Inertial + Floorplan Localization Using CRF

Paper

This implementation is based on the researches done on the following papers,

[1] Z. Xiao, H. Wen, A. Markham, και N. Trigoni, ‘Lightweight map matching for indoor localisation using conditional random fields’, στο IPSN-14 proceedings of the 13th international symposium on information processing in sensor networks, 2014, σσ. 131–142.

[2] J. Zhang, M. Ren, P. Wang, J. Meng, και Y. Mu, ‘Indoor localization based on VIO system and three-dimensional map matching’, Sensors, τ. 20, τχ. 10, σ. 2790, 2020.

Note that due to unavailability of exact dataset used for above researchers, I had to use following dataset and convert that accordingly.

[3] S. Herath, S. Irandoust, B. Chen, Y. Qian, P. Kim, και Y. Furukawa, ‘Fusion-DHL: WiFi, IMU, and Floorplan Fusion for Dense History of Locations in Indoor Environments’, στο 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, σσ. 5677–5683.

Theory

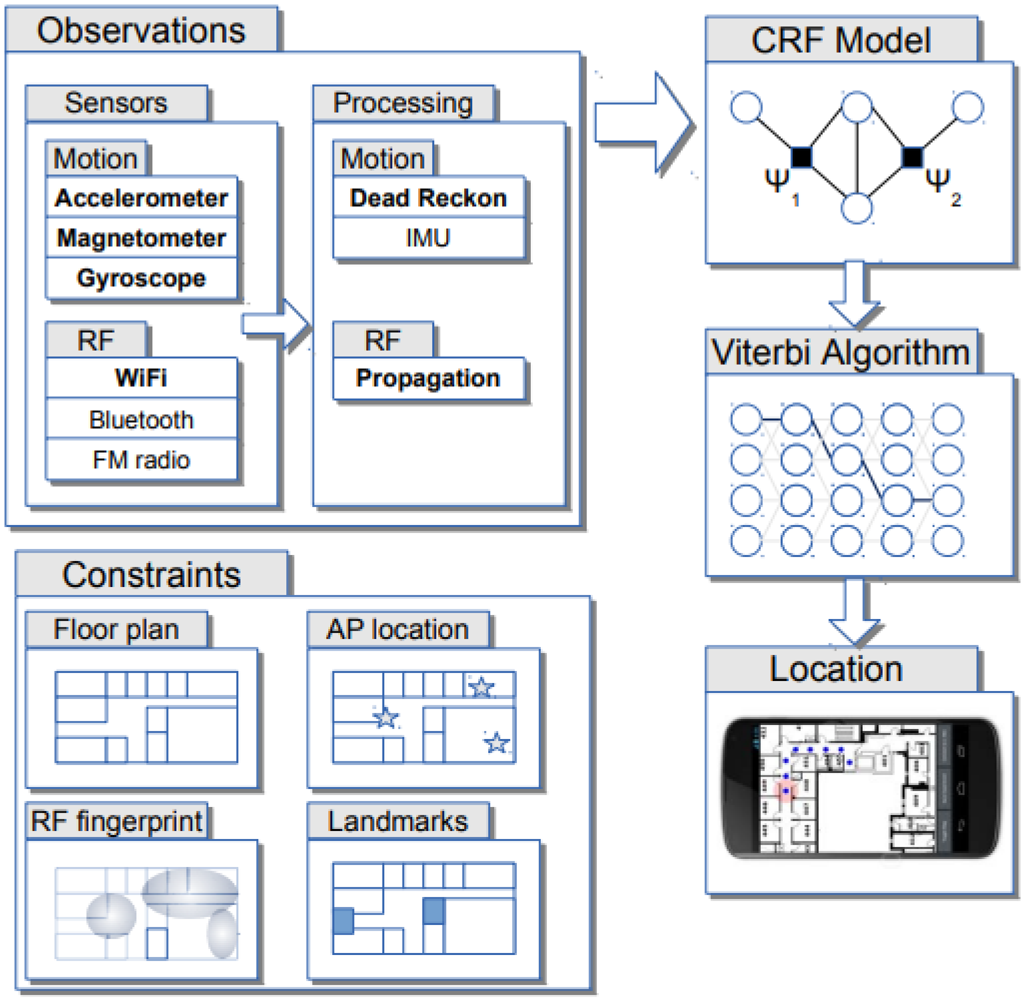
In this notebook, I am going to implement indoor localization mechanism using Linear Chain Conditional Random Fields. By using this model we can predict location of a user when starting position, IMU observations (velocity vectors) and floorplan of the building is given.

Inputs,

- Stating Position - (Meters In X direction(TopLeft|LeftRight), Meters In Y direction(TopLeft|TopBottom))
- Sequence of Velocity Vectors Captured In Small Time Range (20 seconds) : Velocity_Values(ms-1), Velocity_Angles(radian)
- Graph of Floorplan

Overall Architecture

Here is the overall system architecture



The input is a velocity vector observed using IMU data Z = {Z0,...,ZT }, and the task is to predict a sequence of states S = {S0,...,ST } given input Z.

Viterbi Algorithm

We use Viterbi algorithm, which can dynamically solve the optimal state points sequence that is most likely to produce the currently given observation value sequence. The solution steps of Viterbi algorithm are as follows:

(1) Initialization: Compute the non-normalized probability of the first position for all states, where m is the number of states.

$$\delta_1(j) = w \cdot F_1(y_0 = start, y_1 = j, x) \ j = 1, 2, \cdots m,$$

(2) Recursion: Iterate through each state from front to back, find the maximum value of the non-normalized probability of each state l = 1, 2, · · · , m at position i = 2, 3, · · · , n, and record the state sequence label Ψi(l) with the highest probability.

$$\delta_i(l) = \max_{1\leq j\leq m}\{\delta_{i-1}(j) + w\cdot F_i\ (y_{i-1} = j, y_i = l, x)\}\ l = 1, 2, \cdots m,$$

$$\Psi_i(l) = \underset{1\leq j\leq m}{argmax}\{\delta_{i-1}(j) + w\cdot F_i\ (y_{i-1} = j, y_i = l, x)\}\ l = 1, 2, \cdots m,$$

(3) When i = n, we obtain the maximum value of the non-normalized probability and the terminal of the optimal state points sequence

$$\max_y(w\cdot F(y, x)) = \max_{1\leq j\leq m}\delta_n(j),$$

$$y_n^* = \underset{1\leq j\leq m}{argmax}\delta_n(j),$$

(4) Calculate the final state points output sequence

$$y_i^* = \Psi_{i+1}\left(y_{i+1}^*\right)i = n-1, n-2, \cdots, 1,$$

(5) Finally, the optimal sequence of state points is as follows:

$$y^* = \left(y_1^*, y_2^*, \cdots, y_n^*\right)^T.$$

Defined F and W

We can use w and F(y, x) to represent the weight vector and the global state transfer function vector.

$$w = (w_1, w_2, \cdots, w_K)^T$$
$$F(y, x) = (f_1(y, x), f_2(y, x), \cdots f_K(y, x))^T\ I(y_{i-1}, y_i)$$

where I(Yt−1, Yt) is an indicator function equal to 1 when states Yt−1 and Yt are connected and 0 otherwise.

We use two functions f1 anf f2

$$f_1\left(y_t, y_{t-1}, x_t^d\right) = Ln\frac{1}{\sigma_d\sqrt{2\pi}} - \frac{\left(x_t^d - d(y_{t-1}, y_t)\right)^2}{2\sigma_d^2}$$

where xdt is the Euclidean distance between two consecutive observations, d(yt−1, yt) is the Euclidean distance between two consecutive state points, and σ2d is the variance of the distance in the observation data.

$$f_2\left(y_t, y_{t-1}, x_t^\theta\right) = Ln\frac{1}{\sigma_\theta\sqrt{2\pi}} - \frac{\left(x_t^\theta - \theta(y_{t-1}, y_t)\right)^2}{2\sigma_\theta^2},$$

where xθt is the orientation of two consecutive observations, θ(yt−1, yt) is the orientation between two consecutive state points, and σ2θ is the variance of the orientation in the observation data.

Load Libraries

In [1]:
```
import pandas as pd
from matplotlib import image
from matplotlib import pyplot as plt
from math import cos, asin, sqrt, pi, atan2
```

Data Preprocessing

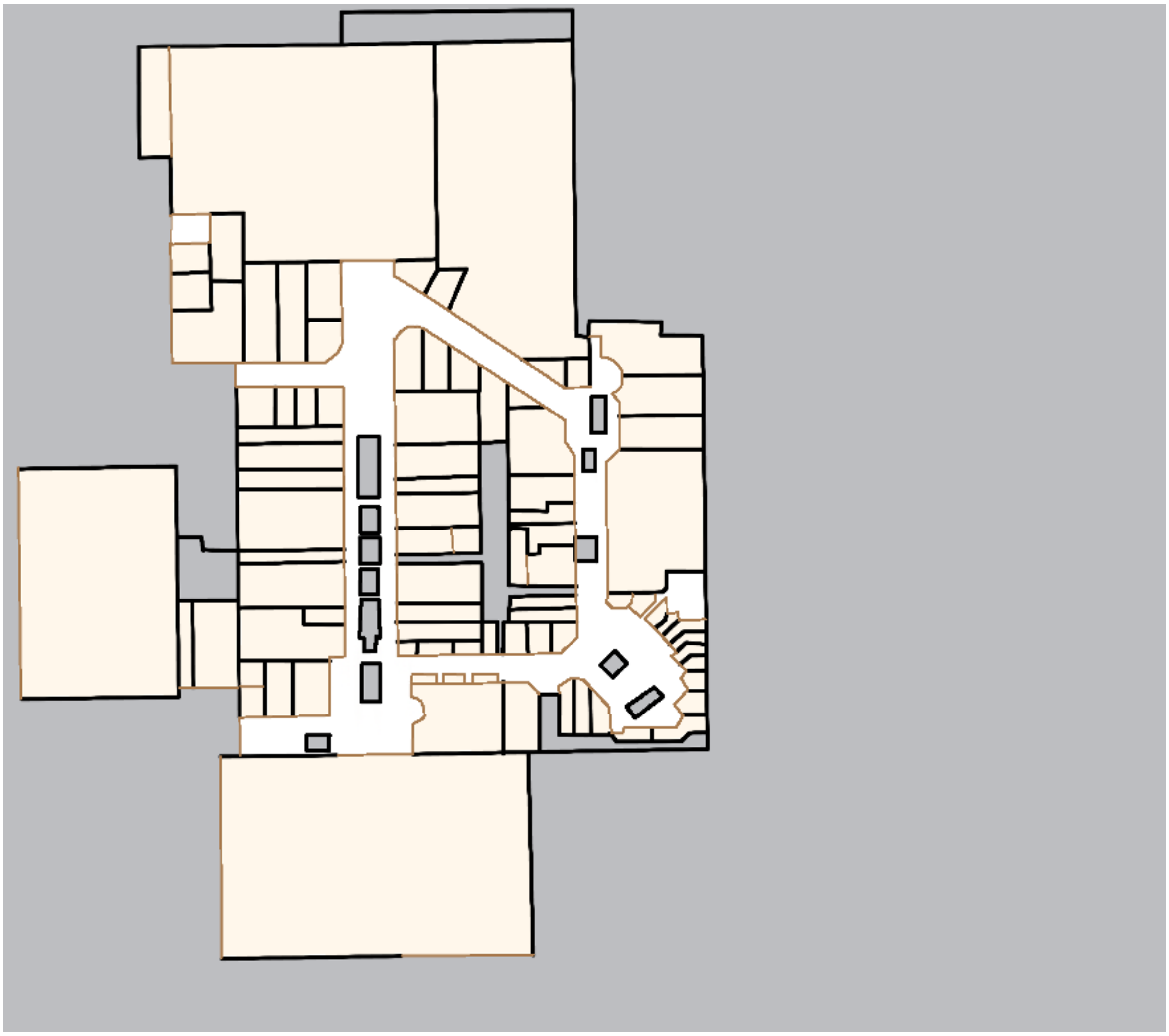
Here are units we use,

- location coordinates (x,y) : In pixels
- angle : radian
- distance - meter
- velocity - meter per second

Prepare Floorplan

For this we use Lougheed Floorplan

2.5 Pixels = 1 m



Here are the constrains,

Coordinate Center - (TOP LEFT, LR-> X+, TB -> Y+)

- Top Left (TL) - (0,0) in pixels, (49.252463,-122.897553) in Lat,Lon
- Bottom Left (BL) - (0,776) in pixels, (49.249681,-122.897553) in Lat,Lon
- Top Right (TR) - (887,0) in pixels, (49.252463,-122.892735) in Lat,Lon
- Bottom Right (BR) - (887,776) in pixels, (49.249681,-122.892735) in Lat,Lon

Now Let's load data we have. For this we use Fusion Location Provider's Data. (Which is in file FLP)

Load Data Set 1

In [2]:
```
dataset1 = pd.read_csv("dataset1.csv")
df1=pd.DataFrame(dataset1)
df1.head()
```

Out[2]:

	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Colu
0	422.225000	49.250524	-122.896379	21.393999	35.099998	4.599998	150.599854	0.0	0.21
1	423.342410	49.250518	-122.896356	20.046000	35.099998	4.599998	121.818413	0.0	0.62
2	424.343861	49.250517	-122.896350	19.625999	35.099998	4.599998	114.648056	0.0	0.52
3	425.345205	49.250519	-122.896352	18.134001	35.099998	4.599998	105.172920	0.0	0.26
4	426.340767	49.250519	-122.896351	17.996000	35.099998	4.599998	101.856239	0.0	0.19

Load Data Set 2

In [3]:
```
dataset2 = pd.read_csv("dataset2.csv")
df2=pd.DataFrame(dataset2)
df2.head()
```

Out[3]:

	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Colu
0	4214.342792	49.251212	-122.896010	8.834	39.699997	4.700001	97.656136	0	0.00
1	4215.344235	49.251207	-122.896008	8.408	39.699997	4.700001	95.627380	0	0.00
2	4216.343776	49.251202	-122.896005	7.797	39.699997	4.700001	96.607895	0	0.00
3	4216.973000	49.251198	-122.896002	7.536	35.099998	4.700001	95.984741	0	0.00
4	4218.344638	49.251190	-122.896000	7.042	35.099998	4.700001	130.190704	0	0.00

Concat Two Datasets

In [4]:
```
df_merged=pd.concat([df1,df2])
df_merged.reset_index(inplace=True, drop=True)
df_merged.head()
```

Out[4]:

	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Colu
0	422.225000	49.250524	-122.896379	21.393999	35.099998	4.599998	150.599854	0.0	0.21
1	423.342410	49.250518	-122.896356	20.046000	35.099998	4.599998	121.818413	0.0	0.62
2	424.343861	49.250517	-122.896350	19.625999	35.099998	4.599998	114.648056	0.0	0.52
3	425.345205	49.250519	-122.896352	18.134001	35.099998	4.599998	105.172920	0.0	0.26
4	426.340767	49.250519	-122.896351	17.996000	35.099998	4.599998	101.856239	0.0	0.19

Get Dataframe Informations

In [5]:
```
print(df1.shape[0])
print(df2.shape[0])
print(df_merged.shape[0])
```

```
986
747
1733
```

Select Only required data

In [6]:
```
sub_df=df_merged[['Column1','Column2','Column3']]
sub_df = sub_df.rename(columns={'Column1': 'TimeStamp', 'Column2': 'Latitude',
print(sub_df.shape[0])
sub_df.head()
```

1733

Out[6]:

	TimeStamp	Latitude	Longitude
0	422.225000	49.250524	-122.896379
1	423.342410	49.250518	-122.896356
2	424.343861	49.250517	-122.896350
3	425.345205	49.250519	-122.896352
4	426.340767	49.250519	-122.896351

However we can't deal with Latitude and Longitude, We have to convert it to pixels or meters.

In [7]:
```
# Calculate X direction,
# X direction -- Longitude
# X's Plus direction = Longitude's Plus Direction

# Calculate Y direction,
# Y direction -- Latitude
# Y's Plus direction = Latitude's Negative Direction

X_0_in_longitude=-122.897553
Y_0_in_latitude=49.252463
pixelspermeter=2.5
number_of_node_in_graph=30

# Distance between Two Lat,Lon


def distanceLatLonInMeters(lat1, lon1, lat2, lon2):
    p = pi/180
    a = 0.5 - cos((lat2-lat1)*p)/2 + cos(lat1*p) * cos(lat2*p) * (1-cos((lon2-
    return 12742000 * asin(sqrt(a))

x_dir_pixels=[]
y_dir_pixels=[]
x_dir_meters=[]
y_dir_meters=[]

for tuple in sub_df.itertuples():
    meters_in_x_direction=abs(distanceLatLonInMeters(Y_0_in_latitude,X_0_in_lo
    meters_in_y_direction=abs(distanceLatLonInMeters(Y_0_in_latitude,X_0_in_lo

    pixels_in_x_direction=round(meters_in_x_direction*2.5)
    pixels_in_y_direction=round(meters_in_y_direction*2.5)

    x_dir_pixels.append(pixels_in_x_direction)
    y_dir_pixels.append(pixels_in_y_direction)

    x_dir_meters.append(meters_in_x_direction)
    y_dir_meters.append(meters_in_y_direction)
```

```
    ##print(pixels_in_x_direction,pixels_in_y_direction,meters_in_x_direction,

updated_df=sub_df.copy()
updated_df["Pixels In X Direction"] = x_dir_pixels
updated_df["Pixels In Y Direction"] = y_dir_pixels
updated_df["Meters In X Direction"] = x_dir_meters
updated_df["Meters In Y Direction"] = y_dir_meters
```

Let's see updated dataframe

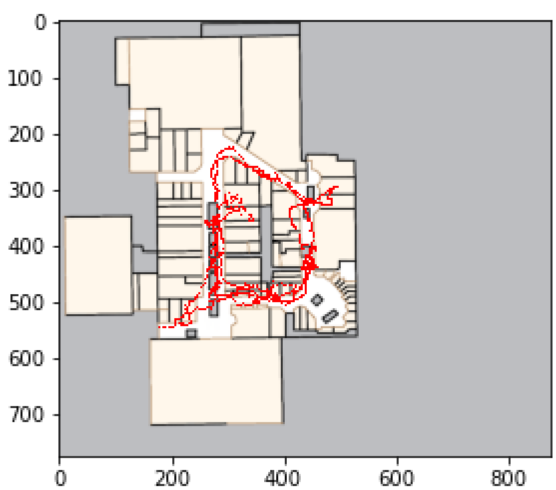
In [8]:
```
updated_df.head()
```

Out[8]:

	TimeStamp	Latitude	Longitude	Pixels In X Direction	Pixels In Y Direction	Meters In X Direction	Meters In Y Direction
0	422.225000	49.250524	-122.896379	213	539	85.208858	215.606966
1	423.342410	49.250518	-122.896356	217	541	86.878209	216.274132
2	424.343861	49.250517	-122.896350	218	541	87.313688	216.385327
3	425.345205	49.250519	-122.896352	218	540	87.168528	216.162941
4	426.340767	49.250519	-122.896351	218	540	87.241094	216.162941

Load Image and Mark Visited Areas

In [9]:
```
floorplan0 = image.imread('lougheed_00.png')
plt.imshow(floorplan0)
for row in updated_df.itertuples():
    plt.plot(row[4],row[5] , marker=',', color="red")
plt.savefig('visited_areas.png', bbox_inches='tight')
plt.show()
```



Now We can see that the area where measurements are taken

Now let's select some points to create graphs. In here we should **create several connected graph and give them unique graph ids**. However, the map we selected contains only one connected graph.

In [10]:
```
graph=[{'nodeid': 0, 'x_dir_pixels': 392, 'y_dir_pixels': 285, 'connected_grap
graphtable=pd.DataFrame(graph)
```

```
graphtable['x_dir_meters'] = graphtable.apply(lambda row: row['x_dir_pixels']/
graphtable['y_dir_meters'] = graphtable.apply(lambda row: row['y_dir_pixels']/
graphtable = graphtable[["nodeid","x_dir_pixels","y_dir_pixels","x_dir_meters"
graphtable
```

Out[10]:

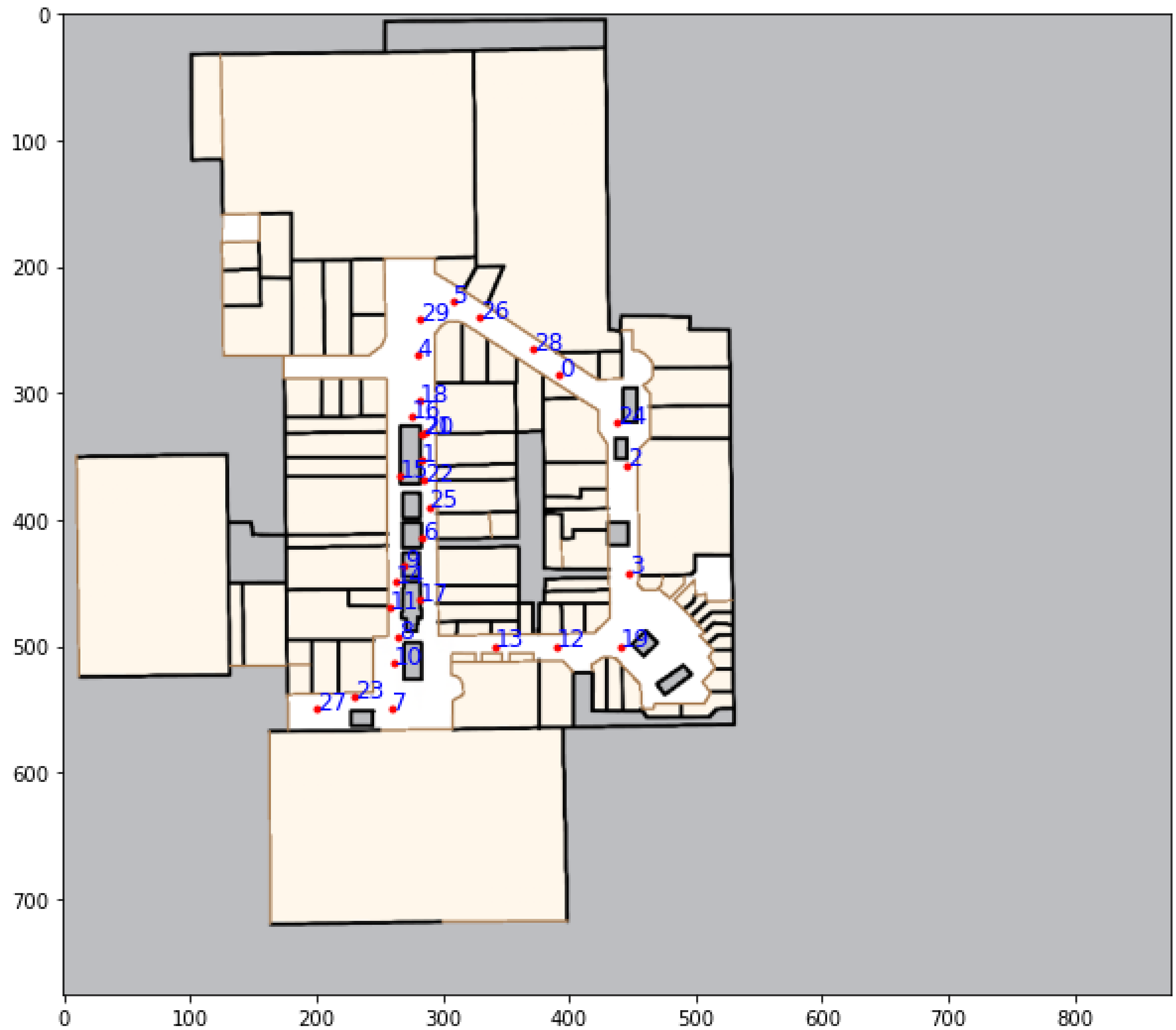
	nodeid	x_dir_pixels	y_dir_pixels	x_dir_meters	y_dir_meters	connected_graph_id
0	0	392	285	156.8	114.0	G1
1	1	283	353	113.2	141.2	G1
2	2	445	357	178.0	142.8	G1
3	3	447	442	178.8	176.8	G1
4	4	280	270	112.0	108.0	G1
5	5	308	228	123.2	91.2	G1
6	6	283	415	113.2	166.0	G1
7	7	260	550	104.0	220.0	G1
8	8	265	493	106.0	197.2	G1
9	9	269	436	107.6	174.4	G1
10	10	261	513	104.4	205.2	G1
11	11	258	470	103.2	188.0	G1
12	12	390	500	156.0	200.0	G1
13	13	341	500	136.4	200.0	G1
14	14	263	449	105.2	179.6	G1
15	15	266	366	106.4	146.4	G1
16	16	275	319	110.0	127.6	G1
17	17	282	463	112.8	185.2	G1
18	18	282	306	112.8	122.4	G1
19	19	440	500	176.0	200.0	G1
20	20	286	331	114.4	132.4	G1
21	21	283	332	113.2	132.8	G1
22	22	285	369	114.0	147.6	G1
23	23	230	540	92.0	216.0	G1
24	24	438	323	175.2	129.2	G1
25	25	289	390	115.6	156.0	G1
26	26	329	240	131.6	96.0	G1
27	27	200	550	80.0	220.0	G1
28	28	371	265	148.4	106.0	G1
29	29	282	242	112.8	96.8	G1

Let's draw graph,

In [11]:

```
floorplan1 = image.imread('lougheed_00.png')
```

```
plt.figure(figsize = (10,10))
plt.imshow(floorplan1)
for row in graphtable.to_dict('records'):
    plt.plot(row["x_dir_pixels"],row["y_dir_pixels"] , marker='.', color="red"
    plt.text(row["x_dir_pixels"],row["y_dir_pixels"] , str(row["nodeid"]), col
plt.show()
```



Define Reachable

In [12]:
```
# Node ID , Reachable IDs
reachable={}
for from_row in graphtable.to_dict('records'):
    from_node =  from_row["nodeid"]
    from_x = from_row["x_dir_meters"]
    from_y = from_row["y_dir_meters"]
    distances = {}
    for to_row in graphtable.to_dict('records') :
        to_node = to_row["nodeid"]
        if from_node == to_node :
            continue
        to_x = to_row["x_dir_meters"]
        to_y = to_row["y_dir_meters"]
        distance= sqrt((to_x-from_x)**2 + (to_y-from_y)**2)
        distances[to_node]=distance
    nearest = sorted(distances, key=distances.get)[:2]
```

```
    reachable[from_node]=nearest
print(reachable)
```

```
{0: [28, 24], 1: [22, 21], 2: [24, 3], 3: [19, 12], 4: [29, 18], 5: [26, 29],
6: [9, 25], 7: [23, 10], 8: [10, 11], 9: [14, 6], 10: [8, 7], 11: [14, 8], 12:
[13, 19], 13: [12, 17], 14: [9, 11], 15: [22, 1], 16: [18, 21], 17: [14, 11],
18: [16, 20], 19: [12, 3], 20: [21, 16], 21: [20, 16], 22: [1, 15], 23: [7, 2
7], 24: [2, 0], 25: [22, 6], 26: [5, 29], 27: [23, 7], 28: [0, 26], 29: [4,
5]}
```

Add reachable to columns

In [13]:

```
reachable_1 =[]
reachable_2 =[]
#reachable_3 =[]

for value in reachable.values():
    reachable_1.append(value[0])
    reachable_2.append(value[1])
   # reachable_3.append(value[2])

graphtable["reachable_1"]=reachable_1
graphtable["reachable_2"]=reachable_2
#graphtable["reachable_3"]=reachable_3
graphtable = graphtable[["nodeid","x_dir_pixels","y_dir_pixels","x_dir_meters"
graphtable
```

Out[13]:

	nodeid	x_dir_pixels	y_dir_pixels	x_dir_meters	y_dir_meters	reachable_1	reachable_2	conne
0	0	392	285	156.8	114.0	28	24	
1	1	283	353	113.2	141.2	22	21	
2	2	445	357	178.0	142.8	24	3	
3	3	447	442	178.8	176.8	19	12	
4	4	280	270	112.0	108.0	29	18	
5	5	308	228	123.2	91.2	26	29	
6	6	283	415	113.2	166.0	9	25	
7	7	260	550	104.0	220.0	23	10	
8	8	265	493	106.0	197.2	10	11	
9	9	269	436	107.6	174.4	14	6	
10	10	261	513	104.4	205.2	8	7	
11	11	258	470	103.2	188.0	14	8	
12	12	390	500	156.0	200.0	13	19	
13	13	341	500	136.4	200.0	12	17	
14	14	263	449	105.2	179.6	9	11	
15	15	266	366	106.4	146.4	22	1	
16	16	275	319	110.0	127.6	18	21	
17	17	282	463	112.8	185.2	14	11	
18	18	282	306	112.8	122.4	16	20	
19	19	440	500	176.0	200.0	12	3	
20	20	286	331	114.4	132.4	21	16	
21	21	283	332	113.2	132.8	20	16	
22	22	285	369	114.0	147.6	1	15	
23	23	230	540	92.0	216.0	7	27	
24	24	438	323	175.2	129.2	2	0	
25	25	289	390	115.6	156.0	22	6	
26	26	329	240	131.6	96.0	5	29	
27	27	200	550	80.0	220.0	23	7	
28	28	371	265	148.4	106.0	0	26	
29	29	282	242	112.8	96.8	4	5	

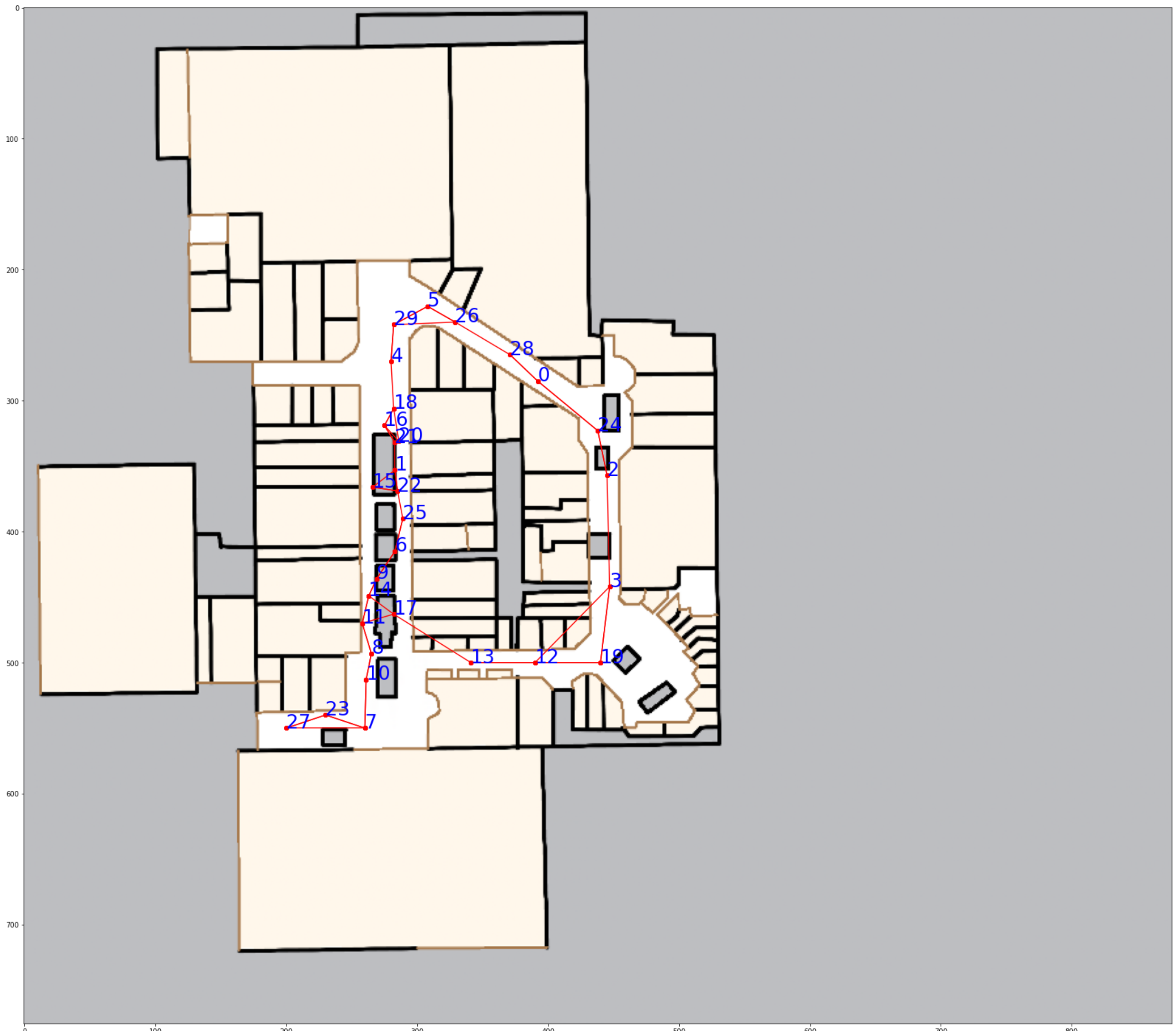
Draw Graph

In [14]:

```
floorplan1 = image.imread('lougheed_00.png')
plt.figure(figsize = (30,30))
plt.imshow(floorplan1)
for row in graphtable.to_dict('records'):
```

```
    reachable_ids=[row["reachable_1"], row["reachable_2"]]
    for reachablerow in graphtable.to_dict('records'):
        if reachablerow["nodeid"] in reachable_ids:
            plt.plot([row["x_dir_pixels"], reachablerow["x_dir_pixels"]],[row[

    plt.plot(row["x_dir_pixels"],row["y_dir_pixels"] , marker='.', color="red"
    plt.text(row["x_dir_pixels"],row["y_dir_pixels"] , str(row["nodeid"]), col
plt.savefig('floor_plan_graph.png', bbox_inches='tight')
plt.show()
```



Now we have successfully created a graph for the floor plan

Prepare DataSet

Let's revisit updated_df

In [15]:
```
updated_df
```

Out[15]:

	TimeStamp	Latitude	Longitude	Pixels In X Direction	Pixels In Y Direction	Meters In X Direction	Meters In Y Direction
0	422.225000	49.250524	-122.896379	213	539	85.208858	215.606966
1	423.342410	49.250518	-122.896356	217	541	86.878209	216.274132
2	424.343861	49.250517	-122.896350	218	541	87.313688	216.385327
3	425.345205	49.250519	-122.896352	218	540	87.168528	216.162941
4	426.340767	49.250519	-122.896351	218	540	87.241094	216.162941
...	...	...	...	...	...	...	...
1728	5139.091000	49.250493	-122.896482	194	548	77.733134	219.053999
1729	5139.592000	49.250492	-122.896509	189	548	75.773474	219.165203
1730	5140.198000	49.250492	-122.896525	187	548	74.612179	219.165203
1731	5140.773000	49.250491	-122.896546	183	548	73.088002	219.276392
1732	5141.852000	49.250491	-122.896573	178	548	71.128362	219.276392

1733 rows × 7 columns

Denfine Helper Functions

In [16]:

```
def calcVelocityVal(prev_row,row):
    distance=sqrt((row[6]-prev_row[6])**2+(row[7]-prev_row[7])**2)
    timediff=row[1]-prev_row[1]
    return distance/timediff

def calcVelocityAngle(prev_row,row):
    return atan2(row[7]-prev_row[7], row[6]-prev_row[6])

def calcNearestState(row):

    currentNearestStateID=None
    minDistance=float('inf')

    for points in graph:

        stateid=points["nodeid"]
        distanceToState=sqrt((points['y_dir_pixels']-row[5])**2+(points['x_dir

        if distanceToState<=minDistance:
            minDistance=distanceToState
            currentNearestStateID=stateid
    return currentNearestStateID
```

Now let's calculate velocities and nearest states

In [17]:

```
velocity_value=["N/A"]
velocity_angle=["N/A"]
nearest_state=["N/A"]
isValid=[]
prev_row=None

for row in updated_df.itertuples():
```

```
    # Process First Value
    if row[0]==0:
        prev_row=row
        isValid.append(0)
        continue

    timediff=row[1]-prev_row[1]
    if timediff>=2:
        isValid.append(0)
    else:
        isValid.append(1)


    velocity_value.append(calcVelocityVal(prev_row,row))
    velocity_angle.append(calcVelocityAngle(prev_row,row))
    nearest_state.append(calcNearestState(row))
    prev_row=row

updated_df["Velocity_Value"]=velocity_value
updated_df["Velocity_Angle"]=velocity_angle
updated_df["Nearest_State"]=nearest_state
updated_df["isValid"]=isValid
```

Let's check new dataframe

In [18]:
```
updated_df
```

Out[18]:

	TimeStamp	Latitude	Longitude	Pixels In X Direction	Pixels In Y Direction	Meters In X Direction	Meters In Y Direction	Velocity_Va
0	422.225000	49.250524	-122.896379	213	539	85.208858	215.606966	I
1	423.342410	49.250518	-122.896356	217	541	86.878209	216.274132	1.6088
2	424.343861	49.250517	-122.896350	218	541	87.313688	216.385327	0.4488
3	425.345205	49.250519	-122.896352	218	540	87.168528	216.162941	0.2652
4	426.340767	49.250519	-122.896351	218	540	87.241094	216.162941	0.072
...	...	...	...	...	...	...	...	
1728	5139.091000	49.250493	-122.896482	194	548	77.733134	219.053999	1.1501
1729	5139.592000	49.250492	-122.896509	189	548	75.773474	219.165203	3.9177
1730	5140.198000	49.250492	-122.896525	187	548	74.612179	219.165203	1.9163
1731	5140.773000	49.250491	-122.896546	183	548	73.088002	219.276392	2.6577
1732	5141.852000	49.250491	-122.896573	178	548	71.128362	219.276392	1.8161

1733 rows × 11 columns

Now lets created filtered dataframe which only contains, required columns and valid values

In [19]:
```
filtered_df=updated_df.copy()
filtered_df=filtered_df[filtered_df['isValid'] == 1]
filtered_df=filtered_df[["TimeStamp","Meters In X Direction","Meters In Y Dire
```

```
filtered_df.reset_index(inplace=True, drop=True)
filtered_df.head()
```

Out[19]:

	TimeStamp	Meters In X Direction	Meters In Y Direction	Velocity_Value	Velocity_Angle	Nearest_State
0	423.342410	86.878209	216.274132	1.608839	0.38021	23
1	424.343861	87.313688	216.385327	0.448801	0.249999	23
2	425.345205	87.168528	216.162941	0.265213	-2.149096	23
3	426.340767	87.241094	216.162941	0.07289	0.0	23
4	427.343544	87.241094	216.162941	0.0	0.0	23

Let's check first index of second data set

In [20]:

```
print(filtered_df.loc[873:877])
```

```
       TimeStamp  Meters In X Direction  Meters In Y Direction Velocity_Value  \
873  1724.342825             104.732868             145.887755       0.494914
874  1725.344916             105.531256             146.888500       1.277529
875  4215.344235             112.136031             139.660827       0.573771
876  4216.343776             112.353760             140.216800       0.597361
877  4216.973000             112.571511             140.661598       0.787062

    Velocity_Angle Nearest_State
873       0.743327            15
874       0.897402            15
875       1.315392             1
876       1.197537             1
877       1.115543             1
```

In [21]:

```
first_index_of_second_set=875
```

Now Let's create a dataset

In [22]:

```
data=[]
for row in filtered_df.loc[19:first_index_of_second_set-1].itertuples():
    datadict={}
    start_x = filtered_df.loc[row[0]-19][1]
    start_y = filtered_df.loc[row[0]-19][2]
    datadict["startX_in_meters"]=start_x
    datadict["startY_in_meters"]=start_y
    i=0
    for subrow in filtered_df.loc[row[0]-19:row[0]].itertuples():
        datadict["velocity_value_"+str(i+1)]=subrow[4]
        i+=1
    j=0
    for subrow in filtered_df.loc[row[0]-19:row[0]].itertuples():
        datadict["velocity_angle_"+str(j+1)]=subrow[5]
        j+=1
    k=0
    for subrow in filtered_df.loc[row[0]-19:row[0]].itertuples():
        datadict["state_"+str(k+1)]=subrow[6]
        k+=1
    data.append(datadict)
```

```
prepared_dataset1=pd.DataFrame(data)
prepared_dataset1
```

Out[22]:

	startX_in_meters	startY_in_meters	velocity_value_1	velocity_value_2	velocity_value_3	veloci
0	86.878209	216.274132	1.608839	0.448801	0.265213	
1	87.313688	216.385327	0.448801	0.265213	0.072890	
2	87.168528	216.162941	0.265213	0.072890	0.000000	
3	87.241094	216.162941	0.072890	0.000000	0.072421	
4	87.241094	216.162941	0.000000	0.072421	2.113376	
...	...	...	...	...	...	
851	108.869936	146.999699	0.716452	1.070863	0.316556	
852	108.507033	146.443721	1.070863	0.316556	0.332816	
853	108.361861	146.221317	0.316556	0.332816	0.379091	
854	107.998976	146.221317	0.332816	0.379091	0.717360	
855	107.636062	146.110143	0.379091	0.717360	0.866805	

856 rows × 62 columns

In [23]:

```
data=[]
for row in filtered_df.loc[first_index_of_second_set:].itertuples():
    datadict={}
    start_x = filtered_df.loc[row[0]-19][1]
    start_y = filtered_df.loc[row[0]-19][2]
    datadict["startX_in_meters"]=start_x
    datadict["startY_in_meters"]=start_y
    i=0
    for subrow in filtered_df.loc[row[0]-19:row[0]].itertuples():
        datadict["velocity_value_"+str(i+1)]=subrow[4]
        i+=1
    j=0
    for subrow in filtered_df.loc[row[0]-19:row[0]].itertuples():
        datadict["velocity_angle_"+str(j+1)]=subrow[5]
        j+=1
    k=0
    for subrow in filtered_df.loc[row[0]-19:row[0]].itertuples():
        datadict["state_"+str(k+1)]=subrow[6]
        k+=1
    data.append(datadict)

prepared_dataset2=pd.DataFrame(data)
prepared_dataset2
```

Out[23]:

	startX_in_meters	startY_in_meters	velocity_value_1	velocity_value_2	velocity_value_3	veloci
0	107.055423	146.110143	0.717360	0.866805	0.687368	
1	106.547363	146.554919	0.866805	0.687368	0.868035	
2	106.039313	147.222090	0.687368	0.868035	1.374921	
3	105.894149	146.777309	0.868035	1.374921	1.548056	
4	106.039313	145.887755	1.374921	1.548056	1.219290	
...	...	...	...	...	...	
673	82.450823	214.050235	0.263278	0.427974	0.170369	
674	82.305671	214.272627	0.427974	0.170369	0.143115	
675	82.233092	214.495019	0.170369	0.143115	0.106922	
676	82.160495	214.606212	0.143115	0.106922	0.286135	
677	82.160495	214.717411	0.106922	0.286135	0.841608	

678 rows × 62 columns

In [24]:

```
final_prepared_dataset=pd.concat([prepared_dataset1,prepared_dataset2])
final_prepared_dataset.reset_index(inplace=True, drop=True)
final_prepared_dataset
```

Out[24]:

	startX_in_meters	startY_in_meters	velocity_value_1	velocity_value_2	velocity_value_3	veloc
0	86.878209	216.274132	1.608839	0.448801	0.265213	
1	87.313688	216.385327	0.448801	0.265213	0.072890	
2	87.168528	216.162941	0.265213	0.072890	0.000000	
3	87.241094	216.162941	0.072890	0.000000	0.072421	
4	87.241094	216.162941	0.000000	0.072421	2.113376	
...	...	...	...	...	...	
1529	82.450823	214.050235	0.263278	0.427974	0.170369	
1530	82.305671	214.272627	0.427974	0.170369	0.143115	
1531	82.233092	214.495019	0.170369	0.143115	0.106922	
1532	82.160495	214.606212	0.143115	0.106922	0.286135	
1533	82.160495	214.717411	0.106922	0.286135	0.841608	

1534 rows × 62 columns

Save Outcomes

Now let's save out results

In [25]:

```
graphtable.to_csv("floor_plan_graph.csv", encoding='utf-8', index=False)
final_prepared_dataset.to_csv("final_prepared_dataset.csv", encoding='utf-8',
```

In [ ]:

```
```