Enhancements to the DIDO© Optimal Control Toolbox

I. M. Ross*
Monterey, CA 93940

In 2020, DIDO© turned 20! The software package emerged in 2001 as a basic, user-friendly MATLAB® teaching-tool to illustrate the various nuances of Pontryagin's Principle but quickly rose to prominence in 2007 after NASA announced it had executed a globally optimal maneuver using DIDO. Since then, the toolbox has grown in applications well beyond its aerospace roots: from solving problems in quantum control to ushering rapid, nonlinear sensitivity-analysis in designing high-performance automobiles. Most recently, it has been used to solve continuous-time traveling-salesman problems. Over the last two decades, DIDO's algorithms have evolved from their simple use of generic nonlinear programming solvers to a more sophisticated employment of fast spectral Hamiltonian programming techniques. A description of the internal enhancements to DIDO that define its mathematics and algorithms are described in this paper. A challenge example problem from robotics is included to showcase how the latest version of DIDO is capable of escaping the trappings of a "local minimum" that ensnare many other trajectory optimization methods.

I. Introduction

On November 5, 2006, the International Space Station executed a globally optimal maneuver that saved NASA $1,000,000[1]. This historic event added one more first[2] to many of NASA's great accomplishments. It was also a first flight demonstration of DIDO©, which was then a basic optimal control toolbox. Because DIDO was based on a then new pseudospectral (PS) optimal control theory, the concept emerged as a big winner in computational mathematics as headlined on page 1 of *SIAM News*[3]. Subsequently, *IEEE Control Systems Magazine* did a cover story on the use of PS control theory for attitude guidance[4]. In exploiting the widespread publicity in the technical media (see Fig. 1) "new" codes based on naive PS discretizations materialized with varying degrees of disingenuous claims. Meanwhile, the algorithm implemented in DIDO moved well beyond its simple use of generic nonlinear programming solvers[5, 6] to a more sophisticated guess-free, spectral algorithm[7, 8]. That is, while the 2001-version of DIDO was indeed a simple code that patched PS discretization to a generic nonlinear programming solver, it quietly and gradually evolved to an entirely new approach that we simply refer to as ***DIDO's algorithm***. DIDO's algorithm is actually a suite of algorithms that work in unison and focused on three key performance elements:

1. Extreme robustness to the point that a "guess" is not required from a user[7];
2. Spectral acceleration based on the principles of discrete cotangent tunneling (see Theorem 1 in Appendix A of this paper); and
3. Verifiable accuracy of the computed solution in terms of feasibility and optimality tests[9].

Despite its current internal sophistication, the usage of DIDO from a user's perspective has remained virtually the same since 2001. This is because DIDO has remained true to its founding principle: to provide a minimalist approach to solving optimal control problems in a format that resembles writing the problem by hand ("pen-to-paper"). Once this is done, it should be the task of a good software to solve the problem as efficiently as possible with the least amount of iterative hand-tuning by the user. This is because it is more important for a user to focus on solving the "problem-of-problems" rather than expend wasteful energy on the minutia of coding and dialing knobs[9]. In showcasing this philosophy, we first describe the input features of DIDO that are as intuitive as formulating an optimal control problem in a standard format. The outputs of DIDO include the collection of necessary conditions generated by applying Pontryagin's Principle[9]. Because the necessary conditions are in the ***outputs*** of DIDO (and not its inputs!) a user may apply Pontryagin's Principle to "independently" test the optimality of the computed solution. These input-output features of DIDO are in accordance with Pontryagin's Principle whose fundamentals have largely remained unchanged since its inception, circa 1960[10]. Appendixes A and B describe the mathematical and algorithmic details of DIDO; however, no knowledge of its inner workings is necessary to employ the toolbox effectively. The only tool that is needed to use DIDO to its full capacity is a clear understanding of Pontryagin's Principle. This point is amplified in Sec. V of this paper by way of a challenge problem from robotics to show how DIDO can escape the trappings of a "local minimum" and emerge on the other side to present a viable solution.

II. DIDO Structures in a Generic Problem Formulation

A generic two-event[a] optimal control problem may be formulated as[b]

$$
\left.\begin{array}{ll} \mathbb{X} \subset \mathbb{R}^{N_x} & \mathbb{U} \subset \mathbb{R}^{N_u} \\ \boldsymbol{x} = (x_1, \ldots, x_{N_x}) & \boldsymbol{u} = (u_1, \ldots, u_{N_u}) \end{array}\right\} \textbf{\textit{preamble}}
$$

$$
\textbf{\textit{problem}}\ \overbrace{(\boldsymbol{G})}\ \left\{\begin{array}{ll}
\left.\begin{array}{l}\text{Minimize} \\ \quad J[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_0, t_f, \boldsymbol{p}] := \\ \quad\quad E(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f, \boldsymbol{p}) \\ \quad\quad + \displaystyle\int_{t_0}^{t_f} F(\boldsymbol{x}(t), \boldsymbol{u}(t), t, \boldsymbol{p})\, dt\end{array}\right\} & \textbf{\textit{cost}} \\
\text{Subject to} \quad \dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t, \boldsymbol{p}) & \textbf{\textit{dynamics}} \\
\boldsymbol{e}^L \le \boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f, \boldsymbol{p}) \le \boldsymbol{e}^U & \textbf{\textit{events}} \\
\boldsymbol{h}^L \le \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), t, \boldsymbol{p}) \le \boldsymbol{h}^U & \textbf{\textit{path}} \\
\boldsymbol{K}^L \le \boldsymbol{K}(\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t, \boldsymbol{p}) \le \boldsymbol{K}^U & \textbf{\textit{functional}}
\end{array}\right.
$$

The basic continuous-time optimization variables are packed in the DIDO structure called primal according to:

$$
\begin{aligned}
\text{primal.}\textbf{states} &\longleftarrow \boldsymbol{x}(\cdot) && \text{(1a)}\\
\text{primal.}\textbf{controls} &\longleftarrow \boldsymbol{u}(\cdot) && \text{(1b)}\\
\text{primal.}\textbf{time} &\longleftarrow t && \text{(1c)}
\end{aligned}
$$

*Author of the textbook, *A Primer on Pontryagin's Principle on Optimal Control*, Collegiate Publishers, San Francisco, CA, 2015.

[a] The two-event problem may be viewed as an "elementary" continuous-time traveling salesman problem[11].

[b] Although the notation used here is fairly self-explanatory, see [9] for additional clarity.

Additional optimization variables packed in primal are:

$$\text{primal.}\textbf{initial.states} \longleftarrow \boldsymbol{x}_0 \tag{2a}$$
$$\text{primal.}\textbf{initial.time} \longleftarrow t_0 \tag{2b}$$
$$\text{primal.}\textbf{final.states} \longleftarrow \boldsymbol{x}_f \tag{2c}$$
$$\text{primal.}\textbf{final.time} \longleftarrow t_f \tag{2d}$$
$$\text{primal.}\textbf{parameters} \longleftarrow \boldsymbol{p} \tag{2e}$$

The names of the five functions that define Problem $(\boldsymbol{G})$ are stipulated in the structure problem according to:

$$\text{problem.}\textbf{cost} \leftarrow [E(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f, \boldsymbol{p}), F(\boldsymbol{x}(t), \boldsymbol{u}(t), t, \boldsymbol{p})] \tag{3a}$$
$$\text{problem.}\textbf{dynamics} \leftarrow \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), t, \boldsymbol{p}) \tag{3b}$$
$$\text{problem.}\textbf{events} \leftarrow \boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f, \boldsymbol{p}) \tag{3c}$$
$$\text{problem.}\textbf{path} \leftarrow \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), t, \boldsymbol{p}) \quad (optional) \tag{3d}$$
$$\text{problem.}\textbf{functional} \leftarrow \boldsymbol{K}[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t, \boldsymbol{p}] \quad (optional) \tag{3e}$$

The bounds on the constraint functions of Problem $(\boldsymbol{G})$ are also associated with the structure problem but under its fieldname of **bounds** (i.e. problem.**bounds**) given by[c],

$$\text{bounds.}\textbf{events} \leftarrow [\boldsymbol{e}^L, \boldsymbol{e}^U] \tag{4a}$$
$$\text{bounds.}\textbf{path} \leftarrow [\boldsymbol{h}^L, \boldsymbol{h}^U] \quad (optional) \tag{4b}$$
$$\text{bounds.}\textbf{functional} \leftarrow [\boldsymbol{K}^L, \boldsymbol{K}^U] \quad (optional) \tag{4c}$$

Although it is technically unnecessary, a ***preamble*** is highly recommended and included in all the example problems that come with the full toolbox.[d] The preamble is just a simple way to organize all the primal variables in terms of symbols and notation that have a useful meaning to the user. In addition, the field **constants** is included in both primal and problem so that problem-specific constants can be managed with ease. Thus, problem.**constants** is where a user defines the data that can be accessed anywhere in the user-specific files by way of primal.**constants**.

A "search space" must be provided to DIDO using the field **search** under problem. The search space for states and controls are specified by

$$\text{search.}\textbf{states} \leftarrow [\boldsymbol{x}^{LL}, \boldsymbol{x}^{UU}] \tag{5a}$$
$$\text{search.}\textbf{controls} \leftarrow [\boldsymbol{u}^{LL}, \boldsymbol{u}^{UU}] \tag{5b}$$

where, $(\boldsymbol{x}, \boldsymbol{u})^{LL}$ and $(\boldsymbol{x}, \boldsymbol{u})^{UU}$ are nonactive bounds on the state and control variables; hence, ***they must be nonbinding constraints***.

The inputs to DIDO are simply two structures given by problem and algorithm. The algorithm input provides some minor ways to test certain algorithmic options. The reason the algorithm structure is not extensive is precisely this: To maintain the DIDO philosophy of not wasting time in fiddling with various knobs that only serves to distract a user from solving the problem. That is, it is not good-enough for DIDO to run efficiently; it must also provide a user a quick and easy approach in taking a problem from ***concept-to-code***[e].

Before taking a concept to code fast, it is critically important to perform a mathematical analysis of the problem; i.e., an investigation of the necessary conditions. This investigation is necessary (pun intended!) not only to understand DIDO's outputs but also to eliminate common conceptual errors that can easily be detected by a quick rote application of Pontryagin's Principle.

[c] Because some DIDO clones still employ the old DIDO format (e.g., "bounds.lower" and "bounds.upper"), the user-specific codes must be remapped to run on such imitation software.

[d] DIDOLite, the free version of DIDO, has limited capability.

[e] Warning: The quick and easy concept-to-code philosophy does not mean a user should code a problem first and ask questions later. DIDO is user-friendly, but it's not for dummies! Nothing beats analysis before coding; see Sec. 4.1.4 of [9] for a simple "counter" example and Sec. V of this paper for the recommended process.

Figure 1. Sample media coverage of NASA's successful flight operations using PS optimal control theory as implemented in DIDO.

Although proving Pontryagin's Principle is quite hard, its application to any problem is quite easy! See [9] for details on the "HAMVET" process. The first step in this process is to construct the Pontryagin Hamiltonian. For Problem $(\boldsymbol{G})$ this is given by[f],

$$H(\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) := F(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) + \boldsymbol{\lambda}^T \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) \tag{6}$$

That is, constructing (6) is as simple as taking a dot product. Forming the ***Lagrangian of the Hamiltonian*** requires taking another dot product given by,

$$\overline{H}(\boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) := H(\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) + \boldsymbol{\mu}^T \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) \tag{7}$$

where, $\boldsymbol{\mu}$ is a path covector (multiplier) that satisfies the ***complementarity conditions***, denoted by $\boldsymbol{\mu} \dagger \boldsymbol{h}$, and given by,

$$\boldsymbol{\mu} \dagger \boldsymbol{h} \Leftrightarrow \mu_i \begin{cases} \leq 0 & h_i(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) = h_i^L \\ = 0 & \text{if} \quad h_i^L < h_i(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) < h_i^U \\ \geq 0 & h_i(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) = h_i^U \\ unrestricted & h_i^L = h_i^U \end{cases} \tag{8}$$

Constructing the ***adjoint equation*** now requires some calculus of generating gradients:

$$-\dot{\boldsymbol{\lambda}} = \frac{\partial \overline{H}}{\partial \boldsymbol{x}} \tag{9}$$

The adjoint covector (costate) satisfies the ***tranversality conditions*** given by,

$$-\boldsymbol{\lambda}(t_0) = \frac{\partial \overline{E}}{\partial \boldsymbol{x}_0} \qquad \boldsymbol{\lambda}(t_f) = \frac{\partial \overline{E}}{\partial \boldsymbol{x}_f} \tag{10}$$

where, $\overline{E}$ is the ***Endpoint Lagrangian*** constructed by taking yet another dot product given by,

$$\overline{E}(\boldsymbol{\nu}, \boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f, \boldsymbol{p}) := E(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f, \boldsymbol{p}) + \boldsymbol{\nu}^T \boldsymbol{e}(\boldsymbol{x}_0, \boldsymbol{x}_f, t_0, t_f, \boldsymbol{p}) \tag{11}$$

and $\boldsymbol{\nu}$ is the endpoint covector that satisfies the complementarity condition $\boldsymbol{\nu} \dagger \boldsymbol{e}$ (see (8)). The values of the Hamiltonian at the endpoints also satisfy transversality conditions known as the ***Hamiltonian value conditions*** given by,

$$\mathcal{H}[@t_0] = \frac{\partial \overline{E}}{\partial t_0} \qquad -\mathcal{H}[@t_f] = \frac{\partial \overline{E}}{\partial t_f} \qquad (\text{and } \boldsymbol{\nu} \dagger \boldsymbol{e}) \tag{12}$$

[f] We ignore the functional constraints ($\boldsymbol{K}[\cdot]$) in the rest of this paper for simplicity of presentation.

where, $\mathcal{H}$ is the ***lower Hamiltonian***[9, 12]. The ***Hamiltonian minimization condition***, which is frequently and famously stated as

$$\text{(HMC)} \quad \begin{cases} \underset{\boldsymbol{u}}{\text{Minimize}} & H(\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) \\ \text{Subject to} & \boldsymbol{u} \in \mathbb{U} \end{cases} \tag{13}$$

reduces to the function-parameterized nonlinear optimization problem given by,

$$\text{(HMC for (G))} \quad \begin{cases} \underset{\boldsymbol{u}}{\text{Minimize}} & H(\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) \\ \text{Subject to} & \boldsymbol{h}^L \leq \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{u}, t, \boldsymbol{p}) \leq \boldsymbol{h}^U \end{cases} \tag{14}$$

The necessary conditions for the subproblem stated in (14) are given by the Karush-Kuhn-Tucker (KKT) conditions:

$$\frac{\partial \overline{H}}{\partial \boldsymbol{u}} = \boldsymbol{0} \qquad\qquad \boldsymbol{\mu} \dagger \boldsymbol{h} \tag{15}$$

Similarly, the necessary condition for selecting an optimal parameter $\boldsymbol{p}$ is given by,

$$\frac{\partial \overline{E}}{\partial \boldsymbol{p}} + \int_{t_0}^{t_f} \left(\frac{\partial \overline{H}}{\partial \boldsymbol{p}} \right) dt = \boldsymbol{0} \tag{16}$$

Finally, the ***Hamiltonian evolution equation***,

$$\frac{d\mathcal{H}}{dt} = \frac{\partial H}{\partial t} \tag{17}$$

completes the HAMVET process defined in [9].

All of these necessary conditions can be easily checked by running DIDO using the single line command,

$$[\mathsf{cost}, \mathsf{primal}, \mathsf{dual}] = \textbf{dido}(\mathsf{problem}, \mathsf{algorithm}) \tag{18}$$

where cost is the (scalar) value of J, primal is as defined in (1) and (2), and dual is the ***output*** structure of DIDO that packs the entirety of the dual variables in the following format:

$$\text{dual.}\textbf{Hamiltonian} \longleftarrow \mathcal{H}[@t] \tag{19a}$$
$$\text{dual.}\textbf{dyanmics} \longleftarrow \boldsymbol{\lambda}(\cdot) \tag{19b}$$
$$\text{dual.}\textbf{events} \longleftarrow \boldsymbol{\nu} \tag{19c}$$
$$\text{dual.}\textbf{path} \longleftarrow \boldsymbol{\mu}(\cdot) \tag{19d}$$

Note that the necessary conditions are not supplied to DIDO; rather, the code does all of the hard work in generating the dual information. It is the task of the user to check the optimality of the computed solution, and potentially discard the answer should the candidate optimal solution fail the test.

III. Overview of DIDO's Algorithm

From Secs. I and II, it is clear that DIDO interacts with a user in much the same way as a direct method but generates outputs similar to an indirect method. In this regard, ***DIDO is both a direct and an indirect method***. *Alternatively, DIDO is neither a direct nor an indirect method!* To better understand the last two statements and DIDO's unique suite of algorithms, recall from Sec. II that the totality of variables in an optimal control problem are,

$$\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_0, t_f, \boldsymbol{p}, \boldsymbol{\lambda}(\cdot), \boldsymbol{\mu}(\cdot), \text{ and } \boldsymbol{\nu}$$

Internally, DIDO introduces two more function variables $\boldsymbol{v}(\cdot)$ and $\boldsymbol{\omega}(\cdot)$ by rewriting the dynamics and adjoint equations as,[g]

$$\dot{\boldsymbol{x}} = \boldsymbol{v}; \qquad \boldsymbol{v} = \partial_{\boldsymbol{\lambda}} \overline{H} \tag{20a}$$
$$-\dot{\boldsymbol{\lambda}} = \boldsymbol{\omega}; \qquad \boldsymbol{\omega} = \partial_{\boldsymbol{x}} \overline{H} \tag{20b}$$

Thus, all nonlinear differential equations are split into linear differentials and nonlinear algebraic components at the price of adding two new variables. The computational cost for adding these two variables is quite cheap; see Sec. VI.C for details. The linear components are handled separately and efficiently through the use of $N \in \mathbb{N}$ Birkhoff basis functions[13, 14, 15, 16, 17]. DIDO then focuses on solving the nonlinear equations with the linear equations tagging along for feasibility. Of all the nonlinear equations, special attention is paid to the feasibility of the equations resulting from the Hamiltonian minimization condition; see (13). This is because the gradient of the cost function is damped by a "step size" when associated with the gradient of the Hamiltonian with respect to $\boldsymbol{u}$[18]. Thus, unlike nonlinear programming methods that treat all variables the same way, DIDO treats different variables differently. The differences in treatment is a product of the fact that a nonlinear programming algorithm is based on equations generated by a Lagrangian while DIDO generates processes that are centered around a Hamiltonian. Because a Hamiltonian is not a Lagrangian, a state variable is treated differently than a control variable and an adjoint covector function (costate) is handled differently than a path covector function. These differences in treatments of different functions and constraints are in sharp contrast to nonlinear programming methods where all variables and functions are treated identically.[h] The mathematical details of this computational theory are described in Appendix A.

In addition to treating the optimal control variables differently, DIDO also treats the Birkhoff pseudospectral problem differently for different values of N and the "state" of the iteration. As noted in Sec. I, DIDO's algorithm is actually a suite of algorithms. There are three major algorithmic components to DIDO's main algorithm. A schematic that explains the three components is shown in Fig. 2. The basic idea behind the three-component formula is to address the three different objectives of DIDO, namely, to be ***guess-free, fast and accurate***.[i] The

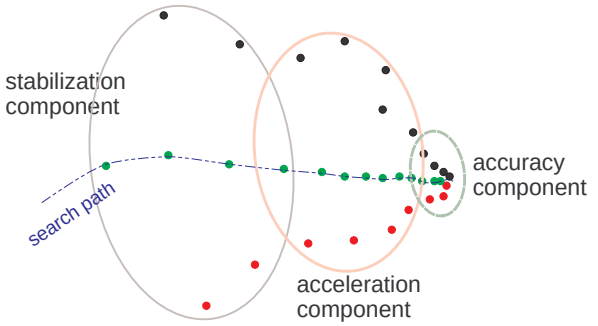

Figure 2. A schematic for the concept of the three main algorithmic components of DIDO.

functionality of the three major components of DIDO are defined as follows:

1. The stabilization component: The task of this component of the algorithm is to drive an "arbitrary" point to an "acceptable" starting point for the acceleration component.

2. The acceleration component: The task of this suite of algorithms is to rapidly guide the sequence of functional iterates to a capture zone of the accurate component.

3. The accurate component: The task of this component is to generate and refine a solution that satisfies the precision requested by the user.

DIDO's main algorithm ties these three components together by monitoring their progress (i.e., "state" of iteration) and triggering automatic switching between components based on various criteria. Details of all three components of DIDO are described in Appendix B.

[g] This rewriting is done after a domain transformation; see Appendix A for details.

[h] Although nonlinear programming algorithms may treat linear and nonlinear constraints differently, this concept is agnostic to the differences betweens state and control variables.

[i] The word fast is used here in the sense of true-fast; i.e. fast that is agnostic to the specifics of computer implementations. A more precise meaning of fast is defined in Appendix B.

IV. Best Practices in Using DIDO

Despite the multitude of internal algorithms that comprise DIDO's algorithm, ***the only concept that is needed to use DIDO effectively is Pontryagin's Principle[9]***. That is, the details of DIDO's algorithms are quite unnecessary to use DIDO effectively. Best practices in producing a good DIDO application code in the least amount of time are:

1. Using Pontryagin's Principle to generate the necessary conditions for a given problem as outlined in Sec. II.
2. Determining if the totality of necessary conditions are free of concept/computational problems; for example, square roots, division by zero etc. If yes, then reformulating the problem as discussed in [9], Chs. 1 and 3.
3. Identifying the subset of the necessary conditions that can be easily checked. Examples are constant/linear costate predictions, switching conditions etc.
4. Scaling and balancing the equations. A detailed process for scaling and balancing is described in [19].

If a problem is not well-balanced, DIDO will execute several subalgorithms to generate an extremal solution. This will effectively bypass its main acceleration component. ***Tell-tale signs of imbalanced equations are if the costates and/or Hamiltonian values are very large or very small; e.g.,*** $10^{\pm 9}$.

Once DIDO generates a candidate solution, it is critical for a user to ***verify and validate (V&V)*** the computed solution. Best practices for post computation are:

1. Independently testing the feasibility of the computed solution. If the solution is not feasible, it is not optimal!
2. Testing the optimality of the computed solution using the dual outputs generated by DIDO.

DIDO also contains a number of other computational components such as the DIDO Doctor Toolkit™ to assist the user in avoiding certain common mistakes in coding and problem formulation. Additional features continue to be added, particularly when new theoretical advancements are made and/or new improvements in computational procedures are possible.

V. Solving A Challenge Problem from Robotics

One of the earliest applications of DIDO that went beyond its aerospace boundaries were problems in robotics[20]. A motion planning problem with obstacle avoidance can easily be framed as an optimal control problem with path constraints[21]. A particular time-optimal motion planning problem for a ***differential-drive robot*** can be framed as:

$$
\begin{array}{lll}
\mathbb{X} = \mathbb{R}^3 & \mathbb{U} = \mathbb{R}^2 & \left.\vphantom{\begin{array}{l}a\\a\end{array}}\right\} \textbf{\textit{preamble}} \\
\boldsymbol{x} = (x, y, \theta) & \boldsymbol{u} = (\omega_r, \omega_l) & \\
\end{array}
$$

$$
\underbrace{\text{problem}}_{(R)} \left\{ \begin{array}{lll}
\text{Minimize} & J[\boldsymbol{x}(\cdot), \boldsymbol{u}(\cdot), t_0, t_f] & \left.\vphantom{\begin{array}{l}a\\a\end{array}}\right\} \textbf{\textit{cost}} \\
 & \quad := t_f - t_0 & \\
\text{Subject to} & \dot{x} = \dfrac{\cos\theta}{2}(\omega_r + \omega_l) & \left.\vphantom{\begin{array}{l}a\\a\\a\end{array}}\right\} \textbf{\textit{dynamics}} \\
 & \dot{y} = \dfrac{\sin\theta}{2}(\omega_r + \omega_l) & \\
 & \dot{\theta} = c(\omega_r - \omega_l) & \\
 & \boldsymbol{x}(0) = (x^0, y^0, \theta^0) & \left.\vphantom{\begin{array}{l}a\\a\end{array}}\right\} \textbf{\textit{events}} \\
 & \boldsymbol{x}(t_f) = (x^f, y^f, \theta^f) & \\
(x - x_i)^2 + (y - y_i)^2 - r_i^2 \geq 0 & & \left.\vphantom{\begin{array}{l}a\\a\\a\end{array}}\right\} \textbf{\textit{path}} \\
i = 1, \ldots, N_{obs} & & \\
\omega^L \leq \omega_{r,l} \leq \omega^U & &
\end{array} \right.
$$

where, $\boldsymbol{x} := (x, y, \theta)$ is the state of a differential-drive robot, $\boldsymbol{u} = (\omega_r, \omega_l)$ is the control and c is a constant[22]. Obstacles the robot must avoid during navigation are modeled as circular path constraints[20] of radius R_i centered at (x_i, y_i), $i = 1, \ldots, N_{obs}$. The robot itself has a "bubble" of radius R_0; hence, $r_i := R_i + R_0$.

Although it is very tempting to take the posed problem and code it up in DIDO, this is highly ***not recommended*** as noted in Sec. II. This is because *DIDO is not an application software*; rather, it is a mathematical toolbox for solving a generic optimal control problem (see Problem (G) posed in Sec. I). Thus, a user is creating an "app" using DIDO as the "operating system." Consequently, an analysis of the posed optimal control problem is ***strongly recommended*** before coding. This advice is in sharp contrast to the brute-force advocacy of "nonlinear programming methods for optimal control."

A. Step 1: Pre-Code Analysis

A pre-code analysis is essentially an analysis of the posed problem using Pontryagin's Principle[9]. In following the "HAMVET" procedure enunciated in Sec. II, we first construct the Pontryagin Hamiltonian,

$$
H(\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{u}) := \frac{\lambda_x \cos\theta + \lambda_y \sin\theta}{2}(\omega_r + \omega_l) + c\lambda_\theta(\omega_r - \omega_l) \quad (21)
$$

where $\boldsymbol{\lambda} := (\lambda_x, \lambda_y, \lambda_\theta)$ is an adjoint covector (costate). The Lagrangian of the Hamiltonian is given by,

$$
\overline{H}(\boldsymbol{\mu}, \boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{u}) := H(\boldsymbol{\lambda}, \boldsymbol{x}, \boldsymbol{u}) + \sum_{i=1}^{N_{obs}} \mu_i (x - x_i)^2 + (y - y_i)^2 + \mu_r \omega_r + \mu_l \omega_l \quad (22)
$$

where, $\boldsymbol{\mu} := (\mu_1, \ldots, \mu_{N_{obs}}, \mu_r, \mu_l)$ satisfies the complementarity conditions,

$$
\mu_i \left\{ \begin{array}{lll} \leq 0 & \text{if} & (x - x_i)^2 + (y - y_i)^2 = r_i^2 \\ = 0 & \text{if} & (x - x_i)^2 + (y - y_i)^2 > r_i^2 \end{array} \right. \quad (23a)
$$

$$
\mu_{r,l} \left\{ \begin{array}{lll} \leq 0 & \text{if} & \omega_{r,l} = \omega^L \\ = 0 & \text{if} & \omega^L < \omega_{r,l} < \omega^U \\ \geq 0 & \text{if} & \omega_{r,l} = \omega^U \end{array} \right. \quad (23b)
$$

The adjoint equations are now given by,

$$
-\dot{\lambda}_x := \frac{\partial \overline{H}}{\partial x} = \sum_{i=1}^{N_{obs}} 2\mu_i (x - x_i) \quad (24a)
$$

$$
-\dot{\lambda}_y := \frac{\partial \overline{H}}{\partial y} = \sum_{i=1}^{N_{obs}} 2\mu_i (y - y_i) \quad (24b)
$$

$$
-\dot{\lambda}_\theta := \frac{\partial \overline{H}}{\partial \theta} = \frac{-\lambda_x \sin\theta + \lambda_y \cos\theta}{2}(\omega_r + \omega_l) \quad (24c)
$$

Hence, from (24) and (23a), we get,

$$
\boxed{(\dot{\lambda}_x, \dot{\lambda}_y) = (0, 0) \quad \text{if } \mu_i = 0 \; \forall \; i = 1, \ldots, N_{obs}} \quad (25)
$$

Equation (25) is thus a special necessary condition for Problem (R). Any claim of an optimal solution to Problem (R) must be supported by (25) for mathematical legitimacy. *Hence, before writing the first line of code, we expect DIDO to generate constant co-position trajectories* $t \mapsto (\lambda_x, \lambda_y)$ *whenever the robot is sufficiently far from all obstacles.* Furthermore, over regions where $\mu_i \neq 0$, we expect $t \mapsto (\lambda_x, \lambda_y)$ to be different. More importantly, because $t \mapsto \boldsymbol{\mu}$ is allowed to be a Dirac delta function (see [9] for details) the co-position trajectories may jump instantaneously. In addition, because $\mu_i(t) \leq 0$, the jumps (if any) in the co-position trajectories are always positive (upwards) if $x(t) > x_i$ for $\lambda_x(t)$ and $y(t) > y_i$ for $\lambda_y(t)$. *Hence, before any code is written, we expect DIDO to generate constant co-position trajectories with possible up-jumps (or down-jumps).* Note that these are ***all necessary conditions*** for optimality. Stated differently, any candidate solution

that cannot satisfy these conditions is certainly not optimal. Consequently, when optimality claims are made on computed solutions without a demonstration of such necessary conditions, it is questionable, at the very least.

Next, from the Hamiltonian minimization condition, we get,

$$\frac{\lambda_x \cos\theta + \lambda_y \sin\theta}{2} + c\lambda_\theta + \mu_r = 0 \tag{26a}$$

$$\frac{\lambda_x \cos\theta + \lambda_y \sin\theta}{2} - c\lambda_\theta + \mu_l = 0 \tag{26b}$$

If $\mu_r(t) = 0$ or $\mu_l(t) = 0$ over a nonzero time interval, then (26) must be analyzed for the existence of singular arcs. ***If singular arcs are absent***, then, from (23b) we get the condition that $t \mapsto (\omega_r, \omega_l)$ must be "bang-bang" and in accordance with the "law:"

$$\boxed{\omega_{r,l}(t) = \begin{cases} \omega^L & \text{if} \quad \mu_{r,l}(t) \le 0 \\ \omega^U & \text{if} \quad \mu_{r,l}(t) \ge 0 \end{cases}} \tag{27}$$

Finally, combining the Hamiltonian value condition with the Hamiltonian evolution equation, we get the condition,

$$\boxed{\mathcal{H}[@t] = -1} \tag{28}$$

B. Step 2: Scaling/Balancing and Setting up the "Problem of Problems"

The second critical preparatory step in setting up a problem for DIDO is scaling and balancing the equations[19]. This aspect of the the problem set up is discussed extensively in [19] and the reader is strongly encouraged to read and understand the principles and practices presented in this article. At this point, we simply note that it is often advantageous to "double-scale" a problem. That is, the first scaling is done "by-hand" to reduce the number of design parameters (constants) associated with a problem[9]. Frequently, canonical scaling achieves this task[19]. The second scaling is done numerically to balance the equations[19].

Once a problem is scaled to its lowest number of constants, it is worthwhile to set up a "problem of problems"[9] even if the intent is to solve a single problem. To better explain this statement, observe that although Problem (R) is a specific optimal control problem in the sense of its specificity with respect to Problem (G) defined in Sec. I, one can still create a menu of Problem $(R's)$ by changing the number of obstacles N_{obs}, the location of the obstacles (x_i, y_i), the size of the obstacles R_i, the size of the robot R_0, the exact specification of the boundary conditions $(\boldsymbol{x}^0, \boldsymbol{x}^f)$ etc. This is the problem of problems. The pre-code analysis and the necessary conditions derived in Sec. V.A are agnostic to these changes in the data. Furthermore, once a DIDO-application-specific code is set up, it is easy to change such data and generate a large volume of plots and results. This volume of data can be reduced if the problem is first scaled by hand to reduce the number of constants down to its lowest level[9]. In any case, the new challenge generated by DIDO-runs is a process to keep track of all these variations. This problem is outside the scope of this paper. The problem that is in scope is a challenge problem, posed by D. Robinson[23]. Robinson's challenge comprises a clever choice of boundary conditions and the number, location and size of the obstacles. The challenge problem is illustrated in Fig. 3.

As shown in Fig. 3, two obstacles are placed in a figure-8 configuration and the boundary conditions for the robot are on opposite sides of the composite obstacle. The idea is to "trick" a user in setting up a guess in the straight line path shown in blue. In using this guess, an optimal control solver would get trapped by straight line solutions and would claim to generate an optimal solution by "jumping" over the single intersection point of the figure-8 configuration. Attempts at "mesh refinement" would simply redistribute the mesh so that the discrete solution would continue to jump across the single obstacle point in the middle. According to Robinson, all optimal control codes failed to solve this problem with such an initialization.

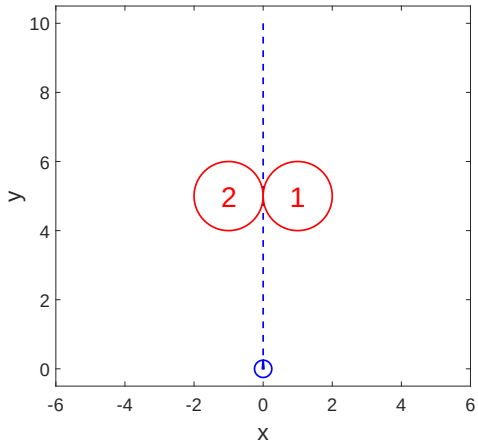

Figure 3. "Trick" boundary conditions and obstacles set up for Problem (R).

C. Step 3: Generating DIDO's Guess-Free Solution

Because DIDO is guess-free, it requires no initialization. Thus, one only defines the problem as stated and runs DIDO. It is clear from Fig. 4 that DIDO was able to provide the robot a viable path to navigate around the obstacles. At this point, one can simply declare success

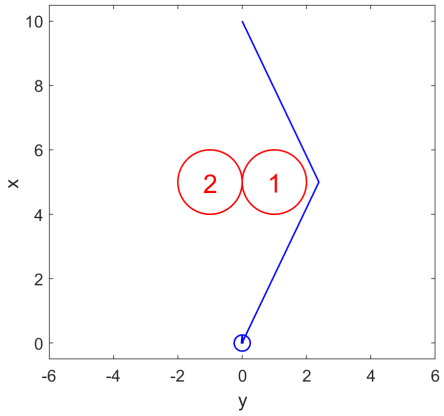

Figure 4. DIDO-solution to Problem (R) obtained via its default guess-free mode.

relative to the posed challenge and "collect the reward." Nonetheless, in noting that DIDO is purportedly capable of solving optimal control problems, it is convenient to use Problem (R) to illustrate how the "path" shown in Fig. 4 may have actually solved the stated problem. In effect, there are (at least) two steps in this ***verification and validation*** process (known widely as ***V&V***):

1. Using a process that is independent of the optimization, show that the computed solution is feasible; and
2. Show that the necessary conditions (as many as possible[j]) are satisfied.

In engineering applications, Step 1 is quite critical for safety, reliability and success of operations. Furthermore, it is also a critical feature of any numerical analysis procedure to "independently" check the validity of a solution.

D. Step 4: Producing an Independent Feasibility Test

To demonstrate the independent feasibility of a candidate solution to an optimal control problem, we simply take the computed control tra-

[j] See, for example, (25), (27) and (28).

jectory $t \mapsto \boldsymbol{u}^\sharp$ and propagate the initial conditions through the ODE,

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}^\sharp(t)), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}^0 \tag{29}$$

Let $t \mapsto \boldsymbol{x}^\sharp$ be the propagated trajectory; i.e., one obtained by solving the initial value problem given by (29). Then, $\boldsymbol{x}^\sharp(\cdot)$ is the truth solution that is used to test if other constraints (e.g. terminal conditions, path constraints etc.) are indeed satisfied.

In performing this test for Problem (R), we first need the DIDO-computed control trajectory. This is shown in Fig. 5. It is clear that both

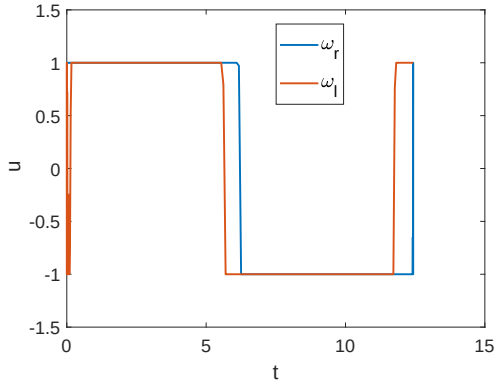

Figure 5. Candidate DIDO control trajectory to Problem (R).

controls take on "bang-bang" values with no apparent singular arcs.

Remark 1 *It is abundantly clear from Fig. 5 that DIDO is easily able to represent discontinuous controls. Consequently, the frequent lamentation in the literature that PS methods cannot represent discontinuities in controls is quite false.*

Using $t \mapsto (\omega_r, \omega_l)$ shown in Fig. 5 to propagate the initial conditions through the "dynamics" of the differential drive robot generates the solution shown in Fig. 6. The independent propagator used to generate Fig. 6 was ode45[k] in MATLAB®. Also shown in Fig. 6 are the prop-

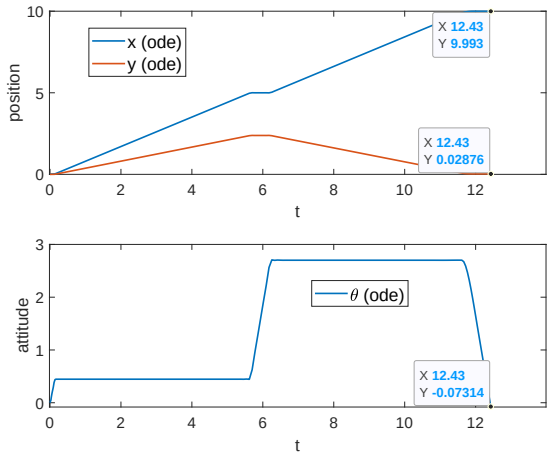

Figure 6. Propagated state trajectory using controls from Fig. 5.

agated values of the terminal states. The errors in the terminal values of the propagated states are the "true" errors. They are significantly higher than those obtained by the optimization process as indicated in Table 1. The error values generated by any optimization process (including DIDO) must be viewed with a significant amount of skepticism because they do not reflect reality. Despite this apparently obvious statement, outrageously small errors that are not grounded in basic

[k] Different propagators can generate different results; this aspect of solving (29) is not taken into account in the ensuing analysis.

	x	y	θ
Target	10.0000	0.0000	0.0000
DIDO (optimization)	10.0000	0.0000	0.0000
Propagated (true)	09.993	0.02876	-0.07314

Table 1. True accuracy vs optimization errors

science or computational mathematics are frequently reported. For instance, in [24], a minimum time of 324.9750302 seconds is reported (pg. 146). The purported accuracy of this number is ten significant digits. Furthermore, a clock to measure this number must have a range of a few hundred seconds and an accuracy of greater than one microsecond! To better understand the meaning of "small" errors (reasonable vs outrageous), recall that a scalar control function $u(\cdot)$ is computationally represented as a vector, $U^N := (u_0, u_1, \ldots, u_N)$. Let $U^N_\star$ be the optimal value of U^N for a fixed N. Then, a Taylor-series expansion of the cost function $U^N \mapsto J^N$ around $U^N_\star$ may be written as,

$$\begin{aligned} J^N(U^N) = J^N(U^N_\star) + \partial J^N(U^N_\star) \cdot (U^N - U^N_\star) \\ + (1/2)(U^N - U^N_\star) \cdot \partial^2 J^N(U^N_\star) \cdot (U^N - U^N_\star) \\ + \text{higher order terms} \end{aligned} \tag{30}$$

Setting $\partial J^N(U^N_\star) = 0$ in (30), we get,

$$\begin{aligned} 2\left|J^N(U^N) - J^N(U^N_\star)\right| = \left|\widehat{U} \cdot \partial^2 J^N(U^N_\star) \cdot \widehat{U}\right| \left\|U^N - U^N_\star\right\|^2 \\ + \text{higher order terms} \end{aligned} \tag{31}$$

where $\widehat{U}$ is a unit vector. In a well conditioned system, the Hessian term in (31) is unity; hence, in this best-case scenario we have[l],

$$\|U^N - U^N_\star\| \sim \mathcal{O}\left(\sqrt{\left|J^N(U^N) - J^N(U^N_\star)\right|}\right) \tag{32}$$

Let ϵ_M be the machine precision. Then from (32), we can write,

$$\|U^N - U^N_\star\| > \mathcal{O}\left(\sqrt{\epsilon_M}\right) \sim 10^{-8} \tag{33}$$

where, the numerical value in (33) is obtained by setting $\epsilon_M \sim 10^{-16}$ (double-precision, floating-point arithmetic).

The vector $U^N_\star$ is only an "N^{th} degree" approximation to the optimal function $u(\cdot)$. The "best" convergence rate in approximation of functions is exponential[26]. This record is held by PS approximations[26, 27, 28] because they have near-exponential or "spectral" rates of convergence (assuming no segmentation, no knotted concatenation of low-order polynomials for mesh-refinement[6, 29], perfect implementation etc.). In contrast, Runge-Kutta rates of approximation are typically[30] only $\mathcal{O}(4)$ or $\mathcal{O}(5)$ (fourth-order or fifth-order). Assuming convergence to within a digit (very optimistically!) we get,

$$\left\|U^N(t) - u(t)\right\| > 10 \times \mathcal{O}\left(\sqrt{\epsilon_M}\right) \sim 10^{-7} \tag{34}$$

where, we have abused notation for clarity in using $U^N(t)$ to imply an interpolated function. In practice, the error may be worse than 10^{-7} (seven significant digits) if the derivatives of the data functions in a given problem (gradient/Jacobian) are computed by finite differencing. Of course, none of these errors take modeling errors into account! *An independent feasibility test is therefore a quick and simple approach for estimating the true feasibility errors of an optimized solution. Any errors reported without such a test must be viewed with great suspicion!*[m]

[l] Equation (32) is well-known in optimization; see, for example [25].

[m] It is possible for the errors given by (32) (and hence (34)) to be smaller in certain special cases.

Given the arguments of the preceding paragraphs, it is apparent that a "very small" error that qualifies for the term "very accurate" control may be written as,

$$\|U^N(t) - u(t)\| \sim 10^{-6} \tag{35}$$

Consequently, achieving an accuracy for $u(t)$ beyond six significant digits in a generic problem is highly unlikely. Furthermore, recall that the estimate given by (35) is based on optimistic intermediate steps! Therefore, practical errors may be worse than 10^{-6}. Regardless, to achieve high accuracy, the problem must be well conditioned. To achieve better conditioning, the problem must be properly scaled and balanced regardless of which optimization software is used. See [19] for details. Providing DIDO a well-scaled and balanced problem also triggers the best of its true-fast suite of algorithms.

Remark 2 *The optimistic error estimate given by* (34) *or* (35) *does not mean greater accuracy is not possible. The main point of* (35) *is to treat "accuracy" claims of computed solutions with great caution. It is also important to note that the practical limit given by* (35) *does not preclude the possibility of achieving highly precise solutions. See, for example, [31] where accuracies of the order of* 10^{-7} *radians were achieved in pointing the Kepler space telescope.*

E. Step 5: Testing for Optimality

This step closes the loop with Step 1. In the pre-code analysis step, we noted (see (25)) that the co-position trajectories must be constants with potential jumps. Recall that (see (18)) DIDO automatically generates all the duals as part of its outputs. Thus, the dual variables that are of interest in testing (25) is contained in dual.**dynamics**. A plot of the costate trajectories generated by DIDO is shown in Fig. 7. It is

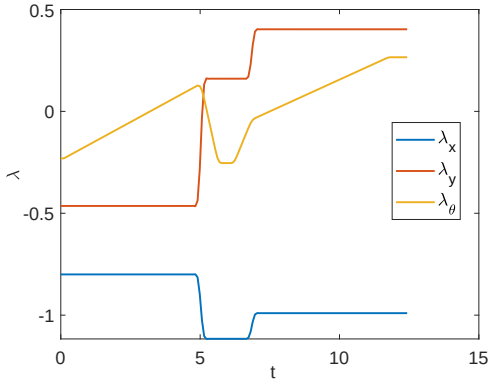

Figure 7. DIDO-generated costate trajectories for Problem (R)**.**

apparent from this figure that the co-position trajectories $t \mapsto (\lambda_x, \lambda_y)$ are indeed constants with jumps over two spots. If t_c is the time when a jump occurs, then it follows that $\lambda_y(t)$ must jump upwards because $y(t_c) > y_i$ where t_c is the point in time when the robot "bubble" just touches the obstacle bubble. In the case of the x-coordinates, we have $x(t_c) < x_i$ at the first touch of the bubbles; see Fig. 4. Hence, $\lambda_x(t)$ must jump downward, which it indeed does in Fig. 7. Subsequently, at the second touch of the bubbles $x(t_c) > x_i$; hence, $\lambda_x(t)$ jumps upwards.

Because DIDO also outputs $\boldsymbol{\mu}(\cdot)$ in dual.**path**, it is a simple matter to test (27). The path covector trajectory $\mu_l(\cdot)$ is plotted in Fig. 8 over the graph of $\omega_l(\cdot)$. It is clear from this plot that $\mu_l(\cdot)$ is indeed a switching function for $\omega_l(\cdot)$. The same holds for ω_r and μ_r. This plot is omitted for brevity.

Because there are two obstacles in the challenge problem (modeled as two inequalities), dual.**path** also contains $\mu_1(\cdot)$ and $\mu_2(\cdot)$ that satisfy the complementarity conditions given by (23a). It is apparent from the candidate solution shown in Fig. 4 that if this trajectory is optimal, we require $\mu_2(t) = 0$ over the entire time interval $[t_0, t_f]$. That this is indeed true is shown in Fig. 9. Also shown in Fig. 9 is the path covector function $\mu_1(\cdot)$ generated by DIDO. Note first that $\mu_1(t) \leq 0 \; \forall \; t$ as re-

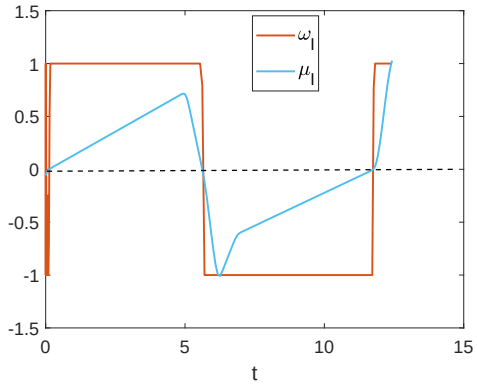

Figure 8. Verification of (27) **for the** $(\omega_l(\cdot), \mu_l(\cdot))$ **pair generated by DIDO.**

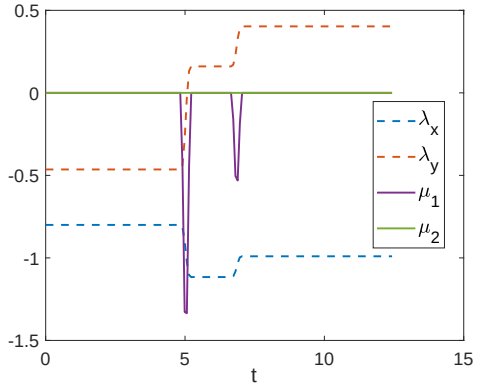

Figure 9. DIDO "predicts" $\mu_1(t)$ **to be the sum of two impulse functions centered at the jump points of the co-position trajectories** $t \mapsto (\lambda_x, \lambda_y)$**.**

quired by (23a). This function is zero everywhere except for two "sharp peaks" that are located exactly where the co-position trajectories jump. These sharp peaks are approximations to atomic measures[9]. As noted in [9], we can write $\boldsymbol{\mu}(\cdot)$ as the sum of two functions: one in L^∞ and another as a finite sum of Dirac delta functions. From this concept, it follows that

$$\mu_1(t) = \sum_j \eta_j \delta_D(t - t_j) \tag{36}$$

where δ_D is the Dirac delta function centered at t_j and η_j is finite. From (24), (36) and the signs of $(x(t_j) - x_1))$ and $(y(t_j) - y_1))$ (see Fig. 4) it follows that $\lambda_x(t)$ must jump downwards first and upwards afterwards while $\lambda_y(t)$ must jump upwards at both touch points. This is exactly the solution generated by DIDO in Fig. 9.

Yet another verification of the optimality of the computed trajectory is shown in Fig. 10. According to (28), the lower Hamiltonian must be a constant and equal to -1. DIDO's computation of the Hamiltonian evolution equation is not perfect as apparent from Fig. 10. This is, at least, in part because its computations of the Dirac delta functions is not perfect.

VI. Miscellaneous

DIDO's algorithm as implemented in DIDO is not perfect! In recognition of this distinction, it should not be surprise to any user that DIDO has bugs. It should also not be a surprise to any reader that DIDO and/or PS methods get frequently compared to other software, methods etc. In this section we address such miscellaneous items.

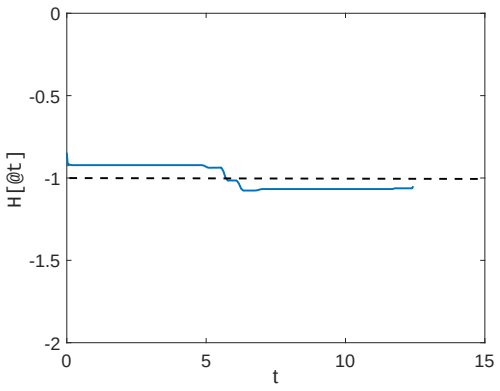

Figure 10. DIDO-computed evolution of the lower Hamiltonian $t \mapsto \mathcal{H}[@t]$ is on average a constant and equal to -1.

A. Known Bugs

In general, DIDO will run smoothly if the coded problem is free of singularities, well posed, scaled and balanced. Both beginners and advanced users can avoid costly mistakes in following the best practices as outlined in Sec. IV, together with using the DIDO Doctor Toolkit,™ and a preamble in coding the DIDO files. Even after exercising such care, a user may inadvertently make a mistake that is not detected by the DIDO Doctor Toolkit. In this case, DIDO may exit or quit with cryptic error messages. Examples of these are:

1. *Grid Interpolant Error*

This problem occurs when a user allows the possibility of $t_f - t_0 = 0$ in the form of the bounds on the initial and final time. The simple remedy is to ensure that the bounds are set up so that $t_f - t_0 > 0$.

2. *Growthcheck Error*

This is frequently triggered because of an incorrect problem formulation that the DIDO Doctor Toolkit is unable to detect. Best remedy is to check the coded problem formulation (and ensure that it is the same as one conceived). Example: if "x" is coded as "y."

3. *SOL Error*

This is not a bug; rather, it is simply as an error message where DIDO has gone wrong so badly that it is unable to provide any useful feedback to a user. The best remedy for a user is to start over or to go back to a prior version of his/her files to start over.

4. *Hamiltonian Error*

This is neither a bug nor an error but is listed here for information. Consider the case where the lower Hamiltonian is expected to be a constant. The computed Hamiltonian may undergo small jumps and oscillations around the expected constant. These jumps and oscillations may be amplified when one or more of the path covector functions contain Dirac-delta functions as evident in Fig. 9. When these are absent, the lower Hamiltonian computed by DIDO is reasonably accurate; see [9] for additional details on impulsive path covector functions and for the meaning of the various Hamiltonians.

B. Comparison With Other Methods and Software

It is forgone conclusion that every author of a code/method/software will claim superiority. In maintaining greater objectivity, we simply identify scientific facts regarding DIDO, PS theory and its implementations.

1. *PS vs RK; PS vs PS*

DIDO is based on PS discretizations. The convergence of a PS discretization is spectral; i.e., almost exponentially fast[26, 27, 28]. Convergence of Runge-Kutta (RK) discretizations are typically $\mathcal{O}(4)$. ***Consequently, it is mathematically impossible for a fourth-order method to converge faster than a higher-order method, particularly an "exponentially fast" method***. Claims to the contrary are based on a combination of one or more of the following:

1. Not all PS discretizations are convergent just as not all RK methods are convergent[30]. Unfortunately, even a convergent PS method may be inappropriate for a given problem[32] because of mathematical technicalities associated with inner products and boundary conditions[33]. By using an inappropriate PS discretization over a set of numerical examples, it possible to show that it performs poorly or similarly to a convergent RK method[34].

2. The "exponential convergence rate" of a PS discretization is based on the order of the interpolating polynomial: higher the order, the better the approximation[26]. Consequently, an RK method can be easily made to outperform a sufficiently low-order PS discretization that is based on highly-segmented mesh refinements; see [16] for details.

3. In a correct PS discretization, the control is ***not*** necessarily represented by a polynomial[32]; this fact has been known since at least 2006[35]. It is possible to generate a bad PS discretization by setting the control trajectory to be a polynomial. In this case, even an Euler method can be made to outperform a PS discretization.

4. Low order PS discretizations are claimed to outperform high-order PS methods based on the assumption that a high-order method exhibits a Gibbs phenomenon. It is apparent from Fig. 5 that there is no Gibbs phenomenon in a properly implemented PS discretization.

2. *Implementation vs Theory; Theory vs Implementation*

PS discretizations are deceptively simple. Its proper implementation requires a significant amount of computational finesse[17]. For instance, in Lagrange PS methods (see Fig. 11), it is very easy to generate differentiation matrices with large round-off errors[36], or compute Gaussian grid points poorly, or implement closure conditions[33, 37] incorrectly. In doing so, it is easy to show poor numerical performance of theoretically convergent PS discretizations. Furthermore it is easy to show large run times of fast PS discretizations by simply patching it with generic nonlinear programming solvers. It is also easy to claim poor computational performance of PS theory by using DIDO clones or other poor implementations of PS techniques[34]. It is also easy to show poor performance of DIDO by improper scaling or balancing or other inappropriate use of the toolbox.

To maintain scientific integrity, it is important to compare and contrast performance of trajectory optimization methods in a manner that is agnostic to its implementation and/or computational hardware. For instance, in static optimization, it is a scientific fact that a Newton method converges faster than a gradient method irrespective of its implementation on a computer[25]. This is because computational speed is defined precisely in terms of the order of convergence and not in terms of computational time. Furthermore, each Newton iteration is expensive relative to a gradient method because it involves the computation of a Hessian and solving a system of linear equations[25]. This fact remains unchanged no matter the implementation or the computer hardware. It is possible to perform computer-agnostic analysis of trajectory optimization methods; see [18]. ***This topic is mostly uncharted and remains a wide open area of research***. Nonetheless, there is no shortage of papers that use "alternative facts" and/or compare selective performance of codes as part of a self-fulfilling prophecy.

3. *DIDO $\gtreqless$ PS*

It is quite possible for an algorithm to perform better than theory (e.g. simplex vs ellipsoid algorithm) or for a code to perform worse than theory. The latter case is also true if a theory is implemented incorrectly, poorly or imperfectly. Consequently, using a code to draw conclusions about a theory is not only unscientific, it may also be quite misleading. DIDO is not immune to this fact. Furthermore, because PS discretization is only an element of DIDO's algorithm, it is misleading to draw conclusions about PS theory using DIDO. This fact is strengthened by the point noted earlier that even DIDO's algorithm as implemented in DIDO is not perfect! In fact, even the implementation is not complete! Despite all these caveats, it is still very easy to refute some of the claims made in the literature by simply running DIDO for the purportedly hard problems. For example, the claims made in [34] were easily refuted in [38] by simply using DIDO.

4. *DIDO vs DIDO's algorithm*

In principle, DIDO's algorithm can generate "near exact" solutions;[n] however, its precision is machine limited by the fundamental "square-root equation" given by (32). In practice, the precision is further limited than the square-root formula because of other machine and approximation errors (see (35)). In addition, DIDO's algorithm, as implemented in DIDO, the toolbox, is practically "tuned" to generate the best performance for "industrial strength" problems. This is because the toolbox, is designed to work across the "space of all optimal control problems." Consequently, it is not too hard to design an academic problem that shows a "poor performance" of DIDO, the toolbox, even though DIDO's algorithm may be capable of significantly superior performance.

C. Computer Memory Requirements

From (48) in Appendix A, it follows that the memory requirements for the variables is $\mathcal{O}(N)$, where $(N+1)$ is the number of points that constitute the grid (in time). This estimate can be refined further to $\mathcal{O}(nN)$ where n is the number of continuous-time variables. For instance, for Problem (G), $n \sim (4N_x + N_u + N_h)$. Assuming 8 bytes per variable, the memory requirements for variables can be written as $\mathcal{O}(nN \times 10)$ bytes with 8 replaced by 10. Thus, for example, for $N = 1,000$, the memory estimate can be written as $10n$ KB or 10 KB per continuous variable. Clearly, the number of variables is not a serious limiting factor for DIDO's algorithm. This is part of the reason why it was argued in [39] that the production of solutions over a million grid points was more connected to numerical science than computer technology.

Carrying out an analysis similar to that of the previous paragraph, it is straightforward to show that the number of constraints is also not a significant limiting factor to execute DIDO's algorithm. By the same token, the gradients of the nonlinear functions (see (48)) are also not memory-intensive because Hamiltonian programming naturally exploits the separability property inherent in $(P^{\lambda,D,N})$. This conclusion carries over to Problem (G) as well (see also [17]). Thus, the dominant sources of memory consumption are the matrices $\boldsymbol{A}$ and $\boldsymbol{A}^\star$ introduced in (46) and (47) respectively (see Appendix A). In choosing discretizations that satisfy Theorem 1, the memory requirements for the linear system given in (48) is $2N^2$ variables, and hence $2N_xN^2$ for solving Problem (G). In using this number and the analysis of the previous paragraph, it follows that the memory cost for a grid size of 250 points is about 1 MB per state variable (and independent of the number of other variables). For $1,000$ grid points, the memory requirement is about 10 MB per state variable.

Because its memory requirements are relatively low and convergence is fast, it follows that DIDO's algorithms can be easily embedded on a low end microprocessor. Specific details on a generic computer architecture for embedded optimal control are described in [32]. As noted earlier, it is important to note the critical difference between DIDO's algorithms and DIDO, the optimal control toolbox. The latter is not designed for embedding while the former is embed-ready. A proof-of-concept in embedding was shown in [32]. Consequently, it is technically feasible to generate real-time optimal controls by coordinating the Nyquist frequency requirements of digitized signals to the Lipschitz frequency production of optimal controls[9]. The mathematical foundations of this theory along with some experimental results are described in [40] and [41].

[n] The statement follows from the fact that DIDO's algorithm is grounded on the convergence property of the Stone-Weierstrass theorem and approximations in Sobolev spaces; see [32].

VII. Conclusions

DIDO's solution to the "space station problem" that created headlines in 2007 is now widely used by other codes to benchmark their performance. The early versions of DIDO are extensively cited as a model for new software. Later versions of the optimal control toolbox execute a substantially more sophisticated algorithm that is focused on being guess-free, true-fast and verifiably accurate. Despite major internal changes, all versions of DIDO have remained true to its founding principle: to facilitate a fast and quick process in taking a concept to code using nothing but Pontryagin's Principle as a central analysis tool. In this context, DIDO as a tool is more about a minimalist approach to solving optimal control problems rather than pseudospectral theory or its implementation. This is the main reason why the best way to use DIDO is via Pontryagin's Principle![o]

Because DIDO does not ask a user to supply the necessary conditions of optimality, it is frequently confused for a "direct" method. Ironically, it is also confused with an "indirect" method because DIDO also provides costates and other multipliers associated with Pontryagin's Principle. From the details provided in this paper, it is clear that DIDO does not fit within the traditional stovepipes of direct and indirect methods. DIDO achieves its true-fast computational speed without requiring a user to provide a "guess," or analytical gradient/Jacobian/Hessian information or demanding a high-performance computer; rather, the "formula" behind DIDO's robustness, speed and accuracy is based on new fundamentals in dynamic optimization that are are agnostic to pseudospectral discretizations. This is likely the reason why DIDO is able to solve a wide variety of "hard" differential-algebraic problems that other codes/methods have admitted difficulties.

References

[1] N. Bedrossian, S. Bhatt, M. Lammers and L. Nguyen, "Zero Propellant Maneuver: Flight Results for 180° ISS Rotation," *NASA CP Report 2007-214158*; *20th International Symposium on Space Flight Dynamics*, September 24-28, 2007, Annapolis, MD.

[2] N. Bedrossian, S. Bhatt, M. Lammers, L. Nguyen and Y. Zhang, "First Ever Flight Demonstration of Zero Propellant Maneuver Attitude Control Concept," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2007, AIAA 2007-6734.

[3] W. Kang and N. Bedrossian, "Pseudospectral Optimal Control Theory Makes Debut Flight: Saves NASA $1M in in under 3 hrs," *SIAM News*, page 1, Vol. 40, No. 7, 2007.

[4] N. Bedrossian, S. Bhatt, W. Kang and I. M. Ross, "Zero Propellant Maneuver Guidance," *IEEE Control Systems Magazine*, Vol. 29, Issue 5, October 2009, pp. 53-73.

[5] I. M. Ross and F. Fahroo, "Legendre Pseudospectral Approximations of Optimal Control Problems," *Lecture Notes in Control and Information Sciences*, Vol. 295, Springer–Verlag, New York, 2003, pp. 327–342.

[6] I. M. Ross and F. Fahroo, "Pseudospectral Knotting Methods for Solving Optimal Control Problems," *Journal of Guidance, Control and Dynamics*, Vol. 27, No. 3, pp. 397-405, 2004.

[o] This philosophy of analysis before coding, as exemplified in Sec. V, is in sharp contrast to the traditional "brute-force" direct method. The widely-advertised advantage of the latter approach is that a user apparently need not know/understand any theory to create/run codes for solving problems.

[7] I. M. Ross and Q. Gong, "Guess-Free Trajectory Optimization," *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, 18-21 August 2008, Honolulu, Hawaii. AIAA 2008-6273.

[8] Q. Gong, F. Fahroo and I. M. Ross, "Spectral Algorithm for Pseudospectral Methods in Optimal Control," *Journal of Guidance, Control, and Dynamics*, vol. 31 no. 3, pp. 460-471, 2008.

[9] I. M. Ross, *A Primer on Pontryagin's Principle in Optimal Control*, Second Edition, Collegiate Publishers, San Francisco, CA, 2015.

[10] L. S. Pontryagin, V. G. Boltayanskii, R. V. Gamkrelidze and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*, Wiley, 1962 (Translated from Russian).

[11] I. M. Ross, M. Karpenko and R. J. Proulx, "A Nonsmooth Calculus for Solving Some Graph-Theoretic Control Problems," IFAC-PapersOnLine 49-18, 2016, pp. 462–467.

[12] F. Clarke, *Functional Analysis, Calculus of Variations and Optimal Control*, Springer-Verlag, London, 2013; Ch. 22.

[13] G. G. Lorentz and K. L. Zeller, "Birkhoff Interpolation," *SIAM Journal of Numerical Analysis*, Vol. 8, No. 1, pp. 43-48, 1971.

[14] W. F. Finden, "An Error Term and Uniqueness for Hermite-Birkhoff Interpolation Involving Only Function Values and/or First Derivative Values," *Journal of Computational and Applied Mathematics*, Vol. 212, No. 1, pp. 1-15, 2008.

[15] L.-L Wang, M. D. Samson and X. Zhao, "A Well-Conditioned Collocation Method Using a Pseudospectral Integration Matrix," *SIAM Journal of Scientific Computaton*, Vol. 36, No. 3, pp. A907-A929, 2014.

[16] N. Koeppen, I. M. Ross, L. C. Wilcox and R. J. Proulx, "Fast Mesh Refinement in Pseudospectral Optimal Control," *Journal of Guidance, Control and Dynamcis*, (42)4, 2019, 711–722.

[17] I. M. Ross and R. J. Proulx, "Further Results on Fast Birkhoff Pseudospectral Optimal Control Programming," *J. Guid. Control Dyn.* 42/9 (2019), 2086–2092.

[18] I. M. Ross, "A Direct Shooting Method is Equivalent to An Indirect Method," arXiv preprint (2020) arXiv:2003.02418v1.

[19] I. M. Ross, Q. Gong, M. Karpenko and R. J. Proulx, "Scaling and Balancing for High-Performance Computation of Optimal Controls," *Journal of Guidance, Control and Dynamics*, Vol. 41, No. 10, 2018, pp. 2086–2097.

[20] L. R. Lewis, I. M. Ross and Q. Gong, "Pseudospectral motion planning techniques for autonomous obstacle avoidance," *Proceedings of the 46th IEEE Conference on Decision and Control*, New Orleans, LA, pp. 5997-6002 (2007).

[21] K. Bollino, L. R. Lewis, P. Sekhavat and I. M. Ross, "Pseudospectral Optimal Control: A Clear Road for Autonomous Intelligent Path Planning," *Proc. of the AIAA Infotech@Aerospace 2007 Conference*, CA, May 2007.

[22] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*, Cambridge University Press, Cambridge, MA, 2019.

[23] D. Robinson, Private communication.

[24] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*, SIAM, Philadelphia, PA, 2001.

[25] P. E. Gill, W. Murray and M. H. Wright, *Practical Optimization*, Academic Press, London, 1981.

[26] L. N. Trefethen, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, PA, 2013.

[27] W. Kang, I. M. Ross and Q. Gong, "Pseudospectral Optimal Control and its Convergence Theorems," *Analysis and Design of Nonlinear Control Systems*, Springer-Verlag, Berlin Heidelberg, 2008, pp. 109–126.

[28] W. Kang, "Rate of Convergence for a Legendre Pseudospectral Optimal Control of Feedback Linearizable Systems," *Journal of Control Theory and Applications*, Vol. 8, No. 4, pp. 391–405, 2010.

[29] Q. Gong and I. M. Ross, "Autonomous Pseudospectral Knotting Methods for Space Mission Optimization," *Advances in the Astronatuical Sciences*, Vol. 124, 2006, AAS 06-151, pp. 779–794.

[30] E. Hairer, S. P. Nørsett and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, Springer-Verlag Berlin Heidelberg, 1993.

[31] M. Karpenko, I. M. Ross, E. Stoneking, K. Lebsock and C. J. Dennehy, "A Micro-Slew Concept for Precision Pointing of the Kepler Spacecraft," *AAS/AIAA Astrodynamics Specialist Conference*, August 9-13, 2015, Vail, CO. Paper number: AAS-15-628.

[32] I. M. Ross and M. Karpenko, "A Review of Pseudospectral Optimal Control: From Theory to Flight," *Annual Reviews in Control*, Vol.36, No.2, pp.182–197, 2012.

[33] F. Fahroo and I. M. Ross, "Advances in Pseudospectral Methods for Optimal Control," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2008–7309, Honolulu, Hawaii, August 2008.

[34] S. L. Campbell and J. T. Betts, "Comments on Direct Transcription Solution of DAE Constrained Optimal Control Problems with Two Discretization Approaches," *Numer Algor* (2016) 73:807-838.

[35] Q. Gong, W. Kang and I. M. Ross, "A Pseudospectral Method for the Optimal Control of Constrained Feedback Linearizable Systems," *IEEE Transactions on Automatic Control*, Vol. 51, No. 7, July 2006, pp. 1115-1129.

[36] R. Baltensperger and M. R. Trummer, "Spectral Differencing with a Twist," *SIAM J. Sci. Comput.*, 24(5), 1465–1487, 2003.

[37] I. M. Ross and F. Fahroo, "Discrete Verification of Necessary Conditions for Switched Nonlinear Optimal Control Systems," *Proceedings of the American Control Conference*, June 2004, Boston, MA.

[38] H. Marsh, M. Karpenko and Q. Gong, "A Pseudospectral Approach to High Index DAE Optimal Control Problems," arXiv preprint 2018, arXiv:1811.12582.

[39] I. M. Ross, M. Karpenko and R. J. Proulx, "The Million Point Computational Optimal Control Challenge," *SIAM Conference on Control and Its Applications*, MS24, Pittsburgh, PA, July 2017.

[40] I. M. Ross, P. Sekhavat, A. Fleming and Q. Gong, "Optimal Feedback Control: Foundations, Examples, and Experimental Results for a New Approach," *Journal of Guidance, Control and Dynamics*, Vol. 31, No. 2, March-April 2008.

[41] I. M. Ross, Q. Gong, F. Fahroo and W. Kang, "Practical Stabilization Through Real-Time Optimal Control," *2006 American Control Conference*, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, 14–16 June 2006.

[42] W. Chen, Z. Shao and L. T. Biegler, "A Bilevel NLP Sensitivity-based Decomposition for Dynamic Optimization with Moving Finite Elements," AIChE J., 2014, 60, 966–979.

[43] W. Chen and L. T. Biegler, "Nested Direct Transcription Optimization for Singular Optimal Control Problems," AIChE J., 2016, 62, 3611–3627.

[44] L. Ma, Z. Shao, W.Chen and Z. Song, "Trajectory optimization for lunar soft landing with a Hamiltonian-based adaptive mesh refinement strategy," *Advances in Engineering Software* Volume 100, October 2016, pp. 266–276.

[45] Q. Gong, I. M. Ross and F. Fahroo, "Pseudospectral Optimal Control On Arbitrary Grids," *AAS Astrodynamics Specialist Conference*, AAS-09-405, 2009.

[46] Q. Gong, I. M. Ross and F. Fahroo, "Spectral and Pseudospectral Optimal Control Over Arbitrary Grids," *Journal of Optimization Theory and Applications*, vol. 169, no. 3, pp. 759-783, 2016.

[47] Q. Gong, I. M. Ross and F. Fahroo, "A Chebyshev pseudospectral method for nonlinear constrained optimal control problems," *Proceedings of the 48h IEEE Conference on Decision and Control (CDC)*, Shanghai, 2009, pp. 5057–5062.

[48] I. M. Ross, An optimal control theory for nonlinear optimization, *J. Comp. and Appl. Math.*, 354 (2019) 39–51.

[49] I. M. Ross, "An Optimal Control Theory for Accelerated Optimization," arXiv preprint (2019) arXiv:1902.09004v2.

[50] K. Yosida, *Functional Analysis*, Sixt Edition, Springer-Verlag, New York, 1980.

[51] E. J. McShane, "Partial Orderings and Moore-Smith Limits," *The American Mathematical Monthly*, Vol. 59, No. 1, 1952, pp. 1–11.

Appendix A: DIDO's Computational Theory

A vexing question that has plagued computational methods for optimal controls is the treatment of differential constraints. DIDO addresses this question by introducing a ***virtual control variable*** $\boldsymbol{v}$ to rewrite the differential constraint as,

$$\frac{d\boldsymbol{x}}{d\tau} := \boldsymbol{v} \tag{37a}$$

$$\left(\frac{d\Gamma}{d\tau}\right) \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}, \Gamma(\tau; t_0, t_f)) = \boldsymbol{v} \tag{37b}$$

where, $\Gamma : \mathbb{R} \to \mathbb{R}$ is an invertible nonlinear transformation[9],

$$t = \Gamma(\tau; t_0, t_f) \quad \Leftrightarrow \tau = \Gamma^{-1}(t; t_0, t_f) \tag{38}$$

and τ is transformed time over a fixed horizon $[\tau^0, \tau^f]$. Because the virtual control variable affects only the dynamical constraints, the rest of the problem formulation remains unchanged. Thus the age-old question of the "best" way to discretize a nonlinear differential equation is relegated to answering the same question for the embarrassingly simple linear equation given by (37a).

Remark 3 *The transformation given by* (37) *is only internal to DIDO. As far as the user is concerned a virtual control variable does not exist.*

As simple as it is, a surprisingly significant amount of sophistication is necessary to implement (37a) efficiently. To explain the details of this computational theory, we employ the special problem defined by,

$$(P) \begin{cases} \textit{Minimize} & J[x(\cdot), u(\cdot)] := \displaystyle\int_{t_0}^{t_f} F(x(t), u(t))\, dt \\ \textit{Subject to} & \dot{x}(t) = f(x(t), u(t)) \end{cases} \tag{39}$$

where, $t_0, t_f, x(t_0)$ and $x(t_f)$ are all fixed at some appropriate values. Furthermore, because the time interval $[t_f - t_0]$ is fixed, there is no need to perform a "τ"-transformation to generate the virtual control variable. Consequently, using (37), we construct the ***DIDO-form*** of (P) as,

$$(P^D) \begin{cases} \textit{Minimize} & J[x(\cdot), u(\cdot), v(\cdot)] := \\ & \qquad \displaystyle\int_{t_0}^{t_f} F(x(t), u(t))\, dt \\ \textit{Subject to} & \dot{x}(t) = v(t) \\ & 0 = f(x(t), u(t)) - v(t) \end{cases} \tag{40}$$

A. Introduction to the Cotangent Barrier

Using the equations and the process described in Sec. II in applying Pontraygin's Principle to Problem (P), it is straightforward to show that the totality of conditions results in the following nonlinear differential-algebraic equations:

$$(P^\lambda) \begin{cases} \dot{x}(t) = & f(x(t), u(t)) \\ -\dot{\lambda}(t) = & \partial_x H(\lambda(t), x(t), u(t)) \\ 0 = & \partial_u H(\lambda(t), x(t), u(t)) \end{cases} \tag{41}$$

Using the concept of the virtual control variable for both the state and the costate equations, the *DIDO-form* of (P^λ) is given by,

$$(P^{\lambda,D}) \begin{cases} \dot{x}(t) = & v(t) \\ -\dot{\lambda}(t) = & \omega(t) \\ 0 = & f(x(t), u(t)) - v(t) \\ 0 = & \partial_x H(\lambda(t), x(t), u(t)) - \omega(t) \\ 0 = & \partial_u H(\lambda(t), x(t), u(t)) \end{cases} \tag{42}$$

DIDO is based on the principle that (42) is computationally easier than (41) to the extent that all the differential components are concentrated in the exceedingly simple linear equations. To appreciate the simple sophistication of this principle, consider now an application of Pontryagin's Principle to Problem (P^D); this generates,

$$(P^{D,\lambda}) \begin{cases} \dot{x}(t) = & v(t) \\ 0 = & f(x(t), u(t)) - v(t) \\ -\dot{\lambda}(t) = & \partial_x H(\mu(t), x(t), u(t)) \\ \lambda(t) = & \mu(t) \\ 0 = & \partial_u H(\mu(t), x(t), u(t)) \end{cases} \tag{43}$$

Comparing (43) to (42), it is clear that,

$$(P^{D,\lambda}) \not\equiv (P^{\lambda,D}) \tag{44}$$

Thus, an application of Pontryagin's Principle to the DIDO-form of the primal problem does not generate a DIDO-form of the primal-dual problem. This is the "insurmountable" ***cotangent barrier***; i.e., (44) seems to suggest that there was no value in using the DIDO form of the primal problem. Despite this apparently disheartening result, we now show that it is possible to tunnel through the barrier, albeit discretely (pun intended!).

B. Discrete Tunneling Through the Cotangent Barrier

Let X^N, U^N, and V^N be $(N+1)$-dimensional vectors that represent discretized state, control and virtual control variables respectively over an arbitrary grid π^N:

$$X^N := (x_0, x_1, \ldots, x_N) \tag{45a}$$

$$U^N := (u_0, u_1, \ldots, u_N) \tag{45b}$$

$$V^N := (v_0, v_1, \ldots, v_N) \tag{45c}$$

From linearity arguments, it follows that a broad class of generic discretizations of $\dot{x} = v$ may be written as,

$$\boldsymbol{A} \begin{pmatrix} X^N \\ V^N \end{pmatrix} \equiv \boldsymbol{A}_x X^N + \boldsymbol{A}_v V^N = C^N \tag{46}$$

where, $\boldsymbol{A} := [\boldsymbol{A}_x \mid \boldsymbol{A}_v]$ comprises two $(N+1) \times (N+1)$ discretization matrices that depend on N, π^N, and a choice of the discretization method. The choice of the discretization method also determines C^N, an $(N+1) \times 1$ matrix. The dependence of $\boldsymbol{A}_x$ and $\boldsymbol{A}_v$ on N and π^N is suppressed for notational convenience. Following the same rationale as in the production of (46), a generic discretization of $\dot{\lambda} = -\omega$ may be written as,

$$\boldsymbol{A}^* \begin{pmatrix} \Lambda^N \\ \Omega^N \end{pmatrix} \equiv \boldsymbol{A}^*_\lambda \Lambda^N + \boldsymbol{A}^*_\omega \Omega^N = C^{*N} \tag{47}$$

where $\Lambda^N \in \mathbb{R}^{N+1}$ and $\Omega^N \in \mathbb{R}^{N+1}$ are defined analogous to (45), $\boldsymbol{A}^*$ is a matrix (that may not necessarily be equal to $\boldsymbol{A}$) that depends on N and π^N, and C^{*N} is a matrix similar to C^N. Collecting all relevant

equations, it follows that a generic discretization of $(P^{\lambda,D})$ may be written as,

$$(P^{\lambda,D,N})\begin{cases} \boldsymbol{A}\begin{pmatrix} X^N \\ V^N \end{pmatrix} = & C^N \\ \boldsymbol{A}^*\begin{pmatrix} \Lambda^N \\ \Omega^N \end{pmatrix} = & C^{*N} \\ f(X^N,U^N)-V^N = & 0 \\ \partial_x H(\Lambda^N,X^N,U^N)-\Omega^N = & 0 \\ \partial_u H(\Lambda^N,X^N,U^N) = & 0 \end{cases} \tag{48}$$

where, f and H are reused as overloaded operators to take in discretized vectors as inputs. Similarly, (P^D) may be discretized to generate the following problem,

$$(P^{D,N})\begin{cases} \textit{Minimize} & J^{D,N}[X^N,U^N,V^N] \\ & \quad := \boldsymbol{q}^T F(X^N,U^N) \\ \textit{Subject to} & \boldsymbol{A}\begin{pmatrix} X^N \\ V^N \end{pmatrix} = C^N \\ & f(X^N,U^N)-V^N = 0 \end{cases} \tag{49}$$

where, $\boldsymbol{q} = (q_0, q_1, \ldots, q_N)$ is an $(N+1)\times 1$ vector of positive quadrature weights associated with the specifics of the discretization given by $\boldsymbol{A}$ (and inclusive of π^N).

Remark 4 *From (48), it follows that the entire approach to discretization is isolated in the matrix pair $\{\boldsymbol{A}, \boldsymbol{A}^*\}$. Part of DIDO's computational speed is achieved by exploiting this linear system.*

Lemma 1 *Let $\boldsymbol{Q}$ be the positive definite diagonal matrix defined by diag $(q_0, q_1, \ldots, q_N)$. Define,*

$$\boldsymbol{A}_x^\dagger := \boldsymbol{Q}^{-1}\boldsymbol{A}_x^T\boldsymbol{Q} \quad \textit{and} \quad \boldsymbol{A}_v^\dagger := \boldsymbol{Q}^{-1}\boldsymbol{A}_v^T\boldsymbol{Q} \tag{50}$$

Then, under appropriate technical conditions at the boundary points, there exist multipliers $\Psi_A^N \in \mathbb{R}^{N+1}$ and $\Psi_d^N \in \mathbb{R}^{N+1}$ for Problem $(P^{D,N})$ such that its dual feasibility conditions can be written as,

$$\boldsymbol{A}_v^\dagger\overline{\Psi}_A^N - \overline{\Psi}_d^N = C^{*N} \tag{51}$$

$$\partial_x H\left(\overline{\Psi}_d^N, X^N, U^N\right) + \boldsymbol{A}_x^\dagger\overline{\Psi}_A^N = 0 \tag{52}$$

$$\partial_u H\left(\overline{\Psi}_d^N, X^N, U^N\right) = 0 \tag{53}$$

where $\overline{\Psi}_A^N$ and $\overline{\Psi}_d^N$ are given by,

$$\overline{\Psi}_A^N := \boldsymbol{Q}^{-1}\Psi_A^N \quad \overline{\Psi}_d^N := \boldsymbol{Q}^{-1}\Psi_d^N \tag{54}$$

Proof. This lemma can be proved by an application of the multiplier theory. Details are omitted for brevity. □

Theorem 1 (A Tunnel Theorem) *Suppose $\boldsymbol{A}$ and $\boldsymbol{A}^*$ are discretization matrix pairs that satisfy the conditions,*

$$\boldsymbol{A}_x = -\boldsymbol{I}^N \tag{55a}$$

$$\boldsymbol{A}_\lambda^* = -\boldsymbol{I}^N \tag{55b}$$

$$\boldsymbol{A}_\omega^* = \boldsymbol{A}_v^\dagger \tag{55c}$$

then, the necessary conditions for Problem $(P^{D,N})$ are equivalent to $(P^{\lambda,D,N})$ under the transformation,

$$\overline{\Psi}_d^N = \Lambda^N \tag{56a}$$

$$\overline{\Psi}_A^N = \Omega^N \tag{56b}$$

Proof. The proof of this theorem follows from Lemma 1 and by a substitution of (55) and (56) in the the definitions of $(P^{D,N})$ and $(P^{\lambda,D,N})$. □

Definition 1 *$(P^{D,N})$ is called a **DIDO-Hamiltonian programming problem** if $\boldsymbol{A}$ and $\boldsymbol{A}^*$ are chosen in accordance with Theorem 1.*

See [18] for a first-principles introduction to the notion of ***Hamiltonian programming***. A DIDO-Hamiltonian programming problem is simply an adaptation of this terminology to Problem $(P^{D,N})$.

Theorem 2 *All Lagrange PS discretizations over any grid (including Gauss, Radau, and Lobatto) fail to satisfy the conditions of Theorem 1.*

Proof. The proof of this theorem is fairly straightforward. It follows from the fact that all Lagrange PS discretizations[32, 16] over any grid[45, 33] are based on a differentiation matrix, $\boldsymbol{D}$. Hence all Lagrange PS discretizations of $\dot{x} = v$ are given by,

$$\boldsymbol{D}X^N = V^N \quad \Rightarrow \quad V^N - \boldsymbol{D}X^N = \boldsymbol{0} \tag{57}$$

This implies $\boldsymbol{A}_x = -\boldsymbol{D}$ and $\boldsymbol{A}_v = \boldsymbol{I}^N$ which violates the conditions of Theorem 1. □

Theorem 3 *Birkhoff PS discretizations of optimal control problems satisfy the conditions of Theorem 1.*

Proof. The proof of this theorem follows from the fact that a Birkhoff PS discretization[16, 17] may be written in a form that generates $\boldsymbol{A}_x = -\boldsymbol{I}^N$; see [17] for further details. □

Theorems 2 and 3 imply that there are two broad categories of PS discretizations for optimal control problems, namely Lagrange and Birkhoff. Within these two main methods of discretizations, it is possible to generate a very large number of variations based on the choice of basis functions and grid selections. ***A process to achieve at least eighteen variations of PS discretizations based on classical orthogonal polynomials is shown in Fig. 11***. Despite this apparently large variety of choices, Theorems 2 and 3 reveal that there are no essential differences between PS discretizations based on different grid points. All that matters is whether they are based on Lagrange or Birkhoff interpolants. Nonetheless, when additional factors are taken into account, the choice of a grid can have a deleterious effect on convergence; see [32, 33] for details.

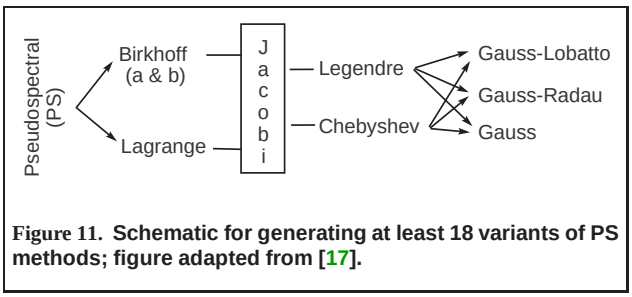

Figure 11. Schematic for generating at least 18 variants of PS methods; figure adapted from [17].

Appendix B: DIDO's Suite of Algorithms

As noted in Sec. I, DIDO's algorithm is actually a suite of algorithms. Several portions of this suite can be found in various publications scattered across different journals and conference proceedings. In this section, we collect and categorize this algorithm-suite to explain how specific algorithmic options are triggered by the particulars of the user-defined problem and inputs to DIDO.

C. Introduction and Definitions

Each main algorithm in DIDO's suite is ***true-fast, spectral and Hamiltonian***. That is, each of DIDO's main algorithm solves the DIDO-Hamiltonian programming problem (see Definition 1) using a modified[16, 17] spectral algorithm[8, 46] that is true-fast. True-fast is different from computationally fast in the sense that the former must be agnostic to the details of its computer implementations whereas the latter can be can be achieved via a variety of simple and obvious ways. For instance, any optimization software, including a mediocre one, can be made computationally fast by using one or more of the following:

1. A good/excellent "guess;"
2. A high-performance computer;
3. A compiled or embedded code; and,
4. Analytic gradient/Jacobian/Hessian information.

A truly fast algorithm should be *independent* of these obvious and other run-time code improvements. ***For an algorithm to be true-fast, it must satisfy the following properties*** (see Fig. 12):

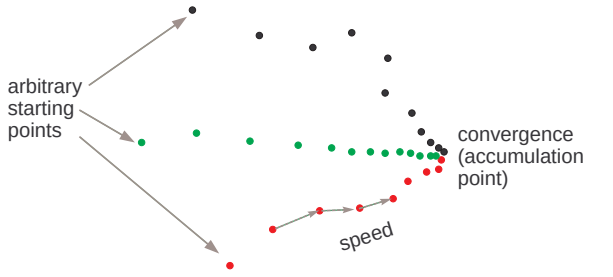

Figure 12. Schematic for defining an algorithm and its fundamental properties.

1. Converge from an arbitrary starting point;
2. Converge to an optimal/extremal solution;[p]
3. Take the fewest number of iterations towards a given tolerance criterion; and/or
4. Use the fewest number of operations per iteration.

Thus, a true-fast algorithm can be easily made computationally fast, but its implementation may not necessarily be computationally fast. Likewise, a computationally fast algorithm may not necessarily be true-fast. DIDO achieves true-fast speed by calling several true-fast algorithms from its toolbox. The specifics of a given problem and user inputs initiate automatic triggers to generate a coordinated DIDO-iteration that defines the flow of the main algorithm. To describe this algorithm suite, we use with the following definitions from [18]:

Definition 2 (Inner Algorithm $\mathcal{A}$) *Let (P^N) denote any discretization of (P). An inner algorithm $\mathcal{A}$ for Problem (P) is defined as finding a sequence of vector pairs $(X_0^N, U_0^N), (X_1^N, U_1^N), \ldots$ by the iterative map,*

$$(X_{i+1}^N, U_{i+1}^N) = \mathcal{A}(X_i^N, U_i^N),\ i \in \mathbb{N}$$

Note that N is fixed in the definition of the inner algorithm.

Definition 3 (Convergence of an Inner Algorithm) *An inner algorithm $\mathcal{A}$ is said to converge if*

$$\lim_{n\to\infty} X_n^N := X_\infty^N, \qquad \lim_{n\to\infty} U_n^N := U_\infty^N$$

is an accumulation point that solves Problem (P^N).

Definition 4 (Hamiltonian Algorithm $\mathcal{A}^\lambda$) *Let $(P^{\lambda,N})$ denote any discretization of (P^λ). A (convergent) inner algorithm $\mathcal{A}^\lambda$ is said to be Hamiltonian if it generates an additional sequence of vectors $\Lambda_0^N, \Lambda_1^N, \ldots$ for a fixed N such that*

$$\lim_{n\to\infty} \Lambda_n^N := \Lambda_\infty^N$$

is an accumulation point that solves $(P^{\lambda,N})$.

Remark 5 *Note that the Hamiltonian algorithm is focused on solving $(P^{\lambda,N})$ not $(P^{N,\lambda})$. In general, $(P^{\lambda,N}) \neq (P^{N,\lambda})$ for finite N even if they are equal in the limit as $N \to \infty$.*

[p]An extremal solution is one that satisfies Pontryagin's Principle.

Lemma 2 *Suppose $(P^{\lambda,N}) = (P^{N,\lambda})$ for finite N. Let $\delta^{\lambda,N} > 0$ be a specified tolerance for satisfying the constraints that define $(P^{\lambda,N})$. If $(P^{N,\lambda})$ is solved to a practical tolerance of $\delta^{N,\lambda} > 0$, then,*

$$\delta^{\lambda,N} > \delta^{N,\lambda} \tag{58}$$

Proof. This lemma is proved in [18] for a shooting method and Euler discretization. Its extension to the general case as given by (58) follows from the assumption that $(P^{\lambda,N}) = (P^{N,\lambda})$ for finite N. □

From (58) it follows that $\delta^{\lambda,N} \neq \delta^{N,\lambda}$. A Hamiltonian algorithm fixes this discrepancy.

As explained in [18], Problem (P^N) is better categorized as a Hamiltonian programming problem rather than as a nonlinear programming problem. This is, in part, because ***a Hamiltonian is not a Lagrangian*** and Problem (P^N) contains "hidden" information that is not accessible via $(P^{N,\lambda})$. A Hamiltonian algorithm aims to solve $(P^{\lambda,N})$ while a nonlinear programming algorithm attempts to solve $(P^{N,\lambda})$. Even when these two problems are theoretically equivalent to one another, a nonlinear programming algorithm does not solve $(P^{\lambda,N})$ to the same accuracy as $(P^{N,\lambda})$ as shown in Lemma 2; hence, a Hamiltonian algorithm is needed to replace or augment generic nonlinear programming algorithms. These ideas are depicted in Fig. 13. Specific details

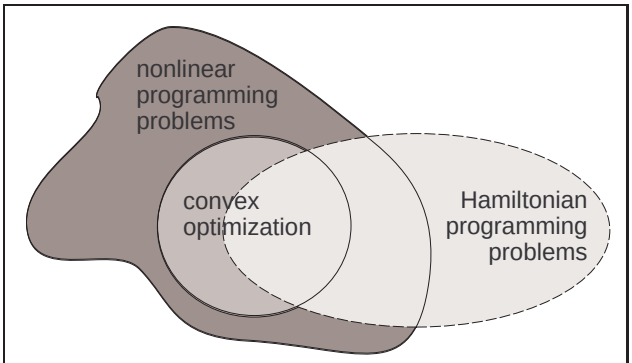

Figure 13. A depiction of the differences between nonlinear and Hamiltonian programming.

in augmenting a nonlinear programming algorithm with a Hamiltonian component may be found in [42, 43, 44]. Alternative implementations that achieve similar effects are discussed in [37, 45, 46, 47]. Note also that in Hamiltonian programming, the separable-programming-type property inherent in $(P^{D,N})$ is naturally exploited. In the multivariable case, this property lends itself to casting the discretized problem in terms of a compact matrix-vector programming problem rather than a generic a nonlinear programming problem[17].

Definition 5 (DIDO-Hamiltonian Algorithm $\mathcal{A}^{\lambda,D}$) *A Hamiltonian algorithm $\mathcal{A}^{\lambda,D}$ is said to be DIDO-Hamiltonian if it generates an additional sequence of vector pairs $(V_0^N \Omega_0^N), (V_1^N, \Omega_1^N), \ldots$ for a fixed N such that*

$$\lim_{n\to\infty} V_n^N := V_\infty^N, \quad \lim_{n\to\infty} \Omega_n^N := \Omega_\infty^N$$

is an accumulation point that solves Problem $(P^{\lambda,D,N})$.

Remark 6 *Because $(P^{\lambda,D}) \not\equiv (P^{D,\lambda})$ (see (44)), we have $(P^{\lambda,D,N}) \not\equiv (P^{D,\lambda,N})$ unless the discretization offers a cotangent tunnel; see Theorem 1. Even if $(P^{\lambda,D,N}) \equiv (P^{D,\lambda,N})$, it follows from Lemma 2 that $\delta^{\lambda,D,N} \neq \delta^{D,\lambda,N}$, where $\delta^{\lambda,D,N}$ and $\delta^{D,\lambda,N}$ denote the constraint tolerances on $(P^{\lambda,D,N})$ and $(P^{D,\lambda,N})$ respectively. The DIDO-Hamiltonian algorithm fixes this discrepancy.*

A solution to Problem (P^N) does not imply a solution to Problem (P). In fact, an optimal solution to Problem (P^N) may not even be a feasible solution to Problem (P). This point is often lost when nonlinear

programming solvers are patched to solve discretized optimal control problems. To clarify this point, we use the following definitions from [18]:

Definition 6 (Convergence of a Discretization) *Let (X_∞^N, U_∞^N) be a solution to Problem (P^N). A discretization is said to converge if*

$$\lim_{N\to\infty} X_\infty^N := X_\infty^\infty, \qquad \lim_{N\to\infty} U_\infty^N := U_\infty^\infty$$

is an accumulation point that solves Problem (P).

Remark 7 *In Definition 6 and subsequent ones to follow, we have taken some mathematical liberties with respect to the precise notion of a metric for convergence. It is straightforward to add such precision; however, it comes at a significant cost of new mathematical machinery. Consequently, we choose not to include such additional mathematical nomenclature in order to support the accessibility of the proposed concepts to a broader audience.*

Definition 7 (Dynamic Optimization Algorithm $\mathcal{B}$) *Let $\mathcal{B}$ be an algorithm that generates a sequence of integer pairs (N_0, m_0), $(N_1, m_1)\cdots$ such that the sequence*

$$(X_{m_0}^{N_0}, U_{m_0}^{N_0}), (X_{m_1}^{N_1}, U_{m_1}^{N_1}), \ldots$$

converges to an accumulation point $(X_\infty^\infty, U_\infty^\infty)$ given by Definition 6. Then, $\mathcal{B}$ is called a dynamic optimization algorithm for solving Problem (P).

From Definitions 6 and 7, it follows that a generic dynamic optimization algorithm involves the production of a double infinite sequence of vectors[18]. From Definition 6 it follows that the production of this double infinite sequence may be greatly facilitated by the use of an inner algorithm; see Definition 2. From Definition 7, it follows that if an inner algorithm is used to construct $\mathcal{B}$, then it need not even be executed to completion! This idea is part of DIDO's process for generating a true-fast optimization algorithm.

Definition 8 (Hamiltonian Dynamic Optimization Algorithm $\mathcal{B}^\lambda$) *A dynamic optimization algorithm $\mathcal{B}^\lambda$ is said to be Hamiltonian if it generates an additional convergent sequence of vectors*

$$\Lambda_{m_0}^{N_0}, \Lambda_{m_1}^{N_1}, \ldots$$

such that they converge to an accumulation point $(X_\infty^\infty, U_\infty^\infty, \Lambda_\infty^\infty)$ that solves (P^λ).

Definition 9 (DIDO-Hamiltonian Dynamic Optimization Algorithm) *A Hamiltonian dynamic optimization algorithm $\mathcal{B}^{\lambda,D}$ is said to be DIDO-Hamiltonian if it generates an additional sequence of vector pairs*

$$(V_{m_0}^{N_0}, \Omega_{m_0}^{N_0}), (V_{m_0}^{N_0}, \Omega_{m_1}^{N_1}), \ldots$$

that converges to an accumulation point $(X_\infty^\infty, U_\infty^\infty, V_\infty^\infty, \Lambda_\infty^\infty, \Omega_\infty^\infty)$ that solves $(P^{\lambda,D})$.

It is clear that Definition 9 relies on Definitions 1–8. From these definitions it follows that DIDO-Hamiltonian dynamic optimization algorithm generates a double-infinite sequence of tuples,

$$(X, U, V, \Lambda, \Omega)_{m_0}^{N_0}, (X, U, V, \Lambda, \Omega)_{m_1}^{N_1}, \ldots, (X, U, V, \Lambda, \Omega)_{m_f}^{N_f}, \ldots, (X, U, V, \Lambda, \Omega)_\infty^\infty \quad (59)$$

whose limit point purportedly converges to a solution of $P^{\lambda,D}$.

D. Overview of The Three Major Algorithmic Components of DIDO

The production of each element of the sequence denoted in (59) is determined by suite of algorithms that call upon different inner-loop DIDO-Hamiltonian algorithms (see Definition 5). The specific calls on the inner loop are based upon the performance of the outer loop (see Definition 9). DIDO uses different inner loops to support its three performance goals of robustness, speed and accuracy as stated in Sec. I. Consequently, the three major algorithmic components of DIDO's suite of algorithms comprise (see Fig. 14):

1. A stabilization component: The task of this component of the algorithm is to drive an "arbitrary" point to an "acceptable" starting point for the acceleration component.
2. An acceleration component: The task of this suite of algorithms is to rapidly guide the sequence of (59) to a capture zone of the accurate component.
3. An accurate component: The task of this component is to generate a solution that satisfies the precision requested by the user.

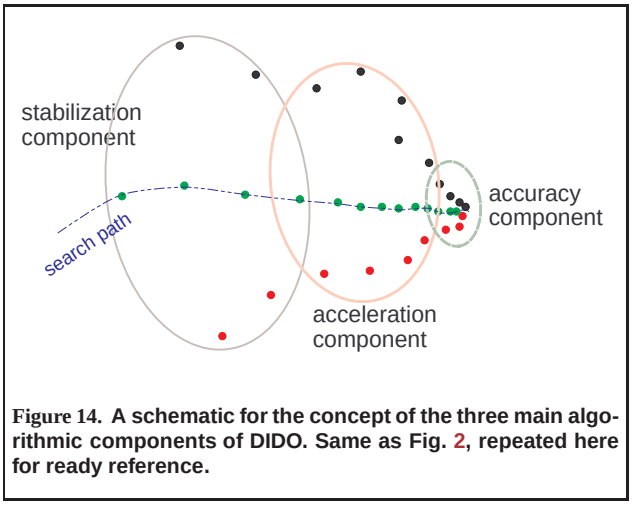

Figure 14. A schematic for the concept of the three main algorithmic components of DIDO. Same as Fig. 2, repeated here for ready reference.

The three-component concept is based on the overarching theory introduced in [48]. According to this theory, an algorithm may be viewed as a discretization of a controlled dynamical system where the control variable is a search vector or its rate of change[49]. In the case of static optimization, the latter case generates accelerated optimization methods that include Polyak's heavy ball method, Nesterov's accelerated gradient method and the conjugate gradient algorithm[49]. In applying this theory for dynamic optimization, the DIDO-sequence given by (59) may be viewed as a discretization of a continuous primal-dual dynamical system. The convergence theory proposed in [48, 49] rests on the Lyapunov stability of this primal-dual system. The specifics of a Lyapunov-stable algorithm rely on the local convexity of the continuous search trajectory. Because convexity cannot be guaranteed along the search path (see Fig. 2), a suite of algorithms is needed to support different but simple objectives. ***DIDO is based on the idea that the three components of stability, acceleration and accuracy achieve the requirements of a true-fast algorithm***. DIDO's main algorithm ties these three components together by monitoring their progress and triggering automatic switching between components based on various criteria as described in the subsections that follow.

E. Details of the Three Components of DIDO

Each of the three components of DIDO contain several subcomponents. Many of these subcomponents are described in detail across several published articles. Consequently, we lean heavily on these articles in describing the details of DIDO's algorithms.

1. *The Stabilization Component*

There are three main elements to the stabilization component of DIDO:

a) The Guess-Free Algorithm;

b) Backtracking and Restart; and

c) Moore-Smith Sequencing.

In the following, we describe the details of these three elements.

a) The Guess-Free Algorithm

In its nominal mode, DIDO is guess-free; i.e., a user does not provide any guess. Thus, one of the elements of the stabilization component is to generate the first point of the sequence denoted in (59). Because the guess-free element itself requires a starting point, this phase of the algorithm runs through a sequence of prioritized starting points and each starting point is used as an input for an elastic programming algorithm described in [7] and [8]. The second priority point is executed only if the first priority point terminates to an infeasible "solution." If all priority points terminate to infeasibility, then DIDO will execute a final attempt at feasibility by minimizing a weighted sum of infeasibilities. If this phase is not successful, then DIDO will present this solution to the user but tag it as infeasible. If, at any point, a feasible solution is found, the guess-free element of the algorithm terminates with a success flag. This point is denoted as $(X, U, V, \Lambda, \Omega)^{N_0}_{m_0}$, the first point of the DIDO sequence.

b) Backtracking and Restarts

The second element of the stabilization component is triggered if the monitor algorithm detects instability (i.e., divergence) in the acceleration component. The task of this element of the stabilization component is to use a "backtracked" point[q] to generate an alternative starting point for the acceleration component by reusing some of the elements of the guess-free algorithm. The production of this alternative point is performed through a priority suite of subalgorithms that comprise restarts and elastic programming[7, 8].

c) Moore-Smith Sequencing

The third element of the stabilization component is triggered if the monitor algorithm detects rapid destabilization in the acceleration component that cannot be handled by backtracking alone. In this situation, a backtracked point is used as a starting point to generate sequential perturbations in Sobolev space[32, 27] by generating continuous-time candidate solutions via interpolation. The perturbed discretized vector is used as starting point to generate a new sequence. If this new sequence continues to exhibit instability, the entire process is repeated until stability is achieved or the number of Sobolev perturbations exceeds a predetermined ceiling. In the latter case, DIDO returns the last iterate as an infeasible solution. The perturbations are also stopped if the projected variations exceed the bounds specified in the search space; see (5) in Sec. II. Because this sequence generation does not fit within the standard Arzelà-Ascoli theorem, we denote it as Moore-Smith[50, 51].

2. *The Acceleration Component*

The nominal phase of the acceleration component is to rapidly generate the DIDO sequence (Cf. (59)) subsequent to the first point handed to it by the stabilization component. DIDO's Hamiltonian algorithm (see Definition 5) produces this sequence. This algorithm uses Lemma 2 and a tentative value of δ^{λ,D,N_k} for each $N_k, k = 1, 2, \ldots$; see Remark 6. The values of δ^{λ,D,N_k} for $k > 2$ are revised based on the solution generated by the previous sequence and the predicted rate of convergence. The predicted rate of convergence is given by formulas derived in [8, 27]. Although the equations derived in [8, 27] are for Lagrange PS methods, it is not too difficult to show that they are valid for Birkhoff PS methods as well because of the equivalence conditions derived in [16].

[q]Because algorithmic backtracking in dynamic optimization can be computationally expensive, DIDO saves a small number of prior iterates that can be instantaneously recalled for rapid restarts.

The theory for the fast spectral algorithm presented in [8] is founded on an applications of the Arzelà-Ascoli theorem[50] to PS discretization[27]. Because the guarantees provided by Arzelà-Ascoli theorem are only for a subsequence, an implementation of this powerful result is not straightforward[32, 27]. Hence, we augment this result using the theory of optimization presented in [48]. According to this theory, the DIDO sequence denoted in (59) may be viewed as a discretization of a controllable continuous "search trajectory;" see Fig. 14. The dynamics of this search trajectory is given by a primal-dual controllable system that can be stabilized using a Lyapunov function. The Lyapunov function forms a merit function; and under appropriate technical conditions[48, 49], the search trajectory can be designed to drive an arbitrary point to a stable point. The stable point is defined in terms of the necessary conditions for optimality. For optimal control, these necessary conditions are given by Pontryagin's Principle[9]. A first-principles' application of these ideas to analyze a direct shooting method method is described in [18].

A guarantee of convergence is achieved in terms of local convexity of a Riemannian metric space that can be defined using a regularized Hessian;[r] see [49] for theoretical details. Because any point in an iteration can get trapped in a nonconvex region, subalgorithms are needed to detect and escape the trap regions. Detection of trap points can be made by monitoring the decrease in the Lyapunov function. Escape from suspected trap points is made by a call to the stabilization component of DIDO described previously.

An acceleration of the DIDO sequence, denoted by (59), is achieved through a combination of PS discretizations[5, 32, 16, 17, 46, 47]; see also Fig. 11. As shown by Theorem 3, the family of Birkhoff PS discretizations offers a special capability of tunneling through the cotangent barrier. Consequently, an accelerated spectral sequencing is achieved by coordinating the progress of the outer loop with the inner-loop sequence as stipulated in Definition 5. Perhaps the most important aspect of a Birkhoff PS discretization is that it offers a "flat" condition number of the resulting linear system[16, 15]; i.e., an $\mathcal{O}(1)$ growth with respect to N. In contrast, the condition number of a Lagrange PS discretization, be it Gauss, Gauss-Radau or Gauss-Lobatto, grows as $\mathcal{O}(N^2)$ no matter the choice of the orthogonal polynomial[16, 17]. Because condition number affects both accuracy and computational speed, its combination with its cotangent tunneling capability makes a Birkhoff PS discretization a doubly potent method to generate accelerated algorithms.

3. *The Accuracy Component*

Technically, this component of DIDO is the simplest. This is because the accuracy component of DIDO uses the starting point handed to it from a successful termination of the acceleration component. Consequently, an implementation of this algorithm reduces to a simple implementation of a quasi-Newton algorithm (or its accelerated version as presented in [49]) for solving (P^{λ,D,N_k}) for the last few values of N_k. The iterations of this algorithm are largely focused towards the goal of generating the targeted accuracy of satisfying the Pontryagin necessary conditions. To achieve additional primal-dual accuracy, this component of DIDO also tightens the tolerances on the Hamiltonian minimization condition that may not have been met at the termination of the accelerated algorithm. This procedure is done sequentially to maintain dual feasibility that is consistent with the value of $N = N_f$. Details of this process are described in [32, 45, 46, 47].

F. An Overview of the Nominal, True-Fast, Spectral Algorithmic Flow

Because the nominal mode of DIDO is guess-free; i.e., a user does not provide a guess, the first element of the stabilization component is invoked to generate the first element of the DIDO sequence given by $(X, U, V, \Lambda, \Omega)^{N_0}_{m_0}$; see (59). The acceleration component of DIDO rapidly iterates this point to completion or near-completion denoted by $(X, U, V, \Lambda, \Omega)^{N_a}_{m_a}$. If this point satisfies all the termination criterion, the main algorithm terminates; otherwise, $(X, U, V, \Lambda, \Omega)^{N_a}_{m_a}$ is handed

[r]Because the "exact" Hessian may not be positive definite, a "correction" can better ensure progress towards the Lyapunov-stable point[48].

off to the accurate component which iterates it to $(X, U, V, \Lambda, \Omega)_{m_f}^{N_f}$. Because $(V, \Omega)_{m_f}^{N_f}$ are internal variables, DIDO discards these virtual primal and dual variables. The remainder of the variables that support a solution to P^{λ, N_f} are passed to the user through the use of primal and dual structures described in Sec. II. For P^{λ, N_f} these variables are passed according to,

$$\text{primal.}\textbf{states} = X^{N_f} \tag{60a}$$

$$\text{primal.}\textbf{controls} = U^{N_f} \tag{60b}$$

$$\text{dual.}\textbf{dynamics} = \Lambda^{N_f} \tag{60c}$$

It is clear from the entirety of the preceding discussions that most of DIDO's algorithms are completely agnostic to the specifics of a PS discretization. Consequently, they can be adapted to almost any method of discretization!