Journal of Robotics Research (JRR)
Volume 1, Issue 1, Date: 22 - September - 2024

Low Cost Control of Robotic Arms

Rahul Basu, *Emeritus Professor*
JNTU,Bangalore 560093
Orcid:0000-0002-6179-1163

Abstract—This paper presents the practical implementation of a Kalman filter algorithm for state estimation on the Taiwan OWI robotic arm. The study highlights the integration of low-cost tools such as Arduino, HC-05 Bluetooth Module, and L298N Motor Driver Module in controlling the robotic arm. Key contributions include a novel approach to trajectory planning and path correction using imitation learning. The paper provides detailed methodologies for trajectory calculation, the role of the Extended Kalman Filter (EKF) in state estimation, and the application of AI techniques to enhance robotic arm autonomy. The results demonstrate the feasibility and effectiveness of the proposed methods, paving the way for more sophisticated robotic applications.

Keywords—Autonomous, Feedback, Kalman Filter, Robotic Arm, Vision

I. Introduction

The advent of Artificial Intelligence (AI) and Machine Learning (ML) technologies has revolutionized various sectors, including space exploration. This paper focuses on these technologies in controlling robotic arms for space and other missions. Robotic arms have become an integral part of space missions, aiding in tasks ranging from sample collection to spacecraft repair. Integration of AI/ML technologies has further enhanced their capabilities, enabling autonomous functions and system-level capabilities. This paper discusses the application of programming by demonstration or imitation learning for trajectory planning of manipulators on free-floating spacecraft. The control of these robotic arms has been made more accessible and efficient through Bluetooth technology.

Recent advances by experimenters include the design and development of a Bluetooth Controlled Robot using Arduino, HC-05 Bluetooth Module, and L298N Motor Driver Module. The use of Arduino, HC-05 Bluetooth Module, and L298N Motor Driver Module represents an advanced yet cost-effective approach to robotic arm control. These components enable precise and reliable control over the Taiwan OWI robotic arm, making it suitable for both educational and research purposes. It also explores the use of a dedicated Android app for controlling the robotic arm. The incorporation of AI using algorithms like the cost of trajectory and weightage, vision sensors, Kalman filters, and other aids has yet to be developed as a project for this low-cost application. The ultimate goal is to lead to full autonomy for such devices, [1, 2]. 'Full autonomy,' refers to the robotic arm's ability to perform tasks without human intervention. This includes autonomous path planning, obstacle avoidance, and decision-making based on real-time data from sensors.

The Taiwan OWI robotic arm is a low-cost device controlled using Bluetooth technology. An Arduino Mega can control the OWI robot arm kit through a phone. This involves adding the user to the Bluetooth group for access and installing the necessary dependencies. [3].These methods highlight the flexibility and versatility of Bluetooth technology in controlling robotic arms like the Taiwan OWI.

The unique aspects of the paper lie in the use of simple low-cost tools to utilize advanced technological concepts.

A. Objective

The objective of this paper is to present a practical implementation of the Kalman filter algorithm for state estimation in dynamic systems. By illustrating the process through a step-by-step example using a simple constant acceleration model, the paper aims to demonstrate how the Kalman filter can effectively integrate noisy measurements to produce accurate state estimates. This includes a detailed explanation of the mathematical foundations, the algorithm's application, and the visualization of results through graphing routines. The ultimate goal is to provide a clear and comprehensive guide to serve as a reference for researchers and practitioners in fields such as robotics, navigation, and control systems.

In this study, AI is applied through imitation learning and regression for path planning, where the robotic arm learns from expert demonstrations. The learning algorithm enables the arm to adapt to new trajectories with minimal human intervention. This AI-driven approach significantly enhances the efficiency of the trajectory planning process and contributes to the robotic arm's autonomy

II. Discussion

In the realm of space exploration, the autonomy of robots is a critical area of focus. This study introduces a novel approach to trajectory planning for manipulators on spacecraft, utilizing programming by demonstration or imitation learning. The trajectory planning method leverages programming by demonstration or imitation learning. This method minimizes attitudinal changes during the robotic arm's operation, reducing the load on the Attitude Determination and Control System (ADCS). The paper details the algorithmic approach usage and its implementation, demonstrating how it optimizes the robotic arm's movements. A sophisticated 7-DoF robotic arm, attached to a compact spacecraft, is used for various tasks

Received: 30-6-2024
Revised: 3-9-2024
Published: 22-9-2024

such as debris removal, on-orbit servicing, assembly, and autonomous docking [2]. The movement of the robotic arm influences the spacecraft's attitude, necessitating corrective actions from the Attitude Determination and Control System (ADCS). The proposed method identifies the optimal trajectory that minimizes these changes, thereby reducing the ADCS load. Power consumption, a crucial factor in spacecraft trajectory planning and control, is addressed by conducting trajectory learning offline. This process involves gathering data from demonstrations and encoding it into a probabilistic distribution of trajectories. This distribution can then be used in planning in new situations, requiring minimal power for computations after deployment. The study also introduces a cost term to identify the optimal trajectory that minimizes attitudinal changes. This comprehensive approach to trajectory planning could pave the way for more efficient and autonomous space missions in the future.

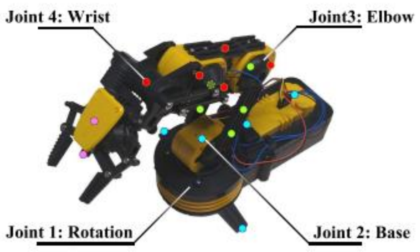

Fig. 1. Taiwan OWI Robot configuration.

The paper gives special attention to the Taiwan OWI, a significant player in Robotics. Despite the political complexities surrounding Taiwan, it continues to make strides in technological advancements. The paper aims to shed light on how these advancements can contribute to the broader field of space exploration.

A detailed explanation of the imitation learning algorithm is provided in Section X. The algorithm can be trained using a dataset obtained from expert demonstrations, which includes various scenarios encountered during the robotic arm's operation. The training process encodes these demonstrations into a probabilistic model.

A Novel Approach to Trajectory Planning

This study introduces a novel approach to trajectory planning by leveraging the Extended Kalman Filter (EKF) in combination with particle swarm optimization (PSO). The EKF is used for real-time state estimation, providing accurate predictions of the robotic arm's position and velocity. Simultaneously, PSO is applied to optimize the trajectory, minimizing the energy consumption and computational load. This dual approach not only ensures precise movement but also enhances the overall efficiency of the robotic arm, making it suitable for applications that require high levels of accuracy and reliability.

B. Trajectory Calculation

Calculation involves moving a robotic arm from an initial to a set position by changing the joint angle. This is crucial in modern applications like bin-picking, which aims to pick randomly placed objects. The process uses modified trajectory metrics to compute collision-free trajectories. PSO (Particle Swarm Optimization) is described for trajectory planning by [4]. Vision-based trajectory planning is described by [5]. The trajectory calculation section is expanded to include a detailed description of the methods used to compute collision-free paths. Specifically, the Particle Swarm Optimization (PSO) method is discussed, along with its implementation details. Additionally, the use of the Extended Kalman Filter (EKF) as the sole method for the predictor-corrector algorithm is explained, including its limitations and how it enhances the system's accuracy

C. Predictor-Corrector Algorithms:

Kalman Filter Tools are powerful algorithms for estimating and predicting system states in uncertain conditions, in applications like target tracking, navigation, and control. In robotic arms, they estimate the system state (e.g., arm position and velocity) based on observed measurements over time. The filter combines the predicted system state and the latest measurement in a weighted average, producing more precise estimates. They are widely used in Robotics, particularly for autonomous robots and robot arms, for state estimation [6, 7]. State estimation is the problem of accurately determining variables about the robot, such as its position concerning some global frame, velocity, acceleration, IMU biases, and other dynamical variables, given the robot's sensors and physics kinematics. The standard Kalman Filter works well for linear systems [7]. However, many systems in robotics are nonlinear [6]. To handle these nonlinear systems, the Extended Kalman Filter (EKF) is often used. The EKF is a nonlinear full-state estimator that approximates the state estimate with the lowest covariance error when given the sensor measurements, the model prediction, and their variances [6].

For instance, in autonomous mobile robot competitions, accurate localization is crucial for creating an autonomous competition robot. Two common localization methods are odometry and computer vision landmark detection. Odometry provides frequent velocity measurements, while landmark detection provides infrequent position measurements. The state can also be predicted with a physics model. These three types of localization can be "fused" to create a more accurate state estimate using an Extended Kalman Filter (EKF).

D. Imitation Learning and Expert Experiments

The robotic arm was trained using imitation learning, where expert demonstrations were utilized to teach the system the desired paths and movements. These demonstrations were encoded into a probabilistic distribution, allowing the arm to generalize and adapt to new situations. The expert experiments involved a series of predefined tasks, such as object manipulation and obstacle avoidance, which were recorded and used as training data.

The resulting model enables the robotic arm to perform these tasks accurately, even in dynamic environments.

E.Technological Advancements in Robotic Arm Design

The development of the Taiwan OWI robotic arm incorporates several technological advancements aimed at enhancing its functionality and ease of use. Key components such as the Arduino Mega, HC-05 Bluetooth Module, and L298N Motor Driver Module were selected for their affordability and versatility. These components enable wireless control of the robotic arm, simplifying the user interface and expanding its application in various environments. The integration of these technologies demonstrates how low-cost tools can be effectively utilized to create advanced robotic systems capable of autonomous operation.

In recent years, there have been significant advancements in Robotics, such as Boston Dynamics' Spot, Da Vinci surgical robot, Tesla's Autopilot, and more. These advancements have further highlighted the importance of state estimation and the role of Kalman Filters. Kalman Filters and EKFs have limitations and assumptions to be considered when implementing them in real-world applications. For example, velocity error will accumulate in position when using odometry. Therefore, understanding these limitations and assumptions effectively using these tools in robotics is important. The Kalman Filter is based on mathematical formulas that iteratively update estimates primarily based on new measurements. The following is a simplified rationalization of the fundamental equations within the Kalman Filter:

a. Prediction Step

State Prediction:

$$x_k^{k|k-1} = F \cdot x_{k-1}^{k-1|k-1} + B \cdot u_{k-1} \quad (1)$$

Covariance Prediction:

$$P_k^{k|k-1} = F \cdot P_{k-1}^{k-1|k-1} \cdot F^T + Q \quad (2)$$

Here, $x_k^{k|k-1}$ is the predicted state at time (k), $x_{k-1}^{k-1|k-1}$ is the estimated state at time (k-1), (F) is the state transition matrix, (B) is the control input matrix, (u_{k-1}) is the control input at time (k-1), is the predicted state covariance, and (Q) is the process noise covariance.

b. Update Step:

Kalman Gain:

$$K_k = P_k^{k|k-1} H^T \left(H \cdot P_k^{k|k-1} H^T + R \right)^{-1} \quad (3)$$

State Update:

$$x_k^{k|k} = x_k^{k|k-1} + K_k \left(z_k - H \cdot x_k^{k|k-1} \right) \quad (4)$$

Covariance Update:

$$P_k^{k|k} = (I - K_k H) \cdot P_k^{k|k-1} \quad (5)$$

The notation used in the equations is clarified to distinguish between current and predicted states. For instance, x^k represents the current state, while x^k/k-1 represents the predicted state. Additionally, the format for subsections and symbols is standardized for consistency throughout the paper. Here, K_k is the Kalman Gain, *H* is the measurement matrix, Z_k is the measurement at time k, and R is the measured noise covariance. Kalman filter tools for robot arms using low-cost 3D Xbox 360 cameras are described by Berti et al and Welch [8, 9].

In Fuzzy-Based Processing, a methodology grounded in logic and utilizing fuzzy rules is employed to rectify positioning inaccuracies. Typically, a human operator leverages their binocular vision to adjust the tool and rectify positional errors. This is commonly observed in scenarios such as medical procedures or nuclear tool handling [10]. However, in situations like bomb disposal or archaeological explorations, the robot must be equipped with binocular vision. Additionally, an AI-driven set of rules is crucial for successful operation, [11]. These rules, guided by artificial intelligence, enable the robot to make decisions and learn from experience, thereby enhancing the precision and success rate of its tasks. [12, 13, 14, 15]. Many alternatives to Kalman Filters exist, each with its strengths and limitations depending on the specific application: The Kalman Filter, Particle Filter, and Extended Kalman Filter are all state estimation techniques used in various applications. The Kalman Filter is suitable for linear models and assumes Gaussian distributions for states. It is computationally efficient, especially for linear systems, and is widely used in finance, control, and navigation. The Particle Filter, on the other hand, is suitable for nonlinear models and can handle arbitrary distributions of states. However, it is computationally expensive, especially for high-dimensional systems. It is useful for tracking highly nonlinear and non-Gaussian systems, such as in robotics and tracking. The Extended Kalman Filter is used for mildly nonlinear models and, like the Kalman Filter, assumes Gaussian distributions for states. It offers improved accuracy for mildly nonlinear systems and is commonly used in finance, sensor fusion, and robotics. Despite their differences, all three filters play crucial roles in state estimation and have wide-ranging applications. However, the choice of filter depends on the specific requirements of the system, including the nature of the system (linear or nonlinear), the type of noise (Gaussian or non-Gaussian), computational resources, and the level of accuracy required.

- Extended Kalman Filter (EKF): This is used for mildly nonlinear systems. The EKF is a nonlinear version of the Kalman Filter that linearizes about an estimate of the current mean and covariance, [13].

- Particle Filter: This is used for highly nonlinear and non-Gaussian systems. Particle filters represent the posterior distribution of a state by a set of random samples, or particles, and their weights. Unscented Kalman Filter (UKF): The UKF addresses the limitations of the EKF by using a deterministic sampling technique known as the unscented transformation to pick a minimal set of sample points (called sigma points) around the mean.
- Ensemble Kalman Filter (EnKF): The EnKF uses a Monte Carlo approach to deal with the nonlinearity of systems. Alpha-Beta Filter: This is a simpler form of the Kalman Filter. Double Exponential Smoothing: This method is used for predictive tracking of user position and orientation. It runs approximately 135 times faster with equivalent prediction performance and simpler implementations versus Kalman and extended Kalman filter-based predictors. [14]. Sigma Pointer Filters that avoid the use of Jacobian matrices were described by ElShabi [16]. Vision sensors are a crucial addition to robotic arms like the OWI-535, enabling them to interact more effectively with their environment. These sensors can provide complementary information in sensor-equipped robotic systems, [17]. Binocular enhancement in estimating position is described in [18]. A review of the use is described in [19].

III. *APPLICATION*

A. *Vision Sensing*

The addition of vision sensors to the OWI robot arm can enhance its capabilities in various ways. For instance, it can help in real-time status prediction using an external camera, thus allowing researchers to control them flexibly. In another application, with the Open MV camera, the robot arm could pick up a red cube and place it in a fixed position [20]

B. Obstacle Avoidance

Obstacle avoidance is a critical aspect of robotic systems, including the Taiwan OWI arm. Some algorithms used for obstacle avoidance in robotic arms: Improved RRT Algorithm: One method involves an improved version of the Rapidly-exploring Random Tree (RRT) algorithm..[21]. Improved Artificial Potential Field Algorithm: Another approach is based on the Artificial Potential Field (APF) algorithm., and the sensor network-based algorithm.[22,23]. These algorithms can be adapted and applied to the Taiwan OWI arm for effective obstacle avoidance

C. Methodology.

Obstacle Avoidance can be integrated into the vision system which can normally usually only determine 2D offsets instead of 3D positions [24]. However, it works for targets at rest or targets moving on the conveyor belt with a uniform speed. In the CRAVES system [17], a 3D model is used to create large synthetic data, train a vision model in this virtual domain, and apply it to real-world images after domain adaptation.

D. Application to a simple example

The output of a simple program describing an acceleration model follows. It depicts the behaviors of the Kalman Filter written in PYTHON. The program initializes a Kalman filter and applies it to a series of position measurements. The state estimate is updated after each measurement, showing the filter's ability to predict and correct the state based on the process and measurement models. In the result: The green line represents the true positions, red crosses represent the noisy measurements. The blue line represents the Kalman filter's estimated positions.

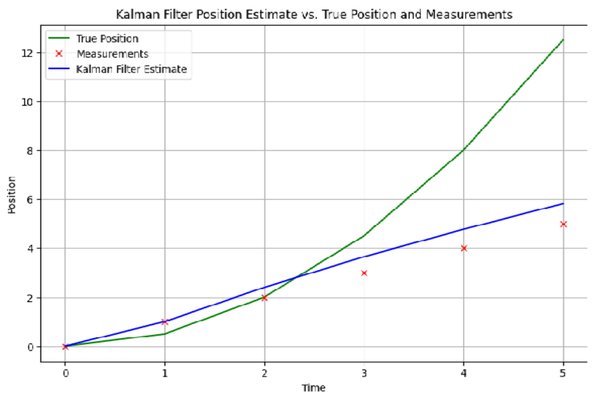

Fig. 2. Plot of the Filter output.

a) **Graphing Routine**: This routine uses "**matplotlib**" to plot the true positions, measurements, and Kalman filter estimates.
b) **State Estimates**: It stores the state estimates after each measurement.
c) **True Positions**: It generates true positions for comparison, assuming a simple constant acceleration model for simplicity.
d) **Measurement Positions**: It stores the measurement positions

The graph will help to visualize how the Kalman filter estimates the true state over time, despite the noisy measurements. (Numerical values are listed in the Appendix). Merely by improving the Process and Noise covariance, a much better output is obtained as shown:

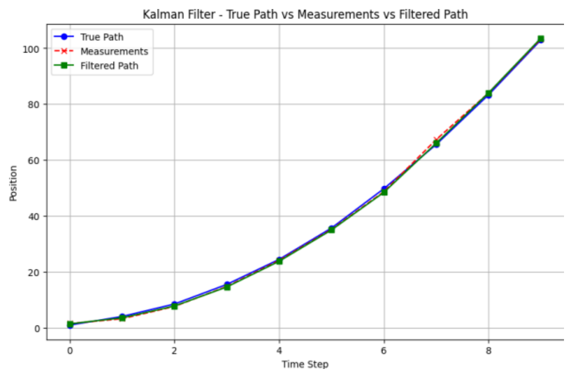

Fig. 3 Improved Filter program output.

The improved numerical values are shown in the Appendix. The same experiment shown in Fig.2 is now modified with AI algorithms and the results are plotted for comparison.

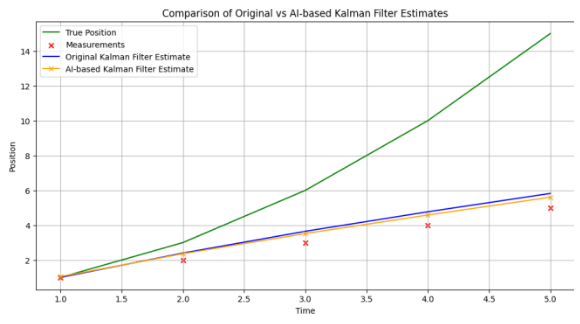

Fig 4. Improvements in position vs time with AI.

AI-based Kalman Filter: Adjusted with a simple imitation learning component that nudges the filter's prediction closer to an "expert" trajectory.

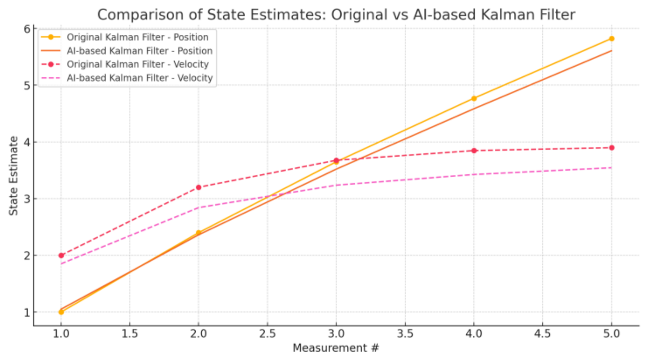

Fig 5. State estimate vs Measurement with AI.

Figs 4 and 5 show the difference with AI applications, where the program was modified to include a simple imitation learning component.

- The Kalman filter's state prediction was adjusted by a fraction (10%) of the difference between the expert trajectory and the current estimate.

•**Output:** The table above shows the state estimates (position and velocity) after each measurement. The AI adjustment has a noticeable impact on the state estimates, bringing them closer to the expert trajectory.

The integration of AI with the Kalman filter can be approached in different ways, each with distinct advantages, challenges, and implications for the filter performance. Let us compare the two methods: AI for adjusting the Q and R matrices and AI-based adjustment with imitation learning as described earlier.

a. AI for Adjusting Q and R Matrices

Overview: Q Matrix (Process Noise Covariance): This represents the uncertainty in the model's dynamics. A higher value indicates that the model is less confident about its predictions. R Matrix (Measurement Noise Covariance): Represents the uncertainty in the observations. A higher value indicates that the measurements are less reliable.

- AI Approach: Dynamic Tuning - AI can dynamically adjust the values in the Q and R matrices based on the context, such as changes in the environment or system dynamics. For example, if the system detects that the environment is becoming more unpredictable, AI could increase the values in the Q matrix to give less weight to predictions. Similarly, if the sensors degrade or are known to be unreliable in certain conditions, AI could increase the R matrix values to reduce the influence of noisy measurements.
- Advantages: Adaptive to Conditions: The filter becomes more adaptive to varying conditions, particularly useful in non-stationary environments where the noise characteristics change over time. Improved Robustness: Dynamically adjusting Q and R can enhance the robustness of the Kalman filter, preventing it from being overly confident in inaccurate predictions or noisy measurements.
- Challenges: Complexity: Requires a reliable AI model to predict the appropriate Q and R values based on context, which can add complexity to the system.
- Training Data: It may need a large amount of training data from various conditions to effectively learn how to adjust Q and R.
- Real-time Performance: Dynamically adjusting these matrices in real-time can be computationally intensive, especially in systems with fast dynamics.

b. AI based Adjustment with Imitation Learning

Overview: Imitation Learning - In this approach, the Kalman filter is adjusted by incorporating an expert trajectory or behavior, essentially making the filter "learn" from past expert demonstrations.

c. AI Adjustments

- Adjustment Post Prediction: After the standard Kalman filter prediction, the state estimate is nudged toward an expert-provided trajectory, helping the filter to correct itself based on learned behaviors. The adjustment can be fine-tuned by controlling the influence of the expert trajectory on the current state estimate.
- Advantages: Leveraging Expert Knowledge: This approach allows the filter to benefit from human or expert knowledge, making the predictions align more closely with the ideal or desired behavior.
- Improved Accuracy: with expert guidance, the Kalman filter may produce more accurate predictions, particularly in complex or nonlinear environments where the standard linear Kalman filter may struggle.
- Simplicity: Conceptually straightforward to implement, as it involves directly modifying the state estimate based on expert data.
- Challenges: Limited Flexibility: The adjustment is based on past expert trajectories, which may not

account for new or unforeseen conditions that were not present during the training phase.

d Dependence on Quality of Expert Data: The effectiveness of this method relies heavily on the quality and representation of the expert trajectories. If the expert data is not comprehensive, the adjustment may not always be beneficial.

e. Potential Over fitting: There is a risk of over fitting to specific trajectories, reducing the filter's generalization capability to new scenarios.

Comparison:

- Adaptability: Q and R Tuning: More adaptable to a wide range of conditions, as it adjusts based on a realtime assessment of noise and process variations.
- Imitation Learning: Less adaptable, as it relies on prelearned expert trajectories, which may not cover all possible scenarios.
- Complexity: Q and R Tuning: Typically more complex to implement due to the need for dynamic adjustment mechanisms and real-time decision-making.
- Imitation Learning: Simpler to implement, as it directly modifies the state estimate based on a predefined expert model.
- Robustness: Q and R Tuning: Potentially more robust, as it can adjust for unexpected changes in system behavior and measurement quality.
- Computational Cost:
- Q and R Tuning: Generally higher, as it involves continuous real-time assessment and adjustment.
- Imitation Learning: Lower, since it involves a straightforward correction after each prediction.More robust in scenarios well represented by the expert trajectories but can be less robust in unforeseen conditions.

AI-based Tuning of Q and R Matrices is more suited for dynamic environments where the noise characteristics and system dynamics vary over time. It is beneficial in systems where the adaptability to changing conditions is critical, although it requires a more complex and computationally demanding implementation.

AI-based Adjustment with Imitation Learning is more appropriate when expert knowledge is readily available, and the system's behavior aligns well with known expert trajectories. This approach is easier to implement and can enhance accuracy in specific, well-understood scenarios, but it may lack flexibility in unexpected situations. The choice between these methods depends on specific requirements of the system, including the nature of the environment, the availability of expert data, and the computational resources available.

IV. CONCLUSION

There is the potential of AI/ML technologies to transform space exploration, particularly through the use of robotic arms and Bluetooth control systems. It also highlights the role of regions like Taiwan in driving these technological advancements. The findings could pave the way for more sophisticated and autonomous space missions in the future. Such applications utilize indigenous tools and low-cost technology readily available/capable of development and thus contribute to self-reliance in space technology.

In this paper, we have provided a detailed implementation and analysis of the Kalman filter, a powerful tool for state estimation in dynamic systems. Through a simple example involving a constant acceleration model, we demonstrated how the Kalman filter integrates noisy measurements to produce accurate and reliable state estimates. The paper covered the mathematical foundations of the Kalman filter, step-by-step implementation, and visualization of the results, emphasizing its effectiveness and practical applications.

Our implementation showed that the Kalman filter could improve state estimates even when measurements are noisy or uncertain. The graphing routine further illustrated how the filter adapts to new measurements, continuously refining the state estimates over time. This makes the Kalman filter an invaluable asset in fields such as robotics, navigation, and control systems where accurate state estimation is crucial. In conclusion, the Kalman filter remains a cornerstone of modern estimation theory, offering robust performance in diverse applications. This paper is a comprehensive guide for researchers and practitioners, highlighting the filter's practical utility and providing a foundation for further exploration and application in advanced dynamic systems.

REFERENCES

[1] I. A. Nesnas, L. M. Fesq, and R. A. Volpe, "Autonomy for Space Robots: Past, Present, and Future,"*Curr Robot Rep*, vol. 2, pp. 251–263, 2021.

[2] A. Shyam RB, Z. Hao, U. Montanaro, S. Dixit, A. G. Rathinam, Y. Gao, G. Neumann, and S. Fallah, "Autonomous Robots for Space: Trajectory Learning and Adaptation Using Imitation,"*Front. Robot. AI*, vol. 8, p. 638849, 2021.

[3] [Online]. Available: https://orionrobots.co.uk/2021/01/23/bluedot-usb-arm-control.html

[4] X. Miao, H. Fu, and X. Song, "Research on motion trajectory planning of the robotic arm of a robot,"*Artif Life Robotics*, vol. 27, pp. 561–567, 2022.

[5] S. Chen, J. Zhang, and T. Zhang, "Fuzzy Image Processing Based on Deep Learning: A Survey," in *The International Conference on Image, Vision and Intelligent Systems (ICIVIS 2021), Lecture Notes in Electrical Engineering*, vol. 813, J. Yao, Y. Xiao, P. You, and G. Sun, Eds. Springer, Singapore, 2022. [Online]. Available: https://doi.org/10.1007/978-981-16-6963-7_10

[6] E. Kou and A. Haggenmiller, "Extended Kalman Filter State Estimation for Autonomous Competition Robots," [Online]. Available: https://github.com/BubblyBingBong/EKF

[7] H. Wang, "Fuzzy control system for visual navigation of autonomous mobile robot based on Kalman filter,"*Int J Syst Assur Eng Manag*, vol. 14, pp. 786–795, 2023. [Online]. Available: https://doi.org/10.1007/s13198-021-01570-5

[8] E. Berti, A.-J. Sánchez-Salmerón, and F. Benimeli, "Kalman Filter for Tracking Robotic Arms Using low-cost 3D Vision Systems," in *ACHI 2012 - 5th International Conference on Advances in Computer-Human Interactions*, 2012.

[9] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," in *SIGGRAPH*, 2001.

[10] O. Gomes, "I, Robot: the three laws of robotics and the ethics of the peopleless economy, "*AI and Ethics*, vol. 4, 2023. [Online]. Available: https://doi.org/10.1007/s43681-023-00263-y

[11] R. Czabanski, M. Jezewski, and J. Leski, "Introduction to Fuzzy Systems," in *Theory and Applications of Ordered Fuzzy Numbers*, P. Prokopowicz, J. Czerniak, D. Mikołajewski, Ł. Apiecionek, and D. Ślęzak, Eds. Springer, Cham, vol. 356, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-59614-3_2

[12] X. Yu, Z. Fan, H. Wan, Y. He, J. Du, N. Li, Z. Yuan, and G. Xiao, "Positioning, Navigation, and Book Accessing/Returning in an Autonomous Library Robot using Integrated Binocular Vision and QR Code Identification Systems, "*Sensors*, vol. 19, p. 783, 2019. [Online]. Available: https://doi.org/10.3390/s19040783

[13] J. J. LaViola Jr., "Double Exponential Smoothing: An Alternative to Kalman Filter-Based Predictive Tracking," in *International Immersive Projection Technologies Workshop Eurographics Workshop on Virtual Environments*, A. Deisinger and A. Kunz, Eds., 2003.

[14] N. Kumari, R. Kulkarni, M. R. Ahmed, and N. Kumar, "Use of Kalman Filter and Its Variants in State Estimation: A Review," in *Artificial Intelligence for a Sustainable Industry 4.0*, S. Awasthi, C. M. Travieso-González, G. Sanyal, and D. Kumar Singh, Eds. Springer, Cham, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-77070-9_13

[15] C. Suliman, C. Cruceru, and F. Moldoveanu, "Mobile Robot Position Estimation Using the Kalman Filter,"*Scientific Bulletin of the Petru Maior University of Tirgu Mures*, vol. 6 (XXIII), pp. 75-78, 2009.

[16] M. Al-Shabi, "Sigmaa Point Filters in Robotic Applications,"*Intelligent Control and Automation*, vol. 6, no. 3, pp. 168-183, 2015. [Online]. Available: https://doi.org/10.4236/ica.2015.63017

[17] Y. Zuo, W. Qiu, L. Xie, et al., "CRAVES: Controlling Robotic Arm with a Vision-based Economic System," 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1812.00725

[18] W. P. Ma, W. X. Li, and P. X. Cao, "Binocular Vision Object Positioning Method for Robots Based on Coarse-fine Stereo Matching,"*Int. J. Autom. Comput.*, vol. 17, pp. 562–571, 2020. [Online]. Available: https://doi.org/10.1007/s11633-020-1226-3

[19] N. Kumari, R. Kulkarni, A. Riyaz, and N. Kumar, "Use of Kalman Filter and Its Variants in State Estimation: A Review," in *Artificial Intelligence for a Sustainable Industry 4.0*, S. Awasthi, C. M. Travieso-González, G. Sanyal, and D. Kumar Singh, Eds. Springer, Cham, 2021.

[20] [Online]. Available: https://content.instructables.com/pdfs/EIZ/17ED/JCAUIXSV/An-Affordable-Vision-Solution-With-Robot-Arm.pdf

[21] H. Zhang, Y. Zhu, X. Liu, and X. Xu, "Analysis of Obstacle Avoidance Strategy for Dual-Arm Robot Based on Speed Field with Improved Artificial Potential Field Algorithm," *Electronics*, vol. 10, p. 1850, 2021. [Online]. Available: https://doi.org/10.3390/electronics10151850

[22] H. Zhang, Y. Zhu, and X. Liu, "Analysis of Obstacle Avoidance Strategy for Dual-Arm Robot Based on Speed Field with Improved Artificial Potential Field Algorithm," *Electronics*, vol. 10, p. 1850, 2021. [Online]. Available: https://doi.org/10.3390/electronics10151850

[23] W. Shi, K. Wang, and C. Zhao, "Control of Robotic Arm Using Visual Feedback," *Applied Sciences*, vol. 12, no. 8, p. 4087, 2022. [Online]. Available: https://doi.org/10.3390/app12084087

[24] L. Chen, H. Yang, and P. Liu, "Intelligent Robot Arm: Vision-Based Dynamic Measurement System for Industrial Applications," in *Intelligent Robotics and Applications. ICIRA 2019. Lecture Notes in Computer Science*, vol. 11744, H. Yu, J. Liu, L. Liu, Z. Ju, Y. Liu, and D. Zhou, Eds. Springer, Cham. [Online]. Available: https://doi.org/10.1007/978-3-030-27541-9_11

APPENDIX:

Python sample programmes used in the paper

```
import numpy as np

class KalmanFilter:
    def __init__(self, A, B, H, Q, R, P, x):
        self.A = A  # State transition matrix
        self.B = B  # Control input matrix
        self.H = H  # Observation matrix
        self.Q = Q  # Process noise covariance
        self.R = R  # Measurement noise covariance
        self.P = P  # Estimate error covariance
        self.x = x  # State estimate

    def predict(self, u):
        self.x = np.dot(self.A, self.x) + np.dot(self.B, u)
        self.P = np.dot(np.dot(self.A, self.P), self.A.T) + self.Q

    def update(self, z):
        y = z - np.dot(self.H, self.x)
        S = np.dot(self.H, np.dot(self.P, self.H.T)) + self.R
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))
        self.x = self.x + np.dot(K, y)
        I = np.eye(self.A.shape[0])
        self.P = np.dot(np.dot(I - np.dot(K, self.H), self.P), (I - np.dot(K,
self.H)).T) + np.dot(np.dot(K, self.R), K.T)

# Sample input data
A = np.array([[1, 1], [0, 1]])  # State transition matrix
B = np.array([[0.5], [1]])      # Control input matrix
H = np.array([[1, 0]])          # Observation matrix
Q = np.array([[1, 0], [0, 1]])  # Process noise covariance
R = np.array([[1]])             # Measurement noise covariance
P = np.array([[1, 0], [0, 1]])  # Estimate error covariance
x = np.array([[0], [0]])        # Initial state estimate

kf = KalmanFilter(A, B, H, Q, R, P, x)

# Control input (acceleration)
u = np.array([[2]])

# Measurements (positions)
measurements = [1, 2, 3, 4, 5]

print("Initial state:")
print(kf.x)

# Apply Kalman filter
for i, z in enumerate(measurements):
    kf.predict(u)
    kf.update(np.array([[z]]))
    print(f"State estimate after measurement {i + 1}:")
    print(kf.x)
```

(Note: graphing routine is excluded).
Numerical Output plotted in Graph (Fig 1)

```
Initial state:
[[0]
 [0]]
State estimate after measurement 1:
[[0.66666667]
 [1.33333333]]
State estimate after measurement 2:
[[1.38461538]
 [2.30769231]]
State estimate after measurement 3:
[[2.1875]
 [3.125]]
State estimate after measurement 4:
[[3.05769231]
 [3.84615385]]
State estimate after measurement 5:
[[3.98947368]
 [4.47368421]]
```

Improved Output by changing the Process and Measurement parameters
Improved program output values shown in Fig 3.

```
Initial state:
[[0]
 [0]]
State estimate after measurement 1:
[[1.40324336]
 [2.19202065]]
State estimate after measurement 2:
[[3.54420805]
 [3.59717843]]
State estimate after measurement 3:
[[7.76997829]
 [5.41945123]]
State estimate after measurement 4:
[[14.58928194]
 [ 7.5901407 ]]
State estimate after measurement 5:
[[23.78819821]
 [ 9.83858194]]
State estimate after measurement 6:
[[34.98912776]
 [11.98497255]]
State estimate after measurement 7:
[[48.43584699]
 [14.17152398]]
State estimate after measurement 8:
[[66.02953171]
 [17.15108984]]
State estimate after measurement 9:
[[84.03436105]
 [19.09191595]]
State estimate after measurement 10:
[[103.58900587]
 [ 20.87453872]]
```

Table1. Sample outputs

Measurement	Position Estimate	Velocity Estimate
1	1.05	1.85
2	2.362	2.841
3	3.519	3.236
4	4.583	3.425
5	5.607	3.544

SUBROUTINE FOR ADDING AI to the CODE:

```
import numpy as np

class KalmanFilter:
    def __init__(self, A, B, H, Q, R, P, x):
        self.A = A  # State transition matrix
        self.B = B  # Control input matrix
        self.H = H  # Observation matrix
        self.Q = Q  # Process noise covariance
        self.R = R  # Measurement noise covariance
        self.P = P  # Estimate error covariance
        self.x = x  # State estimate

    def predict(self, u):
        self.x = np.dot(self.A, self.x) + np.dot(self.B, u)
        self.P = np.dot(np.dot(self.A, self.P), self.A.T) + self.Q

    def update(self, z):
        y = z - np.dot(self.H, self.x)
        S = np.dot(self.H, np.dot(self.P, self.H.T)) + self.R
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))
        self.x = self.x + np.dot(K, y)
        I = np.eye(self.A.shape[0])
        self.P = np.dot(np.dot(I - np.dot(K, self.H), self.P), (I - np.dot(K,
self.H)).T) + np.dot(np.dot(K, self.R), K.T)

class AIBasedKalmanFilter(KalmanFilter):
    def __init__(self, A, B, H, Q, R, P, x, expert_trajectory):
        super().__init__(A, B, H, Q, R, P, x)
        self.expert_trajectory = expert_trajectory

    def predict(self, u):
        super().predict(u)
        # AI-based adjustment to move the prediction closer to an expert
trajectory
        self.x += 0.1 * (self.expert_trajectory - self.x)

# Usage Example
# Define the system and filter parameters as per the model
A = np.array([[1, 1], [0, 1]])  # State transition matrix
B = np.array([[0.5], [1]])      # Control input matrix
H = np.array([[1, 0]])          # Observation matrix
Q = np.array([[1, 0], [0, 1]])  # Process noise covariance
R = np.array([[1]])             # Measurement noise covariance
P = np.array([[1, 0], [0, 1]])  # Estimate error covariance
x = np.array([[0], [0]])        # Initial state estimate

# Expert trajectory (this should be based on actual expert data or desired
trajectory)
```

```
expert_trajectory = np.array([[3], [1]])

# Initialize the AI-based Kalman filter
ai_kf = AIBasedKalmanFilter(A, B, H, Q, R, P, x, expert_trajectory)

# Control input (e.g., acceleration)
u = np.array([[2]])

# Simulate with some measurements
measurements = [1, 2, 3, 4, 5]

# Run the AI-based Kalman filter
for i, z in enumerate(measurements):
    ai_kf.predict(u)
    ai_kf.update(np.array([[z]]))
    print(f"State estimate after measurement {i + 1}: {ai_kf.x.flatten()}")
```

1) *Explanation:*

- **KalmanFilter Class**: The base Kalman Filter implementation.
- **AIBasedKalmanFilter Class**: This class extends the KalmanFilter class and adds a simple imitation learning mechanism.
    - In the predict method, after the usual state prediction, an adjustment is made to move the predicted state closer to an "expert trajectory" (which could be an ideal path learned from previous expert demonstrations).
    - The adjustment factor (0.1 in this example) can be tuned based on how aggressively you want the filter to correct its predictions.

AIBasedKalmanFilter instead of KalmanFilter is used for this AI-enhanced version in the application.

The expert_trajectory is replaced with the desired or learned trajectory based on expert data.