Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

Mini Project - Implementing State Estimation Filters

https://github.com/SasikaA073/AS-PCD-State-Estimation
https://sasikaa073.github.io/AS-PCD-State-Estimation

Y.W.S.P Amarasinghe 210035A

Submitted in partial fulfillment of the requirements for the module
EN 4594 Autonomous Systems

24th of December 2025

Contents

0.1 Introduction

My Final Year Project (FYP) is titled **4D Visual Grounding on Dynamic Point Clouds**. The inspiration for this state estimation mini-project, completed for the Autonomous Systems Module, is derived directly from my FYP research.

To provide context, **Visual Grounding** refers to the downstream task of localizing objects in an image based on natural language descriptions. In our FYP, the objective is to localize objects within a dynamic point cloud scene using language descriptions that contain both temporal and spatial cues.

Examples of such queries include: "The man sitting on the sofa wearing a shirt" or "The human performing an action from 10.00s to 14.00s."

Training a deep learning model to comprehend this context and align point clouds with linguistic descriptions requires a substantial volume of data. Consequently, as part of the FYP, we generated a synthetic dataset using Unity and Infinigen.

For this Autonomous Systems mini-project, I focused specifically on object tracking. Our synthetic dataset contains numerous scenes labeled with human actions, such as "walking" or "walking and turning." The objective of this project was to track a human subject using a Kalman Filter, operating under the assumption that human motion is linear with respect to time (constant velocity).

This report documents the implementation of a **Standard Kalman Filter (KF)** designed to identify and track the 3D trajectory of human subjects.

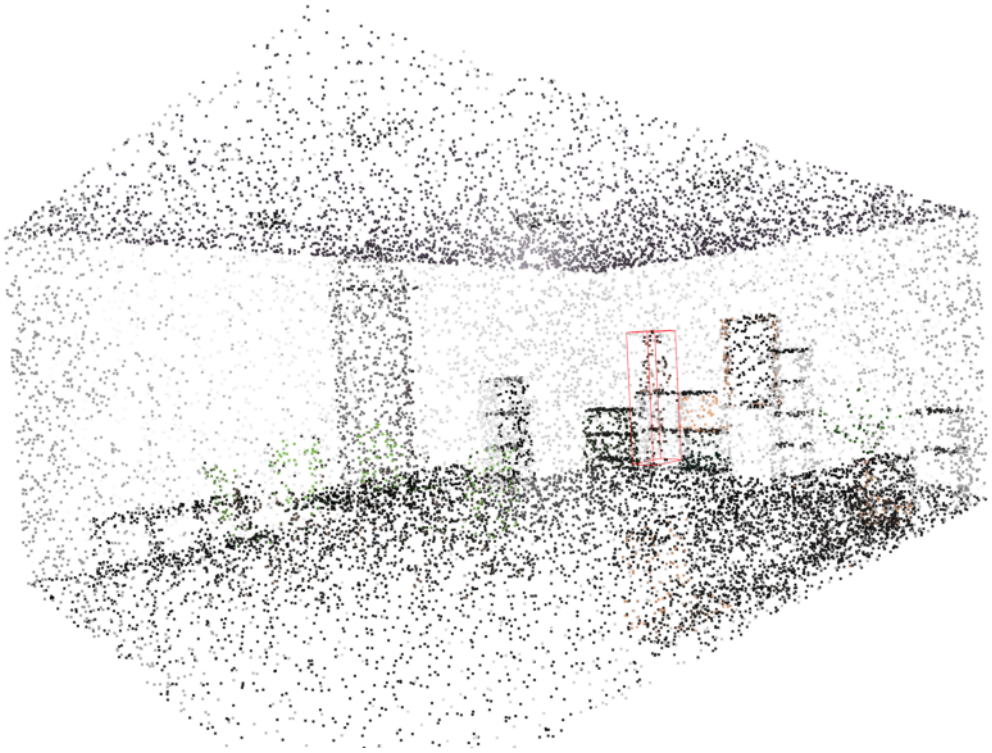

Figure 1: This is one frame from a scene from our synthetic point cloud dataset we created for our Final Year Project. The **red** bounding box shows the points corresponds to the human

0.2 Methodology

0.2.1 System Architecture

In the tracking pipeline, we select a scene from the dataset that depicts a human walking. The dataset already includes ground truth bounding boxes for the human subject in every frame.

All the frames from a scene are passed to the Sonata (Point Transformer v3) model.

In the context of the Kalman Filter, the Sonata model functions as the measurement step. It is responsible for extracting the human point cloud from the scene in each frame. I utilize the centroid of the extracted Human Point Cloud as the measurement vector (x_t, y_t, z_t).

For the prediction step of the Kalman Filter, I assume a Constant Velocity Model for the human's motion.

It is important to note that the Sonata model currently faces challenges in correctly segmenting the human from the scene point cloud. This remains an open problem in our core FYP work.

To properly evaluate the Kalman Filter implementation despite these upstream segmentation issues, I adopted a simulation approach. I added random noise to the ground truth human point cloud centroids to simulate valid but noisy detections, mimicking the output of a theoretically accurate Sonata model.

0.2.2 Mathematical Formulation

Assumptions

The standard Kalman Filter assumes:

1. **Linear Process Model**: The object moves according to linear laws of motion (Constant Velocity).
2. **Linear Measurement Model**: The sensor observes state variables directly or linearly.
3. **Gaussian Noise**: Both process and measurement noise are normally distributed.

State Vector

The state vector $\boldsymbol{x}$ consists of Position ($\boldsymbol{p}$) and Velocity ($\boldsymbol{v}$) in 3D space:

$$\boldsymbol{x} = \begin{bmatrix} p_x & p_y & p_z & v_x & v_y & v_z \end{bmatrix}^T$$

Process Model (Prediction)

We use a **Constant Velocity (CV)** model. The state at time t depends linearly on $t-1$:

$$\boldsymbol{x}_t = \boldsymbol{F}\boldsymbol{x}_{t-1} + \boldsymbol{w}_t$$

The State Transition Matrix $\boldsymbol{F}$ is:

$$\boldsymbol{F} = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Measurement Model (Update)

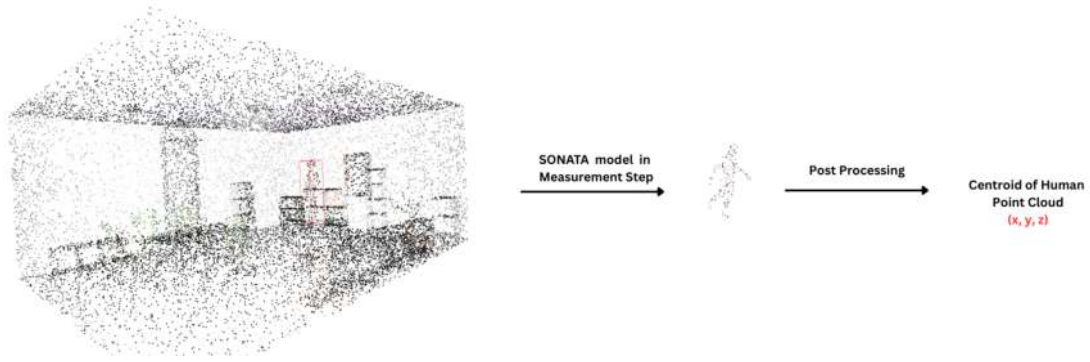

Figure 2: This figure shows how the scene point cloud is processed using the SONATA model to segment the human point cloud so that we can get the measurement for human position in the scene

The sensor measures position (p_x, p_y, p_z) directly. The Measurement Matrix $\boldsymbol{H}$ maps the 6D state to the 3D measurement:

$$\boldsymbol{z}_t = \boldsymbol{H}\boldsymbol{x}_t + \boldsymbol{v}_t$$

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

0.3 Implementation

The core logic is located in `src/tracking/kalman_filter.py`.

In my FYP, I use PyTorch and NumPy to manipulate point clouds. Consequently, I used Python for this Kalman Filter implementation to maintain consistency.

I referred to the following resources while implementing the Kalman Filter:

- `https://github.com/AtsushiSakai/PythonRobotics`
- `https://www.geeksforgeeks.org/python/kalman-filter-in-python/`

0.3.1 Code Snippets

Here we highlight the key components of the implementation.

Initialization

Listing 1: Kalman Filter Initialization

```
class KalmanFilter:
    def __init__(self, dt=1.0, process_noise_std=0.1, measurement_noise_std=0.1):
        self.dt = dt
        self.state_dim = 6
        self.meas_dim = 3

        # State Vector [x, y, z, vx, vy, vz]
        self.x = np.zeros((self.state_dim, 1))

        # State Covariance Matrix P
```

```
        self.P = np.eye(self.state_dim) * 100.0

        # State Transition Matrix F (Constant Velocity)
        self.F = np.array([
            [1, 0, 0, dt, 0, 0],
            [0, 1, 0, 0, dt, 0],
            [0, 0, 1, 0, 0, dt],
            [0, 0, 0, 1, 0, 0],
            [0, 0, 0, 0, 1, 0],
            [0, 0, 0, 0, 0, 1]
        ])
```

Prediction Step

Listing 2: Prediction Step

```
    def predict(self):
        """
        Predict the next state based on the process model.
        x = Fx
        P = FPF' + Q
        """
        self.x = self.F @ self.x
        self.P = self.F @ self.P @ self.F.T + self.Q
        return self.x
```

Update Step

Listing 3: Update Step

```
    def update(self, z):
        z = np.array(z).reshape((self.meas_dim, 1))

        # Innovation (residual)
        y = z - (self.H @ self.x)

        # Innovation Covariance
        S = self.H @ self.P @ self.H.T + self.R

        # Kalman Gain
        K = self.P @ self.H.T @ np.linalg.inv(S)

        # Update State
        self.x = self.x + (K @ y)

        # Update Covariance
        I = np.eye(self.state_dim)
        self.P = (I - (K @ self.H)) @ self.P

        return self.x
```

0.3.2 Filter Configuration

- **State Dimension**: 6
- **Measurement Dimension**: 3
- **Covariance Initialization ($\boldsymbol{P}$)**: Identity matrix scaled by 100.0, representing high initial uncertainty.

0.3.3 Noise Matrices

1. **Process Noise ($\boldsymbol{Q}$)**: Models the uncertainty in the constant velocity assumption. It is approximated as a diagonal matrix scaled by `process_noise_std`.

$$\boldsymbol{Q} = \boldsymbol{I} \times \sigma_{proc}^2$$

2. **Measurement Noise ($\boldsymbol{R}$)**: Models the sensor accuracy.

$$\boldsymbol{R} = \boldsymbol{I} \times \sigma_{meas}^2$$

A higher measurement noise (R) was selected compared to process noise (Q) because the simulated detection noise ($\sigma = 0.5m$) is significant, whereas the constant velocity assumption holds relatively well for walking actions.

In the simulation script `run_tracking.py`, we use:

- `process_noise_std = 0.05`
- `measurement_noise_std = 0.5`
- Time step `dt = 1.0`

0.3.4 Simulation Loop

The `run_tracking.py` script performs the following steps for each frame:

1. **Noise Injection**: Adds Gaussian noise ($\mu = 0, \sigma = 0.2m$) to the Ground Truth center. Ideally Sonata model should be used to get the segment of human.
2. **Prediction**: `kf.predict()` advances the state using $\boldsymbol{F}$.
3. **Correction**: `kf.update(measurement)` fuses the noisy measurement using the optimal Kalman Gain $\boldsymbol{K}$.

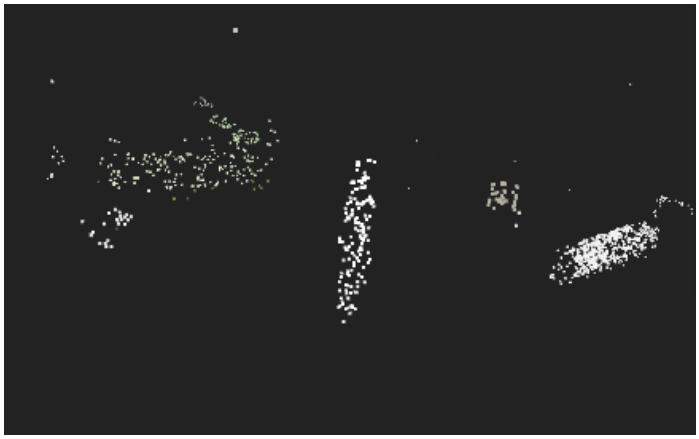

Figure 3: This figure illustrates the segmentation output for **human** obtained when processing a single frame using our custom fine-tuned Sonata model. In the context of our Final Year Project, the fine-tuning process is ongoing; consequently, the model does not yet accurately segment the human subject, as shown in the image above. To address this limitation for the current tracking implementation, I added Gaussian noise to the ground truth bounding boxes. This approach simulates the predictions expected from an ideally functioning Sonata model.

0.4 Results

The filter's performance is quantified by the Root Mean Square Error (RMSE) of the position:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{\boldsymbol{p}}_i - \boldsymbol{p}_{true,i})^2}$$

Here are some of the human tracking for the scenes from our dataset using Kalman Filter.

0.4.1 Typical Performance Observation

- **Measurement RMSE**: Reflects the raw sensor noise.
- **KF Estimate RMSE**: Reflects the filtered position.

You can check **https://sasikaa073.github.io/AS-PCD-State-Estimation** for more results of the scenes from our dataset.

Example 1

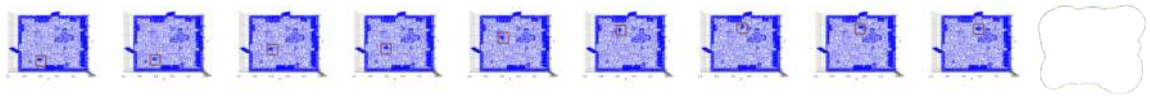

Figure 4: The point cloud sequence visualizing every 10th frame.Red box shows the human.

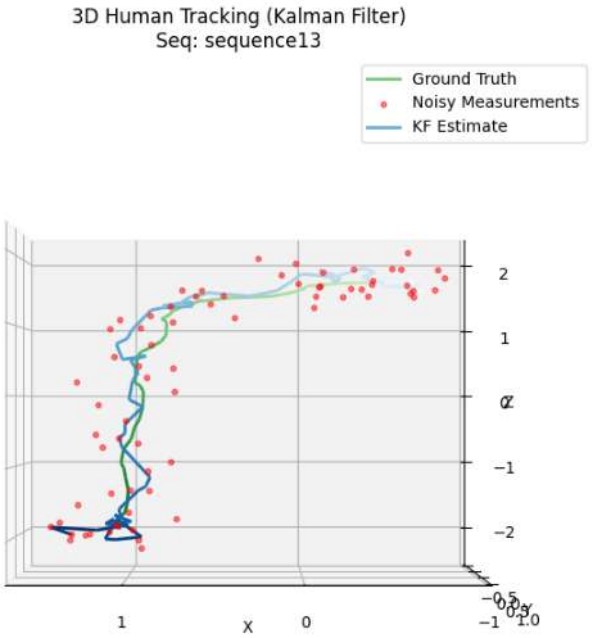

Figure 5: This graph compares the Ground Truth path (green gradient) and the Kalman Filter Estimated path (blue gradient). Darker shades indicate the start of the sequence, fading to lighter shades at the end. Red dots represent the measured positions of the human (noisy Sonata model output).

```
--- Evaluation Results ---
Measurement RMSE: 0.1997 m
KF Estimate RMSE: 0.1195 m
Improvement:      0.0802 m
```

Example 2

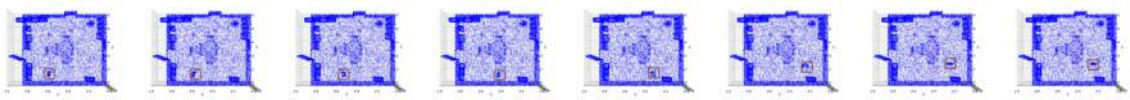

Figure 6: The point cloud sequence visualizing every 10th frame.Red box shows the human.

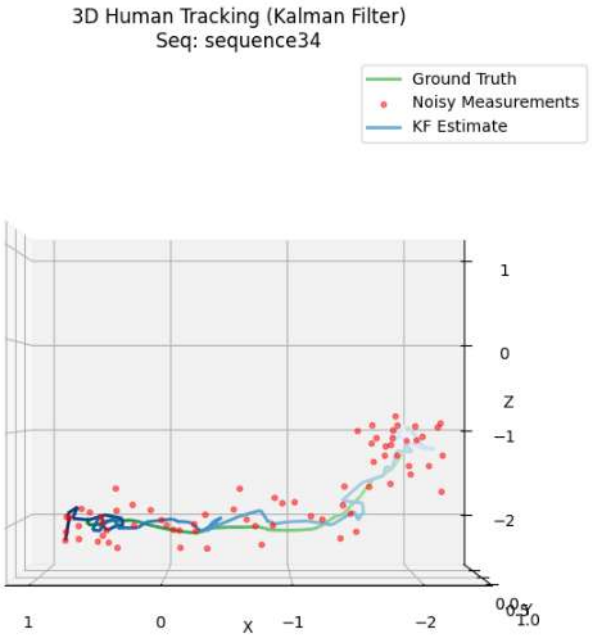

Figure 7: This graph compares the Ground Truth path (green gradient) and the Kalman Filter Estimated path (blue gradient). Darker shades indicate the start of the sequence, fading to lighter shades at the end. Red dots represent the measured positions of the human (noisy Sonata model output).

```
--- Evaluation Results ---
Measurement RMSE: 0.1917 m
KF Estimate RMSE: 0.1058 m
Improvement:      0.0859 m
```

0.5 Discussion

The Standard Kalman Filter consistently produces a lower RMSE than the raw measurements, successfully smoothing the trajectory. By leveraging the motion model (physics), it filters out high-frequency noise, resulting in a cleaner 3D path suitable for autonomous navigation or analysis.

A Standard Kalman Filter was selected over non-linear variants, such as the Extended Kalman Filter (EKF), due to the inherent linearity of the system. The state transition is based on a Constant Velocity model, and the measurement model involves the direct observation of the Cartesian centroid (x, y, z), both of which are linear operations1111. Consequently, the linearization steps (Jacobian calculations) required by an EKF are unnecessary, and the Standard KF remains the optimal estimator for this application under the assumption of Gaussian noise.

0.6 Conclusion

I implemented a Standard Kalman Filter for 3D human tracking in point cloud sequences. The filter effectively reduces measurement noise and provides smooth, consistent trajectory estimates. The implementation demonstrates the practical application of state estimation techniques in autonomous systems, particularly for object tracking in dynamic environments.

Bibliography

[1] X. Wu et al., "Sonata: Self-Supervised Learning of Reliable Point Representations," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2025.

[2] A. Joshi et al., "Procedural Generation of Articulated Simulation-Ready Assets," arXiv preprint arXiv:2505.10755, 2025.

[3] A. Sakai, "PythonRobotics," GitHub repository. [Online]. Available: https://github.com/AtsushiSakai/PythonRobotics. [Accessed: Dec. 24, 2025].

[4] GeeksforGeeks, "Kalman Filter in Python." [Online]. Available: https://www.geeksforgeeks.org/python/kalman-filter-in-python/. [Accessed: Dec. 24, 2025].

Appendix A

Appendices

A.1 AI Usage

In accordance with the assignment guidelines, Artificial Intelligence tools were utilized to assist in the completion of this project.

```
# For research : Perplexity
Can you find me the python implementation of Kalman Filter?
```

```
# Fix Grammar : Google Gemini Pro
I want to check grammar of these passages. I want to write this as clear as
    possible wihtout any ambiguity. And give the output in Latex syntax """


\section{Introduction}

My Final Year Project is \textbf{4D Visual Grounding on Dynamic Point Clouds}. The
     state estimation mini project of Autonomous Systems Module inspiration comes
    from my FYP project.

To give a brief understanding about \textbf{Visual Grounding},  it means a
    downstream task of referring to the objects in an image using language
    descriptions. In our final year project, our goal is to refer to objects in a
    point cloud scene with language descriptions with temporal and spatial cues.

For example: ``What is the color of the shirt the man sitting on the sofa is
    wearing?'' or ``What is the human doing from 10.00s to 14.00s?''

To teach a deep learning model to understand this context and the alignment
    between point clouds and language descriptions, we need a lot of data. As part
     of our Final Year Project, we created synthetic data using Unity and
    Infinigen.

For the Autonomous Systems mini project, I considered object tracking. In our
    synthetic dataset, there are lots of scenes with the human action labels "
    walking", "walking and turning". I wanted to track the human using a Kalman
    Filter, assuming that the human motion is linear with time.
```

```
This report documents the implementation of a \textbf{Standard Kalman Filter (KF)}
    for identifying and tracking the 3D trajectory of human subjects.

\section{Methodology}

\subsection{System Architecture}

In the tracking pipeline, we choose a scene that corresponds to the human walking.
    In the dataset, we had already created bounding box for the human for each
   frame in the sequence.
 This Sonata model acts as the measurement step of in the Kalman Filter. Sonata
   model will extract the human point cloud, for each frame. I use the centroid
   of the Human Point Cloud as the measurement (x_t,y_t,z_t).

For the prediction step, I assume human is moving with a constant velocity.

However since the Sonata model is still incapable of correctly identifying the
   human from the scene point cloud. This is still a challenge in our final year
   project, yet to address. I have got the results of using the sonata model in
   the measurement step in the Kalman Filter. Those results are in the Appendix.

Since those results were not great, I used the following approach.  I added random
    noise to the ground truth human point clouds to simulate the accurate
   detections from the Sonata Model. """
```

I used the same prompt given above to fix grammar and flow of the report content.

```
# For Visualization - VS Code
I want to create a subfolder with the gif file name and save all the frames that
   were contributed when creating the GIF"
```

```
# For Visualization - VS Code
I want to create python script in images folder that will center crop all the
   images in the give folder path by the user (eg : /Users/sasika/my_projects/
   Autonomous Project/Images/11-sequence) and horizontally stack all the images
   according to the alphabetical order of their name"
```