State Estimation

1st Erik Alexander Grieg
Electrical Engineering & Computer Sciences
University of California, Berkeley
Berkeley, USA
erikagr@berkeley.edu

2nd Lars Hammernes Leopold
Electrical Engineering & Computer Sciences
University of California, Berkeley
Berkeley, USA
larshle@berkeley.edu

***Abstract*—This paper implements and compares three state estimation algorithms: dead reckoning, Kalman filter, and the extended Kalman filter. Dead reckoning was the easiest to implement but performed the worst due to the compunding errors over time. The Kalman-based methods provided better accuracy. These results were consistent when tested on both a TurtleBot and a quadcopter, demonstrating that Kalman filtering techniques are necessary for reliable and accurate state estimation in robotic applications.**

I. Introduction

II. Method

A. Dead Reckoning

The dead reckoning estimator in this paper is motivated by the classical forward Euler method for solving ordinary differential equations, where the continuous dynamics

$$\dot{x} = f(x, u)$$

are approximated by the discrete update

$$x_{k+1} = x_k + f(x_k, u_k)\Delta t,$$

as given in Equation (12) in [1]. This first-order approximation, while computationally efficient, accumulates errors over time, especially in the presence of process and measurement noise.

For the unicycle model, the state is updated by integrating its kinematics. A key line in the implementation of the TurtleBot estimator is:

```
x_new = x_prev + (self.r / 2)
* (w_r + w_l) * np.cos(phi_prev) * self.dt
```

This line computes the new x-position by averaging the wheel speeds (dividing by 2), converting the angular velocities to linear velocity via the wheel radius, and projecting this displacement along the x-axis using $\cos(\phi_{\text{prev}})$.

Similarly, for the Quadrotor, a forward Euler scheme is employed, but with an extended state vector that includes vertical motion and rotational dynamics. The estimator updates the Quadrotor's horizontal and vertical positions, orientation, and their corresponding velocities by integrating the accelerations computed from the thrust and angular inputs. For example, the horizontal acceleration is determined by

$$\dot{x}_d = \frac{\text{thrust}}{m} \sin(\phi_{\text{prev}}),$$

which is then integrated over Δt to update the velocity and position.

While these implementations effectively capture the state evolution, with the process noise assumed to be zero, they are prone to drift when noise is present. This limitation motivates the use of higher order integration methods, such as the Kalman filter or the extended Kalman filter, to achieve more robust state estimation.

B. Kalman Filter

The Kalman filter was only implemented for the TurtleBot and leverages a linearized unicycle model at a fixed bearing ($\pi/4$) to recursively estimate the TurtleBot's state. The state update proceeds in two phases: prediction and correction. In the prediction phase, the state is propagated using the system model

$$X_{\text{pred}} = \mathbf{A}X_{\text{prev}} + \mathbf{B}u,$$

and the associated error covariance is updated as

$$P_{\text{pred}} = \mathbf{A}P\mathbf{A}^T + Q.$$

In the correction phase, sensor measurements are incorporated by computing the innovation $y_{\text{tilde}} = z - \mathbf{C}X_{\text{pred}}$ and updating the state with the Kalman gain

$$K = P_{\text{pred}}\mathbf{C}^T \left(\mathbf{C}P_{\text{pred}}\mathbf{C}^T + R\right)^{-1}.$$

This results in the updated state $X_{\text{new}} = X_{\text{pred}} + K\, y_{\text{tilde}}$ and covariance $P_{\text{new}} = (I - K\mathbf{C})P_{\text{pred}}$.

Physically, the matrix Q encapsulates the uncertainty in the process model and accounts for unmodeled dynamics and external disturbances (such as slip or subtle nonlinearities introduced by the fixed linearization at $\pi/4$). A higher Q implies that the model is less trusted, leading the filter to weigh the measurements more heavily during the update. Conversely, if Q is set too low relative to the actual process noise, the filter may become overconfident in its predictions, causing it to under-correct when measurements are received.

The measurement noise covariance R represents the sensor noise characteristics, including inherent sensor inaccuracies, quantization errors, and potential biases. Underestimating R can result in the filter giving excessive trust to noisy measurements, which may lead to erratic estimates. Overestimating R makes the filter rely more on its prediction, which is risky if the process model does not fully capture the true dynamics.

The initial error covariance P_0 reflects the confidence in the initial state x_0. A larger P_0 indicates lower confidence in the starting estimate, allowing the filter to adapt more quickly when new measurements arrive. In contrast, a smaller P_0 implies high confidence in x_0, which might delay the filter's convergence if the initial state is not accurately known.

In simulation, these parameters are typically fine-tuned by comparing the filter's state estimates against the ground truth. By iteratively adjusting Q, R, and P_0 to minimize the estimation error, one can achieve a balance that yields both accurate and stable estimates. This process often starts with values derived from sensor specifications and model uncertainties, and then proceeds through iterative refinement.

For a real robot without access to ground truth, the tuning process must rely on sensor calibration experiments, empirical data, and adaptive techniques. In such scenarios, one might perform controlled experiments to measure the sensor noise characteristics or use statistical analysis of the innovation sequence (the difference between the predicted and measured outputs) to adjust Q and R online. Additionally, domain expertise and knowledge of the system's dynamics are crucial in setting reasonable initial values. Adaptive filtering methods may also be employed to update these covariances in real time as operating conditions change.

This careful balancing of model uncertainty (through Q), measurement noise (through R), and initial state confidence (through P_0) is central to the robustness and performance of the Kalman filter. It ensures that the filter effectively fuses the process model with sensor observations, maintaining reliable state estimates even in the presence of disturbances and measurement errors [2].

Tuning of Covariance Matrices: The tuning process for the Kalman filter's covariances was iterative. Initially, identity matrices were used for Q, R, and P_0. However, after experiments in which the estimated states were compared against ground truth, adjustments were made to better balance the trust between the process model and sensor measurements. Ultimately, the following values were chosen:

$$\mathbf{Q} = 0.01\, I_6,$$

$$\mathbf{R} = \mathrm{diag}(10,\ 1),$$

$$\mathbf{P_0} = \mathbf{0}_6,$$

where I_6 denotes the 6×6 identity matrix, $\mathrm{diag}(10,\ 1)$ is the 2×2 diagonal matrix with entries 10 and 1, and $\mathbf{0}_6$ is the 6×6 zero matrix.

Lowering the values in Q and R (relative to the initial identity matrices) indicated that the process and measurement noises were relatively low, thereby allowing the filter to yield a more accurate and stable estimate. In this configuration, the filter is less overconfident in its predictions, and the balance between prediction and correction leads to reduced drift over time.

The chosen values, reflect a trade-off that minimizes estimation error in our simulated environment. Q was chosen to be smaller than R as we suspected that the forward state calculations were more reliable than the measurements. Additionally, we trusted the measurement of the angle more than the distance measurement. This is reflected in R where the upper left entry is ten times higher than the bottom right entry.

For real-world scenarios, where noise characteristics might vary, a similar iterative tuning process, possibly augmented by adaptive filtering techniques, would be employed in the absence of ground truth. Specifically, the choice of P_0 would most likely have been higher as starting the system in the same state everytime could prove difficult. Since we only worked in simulation and therefore knew the exact initial contidions, we chose P_0 to be zero.

C. The Extended Kalman Filter (EKF)

The extended Kalman filter (EKF) handles nonlinearity by linearizing the nonlinear system dynamics using a first-order Taylor series expansion around the current state estimate. In practice, this means that the EKF approximates the nonlinear functions with their Jacobian matrices, effectively replacing the true nonlinear model with a linear one that is valid in the vicinity of the current estimate.

However, this linearization technique has an important limitation. The linear approximation is only accurate when the system's state remains close to the point of linearization. If the system is highly nonlinear or if the estimation error is large, the Jacobian may poorly approximate the true dynamics. This can lead to suboptimal performance or even filter divergence.

To mitigate this limitation, one can consider several approaches. One alternative is to decrease the time step in the calculation. This helps ensure that the state does not deviate too far between updates, making the linear approximation more valid. Another alternative is to iteratively re-linearize around updated state estimates within the same time step. This can improve the accuracy of the linear approximation. These alternatives can mitigate the limits of EKF, but there is no guarantee.

III. Experimental Results

Figures 1, 2, 3 and 4 illustrate the performance of the state estimators in different scenarios. For the TurtleBot, as shown in Figure 1, dead reckoning closely follows the true trajectory initially, but estimation errors accumulate over time due to its inability to compensate for process and measurement noise. A similar trend is observed for the Quadrotor scenario in Figure 2, where the estimator captures the overall motion in both horizontal and vertical directions, yet the error grows with time. In contrast, Figure 3 demonstrates that the Kalman filter, by integrating sensor measurements, significantly reduces the drift and more accurately tracks the true state. Similarly Figure 4 shows the near perfect accuracy of the EKF. These results confirm that while dead reckoning can provide reasonable estimates in a noise-free context, incorporating sensor fusion through Kalman filtering methods yield a more robust and reliable state estimation in the presence of noise.

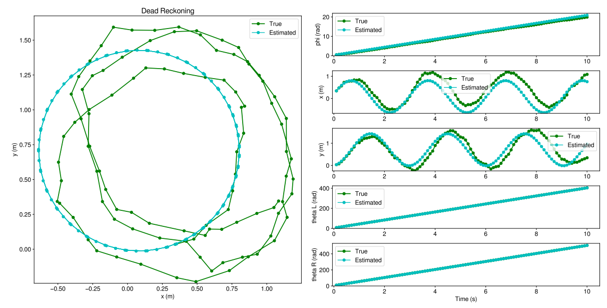

Fig. 1: TurtleBot state estimation using dead reckoning. The estimator initially tracks the true trajectory but accumulates error over time.

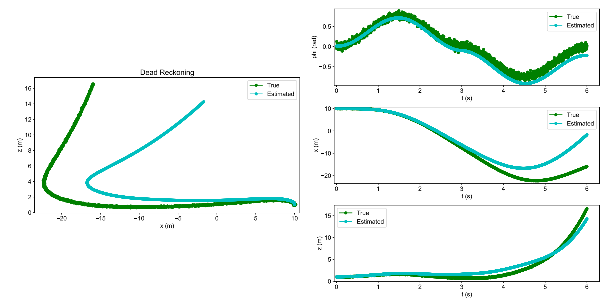

Fig. 2: Quadrotor state estimation using dead reckoning. While the overall motion is captured, error accumulation is quite visible.

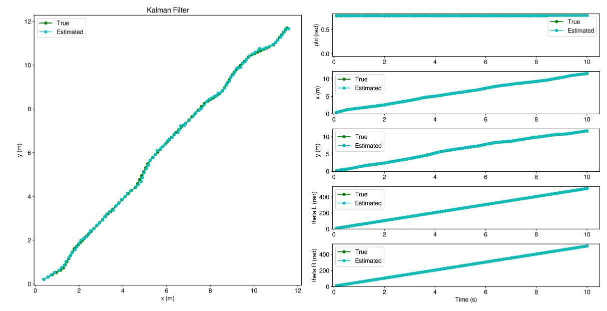

Fig. 3: TurtleBot state estimation using the Kalman filter improves accuracy by enabling the estimator to closely track the true state of the robot.

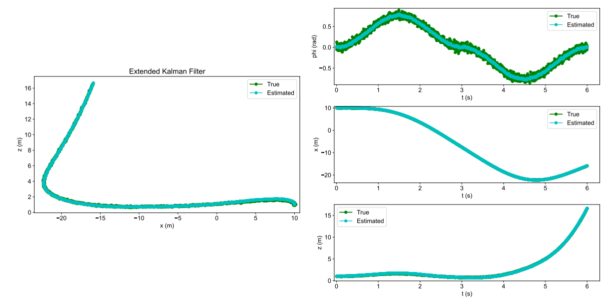

Fig. 4: Quadrotor state estimation using the Extended Kalman filter greatly improves accuracy, achieving nearly perfect estimation.

TABLE I: Quantitative Measurements on Estimation Accuracy and Average Per-Step Computational Running Time.

Estimator	Estimation Error (m)	Average Running Time (ms)
TurtleBot Dead Reckoning	37.30	0.05
TurtleBot Kalman Filter	4.33	0.21
Quadrotor Dead Reckoning	9704.15	0.01
Quadrotor Extended Kalman Filter	213.68	0.05

1) Estimation Error: To quantitatively measure the estimation error of the estimators, we implemented a sum of absolute errors that computes the Euclidean distance between the true state and the estimated state at each timestep. The measurement considers only the x and z coordinates, disregarding orientation. As shown in Table I, the Kalman filter for the TurtleBot achieves nearly an order of magnitude better performance than dead reckoning. Similarly, for the Quadrotor, using the EKF improves performance by more than 45 times compared to dead reckoning. Note that since the number of timesteps in the TurtleBot simulation differs from that in the Quadrotor simulation, the two systems are not directly comparable; however, we can still quantify the performance increase when transitioning from dead reckoning to one of the Kalman filters.

2) Per-step Running Time: Table I also showcases the average running time for each step in the estimators. Using dead reckoning estimation on the TurtleBot yielded an average per-step running time of about 0.05 ms. Transitioning to the Kalman filter increased the running time to about 0.21 ms. For the Quadrotor, the average per-step running time using dead reckoning estimation was about 0.01 ms. Again, transitioning to using the EKF increased the running time, however only up to 0.05 ms—the same as using dead reckoning on the TurtleBot. This could be due to the calculations being more expensive for the TurtleBot than for the Quadrotor, or from our implementations varying between the different estimators. Figures 5, 6, 7, and 8 plot the per-step running time for dead reckoning on the TurtleBot and Quadrotor, the Kalman filter for the TurtleBot, and the EKF for the Quadrotor. Interestingly, when using the EKF on the Quadrotor as shown in Figure 8, the average per-step running time drops from around 0.1 ms to about 0.025 ms during the last two-thirds of the trajectory calculation.

IV. Discussion

A. Dead Reckoning

As illustrated in Figures 1 and 2, dead reckoning accurately tracks the initial motion of both the TurtleBot and the Quadrotor. However, because it relies solely on integrating the motion model without any corrective feedback, small modeling errors and process noise accumulate over time, leading to noticeable drift. This drift becomes especially pronounced for the Quadrotor, whose stronger dependence on accelerations exacerbate error growth compared to the TurtleBot. Dead reckoning is computationally efficient and straightforward, making it suitable for short-term predictions or low-noise scenarios. To mitigate its inherent drift, one could refine the system model to account for effects like wheel slip or aerodynamic disturbances.

B. Kalman Filter

Figure 3 shows that the Kalman filter, while being more computationally expensive, significantly reduces drift compared to dead reckoning. By fusing sensor measurements with the prediction step, the filter continuously corrects the state estimates, maintaining a tighter match with the ground truth. This improvement does come with the added complexity of tuning the noise covariances Q and R. Once well-tuned, however, the Kalman filter provides robust, long-term accuracy, particularly for systems with predominantly linear dynamics like the TurtleBot. For further improvements, adaptive methods that automatically adjust Q and R based on the observed innovation could be implemented, ensuring the filter remains robust even if the noise characteristics change over time.

C. Extended Kalman Filter (EKF)

The Extended Kalman Filter (EKF) addresses nonlinear dynamics, making it particularly effective for systems such as the Quadrotor. It does so by linearizing the nonlinear state equations around the current state estimate. This approach works well for reducing drift in scenarios where the nonlinearity can be adequately approximated without losing significant information. However, the EKF's reliance on a local linearization introduces limitations; if the state deviates significantly from the linearization point or if the system exhibits strong nonlinear behavior, the approximation may become inaccurate, leading to estimation errors. To improve the EKF's performance, one could reduce the update interval to keep the state closer to the linearization point or apply iterative linearization techniques such as the iterated EKF.

References

[1] F. Wu, N. Rahmanian, and K. El-Refai, *EECS C106B: Project 3 - State Estimation*. University of California, Berkeley, Spring 2024.

[2] C. Jiang, Z. Wang, S. Tan, and H. Liang, "A New Adaptive Noise Covariance Matrices Estimation and Filtering Method: Application to Multi-Object Tracking," arXiv preprint, 2021. Available: https://doi.org/10.48550/arXiv.2112.12082.

Appendix A
GitHub Repository

The source code this project are available on GitHub. Any updates and modifications will be reflected in the repository.

Repository Link:
https://github.com/ucb-ee106-classrooms/project-3-404error_bros_kalman_yehaw

Appendix B
Supplementary Figures

This section presents additional performance plots showing the per-step running time for the different estimators on both the TurtleBot and the Quadrotor.

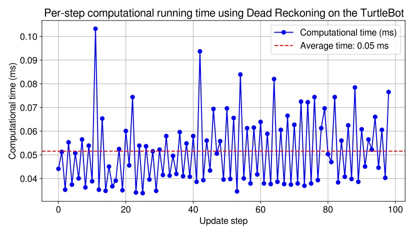

Fig. 5: Running time using dead reckoning on the TurtleBot. The average running time was about **0.05** ms

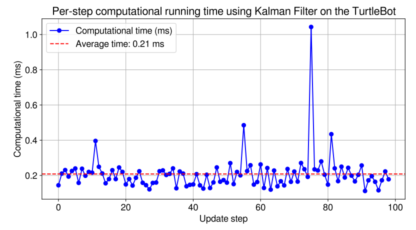

Fig. 6: Running time using Kalman filter on the TurtleBot. The average running time was about **0.21** ms

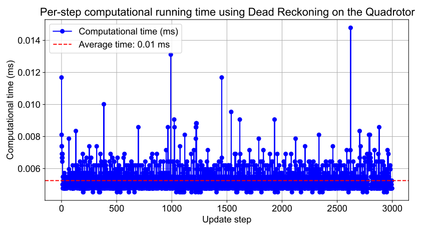

Fig. 7: Running time using dead reckoning on the Quadrotor. The average running time was about **0.01** ms

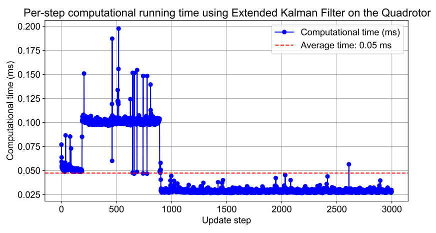

Fig. 8: Running Time using extended Kalman filter on the Quadrotor. The average running time was about **0.05** ms