Genetic Kalman Filtering: An Approach to State Estimation Optimization with Unknown System Dynamics

Daniel Morton
Stanford University
Stanford, CA, USA
dmorton@stanford.edu

***Abstract*—Various versions of the Kalman filter and optimization techniques for state estimation have been researched extensively for decades, with some exploring how to perform state estimation when certain parameters of the system are unknown. However, most of these approaches have focused on noise estimation when the process and measurement noise matrices (Q and R) are unknown — and not in the case where the dynamics matrix (A) is unknown. Therefore, this paper proposes the use of genetic algorithms to solve this problem of unknown dynamics, by initializing a population of chromosomes representing possible contenders for the flattened dynamics matrix, and then iteratively filtering and applying selection, crossover, and mutation to the population. With each iteration, we converge to a dynamics matrix (and corresponding filtered state estimate trajectory) which maximizes the measurement likelihood, a proxy for evaluating the quality of the filtered trajectory when the ground-truth trajectory is unknown. This model is applied to two linear systems: a holonomic robot and a spring-mass-damper system, and through these systems, we see that this technique can produce a Kalman filter that does track the ground truth trajectory, but convergence to the true A matrix values is not guaranteed.**

I. Nomenclature

A	=	Dynamics matrix
B	=	Controls matrix
C	=	Measurement matrix
Q	=	Process noise covariance
R	=	Measurement noise covariance
f	=	Dynamics function
g	=	Measurement function
Σ_t	=	State covariance
μ_t	=	State estimate
y_t	=	Measurement
u_t	=	Control

II. Introduction

Since Kalman's original introduction of his filter in 1960 [1], there have been numerous different filtering methods introduced which build on the original Kalman Filter (KF)'s capabilities. The Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF) [2], for instance, aim to improve the filter's performance on nonlinear systems by either linearizing about the belief state, or propagating a set of sigma points through a nonlinear function to gain a new estimate of the mean and covariance [3]. Just in these few examples, we see that state estimation is inherently intertwined with optimization: the linearization process for the EKF is essentially a Gauss-Newton optimization step, and the UKF follows the same process as generalized pattern search or Hooke-Jeeves. Even more examples can be found when considering the similarities between the KF's Gaussian distributions and optimization techniques which also rely on Gaussian assumptions, population methods and particle filtering, and more.

An interesting challenge in filtering is how to handle uncertainty in the model parameters, and the most common example of this is noise estimation. While the Kalman Filter and its derivatives can handle zero-mean additive Gaussian white noise on the measurements and process noise, the problem must be reformulated if the distribution of this noise is unknown. Recent research has looked into ways to determine the magnitudes of these noise distributions, including using covariance matching [4], Bayesian optimization [5], fuzzy-based evolutionary algorithms [6], recursive approaches [7], or Innovation-based Adaptive Estimation (IAE) [8].

One noise estimation technique, Multiple Model Adaptive Estimation (MMAE) [8] inspired the work in this paper — this technique handles a bank of several different models and computes the Bayesian probability of each being the true system model. This works well for noise, but breaks down when the system dynamics are fully unknown because it relies on the assumption that one model is ”correct” out of the population.

Therefore, if we have a population of filters with unknown system dynamics, how can we converge to the correct model, using a mix of state estimation, system identification [9], and optimization?

Genetic algorithms may serve as a possible solution to this problem. This is a population-based optimization method that optimizes a collection of design points based on biological evolution techniques. Three main processes — selection, crossover, and mutation — are applied at each iteration to select a pair of high-fitness parent chromosomes, combine their genetic information, and pass this on to a new child chromosome (which may have some noise or mutation applied to its values). Here, a chromosome refers to a vector of design points, genetic info refers to the specific values of each design point, and fitness is inversely related to the evaluation function. Since we seek to minimize the evaluation function, chromosomes in the population with a lower evaluation function have a higher fitness, and are more likely to pass along their genetic info to the next generation. [10]

So, in this paper, we apply genetic algorithms to linear-system state estimation with unknown dynamics, maintaining a population of candidate A matrices in their flattened state as our chromosomes, and repeatedly filtering, evaluating, and updating the values in the matrices according to these genetic algorithm processes.

III. Approach

Algorithm 1 Genetic Kalman Filter Algorithm

```
Initialize parameters, dynamics, controls, noise
Simulate, get measurement history
Initialize population
for K iterations do
    for Each chromosome do
        A ← to_A_matrix(chromosome)
        Trajectory ← KF($\mu_t$, $\sigma_t$, $u$, $y$, $Q$, $R$, $f$, $g$, $A$, $C$)
    end for
    Initialize Evaluations ← []
    for Each filtered trajectory do
        Metric ← 0
        for All time t do
            Metric += Mahalanobis(Measured, Expected)
        end for
        Metric += L1_regularize(chromosome) * Scaling
        Evaluations.append(Metric)
    end for
    Parents ← Selection(Population, Evaluations)
    Children ← Crossover(Parents)
    New_Population ← Mutation(Children)
end for
Return best filter from the population
```

Algorithm 1 overviews the main process for this project: At each iteration, we extract a chromosome from the population, convert this into a square matrix, and then run the Kalman filter using this matrix as A. All other matrices in the Kalman filter (B, C, Q, R) are predefined within the system models initialized at the beginning of the algorithm. The Kalman filter is evaluated using the same μ_0 and Σ_0 for each chromosome in the population, so that we can compare the performance across all chromosomes without one candidate performing artificially well due to a "good start". We store each of the trajectories produced by the different filters, and then run the evaluation metric on each of these. For each timestep in the trajectory, we calculate the cumulative Mahalanobis distance for the measurement likelihood, and then add on an L1 regularization term at the end. Once these evaluation values are calculated, we know the relative performance of each chromosome in the population, and can run the selection, crossover, and mutation functions from the genetic algorithm to select the population for the next iteration. After K iterations, we exit the main loop and then run a final evaluation process on the last population to extract the final result.

Note that this approach is only applicable to linear, time-invariant systems, so that all entries in the dynamics matrix (A) are constant in time, and there is a fixed set of values (the design point for the flattened A matrix) to which the genetic algorithm can converge. Without this assumption, a more complex genetic algorithm method would need to account for how the A-matrix parameters vary in time and with the current system state, which has not been considered in this paper. This reason is also why we could not apply one of the common techniques for estimating unknown parameters — appending the unknown parameter to the state and modifying the dynamics matrices/Jacobians to account for this. Doing so inherently introduces nonlinearity into the system because the dynamics Jacobian will depend on the current state, requiring an Extended Kalman Filter or another filter that can handle nonlinearity.

A. Linear System Models

1) Holonomic Robot: The holonomic robot system has a state space of $\{x, y, \theta\}$ with a noisy measurement model representing a GPS reading for the (x, y) position, and a compass reading for the heading angle, θ. The holonomic assumption assumes that x, y, and θ are all independently controllable via v_x, v_y, and v_θ, which differs from the common Dubins car system, where x and y are functions of the forward velocity and the heading angle θ. The Dubins car system is therefore inherently nonlinear because of the $\sin\theta$ and $\cos\theta$ terms introduced by this dependence, and was not considered for this project. For testing purposes, we applied the following arbitrary control policy:

$$u(t) = \begin{bmatrix} v_{x,avg}|(\sin(t))| \\ v_{y,avg}\cos(t) \\ \phi_{avg}\sin(t) \end{bmatrix}$$

The average linear and rotational velocities are also arbitrary and were set to 1 m/s, 2 m/s, and 0.1 rad/s respectively. The simulation was run for 20 seconds with a discrete timestep dt equal to 0.1 s.

2) Spring-Mass-Damper: The spring-mass-damper system has a state space of $\{x, v\}$, with measurements of the position of the mass x only, and a driving force $f(t)$ which acts as the sole control input to the system. The model follows the discretized version of the standard second-order differential equation, $m\ddot{x} + c\dot{x} + kx = f(t)$. The system parameters for the spring constant, damping coefficient, and mass were set to 5 N/m, 1 kg/s, and 10 kg, respectively. These parameters were only known by the simulation.

For testing, we applied the following arbitrary control policy:

$$f(t) = F\sin(\omega t)$$

Here, we set F to be 2 N, and ω as 1.5x the natural frequency of the system, $\sqrt{k/m}$. The simulation was run for 10 seconds with the same discrete timestep dt as before, 0.1 s.

B. Evaluation Function

The overarching goal of a state estimator is to match the ground-truth data as closely as possible, given a set of noisy measurements. Since these state estimators do not have access to the ground truth, we therefore cannot use the ground truth as a comparison in the evaluation function, or else there would not be a point to using a state estimator in the first place. Instead, we rely on the cumulative measurement likelihood to evaluate the quality of the result from the genetic algorithm, via the Mahalanobis distance between the measurement received at a certain timestep, and the predicted measurement based on the current state estimate μ_t. Since the Kalman filter relies on a Gaussian assumption for the measurement and state models, the Mahalanobis distance is an effective measure of how far a measurement is from the mean of the distribution, given the multidimensional covariance of the distribution.

The Mahalanobis distance for the measurement likelihood can be written as follows, where y is the measurement received, $g(\mu_t)$ is the expected measurement function given μ_t, C is the measurement matrix, Σ_t is the state covariance matrix, and R is the measurement noise matrix:

$$(y - g(\mu_t))(C\Sigma_t C^T + R)^{-1}(y - g(\mu_t))$$

The probability density function for a multivariate Gaussian is uniquely defined by this Mahalanobis distance. In the general case:

$$N(\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} e^{\frac{-1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

$$N(\mu, \Sigma) \propto e^{\text{Mahalanobis Distance}}$$

Assuming conditional independence on measurements given the state, we can evaluate the cumulative measurement likelihood as the product of the measurement likelihoods at every timestep. However, this is numerically unstable for large timesteps, as the product of many small measurement likelihoods can result in very small evaluation values, quickly approaching numerical precision. Therefore, instead of maximizing the measurement likelihood, we can instead minimize the negative log-likelihood, which is equivalent to taking the cumulative sum of the Mahalanobis distances at every timestep.

This metric must be calculated for every chromosome in the GA population at every generation, which requires re-running the Kalman Filter with the new A matrix for every evaluation.

As will be shown later, simply using the Mahalanobis distances alone will work, but will produce a filter that is highly sensitive to noise, with higher-magnitude entries in the A matrix. To reduce the sensitivity to noise (effectively like reducing the gains within a proportional controller), we introduce an additional L1 regularization term into the evaluation function, which sums the magnitudes of all entries within the chromosomes and considers this as an additive penalty. The relative scaling factor on this L1 term is a hyperparameter that can be set from 0 (no L1 regularization) to any positive value.

IV. Results

A. Hyperparameters

The most important hyperparameters for this model include which selection, crossover, and mutation methods to use — and for most of the testing process, we applied Roulette selection, Interpolation crossover, and Gaussian mutation. Roulette was chosen because it eliminates the need to specify the number of chromosomes to choose out of the population (as with truncation and tournament selection), and fewer hyperparameters to tune is always appreciated. With crossover, interpolation is well-suited for the real-valued chromosomes in this model. Also, the alternative methods of crossover (single-point, two-point, or uniform) occasionally seemed to result in large spikes in the loss function convergence plot when a poor crossover was made, which was not an issue with interpolation. Lastly, Gaussian mutation is the main mutation method for real-valued chromosomes, so there was only one reasonable choice here.

After running the models multiple times with varying parameters (such as the selection/crossover/mutation functions, the population size, number of iterations, and the regularization scaling), the overall performance of the filter seemed fairly insensitive to most hyperparameter changes, with it being able to track the true trajectory in nearly all instances. The most important parameter by far was the regularization factor though, as this had a significant effect on smoothing the filtered trajectory.

However, though the filtered trajectory was relatively insensitive to hyperparameter changes, the exact values in the chromosomes could see some large changes. For instance, small changes in the regularization scaling could change an entry in the A matrix from 3 to 1, but the effect on the actual filter and how well it tracked the ground truth was not very noticeable

B. Holonomic Robot

In Figure 1, we can see decent tracking in the genetic algorithm result in (A). For the X, Y, and XY trajectory plots, this is nearly as good as the Kalman filter result with the exactly-known A matrix (B), even despite a small amount of tracking error in θ.

The difference between the genetic algorithm with and without regularization is quite noticeable between (A) and (C) — the no-regularization result gives a tracking estimate which is significantly more jagged. While the confidence interval does maintain the ground truth at nearly all times (which differs from the regularized result), the tracking is so poor that this result should not be used.

The evaluation function (D) also plateaus after a relatively small number of iterations (about 10), and from there, additional iterations do not seem to result in a better filter.

Figure 2 shows that the Mahalanobis evaluation does an effective job at selecting the best-tracking filter from the population, even when the ground truth is not known. The alternatives shown in red are much worse at tracking the

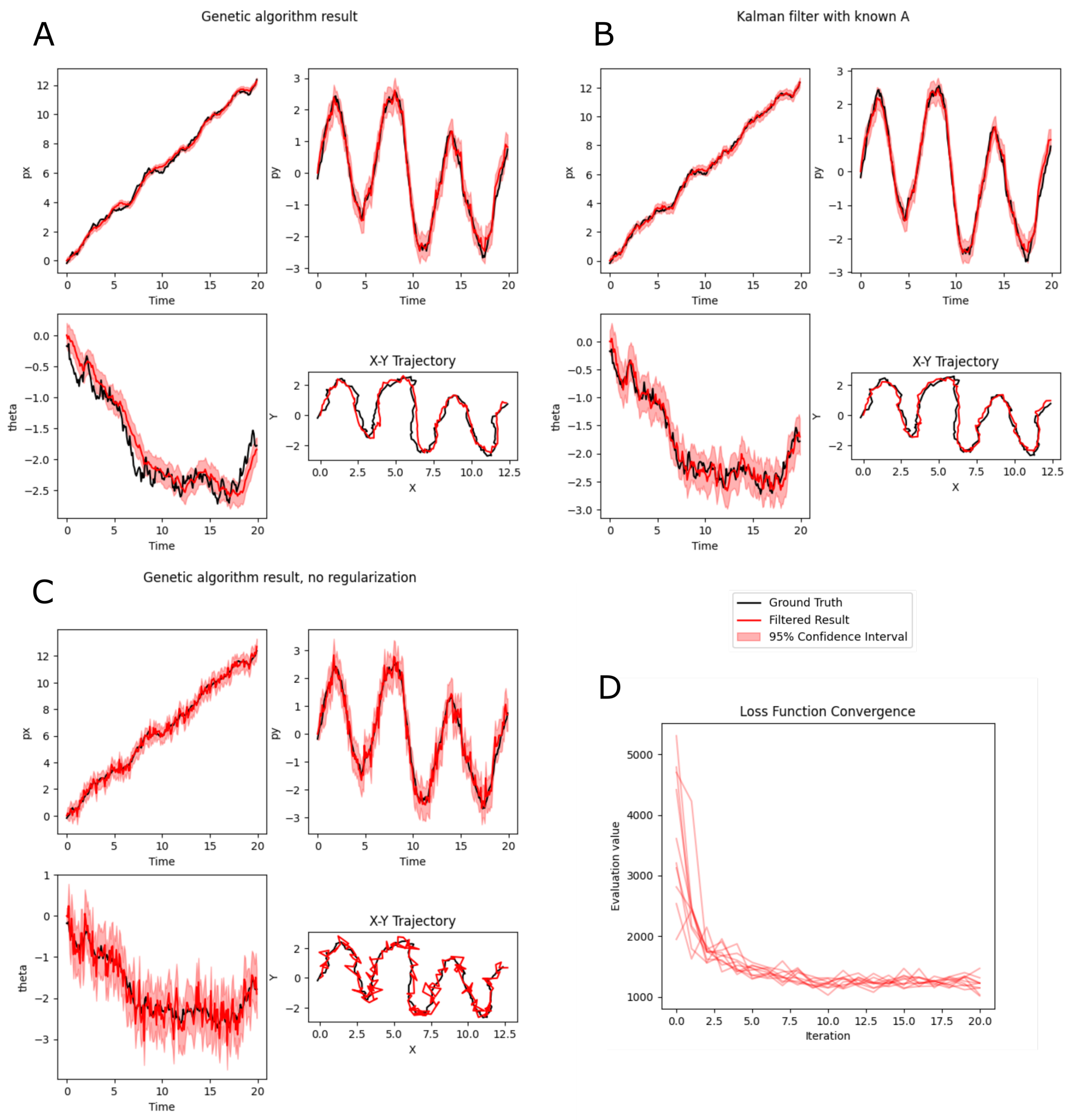

Fig. 1. Genetic algorithm results for the holonomic robot. (A) The results from the best filter as determined by the genetic algorithm; (B) The filtering result when a Kalman filter with the exactly known value of A is used; (C) The results of the genetic algorithm when no regularization is included; (D) The evaluation function values for each of the chromosomes over the iterations of the genetic algorithm

ground truth than the best filter in blue, and even though the best filter is still a bit jagged and has some error, this is quite good for just the first generation of the algorithm.

Figure 3 puts the actual performance of this algorithm in perspective. While the filters produced via the genetic algorithm can yield decent tracking results, this is not to say that the entries of the A matrix converged to their exact values. In fact, only about 6/9 of the entries in the matrix approached the true value, and this convergence is not particularly strong.

The best result in this figure is bit 4, because the final result was quite close to the true value of 1, though there was some instability in the value over the iterations. The other bits that converged all approached 0, which is correct, but there is a chance this is attributed to the L1 regularization more than

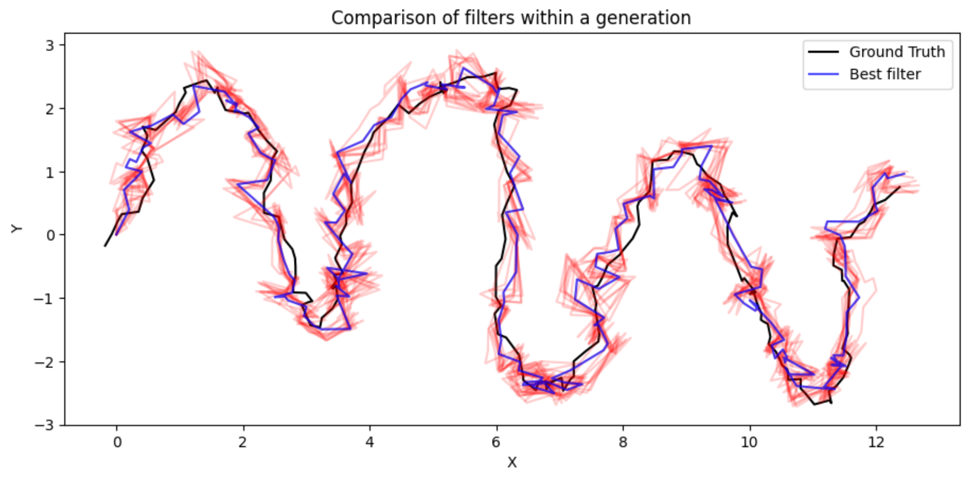

Fig. 2. Comparing the best filter within the first generation of the genetic algorithm to other candidates in the same generation.

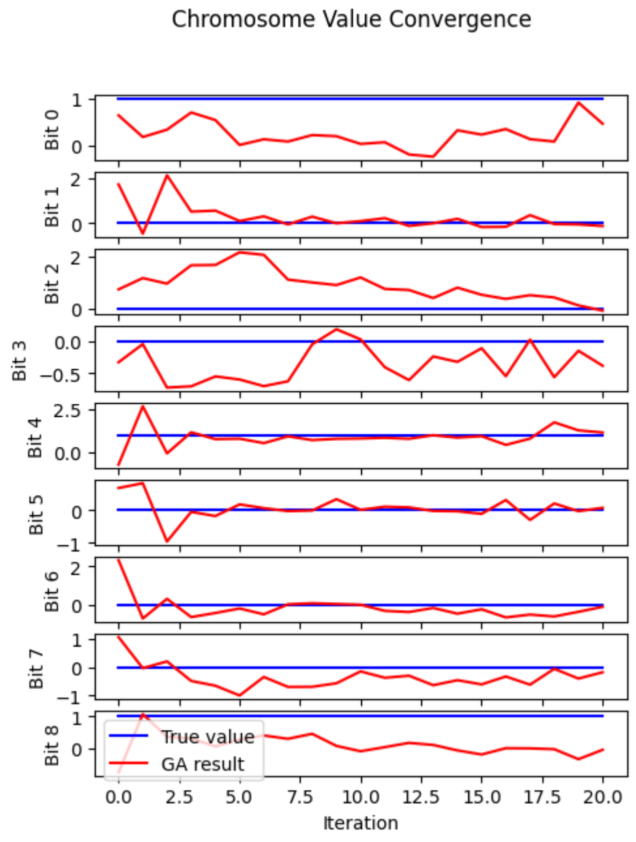

Fig. 3. Evolution of the values within the best chromosome over time.

the Mahalanobis distance metric. For bits 0 and 8, these were supposed to equal 1, and while they came close on occasion, they did not converge to that value.

$$\text{Optimization result: } \begin{bmatrix} 0.459 & -0.120 & -0.082 \\ -0.380 & 1.154 & 0.069 \\ -0.081 & -0.177 & -0.036 \end{bmatrix}$$

$$\text{True A matrix: } \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

C. *Spring-Mass-Damper*

The results from the spring-mass-damper system were very similar to that of the holonomic robot — the result from the genetic algorithm was able to track the ground truth with only minimal error, the L1 normalization was equally as beneficial, and yet the values of the final chromosome still were not able to match the true values. The only major differences of note were as follows:

(1) Whereas with the holonomic robot we had measurements of all elements of the state (x, y, θ), for the spring-mass-damper we only measured the position of the mass, and not the velocity. This seemed to have little effect on the genetic algorithm result, however, as it was still able to effectively track the system velocity.

(2) When comparing the output of the genetic algorithm to other non-optimized candidate filters, many of these other filters produced trajectories which quickly became unbounded, leading to the filter estimate running off to positive or negative infinity, and extremely high objective function values. It is great to see that the genetic algorithm was able to maintain bounded dynamics in the result, even when these unbounded filters existed within the original population.

$$\text{Optimization result: } \begin{bmatrix} 1.705 & -1.222 \\ 0.045 & 0.283 \end{bmatrix}$$

$$\text{True A matrix: } \begin{bmatrix} 1 & dt \\ -kdt/m & 1 - cdt/m \end{bmatrix} = \begin{bmatrix} 1 & 0.1 \\ -0.05 & 0.99 \end{bmatrix}$$

V. Conclusion

When strictly considering the performance of the Kalman filter, genetic algorithms are successful at taking an arbitrary set of unknown dynamics matrices and iteratively generating a result that tracks the ground-truth trajectory. However, this method should not be used as a means of system identification, as the exact values in the final result do not converge to the true dynamics matrix values. This is likely because the Kalman filter can still perform well even with some model mismatch, so it is difficult to distinguish the performance differences between a mismatched model and the true model.

Future work in this area should look into defining a more detailed evaluation function that may be able to incorporate more knowledge about the system, to help improve the convergence to the true values. This may involve adding additional penalties and constraints to prevent unstable systems or systems that can become unbounded. Additionally, more testing should be performed to determine the best use of the regularization term — it is possible that with a more refined evaluation function, this term may not be needed, and the difference between L1 and L2 regularization has not yet been studied.

Additional ideas include training this on multiple simulations in parallel, so that the genetic algorithm does not overfit the result to this one particular simulated trajectory. This work might also be extended to the other matrices in the system, such as the controls matrix, B, or the noise matrices, Q and R. Applying this to C may be difficult due to how essential this matrix is in determining the Mahalanobis distance, but there could be potential there. For Q and R, most likely, some of the other methods for noise prediction seen in the previous work will outperform this one.

Appendix

The code for the project and additional plots for the spring-mass-damper system are available at the project's Github page, https://github.com/danielpmorton/Genetic_Kalman_Filtering

References

[1] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering, Transactions of the ASME*, vol. 82, no. 1, pp. 35–45, 1960.

[2] E. A. Wan and R. van der Merwe, "The Unscented Kalman Filter for Nonlinear Estimation," 2000.

[3] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 1 ed., 2005.

[4] S. D. Brown and S. C. Rutan, "Adaptive Kalman Filtering.," *Journal of Research of the National Bureau of Standards (United States)*, vol. 90, no. 6, pp. 403–407, 1985.

[5] Z. Chen, N. Ahmed, S. Julier, and C. Heckman, "Kalman Filter Tuning with Bayesian Optimization," pp. 1–12, 2019.

[6] R. M. Asl, R. Palm, H. Wu, and H. Handroos, "Fuzzy-Based Parameter Optimization of Adaptive Unscented Kalman Filter: Methodology and Experimental Validation," *IEEE Access*, vol. 8, pp. 54887–54904, 2020.

[7] H. Heffes, "The Effect of Erroneous Models on the Kalman Filter Response," *IEEE Transactions on Automatic Control*, vol. 11, no. 3, pp. 541–543, 1966.

[8] M. Karasalo and X. Hu, "An optimization approach to adaptive Kalman filtering," *Automatica*, vol. 47, no. 8, pp. 1785–1793, 2011.

[9] L. Ljung, "System Identification," in *Signal Analysis and Prediction*, pp. 163–173, New York: Springer, 1998.

[10] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for Optimization*. Cambridge, MA: MIT Press, 2022.