Hands Tracker - Hand Position Tracking with Sensor Fusion and Occlusion Optimization

Tobias Wursthorn, 1629762
Physical Computing, Digital Reality, Faculty of Design, Media and Information, Hamburg University of Applied Sciences
01 March 2023

***Abstract*—This paper presents a novel approach to optimize projection with a video projector on the hands. It combines video-based hand tracking and the use of inertial measurement units (IMUs) with a Kalman filter to achieve accurate and robust hand tracking with low-cost webcams. The proposed method is tested on a real-time system and the results show that the proposed method outperforms existing cam-based method in terms of accuracy and robustness. The results also show that the proposed method can be used to improve the accuracy of projection with a video projector on hands. Finally, the paper discusses the potential applications of this approach.**

***Index Terms*—Kalman Filter, IMU, Hand Tracking, Touchdesigner**

I. Introduction

Projection onto hands has emerged as a promising modality for creating interactive and immersive experiences. However, projecting onto moving and dynamic surfaces such as hands presents several challenges, including latency and dropouts in hand tracking using media pipe and cam-based systems. These challenges can significantly affect the quality of the user experience and limit the potential applications of hand projection technology.

To address these challenges, this paper proposes a method for optimizing the projection of video content onto users' hands by combining video-based hand tracking with the use of smartphone inertial measurement units (IMUs) and a Kalman filter. The idea is to feed both hand tracking and IMU measurement signals into a Kalman filter to provide robust and reliable hand tracking even with some interruptions in the tracking itself and the difference in latency between the two measurement methods.

To design the Kalman filter itself and to evaluate the effectiveness of the proposed method, the author first implemented a simulation model to test the performance of the Kalman filter under different conditions. Then, the author implemented the proposed method in a real-time system using TouchDesigner to work with real-time data. The experimental results showed that the proposed method significantly improves the accuracy and reliability of hand tracking, enabling seamless interaction with projected content.

Overall, the proposed method has potential for various applications, including interactive games and augmented reality. The rest of the paper is organized as follows. Section 2 provides an overview of related work in hand tracking and projection on dynamic surfaces. Section 3 describes in detail the simulation and model building for the proposed method, including the hand tracking system, the IMU sensors, and the Kalman filter. Section 4 presents the implementation and experimental setup and results, while Section 5 shows the results of this approach. Finally, Section 6 concludes the paper and discusses future research directions.

A. Problem Definition

The problem addressed here is therefore summarized in the following question: *Is it possible to improve the tracking by combining the acceleration data from the IMU of a mobile phone and the position data of a hand from a camera-based tracking in a way that compensates for the latency of the camera tracking and the temporary loss of measurement data?*

II. Related Work

Since the projection on the basis of existing position data is a rather standard media technical task, the search for related work was limited to the generation of these position data. The first paper to be mentioned here [1] presents an architecture that combines a monocular camera and an inertial measurement unit (IMU) to track the localization of mobile devices in unknown environments. While the IMU provides high-frequency acceleration and angular velocity, its motion tracking is prone to collapse due to drift integration. Conversely, vision-based motion tracking provides higher accuracy but struggles in weakly textured or dynamic scenes. To address these issues, the paper proposes a loosely coupled method that fuses the IMU and monocular camera data using an error-state Extended Kalman Filter framework. This method achieves real-time self-motion estimation in resource-constrained mobile devices by exploiting the advantages of both sensors. The second paper [2] discusses the limitations of motion tracking devices with limited range, such as the Kinect. To avoid being trapped within the limited range, the paper proposes the use of an inertial measurement unit (IMU) to capture hand posture and position, which does not have a range limitation. However, detecting motion with an IMU requires double integration of acceleration, which can lead to divergent motion due to sensor noise. Therefore, this paper proposes a method to estimate hand position from arm posture and perform hand capture that overcomes these limitations.

III. Model Building and Simulation

In order to model the problem described here, the most important key parameters of the physical setup were first determined. Then, a Constant Acceleration Kalman filter model was implemented and simulated with the physical characteristics of the real setup.

A. *Key data from the targeted real setup*

As can be seen in Figure (cf. 1), the goal of the experiment is to project a dot onto a hand moving in the projection field. The hand in the projection field itself is detected by the webcam already described. The author deals with the detection using MediaPipe in chapter (cf. IV-B). As a second data source, the acceleration data of a smartphone is used, which is transferred to the Kalman filter via network. The author discusses the details of the acceleration data in chapter (cf. IV-C).

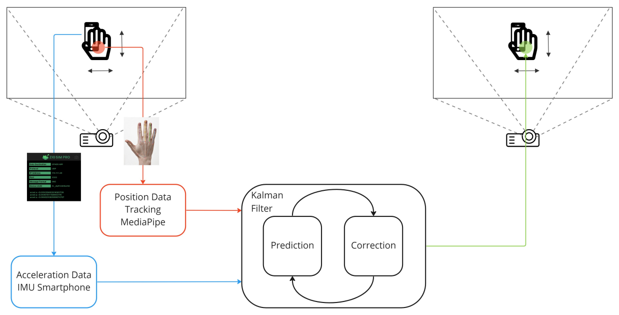


Figure 1. Signal Flow Real Setup

Thus, position and acceleration data are available to the Kalman filter. The first special feature of the physics setup to be simulated is the determination of the latency between the ground truth of the actual position data and the position data in real-time operation from the MediaPipe (cf. IV-B) tracking. Since no additional sensory equipment like an optical IR tracking system with markers was used to determine the ground truth for this setup, the camera stream was recorded in parallel for the ground truth calculation in order to be able to calculate it offline with increased quality settings. Thus, the data streams could be evaluated in Matlab, see figure (cf. 2), by starting the online and offline detection simultaneously and determining an averaged latency of 3 frames at 30fps with a FindDelay function.

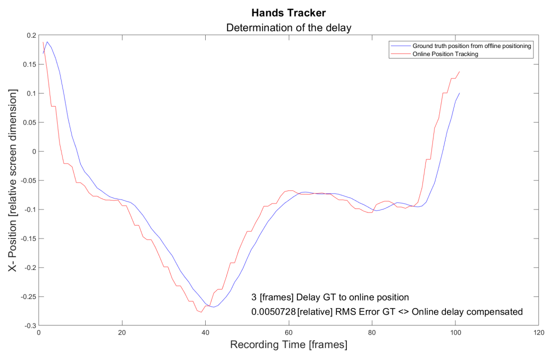


Figure 2. Find Delay Ground Truth

Another relevant latency for the system and the processing of the simulation data is the latency between the position data that can be processed in the real-time system and the acceleration data. These were recorded using a simple test setup with the implemented camera-based tracking and the incoming acceleration data with multiple repetitions and then visualized in Matlab (cf. 3). The rising edges of a vertical motion are well visible with an averaged latency of 4 frames at 30fps.

In addition, the simulation must take into account the fact that, for technical reasons (slower calculation of the MediaPipe Tracking), the position data can be updated less frequently than the incoming acceleration data. In the tests carried out, a factor of about 3 was determined, which means that the Kalman Filter only receives new position data every three incoming acceleration data.

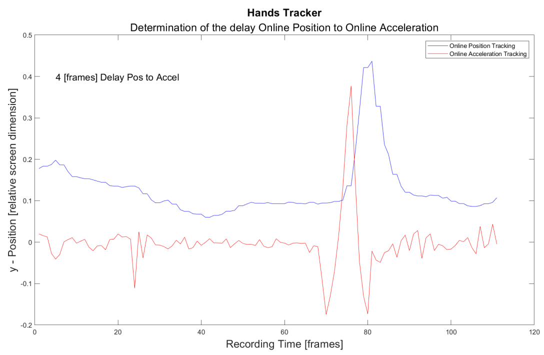


Figure 3. Find Delay Tracking to IMU

B. *Mathematical Description and Defining Kalman Filter*

Four parameters are now available as input data, X and Y of the position and the corresponding acceleration data. The position data are defined in a value range from -0.5 to 0.5 as relative values related to the size of the screen in X and Y direction, where x = 0 and y = 0 represents the center of the screen. Consequently, our state is Vector $\vec{x}$:

$$x_k = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix}$$

This results in the following for the movement update:

$$x_{k+1} = A \cdot x_k + B \cdot u$$

And because there is no control input u in our case this will result to the following Motion Matrix:

$$x_{k+1} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \frac{1}{2}\Delta t^2 \\ 0 & 0 & 1 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \end{bmatrix}_k$$

For the measurement matrix we have only entries for the position and the acceleration data. Consequently, the result for the measurement matrix is H:

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot x$$

After the further matrices for the process noise Q, the measurement noise R, the error covariance P and the Kalman gains K are defined (for complete reproducibility, please refer to the authors github repository [4] for python script to generate the simulation data), all necessary components for the Kalman filter are available for its prediction and correction. See [5] for an incredible resource on setting up Kalman filters in Python.

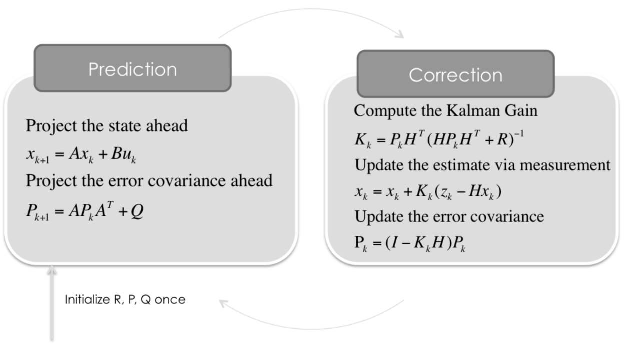

Prediction
Project the state ahead
$x_{k+1} = Ax_k + Bu_k$
Project the error covariance ahead
$P_{k+1} = AP_kA^T + Q$
Correction
Compute the Kalman Gain
$K_k = P_kH^T(HP_kH^T + R)^{-1}$
Update the estimate via measurement
$x_k = x_k + K_k(z_k - Hx_k)$
Update the error covariance
$P_k = (I - K_kH)P_k$
Initialize R, P, Q once

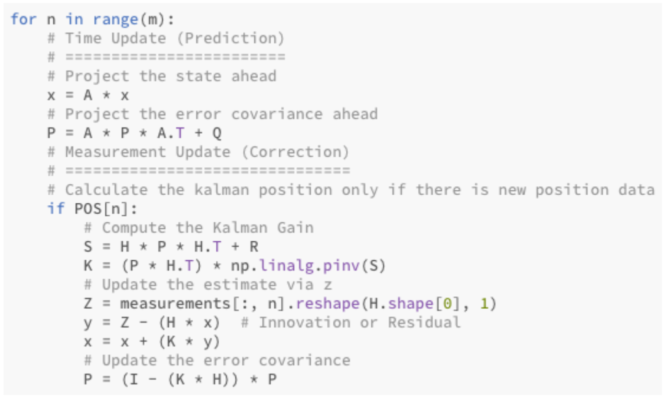
Figure 4. Kalman Filter Loop

As an additional factor it is to be mentioned that as already mentioned in (cf. III-A) not for every incoming value of acceleration data also new position data are available. For the simulation here the factor 10 was decided. To take this factor into account, the Kalman update loop looks as follows.

With respect to the implementation in the Touchdesigner real-time environment (cf. IV-D), the Kalman filter was implemented using only the Numpy Python library and not specialized libraries such as filterPy.

C. Simulation

For the simulation, it is first necessary to convert the generated time-dependent position data into acceleration data



```
for n in range(m):
    # Time Update (Prediction)
    # ========================
    # Project the state ahead
    x = A * x
    # Project the error covariance ahead
    P = A * P * A.T + Q
    # Measurement Update (Correction)
    # ===============================
    # Calculate the kalman position only if there is new position data
    if POS[n]:
        # Compute the Kalman Gain
        S = H * P * H.T + R
        K = (P * H.T) * np.linalg.pinv(S)
        # Update the estimate via z
        Z = measurements[:, n].reshape(H.shape[0], 1)
        y = Z - (H * x)  # Innovation or Residual
        x = x + (K * y)
        # Update the error covariance
        P = (I - (K * H)) * P
```

Figure 5. Kalman Filter Update Loop in Python

by means of second-order differential calculation. For this purpose, the position data was first simulated in a horizontal sinusoidal motion and then slightly upward in the vertical direction. The unprocessed position data then served as the basis for the generation of the acceleration data, while the position data itself was processed in such a way that only every tenth value underwent a position change and the position data itself was shifted according to the delays determined (cf. III-A).

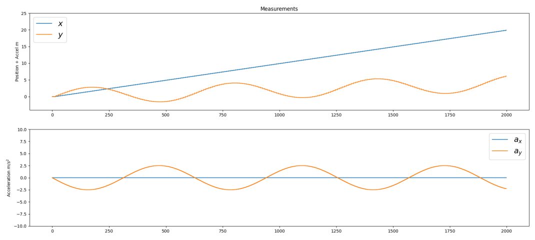


Figure 6. Simulation Position and Acceleration

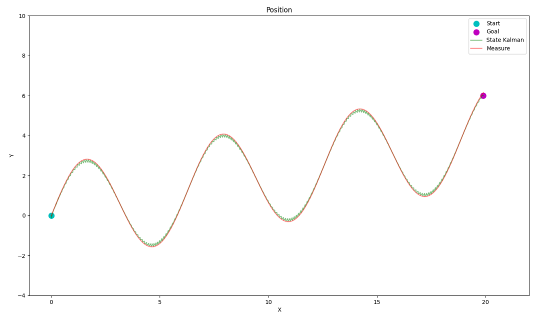


Figure 7. Simulation XY Position

In figure (see 6) are the raw data of position x and y and the acceleration $\ddot{x}$ and $\ddot{y}$ while figure (cf. 7) shows the position trajectory on a virtual screen. To prepare for the real setup, we experimented with more noisy data for the position data to check if the model with adjusted values for process noise Q and measurement noise R is still able to make useful predictions for the position. See figure (cf. 8)

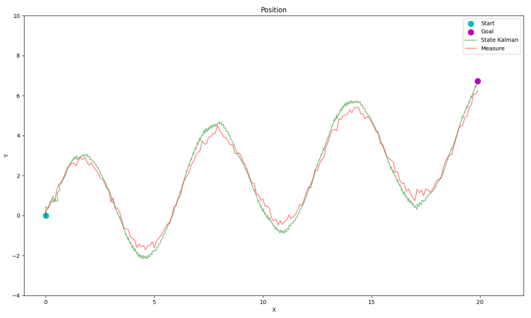


Figure 8. Simulation XY Position

IV. Implementation

In the following, the implementation of the already described project and the setup (cf. 1) by means of the Kalman filter in the real-time system with the generation and accumulation of the sensory data as well as the processing by the filter itself and the logging of all relevant parameters for later evaluation is explained.

A. Technical Setup

The technical setup for the generation of experimental data, which can be used to determine whether the overall question (cf. I-A) can be successfully answered by the generated system, is composed as follows.

- Video projector
- Webcam
  - Evaluation of the webcam livestream with MediaPipe to determine the XY position of the hand
- Smartphone with IMU sensor
  - Transmission of XZ acceleration data via OSC (upright position of the smartphone therefore not XY) using ZigSim Pro App
- Notebook with Touchdesigner

It should be noted that the setup only works when the smartphone is held upright. Compensation for a different position via a position sensor is currently not implemented.

To check the current function of the Kalman filter, two points were projected onto the hand. The green point is the current output of the Kalman filter, while the red point represents the pure position data of the hand tracking alone (i.e. the current path when such setups are realized with camera-based hand tracking). To get a correct input for the position data in the setup with the video projector, the position data must first be mapped to the projection surface, because the camera image never exactly matches the projection image. The second way to check the whole pipeline is to visualize the output directly in the webcam image itself, in this case the mapping is omitted.

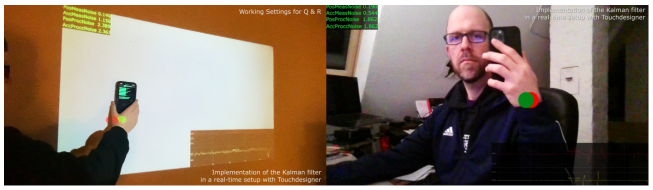

Working Settings for Q & R
Implementation of the Kalman filter in a real-time setup with Touchdesigner
Implementation of the Kalman filter in a real-time setup with Touchdesigner

Figure 9. Physical Setup

B. Hand Tracking

For hand tracking, a Python implementation was first created based on templates from MediaPipe [3], a machine learning framework from Google, in PyCharm. MediaPipe allows the detection of 21 landmarks per hand. Each landmark is described by relative XYZ values. The offline implementation was then implemented in the real-time application Touchdesigner (see next chapter) within the Script Operator. In this way, any video or video file source available to the system could be used as input for hand recognition, and the desired usable data, namely the XY position of the metacarpal, could be made directly available as a variable within Touchdesigner for subsequent processes. Hand detection and processing could be implemented in the Touchdesigner ecosystem at about 11-13fps on the hardware used.

C. IMU Acceleration Data

In order to obtain acceleration data in an easily processable form, it was decided to use a smartphone as the IMU source. The exposed data can be read and transmitted via network (json, osc) by multiple apps, the author used ZigSim Pro by 1-10, Inc.. ZigSim Pro is able to transfer the raw data from the different sensors of a smartphone, in this case an IPhone 12 Pro, via OSC to the processing instance. The refresh rate was set to 60fps. The transfer was done directly from the smartphone to the processing notebook without using a router (hotspot mode) to have only one wifi connection in the signal chain and to keep latency and interference as low as possible.

D. Realtime Application

Since the goal of the entire research project was to enable real-time implementation, an environment was needed that could handle video signals interactively, as well as one that would allow the execution of scripts (Python) with external libraries. In this case the choice fell on Touchdesigner. On the one hand the interactive possibilities to handle video and control signals are quite extensive, on the other hand the necessary Python abilities are available. Furthermore, there is a free non-commercial version of the program available, whose license covers the use for research and free art purposes.

E. Implementation of the Kalman Filter in Touchdesigner

The entire application was implemented in several modules. First of all there is the MediaPipe module (cf. IV-B) and the incoming OSC data from the IMU. On the output side, generative texture primitives (TOPs in Touchdesigner) and the signal operators (CHOPs) are used to display the image

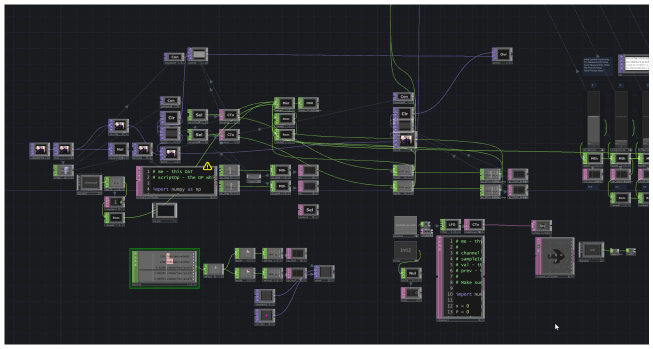


Figure 10. Used Touchdesigner Patch

itself and to transmit the computed position data. The core of the application is the script module of the Kalman filter itself. Here the programming of the simulation in PyCharm could be taken over to a large extent. The transfer of the matrices generated in the init section had to be solved via global variables. Furthermore, all input and output data can be exchanged with the elements outside of the script by a simple assignment. For the variables that had to be processed as Numpy arrays, empty Numpy arrays had to be created first (current best practice). The following features were added:

- Real-time adjustable parameters for Q and R separately for position and acceleration data. This made it possible to directly observe the effects of more or less assumed measurement noise or process noise. Technically, this was implemented by writing the new values to the init after each change and triggering a new init process.
- Real-time error graph. The deviation between the slightly delayed position data of the hand tracking and the continuously updated predictions of the Kalman filter are differentiated and displayed as a trail plot in the image itself.

F. *Data Acquisition and Data Evaluation*

In order to store all data for later analysis in Matlab, a second Touchdesigner patch (logger) was created. This patch received the incoming IMU data in parallel, so there was no delay in recording due to the main patch or the main application. From the main application itself, the logger received the hand tracking data, the output values of the Kalman filter, all measurement and process noise settings, and the interrupt status (cf. V-C). This data was written to an array at 30fps during logging and then saved as a .csv file at the end of each measurement.

V. Results

After the tests with the projection surface and the first mathematical presetting of the parameters for measurement and process noise and the manual fine adjustment of these parameters in the real-time system, the different states could be recorded and evaluated. As a basic guideline for setting the variance of the values for process and measurement noise, it can be said that the smaller the values of the variance for Q (process noise), the greater the confidence in the model, and the smaller the values for the variance R (measurement noise), the greater the confidence in the measurement. The main result was that the implementation worked in real time and that the goal of an improvement by combining both sensor streams was achieved. The results in detail are:

A. *Working Settings with Focus on Sensor Fusion*

To achieve the specification that the earlier arriving acceleration data (cf. 3) can better and more accurately track the moving hand by projection than camera-based tracking alone, the variance of the process noise was set to higher values than the measurement noise (cf. 11). This allowed the Kalman filter to respond quickly to the incoming measurement data. These settings are well suited for a continuous data stream. As can be seen in the evaluation, the calculated ground truth and the values of the Kalman filter are almost on top of each other. The error rate was recorded as an average deviation, which also explains the comparatively low value of -56 dB. Finally, this error rate should also be seen as a value for comparison with the other results. The figure (cf. 11) shows an exemplary excerpt from a whole series of measurements, which did not differ significantly in the deviations achieved. The upper part shows the actual movement of the hand in X- and Y-direction, while the lower part shows the movement in Y-direction over time, as this allows a better representation of the delay or latency compensation. The values for the variance of the measurement and the process noise for the respective evaluation are also included in the plots for better comprehensibility. In the practical test with the projection setup and the hand with the smartphone, these settings could almost reach the set goal. However, it should be noted that a complete simultaneous acquisition of the hand with the projection point was not possible even with this setup, which might be due to the remaining latencies in the calculation and especially in the signal transfer within the system.

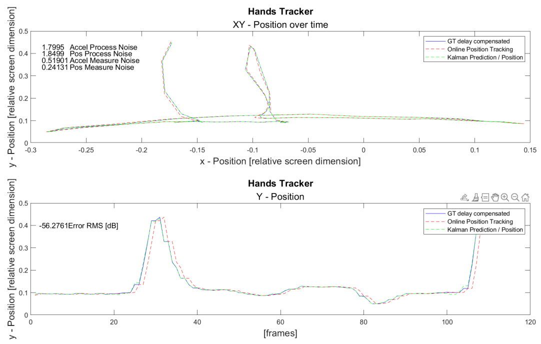


Figure 11. Kalman Filter Working Settings

B. *Overshooting Settings*

To test the function of the Kalman filter and the theoretical prediction that a higher confidence in the model and a lower confidence in the measured data would result in a (not always correct) continuation of the signals in the current direction, tests were also performed with such settings. As a result, the movements caused by the accelerometers are continued by the Kalman filter. In practical application, this means that for an exemplary fast movement to the left, even after the hand has stopped, the prediction by the Kalman filter first continues the recorded movement until the incoming position and acceleration data in the Kalman filter update bring the prediction back to the hand. Thus, the predictions of the Kalman filter overshoot, as can be seen in the evaluation (cf. 12) in the lower panel. It should also be noted that the error rate has increased to about -35dB.

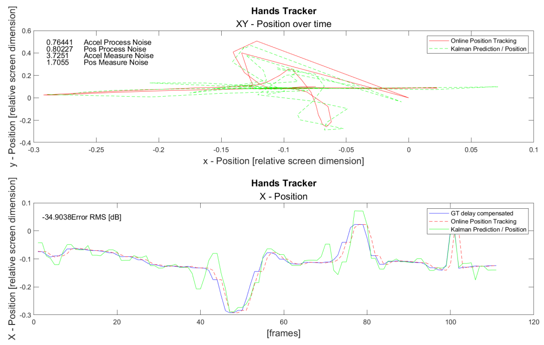


Figure 12. Kalman Filter Overshooting settings Settings

C. Sensor Data Occlusion

Another research goal was to create a configuration of the setup that could compensate for the temporary absence of incoming e.g. position data. For this purpose, the typical dropouts were determined. The typical duration of the dropouts could be determined by practical tests of a setup with a exhibition in relation to a playback rate of 30fps for the durations of 2, 5, 8 and 12 frames. In other words, if the hand tracking was off for a short time, it was mostly off for these lengths. In order to create a realistic scenario, the Touchdesigner programming was extended so that interrupts of these typical frame lengths occur randomly with a minimum time interval between the interrupts themselves, and that these interrupts can be triggered manually. Thus, this behavior could be simulated with the test system. As can be seen in the figure (cf. 13), with a comparatively small value for the variance of the process noise Q and an iterative adjustment of the measurement noise R, a setting was found that can handle the interrupts well, although the quality of the settings from (cf. V-A) could not be reached. I.e. in the practical test there were still small overshoots, but all in all a much better variant than relying only on the incoming position data and making the interrupts directly visible as such.

In order to better quantify the deviations during the interruptions, an additional evaluation was performed. In Matlab, the deviation during the four different interrupt lengths itself was separated in order to be able to perform an evaluation for the different categories using BoxPlot. As can be seen in the

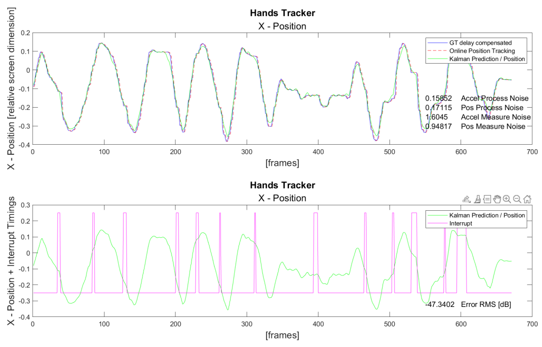


Figure 13. Kalman Filter Interrupt

figure (cf. 14), the deviations themselves as well as the scatter at the values 2, 5 and 8 frames of interruption remain relatively similar and in a tolerable range for practical applications. The deviation at 12 frames of interruption is much more scattered and also the deviation is too large for practical applications with the given settings for the Kalman filter.

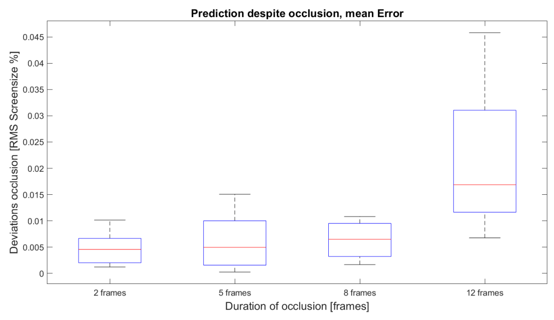


Figure 14. Kalman Filter Interrupt Evaluation

VI. Conclusion and Discussion

After the complete implementation and evaluation it can be stated that the research question can be answered successfully. In the author's opinion, the ability to influence the measurement and system noise parameters in real time using faders, as well as to influence all other parameters of the Kalman filter in a real-time system with direct visibility of the effects of these settings, contributed significantly to the understanding and success of this research. Possible improvements and enhancements are discussed in the following sections.

A. No real Ground Truth

A first improvement to the setup would be to determine the ground truth motion data of the hand independent of the camera-based hand tracking. Currently, a separate recording is used to compute the motion data, which seems to work quite well, but an independent instance, such as an IR-based tracking system with marker, would provide a more valid basis for the ground truth.

B. IMU upright

For the current implementation, the smartphone providing the acceleration data must always be held upright for the acceleration data to be correctly oriented. Of course, it would be possible to use the parallel data from the gyroscope, so that the actual orientation would be independent. The requirement of parallel motion to the screen surface would remain.

C. IMU data pre-filtered?

No valid information can be found about the possible pre-processing of the acceleration data within the used smartphone (IPhone 12 Pro). The graphs of the data look plausible. A comparison with the information of different smartphones and the quality of the individual sensors (see Sensor DB) also positions the quality of the sensors in the middle. Finally, the various statements about the unreliable quality of the smartphone's IMU data indicate that pre-processing takes place in this smartphone.

D. IMU direct connected

An IMU connected directly via USB, for example, would eliminate the possibility of latency via wireless network transmission.

E. Latency Webcam

For the implementation here, the author's device pool of existing cameras was used and the best one was selected according to practical aspects. An industrial camera with e.g. CameraLink interface, which is optimized for low latency transmission, would probably give better results in low light conditions.

F. Latency Projector

Every video projector performs internal signal processing that introduces latency. Typically 1 - 2.5 frames are assumed for professional devices at a refresh rate of 60Hz. Unfortunately, no information could be found for the device used. Knowing the exact value would also improve the measurement.

G. Other Libaries inside Touchdesigner

Currently, the implementation has been done with the widely used library NumPy, which requires the actual implementation of the Kalman filter itself. Specialized libraries such as filterPy would make it easier to test different variants of the Kalman filter, such as the Unscented Kalman filter, to see if a variant specialized for nonlinear data can handle the data better. The installation of external libraries that are not in the Touchdesigner extension catalog (see TD-Pip) is not directly intended. At least the author did not manage it and it would be a good point for future improvement.

Acknowledgment

The author would like to thank Prof. Edeler for his input during the lecture and the implementation which helped to develop the idea.

References

[1] Fang, Wei and Zheng, Lianyu and Deng, Huanjun, "A motion tracking method by combining the IMU and camera in mobile devices" Paper at ResearchGate, 2016.

[2] Takuma Miyazkai, Yuhki Kitazono, "Study on Hand tracking with IMU, Proceedings of the 7th IIAE International Conference on Industrial Application Engineering 2019" Paper at SemanticScholar, 2019.

[3] Camillo Lugaresi and Jiuqiang Tang and Hadon Nash and Chris McClanahan and Esha Uboweja and Michael Hays and Fan Zhang and Chuo-Ling Chang and Ming Yong and Juhyun Lee and Wan-Teh Chang and Wei Hua and Manfred Georg and Matthias Grundmann, "MediaPipe: A Framework for Perceiving and Processing Reality, CVPR" 2019 MediaPipe PDF

[4] Github Repo PhyCon Author, 2023

[5] Roger R. Labbe Jr, "Kalman and Bayesian Filters in Python, an Book on Github with IPyhton Notebooks" 2020 Kalman and Bayesian Filters in Python