SPORT ANALYSIS WITH PYTHON

PYDATA @ PYCON DE
OCTOBER 25-27, 2017

Speaker:

- ✓ **ThuyLe**
- ✓ **Ericsson, Italy**

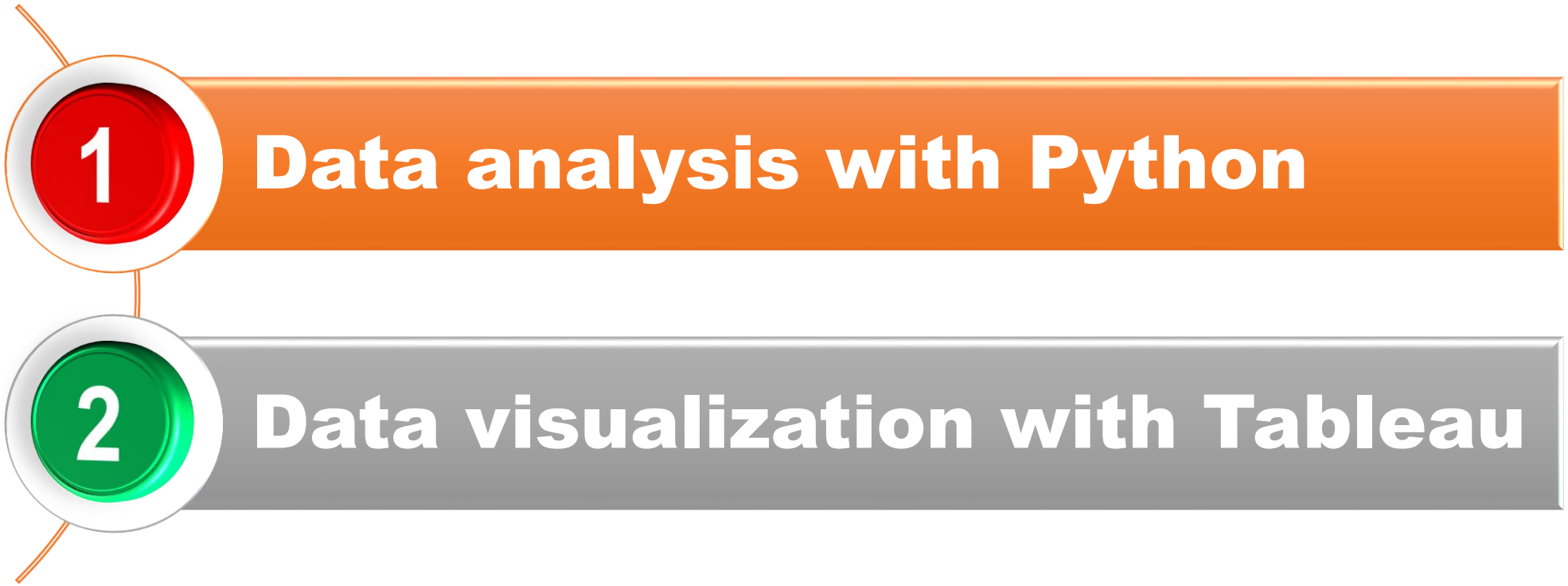
Football analysis

1. Match (score, time, …)
2. Players (performance, goal, award, match, time…)
3. Fan (number, loyalty, scandal…)

   ...

Match analysis (*real-time match tracking*)

OUTLINE


1
Data analysis with Python
2
Data visualization with Tableau

DATA

1. Timestamp
2. Devices_id
3. Positions (x_pos, y_pos)

Calculate

- Real Time
- Distance
- Total distance

1. Velocity
2. Turning points
3. Zone

Load Data

```
#=========================================
# # IMPORT LIBRARIES
#=========================================
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random as rd
import os, time, math, datetime
from math import sqrt



#=========================================
# # Load Data
#=========================================
def loadCSV(csvName, _sep):
    data = pd.read_csv(csvName, sep=_sep)
    return data


print os.getcwd()
```

```
/home/mapr/notebookHome/THUYLE_TEST
```

```
path = "/home/mapr/notebookHome/THUYLE_TEST"

os.chdir(path)
sep = "|"
df = loadCSV("ThuyLe_P1_Demo_TotalDistance.csv", sep)
```

Load Data

```
df.head(15)
```

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	distance	totalDistance
0	1.237747e+12	9	50.875526	25.792036	1	1	0.0000	0.0000
1	1.237747e+12	8	28.525500	41.909308	1	1	0.0000	0.0000
2	1.237747e+12	2	21.108100	12.481253	1	1	0.0000	0.0000
3	1.237747e+12	6	45.247400	26.818432	1	1	0.0000	0.0000
4	1.237747e+12	7	32.231844	12.636959	1	1	0.0000	0.0000
5	1.237747e+12	5	28.107198	40.277207	1	1	0.0000	0.0000
6	1.237747e+12	11	41.571300	61.727806	1	1	0.0000	0.0000
7	1.237747e+12	1	4.295718	45.317342	1	1	0.0000	0.0000
8	1.237747e+12	4	35.724664	29.717300	1	1	0.0000	0.0000
9	1.237747e+12	3	26.620200	56.002000	1	1	0.0000	0.0000
10	1.237747e+12	10	51.444286	49.769524	1	1	0.0000	0.0000
11	1.237747e+12	9	50.907026	25.734036	1	1	0.0660	0.0660
12	1.237747e+12	8	28.521800	41.885908	1	1	0.0237	0.0237
13	1.237747e+12	2	21.154600	12.468753	1	1	0.0482	0.0482
14	1.237747e+12	6	45.247400	26.818432	1	1	0.0000	0.0000

Data analysis

```
#=========================
# # Shape of Data
#=========================

df.shape
```

```
(594011, 8)
```

```
#=========================================================================
# # Columns of Data
#=========================================================================

df.columns
```

```
Index([u'timestamp', u'device_id', u'x_pos', u'y_pos', u'matchTime_minute',
       u'matchTime_second', u'distance', u'totalDistance'],
      dtype='object')
```

```
#=====================================
# # Unique value of column "device_id"
#=====================================

devices = df['device_id'].unique()
print devices
```

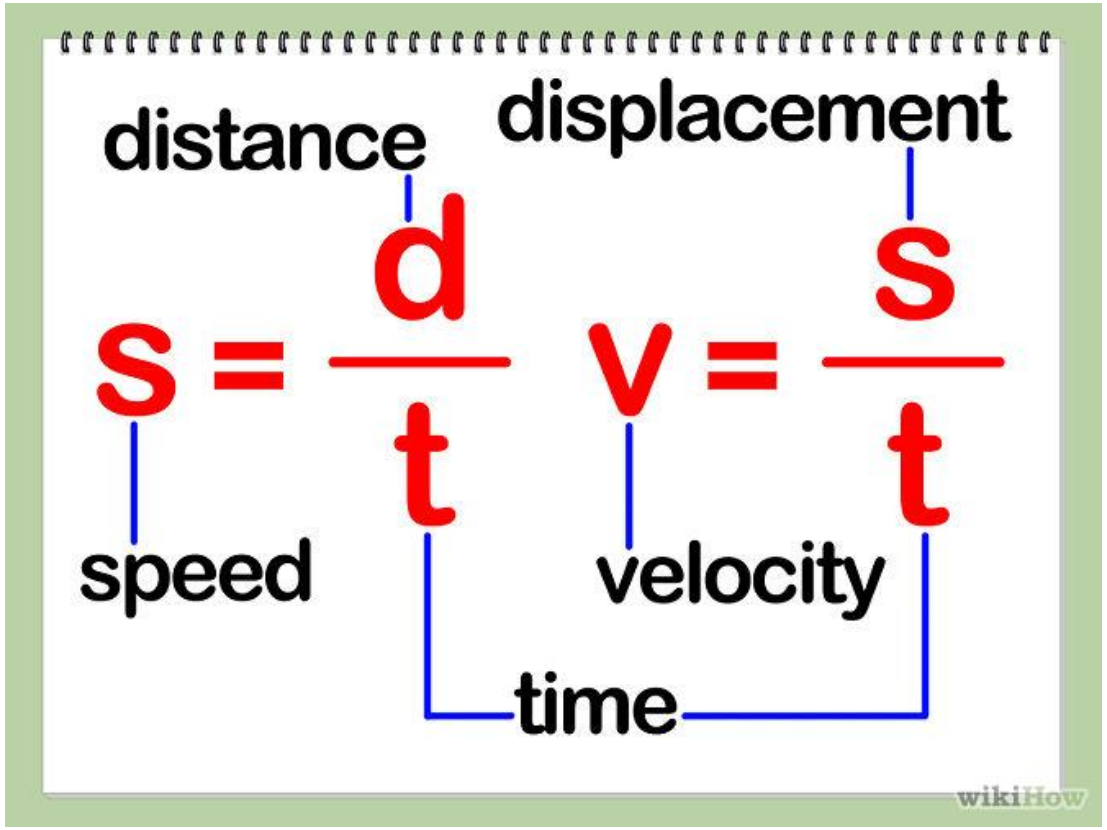
```
[ 9  8  2  6  7  5 11  1  4  3 10]
```

Describe Data

```
#==================
# # Describe Data
#==================

df.describe()
```

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	totalDistance
count	5.940110e+05	594011.00000	594011.000000	594011.000000	594011.000000	594011.000000	594011.000000
mean	1.237748e+12	6.00000	40.995397	43.856025	23.000426	30.499454	3062.034685
std	7.794380e+05	3.16228	23.624175	19.094993	12.987441	17.318422	1757.739790
min	1.237747e+12	1.00000	-3.851690	-3.709834	1.000000	1.000000	0.000000
25%	1.237747e+12	3.00000	22.013650	29.903948	12.000000	15.000000	1598.513550
50%	1.237748e+12	6.00000	42.212144	43.642300	23.000000	30.000000	3025.199000
75%	1.237749e+12	9.00000	57.166875	57.709330	34.000000	45.000000	4474.828350
max	1.237749e+12	11.00000	104.214000	91.831700	46.000000	60.000000	7938.709100

Calculate Velocity



Calculate Velocity

```
#=============================================================================
# # Calculate the difference time
#=============================================================================

def difTime(_df, _devices):
    _df['difTime'] = 0
    for d in _devices:
        idd = _df[_df['device_id'] == d].index
        lend = len(idd)

        for i in range (1, lend):
            _df.loc[idd[i], 'difTime'] = float((_df.loc[idd[i], 'timestamp'] \
                                                - _df.loc[idd[i-1], 'timestamp'])/1000)
    return _df
```

```
df = difTime(df, devices)
df
```

Calculate Velocity

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	distance	totalDistance	difTime
0	1.237747e+12	9	50.875526	25.792036	1	1	0	0.0000	0.00
1	1.237747e+12	8	28.525500	41.909308	1	1	0	0.0000	0.00
2	1.237747e+12	2	21.108100	12.481253	1	1	0	0.0000	0.00
3	1.237747e+12	6	45.247400	26.818432	1	1	0	0.0000	0.00
4	1.237747e+12	7	32.231844	12.636959	1	1	0	0.0000	0.00
5	1.237747e+12	5	28.107198	40.277207	1	1	0	0.0000	0.00
6	1.237747e+12	11	41.571300	61.727806	1	1	0	0.0000	0.00
7	1.237747e+12	1	4.295718	45.317342	1	1	0	0.0000	0.00
8	1.237747e+12	4	35.724664	29.717300	1	1	0	0.0000	0.00
9	1.237747e+12	3	26.620200	56.002000	1	1	0	0.0000	0.00
10	1.237747e+12	10	51.444286	49.769524	1	1	0	0.0000	0.00
11	1.237747e+12	9	50.907026	25.734036	1	1	0.0660	0.0660	0.05
12	1.237747e+12	8	28.521800	41.885908	1	1	0.0237	0.0237	0.05
13	1.237747e+12	2	21.154600	12.468753	1	1	0.0482	0.0482	0.05

Calculate Velocity

```
df['difTime'].value_counts(dropna=False)
```

```
0.05     594000
0.00         11
Name: difTime, dtype: int64
```

Calculate Velocity

```
#=============================================================
# # Calculate Velocity
#=============================================================

def velocity(_df, _devices):
    _df['velocity'] = 0
    for d in _devices:
        idd = _df[_df['device_id'] == d].index
        print type(idd)
        iVe = idd[1:]

        _df['velocity'][iVe] = _df["distance"][iVe].astype("float")/\
                               _df['difTime'][iVe].astype("float")
    return _df
```

```
df = velocity(df, devices)
df
```

Calculate Velocity

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	distance	totalDistance	difTime	velocity
0	1.237747e+12	9	50.875526	25.792036	1	1	0	0.0000	0.00	0.000
1	1.237747e+12	8	28.525500	41.909308	1	1	0	0.0000	0.00	0.000
2	1.237747e+12	2	21.108100	12.481253	1	1	0	0.0000	0.00	0.000
3	1.237747e+12	6	45.247400	26.818432	1	1	0	0.0000	0.00	0.000
4	1.237747e+12	7	32.231844	12.636959	1	1	0	0.0000	0.00	0.000
5	1.237747e+12	5	28.107198	40.277207	1	1	0	0.0000	0.00	0.000
6	1.237747e+12	11	41.571300	61.727806	1	1	0	0.0000	0.00	0.000
7	1.237747e+12	1	4.295718	45.317342	1	1	0	0.0000	0.00	0.000
8	1.237747e+12	4	35.724664	29.717300	1	1	0	0.0000	0.00	0.000
9	1.237747e+12	3	26.620200	56.002000	1	1	0	0.0000	0.00	0.000
10	1.237747e+12	10	51.444286	49.769524	1	1	0	0.0000	0.00	0.000
11	1.237747e+12	9	50.907026	25.734036	1	1	0.0660	0.0660	0.05	1.320
12	1.237747e+12	8	28.521800	41.885908	1	1	0.0237	0.0237	0.05	0.474
13	1.237747e+12	2	21.154600	12.468753	1	1	0.0482	0.0482	0.05	0.964

Calculate Velocity

In this case the difference time are always 50 ms

```
#=============================================
# # Calculate speed (if the diference time is 50 ms)
#=============================================
df1= df.copy()
df1['speed'] = df1["distance"].astype("float")/(0.05)
df1
```

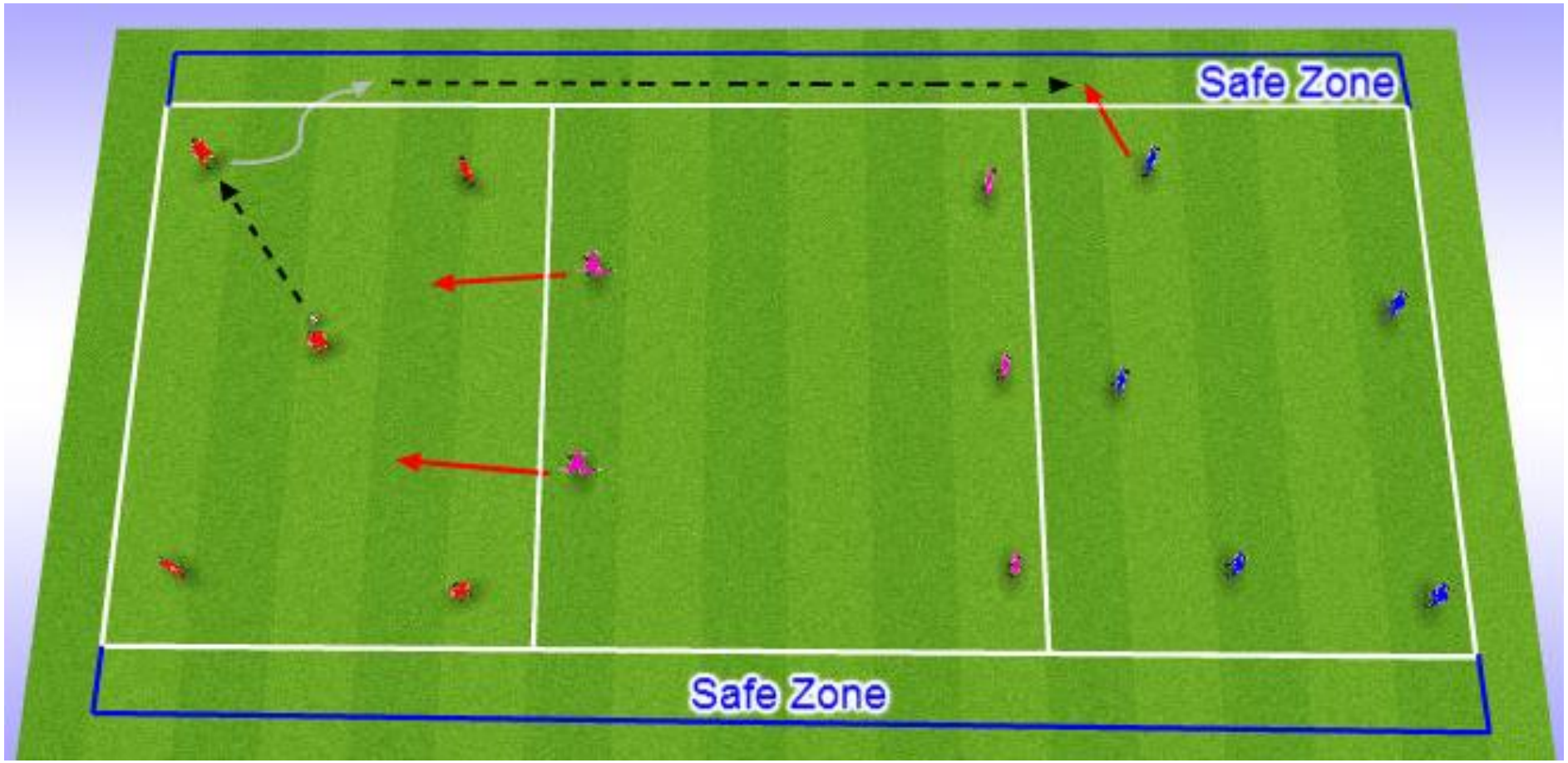
Calculate Velocity

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	distance	totalDistance	difTime	velocity	speed
0	1.237747e+12	9	50.875526	25.792036	1	1	0	0.0000	0.00	0.000	0.000
1	1.237747e+12	8	28.525500	41.909308	1	1	0	0.0000	0.00	0.000	0.000
2	1.237747e+12	2	21.108100	12.481253	1	1	0	0.0000	0.00	0.000	0.000
3	1.237747e+12	6	45.247400	26.818432	1	1	0	0.0000	0.00	0.000	0.000
4	1.237747e+12	7	32.231844	12.636959	1	1	0	0.0000	0.00	0.000	0.000
5	1.237747e+12	5	28.107198	40.277207	1	1	0	0.0000	0.00	0.000	0.000
6	1.237747e+12	11	41.571300	61.727806	1	1	0	0.0000	0.00	0.000	0.000
7	1.237747e+12	1	4.295718	45.317342	1	1	0	0.0000	0.00	0.000	0.000
8	1.237747e+12	4	35.724664	29.717300	1	1	0	0.0000	0.00	0.000	0.000
9	1.237747e+12	3	26.620200	56.002000	1	1	0	0.0000	0.00	0.000	0.000
10	1.237747e+12	10	51.444286	49.769524	1	1	0	0.0000	0.00	0.000	0.000
11	1.237747e+12	9	50.907026	25.734036	1	1	0.0660	0.0660	0.05	1.320	1.320
12	1.237747e+12	8	28.521800	41.885908	1	1	0.0237	0.0237	0.05	0.474	0.474
13	1.237747e+12	2	21.154600	12.468753	1	1	0.0482	0.0482	0.05	0.964	0.964
14	1.237747e+12	6	45.247400	26.818432	1	1	0.0000	0.0000	0.05	0.000	0.000
15	1.237747e+12	7	32.231844	12.636959	1	1	0.0000	0.0000	0.05	0.000	0.000

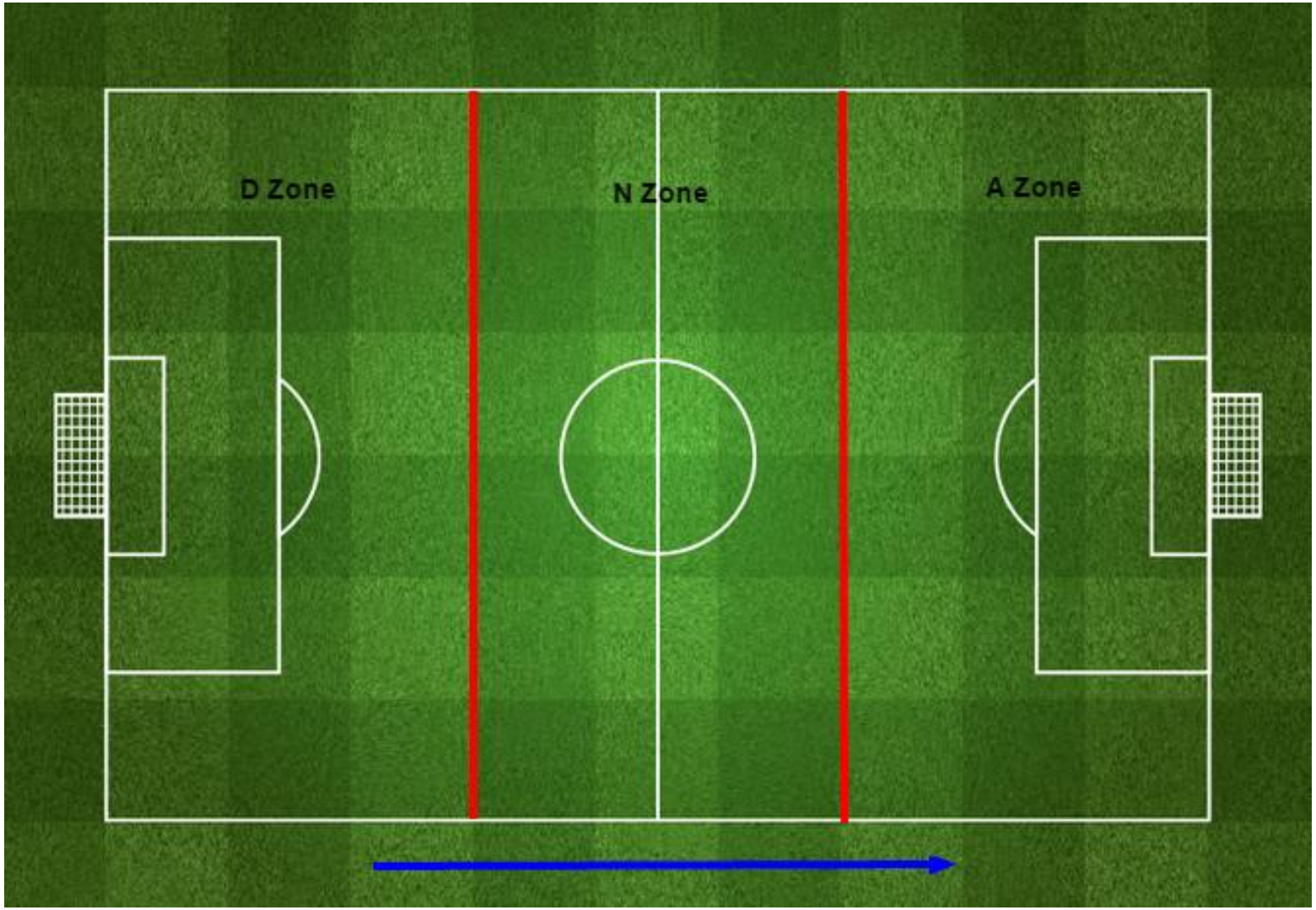
Calculate Velocity

```
#==================
# # Describe Data
#==================

df.describe()
```

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	totalDistance	difTime	velocity
count	5.940110e+05	594011.00000	594011.000000	594011.000000	594011.000000	594011.000000	594011.000000	594011.000000	594011.000000
mean	1.237748e+12	6.00000	40.995397	43.856025	23.000426	30.499454	3062.034685	0.049999	2.250519
std	7.794380e+05	3.16228	23.624175	19.094993	12.987441	17.318422	1757.739790	0.000215	1.419839
min	1.237747e+12	1.00000	-3.851690	-3.709834	1.000000	1.000000	0.000000	0.000000	0.000000
25%	1.237747e+12	3.00000	22.013650	29.903948	12.000000	15.000000	1598.513550	0.050000	1.170000
50%	1.237748e+12	6.00000	42.212144	43.642300	23.000000	30.000000	3025.199000	0.050000	1.998000
75%	1.237749e+12	9.00000	57.166875	57.709330	34.000000	45.000000	4474.828350	0.050000	3.112000
max	1.237749e+12	11.00000	104.214000	91.831700	46.000000	60.000000	7938.709100	0.050000	9.778000


Safe Zone
Safe Zone

Calculate Zone



3 zones

Calculate Zone

```
#=======================================================
# # Calculate Zone (3 zones)
#=======================================================

def calculate3Zones(_df, _x):
    _df["3Zones"] = ""
    lendf = len(_df)
    xZone1 = _x/3
    xZone3 = 2*xZone1
    for i in range (0, lendf):
        if (_df.loc[i, "x_pos"] < xZone1):
            _df.loc[i, "3Zones"] = "Zone_1"
        else:
            if (_df.loc[i, "x_pos"] > xZone3):
                _df.loc[i, "3Zones"] = "Zone_3"
            else:
                _df.loc[i, "3Zones"] = "Zone_2"
    return _df
```

3 zones

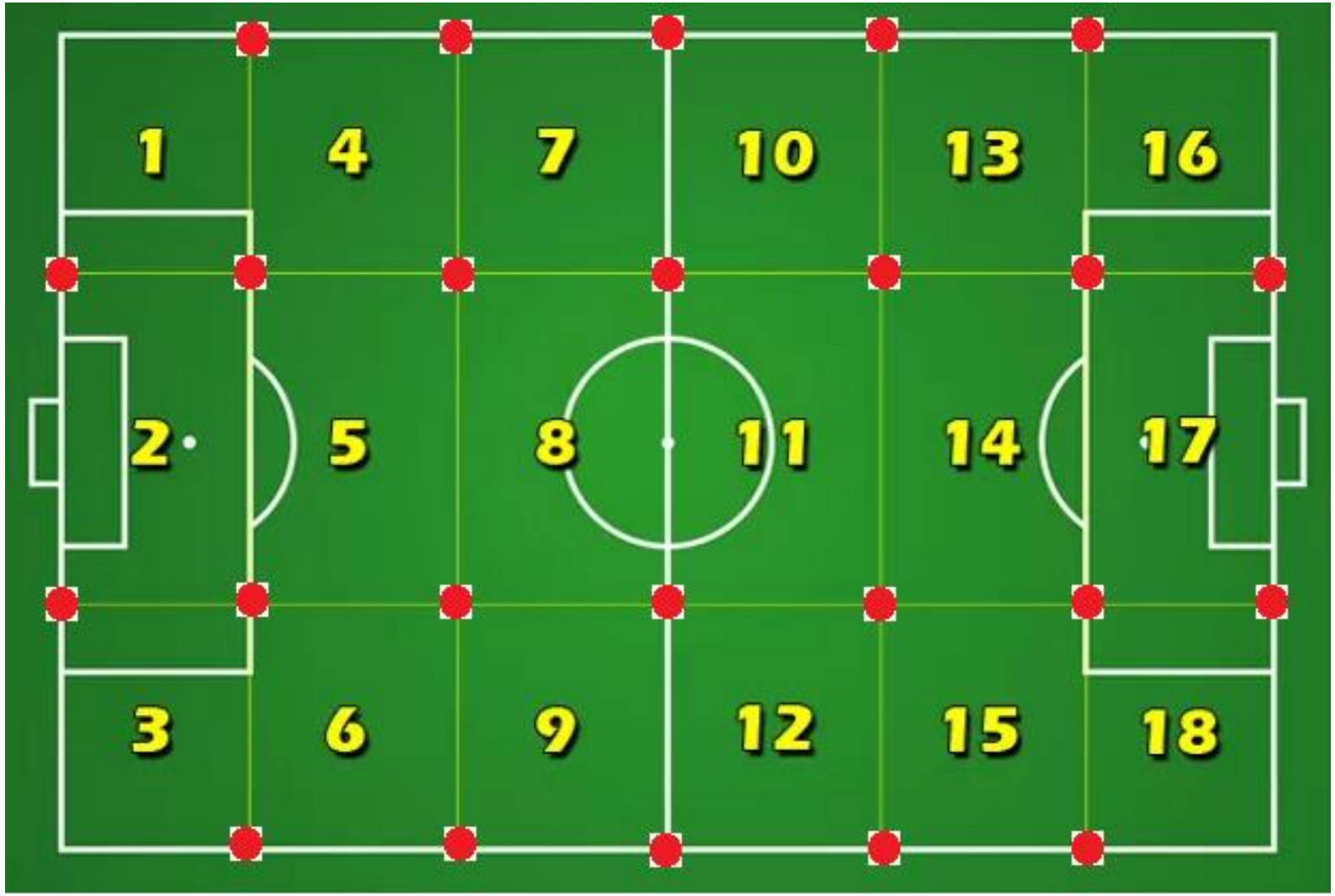
```
x = 100  # The long of the pitch is 100 m
y = 90   # The wide of the pitch is 90m
df = calculate3Zones(df, x)
df
```

Calculate Zone

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	distance	totalDistance	difTime	velocity	3Zones
0	1.237747e+12	9	50.875526	25.792036	1	1	0.0000	0.0000	0.00	0.000	Zone_2
1	1.237747e+12	8	28.525500	41.909308	1	1	0.0000	0.0000	0.00	0.000	Zone_1
2	1.237747e+12	2	21.108100	12.481253	1	1	0.0000	0.0000	0.00	0.000	Zone_1
3	1.237747e+12	6	45.247400	26.818432	1	1	0.0000	0.0000	0.00	0.000	Zone_2
4	1.237747e+12	7	32.231844	12.636959	1	1	0.0000	0.0000	0.00	0.000	Zone_1
5	1.237747e+12	5	28.107198	40.277207	1	1	0.0000	0.0000	0.00	0.000	Zone_1
6	1.237747e+12	11	41.571300	61.727806	1	1	0.0000	0.0000	0.00	0.000	Zone_2
7	1.237747e+12	1	4.295718	45.317342	1	1	0.0000	0.0000	0.00	0.000	Zone_1
8	1.237747e+12	4	35.724664	29.717300	1	1	0.0000	0.0000	0.00	0.000	Zone_2
9	1.237747e+12	3	26.620200	56.002000	1	1	0.0000	0.0000	0.00	0.000	Zone_1
10	1.237747e+12	10	51.444286	49.769524	1	1	0.0000	0.0000	0.00	0.000	Zone_2
11	1.237747e+12	9	50.907026	25.734036	1	1	0.0660	0.0660	0.05	1.320	Zone_2
12	1.237747e+12	8	28.521800	41.885908	1	1	0.0237	0.0237	0.05	0.474	Zone_1

Calculate Zone

Calculate Zone



18 zones

Calculate Zone

```
#=============================================================
# # Get Xzone
#=============================================================

def getXzone(_xp, _xUnit):
    if _xp < _xUnit:
        return "x1"
    else:
        if (_xp >= _xUnit) and (_xp <= 2*_xUnit):
            return "x2"
        else:
            if (_xp >= 2*_xUnit) and (_xp < 3*_xUnit):
                return "x3"
            else:
                if (_xp >= 3*_xUnit) and (_xp < 4*_xUnit):
                    return "x4"
                else:
                    if (_xp >= 4*_xUnit) and (_xp < 5*_xUnit):
                        return "x5"
                    else:
                        return "x6"
```

18 zones

Calculate Zone

```
#=================================================
# # Get Yzone
#=================================================

def getYzone(_yp, _yUnit):
    if _yp < _yUnit:
        return "y1"
    else:
        if (_yp >= _yUnit) and (_yp <= 2*_yUnit):
            return "y2"
        else:
            return "y3"
```

18 zones

Calculate Zone

```
#=============================================================================
# # Calculate Zone (18 zones)
#=============================================================================

def calculate18Zone(_df, _long, _wide):
    _df["18Zones"] = ""
    lendf = len(_df)

    xUnit = _long/6
    yUnit = _wide/3

    dictZones = {"('x1', 'y3')" : "z1", "('x1', 'y2')" : "z2", "('x1', 'y1')" : "z3",\
                 "('x2', 'y3')" : "z4", "('x2', 'y2')" : "z5", "('x2', 'y1')" : "z6",\
                 "('x3', 'y3')" : "z7", "('x3', 'y2')" : "z8", "('x3', 'y1')" : "z9",\
                 "('x4', 'y3')" : "z10", "('x4', 'y2')" : "z11", "('x4', 'y1')" : "z12",\
                 "('x5', 'y3')" : "z13", "('x5', 'y2')" : "z14", "('x5', 'y1')" : "z15",\
                 "('x6', 'y3')" : "z16", "('x6', 'y2')" : "z17", "('x6', 'y1')" : "z18"}

    for i in range (0, lendf):
        xZone = getXzone(_df.loc[i, "x_pos"], xUnit)
        yZone = getYzone(_df.loc[i, "y_pos"], yUnit)
        _df.loc[i, "18Zones"] = dictZones[str((xZone, yZone))]
    return _df
```

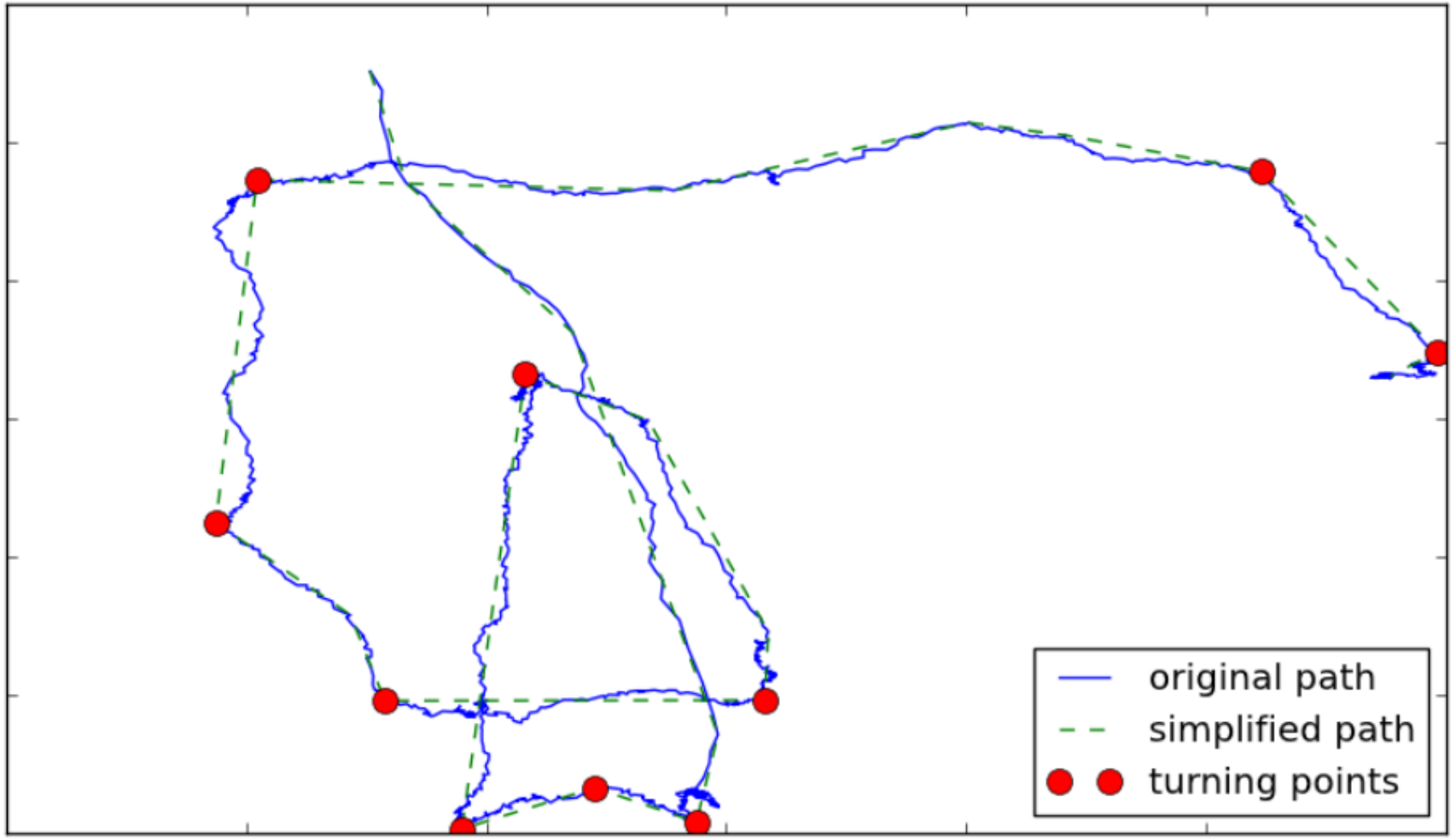
18 zones

```
df = calculate18Zones(df, x, y)
df
```

Calculate Zone

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	distance	totalDistance	difTime	velocity	3Zones	18Zones
0	1.237747e+12	9	50.875526	25.792036	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z12
1	1.237747e+12	8	28.525500	41.909308	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z5
2	1.237747e+12	2	21.108100	12.481253	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z6
3	1.237747e+12	6	45.247400	26.818432	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z9
4	1.237747e+12	7	32.231844	12.636959	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z9
5	1.237747e+12	5	28.107198	40.277207	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z5
6	1.237747e+12	11	41.571300	61.727806	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z7
7	1.237747e+12	1	4.295718	45.317342	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z2
8	1.237747e+12	4	35.724664	29.717300	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z9
9	1.237747e+12	3	26.620200	56.002000	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z5
10	1.237747e+12	10	51.444286	49.769524	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z11
11	1.237747e+12	9	50.907026	25.734036	1	1	0.0660	0.0660	0.05	1.320	Zone_2	z12
12	1.237747e+12	8	28.521800	41.885908	1	1	0.0237	0.0237	0.05	0.474	Zone_1	z5
13	1.237747e+12	2	21.154600	12.468753	1	1	0.0482	0.0482	0.05	0.964	Zone_1	z6

Turning points



Caculate turning points

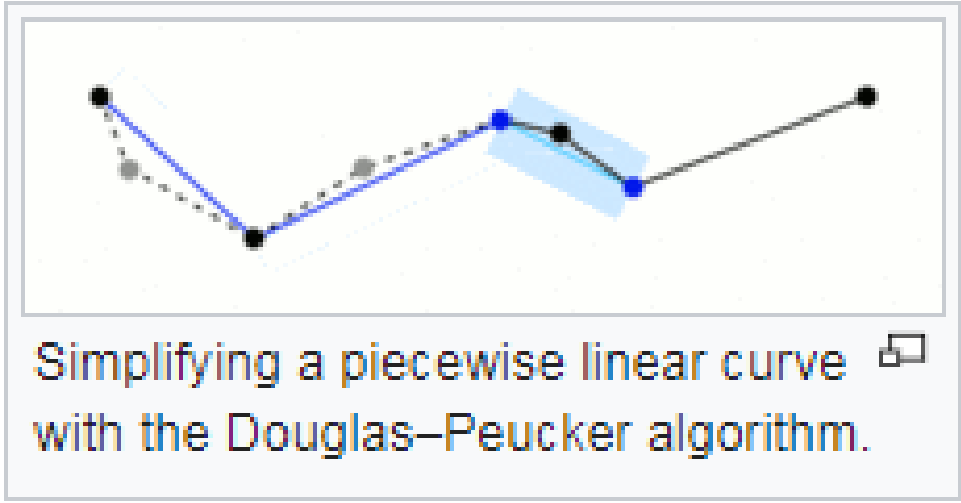
Ramer – Douglas – Peucker algorithm

https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm

Given a curve composed of line segments to find a similar curve with fewer points.

Caculate turning points

The algorithm defines 'dissimilar' based on the maximum distance between the original curve and the simplified curve.



Simplifying a piecewise linear curve with the Douglas–Peucker algorithm.

The simplified curve consists of a subset of the points that defined the original curve.

Caculate turning points

The true turning point is always debatable and will depend on the angle that one specifies that has to lie between points.

1. Calculate distance between subsequent points
2. Calculate angle between subsequent points
3. Look how distance / angle changes between subsequent points

Caculate turning points

```
#=============================================================
# # Calculate the distance bettwen two points
#=============================================================

def distance(_p1, _p2):
    return  sqrt((_p1[0] - _p2[0]) ** 2 + (_p1[1] - _p2[1]) ** 2)
```

```
#=============================================================
# # pointLineDistance
#=============================================================

def pointLineDistance(_point, _start, _end):
    if (_start == _end):
        return distance(_point, _start)
    else:
        n = abs((_end[0] - _start[0]) * (_start[1] - _point[1]) - \
                (_start[0] - _point[0]) * (_end[1] - _start[1]))
        d = sqrt((_end[0] - _start[0]) ** 2 + (_end[1] - _start[1]) ** 2)
        return n / d
```

Caculate turning points

Use the Ramer-Douglas-Peucker (RDP) algorithm to simplify the path

```
#==============================================================================
# # Reduces a series of points to a simplified version that loses detail,
# # but maintains the general shape of the series.
#==============================================================================

def rdp(_points, _epsilon):
    dmax = 0.0
    index = 0
    lenP = len(_points) - 1

    # Find the point with the maximum distance
    for i in range(1, lenP):
        d = pointLineDistance(_points[i], _points[0], _points[-1])
        if d > dmax:
            index = i
            dmax = d

    # If max distance is greater than epsilon, recursively simplify
    if (dmax >= _epsilon):
        # Recursive call
        results = rdp(_points[:index+1], _epsilon)[:-1] +\
                  rdp(_points[index:], _epsilon)
    else:
        results = [_points[0], _points[-1]]
    return results
```

Caculate turning points

```
#==========================================================================
# Returns the angles between vectors.
#    Parameters:
#    dir is a 2D-array of shape (N,M) representing N vectors in M-dimensional space.
#
#    The return value is a 1D-array of values of shape (N-1,), with each value
#    between 0 and pi.
#
#    0 implies the vectors point in the same direction
#    pi/2 implies the vectors are orthogonal
#    pi implies the vectors point in opposite directions
#==========================================================================

def angle(_arr):
    dir2 = _arr[1:]
    dir1 = _arr[:-1]
    return np.arccos((dir1*dir2).sum(axis=1)/\
                     (np.sqrt((dir1**2).sum(axis=1)*(dir2**2).sum(axis=1))))
```

Caculate turning points

```
#=============================================================================
# # Use the Ramer-Douglas-Peucker algorithm to simplify the path
#=============================================================================

def turn_points(_df, _xCol, _yCol, _tolerance, _min_angle, _turnArr, _d):
    num_samples = len(_df)
    if (num_samples > 0):
        _min_angle = np.pi * _min_angle
        npcoord = _df[[_xCol, _yCol]].as_matrix()

        simplified = np.array(rdp(npcoord.tolist(), _tolerance))
        sx, sy = simplified.T

        directions = np.diff(simplified, axis=0)
        theta = angle(directions)

        idx = np.where(theta > _min_angle)[0] + 1
        lenIdx = len(idx)

        for i in range(0, lenIdx):
            exp = (_df[_xCol] == sx[idx[i]] ) & (_df[_yCol] == sy[idx[i]])

            # add index into turnArr
            _turnArr.append(_df.index[exp])

        plotTurning(_df[_xCol], _df[_yCol], sx, sy, idx, _d)

        return _turnArr
    else:
        return _turnArr
```

Caculate turning points

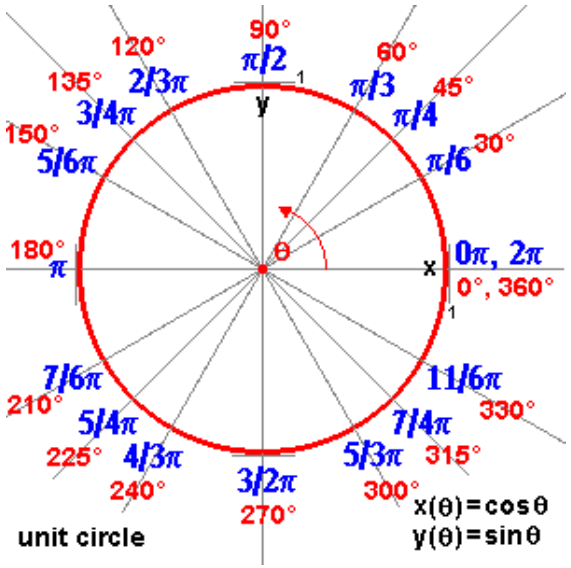
```
#=============================================================================
# # Plot turning point of a player
#=============================================================================

def plotTurning(coord_x, coord_y, sx, sy, idx, _d):

    fig = plt.figure()
    figsize=(1, 1)
    ax = fig.add_subplot(111)
    ax.plot(coord_x, coord_y, 'b-', label='original path')
    ax.plot(sx, sy, 'g--', label='simplified path')
    ax.plot(sx[idx], sy[idx], 'ro', markersize = 15, label='turning points')

    plt.legend(loc='best')
    plt.title('Device : '+ str(_d))
    plt.show()
```

Caculate turning points

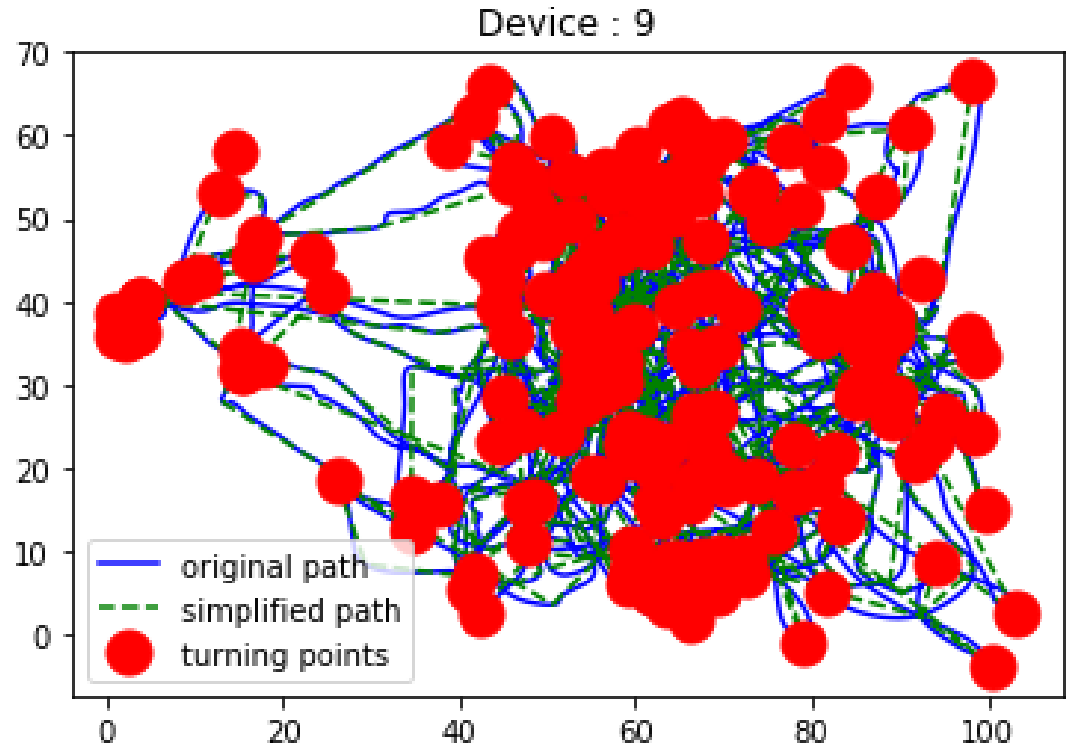
```
#=============================================================================
# Calculate Turning Points
#=============================================================================

def calculateTurningPoints(_df, _devices, _tolerance, _min_angle):
    turnArr = []
    for d in _devices:
        dfd = _df[_df['device_id'] == d]
        print "\n\nTurning points of device >>> ", d
        turnArr = turn_points(dfd, 'x_pos', 'y_pos', _tolerance, _min_angle, turnArr, d)

    # Add turnAng to DataFrame
    _df['turnAng'] = 0
    for i in turnArr:
        _df.loc[i, 'turnAng'] = 1
    return df
```

Caculate turning points

```
#################### TEST

tolerance = 2
min_angle = math.pi/6
df_turning = calculateTurningPoints(df,\
             devices, tolerance, min_angle)
```

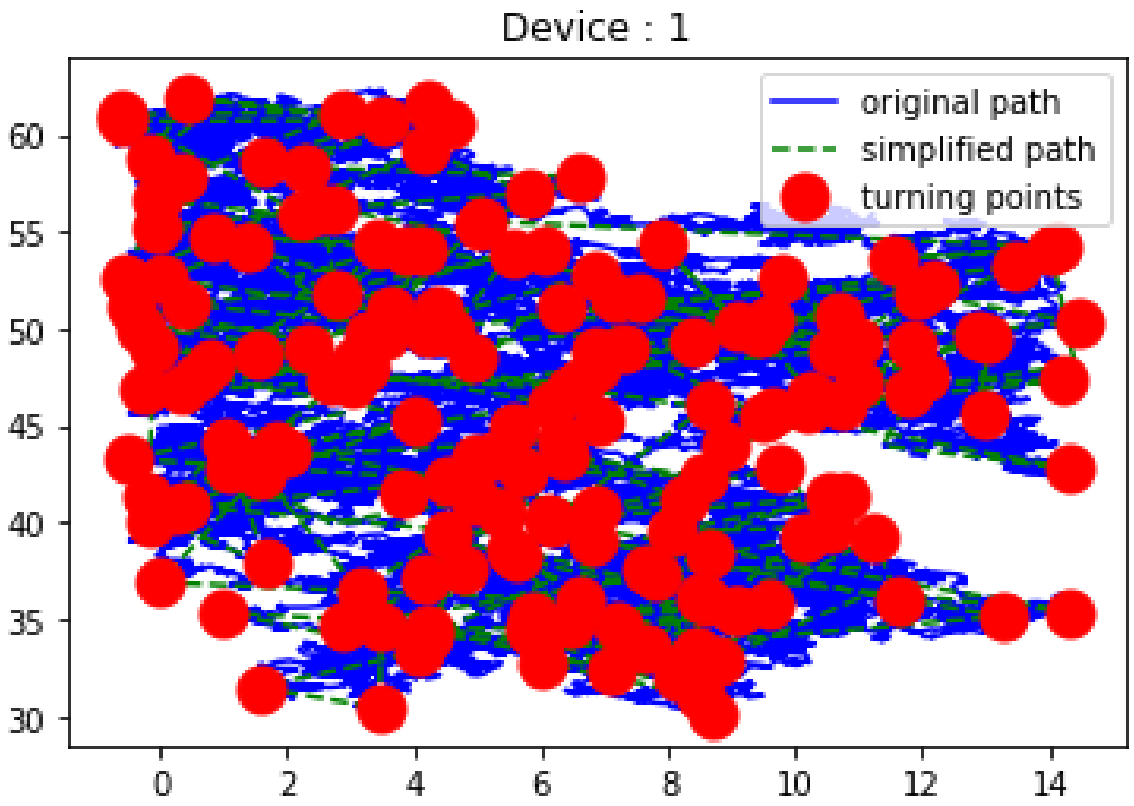



```
Turning points of device >>>  9
```

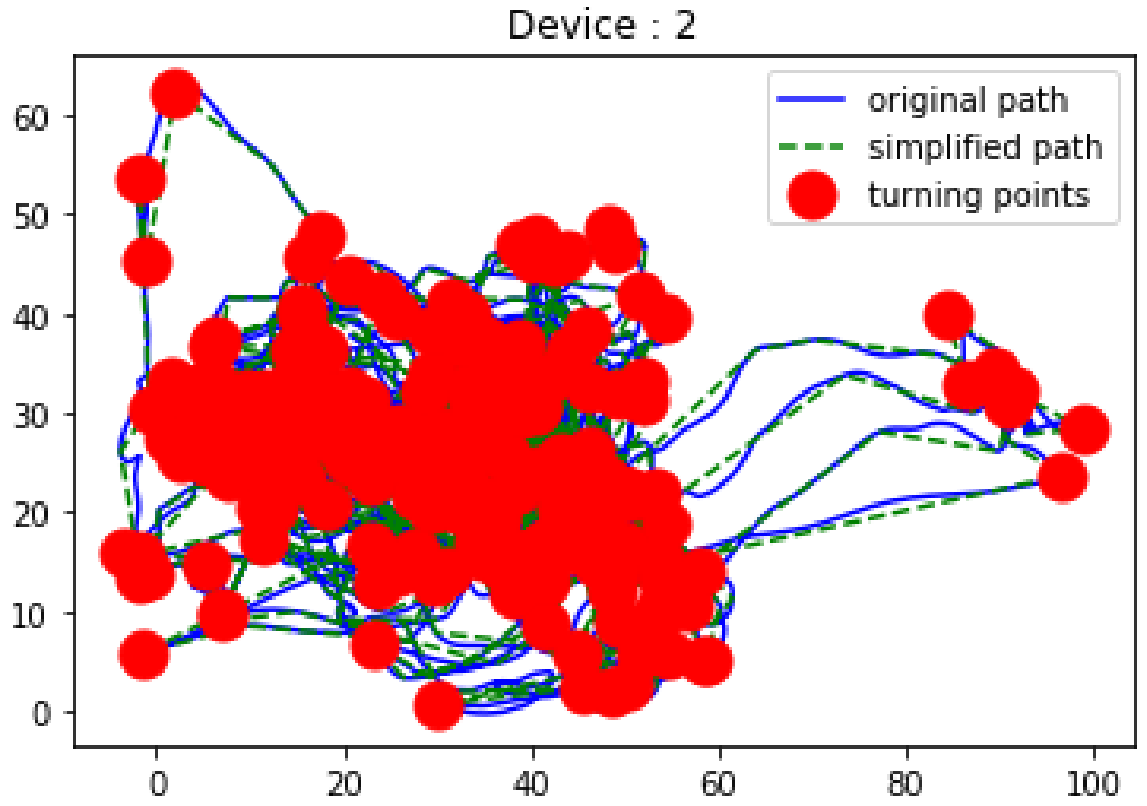


Caculate turning points

```
Turning points of device >>  1
```

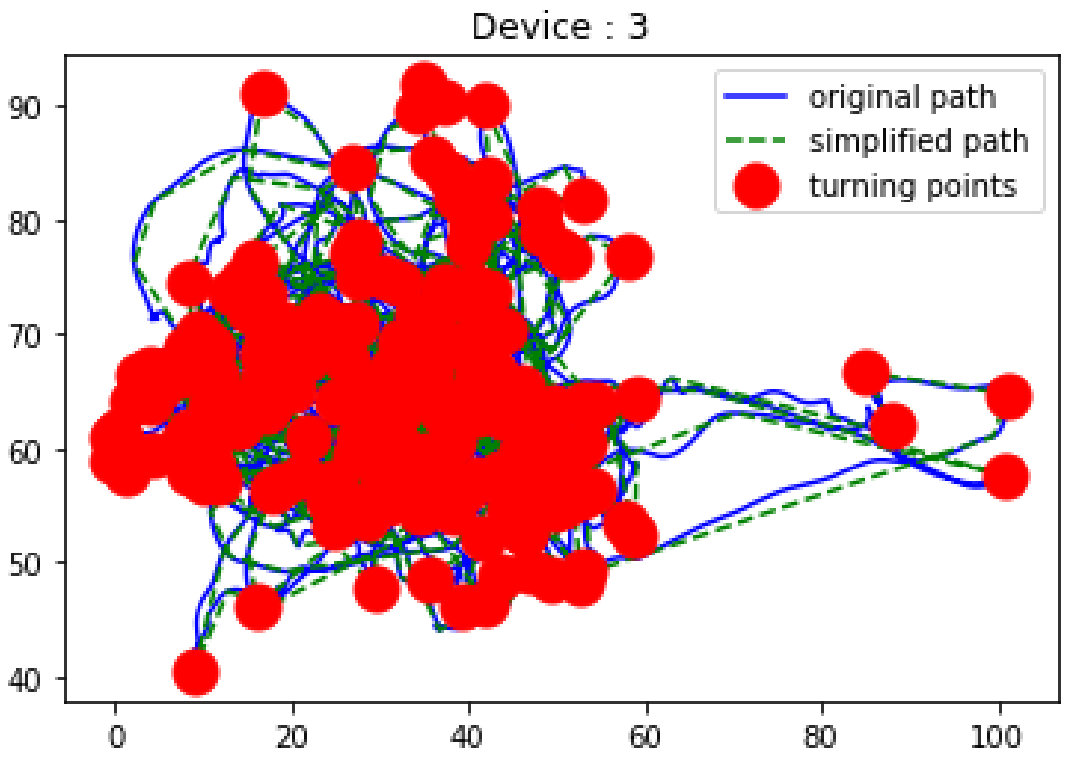



```
Turning points of device >>  2
```

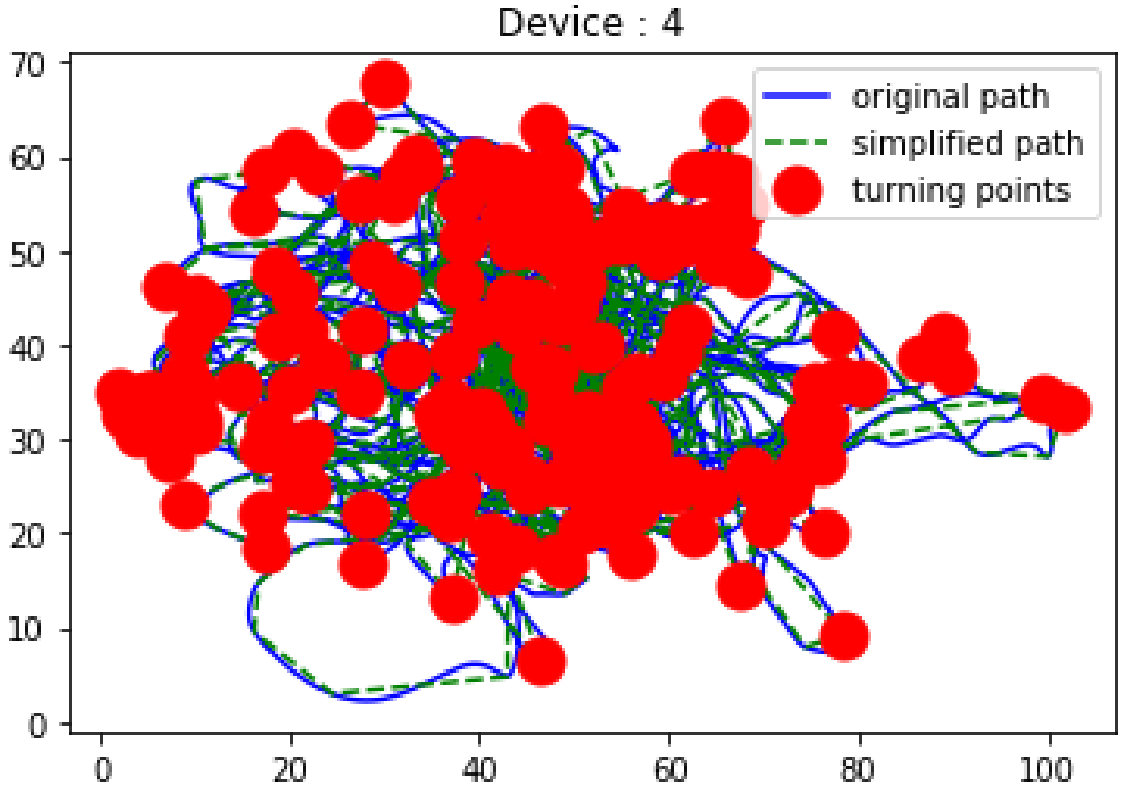


Caculate turning points

```
Turning points of device >>>  3
```

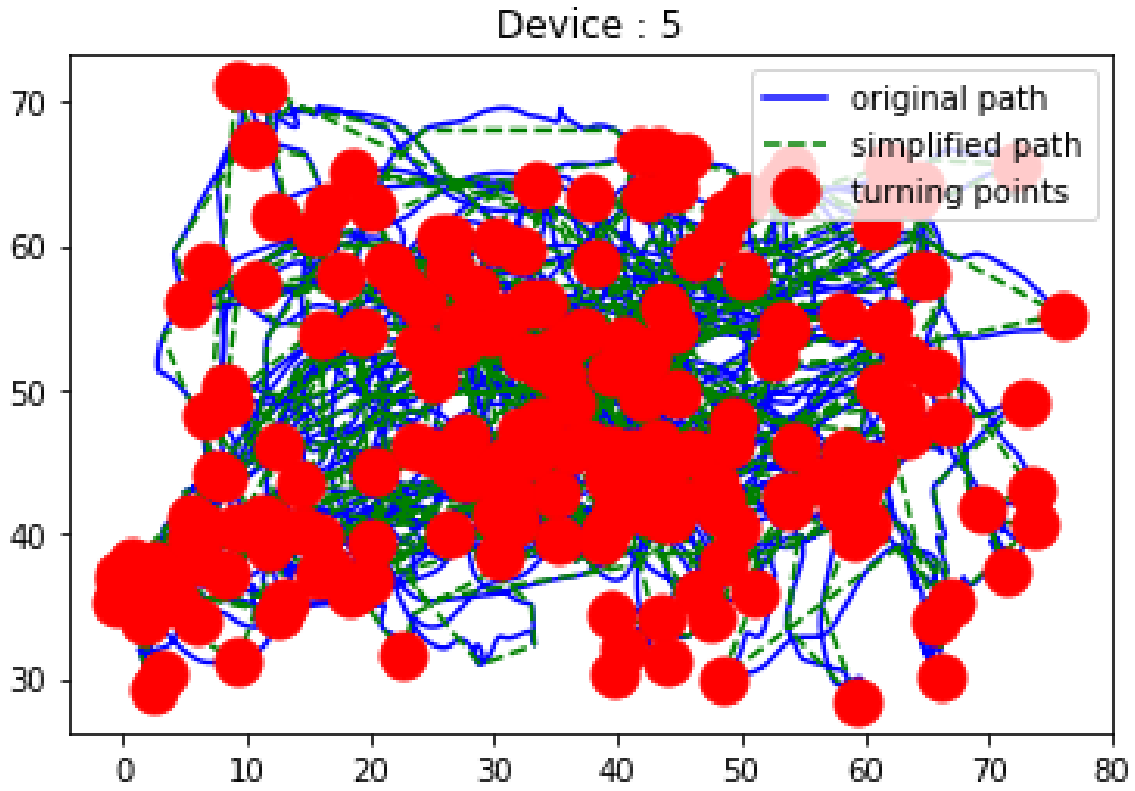



```
Turning points of device >>>  4
```

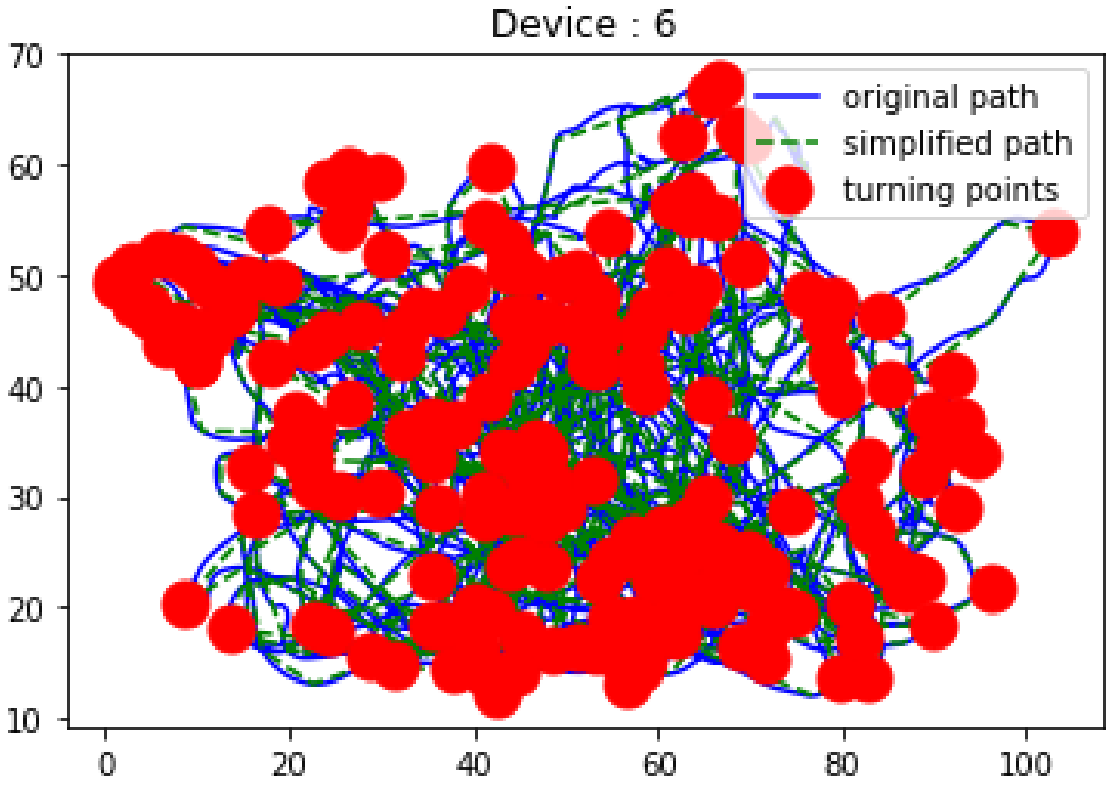


Caculate turning points

```
Turning points of device >>>  5
```

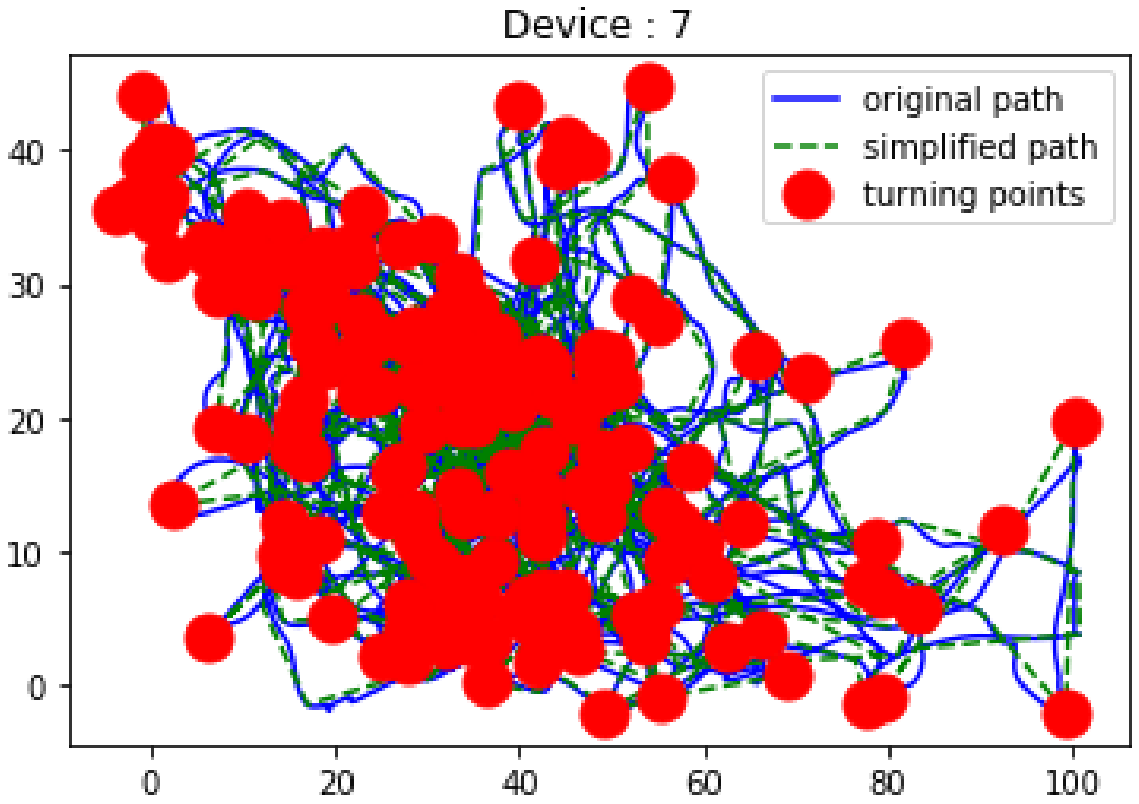



```
Turning points of device >>>  6
```

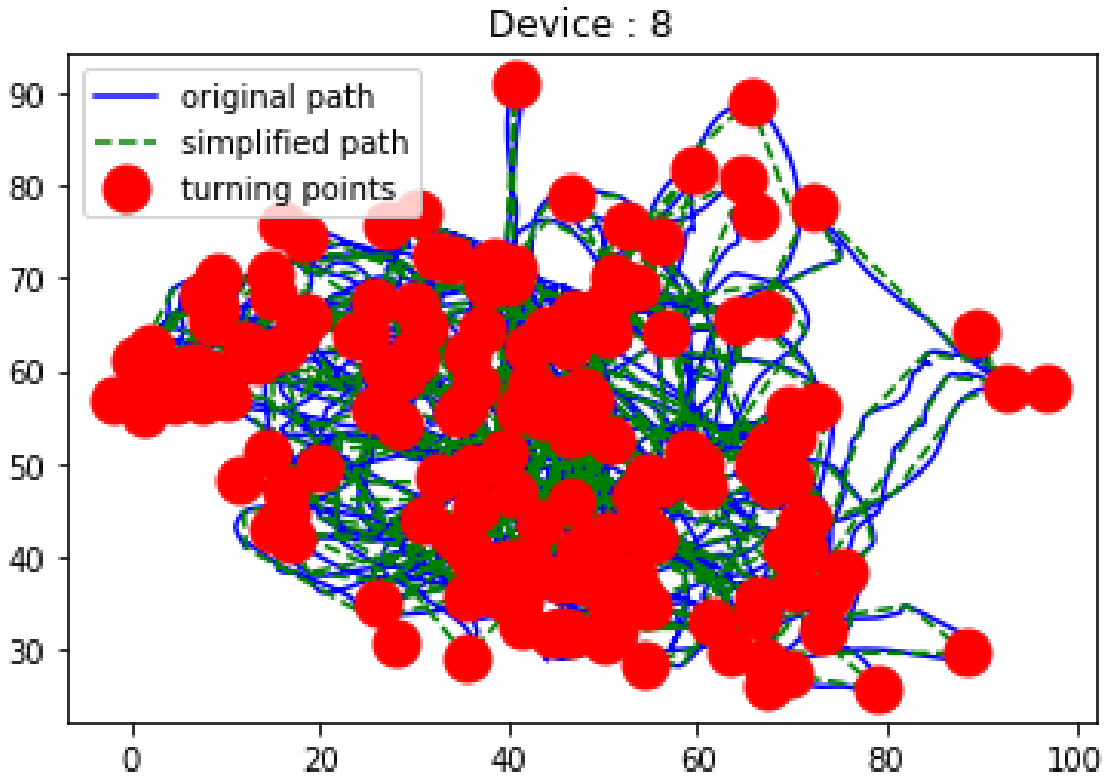


Caculate turning points

```
Turning points of device >>  7
```

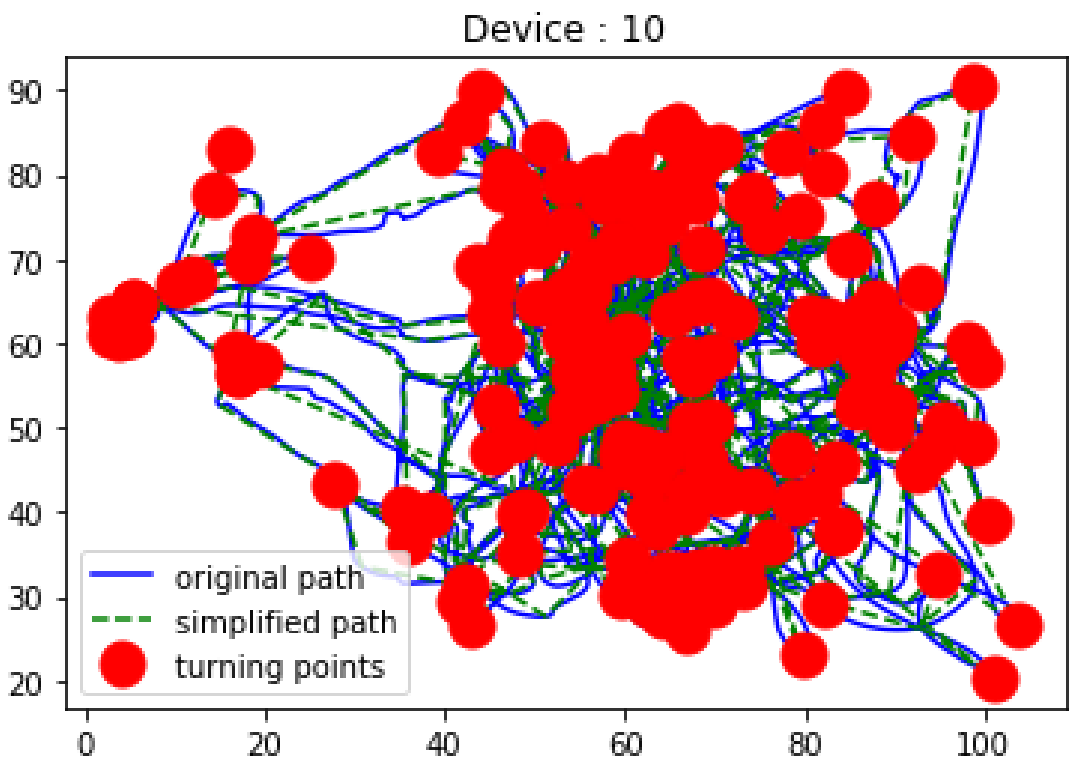



```
Turning points of device >>  8
```

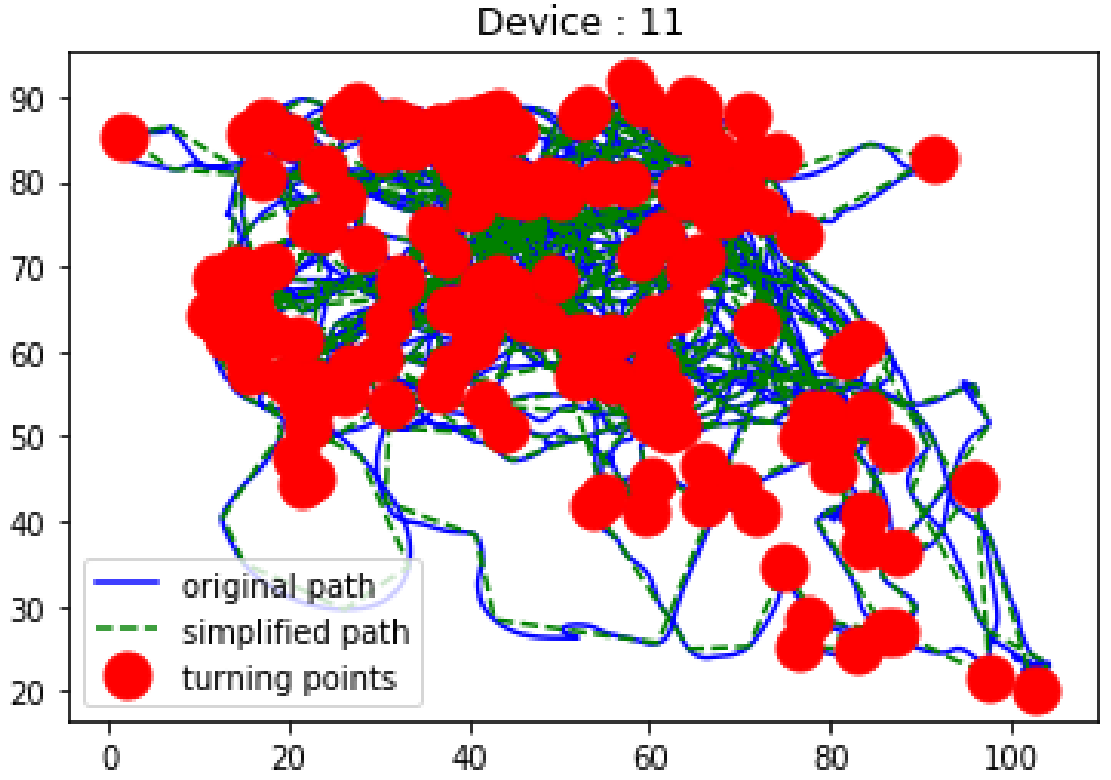


Caculate turning points

```
Turning points of device >>>  10
```




```
Turning points of device >>>  11
```



Caculate turning points

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	distance	totalDistance	difTime	velocity	3Zones	18Zones	turnAng
0	1.237747e+12	9	50.875526	25.792036	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z12	0
1	1.237747e+12	8	28.525500	41.909308	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z5	0
2	1.237747e+12	2	21.108100	12.481253	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z6	0
3	1.237747e+12	6	45.247400	26.818432	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z9	0
4	1.237747e+12	7	32.231844	12.636959	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z9	0
5	1.237747e+12	5	28.107198	40.277207	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z5	0
6	1.237747e+12	11	41.571300	61.727806	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z7	0
7	1.237747e+12	1	4.295718	45.317342	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z2	0
8	1.237747e+12	4	35.724664	29.717300	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z9	0
9	1.237747e+12	3	26.620200	56.002000	1	1	0.0000	0.0000	0.00	0.000	Zone_1	z5	0
10	1.237747e+12	10	51.444286	49.769524	1	1	0.0000	0.0000	0.00	0.000	Zone_2	z11	0
11	1.237747e+12	9	50.907026	25.734036	1	1	0.0660	0.0660	0.05	1.320	Zone_2	z12	0
12	1.237747e+12	8	28.521800	41.885908	1	1	0.0237	0.0237	0.05	0.474	Zone_1	z5	0

Describe Data

```
#================
# # Describe Data
#================

df.describe()
```

	timestamp	device_id	x_pos	y_pos	matchTime_minute	matchTime_second	distance	totalDistance	difTime	velocity	turnAng
count	5.940110e+05	594011.00000	594011.000000	594011.000000	594011.000000	594011.000000	594011.000000	594011.000000	594011.000000	594011.000000	594011.000000
mean	1.237748e+12	6.00000	40.995397	43.856025	23.000426	30.499454	0.112526	3062.034685	0.049999	2.250519	0.004414
std	7.794380e+05	3.16228	23.624175	19.094993	12.987441	17.318422	0.070992	1757.739790	0.000215	1.419839	0.066292
min	1.237747e+12	1.00000	-3.851690	-3.709834	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.237747e+12	3.00000	22.013650	29.903948	12.000000	15.000000	0.058500	1598.513550	0.050000	1.170000	0.000000
50%	1.237748e+12	6.00000	42.212144	43.642300	23.000000	30.000000	0.099900	3025.199000	0.050000	1.998000	0.000000
75%	1.237749e+12	9.00000	57.166875	57.709330	34.000000	45.000000	0.155600	4474.828350	0.050000	3.112000	0.000000
max	1.237749e+12	11.00000	104.214000	91.831700	46.000000	60.000000	0.488900	7938.709100	0.050000	9.778000	1.000000

Describe Data

```
#=============================================
# # Describe Data of each device
#=============================================

for d in devices:
    dfi = df[df["device_id"] == d]
    print "Device >>> ", d, "\n", dfi.describe()
```

Describe Data

```
Device >>  1

          timestamp
count  5.400100e+04
mean   1.237748e+12
std    7.794445e+05
min    1.237747e+12
25%    1.237747e+12
50%    1.237748e+12
75%    1.237749e+12
max    1.237749e+12

       matchTime_se
count      54001.00
mean          30.49
std           17.31
min            1.00
25%           15.00
50%           30.00
75%           45.00
max           60.00

           velocity
count  54001.000000
mean       2.940208
std        1.242827
min        0.000000
25%        2.008000
50%        2.882000
75%        3.812000
max        6.644000
```

```
Device >>  3

          timestamp  devi
count  5.400100e+04    54
mean   1.237748e+12
std    7.794445e+05
min    1.237747e+12
25%    1.237747e+12
50%    1.237748e+12
75%    1.237749e+12
max    1.237749e+12

       matchTime_second
count      54001.000000
mean          30.499454
std           17.318568
min            1.000000
25%           15.000000
50%           30.000000
75%           45.000000
max           60.000000

           velocity      1
count  54001.00000  54001
mean       1.97829      0
std        1.27514      0
min        0.00000      0
25%        1.03200      0
50%        1.72600      0
75%        2.74600      0
max        9.76800      1
```

```
Device >>  10
          timestamp  device_id         x_pos         y_pos  matchTime_minute  \
count  5.400100e+04    54001.0  54001.000000  54001.000000      54001.000000
mean   1.237748e+12       10.0     60.239701     56.744580         23.000426
std    7.794445e+05        0.0     21.614398     15.013511         12.987550
min    1.237747e+12       10.0      2.723298     20.267654          1.000000
25%    1.237747e+12       10.0     50.836486     45.256924         12.000000
50%    1.237748e+12       10.0     62.444511     57.040736         23.000000
75%    1.237749e+12       10.0     73.984686     66.982824         34.000000
max    1.237749e+12       10.0    103.882186     90.441724         46.000000

       matchTime_second      distance  totalDistance       difTime  \
count      54001.000000  54001.000000   54001.000000  54001.000000
mean          30.499454      0.109908    2994.837325      0.049999
std           17.318568      0.073984    1665.685696      0.000215
min            1.000000      0.000000       0.000000      0.000000
25%           15.000000      0.055200    1579.902600      0.050000
50%           30.000000      0.095600    2987.869800      0.050000
75%           45.000000      0.150000    4404.093500      0.050000
max           60.000000      0.483500    5935.146500      0.050000

           velocity       turnAng
count  54001.000000  54001.000000
mean       2.198162      0.006611
std        1.479688      0.081040
min        0.000000      0.000000
25%        1.104000      0.000000
50%        1.912000      0.000000
75%        3.000000      0.000000
max        9.670000      1.000000
```

DATA

1. Timestamp
2. Devices_id
3. Positions (x, y)
4. matchTime_minute
5. matchTime_second
6. Distance
7. totalDistance
8. Velocity
9. 3Zones
10. 18Zones
11. Turning Points

Write to CSV

```
#=============================================================
# Write Data to CSV file
#=============================================================

def writeCSV(_df, _csvName, _sep):
    _df.to_csv(_csvName, sep= _sep, header=True, index=False)
```

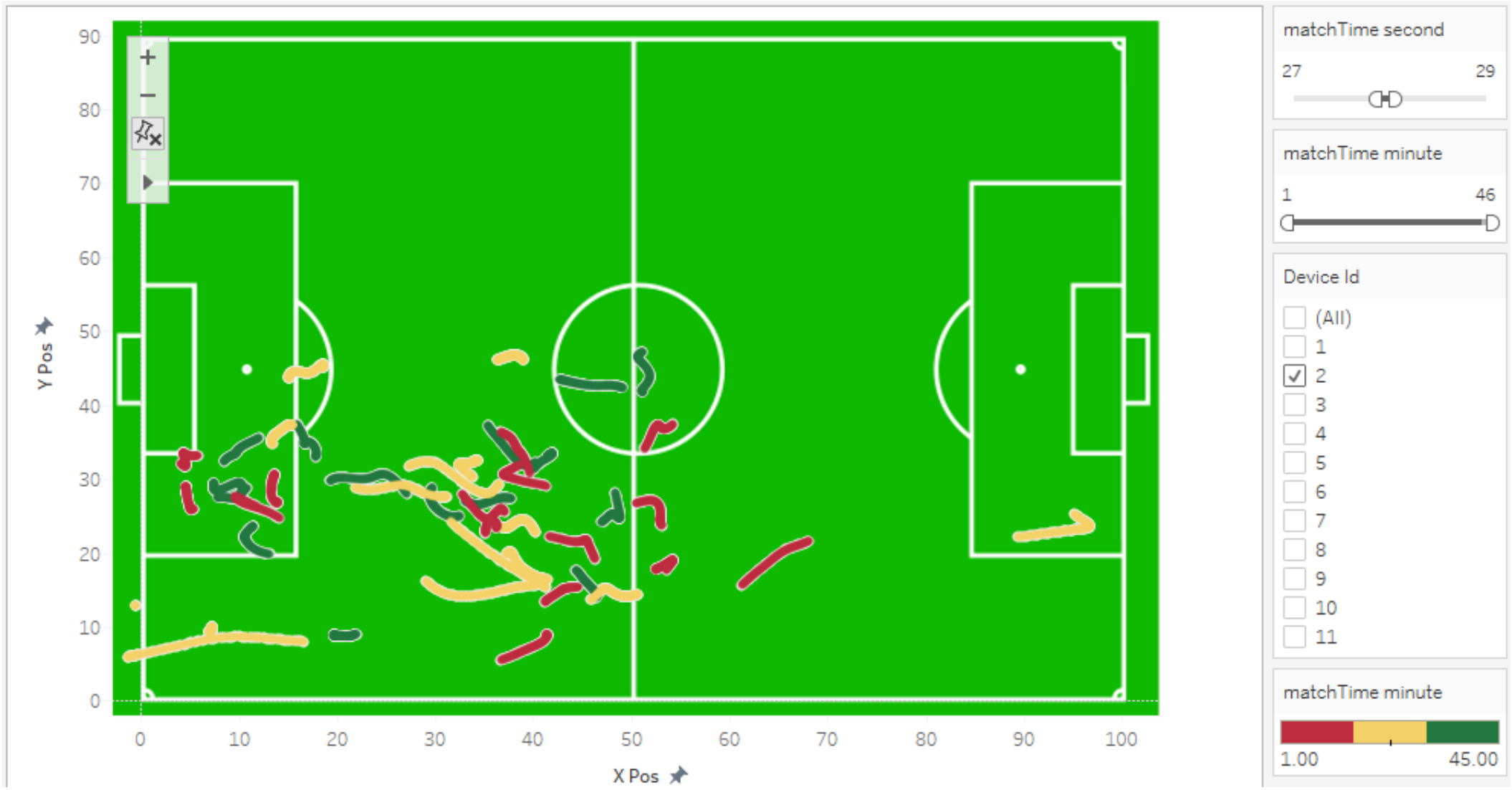
```
#=============================================================
# # Write to csv file
#=============================================================

writeCSV(df, "ThuyLe_P1_Demo_TurningPoints.csv", sep)
```

OUTLINE

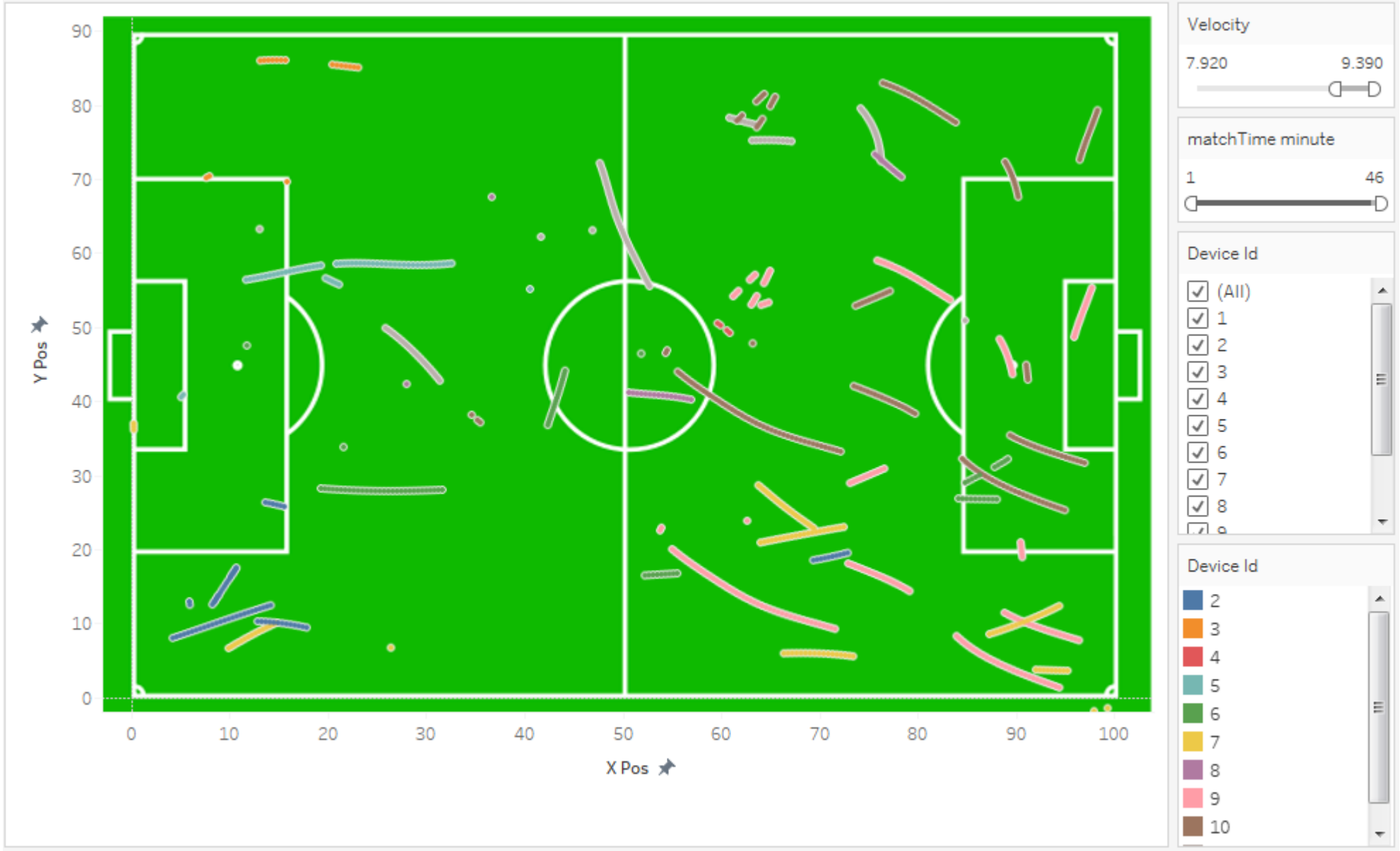
1. Data analysis with Python
2. Data visualization with Tableau


The 4-4-2 Formation
3
11
5
8
10
1
9
4
6
2
7

Visualize


90
80
70
60
50
40
30
20
10
0
Y Pos
0
10
20
30
40
50
60
70
80
90
100
X Pos
matchTime second
27
29
matchTime minute
1
46
Device Id
(All)
1
2
3
4
5
6
7
8
9
10
11
matchTime minute
1.00
45.00

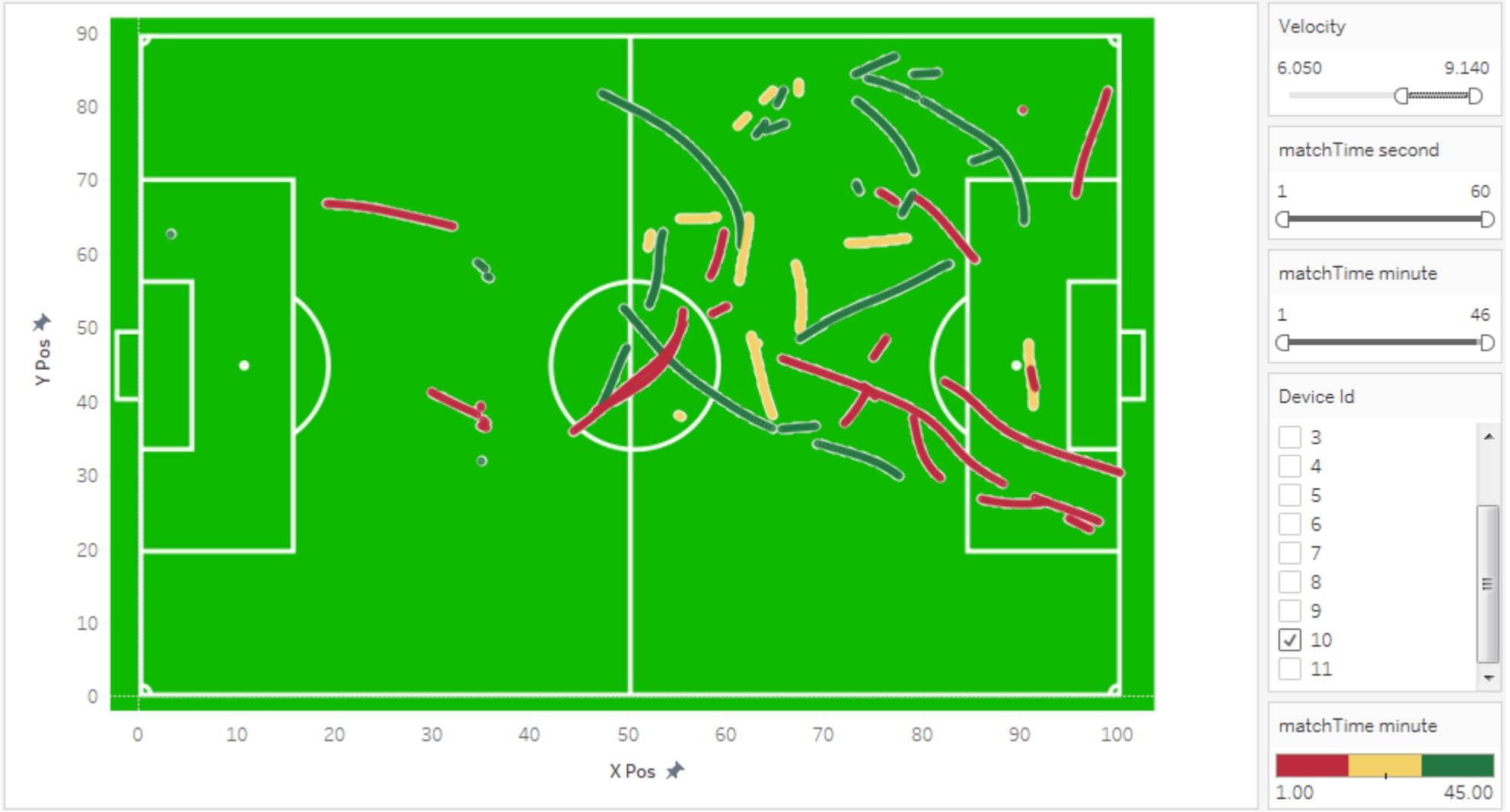
Visualize


Y Pos
X Pos
0
10
20
30
40
50
60
70
80
90
100
Velocity
7.920
9.390
matchTime minute
1
46
Device Id
(All)
1
2
3
4
5
6
7
8
Device Id
2
3
4
5
6
7
8
9
10

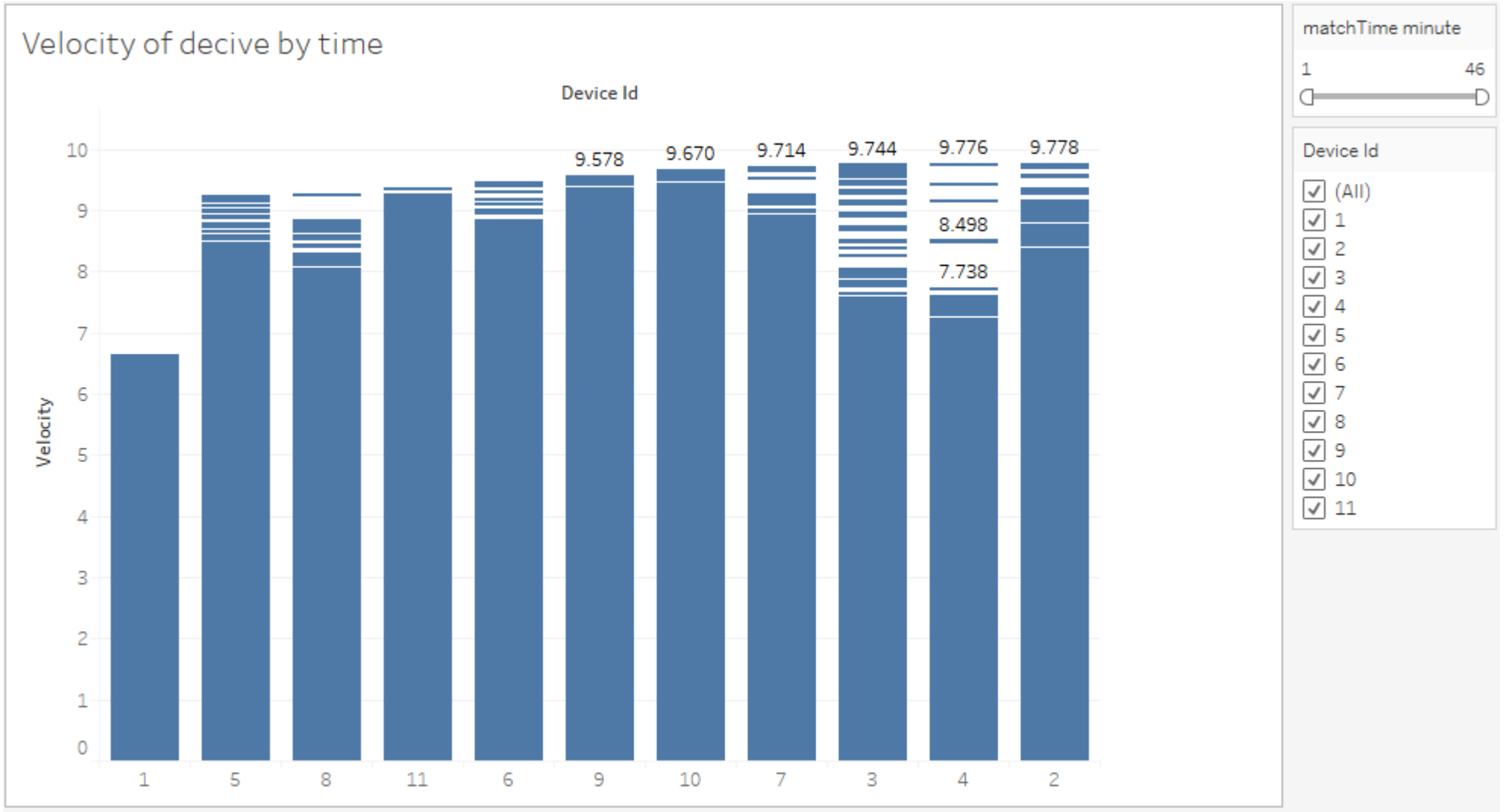
Visualize

Velocity (>6.05 m/s²)

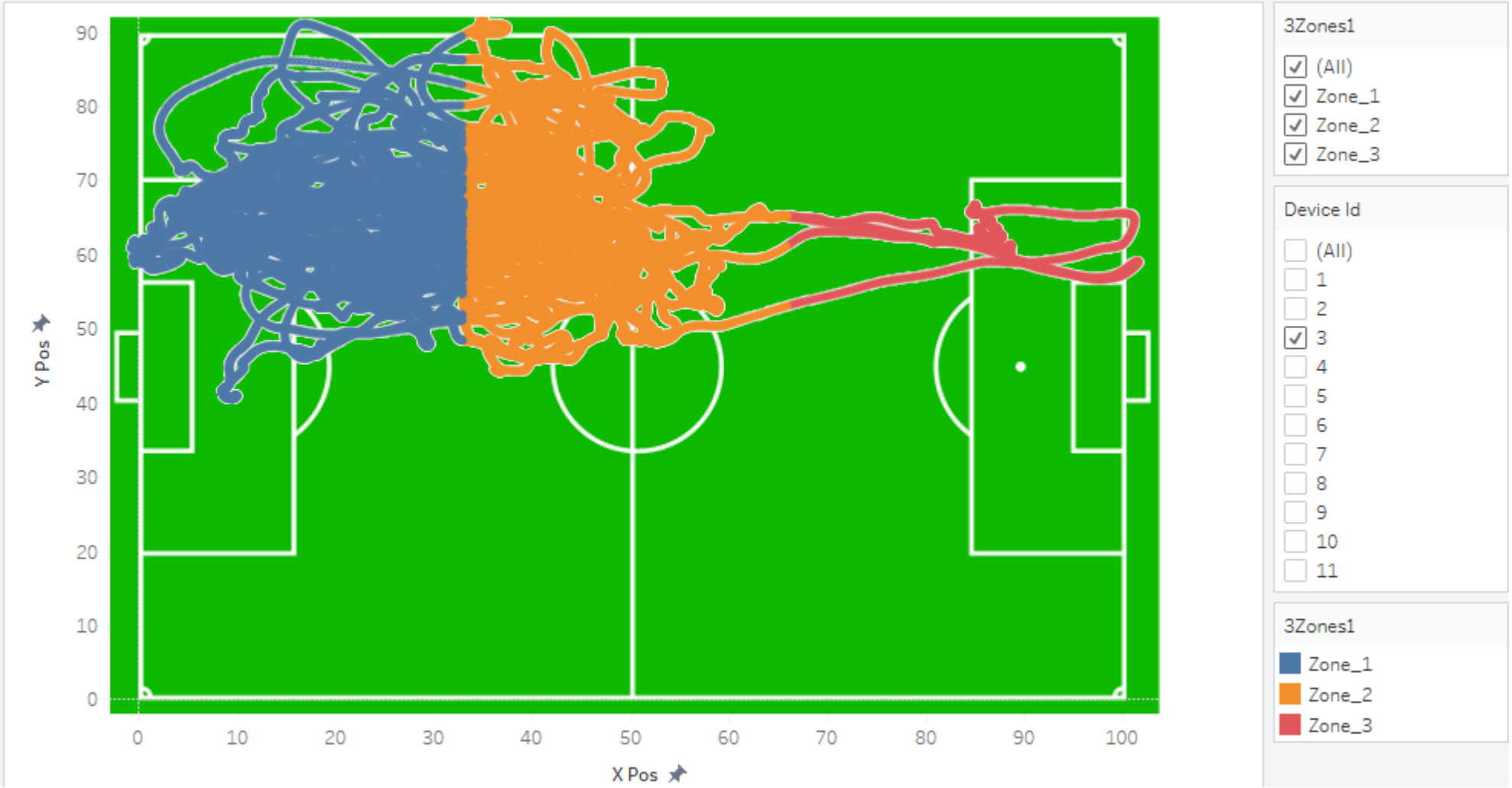
Device_id = 10



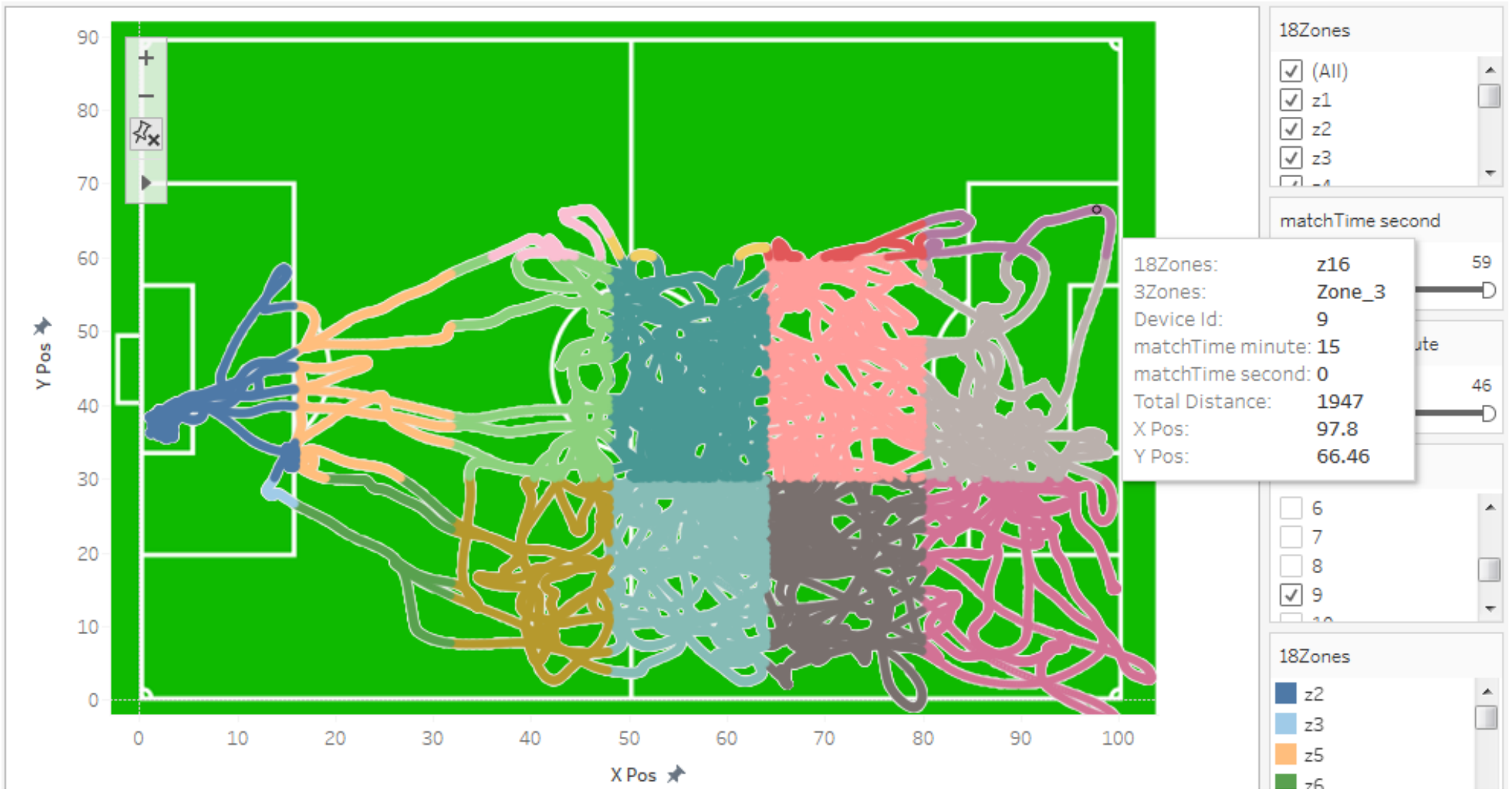
Visualize


Velocity of decive by time
Device Id
Velocity
0
1
2
3
4
5
6
7
8
9
10
1
5
8
11
6
9
10
7
3
4
2
9.578
9.670
9.714
9.744
9.776
8.498
7.738
9.778
matchTime minute
1
46
Device Id
(All)
1
2
3
4
5
6
7
8
9
10
11

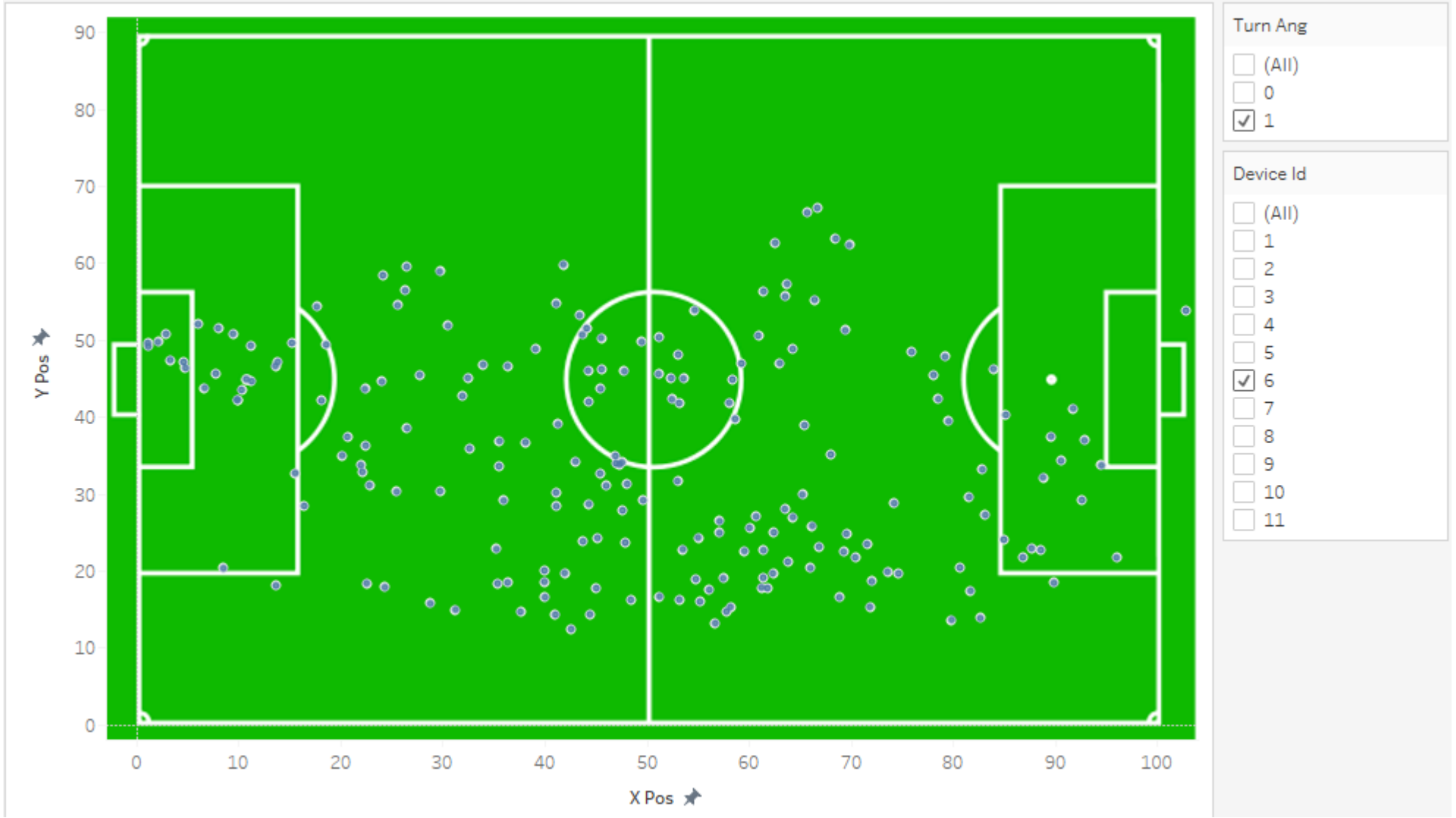
Visualize


90
80
70
60
50
40
30
20
10
0
Y Pos
0
10
20
30
40
50
60
70
80
90
100
X Pos
3Zones1
(All)
Zone_1
Zone_2
Zone_3
Device Id
(All)
1
2
3
4
5
6
7
8
9
10
11
3Zones1
Zone_1
Zone_2
Zone_3

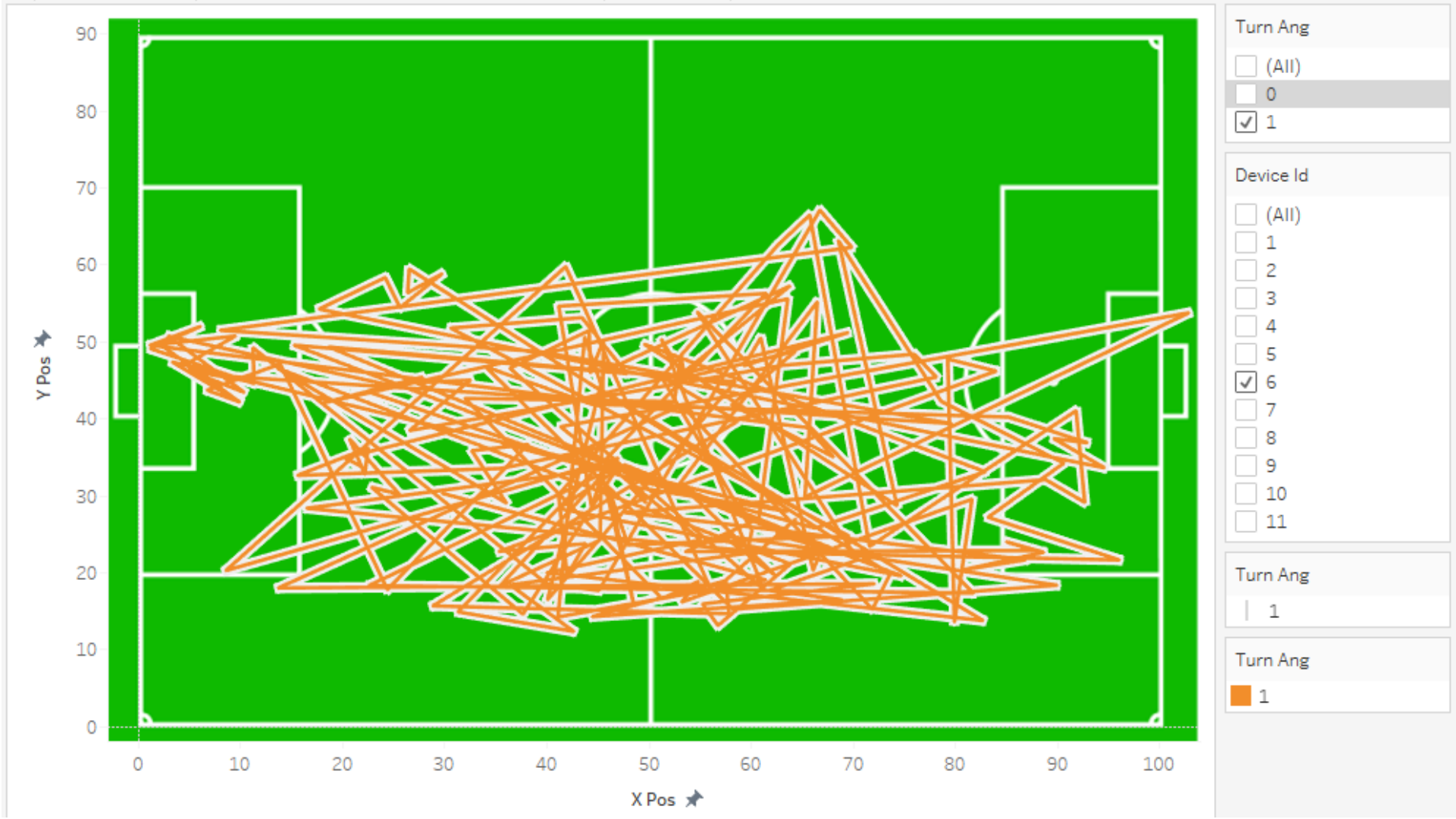
Visualize


Y Pos
X Pos
0
10
20
30
40
50
60
70
80
90
100
18Zones:
z16
3Zones:
Zone_3
Device Id:
9
matchTime minute:
15
matchTime second:
0
Total Distance:
1947
X Pos:
97.8
Y Pos:
66.46
18Zones
(All)
z1
z2
z3
matchTime second
59
46
6
7
8
9
18Zones
z2
z3
z5

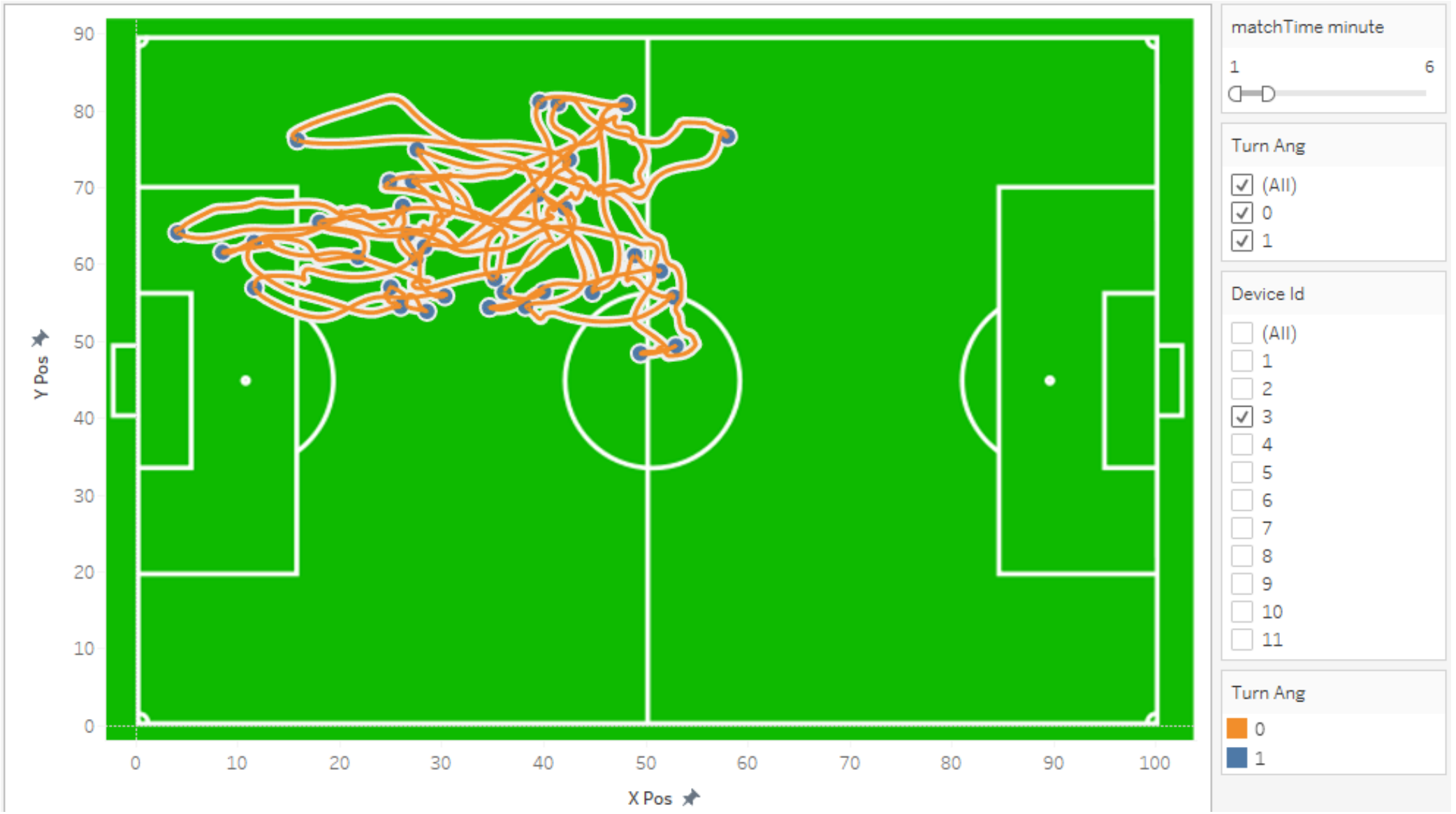
Visualize


Y Pos
X Pos
0
10
20
30
40
50
60
70
80
90
100
Turn Ang
(All)
0
1
Device Id
(All)
1
2
3
4
5
6
7
8
9
10
11

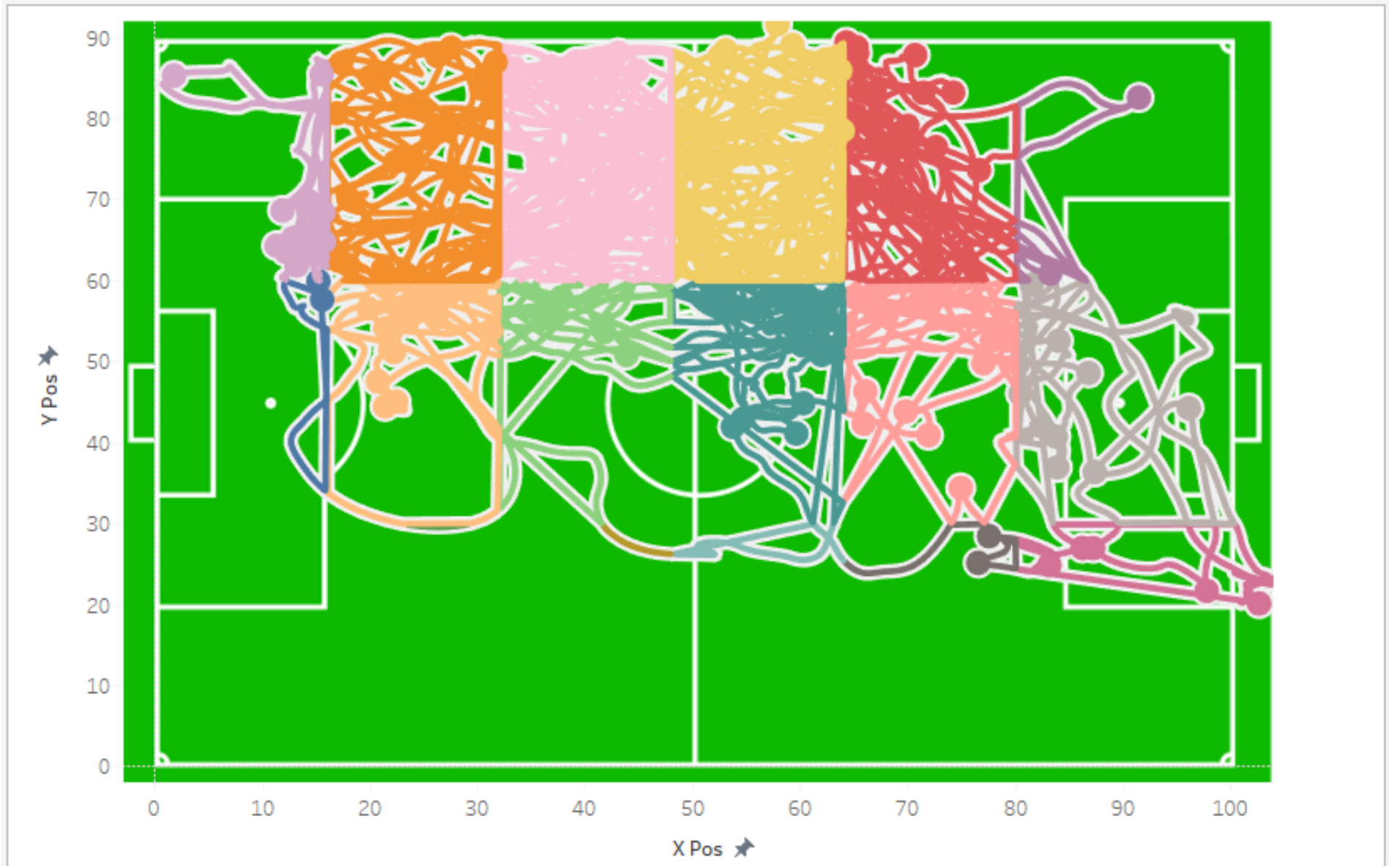
Visualize


Turn Ang
(All)
0
1
Device Id
(All)
1
2
3
4
5
6
7
8
9
10
11
Turn Ang
1
Turn Ang
1
Y Pos
X Pos
0
10
20
30
40
50
60
70
80
90
100

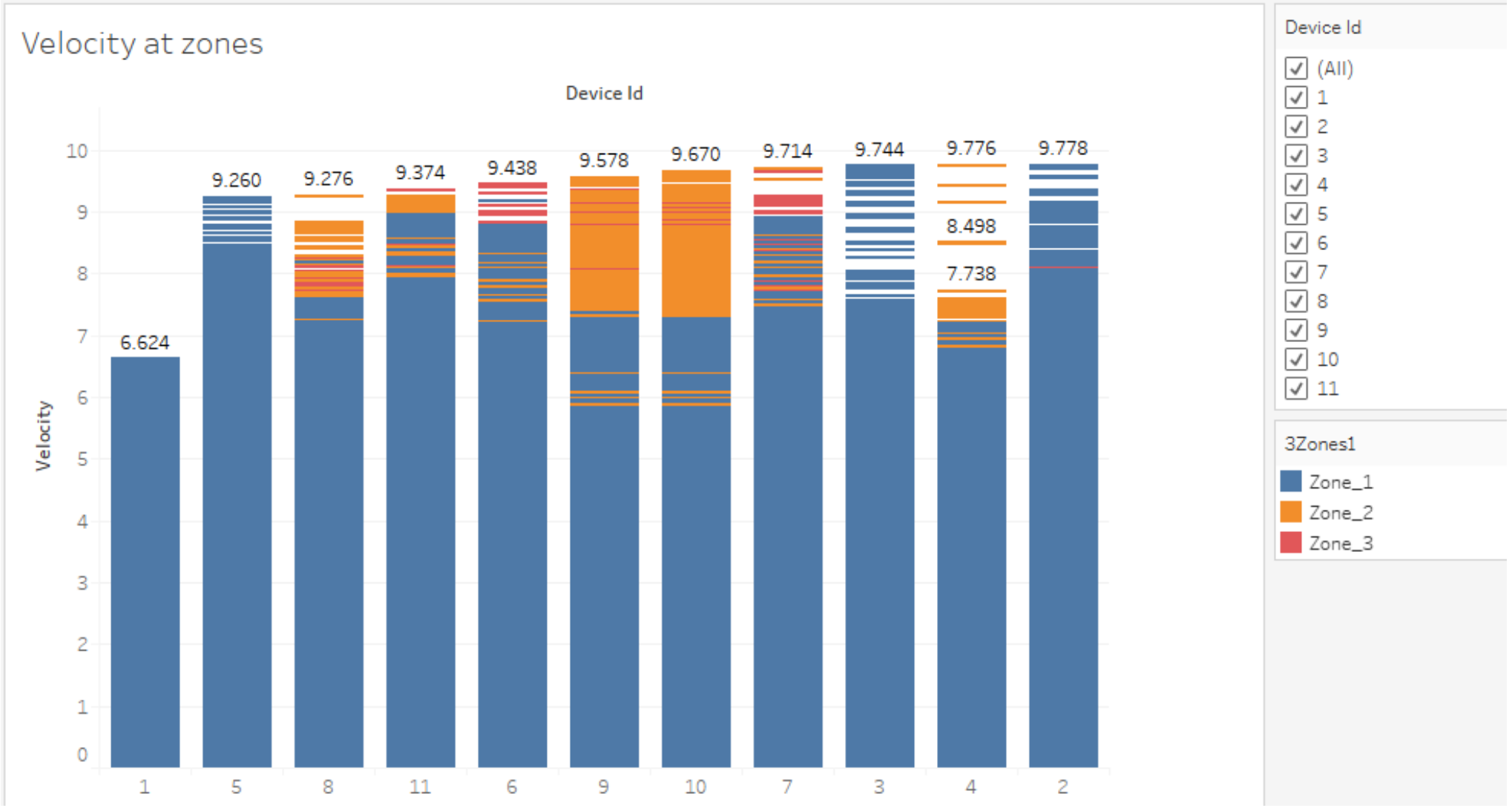
Visualize


Y Pos
X Pos
0
10
20
30
40
50
60
70
80
90
100
matchTime minute
1
6
Turn Ang
(All)
0
1
Device Id
(All)
1
2
3
4
5
6
7
8
9
10
11
Turn Ang
0
1

Visualize



Visualize


Velocity at zones
Device Id
Velocity
0
1
2
3
4
5
6
7
8
9
10
6.624
9.260
9.276
9.374
9.438
9.578
9.670
9.714
9.744
9.776
8.498
7.738
9.778
1
5
8
11
6
9
10
7
3
4
2
Device Id
(All)
1
2
3
4
5
6
7
8
9
10
11
3Zones1
Zone_1
Zone_2
Zone_3

Calculate

1. Real Time
2. Distance
3. Total distance
4. **Velocity**
5. **Zone**
6. **Turning points**

- Energy
- Pass
- Combination
- Create dashboards
- Performance with big Data

….

See you in next workshops

THANK YOU FOR YOUR ATTENTION


Questions
Answers


Tea break 5
minutes