SUPPLEMENTARY MATERIAL: NUMERICAL AND HARDWARE EXPERIMENTS WITH ANALYSIS

Supplementary Material is divided into three parts: 1) the numerical experiments [17] in Sec. A/Fig. 2 are designed by taking all the scenarios in [3, Sec. 4] into consideration, the first one of which is demonstrated to strengthen the confidence in the proposed, computationally improved SDDRE guidance law while similar enhancements in the computational performance can also be observed in other scenarios. Moving forward, 2) the numerical setup in Sec. B/Fig. 3 is modified from the last scenario in [3, Sec. 4], namely the most difficult interception condition as illustrated in [3, Fig. 14], which aims at demonstrating the robustness capability of the guidance design. The same parameter settings in the SDDRE scheme are adopted throughout this section as well as [3, Sec. 4]: $(\mathbf{S}, \mathbf{Q}, \mathbf{R}) = (10^9 \cdot \mathbf{I}_{4\times4}, 10^3 \cdot \mathbf{I}_{4\times4}, \mathbf{I}_{2\times2})$, which are equivalently modified from [3] for the sake of compactness. Note that all the validations in 1) and 2) are built on the MATLAB® platform till the latest R2022a version, and the "`ode45.m`" function therein is used to solve differential equations – with default settings that include the automatic step size of validation time – so as to grow more confidence that benefits real implementations in 3). Specifically, the hardware experiments in 3)/Sec. C/Fig. 4 are subject to computing platforms of microcontroller (Python-based) and FPGA, respectively. The main motivation of Sec. C comes from the long quest among practice engineers for an *efficient* ARE solver (see, for example, a Taylor-approximated but realizable tradeoff in [21] and references [30]–[32] therein), while a main objective aims to re-engage the interest in the doubling algorithms [34] and demonstrate a variety of advantages via the state of the art [23] as compared to the classic benchmark [7], [20]. To more clearly summarize Supplementary Material, Remark 4/Table I is provided in the end to quantify the validated computational enhancements.

A. Rebuild of [3, Fig. 10]

To strengthen the confidence in the proposed results, we consider the very first engagement scenario in [3, Sec. 4] (specifically, Fig. 10 therein) that is subject to the parameter setups: $r = 10^4$ [m], $V_M = 300$ [m/s], $(\theta, \psi, \theta_M, \psi_M)|_{t=0} = (-60^\circ, 40^\circ, 20^\circ, 20^\circ)$, $\eta = 0.1$ [unitless], $x_5(0) = 10$ [unitless], $t_c = 20$ [sec], $|a_M^i| \leq 10 \cdot$ g [m/s^2], where $i = y, z$ and $g = 9.8$ [m/s^2] denotes the gravitational constant, and the pre-specified final lead angles are $(\theta_{T_f}, \psi_{T_f}) = (35^\circ, -29^\circ)$, which can be related to the impact angles $(\theta^*, \psi^*) = (-30^\circ, 35^\circ)$; whereas the other settings are referenced to [3] to remain focused of this brief.

All the validations are illustrated in Fig. 2, where the state responses are depicted in Figs. 2 (a) and (b) while the proposed computational enhancement is shown in Figs. 2 (c)-(e). At first, it is worth mentioning that Figs. 2 (a) and (b) provide further validations that are in addition to those in [3, Fig. 10] while in accordance with the state-space formulation in Eq. (1). In a consistent manner, the state trajectories $\{x_i\}_{i=1}^4$ – the red/dash-dot line and blue/dashed line in Fig. 2 (a) as well as both lines in Fig. 2 (b) – are converging toward zero, whereas the auxiliary variable x_5 – the green/dotted line in Fig. 2 (a) – remains nonzero across the horizon, which more comprehensively validate the design objective. As a step forward, the focus is set upon the proposed results for a *fast* SDDRE-based lead angle guidance. In Fig. 2 (c), the major computational burdens using the classic SDDRE implementations [3], [6] are illustrated, where

- the *dominant* computation time is depicted by the red/dashed line, which corresponds to the "`minreal.m`" routine as suggested by MATLAB® for the pointwise applicability/solvability check of SDDRE;
- the second/other major computational effort is plotted in the blue/solid line, which is associated with the commonly adopted "`lqr.m`" function in MATLAB® [7], [20], [21] that iteratively approaches the pointwise solution to the underlying ARE [6].

Note that, for easy comparison, the red line is normalized with respect to the blue one. It is shown in Fig. 2 (c) that the red line is always above the blue one, which (dominance) amounts to an average ratio of 6.29; in other words, the applicability-checking routine that uses "`minreal.m`" costs more computation time than the primary SDDRE-solving step, on average, around 6.29 times more. Such observation is consistent from the analytical viewpoint: "`minreal.m`" bears $O(n^4)$ flops to the order, which is estimated according to [35], whereas "`lqr.m`" is of minor complexity $O(n^3)$ [23]. However, by virtue of the proposed results, the dominant computational burden is no longer required. Specifically, the checking process via the numerical scheme "`minreal.m`" is replaced by the analytical guarantee in Theorem 1, in an offline manner and throughout the state space. Remarkably, the solving process (blue line) has already caused tremendous concerns/reservations among the control community [14], [15], [27], let alone the more significant and dominant checking process, where the latter burden has been completely removed by Theorem 1. Moreover, with the intention of a more thorough improvement, it is possible to advance the computational enhancement by exhaustively exploiting the many and various ARE solvers until the states of the art [17], which is described below.

Among the major computational burdens [16], the minor one that corresponds to the underlying ARE solving [3], [6] is evaluated in Fig. 2 (d), where

- solid/blue line still shows the computation time using "`lqr.m`" and serves as the normalization reference for the red line;
- dashed/red line alternatively leverages the state-of-the-art ARE solver "Structure-Preserving Doubling Algorithm (SDA) [23]", whose best efficiency stands out from extensive trials among the latest benchmarks [17].

Before proceeding to the focused comparisons of computational efficiency, it is necessary to mention that both "`lqr.m`" and SDA yield almost identical state and control responses as depicted in Figs. 2 (a), (b), and [3, Fig. 10 (e)], respectively, which thus are not repeated herein for brevity. Accordingly, it can now be meaningful to move on to Fig. 2 (d), which highlights that the red line is below the blue one across the

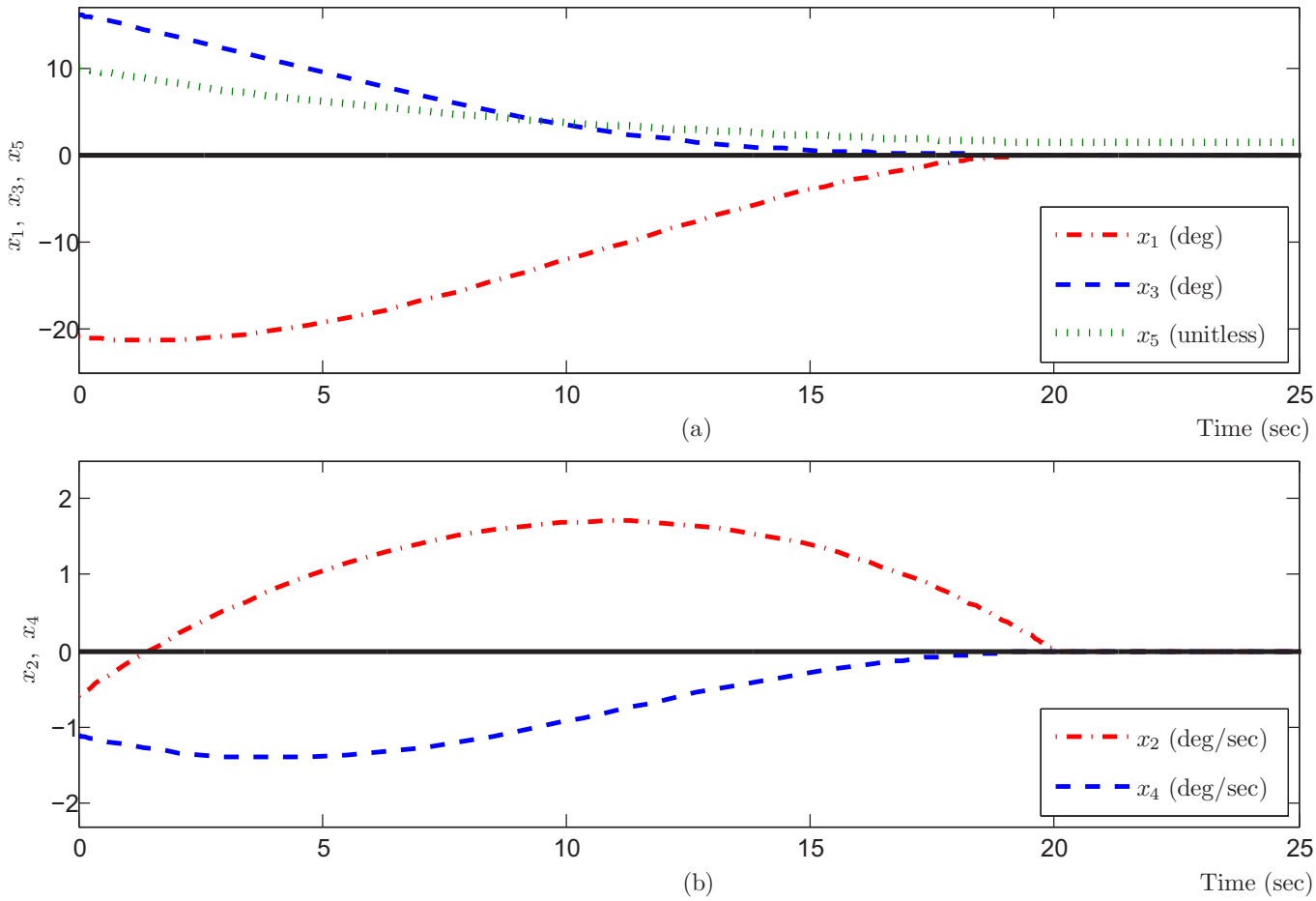

Fig. 2: Time histories of engagement case in Sec. A, where (a) $\{x_i\}_{i=1,3,5}$ and (b) $\{x_j\}_{j=2,4}$ are state variables of guidance system in Eq. (1) [3];

validation time. Averagely speaking, the SDA only requires 42.3% of the computation time used by "`lqr.m`". Such observation can also be supported by the complexity analysis [23]: $O(64n^3)$ via SDA whereas $O(200n^3)$ using "`lqr.m`", to the worst-case scenario. However, such advantage is not limited to this guidance problem, but can be utilized to benefit other Riccati-based designs that boast popularity in the aerospace literature and beyond.

Finally, Fig. 2 (e) manifests an overall evaluation, that is, the total computation time to obtain the guidance command [3, Algorithm 1]. Specifically,

- solid/blue line adopts the classic scheme with regard to the dominant burden, namely "`minreal.m`", and the other/second major burden, namely "`lqr.m`", [8], [13], [16], which is used to normalize the red line for better clarity;
- dashed/red line refers to the proposed/selected alternative, that is, "`minreal.m`" (resp., "`lqr.m`") replaced by Theorem 1 (resp., SDA [23]).

As suggested by Fig. 2 (e), the red line is always below the blue one; in other words, the proposed alternative is faster than the classic guidance command implementation. On average, the total computation time is reduced by a significant portion of 94.14%, which can also be inferred from the specific results in Figs. 2 (c) and (d), namely, $1-0.423/(6.29+1)\approx 94.19\%$. The proposed enhancement re-marks a recent achievement [22] that reduces 65% in the minor ARE solving and amounts to an overall reduction of "$1-(6.29+0.35)/(6.29+1)\approx 8.9\%$", where the MATLAB®-suggested routine "`minreal.m`" is selected for estimating the solvability-checking process as in Fig. 2 (c). As a result, the computational performance for the focused SDDRE-based guidance strategy is further and largely improved, which aligns with the requirement toward agile missile dynamics, while the enduring concerns for SDDRE/SDRE applications [14], [15], [27] are significantly alleviated. Although similar enhancements can also be seen and endorsed by demonstrating other scenarios in [3, Sec. 4], the following section considers a more challenging engagement condition as evolved from the most difficult case in [3, Sec. 4], which aims at a wider validation coverage that highlights the increased robustness.

B. More Challenging Engagement

In a unified framework, the proposed result has removed the *dominant* computational burden pointwise at each time or system state, including all the validations in [3, Sec. 4]; more specifically, Theorem 1 completely removes the tedious

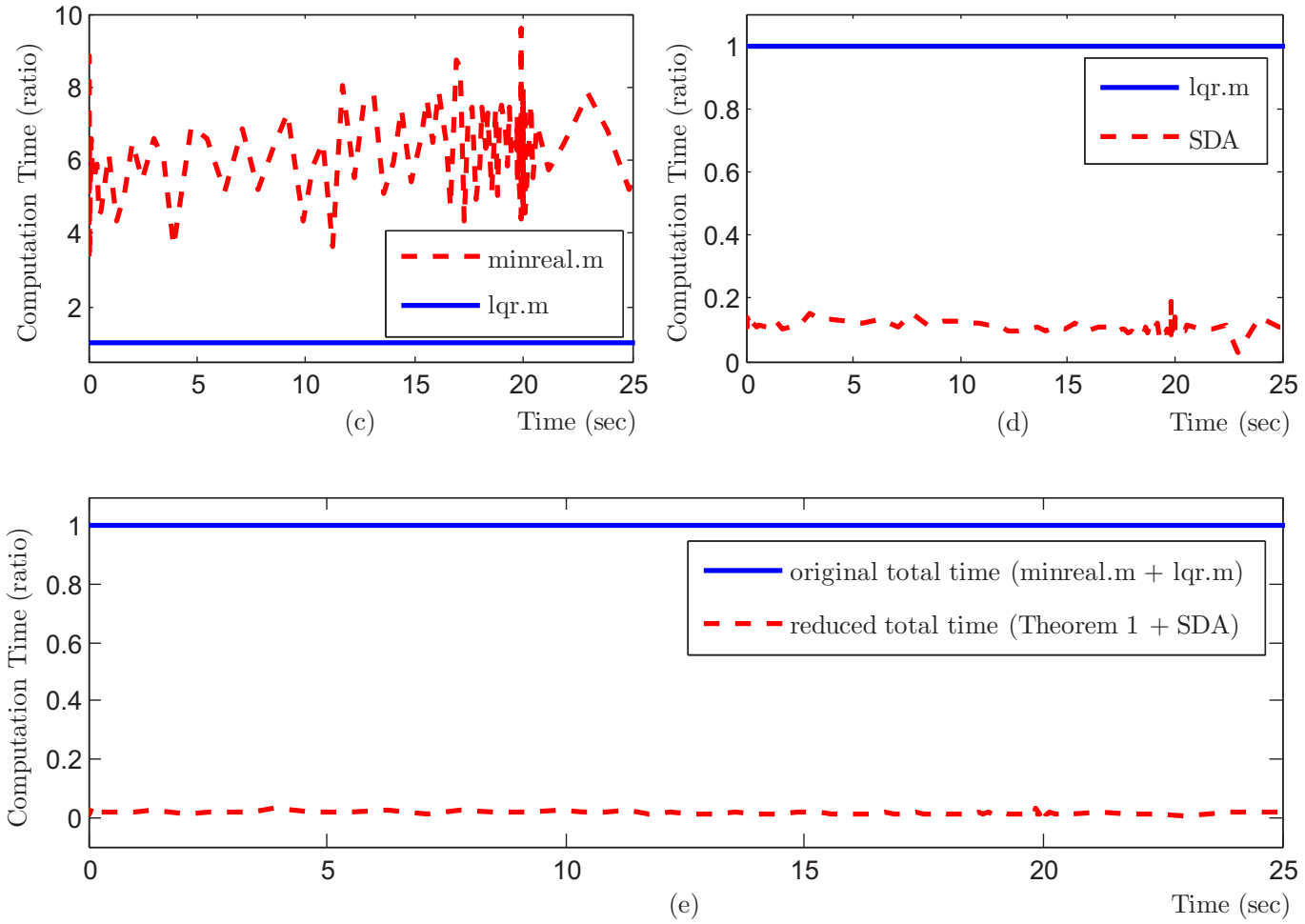

Fig. 2 (Cont.): (c) major computation times in formulating the SDDRE-based guidance command [3], (d) computational improvement for the second/other burden (ARE solving), and (e) overall computational reduction by comparing the classic scheme [6] with the enhanced alternative (Theorem 1 and SDA [23]).

applicability/solvability-checking routine [11], [14], which benefits a *fast* SDDRE-based lead angle guidance law. As a step forward, this section is dedicated to broadening the validation spectrum by considering a more challenging scenario that evolves from the maneuvering-target interception in [3, Fig. 14]. For brevity, the significantly different discussions as compared to Sec. A are presented afterward. To start with, the different parameter settings are summarized: $(\theta_M, \psi_M)|_{t=0} = (25^\circ, 25^\circ)$ in comparison to [3, $(\theta_M, \psi_M)|_{t=0} = (20^\circ, 20^\circ)$], which suggests that the interceptor is initially directed further away from the target; $(\theta_{T_f}, \psi_{T_f}) = (57^\circ, 26^\circ)$, which is associated with $(\theta^*, \psi^*) = (-30^\circ, -20^\circ)$, $t_c = 24.9$ [sec], $(V_M, V_T) = (600, 300)$ [m/s], and $a_T^y = a_T^z = 10$ [m/s^2]. It is worth reiterating that this Sec. B adopts the same parameter settings in the SDDRE scheme, namely the weights $(\mathbf{S}, \mathbf{Q}, \mathbf{R})$, which also strengthen the robustness value in the guidance design, in addition to the proposed results for a robust computational enhancement (Corollary 1).

Similar to the validations in the previous scenario/Fig. 2, it is informative to illustrate both the state and control behaviors as given in Figs. 3 (a)-(d). The results indicate that, even subject to this more difficult engagement condition, the guidance law is still satisfactory by stabilizing all the $\{x_i\}_{i=1}^4$ trajectories and maintaining x_5 nonzero, where the former results are depicted by the red/dash-dot line and blue/dotted line in Fig. 3 (a) as well as both lines in Fig. 3 (b) while the latter by the green/dashed line in Fig. 3 (a), respectively. Note that the x_5-trajectory is gradually decreasing and its final value equals 0.82. Moreover, Fig. 3 (c) adopts another viewpoint to more directly validate the effectiveness of the guidance law, where the missile (the red/solid line whose initial position is marked by the circle) successfully intercepts the target (blue/dotted line that starts from the ball) at the star point. Remarkably, the acceleration commands "a_M^z and a_M^y" in Fig. 3 (d) are smooth and within the constraint of $|10 \cdot g|$ [3].

However, all the above validations can only be argued meaningful if the essential SDDRE's applicability is guaranteed. As a common practice [7], [14], such a guarantee is numerically verified; see Fig. 3 (e) for an instance. Note that the line specifications in Figs. 3 (e)-(g) are the same as the corresponding Figs. 2 (c)-(e), respectively, for the sake of easy comparison. Similar to other scenarios, the applicability-checking process

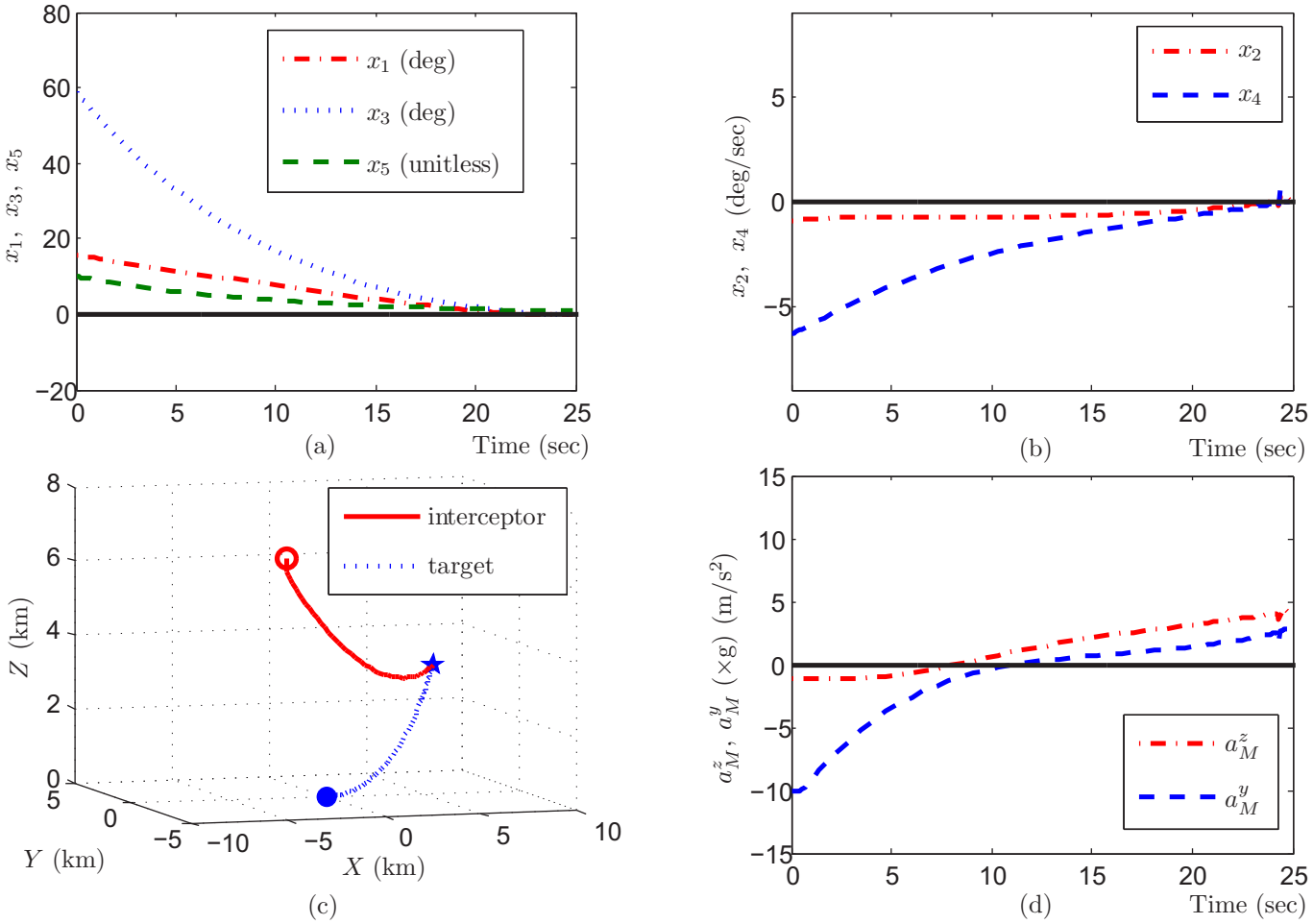

Fig. 3: Time histories of interception scenario in Sec. B, where (a) $\{x_i\}_{i=1,3,5}$ and (b) $\{x_j\}_{j=2,4}$ are system states of guidance design in Sec. II-B [3], (c) spatial trajectories, and (d) control inputs (acceleration commands);

is also found to be *dominant* throughout the horizon in Fig. 3 (e) and, subject to this scenario, it averages 6.31 times more significant than the other major computational burden. Once again, by applying Theorem 1, the dominant load is no longer required. Then, the focus is shifted forward to enhancing the other/second major computational effort, and the SDA solver still yields the best performance among extensive trials in [17] that include the renowned "`lqr.m`" [7], [20], [21]. Notably, both the trajectories using SDA and "`lqr.m`" almost overlap with respect to all the state and control behaviors in Figs. 3 (a)-(d). Nonetheless, according to Fig. 3 (f), the advantage in computational efficiency using SDA is observed prominently across the validation period, the average of which is evaluated to be around 42.3% of the computation time via "`lqr.m`". This is largely attributed to its structure-preserving feature, which is designed differently from the dimension-increased (from $n \times n$ to $2n \times 2n$) Hamiltonian system using "`lqr.m`" [23]. To sum up, both of the computational enhancements with regard to the two major burdens [8], [13], [16] contribute to an overall fast SDDRE implementation, which – as shown in Fig. 3 (g) – merely requires 5.9% on average of the classic scheme [3], [6]. Such an improved efficiency is consistent with the validations in Sec. A as well as the complexity analysis therein, which further promotes the proposed fast SDDRE implementation for the lead angle guidance [3].

C. Experiments on Microcontroller and FPGA

In the control literature, the SDRE, SDDRE, or even Riccati-based designs in general often favor the classic QR-based method [20] to grow confidence until the simulation stage, which is mainly owing to the popularity of MATLAB® that implements the method as "`lqr.m`" [3], [7], [13]. However, it still poses a challenging question on how to efficiently realize the method in hardware practice. Consequently, most practice engineers resort to compromises that tradeoff between solving time and accuracy; in other words, the real-time efficiency in ARE solving has to be compromised. To name one example, [21] and the references [30]–[32] therein adopt a Taylor-series approximation scheme for hardware experiments; however, not only the solving accuracy is worsened from the QR method (due to the limited/finite use of Taylor-series expansion terms), but also a variety of practical control performances are undermined. Indeed, the performance in solving time is preserved, or, more accurately speaking, "tuned" to be acceptable as per the sampling interval. After

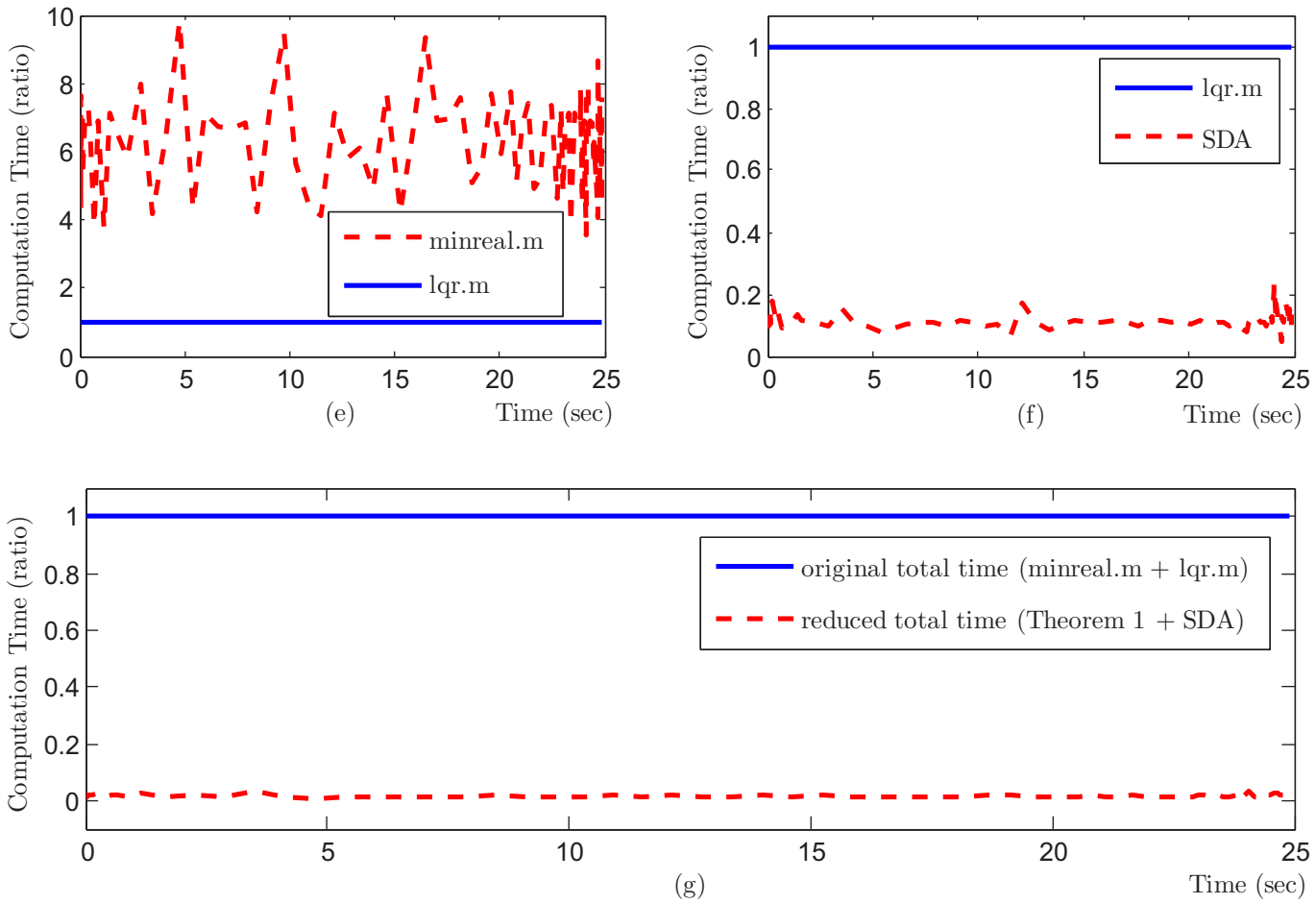

Fig. 3 (Cont.): (e) major computation times in the formulation of SDDRE-based guidance command [3], (f) computational enhancement for the second major effort (ARE solving), and (g) total computational reduction using the proposed/selected scheme (Theorem 1 and SDA [23]) as improved from the classic method [6].

extensive trials in the state-of-the-art ARE solvers [17], the much-less-known subset of doubling algorithms [34] is re-investigated, particularly from the aspects of its latest update "SDA" [23] and the SDRE/SDDRE design framework. In a consistent manner from Secs. A and B, this Sec. C explores other potentials of SDA, namely, simplicity and efficiency in real implementation. Beyond the typical validations [17], the computing platforms in the following experiments are subject to Avnet® Ultra96-V2 (Xilinx® Zynq UltraScale+ MPSoC). To be more detailed, the computational enhancement is evaluated on both the Python-based microcontroller (equipped with ARM® Cortex-A53 MPCore with CoreSight and 2GB LPDDR4 Memory) and FPGA (ZU3EG device), respectively.

Among all experimental results, at first, we highlight that the *dominant* computational burden in this SDDRE-based guidance command [3] (that is, pointwise applicability checking) is almost removed thanks to Theorem 1, which is also observed in Secs. A and B. Therefore, for the sake of brevity, only the mitigation of the other/second major burden is presented afterward, and the results are unified in Fig. 4. Moreover, the most challenging engagement scenario across Secs. A, B, and [3, Sec. 4] is explicitly demonstrated below; while similar results can be observed in other conditions, which further strengthen the merit of the proposed enhancements. In Fig. 4, two sets of computational enhancements are given, where the enhancement in solving time (resp., accuracy) is illustrated in Fig. 4 (a) and (b) (resp., (c)); while three responses associated with computing setups below are depicted in each subfigure, respectively:

- solid/black line plots the popular QR-based method [3], [7], [20] that is also implemented in the Python-based microcontroller, more specifically, embedded in the "`scipy.linalg`" and "`control.matlab`" libraries,
- dotted/red line is also subject to the microcontroller as the black line, but alternatively implements the SDA solver [23];
- within the FPGA environment, only the dash-dot/blue line that is further enhanced from the red line can be provided, whereas the counterpart of the black line has been widely reported as difficult or impossible, and thus it cannot be included for comparisons.

In accordance with Figs. 2 and 3, the computation time in Fig. 4 (a) is also normalized by the QR-based benchmark, such that the proposed enhancement can be easily evaluated for

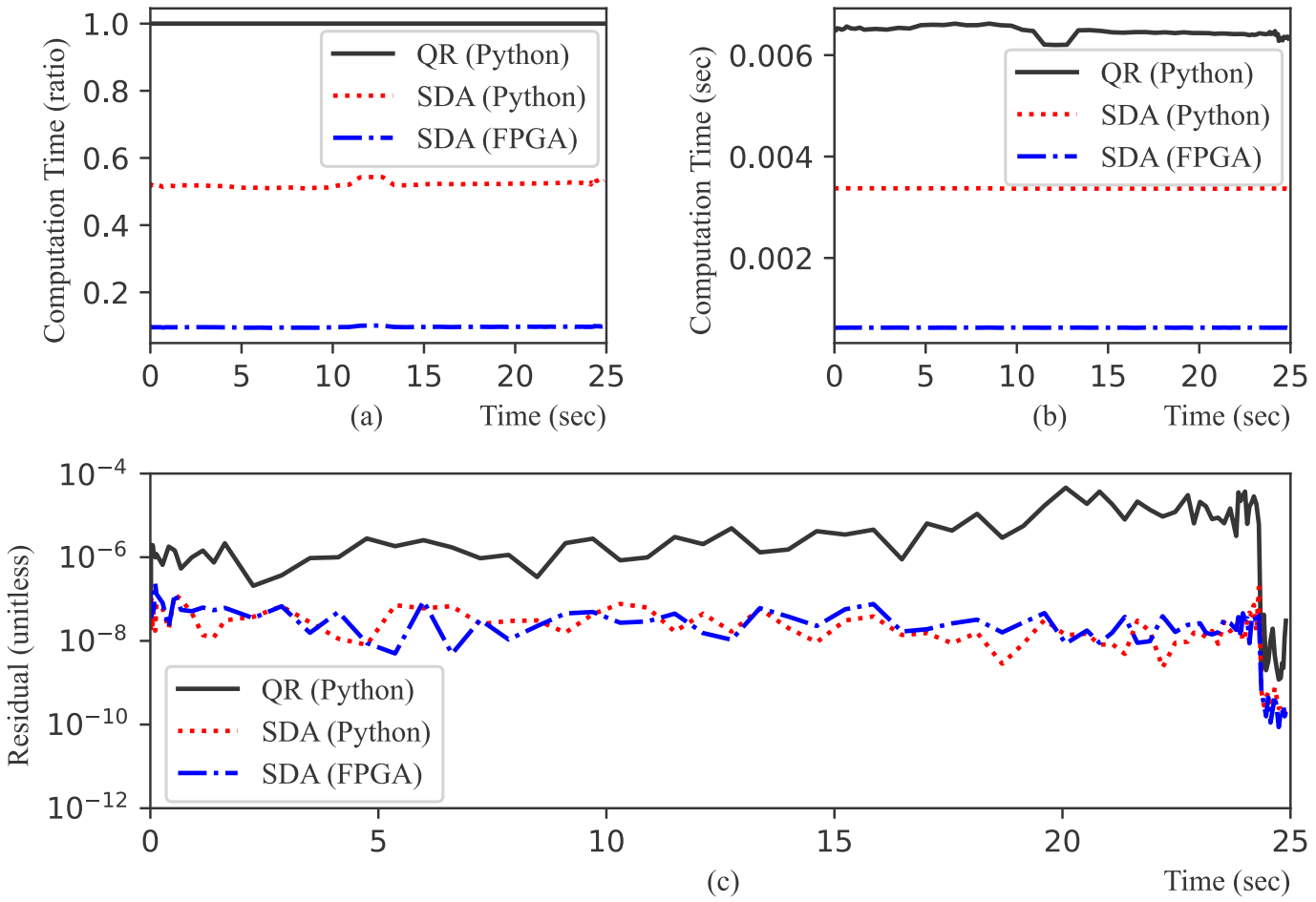

Fig. 4: Time histories of hardware enhancement in Sec. C focusing on the second major burden (pointwise ARE solving [3]), where (a) as well as its unnormalized version (b) (resp., (c)) plots the performance in solving time (resp., accuracy/residual) subject to microcontroller (Python-based) and FPGA platforms, while it compares the Python-implementation of SDA and its FPGA improvement [23] with the classic, QR-based, and Python-implemented approach [20] (normalization reference in (a)).

readers' convenience. In addition, the black (resp., blue) trajectory is always at the top (resp., bottom) throughout the horizon, and there is no overlapping among the three responses. In other words, the black (resp., blue) curve costs the most (resp., least) time in ARE solving at each instant. On average, the red line amounts to only 0.52, while the computation time is further decreased to merely 9% by virtue of FPGA enhancement (blue). Alternatively, the original/unnormalized statistics are also presented in Fig. 4 (b) so as to more realistically evaluate the proposed advantage. The computing platform via microcontroller yields, averagely, 6 (resp., 3) milliseconds using the QR method (resp., Python-implemented SDA), while the FPGA enhancement further reduces the solving time to merely 0.6 milliseconds. It is worth noting that the realistic data in Fig. 4 (b) also endorse another advantage of SDA, that is, low sensitivity in the solving time. To be more in-depth, the red as well as blue line remains almost constant across the time span, whereas there exist noticeable variations with respect to the black line. The advantage also manifests in other scenarios [3], such as the demonstrated interception condition in Sec. A, while it benefits hardware implementation in the light of various practical considerations, for instance, allocation of sampling time.

Simultaneously, in addition to the reduced solving time, the superior performance in solving accuracy is also observed in this experimental setting, as inferred by Fig. 4 (c). The numerics in Fig. 4 (c) are pointwise evaluated in terms of the Frobenius norm of the right-hand side (RHS) of Eq. (5), which associates with the underlying ARE in the SDDRE-solving process [3], [6]. Therefore, such a unitless metric – measuring how close the RHS of Eq. (5) approaches the zero matrix – is utilized to quantify the accuracy in ARE solving. As in the other subfigures of Fig. 4, it is also noted that there exists no intersection of the three lines and the black trajectory is still the highest throughout the horizon. Averagely speaking, the black response amounts to a residual of 6.8×10^{-6}, which is more than 200 times (less accurate) than the other solver on the same computing hardware, specifically, 3.1×10^{-8} for the red trajectory. Moving forward, the FPGA enhancement not just reduces the solving time as observed in Fig. 4 (a), but still maintains the higher accuracy concurrently; in other words, the averages of the blue and red curves are almost identical,

noting that there are many intersections around the residual of 10^{-8}. Such a low invariance across different computing platforms is valued as another merit of SDA, alongside the robust performance in solving time that is illustrated in Fig. 4 (b).

To further enrich Supplementary Material, on one hand, it is worth mentioning that all the experiments using SDA – numerical [17] and hardware (microcontroller and FPGA) – refrain from concerns about excessive memory space. The classic method [20], on the contrary, requires additional subroutines or libraries, to name a few, eigenvalue-reordering, double-shift, and dimension-increasing routines, which also partially explain the difficulty in hardware implementations, not to mention its rare appearance. In certain cases, such memory concerns extend to the entire MATLAB® software package along with required libraries [7, Sec. VI]. On the other hand, the SDA subject to various computing platforms allows users to directly adjust the tradeoff between solving time and accuracy. The positive impacts include that, for instance, a relaxed residual threshold renders early completion of the solving process. As a comparison, such a useful flexibility is not readily available in "`lqr.m`", let alone any possible hardware implementations. To our experiences, it has been an inconvenient design feature until the latest update, namely MATLAB® R2022b.

All of the above merits can be credited to pioneers around 1980s, to name one representative, [34], which enlighten a different research path in addition to the QR-based method [20] and thus broaden the spectrum of foundational control literature. Not until recent years, the latest update of such doubling algorithms is proposed and can be argued to be the most/well developed version, namely, SDA [23]. It is particularly customized for small and medium-sized engineering problems that involve continuous ARE (CARE), discrete ARE, or more general nonlinear matrix equations. Moreover, SDA inherits while refines other advantages in the early doubling algorithms, notably: 1) full exploitation of the special structure of Riccati equations, 2) global, monotonic, and quadratic convergence, 3) simultaneous convergence toward the positive semidefinite solutions to CARE and its dual, respectively, 4) simple implementation that only consists of matrix multiplications and divisions, and 5) about 5-8 iterations in the solving process, where each one costs $O(8n^3)$. As a comparison, the QR-based benchmark more significantly consumes $O(200n^3)$, which is mainly caused by: i) transformation from the original $n \times n$ dimension to the $2n \times 2n$ Hamiltonian matrix and ii) reordering and double-shift steps. From the perspectives of the above observations and evaluations, it is concluded that the SDA is efficiently more suitable than the classic method [20] to be integrated into the SDDRE-based IAG law [3], both numerically [17] and practically in hardware. In addition, the enhanced efficiency can be leveraged to benefit other real applications, beyond aerospace designs that value agile dynamics to more general Riccati-related nonlinear control systems, for example, motor servo system in the up-to-date [21] and the reference [31] therein.

Remark 4: (Concluding Observations) Supplementary Material overall provides a broader scope of validation spectrum for the computationally improved SDDRE-based lead angle guidance law [3], [6], where the main observations are:

- According to the average data in Table I, the *dominant* computational burden (`minreal.m`) is evaluated to be around 6.3 times more significant than the other/second burden (`lqr.m`). The former (numerical applicability checking) has been almost completely removed by virtue of Theorem 1, while the latter (ARE solving) is also largely alleviated thanks to the selected SDA solver – a reduction of around 90% and 98% overall. Given that the applicability using SDDRE has been ensured analytically and practically, the remaining major burden is further mitigated by means of FPGA (resp., Python-based microcontroller), which yields another notable 91% (resp., 48%) reduction as compared to the classic QR-based algorithm [7], [20], [21]. Even so, the performance in solving accuracy is simultaneously evaluated around 10^2 times better by using SDA than the classic algorithm, to the order. Remarkably, such hardware comparisons can be made more fair and meaningful, if the missing piece (namely, FPGA implementation of [20]) can be fulfilled.
- The same parameter setups are adopted in all the validations, specifically, $(\mathbf{S}, \mathbf{Q}, \mathbf{R})$ in the SDDRE scheme, which demonstrate the value in robustness that has been a critical design issue in the aerospace literature [28], [30].
- There are more performance criteria that can be benefited from the enhanced computational efficiency, to name one representative, the stability preservation. Noting that the stability property has been analyzed in the early guidance design [6], the proposed computational improvement serves as a practical prerequisite that leads to the less constrained sampling rate in real implementations [13] – not to mention the fundamental applicability via SDDRE that has been theoretically ensured and numerically verified.

TABLE I: Average computational reductions in Supplementary Material, where the MATLAB® validations correspond to Secs. A and B (Figs. 2 and 3); while the hardware experiments associate with Sec. C (Fig. 4) and are subject to microcontroller (Python-based) and FPGA implementations, respectively

MATLAB	Computation time[1]			
	`minreal.m`	`lqr.m`[2]	`sda.m`[3]	Reduction
Sec. A	6.29	1	0.1	98.5%
Sec. B	6.31	1	0.11	98.4%
Hardware	**Computation time[1]**			
	QR (Python)[2]	SDA (Python)[3]	SDA (FPGA)[3]	Reduction
Sec. C	1	0.52	0.09	48/91%

[1] Considering the MATLAB® (resp., hardware) validations, the computation time is normalized by the "`lqr.m`" (resp., "QR (Python)") routine.
[2, 3] The SDA [23] that solves the underlying ARE (noted by "3" in various platforms) is compared with the classic, QR-based method [20] ("2").