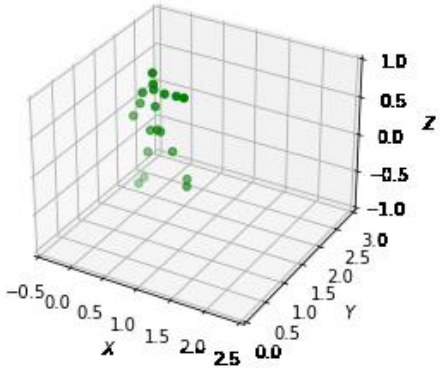
RadHar squats



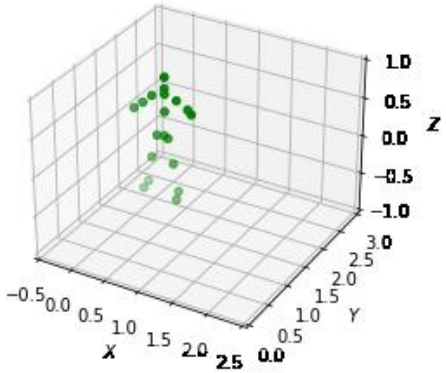
RadHar walk



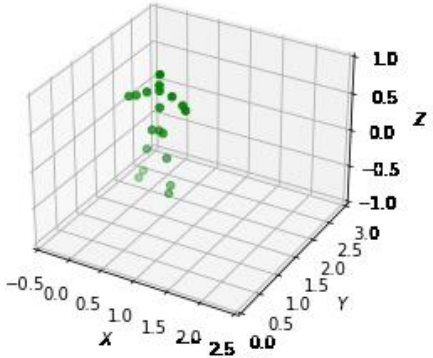
RadHar box



RadHar jum



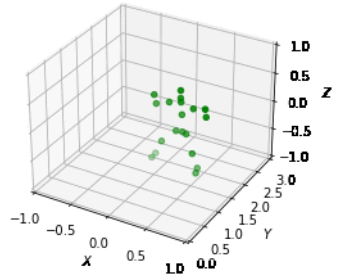
RadHar jack



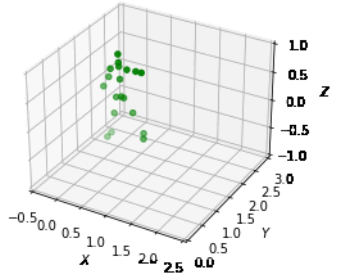
구현 과정 (2) – 학습 과정 및 플로우 구축

5*5 shape 수정 / epoch 100 / batch size 64

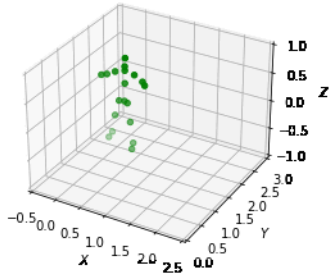
Accuracy : 47.6819%



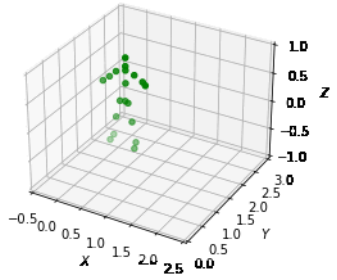
Sitting



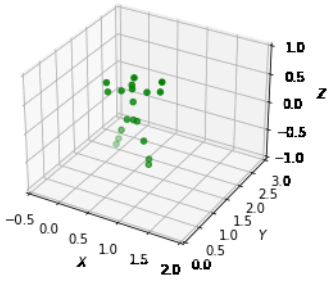
squats



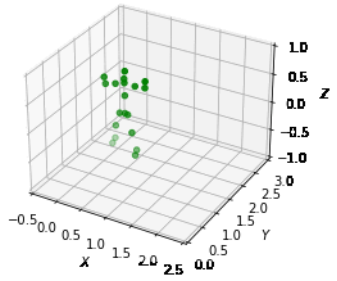
Walking



Boxing



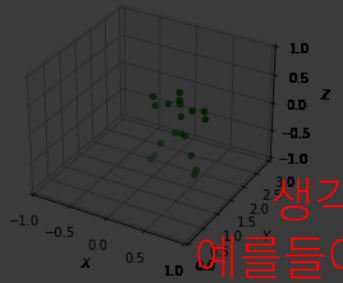
Jumping



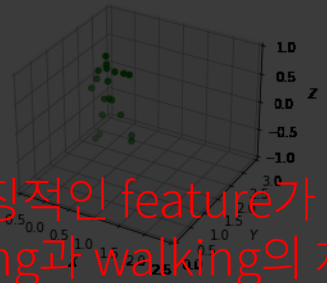
Jumping Jack

구현 과정 (2) – 학습 과정 및 플로우 구축

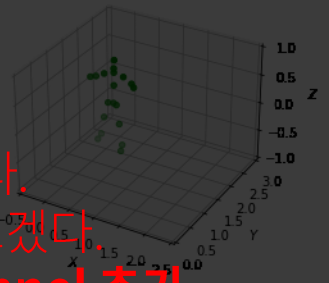
5*5 shape 수정 / epoch 100 / batch size 64



Sitting

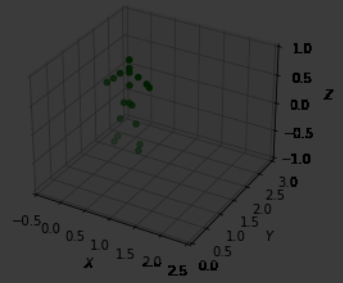


squats

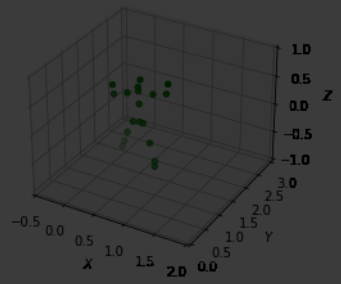


Walking

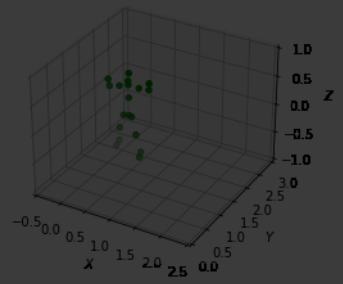
생각보다 특징적인 feature가 잡히지 않는다.
예를들어 jumping과 walking의 차이점을 모르겠다.
-> batch size 수정, epoch증가, intensity channel 추가



Boxing



Jumping

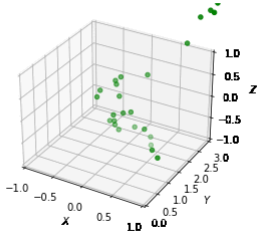


Jumping Jack

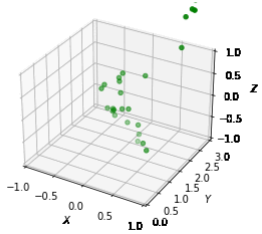
구현 과정 (2) – 학습 과정 및 플로우 구축

5*5 shape 수정 / epoch 120 / batch size 32 ++ intensity channel 추가

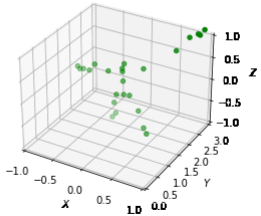
Accuracy : 18.64320%



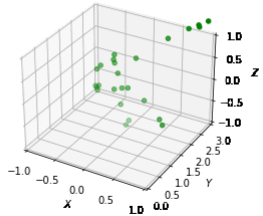
Sitting



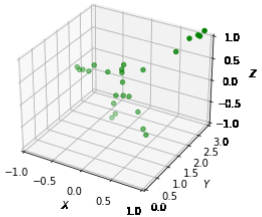
squats



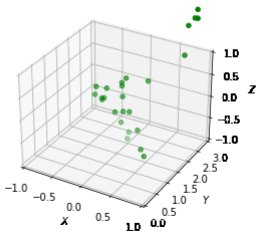
Walking



Boxing



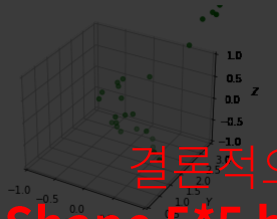
Jumping



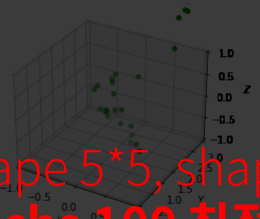
Jumping Jack

구현 과정 (2) – 학습 과정 및 플로우 구축

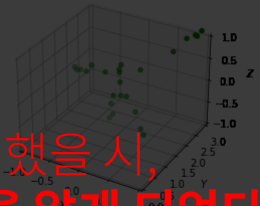
5*5 shape 수정 / epoch 100 / batch size 64 ++ intensity channel 추가



Sitting



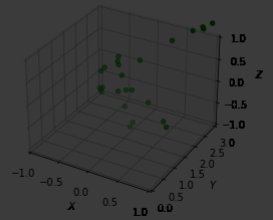
squats



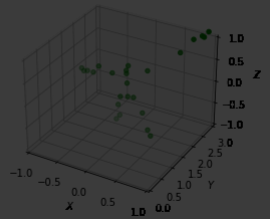
Walking

결론적으로 shape 6*6, shape 5*5, shape 4*4를 진행했을 시,

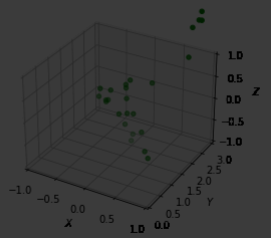
Shape 5*5 batch size 64 epochs 100 최적화 되어있음을 알게 되었다.



Boxing



Jumping

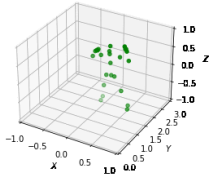


Jumping Jack

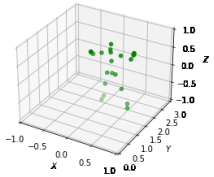
구현 과정 (2) – 학습 과정 및 플로우 구축

```
#define the model - 2.layer 추가
def define_CNN(in_shape, n_keypoints):


    in_one = Input(shape=in_shape)
    conv_one_1 = Conv2D(16, kernel_size=(3, 3), activation='relu', strides=(1, 1), padding = 'same')(in_one)
    conv_one_1 = Dropout(0.3)(conv_one_1)
    conv_one_2 = Conv2D(32, kernel_size=(3, 3), activation='relu', strides=(1, 1), padding = 'same')(conv_one_1)
    conv_one_2 = Dropout(0.3)(conv_one_2)
    conv_one_2 = BatchNormalization(momentum=0.95)(conv_one_2)


    fe = Flatten()(conv_one_2)
    # dense1
    dense_layer1 = Dense(512, activation='relu')(fe)
    dense_layer2 = Dense(256, activation='relu')(dense_layer1)
    dense_layer2 = BatchNormalization(momentum=0.95)(dense_layer2)
    # # dropout

    # dropout
    dense_layer2 = Dropout(0.4)(dense_layer2)

    out_layer = Dense(n_keypoints, activation = 'linear')(dense_layer2)


    # model
    model = Model(in_one, out_layer)
    opt = Adam(lr=0.001, beta_1=0.5)

    # compile the model
    model.compile(loss='mse', optimizer=opt, metrics=['mae', 'mse', 'mape', tf.keras.metrics.RootMeanSquaredError()])
    return model
```

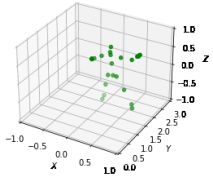
Accuracy : 47.6810%



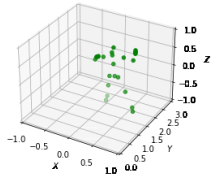
Boxing



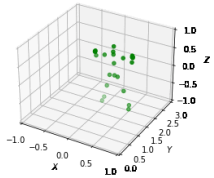
Squat



Jack

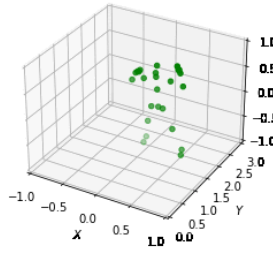


Walking

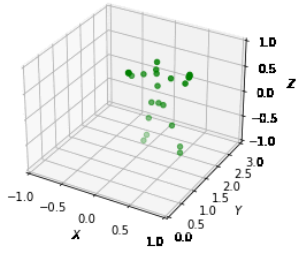


Jumping

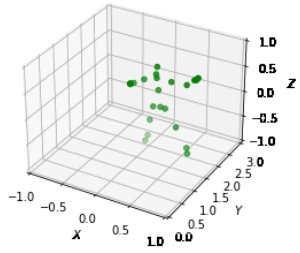
구현 과정 (2) – 학습 과정 및 플로우 구축



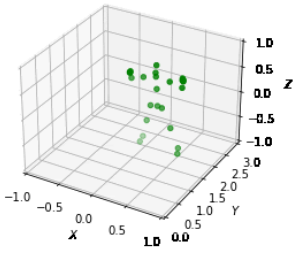
Boxing



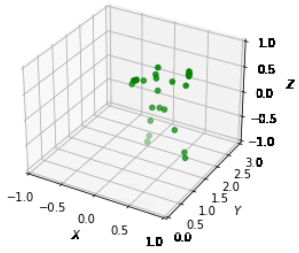
Squat



Jack



Walking



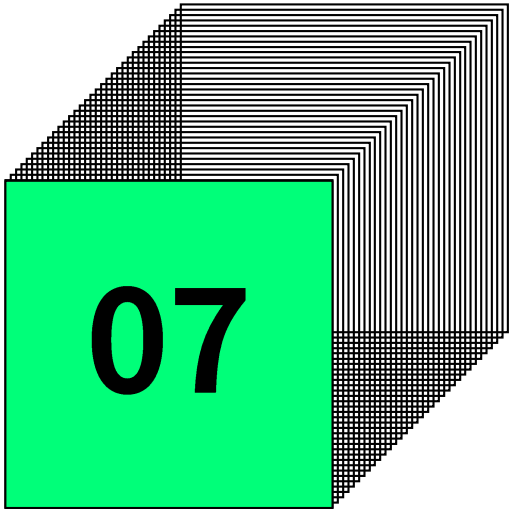
Jumping

12월 10일 기준 최종 학습 결과

Accuracy : 47.6810%

Accuracy 를 높이기 위한 방법을 찾아야 한다.

- 레이더 데이터와 키넥트 데이터 Mapping 에 현재 CNN 모델을 사용하지만, CNN + LSTM 모델을 합성하면 어떨까 하는 시도를 해보고자 한다.



프로젝트 결과

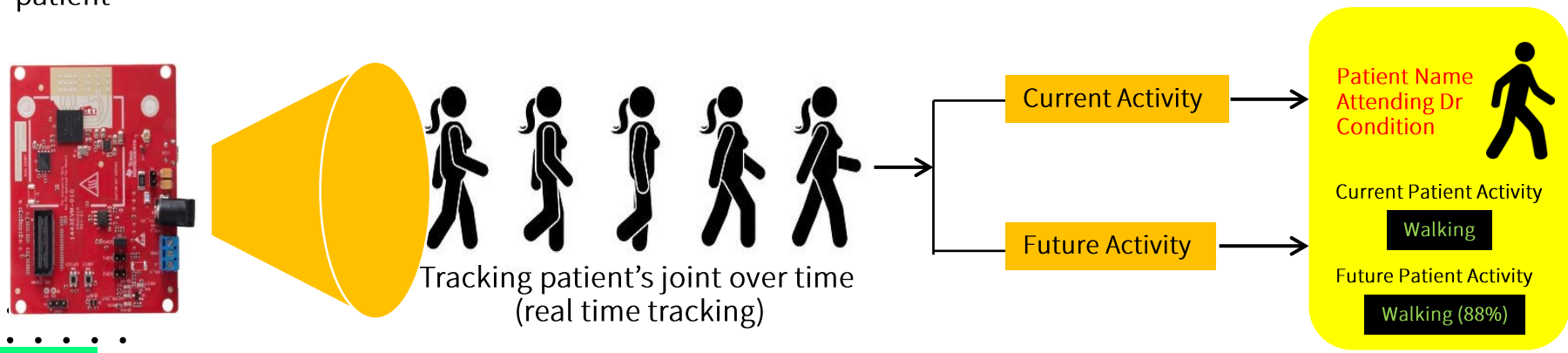
프로젝트 추진 일정

	9월	10월	11월	12월
주제 선정(기획)				
레이더 셋팅 및 하드웨어 셋팅				
학습 알고리즘 레퍼런스				
데이터 셋 학습				

	12월	1월	2월	비고
Mapping 방법론 수정 및 학습 진행				
학생 모집 및 데이터 수집				
데이터 학습 및 GUI제작				
논문 작성				

프로젝트 결과

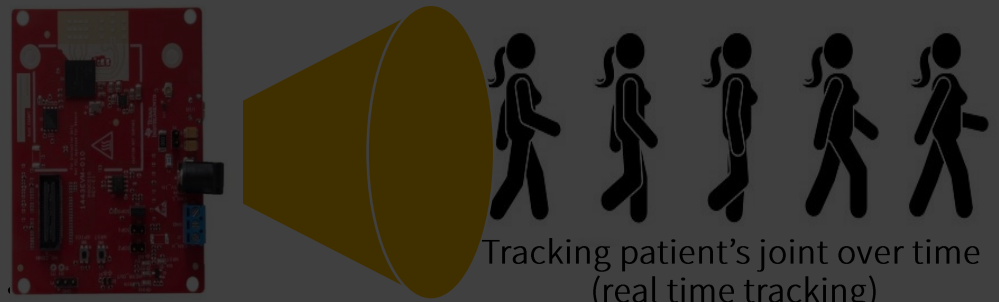

patient
Joint 1
Joint 2
Joint 3
Joint 23
Joint 24
Joint 25
Hidden Layers
mmPose
Architecture
y
x
z
3–D Skeleton


Tracking patient's joint over time
(real time tracking)
Current Activity
Future Activity
Patient Name
Attending Dr
Condition
Current Patient Activity
Walking
Future Patient Activity
Walking (88%)

프로젝트 결과

patient

최종 레이더 activity 탐지 notice를 위한 GUI제작



Current Activity

Future Activity

Patient Name
Attending Dr
Condition

Current Patient Activity

Walking

Future Patient Activity

Walking (88%)

프로젝트 결과 및 추후 계획 설명

1. Tensorflow의 아쉬운 점 : Pytorch로 변경해보며 추이를 지켜보고자 한다.

2. Pyqt를 활용한 GUI 제작 : 최종 GUI를 제작하고자 한다.

3. mapping feature 학습에 대한 방법론 Accuracy를 높이기 위해 mapping 알고리즘 모델의 방법론을 변경하고자 한다.

4. 학습 데이터 셋 모집 (학생 모집) Accuracy을 높이기 위해선 더 많은 데이터 셋이 필요하다 -> 학생들을 모집중에 있다.

이상 발표 마치겠습니다.
감사합니다.

멀티미디어공학과
2018113627
정경은

How To ? – 간단 구조

1

Model pre 트레이닝

[모델 training시의 키넥트 센서 추가]

모델의 정확도를 높이기 위해 다른 센서
(키넥트 v4)의 데이터를 추가 활용하고자 한다.

2

Model 정밀학습

[레이더를 통한 모델 정밀학습]

레이더 데이터 중,
Intensity, boundary 값을 기준으로
정밀학습을 진행한다.

3

학습 완료 모델

[레이더를 통한 행동 파악]

Radar data 하나만 사용해
훈련시킨 모델보다 행동 및
자세 인식률을 향상시킬 수 있다.