Simulation:

For the purpose of simulation we have created five python files

1. Measurement_model.py
2. Encryption.py
3. LocalHub.py
4. CentralNode.py
5. infofilter.py

Let's see the functionality of the files

1. **Measurement_model.py:**

   This file contains the code to generate the measurements assuming that the measurements have been made by four sensors using a process and measurement model following a linear constant velocity model.

   The function
   def get_sensor_measurements(state)

   ```
   """
   :param state: Each element of the list "states" is passed
   :return: motion_states, measurement states

   """
   ```

   Once the measurements are generated, respective measurements, measurement vectors and matrices are calculated , stored in a dictionary "sensors".

2. **Encryption.py:**

   In this piece of code, we take the measurement vectors from the dictionary **"sensors"** and encrypt the measurement vectors element wise using the paillier **"encrypt"** method which encodes and encrypts into a cipher text. The cipher text is stored in the form of an objects in the dictionary **" summationmap"**

3. **LocalHub.py**

   In a real life scenario we consider one of the four sensors to be a local hub and use this piece of code to aggregate the cipher measurement vectors. But for simulation we consider this simply to be another python file running on the same system.

   ```
   def getSummation(summationmap,pk):
     """
     Paillier homomorphic addition operation is performed
     :param summationmap: dic: Encrypted measurement vectors represented as
   objects
     :param pk: public key used for aggregation
   ```

```
    :return:resultmap: dic: Aggregated encrypted vectors represented as objects
    """
```

We use paillier homomorphic addition to aggregate all the respective ciphers of the sensors belonging to the same time instance using the public key given by the central node. The resultant aggregated cipher is stored as objects in the dictionary “resultmap”

4. **Centralnode.py**
   Central node in the real scenario has to be one raspberry pi which generates the paillier public key and respective private key, decrypts the data using this private key and runs the information filter. But for simulation, we simply consider this as another python instance.

   For key generation

```
"""Key generation using paillier library
  Args:
    n_length: key size in bits.
  Returns:
    tuple: The generated :class:`PaillierPublicKey` and
    :class:`PaillierPrivateKey`
"""
```

   Decryption:

```
def getDecryptedSummation(sk,resultmap,pk=None):
  """
  Performs decryption of the aggregated cipher texts

  :param sk: class: Private key which has been generated
  :param resultmap: dic: Aggregated cipher texts represented as objects
  :param pk: class: None
  :return: dic(finalresultmap): Decrypted measurement vectors
  """
```

5. **Infofilter.py**

Once we get the decrypted information, we pass this to the update function of the information filter. Initially, we initialize the state with a bad estimate.

```
class InfoFilter(object):

  def predict(self, x, P = None):
  """
  Predicts the state of the object given the state estimate of the previous state or an
initial estimate
  :param x: Initial state estimate/ Previous state estimate
  :param P: Initial error covariance matrix/ Updated error covariance matrix
  :return: Y : Measurement matrix
         y:  Measurement vector
         x:  Predicted state estimate
  """

  def update(self, Y,y,i,prediction = None):
  """
  Estimates the state of the object
  :param Y: Predicted measurement matrix
  :param y: Predicted measurement vector
  :param i: Aggregated and decrypted measurement vectors
  :param prediction: None
  :return:
  X: State estimate of the object
  P : Respective error covariance of the state estimate
  """
```

Instructions to run the code:

- To run the simulation code, run the file “infofilter.py” to see the estimates and the plot.
- To see the plot between information filter, ground truth or the error between the ground truth and filter, you can find two sections which belong to the plot at the end of the “infofilter.py” where you can comment off the section. By default, a plot between ground truth and the state estimates will appear.
- You can change the “ number_of_steps” in line no.5 from “Measurement_model.py” to set the iterations. By default “ number_of_steps” has been set to 100.