Extended Kalman Filter for Flywheel Systems

Cory Ness and Jun Chen, *Senior Member, IEEE*

***Abstract*—Smart motor controllers usually provide plentiful information that makes it possible to model the various mechanisms they control. One such mechanism, a flywheel, is one of the simplest models to implement, yet has various characteristics that need to be identified for the model to be accurate. To ensure accurate control of the flywheel, not only is real-time state information necessary such as position, velocity, and acceleration, but also system characteristics such as inertia, friction, and drag. This paper investigates the use of extended Kalman filter to simultaneously estimate the system state and parameters, providing rich information that the user could use to optimally control the end mechanism.**

***Index Terms*—Extended Kalman Filter, Smart Motor Controller, Flywheel, Firmware, Embedded Implementation**

I. Introduction

Motor controllers are electronic devices that take a control signal and output a voltage to a motor responding to that signal. The smart motor controllers [1] provide additional capabilities to offload computation from the main controller to the device. By offloading the computation, the motor controller can react to sensor input faster than the main controller and abstract functionality from the main controller for faster development.

An example of such integrated smart motor controllers is the Talon FX [2], where users can configure PID gains, set position/velocity setpoints, and get information such as motor position/velocity from the motor controller, all through the CAN bus. Since the Talon FX is integrated with the motor, it also has a rotor position sensor available along with supply and stator current measurements. Moreover, the Talon FX is capable of executing some basic estimation software in the firmware to estimate system states and parameters utilizing these onboard sensors, thus providing real-time analysis of the system.

The easiest model to run in the firmware would be a simple flywheel. By modeling the flywheel's characteristics and taking into account the measurements available, the motor controller can reasonably estimate the position, velocity, and acceleration of the system. These estimates are sufficient to control the mechanism to a desired position or velocity, as is often the end goal of such mechanisms. However, in most circumstances, the characteristics of the system, such as inertia, friction, and drag, are unknown. To address this issue, this paper investigates the use of an extended Kalman filter (EKF), a state estimation technique for systems with nonlinear dynamics [3]–[9], in the firmware to estimate the system characteristics. The identified characteristics can then be used for further control or general identification of the system. Experiments demonstrate that the firmware implementation produces identical results as compared to Matlab implementation, illustrating the capability of Talon FX in running complex software in real-time.

The rest of this paper is organized as follows. Section II provides preliminaries on EKF and real-time requirements, while Section III describes the physical system and the equations used to model it. The experimental setup is presented in Section IV, together with results discussed in Section V. The paper is concluded in Section VI.

This work is supported in part by National Science Foundation through Award #2430374.

The authors are with the ECE Department, Oakland University, Rochester, MI 48309, USA (email: {coryness,junchen}@oakland.edu). *Corresponding author: J. Chen.*

II. Preliminaries

A. Extended Kalman Filter

A linear Kalman filter estimates the state of a linear system given the model, measurements, and control inputs [10]–[12]. It relies on the system being linear and is the optimal method of estimating the state for linear systems.

Extended Kalman filter (EKF), on the other hand, supports nonlinear systems by locally linearizing the system dynamics at each time step [3]–[9]. This is done by linearizing the system around the current state estimation for propagating the covariance matrix and calculating Kalman gain L [13]. This results in a non-optimal method of estimating the state, but is relatively easy to compute and still performs well under most circumstances. More specifically, given the following nonlinear dynamics,

$$x_{k+1} = f(x_k, u_k) + w_k \tag{1a}$$
$$y_k = g(x_k) + v_k, \tag{1b}$$

EKF performs the following two-step updates.

- Time update:

$$\hat{x}_{k+1} = f(\hat{x}_k^+, u_k) \tag{2a}$$
$$A_k = \left.\frac{\partial f(x,u)}{\partial x}\right|_{x=\hat{x}_k^+=,u=u_k} \tag{2b}$$
$$P_{k+1} = A_k P_k^+ A_k^T + Q \tag{2c}$$

- Measurements update:

$$C_{k+1} = \left.\frac{\partial g(x)}{\partial x}\right|_{x=\hat{x}_{k+1}} \tag{2d}$$
$$L_{k+1} = P_{k+1} C_{k+1}^T (C_{k+1} P_{k+1} C_{k+1}^T + R)^{-1} \tag{2e}$$
$$\hat{x}_{k+1}^+ = \hat{x}_{k+1} + L_{k+1}(y_{k+1} - g(\hat{x}_{k+1})) \tag{2f}$$
$$P_{k+1}^+ = (I - L_{k+1} C_{k+1}) P_{k+1} \tag{2g}$$

As will be described shortly, the flywheel system is a nonlinear system that is mostly locally linear. By putting the

inertia, friction, and drag characteristics into the state matrix, the system becomes nonlinear, and the friction component itself is nonlinear. However, when zoomed into any part of the system, it does behave linearly, making EKF an ideal approach for such a system.

B. Real Time Requirements

In order to model such a mechanism in firmware, the estimation algorithm should be implemented on a processor, which usually has limited computation. The processor used in the Talon FX is a 16-bit processor with no floating point unit. This results in floating point operations being computationally expensive and should be minimized to meet the timing requirement for real-time analysis.

The drive to change from floating point to fixed point comes from the lack of a floating point unit on the micro controller used. This means for any floating-point calculation to be performed, it needs to happen in "software", where the floating point operation needs to be broken down into exponent, mantissa, and sign operations, then pieced back together, multiplying the time taken for any operation to take place [14]. This is significantly reduced by moving to fixed point operations, as the numbers can be treated essentially as full integers for the purpose of computation.

The easiest way to reduce floating-point operations is to eliminate them entirely by changing to a fixed point notation. Fixed point is a different representation of a number, where a "fixed" number of bits is above the decimal point and the remainder are below the point [15]. This means the resolution of the number is fixed, and cannot be adjusted to best represent large and small numbers. For this particular micro controller, the computationally optimal fixed point representation is a signed 15.16 number, that is, 1 sign bit, 15 bits above the decimal, and 16 bits below the decimal. This results in a range of [-32768, 32768) with a resolution of 1.526e-6. This is sufficient for essentially any position, velocity, acceleration, inertia, friction, and drag value that would be encountered in this work.

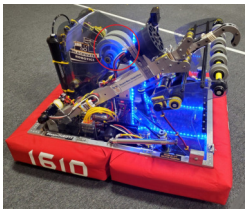
III. Flywheel Driven by DC Motor Model

A flywheel system is a mechanical system in which a rotational actuator spins a mass. There may or may not be a gear reduction between the actuator and the mass, and the mass may be any size. The rotational inertia of the mass resists any change in rotational velocity, following the rotational kinematic equations. In addition, friction and viscous friction act against the velocity of the flywheel resulting in a force counteracting the instantaneous velocity of the flywheel.

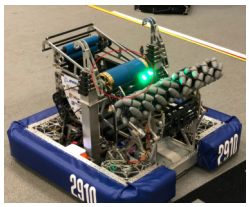
In the field of competitive robotics, the flywheel system is designed to interact with a game piece and launch it. In order to maintain consistency in the launch profile, the flywheel needs to target a rotational velocity and hold that velocity while the game piece interacts with it during the launch. Ideally, the flywheel reaches the target velocity in as little time as possible so the delay between spin-up and launch is minimized. Fig. 1 provides some examples of robots

(a) Team 254's robot in 2012.

(b) Team 1323's robot in 2017.



(c) Team 1610's robot in 2020.



(d) Team 2910's robot in 2022.

Fig. 1: Robots teams created utilizing flywheel mechanism to launch game objects from various years. Circled in red is the flywheel of each robot.

teams have created that utilize flywheels to achieve the game's objective.

A. Model Description

The flywheel model can be found by using standard kinematic equations that have been extended to the rotational domain [16]. The equations (3) and (4) below describe how the acceleration affects position (θ) and velocity (ω) over time.

$$\Delta\theta = \omega_0 t + \frac{1}{2}\alpha t^2 \tag{3}$$

$$\omega = \omega_0 + \alpha t, \tag{4}$$

where (α) is the acceleration. Note that the acceleration (α) is determined by a different equation. The first aspect of this equation is how the applied power of the motor corresponds to the angular acceleration of the flywheel, which can be described as follows [17].

$$\alpha = \frac{k_\mathrm{T} I_\mathrm{stator}}{J_\mathrm{flywheel}}, \tag{5}$$

where k_T is the motor constant describing the relationship between stator current and angular torque, I_stator is the current in the motor's stators, and J_flywheel is the flywheel's inertia. In this case, the flywheel's inertia and the motor's k_T constant are both constant assuming the system does not change, so we can lump them together to a k_I term for the system's rotational inertia with respect to applied stator current, resulting in the following simplified equation.

$$\alpha = k_\mathrm{I} I_\mathrm{stator}. \tag{6}$$

This also has an additional benefit of removing a division in the kinematic equations, making the linearization step easier and the computation faster. This also means that the "inertial" term acts more as an "inertial conductance" term, with larger

numbers indicating less inertia and faster acceleration for the same applied torque.

The next step is to take into account the drag of the system. For simplicity, the system is assumed to operate under low Reynolds numbers, resulting in linear drag or viscous friction [18]. This form of drag increases linearly with respect to velocity as follows.

$$\alpha = -D\omega. \tag{7}$$

Here D corresponds to the drag characteristic of the flywheel. The last component of the acceleration is the frictional component, which is a constant force acting against the current velocity signage. That is, it will accelerate the system in reverse if the system is moving forward and vice versa. This is described by 8

$$\alpha = -F \cdot \text{sign}(\omega), \tag{8}$$

where F is the frictional term and the `sign` function is defined as

$$\text{sign}(x) = \begin{cases} 1 & x > 0 \\ -1 & x < 0 \\ 0 & x = 0 \end{cases}. \tag{9}$$

However, due to the discretization inherent in digital signal processing, the frictional term needs to be adjusted to ensure that the resultant velocity does not change signs. This can be found using (4), finding the maximum rotational acceleration to result in $\omega = 0$. This is simplified to effectively capping the rotational acceleration to the instantaneous velocity divided by the time step, resulting in the following.

$$\alpha = \begin{cases} -F \cdot \text{sign}(\omega) & \frac{|\omega|}{t} > F \\ -\frac{\omega}{t} & \text{otherwise} \end{cases}, \tag{10}$$

which can be further simplified using the `min` function, as follows.

$$\alpha = -\text{sign}(\omega) * \min\left(F, \frac{|\omega|}{t}\right) \tag{11}$$

All these forces are combined to result in the final rotational acceleration of the system described by (12).

$$\alpha = k_{\text{I}} I_{\text{stator}} - \text{sign}(\omega) * \min\left(F, \frac{|\omega|}{t}\right) - D\omega. \tag{12}$$

B. Matrix Representation of the Model

Given the system dynamics described in (3), (4), and (12), the matrix representation can be derived as follows. Define the state variable as $x = \begin{bmatrix} \theta & \omega & \alpha & k_{\text{I}} & F & D \end{bmatrix}^T$ and the control variable as $u = \begin{bmatrix} I_{\text{stator}} \end{bmatrix}$. Note that here the state vector includes system parameters k_{I}, F and D for the purpose of online parameter estimation. The Jacobian matrix with respect to x is then given as follows.

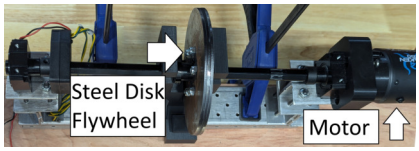


Fig. 2: Top down view of the mechanism. The motor is directly coupled to the shaft with a large steel disk connected.

- When $\frac{|\omega|}{t} > F$

$$A = \begin{bmatrix} 1 & t & \frac{1}{2}t^2 & 0 & 0 & 0 \\ 0 & 1 & t & 0 & 0 & 0 \\ 0 & 0 & 0 & I_{\text{stator}} & -\text{sign}(\omega) & -\omega \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}; \tag{13}$$

- Otherwise

$$A = \begin{bmatrix} 1 & t & \frac{1}{2}t^2 & 0 & 0 & 0 \\ 0 & 1 & t & 0 & 0 & 0 \\ 0 & \frac{1}{t} & 0 & I_{\text{stator}} & 0 & -\omega \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{14}$$

For the output calculation, since only the rotor position is measured by the sensor, the output equation is truly linear, as follows.

$$y = \begin{bmatrix} \theta_{\text{meas}} \end{bmatrix} \tag{15}$$

Therefore, the corresponding C matrix for EKF is given by

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{16}$$

IV. Experimental Setup

This model was then verified using a real-world mechanism to collect data. See Figs. 2 and 3. This mechanism was created using a Kraken X60 brushless motor with an integrated Talon FX smart motor controller connected 1:1 to a hex shaft spinning a large steel flywheel.

The mechanism was designed so that artificial friction and drag could be introduced and removed in order to prove that the EKF real-time estimator can correctly identify a change in friction and/or drag, and that the inertia of the system remains constant. This is done by adding and removing a 3D-printed structure with holes for magnets (See Fig. 4). The introduction of the structure itself adds some friction and a small amount of drag, while the magnets will add primarily drag to the system. This is due to Lenz's law, where a magnetic field moving relative to a piece of conductive material will produce a magnetic field proportional to the velocity of the movement [19]. That magnetic field will oppose the original magnetic field, and so there is a force that acts as a brake. Since the magnitude of that force is proportional to the velocity, the overall effect on the system is an apparent drag, a braking force that is linear to the velocity of the system, captured by the D term of (12).

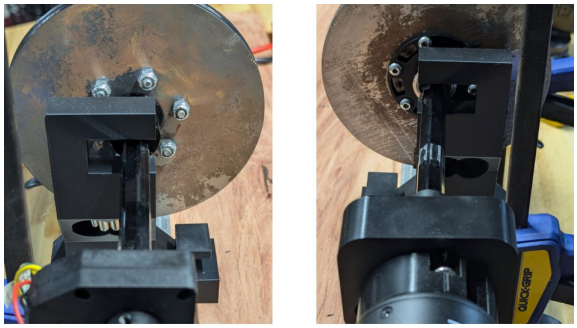


Fig. 3: Side view of the mechanism. (a) With the removable magnets in place to provide an artificial drag component to the mechanism. (b) The magnets removed so there is no artificial drag component.

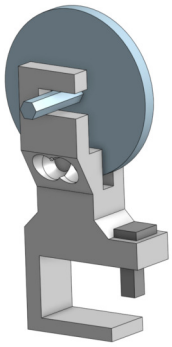


Fig. 4: Model of component added to artificially increase drag and friction.

A. Data Collection

The Talon FX can be communicated with via CAN using a CANivore on a Windows Computer. A Python project was created to control the motor and acquire data from the motor. The Python code uses a modified Phoenix 6 library to provide inertia/friction/drag variables from the Talon FX that runs the EKF estimator. The primary data collected during the initial runs were the position and the torque current reported by the Talon FX. This data was then analyzed (by Matlab) to verify that all 6 state variables were able to be calculated and were stable. After this was confirmed, the EKF was then implemented in C language to run on the firmware inside the Talon FX, with all 6 state variables available for comparison against the MATLAB implementation.

B. Implementation in MATLAB

The EKF estimator for flywheel systems was first implemented in MATLAB, which is a generic implementation that utilizes a nonlinear function for the prediction and observation steps, and a linearization function to get the A and C matrices. Some key things to note are that the time steps, as captured by the acquired data, are not constant, so the change in time is represented with a dT variable used to adjust both the prediction/observation steps and the linearized A and C matrices to account for the time-varying step size. The linearization of the A matrix has two solutions depending on if the angular velocity is so slow that the frictional term brings the velocity to 0 or if there's sufficient velocity for friction to take full effect.

C. Implementation in C

After the MATLAB implementation was verified, the code was ported to C to run on the micro controller inside the Talon FX. The first implementation of the EKF was using floating point to verify the implementation was correct and any issues present were not due to the fixed point implementation. Despite this, there were various issues that were encountered.

1) Matrix implementation: The first issue was the C implementation of matrix operations. Since the code was running on a small micro controller, the matrix operations had to be created by hand. The debug and verification of matrix implementation was done by creating a suite of unit tests to compare the hand-written C code against the MATLAB code.

2) Not A Number (NaN) results: After the Matrix operations were confirmed to be correct, there was occasional NaN results that propagated throughout the model. This was due to a float conversion to int that was later being used as a dividend. The floating point number could occasionally be less than 1, which would result in a division by 0 and the result of NaN. This was discovered by logging all the original data getting passed into the firmware EKF and passing it into a C version of the EKF running on Desktop C++. Through the use of breakpoints the NaN was found and the bug was fixed.

3) Discrepancies with MATLAB: The small discrepancies with the MATLAB implementation, particularly in the covariance propagation,s was worked through by debug-stepping iteration over iteration between the Desktop C version of the EKF and the MATLAB version. This resulted in the error being attributed to standard floating point error, particularly as the MATLAB implementation uses "double" precision under the hood while the C version explicitly used the"float" keyword specifying "single" precision, resulting in significantly less overall precision.

4) Very large covariance values: While not specific to the C implementation, it was also noted that the covariance terms grew to be very large in both the MATLAB and C implementations, on the order of 100s of thousands. This posed to be a problem in the eventual fixed point conversion, as a 15.16 fixed point number can only represent values between [-32768, 32768). After stepping through a few covariance propagation cycles, it became obvious that the initial noise terms used were way too high. The prediction noise was on the order of 1 when it should be closer to an order of 0.01, while the measurement noise was 10 when it should be closer to 0.001. This change makes sense when taking into account the units of the model and sensor, the model operates in rotations and the measurement is also in rotations. The sensor used is a 11-bit sensor with very low noise, so should be accurate

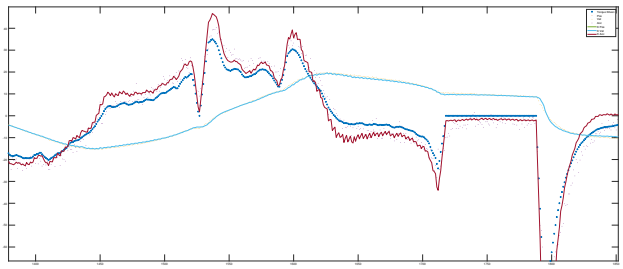


Fig. 5: Estimated Position/Velocity/Acceleration from MATLAB EKF to time-derivative estimation. The variables with a K prefix are from the MATLAB EKF.

within 0.0005. After making these corrections, the covariance terms dropped significantly and were well within the bounds of being represented in fixed point notation.

After these issues were worked through, the floating point implementation was changed to fixed point with no undesirable effects. By making the switch from floating point to fixed point, the time taken to calculate the model also dropped from 2.426 ms to 769.29 us, a nearly 3x improvement. Getting the full calculation under 1 ms is particularly important, as the goal is to run the model once every millisecond.

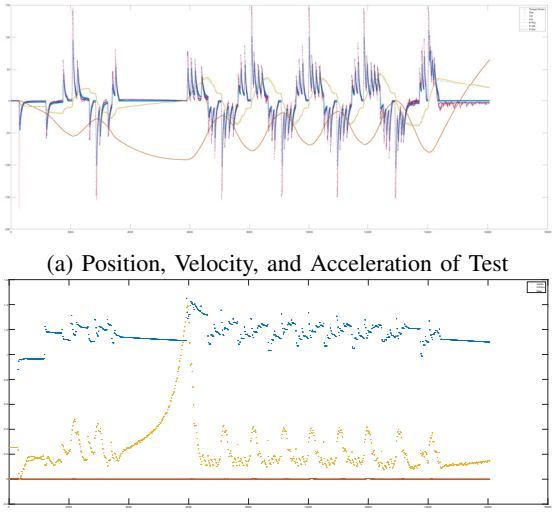
V. Results and Discussions

The overall results were exceptional. As discussed below, the EKF correctly identified systems with additional friction and drag and resulted in cleaner and more responsive accelerations compared to classic time-derived velocity and acceleration estimates.

In Fig. 5, the experiment was to drive the flywheel and measure the position/velocity/acceleration/stator current of the motor during the data collection period. Note that velocity and accelerations are measured here for comparison purpose, and they are not utilized by the EKF for estimation. The position and stator current values are then fed into the MATLAB implementation of EKF to estimate the characteristic inertia/friction/drag and provide a filtered position/velocity/acceleration. By comparing the acceleration, the noisiest variable under typical circumstances, the filter performance can be subjectively determined. In this case, the filtered acceleration from EKF is significantly less noisy while still following the general trend, even leading the measured acceleration. This indicates a strong correlation between the model and the real world, as the model is able to predict what the acceleration will be while the measurement lags behind.

Figs. 6, 7, and 8 show the results with EKF running in the firmware to estimate Inertia/Friction/Drag characteristics on the fly. In particular, in Fig. 6, the experiment was conducted without any artificial friction or drag added. In Fig. 7, the experiment was conducted with artificial friction added, while in Fig. 8, the experiment was conducted with both artificial friction and drag added.

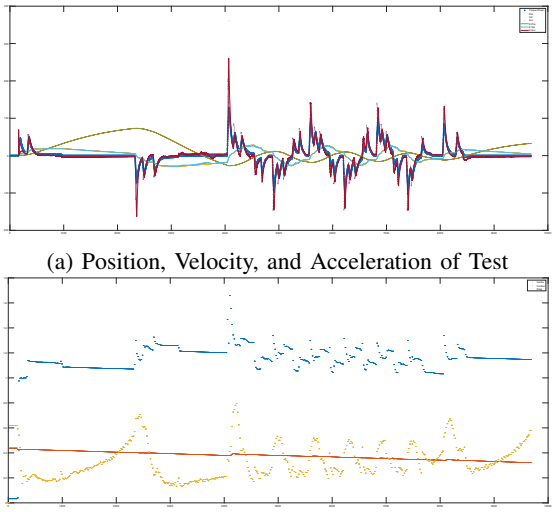
By analyzing EKF estimation results and comparing them against each other knowing the general state of the mechanism,



(a) Position, Velocity, and Acceleration of Test

(b) Inertia, Friction, and Drag of Test

Fig. 6: Results of the EKF running in firmware without the artificial drag/friction. At the end, inertia is roughly 1.1, friction is essentially 0, and drag is about 0.14.

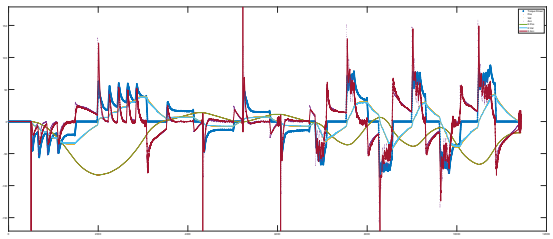


(a) Position, Velocity, and Acceleration of Test

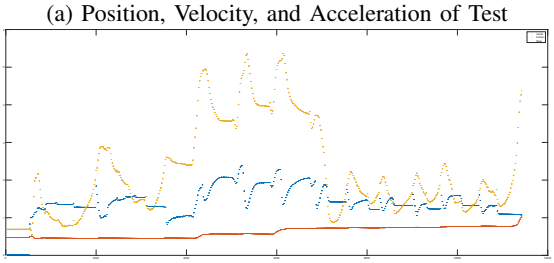
(b) Inertia, Friction, and Drag of Test

Fig. 7: Results of the EKF running in firmware with artificial friction added. At the end, inertia is roughly 1.1, friction increases to 0.3, and drag increases to $\sim$0.4.

the model can be examined and determined if it's able to correctly identify systems with additional friction/drag and correctly diagnose that the differing levels of effort required to drive the mechanism isn't due to a change in inertia. In all 3 cases, the inertia was identified as being around 1.1, correctly identifying that there was no significant impact on the system's inertia between the 3 cases. In addition, as friction and/or drag was added to the system, the model was able to correctly identify and provide an increasing friction/drag characteristic value. Throughout all the experiments, the position/velocity/acceleration was very clean, further indicating the



(a) Position, Velocity, and Acceleration of Test



(b) Inertia, Friction, and Drag of Test

Fig. 8: Results of the EKF running in firmware with artificial drag and friction added. At the end, inertia is roughly 1.1, friction increases to 0.7, and drag increases to ∼2.4.

model was a good representation of the real world mechanism.

Based on the results presented, it's clear that the proposed EKF estimator in a smart motor controller can provide meaningful improvement to key variables that are reported, and opens the door to additional opportunities.

- First, it is possible to take the flywheel characteristics and use them to calculate ideal PID gains for position and velocity control. This reduces the problem of finding optimal closed loop gains to simply driving the mechanism with the model running in the background, allowing data to collect, and finally plugging the found characteristics into a calculator rather than the tedious work of hand-tuning the PID gains.
- In addition, after the characteristics are found, the model simplifies to an even more linear model, with the only nonlinear term becoming the frictional term. If the frictional component of the system is small enough, such as the mechanism in Fig. 6, it can be dropped and turned into a completely linear system that can be controlled through a linear discrete Model Predictive Controller [20].
- One last possibility is health monitoring and failure diagnosis through constant monitoring of system characteristics and identification [21]. For example, in Fig. 7, if the inertia were to change significantly to a value of 4, that would indicate a significant decrease in inertia and could be caused by the flywheel no longer being attached to the motor, a catastrophic failure.

VI. Conclusions

This paper investigates the use of extended Kalman filter (EKF) for the real-time state and parameter estimation in flywheel systems. Both Matlab and firmware implementation (in C) are developed. Several experiments are conducted with and without additional artificial friction/drag added. Experiment results demonstrate that the proposed EKF estimator, with only position measurements, can accurately estimate the position, velocity, and acceleration, while at the same time providing accurate estimation of the system characteristics such as inertia, friction, and drag. The proposed method requires less than 1 ms computation time in firmware, and hence is suitable for real-time applications. Future work include utilizing the estimated system characteristic for control design and health monitoring.

References

[1] Z. Huang, S. Qiu, B. Wang, and Q. Liu, "A real-time field bus architecture for multi-smart-motor servo system," *Scientific Reports*, vol. 14, no. 1, p. 3918, 2024.

[2] C. T. R. Electronics, "Talonfx," accessed December 6, 2024. [Online]. Available: https://v6.docs.ctr-electronics.com/en/stable/docs/hardware-reference/talonfx/index.html

[3] L. Nuculaj, A. Kidwell, C. Homayouni, A. Fillmore, D. Hanna, and J. Chen, "Optimal FPGA implementation of dense extended Kalman filter for simultaneous cell state estimation," in *IEEE Int. Midwest Sym. on Circuits and Systems*, Springfield, MA, August 11–14, 2024.

[4] J. Chen, "Extended kalman filter steady gain scheduling using k-means clustering," *International Journal of Modelling, Identification and Control*, vol. 34, no. 2, pp. 158–162, February 2020.

[5] B. Sinopoli, L. Schenato, M. Franceschetti, K. Poolla, M. I. Jordan, and S. S. Sastry, "Kalman filtering with intermittent observations," *IEEE transactions on Automatic Control*, vol. 49, no. 9, pp. 1453–1464, 2004.

[6] G. Y. Kulikov and M. V. Kulikova, "Accurate numerical implementation of the continuous-discrete extended kalman filter," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 273–279, 2013.

[7] H. R. Hashemipour, S. Roy, and A. J. Laub, "Decentralized structures for parallel kalman filtering," *IEEE Transactions on Automatic Control*, vol. 33, no. 1, pp. 88–94, 1988.

[8] A. Tsiamis and G. J. Pappas, "Online learning of the kalman filter with logarithmic regret," *IEEE Transactions on Automatic Control*, vol. 68, no. 5, pp. 2774–2789, 2022.

[9] S. Liu, Z. Wang, Y. Chen, and G. Wei, "Protocol-based unscented kalman filtering in the presence of stochastic uncertainties," *IEEE Transactions on Automatic Control*, vol. 65, no. 3, pp. 1303–1309, 2019.

[10] Y. Kim, H. Bang *et al.*, "Introduction to kalman filter and its applications," *Introduction and Implementations of the Kalman Filter*, vol. 1, pp. 1–16, 2018.

[11] D. Simon, "Kalman filtering," *Embedded systems programming*, vol. 14, no. 6, pp. 72–79, 2001.

[12] G. Welch, "An introduction to the kalman filter," 1995.

[13] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," *Institute for Systems and Robotics*, vol. 43, no. 46, pp. 3736–3741, 2004.

[14] Louca, Cook, and Johnson, "Implementation of ieee single precision floating point addition and multiplication on fpgas," in *Proc. IEEE Sym. on FPGAs for Custom Computing Machines*, 1996, pp. 107–116.

[15] R. Yates, "Fixed-point arithmetic: An introduction," *Digital Signal Labs*, vol. 81, no. 83, p. 198, 2009.

[16] P. Peter Urone and R. Hinrichs, *Physics*. OpenStax, Mar 2020. [Online]. Available: https://openstax.org/books/physics/pages/6-3-rotational-motion#Table_06_03

[17] D. Lin, P. Zhou, and Z. J. Cendes, "In-depth study of the torque constant for permanent-magnet machines," *IEEE Transactions on Magnetics*, vol. 45, no. 12, pp. 5383–5387, 2009.

[18] I. Eames and C. A. Klettner, "Stokes and lamb's viscous drag laws," *European Journal of Physics*, vol. 38, no. 2, p. 025003, jan 2017.

[19] P. P. Urone and R. Hinrichs, "23.2 faradays law of induction: Lenzs law," *College Physics*, 2016.

[20] J. Chen, L. Zhang, and W. Gao, "Reconfigurable model predictive control for large scale distributed systems," *IEEE Systems Journal*, vol. 18, no. 2, pp. 965–976, June 2024.

[21] J. Chen and R. Kumar, "Fault detection of discrete-time stochastic systems subject to temporal logic correctness requirements," *IEEE Trans. Auto. Sci. Eng.*, vol. 12, no. 4, pp. 1369–1379, October 2015.