A Tutorial for Applying DMOC to Solve Optimization Control Problems

Weizhong Zhang, Tamer Inanc
Department of Electrical and Computer Engineering
University of Louisville
Louisville, USA
w0zhan08@gwise.louisville.edu, t.inanc@louisville.edu

***Abstract*—This paper presents a detailed procedure to apply newly-proposed DMOC (Discrete Mechanics and Optimal Control) methodology to solve optimal control problems. DMOC is based on a direct discretization of Lagrange-d'Alembert principle for a system. First, this tutorial explains the principle of DMOC, and how to formulate the problem in DMOC. Next the steps are shown about how to install and configure nonlinear programming solver IPOPT, and how to use the modeling language AMPL. In particular, the user-defined function is involved with AMPL to solve a more complicated problem. Furthermore, a glider example is provided in this tutorial to solve optimal control problem with the user-defined 2D time-varying B-spline ocean current model. The ocean current original data was collected by HF-Radar stations located around Monterey Bay, CA in August 2000. Practically, this tutorial is shown how to use DMOC to solve optimal control problems with IPOPT and AMPL as the components. The possible users are robotic researchers, control system engineers, operations management researchers, and so on.**

***Index Terms*—Tutorial, Discrete Mechanics, Optimal Control, IPOPT, AMPL**

I. Introduction

Optimization control problems are widely available across different fields, from controlling a single ground or air robot to deploying a group of vehicles. The people tend to save as much as possible energy and effort to achieve the maximum benefits. This requires control community to provide methods to tackle these problems. Discrete Lagrangian mechanics and corresponding variational integrators are well developed mathematical methods [1] [2]. Based on those theories DMOC (Discrete Mechanics and Optimal Control) [3] is proposed to solve optimization control problems both for single unit and a group of units. Its application is versatile not restricted in control of mechanic systems, but also for any suitably modeled optimization problems, name a few, such as trajectory generation for a glider [4], control of Compass Gait Biped [5], formation of flying spacecrafts [6], or solving the variational problems in computer vision and graphics [8], poit vortices [9].

However, a complete tutorial for applying DMOC to solve optimal control problem is still unavailable regarding the technical details such as problem formulation, nonlinear programming solver selection and programming language choice. In this paper, a systematical procedure of applying DMOC to solve optimal control problems is proposed which includes detailed instructions to install the nonlinear programming solver IPOPT [10], modeling the problem in the language of AMPL (Adanced Mathematical Programming Language) [11], hooking the solver with the programming language, and emphatically make AMPL work with user-defined functions. An explicit example for using DMOC is given to control a glider moving in the 2D time-varying B-spline ocean current model [12].

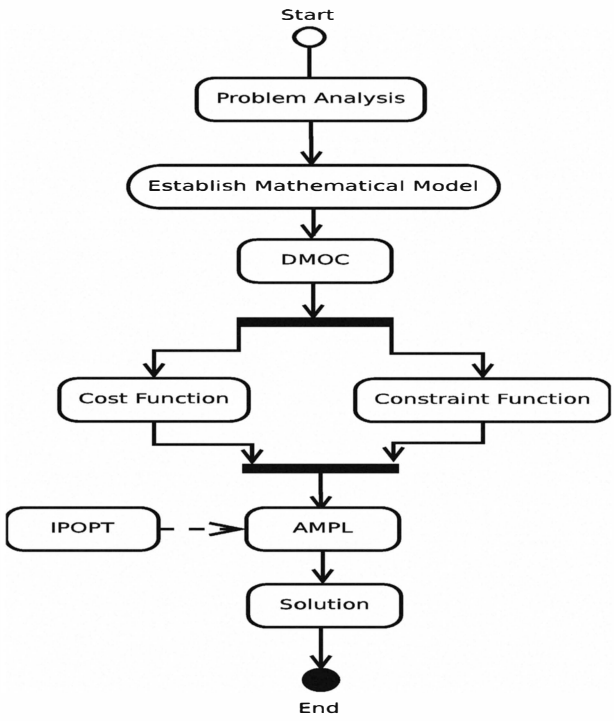

Fig. 1. DMOC procedure to solve the optimization problem.

The paper is organized as follows, in Section II, optimal control problem is defined. Then in order to solve the optimization control problem, DMOC is briefly explained in Section III for reader's convenience. IPOPT (Interior Point Optimizer) as an open source solver for large scale nonlinear optimization problems is introduced in Section IV. Section V presents AMPL which is applied to model optimal control problems. The implementation details about installation and configuration of IPOPT and AMPL are given in Section VI.

978-1-4244-6588-0/10/$25.00 ©2010 IEEE

With all the preceding preparations, in Section VII the control problem of a glider simulated in Monterey Bay is shown to be solved by the DMOC technique with IPOPT and AMPL. The conclusion is given in Section VIII with future work directions.

II. Problem Definition

Assume the problem of a system under investigation has been modeled as an optimization control problem, the cost function at time t is listed as

$$\mathbf{J_f(t)} = \mathbf{f(q(t), \tau(t), t)} \tag{1}$$

where $q(t)$ is the state of the system, it can be further represented as $q(t) = (q_1(t), q_2(t), ..., q_n(t))$ assuming it has n states. $\tau(t)$ is the control vector which composes of m componets as $\tau_1(t), \tau_2(t), ..., \tau_m(t)$ with $m \leq n$. The constraints for the system are

$$\mathbf{bu} \geq \mathbf{C(t)} = \mathbf{g(q(t), \tau(t), t)} \geq \mathbf{lu} \tag{2}$$

where bu and lu are the bound vectors for the constraints. $C(t)$ is a vector consists of k constraint functions, as to say the constraints are $g_1, g_2, ..., g_k$ for $g(q(t), \tau(t), t)$. where τ is the multiple k-variable vector for the constraint functions. For the unequal constraints in (2), by introducing slack variables [13], it can be transformed into the equal constraint functions, say,

$$\mathbf{g_e(q(t), \tau(t), t)} = 0 \tag{3}$$

Therefore, the Lagrangian function for this optimization problem (1) and (2) is

$$\mathbf{L(q(t), \tau(t), t)} = f(q(t), \tau(t), t) + \lambda^T \cdot g_e(q(t), \tau(t), t) \tag{4}$$

Where T indicates to transpose the vector. Considering to extremize the cost function $J(t)$ over the time period of t_0 to t_f. The cost function is shown as:

$$\mathbf{J} = \int_{t_0}^{t_f} f(q(t), \tau(t), t)dt \tag{5}$$

By considering (3) and (4), the cost function (5) can be transformed into (6).

$$\mathbf{J} = \int_{t_0}^{t_f} L(q(t), \tau(t), t)dt \tag{6}$$

Thus the optimization control problem is represented by (6) and (3).

III. DMOC

DMOC procedure is briefly listed here for reader's convenience. To solve the problem described in Section II, standard methods such as shooting [14], collocation [15] need to derive the Euler-Lagrange equation from the cost function. The DMOC techniques directly discretize the cost function by approximating the integration. The constraints are also discretized, specifically, some of the constraints of mechanical system are derived from Lagrange-d'Alembert principle, for details see [3]. The approximation of the integration (6) is dependent on the rule set up by the user, for example, the states $q(t)$ is discretized into $q(0), q(h), q(2h), ..., q(Nh)$, h is the step size, N is the number of the steps in the specified time period of t_0 to t_f. Thus the discrete Lagrangian L_d:

$$\mathbf{L_d(q_k, \tau k, k)} \approx \int_{kh}^{(k+1)h} L(q(t), \tau(t), t)dt \tag{7}$$

With (6) and (7), the integration of the cost function J_d is expressed as

$$\mathbf{J_d} = \sum_{k=0}^{k=N-1} L_d(q_k, \tau k, k) \tag{8}$$

The discretization of the constraints functions (3) is

$$\mathbf{g_e(q_k, \tau_k, k)} = 0 \tag{9}$$

where $k = 0, ...N$, q_k and τ_k are the discrete states and control forces. When the DMOC is applied to the specific mechanic control problems [3], the constraints as Euler-Lagrange equations have the specific formulation which are derived from Lagrange-d'Alembert principle, the discrete forces are introduced as the concept. Since the optimization control problem has been discretized as (8) and (9), the finite dimensional nonlinear programming problem now can be solved by standard method like sequential quadratic programming (SQP) [16]. IPOPT is that kind of solver for this optimization problem, which this paper prefers.

IV. IPOPT

IPOPT is an open source nonlinear programming solver developed by Dr Andrew Wächter as his Ph.D thesis project in Carnegie Mellon University. It is a primal-dual interior-point [17] algorithm with a filter line-search method. IPOPT has been proved attractable using CUTEr test set (954 problems), compared with other two interior-point optimization codes KNITRO and LOQO [10]. In IPOPT, an original optimal control problem is transformed into a sequence of barrier (interior-point) problems for a decreasing sequence of barrier parameters converging to zero. Equally, it can be interpreted as a primal-dual equations by applying a homotopy method [18] with the homotopy parameter which is driven to zero. IPOPT includes a line-search filter method with the feasibility restoration phase, second-order corrections which are supposed to improve the proposed step if a trial point has been rejected, and initial correction of the Karush-Kuhn-Tucker matrix which is the necessary optimal condition for nonlinear programing. For the IPOPT algorithm details, see [10].

IPOPT package is available from COIN-OR (www.coin-or.og) under the Common Public License. The user can download and use it free of charge even for commercial purposes. Some third party components are required for the execution of IPOPT, these components consist of BLAS (Basic Linear Algebra Subroutines), LAPACK (Linear Algebra PACKage), a sparse symmetric indefinite linear solver such as MA27 or other one. While only ASL (AMPL Solver Library) is required for using with AMPL. The software sources including the dependent solvers are located in the website, the detailed procedure to download and install IPOPT can be found in the IPOPT manual [19].

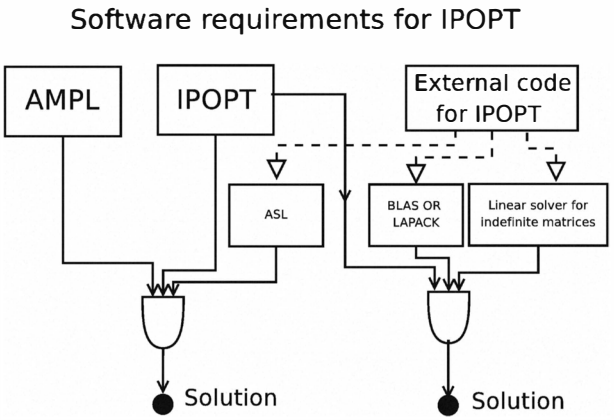

Fig. 2. Software requirements for IPOPT, where ALS is referred to AMPL Solver Library, BLAS represents Basic Linear Algebra Subroutines, LAPACK means Linear Algebra PACKage, one Liner solver for indefinite matrices can be MA27, MA57 or other solvers, the details is described in [19].

After installation, the users may model their own problem in nonlinear programing formulation, then interface the problem with IPOPT through code such as C++, C, or Fortran. For programming problems in C++ interfacing with IPOPT, the user must provide the Jacobian matrix, and Hessian matrix which may be approximated by setting up the IPOPT option "hessian_approximation" as "limited-memory". The eight functions need to be implemented to define the problem and supply the information, the eight functions are get_nlp_info(), get_bounds_info(), get_starting_point(), eval_f(), eval_g(), eval_jac_g(), eval_h(), finalize_solution() separately. As their names indicate, the functions provide the IPOPT with the necessary information like the number of variables, the bounds, the starting points, the constraints Jacobian, the Hessian of the Lagrangian for the solver to generate the solution for the problem. But the difficulty exists in the Jacobian and Hessian parts. The Hessian matrix can be approximated and this eval_h() can be disregarded. Finally we still have to provide the IPOPT with the Jacobian of the constraints, which is not easy. For example, in our problem, the constraints have 104 equations, and the number of the variables is 151, therefore, the Jacobin of the constraints has the dimension of 104×151, every element in the matrix needs to be specified, even there is some kind of principle implicit in these elements, it is still not easy to avoid the mistakes for constructing the Jacobian matrix in the programming.

Therefore, the easier way for IPOPT to solve the problem is to interface it with AMPL, because AMPL automatically provide some necessary information to IPOPT to solve the problem, the information include and not limited to the Jacobian, Hessian matrix.

V. AMPL

AMPL, developed in Bell Laboratories, is a comprehensive and powerful algebraic modeling language for linear and nonlinear, continuous or discrete system optimization problems. It is user friendly, making the user focus on the modelling of the problem, not the technical details for programming. All the variables, parameters, cost functions, constraint functions are defined intuitively and straight forward. The main difference between AMPL with other programming languages such as C or Fortran are the expressions of the variables. In AMPL, "set" and "index" are used to invoke the specific variable. On the other hand, the mathematical expression is generally adapted from an advanced programming language, for example, "sum" or ">" and so on as arithmetic or logical operators are used. Since AMPL is based on algebratic expressions of constraints and objectives. Its syntax is easily learned by referring to the manual [11] or through some examples [20]. With the AMPL scripture of the program, the AMPL translator can read the optimization model and data provided through the language. The seven logic phases are executed like *parse, read data, compile, generate, collect, presolve, out*. The AMPL needs call the solvers to generate the solution for the formulated problem.

VI. Implementation Details

A. IPOPT Installation

At first step, IPOPT can be downloaded from COIN-OR [22] by retrieving the IPOPT tarball code. The latest (Apr 26, 2008) c++version of IPOPT tarball is $Ipopt$-3.4.0.tgz. Assume the tarball is downloaded to the folder $Program/IPOPTtutorial$. Unpack the archive file by $gunzip\ IPOPT$3.4.0.tgz, resulting into $IPOPT$3.4.0.tar. Using $tar\ xvf\ IPOPT$3.4.0.tar, the tarball is extracted into $IPOPT$3.4.0.tar. For convenience, you may change the name of the directory $IPOPT$3.4.0 to $CoinIpopt$. According to Fig. IV, IPOPT needs a few external packages to make it work, including AMPL solver library–ASL, basic linear algebra subroutines–BLAS or Linear Algebra Package–LAPACK, a linear solver for symmetric indefinite matrices such as MA27 or MA57. If IPOPT is used with AMPL as it is in our example, only ASL is required. However, IPOPT can work independently from AMPL, so the procedure to download BLAS, LAPACK, MA27 is listed in the following by utilizing the scripts included in the IPOPT distribution.

- $cd\ CoinIpopt/ThirdParty/Blas$ go to the Blas directory
- $./get.Blas$ run the script to download BLAS from the Netlib Repository, after succession, the message "Done downloading the source code for BLAS" appears.
- $cd\ ../Lapack$ go to the Lapack directory
- $./get.Lapack$ download Lapack, get the message "Done downloading the source code for LAPACK".
- $cd\ ../ASL$ go to the ASL directory
- $./get.ASL$ download ASL, get the message "Done downloading the source code for ASL".

For the sparse symmetric linear solver MA27, go to $http://hsl.rl.ca.uk/archive/hslarchive.html$ to register and get MA27. save $ma27ad.f$ to $CoinIpopt/ThirdParty/HSL$. As it is indicated before, you may use other linear solver for symmetric indefinite matices instead of MA27. After the third party codes are installed, IPOPT needs to be compiled and installed by the generally command.

- *cd CoinIpopt*
- *./configure* get the message "configure: Configuration of IPOPT successful,configure: Main configuration of Ipopt successful"
- *make*
- *make install*

After the proceeding procedures, the IPOPT is successfully installed in $CoinIpopt/Ipopt$, now you may begin to test the examples to make sure it work. For instance, if we go to $CoinIpopt/Ipopt/examples/Cpp\_example$, type *make*, then *./cpp_example*, the screen output should be "Optimal Solution Found.*** The problem solved in 6 iterations!*** The final value of the objective function is -4.000000e+00". Generally, it means that IPOPT is ready to use for solving your optimization problem.

B. AMPL Installation

The easiest way to make IPOPT solve your optimization problem is to make it work with AMPL, even also you can program your own problems in c, c++ or Fortran language. In this paper, we are considering the problem to make IPOPT work with AMPL. Firstly, AMPL can be downloaded from $http : //www.ampl.com/DOWNLOADS$ without any charges if the variables are less than 300. In our case, the experimental system is HP Pavilion a1430n, Memory 2.0 GB, AMD Athlon(tm) 64 $\times$ 2 Dual Core Processor 3800+. The operating system is Ubuntu 7.10, Kernel Linux 2.6.22-14-386. Thus "Intel (Pentium-compatible) PCs running Linux" AMPL is downloaded to $Ipopt/examples/AMPLex$. Using $gunzip\ ampl.gz$ to uncompress the file into $ampl$, by typing $chmod\ +x\ ampl$ to make sure you have the privilege to execute the AMPL. Then AMPL is ready to use when you model your problem in AMPL and the problem is in the format of $test.mod$ in which the solver is specified as IPOPT. $./ampl\ test.mod$ is used to solve the problem, the output can be saved into a file and shown on the screen.

C. User-Defined Function

The useful feature of AMPL is that it can include user-defined function as externally added function to solve more complex problems. In order to make user-defined function work with AMPL, the "funcadd.c" should be download from the server [20], and modify it according to your purpose, basically the user needs to embed his or her own program to the downloaded function making the user-defined function work like the example function. Then compile the "funcadd.c" by the different makefile which is dependent on the work station where the program is supposed to execute. You can download the makefile from AMPL website and modified it according to your system. The Makefile.Linux listed in the appendix in the appendix is tested successfully in Ubuntu 7.10, Kernel Linux 2.6.22-14-386, Memory 2.0 GB, AMD Athlon(tm) 64 $\times$ 2 Dual Core Processor 3800+. After the funcadd.c is compiled by make f Makefile.Linux, the amplfunc.dll will be created, now the user-defined function is ready to be called during the optimization process.

D. Hook AMPL with IPOPT

AMPL can hook with different kinds of solvers such as ACRS, MINOS, NPSOL, IPOPT and so onto generate the solution for the optimization problem. The "solve" command in AMPL language make AMPL send the problem information to the solver which is regarded as a separate program, then read the solution back from the solver. The files for the communication between AMPL and the solver is called *stub.suffix* [21]. At the beginning, the initial file from AMPL is *stub.nl* which describe the problem information, after solver received this information and with the specified toleration and iteration parameters, the solver write the solution or resulting information to a file named *stub.sol*. Practically, in order to make AMPL work with IPOPT, the user just need to install AMPL to the right directory, then specify solver option in AMPL program as "option solver IPOPT" at the beginning of the program.

VII. Example

The following example is shown how DMOC work with IPOPT and AMPL to solve the optimal control problem. A dynamical glider (10) is simulated moving in Monterey Bay from $(-122.1780, 36.8557)$ to $(-122.2420, 36.6535)$ which represents the position in degrees (Longitude and Latitude), it is controlled by the gyroscopic forces (11). The AMPL program

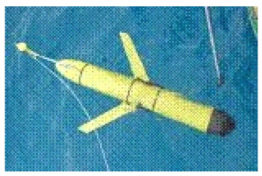

Fig. 3. The SLOCUM glider [23]

listed in the following with the solver specified as IPOPT. For calculation purpose, the position unit is transformed into centimeter based on the reference point as $(-122.3246, 36.5658)$. The ocean current flows are modeled as time-varying 2D B-spline models, optimal trajectory generation with NTG for the glider in this model has been presented in [12].

$$\begin{array}{rcl} \ddot{\mathbf{q_1}} & = & -V\frac{d\theta}{dt}\sin\theta + \dot{u} \\ \ddot{\mathbf{q_2}} & = & V\frac{d\theta}{dt}\cos\theta + \dot{v} \end{array} \tag{10}$$

where u, v are the ocean current velocities from B-spline model in q_1(Longitude), q_2(Latitude) direction respectively. V, θ are the glider speed, orientation respectively. The gyroscopic force:

$$\mathbf{F_{gyr}} = \begin{pmatrix} -\frac{d\theta}{dt}(\dot{q_2} - v) \\ \frac{d\theta}{dt}(\dot{q_1} - u) \end{pmatrix} \tag{11}$$

The time-varying 2D ocean current model is presented in (12) mathematically, its figures at $t = 13hours$ are shown in Fig. 4.

$$\begin{array}{rcl} \mathbf{u(x,y,t)} & = & \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{o} B_{i,k_{ux}} B_{j,k_{uy}} B_{k,k_{ut}} a_{ijk} \\ \mathbf{v(x,y,t)} & = & \sum_{i=1}^{p}\sum_{j=1}^{r}\sum_{k=1}^{s} B_{i,k_{vx}} B_{j,k_{vy}} B_{k,k_{vt}} b_{ijk} \end{array} \tag{12}$$

where a_{ijk} andb_{ijk} represent coefficients of B-spline for $u(x,y,t)$ and $v(x,y,t)$, respectively. $B_{i,k}$, $B_{j,k}$ and $B_{k,k}$ represent B-spline basis functions for the $x-$, $y-$ and $t-$ direction, respectively. The order of the polynomials used were $k_{ux} = k_{uy} = k_{vx} = k_{vy} = k_{ut} = k_{vt} = 4$ and the number of the coefficients were $m = p = 32$, $n = r = 22$ and $o = s = 25$, which are determined by the original data. Here x, y are representing q_1, q_2 respectively.

In AMPL, the modeled problem is shown as follows:

- Cost function:

$$\text{minimize force\_energy:}$$

$$\text{sum } \{\text{j in 0..N-2}\}0.5 * (f_1^+[j] * f_1^+[j] +$$

$$f_1^-[j] * f_1^-[j] + f_2^+[j] * f_2^+[j] + f_2^-[j] * f_2^-[j]) * h;$$

- Start and final Constraints:

$$\text{subject to x\_start:} q_1[0] = a_x;$$

$$\text{subject to y\_start:} q_2[0] = a_y;$$

$$\text{subject to x\_destination:} q_1[N-1] = b_x;$$

$$\text{subject to y\_destination:} q_2[N-1] = b_y;$$

- Trajectory Constraints:

$$\text{subject to Euler\_Lagrange\_x } \{\text{j in 0..N-3}\} :$$

$$-KEq1p[j+1]+KEq1p[j]+0.5*h*(KEq1[j]+KEq1[j])+$$

$$0.5 * h * (Vq1[j+1] + Vq1[j]) + f_1^+[j] + f_1^-[j] = 0;$$

$$\text{subject to Euler\_Lagrange\_y } \{\text{j in 0..N-3}\} :$$

$$-KEq2p[j+1]+KEq2p[j]+0.5*h*(KEq2[j]+KEq2[j])+$$

$$0.5 * h * (Vq2[j+1] + Vq2[j]) + f_2^+[j] + f_2^-[j] = 0;$$

where f_1^-, f_2^-, f_1^+ and f_2^+ are the left and right discrete forces [3]. for the components of the gyroscopic force.$F_{gyr}(11)$. N is the number of knots in the trajectory. a_x,a_y,b_x,and b_y are the starting and destination point, it has been transformed in the program to the centimeter assuming the radius of the earth is $6378km$ and the earth is a perfect sphere. The trajectory constraints are introduced because the glider is controlled by the gyroscopic force and its motion is satisfied with Lagrange-d'Alembert principle. These constraints are called discrete Euler-Lagrangian equations [3]. In the trajectory constraints, $KEq1p$, $KEq2p$ is the derivative of the glider kinetic energy according to $\dot{q_1}$, $\dot{q_2}$ respectively. $Vq1$, $Vq2$ is the potential energy of the glider. The index of all these variables shows the states are discrete in the program. For the glider travels in the ocean current. The function is defined in AMPL in the way shown in the following:

- Define function:

$$\text{function splineinfo;}$$

- Call function to retrieve the ocean current velocities:

$$\text{var u } \{i \text{ in } VEL\_NODES\}$$

$$= \text{splineinfo}(x[i], y[i], h * i/3600, u, v);$$

$$\text{var v } \{i \text{ in } VEL\_NODES\} = v;$$

- Discrete forces are connected with currents:

$$\text{var } f_1^+\{i \text{ in } VEL\_NODES\}$$

$$= 0.5 * h * (-taum[i] * (q2p[i] - v[i]));$$

$$\text{var } f_1^-\{i \text{ in } VEL\_NODES\}$$

$$= 0.5 * h * (-taum[i] * (q2p[i] - v[i]));$$

$$\text{var } f_2^+\{i \text{ in } VEL\_NODES\}$$

$$= 0.5 * h * (taum[i] * (q1p[i] - u[i]));$$

$$\text{var } f_2^-\{i \text{ in } VEL\_NODES\}$$

$$= 0.5 * h * (taum[i] * (q1p[i] - u[i]));$$

where $VEL\_NODES = \{0, ...N-1\}$, h is the step size of optimization, $taum$ is the control force, $q2p$, $q1p$ is the derivative approximation of $q1$ and $q2$. The index of variables show that the states are discrete. By combining DMOC,

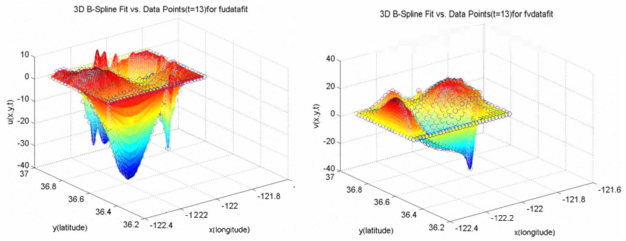

Fig. 4. Time-varying 2D ocean current B-spline model at t=13 [12], the above two figures show the ocean current velocities at x and y directions correspondingly at that moment.

IPOPT, AMPL and time-varying 2D B-spline ocean current model, the glider trajectory is generated and shown in Fig. 5.

As shown in Table. I, DMOC has successfully generated

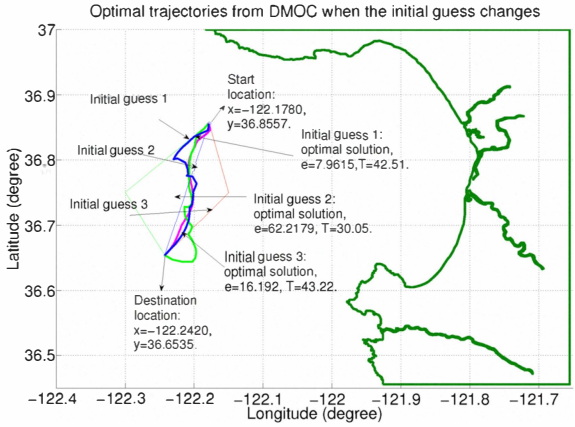

Fig. 5. Dynamical glider trajectory in time-varying 2D ocean current model.

the local solutions for the optimal control of the glider with a complex ocean current model. It illustrates the promising aspects of DMOC methodologies combined with IPOPT to

TABLE I
DMOC FOR GLIDER IN COMPLEX OCEAN MODEL.

$DMOC$	$Interval$	$T(hours)$	$Iter$	$Time(s)$	$EnergyCost$
$Guess$ 1	40	42.51	3000	23.10	7.9615
$Guess$ 2	42	30.05	3000	43.30	62.2179
$Guess$ 3	40	43.22	3000	23.10	16.192

solve other optimization control problems. Furthermore, an efficient method is needed to choose a better solution from local solutions. The obtained solution will be an approximate global solution for the optimal control problem.

VIII. CONCLUSION

In this paper, a tutorial on solving optimal control problems with DMOC is presented. It is shown that DMOC combined with AMPL and IPOPT can solve complex optimal control problems. Especially, it is shown that user-defined functions can be invoked in this procedure. The fundamentals of IPOPT and AMPL are explained and procedure to solve problems is presented. As an example, a dynamic glider is simulated moving in Monterey Bay California where the ocean current is modeled as time-varying 2D B-spline function. The minimizing energy local solution trajectories are obtained by DMOC methodology. Consequently, this tutorial proposes a feasible approach and procedure to solve optimal control problems with available resources including DMOC methodology, the open source IPOPT, the AMPL with a free student version with 300 variables limit. Further research will propose an efficient method to choose an approximate global optimal solution from a set of local solutions. The potential users of this tutorial may include robotics researchers for designing operational trajectories with DMOC, or control system engineers to design optimal control methods, even operation management researchers.

ACKNOWLEDGMENT

This work has been supported by KY NASA EPSCoR grants WKURF 5162020909 and WKURF 5968550802. Thank Prof. Jerry E Marsden and Dr Sina Ober-Blöbaum from California Institute of Technology for giving us valuable suggestions on this paper. Thank IPOPT author Dr A. Wächter. Thank Dr R. Vanderbei for posting AMPL nonlinear programing model examples for reference. Thank conference reviewers and chairs for precious time and constructive advices.

REFERENCES

[1] J. E. Marsden, and M. West, "Discrete mechanics and variational integrators," *Acta Numerica*, vol. 10, Cambridge University Press, 2001.

[2] S. Lall, and M. West, "Discrete variational Hamiltonian mechanics," *Journal of Physics A: Mathematical and general*, 39, 2006, pp. 5509–5519.

[3] O. Junge, J. E. Marsden, S. O. Blőbaum, "Discrete mechanics and optimal control,"in *Proc. IFAC World Congress*, 2005.

[4] W. Zhang, T. Inanc, S. Ober-Blöbaum, and J. E. Marsden, "Optimal trajectory generation with DMOC versus NTG: Application to a glider," submitted to *47th IEEE Conf. on Decision and Control*, Cancun, Mexico, on Dec 9-11, 2008.

[5] D. Pekarek, A. D. Ames, and J. E. Marsden, "Discrete mechanics and optimal control applied to the compass gait biped," in *Proceedings of the 46th IEEE Conf. on Decision and Control*, New Orleans, LA, USA, Dec 12-14, 2007, pp. 5376–5382.

[6] O. Junge, J. E. Marsden, and S. Ober-Blöbaum, "Optimal reconfiguration of formation flying spacecraft–a decentralized approach," in *Proceedings of the 45th IEEE Conf. on Decision and Control*, San Diego, CA, USA, Dec 13-15, 2006.

[7] E. Arnold, J. Neupert, O. Sawodny, and K. Schneider, "Trajectory tracking for boom cranes based on nonlinear control and optimal trajectory generation," in *Proceedings of the 41st IEEE Conf. on Decision and Control*, Las Vegas, Nevada USA, Dec 2002, pp.1521–1527.

[8] M. Desbrun, A. N. Hirani, and J. E. Marsden, "Discrete Exterior calculus for variational problems in computer vision and graphics," in *Proceedings of the 42nd IEEE Conf. on Decision and Control*, Maui, Hawaii USA, Dec 2003, pp. 4902–4907.

[9] C. W. Rowley, and J. E. Marsden, "Variational integrators for degenerate Lagrangians, with application to point vortices," in *Proceedings of the 41st IEEE Conf. on Decision and Control*, Las Vegas, Nevada USA, Dec 2002, pp. 1521–1527.

[10] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming*, 106(1), pp. 25–57, 2006.

[11] R. Fourer and D. M. Gay and B. W. Kernighan, "A modeling language for mathematical programming," *Manage. Sci.*, vol. 36, pp. 519–554, 1990. Available: http://www.ampl.com

[12] W. Zhang, T. Inanc, S. Ober-Blöbaum, J. E. Marsden, "Optimal trajectory generation for a glider in 2D and time-varying ocean flows B-spline model," presented at *Int. Conf. on Robot. Autom.*, Pasadena, California, on May 19-23, 2008.

[13] K. Tanaka, "A formal linearization procedure of constrained nonlinear automatic lens design problems II. use of slack variables and Lagrangian multipliers," *J. Optics (Paris)*, vol. 22, pp. 7-9, 1991.

[14] G. A. Hicks and W. H. Ray, "Approximation methods for optimal control systems," *Can. J. Chem. Engng*, 49, pp. 552–528, 1971.

[15] O. V. Stryk, "Numerical solution of optimal control problems by direct collocation," *Optimal Control Calculus of Variations, Optimal Control Theory and Numerical Methods*, Int. Ser. Number. Math. 111, Birkhaäuser, Basel, pp. 129–143, Freiburg, 1993.

[16] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming for large-scale nonlinear optimization", *J. Computational and Applied Mathematics*, vol. 124, pp. 123–137, 2000.

[17] S. J. Wright, "Primal-dual Interior-Point Methods ," Society for Industrial Mathematics, Jan 1, 1987.

[18] I. Drori, D. L. Donoho, "Solution of l1 minimization problems by LARS/Homotopy methods," in *Proceedings of IEEE Inte. Conf. on Acoustics, Speech and Signal Processing*, 2006, pp. III 636-639.

[19] A. Wächter, "Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT, "[Online]. Available: http://www.coin-or.org/Ipopt/documentation/

[20] R. Vanderbei, "Nonlinear optimization models," Princeton University, [Online]. Available:http://www.sor.princeton.edu/ rvdb/ampl/nlmodels/

[21] D. M. Gay, "Hooking your solver to AMPL," Technical report, Bell Laboratories, Murray Hill, NJ (1993; revised 1994, 1997). [Online]. Available:http://www.ampl.com/REFS/abstracts.html#hooking2

[22] COmputational INfrastructure for Operations Research, Available:https://projects.coin-or.org/Ipopt

[23] Stommel,H.1989."The Slocum Mission", *Oceanography*, 2(1), 22-25.

APPENDIX

```
# Makefile.Linux
.SUFFIXES: .c .o # $S = ampl/solvers directory S = .. CC
= cc CFLAGS = -I$S -O2 .c.o: $(CC) -c $(CFLAGS) $*.c
amplfunc.dll: funcadd.c $(CC) -c $(CFLAGS) -fPIC fun-
cadd.c $(CC) -shared -o amplfunc.dll funcadd.o
## Sample solver creation...
# $(myobjects) = list of .o files myobjects = ....
mysolver: $(myobjects) $(CC) -o mysolver $(myobjects)
$S/amplsolver.a -lm -ldl
```