Validating Deployed Model

Using the sketch with the deployed model raw data and predicted values are printed and saved to files. First we define the helper class for plotting and then let's have a look at the data.

In [1]:

```
import matplotlib.pyplot as plt
class visualizer2D():
    def __init__(self):
        self.datasets=[]
        self.labels=[]
        self.point_count=5000
        pass
    def add(self,x,y,label):
        assert x.shape[0]==x.size, "Data shape invalid shape: {}.".format(x.shape[0]
        assert y.shape[0]==y.size, "Data shape invalid shape: {}.".format(y.shape[0]
        self.datasets.append(np.vstack([x,y]).T)
        self.labels.append(label)
        pass
    def show_2D(self,draw_ref_circle=True):
        plt.figure(figsize=(10, 10))
        if draw_ref_circle:
            circle=[]
            import math
            for i in range(500):
                circle.append([math.sin(6.28*i/500),math.cos(6.28*i/500)])
            circle=np.array(circle)
            plt.plot(circle[:,0],circle[:,1],linewidth=1,markersize=0,label="Referen
        i=0
        for data in self.datasets:
            plt.plot(data[:self.point_count,0],data[:self.point_count,1],linewidth=0
            i+=1
        plt.legend()
        plt.axhline(y=0, color='k')
        plt.axvline(x=0, color='k')
        plt.show()
        pass
```

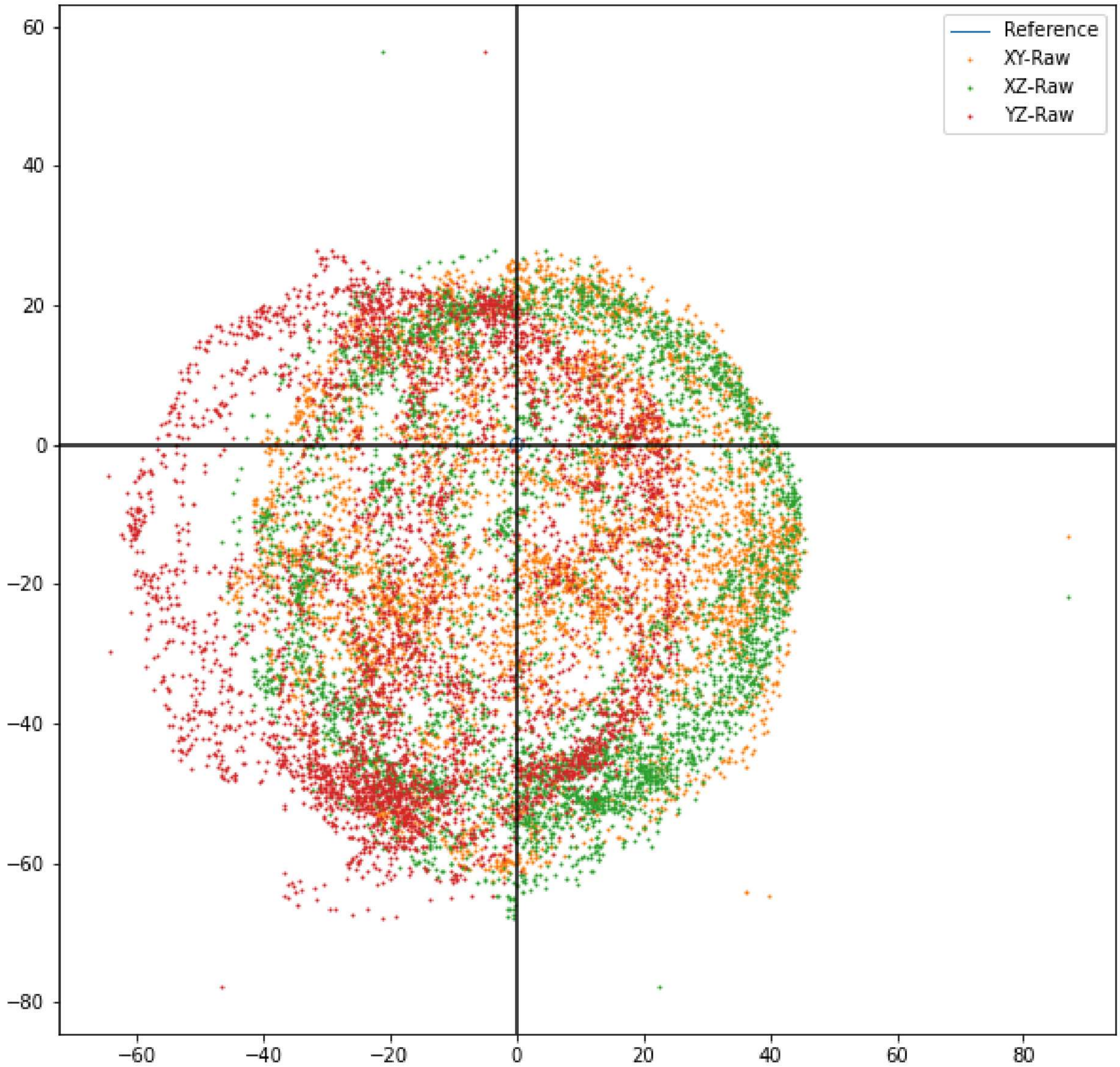
In [2]:

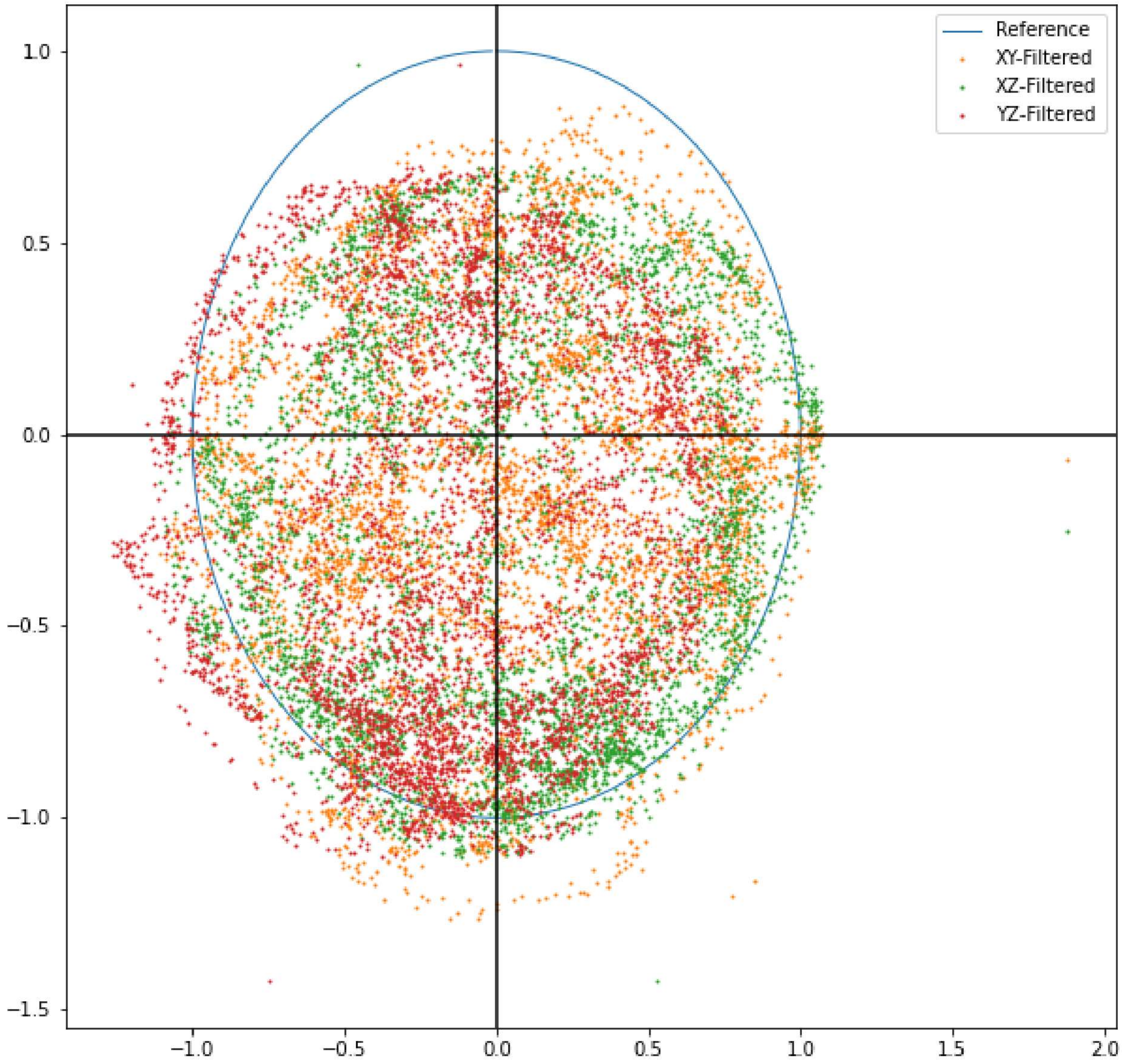
```
import numpy as np
import csv
def plot_data(dataset_file_name):
    data_set=[]
    with open(dataset_file_name, newline = '') as data_file:
        mag_data = csv.reader(data_file, delimiter='\t')
        for data_point in mag_data:
            data_set.append([float(i) for i in data_point])
        data_set=np.array(data_set)
    print("Data set length: {} points".format(data_set.shape[0]))


    vis=visualizer2D()
    vis.add(data_set[:,3],data_set[:,4],'XY-Raw')
    vis.add(data_set[:,3],data_set[:,5],'XZ-Raw')
    vis.add(data_set[:,4],data_set[:,5],'YZ-Raw')
    vis.show_2D()
    vis=visualizer2D()
    vis.add(data_set[:,0],data_set[:,1],'XY-Filtered')
    vis.add(data_set[:,0],data_set[:,2],'XZ-Filtered')
    vis.add(data_set[:,1],data_set[:,2],'YZ-Filtered')
    vis.show_2D()
```

```
plot_data('validation_data_step1en3.txt')
```

```
Data set length: 4578 points
```





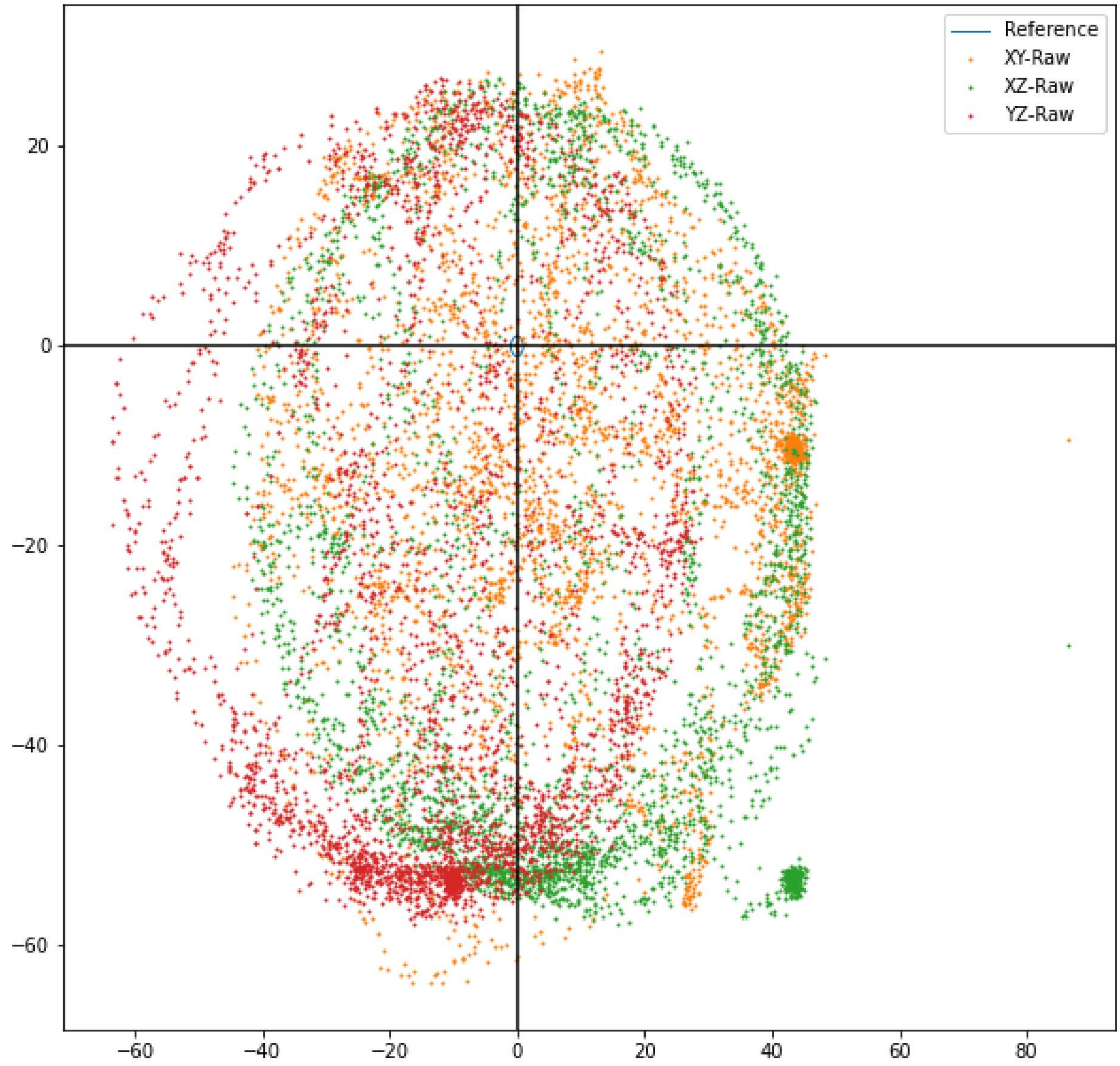
Tweaking learning rate by hand ..

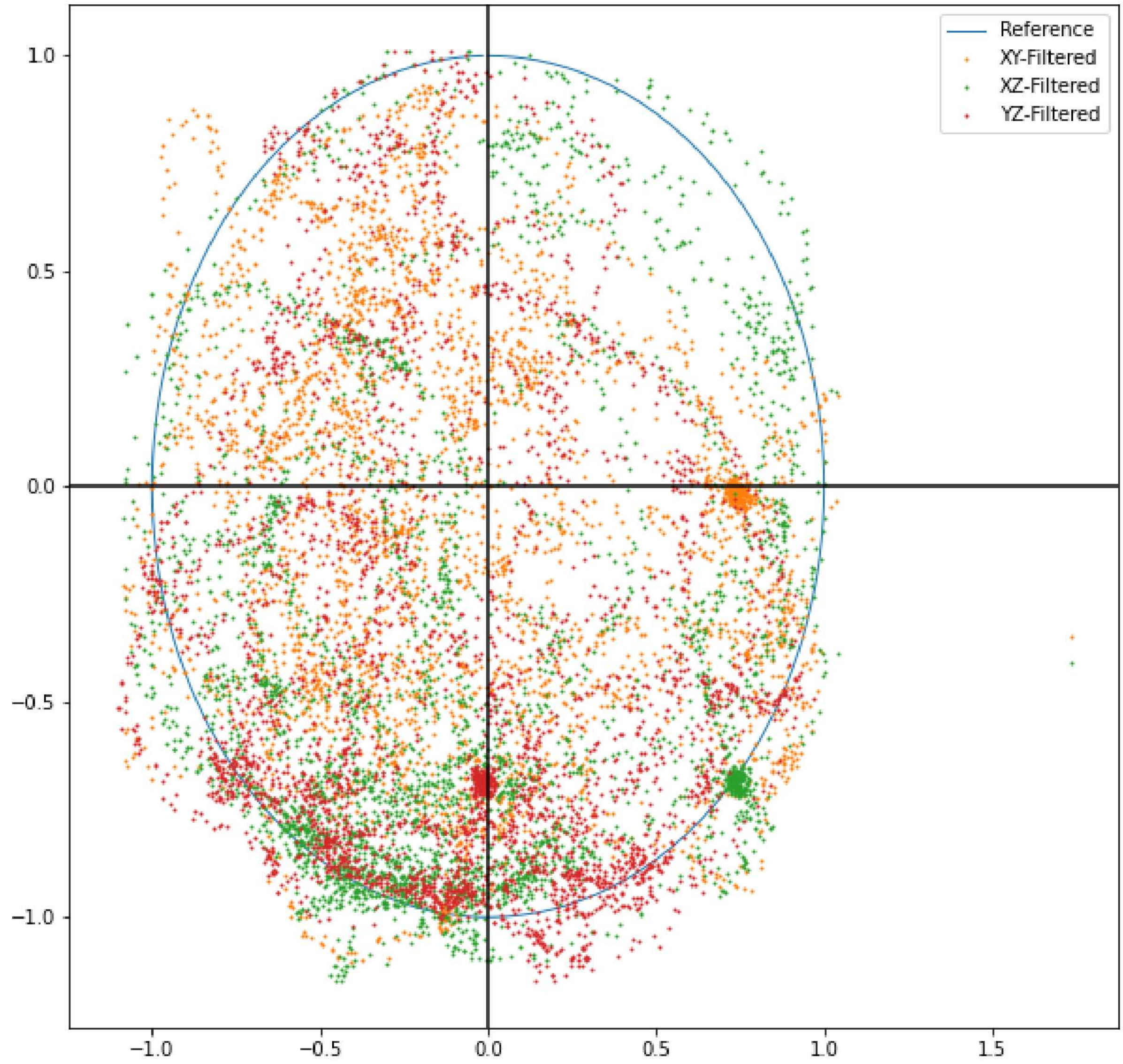
As seen above, the model does a good job of estimating the parameters. When running on hardware I tried three different settings for the step sice 1e-3, 1e-2, and 1e-1. The two later step sizes are plotted below.

In [3]:

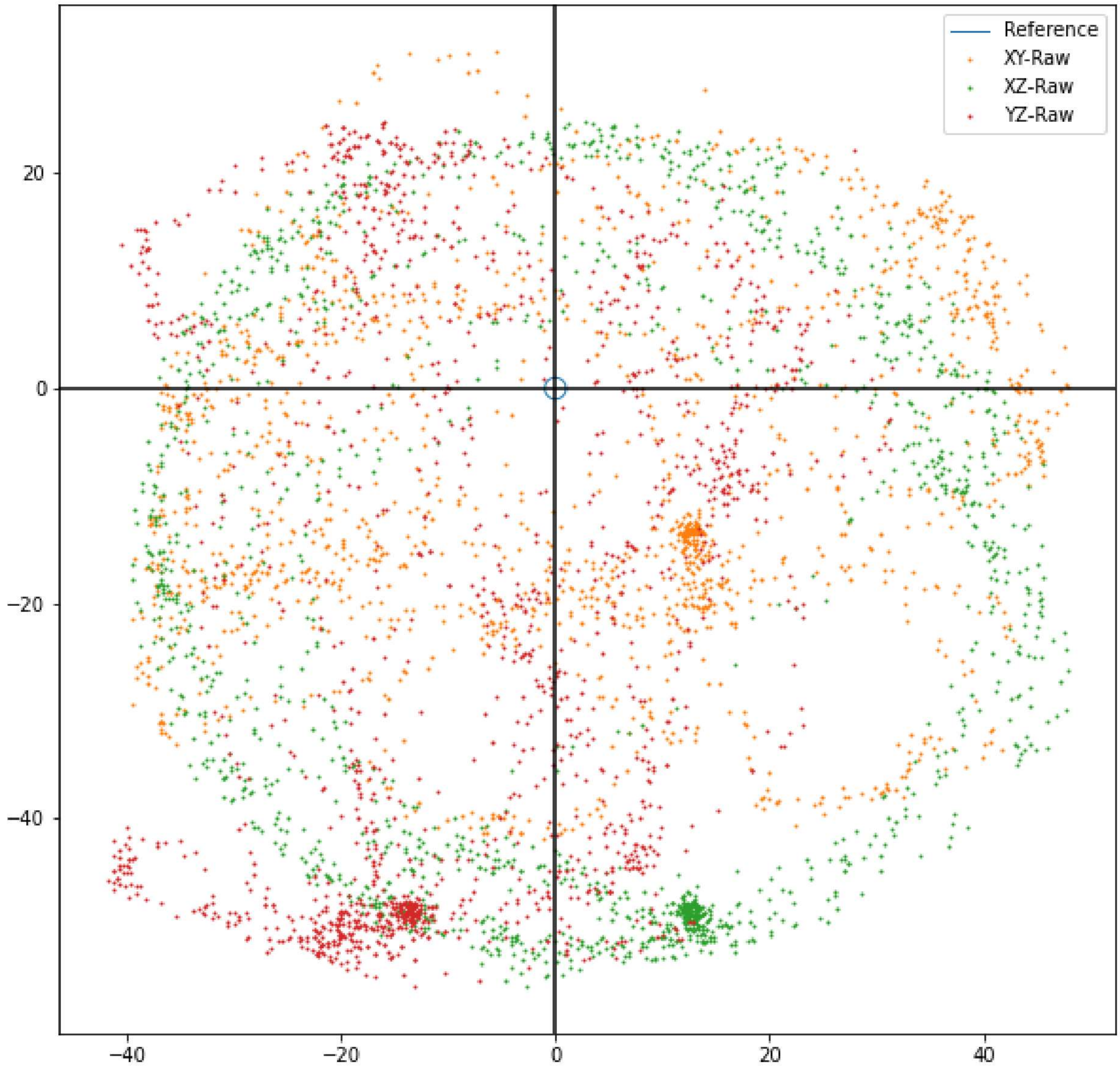
```
plot_data('validation_data_step1en2.txt')
plot_data('validation_data_step1en1.txt')
```

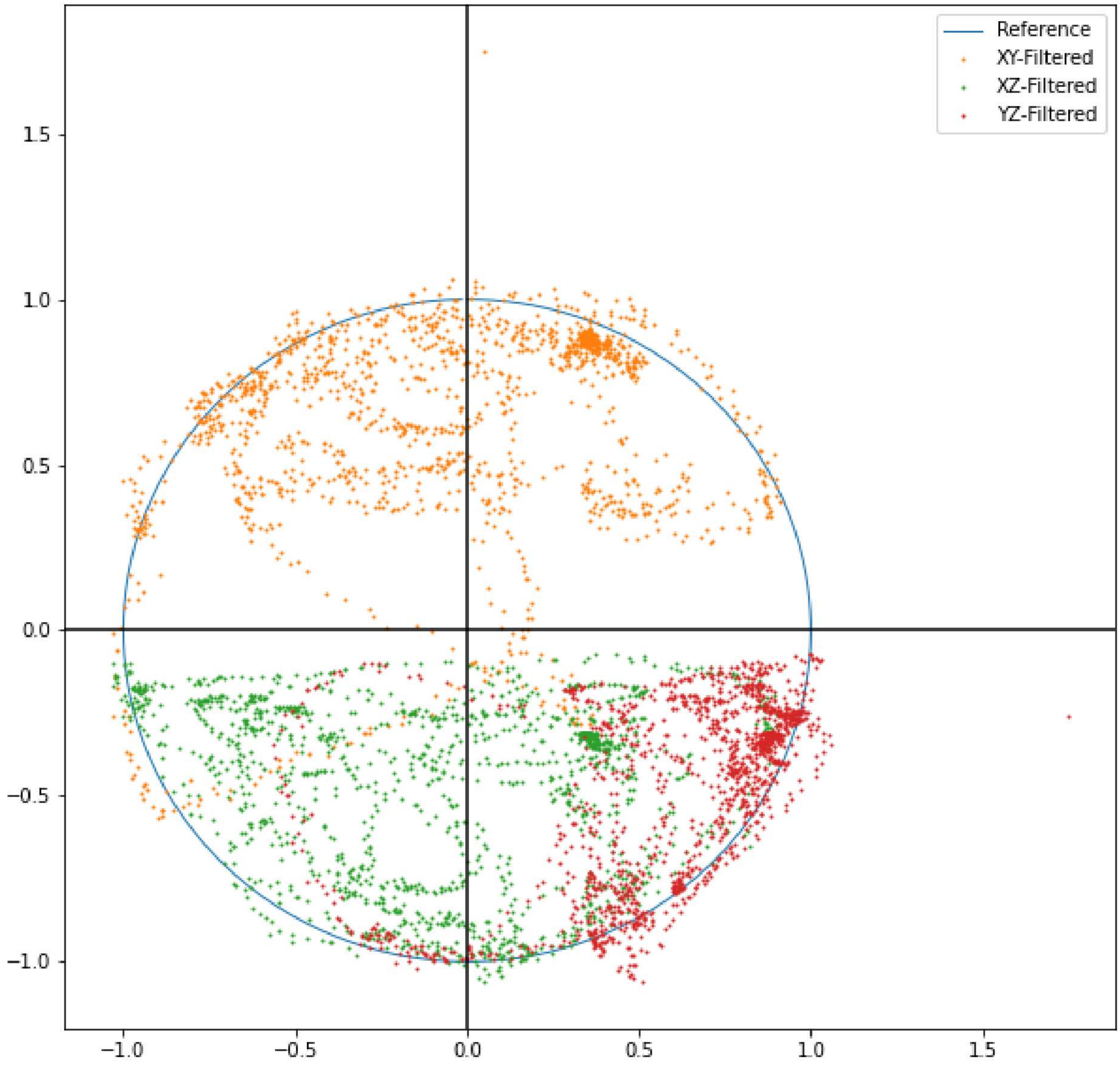
```
Data set length: 3459 points
```


Reference
XY-Raw
XZ-Raw
YZ-Raw
20
0
−20
−40
−60
−60
−40
−20
0
20
40
60
80




```
Data set length: 1505 points
```


Reference
XY-Raw
XZ-Raw
YZ-Raw
20
0
−20
−40
−40
−20
0
20
40



.. and a quantitative analysis

So as we have already implemented the model in python during development, we can now pull in the experimental data and split it up to use part of it for testin and another part for validation. We can then make a quantitative analysis of a series of different values for the stepsizes.

In [4]:
```
class Model():
    def __init__(self,eta=1e-1,a=np.ones((3,1)),b=np.zeros((3,1)), norm=0):
        self.a=a
        self.b=b
        self.eta=eta
        self.norm_x=norm

    def update(self,x):
        if self.norm_x==0:
            self.norm_x=np.sqrt(np.sum(np.square(x)))
            assert self.norm_x != 0, "Input data is zero vector"
        if x.shape==(3,):
            x=x.reshape(3,1)
        assert x.shape==(3,1), "x should be column vector, found shape {}".format(x.
        m=self.predict(x)
        norm_m=np.sqrt(np.sum(np.square(m)))
        assert norm_m!=0, "prediction is zero vector"
        x_normalized=x/self.norm_x
        self.a-=self.eta*2*(1-1/norm_m)*m*x_normalized
```

```
        self.b-=self.eta*2*(1-1/norm_m)*m

    def predict(self,x):
        if x.shape==(3,):
            x=x.reshape(3,1)
        assert x.shape==(3,1), "x should be column vector, found shape {}".format(x.
        x_normalized=x/self.norm_x
        m=self.a*x_normalized+self.b
        return m.reshape(3,1)

def train_model(eta, data, epochs=1):
    model=Model(eta,a=np.ones((3,1)),b=np.zeros((3,1)))
    for i in range(epochs):
        for data_point in data:
            model.update(data_point)
    return model

def eval_model(model, data):
    Y=np.array([])
    for data_point in data:
        pred=model.predict(data_point).T
        if Y.size==0:
            Y=pred
        else:
            Y=np.vstack((Y,pred))
    L=[]
    for prediction in Y:
        L.append(np.square(np.sqrt(np.sum(np.square(prediction)))-1))

    return np.average(np.array(L))
```

In [5]:

```
def load_data_sets(train_file, test_file):
    #Data format is tab-separated where first three columns are the filtered data an
    train_data=[]
    with open(train_file, newline = '') as data_file:
        mag_data = csv.reader(data_file, delimiter='\t')
        for data_point in mag_data:
            train_data.append([float(i) for i in data_point])
        train_data=np.array(train_data)
        train_data=train_data[:,3:6] #Use raw data
    print("Train data set length: {} points".format(train_data.shape[0]))

    test_data=[]
    with open(test_file, newline = '') as data_file:
        mag_data = csv.reader(data_file, delimiter='\t')
        for data_point in mag_data:
            test_data.append([float(i) for i in data_point])
        test_data=np.array(test_data)
        test_data=test_data[:,3:6] #Use raw data
    print("Test data set length: {} points".format(test_data.shape[0]))
    return (train_data,test_data)
```

In [6]:

```
def eval_eta(train_data, test_data):
    #Train models with different learning rate
    models={}
    for steps in range(1,200):
        eta=steps*0.0001
        models[eta]=train_model(eta,train_data)
    x,y=[],[]
    for eta in models.keys():
        x.append(eta)
        y.append(eval_model(models[eta],test_data))
```

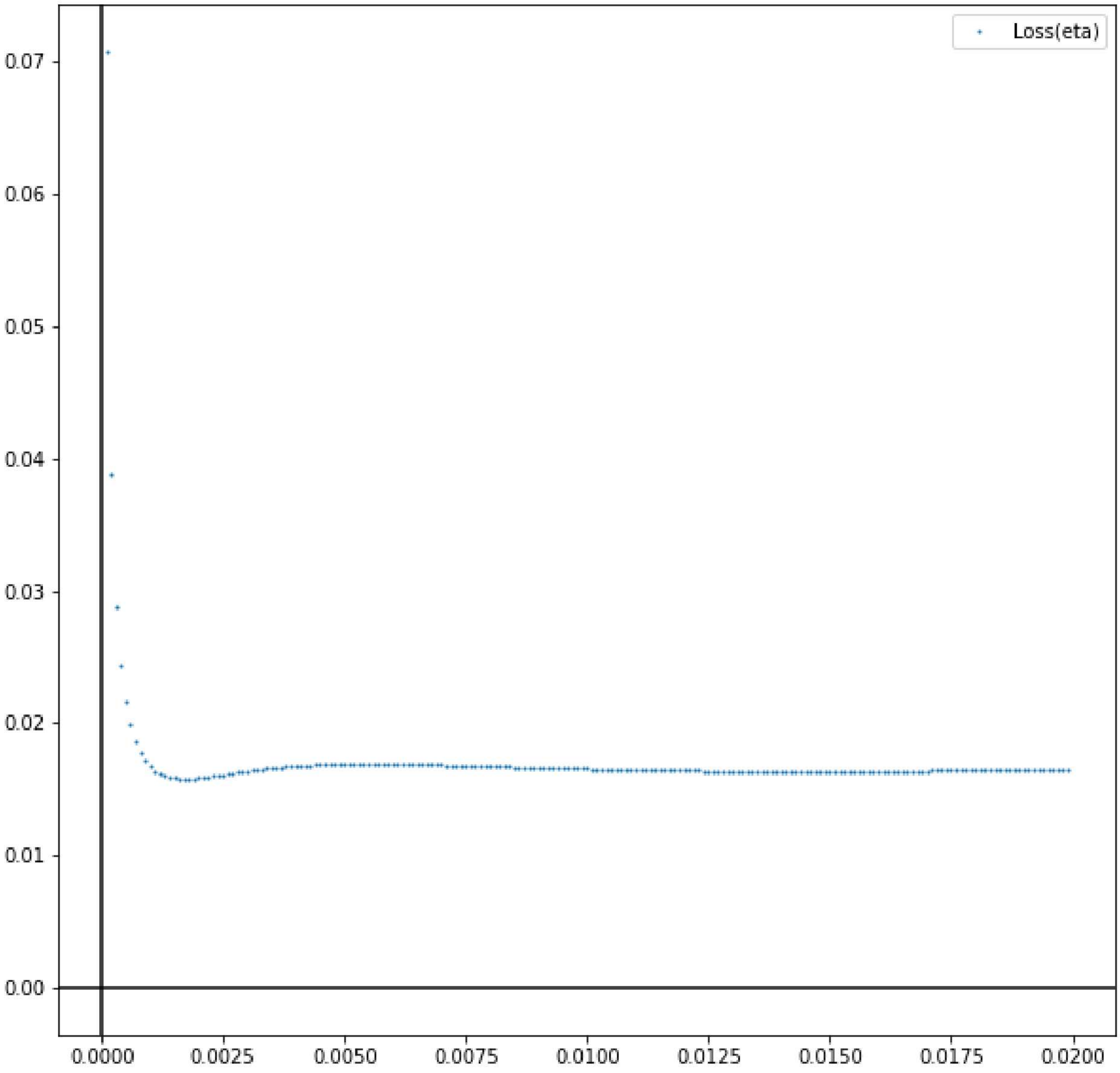
```
    x=np.array(x)
    y=np.array(y)

    vis=visualizer2D()
    vis.add(x,y,'Loss(eta)')
    vis.show_2D(draw_ref_circle=False)

    optimal_eta=x[np.argmin(y)]
    print("Minimum average loss of {:.5f} achieved at eta={:.3f}".format(np.amin(y),
    return optimal_eta

X_train, X_test=load_data_sets('validation_data_step1en3.txt', 'validation_data_step
optimal_eta = eval_eta(X_train, X_test)
```

```
Train data set length: 4578 points
Test data set length: 3459 points
```




```
Minimum average loss of 0.01580 achieved at eta=0.002
```

In [7]:

```
X_train, X_test = load_data_sets("validation_data_step1en3.txt", "validation_data_st
model = Model(eta=optimal_eta,a=np.ones((3,1)),b=np.zeros((3,1)))
epochs = 1
for i in range(epochs):
    for data_point in X_train:
        model.update(data_point)

Y=np.array([])
for data_point in X_test:
```

```
    pred=model.predict(data_point).T
    if Y.size==0:
        Y=pred
    else:
        Y=np.vstack((Y,pred))

vis=visualizer2D()
vis.add(Y[:,0],Y[:,1],'XY-Predictions')
vis.add(Y[:,0],Y[:,2],'XZ-Predictions')
vis.add(Y[:,1],Y[:,2],'YZ-Predictions')
vis.show_2D()
```

```
Train data set length: 4578 points
Test data set length: 3459 points
```