Optimization Techniques for State Estimation: EKF Descent Methods and KF Genetic Algorithms

Daniel Morton
dmorton@stanford.edu

***Abstract*—State estimation and optimization are two inherently linked fields of research and engineering — many state estimation techniques either directly apply optimization algorithms, or are highly similar in their approach. For instance, the iterated EKF applies a series of Gauss-Newton steps to converge to a better linearization point of the inherently nonlinear EKF objective function. But, the Gauss-Newton method is only one out of many descent methods, which raises the question of what happens when other descent methods are applied to the function? Also of interest is what happens when the parameters in the system are not fully known — previous research has commonly looked into noise estimation, but not as much when the dynamics matrix (A) is fully unknown. Therefore, this paper covers two explorations of merging optimization and state estimation: first, a study and comparison of different descent methods for the EKF, and second, a study of applying genetic algorithms to iteratively construct a Kalman filter when the dynamics matrix is unknown. Through an example with a Dubins car system, we show that while other descent methods (Gradient Descent, Adagrad, and Nesterov Momentum) can occasionally produce a better result than the iEKF, the convergence rate, reliability, and speed of the Gauss-Newton step in the iEKF is unparalleled. And, using a holonomic robot system, we show that genetic algorithms can produce a Kalman filter that does track the ground truth trajectory, but convergence to the true A matrix values is not guaranteed.**

I. 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 (and likewise, the iterated 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.

Focusing on the EKF specifically, we can raise a number of questions about this linearization process and how to best handle finding the optimal point for evaluating the Jacobians. Namely, when we have a nonlinear model, does the single Gauss-Newton step for linearization always lead to a good estimate, or will iterative descent methods improve the solution? Conversely, will these iterative methods ever decrease the performance of the EKF? And, given that there is a broad range of different descent methods, how does the performance compare for each of these? The first half of this report explores these questions, focusing on how different first-order descent methods (Gradient Descent, Adagrad [4], and Nesterov Momentum [5]) compare with the EKF and the iEKF.

Another 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 [6], Bayesian optimization [7], fuzzy-based evolutionary algorithms [8], recursive approaches [9], or Innovation-based Adaptive Estimation (IAE) [10].

One noise estimation technique, Multiple Model Adaptive Estimation (MMAE) [10] inspired the genetic algorithms 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 [11], and optimization? Genetic algorithms may serve as the answer to this problem, which has been explored in the second part of this paper.

II. Descent Methods for the EKF

A. System Overview: Dubins Car

Here, we evaluate the performance of each of the descent methods on a Dubins car (an inherently nonlinear dynamic system requiring the use of the EKF). To further increase the nonlinearity, we use a range-and-bearing measurement model to four fixed map markers, with the angles from the bearing dictating this nonlinearity. Originally, a GPS measurement model was used, but this is a linear measurement model, and resulted in even less of a noticeable effect from the descent methods. Therefore, we have a state $\in \mathbb{R}^3$: $\{x, y, \theta\}$ and a measurement model $\in \mathbb{R}^{12}$ (where each range measurement $\in \mathbb{R}^1$ and each bearing measurement is a unit vector $\in \mathbb{R}^2$).

For the noise, we have a magnitude of 0.1 on each diagonal element in the noise covariance matrix R, and likewise, 0.01

on the diagonals of Q. And for control, we apply a constant forward velocity v, and a sinusoidally-varying rotation rate ϕ.

Evaluating the EKF on this system leads to the following result (Figure 1). As we can see, the EKF generally does a decent job at fitting the filtered trajectory to the ground truth, with a relatively low covariance. This result is not perfect, however, and we will use this as a baseline to compare the other models against and see if there are any noticeable improvements.

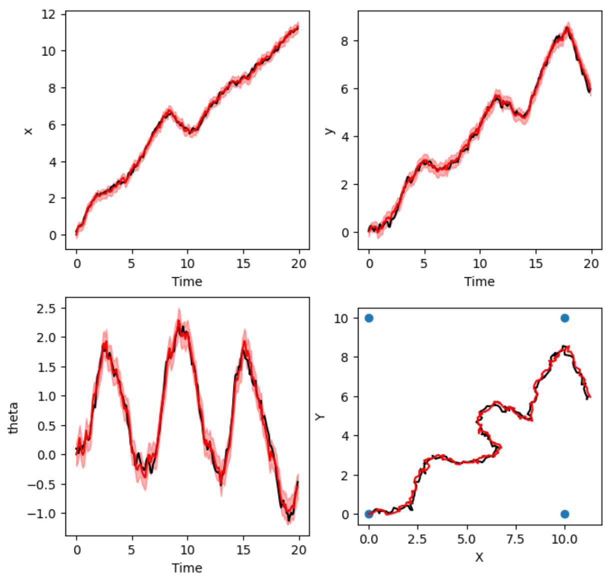

Fig. 1. The baseline system for the descent methods test, filtered using the EKF

B. Approach

In this section, we focus primarily on three first-order descent methods for determining the linearization point for the EKF, and compare it to the results from standard methods, the EKF and the iEKF. First, gradient descent is one of the simplest optimization descent methods (taking a step size in the direction of the current gradient), and will therefore serve as a good reference point for the other algorithms. Second, Nesterov Momentum gradually accelerates the descent by adding a momentum term, and "looks ahead" to the future position to avoid overshooting the minimum. And third, Adagrad adapts the learning rate to each component of the design vector, as opposed to the constant rate from the other methods. [12] [4] [5]

For this project, the implementations of Adagrad and Nesterov Momentum are vanilla, but the gradient descent uses two small modifications — normalization of the gradient prior to evaluating the step, and a quadratically-decaying step size. Without the gradient normalization, the system was quite unstable — a high gradient at the start could lead to a very poor initial linearization point, leading to further compounded error later. And without the decaying step size, the gradient descent had a difficult time converging to the 1e-3 level within the 20-iteration limit, due to oscillation about the minimum. The quadratic decay was experimentally observed to out-perform a more simple linear decay.

C. Performance

To evaluate the relative differences in performance between each method, we will use the cumulative sum of Mahalanobis distances between the filter estimate and the ground-truth value at each timestep, parameterized by the filter covariance. This will give a reliable measure of how well the filter tracks the ground truth, similar to a cumulative error (lower values are better).

$$Performance = \sum_{t=0}^{T} (\mu_t - x_t)\, \Sigma_t^{-1} (\mu_t - x_t)$$

The following Table I summarizes this performance on an example trajectory, across each method. As expected, iEKF outperforms the EKF by selecting a better point for the linearization. The more interesting result is in each of the other descent methods — all of these outperform the EKF, but only Adagrad and Nesterov Momentum outperform the iEKF as well (and only by a small amount).

TABLE I
DESCENT METHOD PERFORMANCE METRIC COMPARISON

	Metric	% Improvement over EKF	% Improvement over iEKF
EKF	543.24	0.00%	-4.85%
iEKF	518.11	4.62%	0.00%
Gradient Descent	523.27	3.67%	-1.00%
Adagrad	517.38	4.76%	0.14%
Nesterov Momentum	515.41	5.12%	0.52%

We can also plot the trajectories from each method to see how these compare to each other. However, the differences in performance are so small that the trajectories are imperceptibly different from the standard EKF result from Figure 1. So instead, to visualize the differences in the filter, we focus on the errors with respect to the ground truth (Figure 2) and the difference with respect to the standard EKF result (Figure 3).

If we subtract out the ground truth from the history of each filtered value (Figure 2), we can directly observe the error at each timestep. However, since each filter performs very similarly, any difference between the methods is not very visible here, save for a few instances where gradient descent leads to slightly higher error. Alternatively, if we subtract away the results from the standard EKF (Figure 3), we can more clearly observe how these iterative descent methods affect the filtered result. Here, gradient descent deviates from the EKF the most, whereas iEKF and Nesterov Momentum both track very closely with each other, and these two also had the least difference to the EKF result.

D. Convergence

The proximity of the filtered result to the ground truth is only one part of determining how well these methods perform — convergence and runtime are equally important, if not more so. To determine this, we ran each method on the example

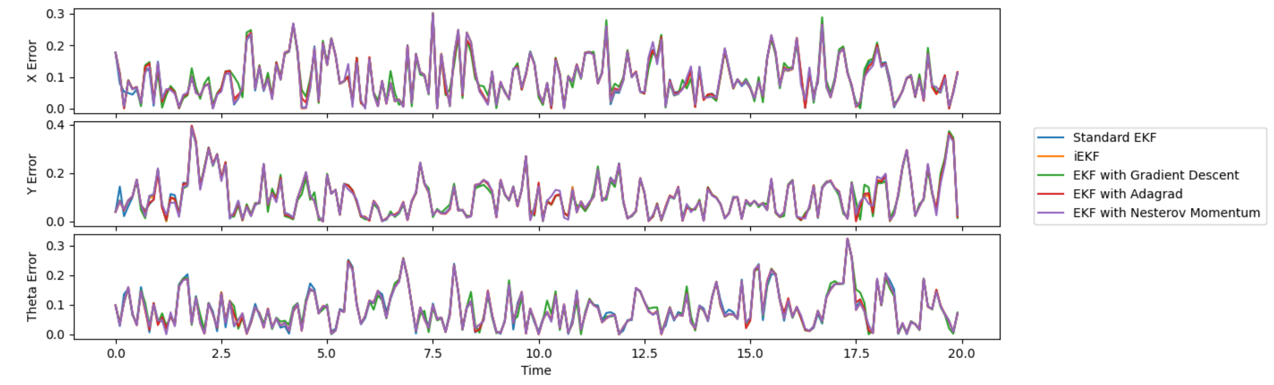

Fig. 2. Error in filter output with respect to the ground truth

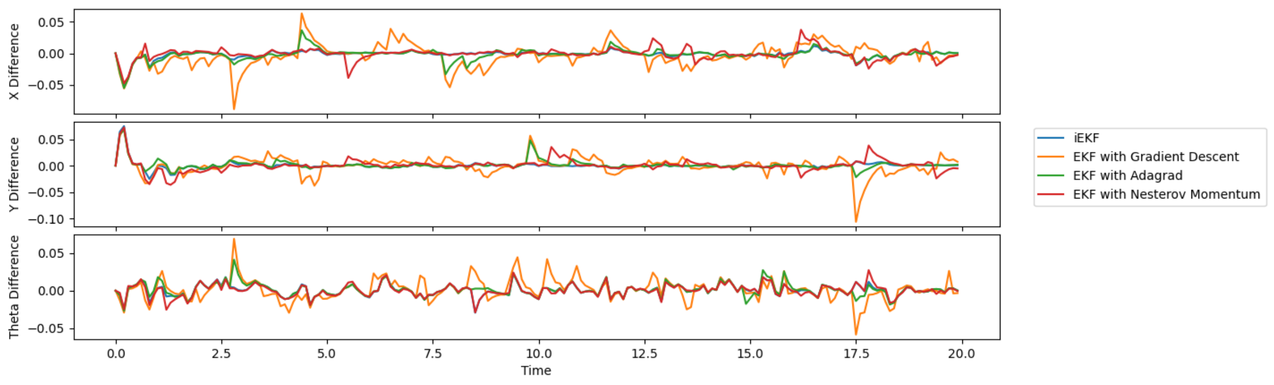

Fig. 3. Difference in filter output with respect to the EKF

TABLE II
CONVERGENCE AND TIMING COMPARISON FOR EACH DESCENT METHOD

	Convergence %	Average iterations until convergence	Std. dev.	Average residual when not converged	Std. dev	Average time per iteration	Std.dev
iEKF	100.00%	3.45	0.74	N/A	N/A	0.0013	0.0005
Gradient Descent	60.30%	14.43	3.53	0.0017	0.0010	0.0141	0.0058
Adagrad	65.33%	10.79	4.31	0.0095	0.0135	0.0097	0.0047
Nesterov	69.35%	12.91	3.81	0.0025	0.0039	0.0126	0.0047

TABLE III
PERFORMANCE COMPARISON ACROSS DIFFERENT RANDOM SEEDS

	Random Seed											
	0*	1	2	3	4	5	6	7	8	9	10	Average
EKF*	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
iEKF	4.62%	-0.61%	1.31%	0.67%	0.09%	-1.37%	2.07%	1.55%	1.67%	1.92%	0.71%	1.15%
Gradient Descent	3.67%	-2.56%	-1.31%	0.12%	-1.49%	-2.30%	0.50%	0.71%	-0.40%	2.71%	-1.23%	-0.14%
Adagrad	4.76%	-0.71%	0.41%	0.26%	0.37%	-1.64%	1.98%	1.50%	-10.26%	2.00%	-0.19%	-0.14%
Nesterov Momentum	5.12%	-0.54%	0.89%	-1.72%	0.22%	-2.82%	-0.71%	-0.02%	1.31%	4.07%	-1.03%	0.43%

trajectory, and then averaged the convergence and timing statistics over every timestep. Based on these results in Table II, we see that iEKF converges in all cases, typically in just a few iterations. Also, each iteration is very fast to compute, nearly an order of magnitude faster than the other methods — resulting in this being by far the most computationally efficient algorithm. Gradient descent reaches the convergence criterion within the allotted 20 iterations only 60% of the time, and therefore performs the worst out of all of these methods, due to a high number of iterations until convergence and a long average time per iteration. Adagrad and Nesterov both result in small improvements over gradient descent, with slightly higher convergence percentages and fewer iterations required for convergence. Adagrad takes less time than gradient descent and Nesterov, but can have a large residual in the cases when it does not converge. This "worst-case performance scenario" is also visible in the random seed (8) evaluation (Table III).

So, overall, this evaluation highlights the effectiveness and

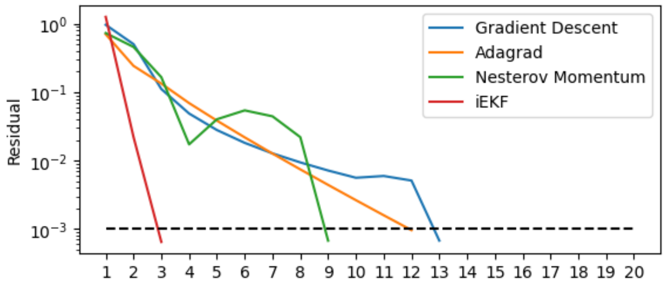

Fig. 4. Convergence comparison for each descent method

efficiency of the Gauss-Newton method within the iEKF. We can also visualize this via an example convergence plot, run during a single timestep of the sample trajectory (Figure 4). While this can vary depending on the state of the system, in general, this plot represents the main trends that typically appear. Firstly, iEKF converges extremely quickly, in only a few iterations. Adagrad and Nesterov generally perform similarly, converging in-between the iEKF (the best case) and gradient descent (the worst performer). Nesterov sometimes has a few places in the plot where the residual increases, but this is something to be expected on occasion with a momentum-based method.

E. Hyperparameters

TABLE IV
SELECTED HYPERPARAMETERS

Gradient Descent Step Size	1.00E-01
Gradient Descent Step Decay	Quadratic
Adagrad Learning Rate	5.00E-02
Nesterov Learning Rate	6.00E-03
Nesterov Momentum Decay	0.88
Max Iterations	20
Epsilon	1.00E-03

Given these results so far, one key question to answer is if this selection of hyperparameters generalizes to other trajectories with the same system. In short — no, not necessarily. The hyperparameters were selected based on the performance of the trajectory corresponding to a random seed of 0, so Table III summarizes the performance when different random seeds were used (which corresponded to entirely new trajectories for the system). Clearly, there is some overfitting to random seed 0, because the benefits of all descent methods decrease significantly for the other examples. However, it is important to note that this decrease in performance was based on the Mahalanobis performance measure, and since these numbers were quite small to begin with for the EKF, a 10% reduction in performance (in the case of Adagrad in random seed 8) only created a nearly imperceptible tracking error in the plot.

Also of interest was how these methods would perform with order-of-magnitude differences in some of the key system parameters, including dt and the entries in Q and R. In general, iEKF and Nesterov stayed very accurate to the ground-truth trajectory and the result from the EKF, typically outperforming the EKF by a few percent. There were a few cases where the iEKF struggled with very large amounts of noise, but overall, performance remained strong. However, gradient descent and Adagrad would occasionally result in a performance metric value which was 2–4x as large as the EKF's value — resulting in an observable increase in tracking error in the plot, but not enough to cause significant issues (the ground truth generally still remained within the 95% confidence interval).

Overall, a commonality between the descent methods we examined (other than the iEKF) were that they were highly sensitive to the hyperparameter selection — particularly the step size / learning rate. With just minor changes to the hyperparameters, the convergence rate for these methods (within the maximum 20 iterations specified) could vary wildly between 10% - 70%, leading to significantly decreased performance if chosen improperly. For instance, reducing the learning rate could slow the convergence enough that the methods reach the max iterations without achieving the specified 1e-3 tolerance. The decaying step method used for gradient descent is also essentially another hyperparameter, and as mentioned previously, running vanilla gradient descent led to instability and/or oscillation about the minimum.

III. GENETIC ALGORITHMS FOR KFS WITH UNKNOWN SYSTEM DYNAMICS

Genetic algorithms are 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 (design points), combine their genetic information, and pass this on to a new child chromosome (which may have some noise or mutation applied to its values). Since we seek to minimize the evaluation function, chromosomes in the population with a lower evaluation function have higher fitness, and are more likely to pass along their genetic info to the next generation. [12]

In this section, 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.

A. System Overview: 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. 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.

B. Approach

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.

C. Evaluation Function

Since we cannot use the ground truth to evaluate the results from the G.A., we instead 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. This 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:

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

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.

D. 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 selection eliminates the need to specify the number of chromosomes to choose out of the population (an additional hyperparameter which would need to be tuned), and interpolation crossover and Gaussian mutation are both well-suited for the real-valued chromosomes in this model.

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, as this had a large 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

E. Results

In Figure 5, 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

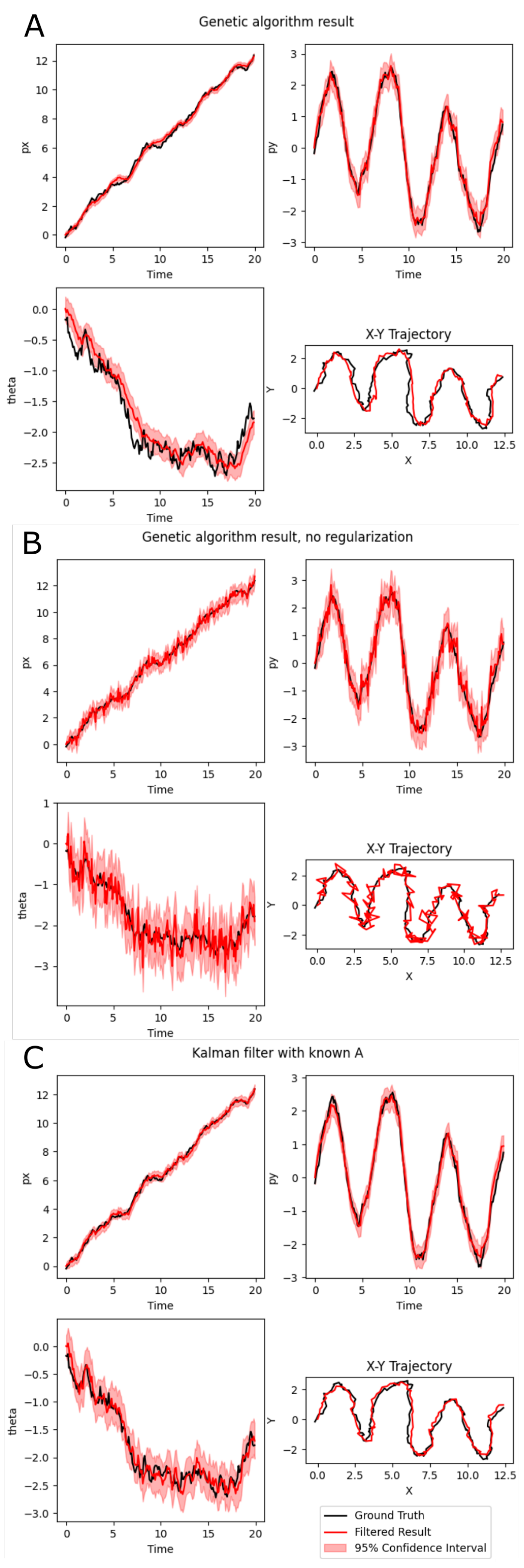

Fig. 5. Genetic algorithm results for the holonomic robot. (A) The results from the best filter as determined by the genetic algorithm; (B) The results of the genetic algorithm when no regularization is included; (C) The filtering result when a Kalman filter with the exactly known value of A is used

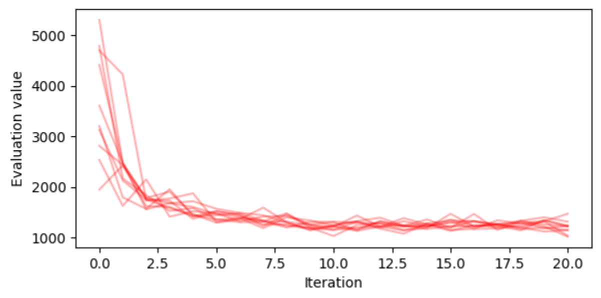

Fig. 6. Loss function convergence for each chromosome in the population

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 (Figure 6) 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.

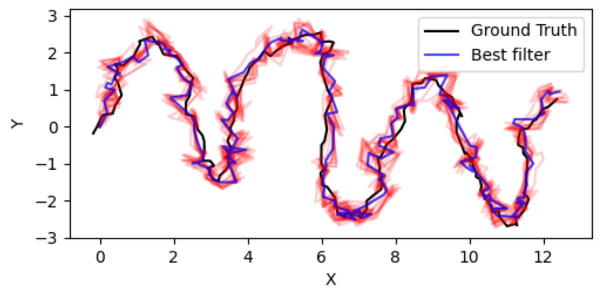

Fig. 7. Comparing the best filter within the first generation of the genetic algorithm to other candidates in the same generation.

Figure 7 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 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.

However, the final result for the A matrix from the genetic algorithm puts the actual performance of the 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 the convergence is not strong. When observed over multiple iterations, these values would still see occasional jumps, indicating that the population of candidate matrices still had large variances.

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

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

IV. Conclusion

From the performance and convergence comparisons, it is unmistakable that the Gauss-Newton descent method in iEKF vastly outperforms the three candidate first-order methods (Gradient Descent, Adagrad, and Nesterov Momentum). It converges reliably, in a short number of iterations, and with a low computation time per iteration. However, for the system used to evaluate these methods, the improvements from using an iterative descent approach to find a better linearization point were quite small, and nearly unnoticeable on the plots. This held true for all of the descent methods, which were each able to improve on the EKF via the Mahalanobis metric, albeit with much longer computation times. Therefore, for this system, even with the high efficiency of the iEKF, it is likely preferable to stick with the standard EKF.

Future work in this area could explore other systems with even more nonlinearity, where the standard EKF performs poorly. Potentially, these methods could have more of an impact on performance than what was seen here. Additionally, a full hyperparameter sweep which considers a large number of random trajectories of the system would help determine the optimal parameters for generally strong performance, rather than overfitting to a single example. And finally, other descent methods could be considered — one interesting algorithm to examine would be Levenberg-Marquardt, which is a variation of Gauss-Newton which incorporates some aspects of gradient descent, often leading to more robust performance with some additional computational cost. [13]

For the genetic algorithms, this approach was 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, and the genetic algorithm should be trained on multiple simulations so that we do not overfit the result to just this one example.

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] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," *Journal of Machine Learning Research*, vol. 12, 2011.

[5] Y. E. Nesterov, "A Method of Solving A Convex Programming Problem With Convergence Rate O(1/k2)," *Soviet Math. Dokl.*, vol. 27, no. 2, pp. 372–376, 1983.

[6] 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.

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

[8] 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.

[9] 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.

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

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

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

[13] K. Levenberg, "A Method for the Solution of Certain Non-Linear Problems in Least Squares," *Quarterly of Applied Mathematics*, vol. 2, pp. 164–168, 1944.