# Local Pursuit as a Bio-Inspired Computational Optimal Control Tool

D. Hristu-Varsakelis[†*], C. Shao[‡], and N. Samaras[†]

*Abstract*— **This work explores the use of a cooperative optimization algorithm, known as "local pursuit", as a numerical tool for computing optimal control-trajectory pairs. Local pursuit is inspired by the foraging activities of ant colonies and has been the focus of recent work on cooperative control. We present a hybrid approach to numerical optimization that combines local pursuit with existing methods, and solves an optimal control problem in many small pieces, in a manner to be made precise. Our approach can overcome some important limitations of traditional nonlinear programming based techniques, leading to the ability to handle larger problems with the same resources, while avoiding difficulties due to ill-conditioning. The performance of our method is illustrated in a numerical example involving optimal orbit transfer of a simple satellite.**

*Index Terms*— cooperative control, computational methods, agents and autonomous systems, bio-inspired optimization

## I. Introduction

Over the past decade, researchers have increasingly looked to cooperative strategies as a means for addressing systems problems which are difficult to solve at the level of an individual system [22], [6]. The push for cooperation coincides with the maturation of technology that makes it possible to develop meaningful interactions among, say, groups of wheeled or aerial vehicles, and is partly motivated by the obvious success of biological collectives which have evolved to perform as "more than the sum of their parts" [8], [16]. Notable examples include worker bees, which share information by "dancing" and distribute themselves among flowers according to the "profitability" of each location; schools of fish that swim in agile, tight formations; and ants, which use pheromone secretions for recruiting nest-mates and for discovering efficient paths between their nest and food sources [4]. Observations of these and other natural collectives have motivated a series of works on the modeling of movement in animal groups[4], [2], [12] as well as other work on cooperative control strategies, including distributed searching [22], [17], estimation [18], [14], and optimization [6], [3], [9].

This paper investigates the use of a biomimetic cooperative control strategy for numerical optimal control, in problems where analytical solutions are not possible. The strategy, known as "local pursuit", was originally proposed in [2], and was recently developed further [9], [20], [11] for a broad class of optimal control problems, and systems with nonlinear dynamics. Local pursuit mimics the way in which ant colonies appear to optimize their foraging movements, gradually transitioning from a few ants traveling on randomly-discovered paths, to columns of thousands of individuals, marching on shorter — sometimes globally optimal — trails. Under local pursuit, members of a group of dynamical systems (termed "agents") evolve along a similar formation. Starting from an initial feasible trajectory, each agent iteratively improves upon its predecessor by always moving to intercept her along an optimal trajectory. This process requires only local interactions with nearby agents, and allows the group to solve an optimal control problem in many short pieces, in a manner to be made precise shortly.

The ability to compose an optimal solution by solving a sequence of smaller (and perhaps easier) subproblems is a key feature of local pursuit and will prove computationally useful. Given a control system which must be steered optimally between two states in a fixed amount of time, one typically begins by sampling the system trajectory, and solves a Non-linear Programming (NLP) problem to determine the best placement of the samples in the state space, subject to constraints which include the equations of motion and continuity [1], [5], [7]. In that setting, there is a balance to be struck between the number of trajectory samples and the size of the time intervals that separate them. With a small number of samples, the corresponding NLP problem has reasonable storage requirements (they are typically quadratic in the number of states and sampling density), but propagating the state vector from one sample point to the next may require high-order approximation of the equations of motion, and thus become time-consuming. On the other hand, if the trajectory is sampled densely, then there may not be adequate memory to solve the associated NLP problem, which at the same time is prone to a greater accumulation of numerical errors.

Up to now, discussions of local pursuit have focused on the algorithm's limiting behavior and proof of convergence to an optimal solution. The contribution of this work is: i) to explore local pursuit in a computational setting, and ii) to construct a bio-inspired numerical method for optimal control by combining a recently-reported version of local pursuit with existing numerical techniques. Compared to traditional NLP-based approaches [1], ours will make it possible to solve much larger problems with higher accuracy, given a fixed amount of storage, by solving a sequence of lower-dimensional problems instead. The use of cooperation will allow us to always operate in the domain of manageable-sized problems while still being able to handle very high sampling densities, at the expense of increased running time. At the same time, our method extends the domain of

†Department of Applied Informatics, University of Macedonia, Thessaloniki 54006, Greece. {dcv,samaras}@uom.gr

‡SAC Capital Advisor LLC, New York, NY 10002, USA Cheng.Shao@sac.com

* Corresponding author. Tel: +30-2310-891721, Fax: +30-2310-891290

applicability of existing techniques and can reduce errors due to ill-conditioning of the NLP problem to be solved.

The remainder of this paper is organized as follows: In Section II we fix notation and describe the optimal control problem under consideration. Section III reviews the basics of numerical trajectory optimization. Section IV describes local pursuit, a bio-inspired cooperative control algorithm (from [9], [20]) which uses multiple copies of a dynamical system to solve a broad range of optimal control problems. Section V discusses the potential advantages of a hybrid method which combines local pursuit with existing numerical techniques (specifically, direct multiple shooting), and which employs cooperation in order to solve an optimal control problem with reduced resources. In Section VI we compare the performance of multiple shooting to that of our proposed hybrid method in a satellite orbit transfer problem.

## II. Problem Statement and Notation

This work is concerned with the following optimization problem:

*Problem 1:* Let

$$\dot{x} = f(x,u),\ x(t) \in \mathbb{R}^n, u(t) \in \Omega \subseteq \mathbb{R}^m. \tag{1}$$

Find a trajectory-control pair $x^*(t), u^*(t)$, and a final state $x^*(T)$, ($T$ fixed) that minimize

$$\begin{aligned} J(x,u,T) &= \int_0^T g(x,u)dt + G(x(T)), \\ s.t. &\quad x(0) = x_I \text{ fixed},\ Q(x(T)) = 0, \end{aligned} \tag{2}$$

where $g(x(t),u(t)) \geq 0,\ G(x) \geq 0$, and $Q(\cdot)$ is an algebraic function of the state.

We are specifically interested in instances of Problem 1 which have no analytical solution (e.g., due to nonlinearities in $f(x,u)$), so that a computational approach is required. For simplicity, we consider only the case where the final time $T$ is fixed. The discussion that follows can be easily modified to apply to free-final-time problems as well. See [20], [11] for details.

We will make use of the following notation.

*Definition 1:* Given the final state constraint $Q(x) = 0$, the *constraint set* of $x$ is

$$S_Q = \{x \in \mathbb{R}^n | Q(x) = 0\}.$$

We will take $\partial Q/\partial x$ to have constant non-zero rank in a neighborhood of the set $S_Q$.

For any pair of fixed states $a, b \in \mathbb{D} \subseteq \mathbb{R}^n$, let $u^*(t), x^*(t)$ be the optimal control/trajectory pair for steering (1) from $a$ to $b$ in $T$ time units. Then, the cost of following $x^*(\cdot)$ is denoted by:

$$\eta(a,b,T) \triangleq \int_0^T g(x^*(t),u^*(t))dt \tag{3}$$

subject to $x^*(0) = a,\ x^*(T) = b$.

Now, let $x^*(t)$ be the optimal trajectory (over $T$ time units) from an initial state $a$ to the constraint set $S_Q$. The cost of following $x^*(\cdot)$ is denoted by:

$$\begin{aligned} \eta_Q(a,T) &\triangleq \int_0^T g(x^*,u^*)dt + G(x^*(T)) \\ &= \min_{x,u} J(x,u,T), \end{aligned} \tag{4}$$

subject to $x(0) = a,\ Q(x(T)) = 0$.

The cost of following a trajectory $x(t)$ of (1) produced by a generic input $u$ during $[t', t'+\sigma]$ will be denoted by:

$$C(x,t',\sigma) \triangleq \int_{t'}^{t'+\sigma} g(x,u)dt, \tag{5}$$

with $x(t), u(t)$ defined on an interval containing $[t', t'+\sigma]$.

## III. Computing Optimal Trajectory-Control Pairs

There are various methods which have been developed for solving Problem 1 numerically. Here, we will consider multiple shooting (MS) [1], which is widely used and could be considered as a workhorse of numerical optimization. We do this for the sake of concreteness, and because MS is straightforward to expose. However, the discussion that follows applies to more sophisticated numerical techniques as well (e.g., [7], [5]). In the following, we proceed to give a brief introduction to computational optimal control via MS.

### A. A Brief Review of Multiple Shooting

In principle, MS is a type of NLP method. Suppose, for now, that we are interested in minimizing the cost function $F(x) : \mathbb{R}^n \to \mathbb{R}$, subject to $m$ ($m \leq n$) equality constraints $c(x) = 0$, where $c(x) \in \mathbb{R}^m$. The necessary conditions for the point $(x^*, \lambda^*)$ to be an optimum are satisfied by the stationary points of the Lagrangian $L(x,\lambda) = F(x) + \lambda^T c(x)$:

$$\nabla_x L(x,\lambda) = \nabla_x F(x) + \nabla_x c(x)^T \lambda = 0 \tag{6}$$

$$\nabla_\lambda L(x,\lambda) = c(x) = 0, \tag{7}$$

where for $A : \mathbb{R}^n \to \mathbb{R}^n$, we write $\nabla_x A(x)_{ij} \triangleq \partial A_i(x)/\partial x_j$.

The equations (6)$\sim$(7) can be solved via Newton's method, after forming the Karush-Kuhn-Tucker (KKT) system:

$$H \cdot \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -\nabla_x F(x) - \nabla_x c(x)^T \lambda \\ -c(x) \end{bmatrix}, \tag{8}$$

where $\Delta x$ is the "search-direction",

$$H \triangleq \begin{bmatrix} H_L & \nabla_x c(x)^T \\ \nabla_x c(x) & 0 \end{bmatrix}, \tag{9}$$

and $H_L$ is the Hessian of the Lagrangian:

$$H_L = \nabla_x^2 F(x) + \sum_{i=1}^m \lambda_i \nabla_x^2 c_i. \tag{10}$$

Suppose now that we are interested in optimizing (2) subject to the dynamics (1). Doing so with MS (here we mainly refer to direct MS) involves transforming the optimal control problem into an instance of the function minimization

problem discussed in the previous paragraph. First, the trajectory of (1) is broken up into "shorter" pieces by partitioning the time domain $[0,T] = \cup_{i=1}^{N-1}[t_i, t_{i+1}),\ 0 = t_1 < \cdots < t_N = T$ into segments $[t_i, t_{i+1})$.

Multiple shooting uses NLP to find the optimal trajectory by defining the NLP variable

$$\nu = \{x_1, u_1, x_2, u_2, \ldots, x_N, u_N\}, \tag{11}$$

where $x_i \triangleq x(t_i),\ u_i \triangleq u(t_i),\ i = 1, ..., N$. Then, the stationary point $\nu^*$ obtained via NLP will contain samples of the optimal trajectory.

As the NLP variables are adjusted during MS, one must ensure that they satisfy the system dynamics (1), and that sequential trajectory segments obey a matching condition at their boundaries (continuity of trajectories). Specifically, $\nu$ must be such that if the input $u_i$ is applied to (1) starting from $x_i$ at $t_i$, the system will evolve to $x_{i+1}$ at $t_{i+1}$. This gives rise to the following constraints:

$$c(x) = \begin{bmatrix} x_2 - \bar{x}_2 \\ x_3 - \bar{x}_3 \\ \vdots \\ x_N - \bar{x}_N \\ \phi_0(x_1, t_0) \\ \phi_T(x_N, t_f) \end{bmatrix} = 0, \tag{12}$$

where the functions $\phi_0(x_1, t_0)$ and $\phi_T(x_N, t_f)$ represent any initial and terminal constraints imposed on the trajectory, and $\bar{x}_i$ are to be calculated by integrating the differential equation (1) from $t_i$ to $t_{i+1}$.

In practice, one often computes $\bar{x}_i$ only approximately. For example, Euler's method provides a first-order approximation to the integration of (1):

$$\bar{x}_{i+1} = x_i + hf(x_i, u_i), \tag{13}$$

with a time step of $h \triangleq t_{i+1} - t_i$, assuming uniform time sampling. The choice of approximation should be based on the following considerations: First, the controls have been sampled at the segment boundaries, thus we cannot compute the precise state evolution by integration on $[t_i, t_{i+1}]$ unless, for example, we assume piecewise constant inputs in (1). Second, using linear approximation reduces the computational burden, at the expense of accuracy; higher-order approximations produce trajectories which are closer to the true evolution of (1), but also give rise to more complicated equations in (6)$\sim$(12), and the resulting NLP problem requires more time and storage to solve.

One shortcoming of linear approximation is that the segment length $h$ must be kept "small", otherwise $\bar{x}_{i+1}$ will not be a good estimate of the true state of (1) as it evolves from $x_i$ to $x_{i+1}$ under $u_i$, and NLP will produce erroneous results. The approximation error is of $O(h^q)$, where $q \in \mathbb{N}$ is the order of the approximation method used [13]. If we fix $h$ and assume that $T$ is a multiple of $h$, then the required number of segments, $T/h$, may have to be very large to ensure that the estimates $\bar{x}_i$ are precise enough and that the associated NLP will convergence.

If the dimensions of the state $x$ and control $u$ are $n$ and $m$ respectively, and the number of trajectory segments is $N-1$, then the dimension of NLP variables for a multiple shooting method is $(n+m)N$, and the NLP has at least $n \cdot N$ constraints (the number of constraints could be larger if the final states are fixed). Thus, each iteration of NLP requires solving a set of $(m+2n)N$ algebraic equations (8). Then, we see that, as with other numerical methods, MS is $O(N^2)$ in the number of trajectory sampling points (in both storage and execution time), as well as in the number of state variables. This is the reason for the large storage requirements of MS, which limit the so-called "permitting size" of problems which can be solved on a digital computer [1]. In the next Section we propose a cooperative control strategy which solves optimal control problems in many smaller pieces, leading to lower storage requirements and allowing us to consider much larger problems, compared to what is possible with MS.

## IV. Optimal Control via Local Pursuit

Local pursuit is a bio-inspired cooperative strategy that solves Problem 1 using a group of cooperating "agents", where the term "agent" refers to a copy of (1):

$$\dot{x}_k = f(x_k, u_k),\ x_k(t) \in \mathbb{R}^n, u_k(t) \in \Omega \subset \mathbb{R}^m, \tag{14}$$

with $x_k(0) = x_I$, for $k = 0, 1, 2 \ldots$.

We give here a brief description of the algorithm along with the main result regarding its convergence. For further details, the reader may consult [9], [10], [19]. For a version of local pursuit that applies to problems with free final time see [20], [11].

Local pursuit assumes that there is an available initial (but sub-optimal) feasible control/trajectory pair $(u_{feas}(t), x_{feas}(t))$ for (14), which could have been obtained through exploration, or from a-priori knowledge. Agents are scheduled to leave the starting state $x_I$ sequentially, separated by an interval of $\Delta$ time units, i.e., if the first (initial) agent leaves at time $t_0$, the $k^{th}$ agent will leave at $t_k = t_0 + k\Delta, k = 1, 2, \ldots$. The first agent is required to follow $x_{feas}$, from $x_I$ to the target set $S_Q$. The next agent attempts to intercept its predecessor along optimal trajectories defined by (3), as long as the predecessor has not yet reached $S_Q$. If the predecessor is already in $S_Q$ then the follower moves along the optimal trajectory defined by (4) (see Fig. 1). Agents do not monitor their predecessors continuously, but instead update their trajectories with a sampling rate of $1/\delta$. The precise rules that govern agent evolution are:

(Sampled Local Pursuit [20], [11]): Let $x_0(t), t \in [0, T_0]$ be an initial trajectory satisfying (14) with $x_0(0) = x_I,\ Q(x_0(T_0)) = 0$. Choose $\Delta$ such that $0 < \Delta < T_0$.

1) DO $k = 1, 2, 3 \ldots$,
   Let $t_k = k\Delta$ be the starting time[1] of the $k^{th}$ agent, i.e., $x_k(t_k) = x_I$.

[1]For convenience, we assume that each agent can somehow wait at the initial state, $x_I$, until its time to begin pursuit. One way to ensure this is to require that $x_I$ is an equilibrium point of (1).

2) DO $i = 0, 1, 2, 3, \ldots$,
3) define $t_k^i = t_k^{i-1} + i\delta$, $t_k^0 = t_k$ where $0 < \delta < \Delta$
4) Let $u^*_{t_k^i}(\tau)$ be a control that achieves
$$\begin{cases} \eta(x_k(t_k^i), x_{k-1}(t_k^i)) & x_{k-1}(t_k^i) \notin S_Q, \\ \eta_Q(x_k(t_k^i)) & x_{k-1}(t_k^i) \in S_Q, \end{cases}$$
subject to (1), where $\tau \in$
$$\begin{cases} \left[t_k^i, t_k^i + \Gamma^*(x_k(t_k^i), x_{k-1}(t_k^i))\right] & x_{k-1}(t_k^i) \notin S_Q, \\ \left[t_k^i, t_k^i + \Gamma^*_Q(x_k(t_k^i), S_Q)\right] & x_{k-1}(t_k^i) \in S_Q. \end{cases}$$
where $\Gamma^*$ are the optimal times associated with Eq. (3) and (4).
5) Apply $u_k(t) = u^*_{t_k^i}(t)$ to the $k^{th}$ agent for $t \in [t_k^i, t_k^{i+1})$.
6) UNTIL the $k^{th}$ agent reaches $S_Q$.
7) UNTIL $C(x_k, t_k, t_k+T) - C(x_{k+1}, t_{k+1}, t_k+T) < \varepsilon$, for some fixed (small) $\varepsilon > 0$.

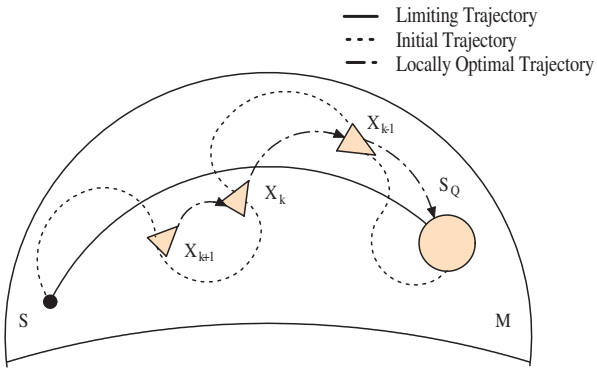


Fig. 1. Illustration of local pursuit. Each agent (except the first) attempts to "catch up" with its predecessor (as long as its predecessor has not reached $S_Q$), or to reach $S_Q$ optimally.

The two adjustable parameters in sampled local pursuit (SLP) are the *pursuit interval* $\Delta$, which defines how frequent agents depart from the initial state $x_I$ (consequently, how far agents are separated), and the *updating interval* $\delta$, which defines the frequency with which each agent updates its trajectory. Because $\delta$ is fixed, the total number of updates performed by each agent (Steps 2-5 of SLP) is at most $\lceil T/\delta \rceil$. We will refer to successive agents (e.g., the $k-1$ and $k$th agents) as "leader", and "follower". The times $t_k^i = t_k + i\delta, i = 0, 1, 2, 3 \ldots$ will be termed "updating times".

SLP is comprised of two types of policies – "catching up" and "free running" – depending on whether the leader has reached the final constraint set $S_Q$ or not. The former lets agents "learn" from their leaders, meanwhile the latter enables them to find the optimal final state.

The following theorem is a rephrasing of the main result from [9] and concerns the limiting behavior of SLP.

*Theorem 1 ([9]): Suppose that a group of agents (14) evolve under SLP, and that at every updating time $t_k^i$, the locally optimal trajectories from follower to leader exist and are unique. If the updating interval $\delta$ and pursuit interval $\Delta$ satisfy $0 < \delta < \Delta$, then the agents' trajectories converge to a unique, locally optimal trajectory. Convergence is monotonic in cost, and the limiting trajectory is smooth if the locally optimal trajectories calculated by the follower(s) at every updating time are smooth.*

Proof: See [9], [10]. For a similar result that applies to free final time and partially-constrained final state problems, see [11], [20].

SLP was initially conceived with remote exploration tasks in mind. In that setting, SLP could be useful for a group of autonomous vehicles, because of its reduced information and sensing range requirements. Indeed, as Step 4 of the algorithm indicates, agents need only communicate with their nearby neighbors during pursuit. Computing the optimal trajectory to one's leader requires knowledge of the geometry in a small patch surrounding the agent, as opposed to a map of the entire area. In fact, agents may not even know the coordinates of the target state (or set), having only a set of instructions for reaching it (the initial feasible control). See [9] for a detailed discussion of the information requirements of SLP vs. single-agent approaches.

## V. Local Pursuit as a Computational Tool

For our purposes, the central feature of SLP is that it solves the optimal control problem in pieces whose duration is determined by the agent separation $\Delta$, instead of computing on the entire $[0, T]$ all at once. Computationally, SLP gives us a way to relax the trade-off between a small time step $h$ and a manageable number of segments in MS. What we have in mind is a simulated sequence of agents, which evolve from the initial state $x_I$ to the target set $S_Q$. Each agent proceeds by calculating (via MS or other numerical technique) the optimal trajectory from its current state to that of its predecessor, giving rise to a series of smaller NLP problems, which it must solve along the way to $S_Q$. Although there will be more of these NLP problems to solve, their lower dimension will make it possible to handle larger optimization problems overall. Convergence to a local optimum is ensured by Theorem 1.

We will refer to our hybrid method as "Pursuit-based multiple shooting" (PBMS). For a given trajectory time step $h$, PBMS will be able to handle problems with longer time horizons, compared to MS; conversely, for a fixed time horizon, PBMS will allow us to sample trajectories more densely with a fixed amount of computing resources. We note that for the purposes of SLP, MS is essentially treated as a "subroutine", to be called by each agent in Step 4 of the algorithm; it is possible to use other numerical methods instead. SLP's advantage in handling large problems would still apply, as long as the numerical method of choice has a computational or storage complexity which is worse than linear in the number of trajectory segments.

### A. Decreasing the size of the NLP problems during computation

For simplicity, we fix $h = T/(N-1)$ and select the pursuit interval $\Delta = (N_\Delta - 1)h$, $N_\Delta \in \mathbb{N}$, where $N_\Delta << N$ is the number of trajectory segments within $\Delta$ time units. We will also take the updating interval to be an integer multiple

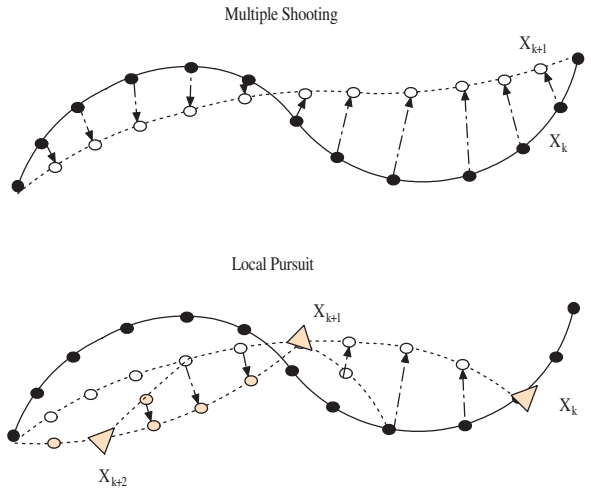


Fig. 2. Local pursuit decreases the effective NLP problem size. The number of variables updated in MS is $N = 13$; the number of variables updated by each agent during local pursuit is $N_\Delta = 5$.

of segment size, i.e., $\delta = N_\delta h, \quad N_\delta \in \mathbb{N}$, so that the agents are always updating their trajectories at the times $t_i$ at which we chose to sample the trajectory of (1). Now, at each updating time $t = i\delta$, each agent solves a NLP problem with $N_\Delta$ trajectory sampling points, over a time span of $(N_\Delta - 1)h$ instead of $(N-1)h$, as illustrated in Fig. 2. Table I shows the dimensions of the NLP vector, $\nu$, the constraints $c(x)$, and the number of entries of $H$ in (9), for MS versus PBMS. Solving the associated KKT system is

TABLE I

COMPARING THE DIMENSIONS OF THE NLP PROBLEM VARIABLES WHEN USING MS VS. PBMS.

| | **MS** | **PBMS** |
|---|---|---|
| $\dim(\nu)$ | $(n+m)N$ | $(m+n)N_\Delta$ |
| $\dim(c(x))$ | $nN$ | $nN_\Delta$ |
| $\dim(H)$ | $((m+2n)N)^2$ | $((m+2n)N_\Delta)^2$ |

$O(N^2)$, for each iteration of MS. Under PBMS, each agent solves $(N - N_\Delta)/N_\delta$ MS problems of $O(N_\Delta^2)$ each. Thus, with $N_\Delta << N$ the computational burden for each agent can be decreased significantly.

The total running time for PBMS may be greater than that for MS, because local pursuit requires multiple agents to converge to the optimum. The computing time per agent could be minimized by selecting $N_\Delta$ small, and taking $N_\delta$ close to $N_\Delta$. On the other hand, we expect (and have observed in numerical experiments) that reducing $N_\delta$ leads to convergence with a smaller number of agents. That is because each agent has more opportunities to adjust its trajectory, resulting in a greater reduction in cost over its predecessor, compared to when the same agent makes a small number of updates. In general, decreasing $N_\Delta$ tends to slow the convergence of PBMS. However, the added running time comes with the benefit of handling problems with larger state vectors and longer time horizons (under PBMS, the NLP problem is scaled down by a factor of $(N_\Delta/N)$). At the same time, if $N_\Delta$ is decreased and the available storage is fixed, one can also afford to decrease the segment size $h$. Doing so has the effect of improving the state estimates $\bar{x}_i$ used to formulate the NLP, without making the problem ill-conditioned. This situation will be illustrated in the next Section.

### *B. Reducing numerical error when $H$ is ill-conditioned*

Another important consideration in numerical optimal control, besides maximum problem size, is the computational error introduced by the finite precision of digital computers and by algorithmic accuracy. It is possible that the error is too large to obtain useful results, e.g., solving a linear system with large condition number can result in unacceptable errors and prevent convergence. In such settings, correction algorithms, such as Tikhonov regularization, can be applied [13], [15]; they are, however, time and storage consuming, and do not always succeed.

For the optimal control problem of interest, consider, for example, using Gaussian elimination to solve (8) with limited digital precision. Every step of the elimination algorithm introduces some truncation error, so that the total accumulated error when solving a large linear system will generally be much larger than that associated with solving one of lower dimension, because the number of steps required by the algorithm is proportional to the system size. Furthermore, if in (9) $H$ is ill-conditioned, then the numerical solution of (8) introduces small errors which accumulate towards the final segment, $N-1$. If the problem's time horizon is long, the accumulated error may lead to erroneous results, or prevent MS from converging altogether. PBMS can help reduce these numerical errors because the algorithm's simulated agents solve MS problems with a shorter time horizon, compared to that of the original problem. This implies that the dimension of (8) for each agent is reduced and there will be cases in which PBMS will succeed where MS fails to converge. One such example is presented in the next Section.

## VI. EXAMPLE: AN ORBIT TRANSFER PROBLEM

Consider an idealized spacecraft which must be transfered from one stable orbit[2] to another, within some fixed time $T$, using a low-power thruster. For simplicity, we only consider the effect of the Earth's gravity, and restrict the problem to a plane, as illustrated in Fig. 3.

The dynamics of this system are

$$\begin{aligned} \ddot{r} &= \dot{\theta}^2 r - \frac{u_E}{r^2} + \frac{P u_1}{mg} \\ \ddot{\theta} &= -\frac{2\dot{\theta}\dot{r}}{r} + \frac{P u_2}{mgr} \\ \dot{\mu} &= -\frac{P(u_1^2 + u_2^2)}{g^2} \\ u_1 &= I(t)\sin(\varphi(t)) \\ u_2 &= I(t)\cos(\varphi(t)) \end{aligned} \tag{15}$$

[2]The term "stable orbit" means that the spacecraft will remain on this orbit if no external force (other than gravity) is applied.

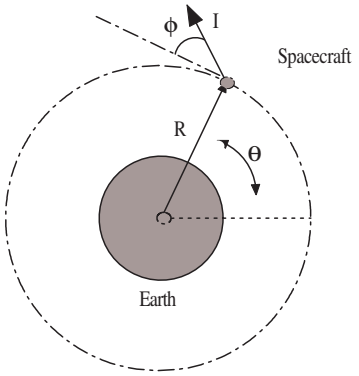


Fig. 3. A planar spacecraft orbiting the Earth.

where $r$ is the spacecraft altitude measured from the center of Earth, $\theta$ is its longitude, $\mu$ is the mass of the spacecraft, $u_E = 3.98 \times 10^{14}\ m^3/s^2$ is the gravitational parameter of Earth, $P = 10^7\ kg \cdot m/s^2$ is a constant concerning engine power, and $g = 9.81\ m/s^2$ is the acceleration of gravity at sea level [21]. The control inputs, $u_1$ and $u_2$, are functions of the thrust force $I(t)$ and steering angle $\varphi$, the latter being measured with respect to the tangent of the local orbit.

We would like to minimize the fuel consumed (equivalently, maximize the final mass $\mu(T)$):

$$J = \int_0^T (u_1(t)^2 + u_2(t)^2)dt,$$

while steering the system (15) from an initial stable orbit $r(0) = 9.6 \times 10^6\ m$, $\dot{r}(0) = 0$, $\dot{\theta}(0) = \sqrt{u_E/r(0)^3}$, $\mu(0) = 525\ kg$, to a final one, $r(T) = R_T, \dot{r}(T) = 0, \dot{\theta}(T) = \sqrt{u_E/R_T^3}$, for some fixed $T$. For simplicity, we ignored any bounds on the magnitude of the thrust force $I(t)$.

### A. Applying MS vs. PBMS

To explore the performance of PBMS versus that of simple MS, we first produced a trajectory which steered the satellite suboptimally from its initial state to a desired altitude. For example, the inputs

$$\begin{aligned} I(t) &= \begin{cases} 60(1.5 + \sin(\frac{6\pi t}{T})) \cdot 10^{-4}, & 0 \le t < \frac{T}{2} \\ 75 \cdot 10^{-4}, & \frac{T}{2} \le t \le T \end{cases} \\ \phi(t) &= \begin{cases} (1 + \sin(2\pi t/T))\pi, & 0 \le t < \frac{T}{2} \\ 0.4417\pi, & \frac{T}{2} \le t \le T \end{cases} \end{aligned} \quad (16)$$

brought the spacecraft to an altitude of $r(T) = R_T \stackrel{\triangle}{=} 12.7778 \times 10^6\ m$, with $\dot{r}(T) = 96.67\ m/s$, $\dot{\theta}(T) = 4.4 \times 10^{-4}\ rad/s$, in $T = 300\ min$, using $53.5\ kg$ of fuel. Our goal was to calculate the optimum-fuel transfer to a stable orbit at $r(T)$ starting from the same initial conditions.

When solving the problem by MS, we used Euler's method to estimate the evolution of (1) and imposed the following constraints:

$$\begin{aligned} 0 &= r_{j+1} - \bar{r}_j = r_{j+1} - (r_j + h_j \dot{r}_j) \\ 0 &= \dot{r}_{j+1} - \bar{\dot{r}}_j \\ &= \dot{r}_{j+1} - \left( \dot{r}_j + h_j \left( \frac{\dot{\theta}_j^2}{r_j} - \frac{u_E}{r_j^2} + \frac{P u_{1j}}{m_j g} \right) \right) \\ 0 &= \dot{\theta}_{j+1} - \bar{\dot{\theta}}_j \\ &= \dot{\theta}_{j+1} - \left( \dot{\theta}_j + h_j \left( -\frac{2\dot{\theta}_j \dot{r}_j}{r_j} + \frac{P u_{2j}}{m_j g r_j} \right) \right) \\ 0 &= \mu_{j+1} - \bar{\mu}_j \\ &= \mu_{j+1} - \left( \mu_j - h_j \frac{P(u_1^2 + u_2^2)}{g^2} \right) \end{aligned} \quad (17)$$

for $j = 1, 2, 3 \ldots, N-1$, and

$$\begin{aligned} r_1 - R_0 &= \dot{r}_1 - \dot{R}_0 = 0 \\ \dot{\theta}_1 - \dot{\theta}_0 &= m_1 - M_0 = 0 \\ r_T - R_T &= \dot{r}_T - \dot{R}_T = 0 \\ \dot{\theta}_T - \dot{\theta}_T &= 0, \end{aligned} \quad (18)$$

where $r_j = r((j-1)h), \dot{r}_j = \dot{r}((j-1)h), \ldots$, and $h = T/(N-1)$ was the time step. The dimension of the constraint vector $c(x)$ was $(4 \times (N-1) + 7)$. The NLP variable

$$\begin{aligned} \nu &= \{r_1, \ldots, r_N, \dot{r}_1, \ldots, \dot{r}_N, \dot{\theta}_1, \ldots, \dot{\theta}_N, m_1, \ldots, m_N \\ &\quad , u_{11}, \ldots, u_{1N-1}, u_{21}, \ldots, u_{2N-1}\} \end{aligned} \quad (19)$$

was of dimension $(6 \times N - 2)$.

When applying local pursuit, $N$ should be replaced by $N_\Delta$ in the above equations. In that case, the dimension of the constraint vector and the NLP variable are $(4 \times (N_\Delta - 1) + 7)$ and $(6 \times N_\Delta - 2)$, respectively. The constraint set $S_Q$ was the set of states $\{(r, \dot{r}, \theta, \dot{\theta}, \mu) : r = R_T, \dot{r} = 0, \dot{\theta} = \sqrt{u_E/R_T}\}$, i.e., a stable orbit with the final longitude unconstrained.

The results that follow were obtained by implementing MS and PBMS in MATLAB 7.0, running on a $3GHz$ desktop PC with $1Gb$ of RAM. When simulating the equations of motion, we scaled length and mass by a factor of $10^{-6}$ and $10^{-3}$, respectively, in order to reduce truncation errors. All operations were carried out with 64-bit precision.

### B. Results

We solved the optimal orbit transfer problem both by PBMS and standard MS. The criteria for convergence were $\|\mu(T)_{i+1} - \mu(T)_i\| \le 10^{-8}$ (to guarantee little improvement with additional iterations) and $\|c(x)\| \le 10^{-15} \times dim(c(x))$ (to enforce continuity of the trajectory).

*1) Well-conditioned case:* With $N = 101, 201, 301$, both MS and PBMS converged to trajectories with virtually identical costs. The KKT system (8) was well conditioned, resulting in fast convergence for MS. Table II summarizes the performance of both methods for the case $N = 301$. The "iterations" row lists the iterations required for convergence in MS, and the number of agents needed for local pursuit to converge, respectively. The quantity cond$(H)$ refers to the condition number of matrix $H$. The minimum fuel

expenditure was approximately $1.021kg$; the two methods agreed on that quantity to within $4gr$. With PBMS, solving the problem in small pieces meant lower condition numbers for the $H$ in (8) than those of MS. The initial suboptimal trajectory together with the final trajectories obtained by both methods are shown in Fig. 4.

TABLE II
COMPARISON BETWEEN MS AND PBMS – WELL-CONDITIONED CASE

| N=301 | **Multiple Shooting** | **PBMS** ($N_\Delta = 60$, $N_\delta = 32$) | **PBMS** ($N_\Delta = 120$, $N_\delta = 64$) |
|---|---|---|---|
| CPU Time (s) | 938 | 47602 | 16704 |
| Iterations | 13 | 3344 | 261 |
| $\mu(T)$ $(kg)$ | 523.979819 | 523.975470 | 523.979594 |
| $\|c(x)\|$ | $1.84 \cdot 10^{-14}$ | $1.68 \cdot 10^{-14}$ | $1.78 \cdot 10^{-14}$ |
| Avg(cond($H$)) | $1.74 \cdot 10^{6}$ | $1.24 \cdot 10^{5}$ | $1.25 \cdot 10^{5}$ |
| Memory usage (Mb) | 139 | 7 | 23 |

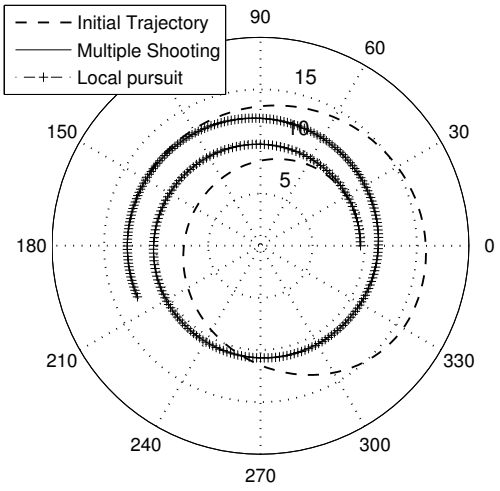


Fig. 4. Orbit transfer - well-conditioned case with $N = 301, N_\Delta = 30, N_\delta = 16$. The origin $(0, 0)$ corresponds to the Earth's center. Radial distance in the plot is in units of $10^3 km$. At this scale, the trajectories produced by MS and PBMS appear identical.

For the case $N = 101$, MS converged in 34 sec (9 iterations), compared to 1247 sec for PBMS with $N_\Delta = 32$, $N_\delta = 16$ (746 agents). For $N = 201$, the convergence times were 320 sec for MS (12 iterations), and 41193 sec for PBMS (8955 agents) with the same choice of $N_\Delta$, $N_\delta$. In every case, both methods were successful and produced virtually identical trajectories. As expected, PBMS was slower than MS but also used a fraction of the memory required by MS. Increasing $N_\Delta$ resulted in increased need for storage and decreased running time (for $N_\Delta = N$, PBMS reduces to MS).

*2) Ill-conditioned case:* Using a larger input thrust

$$I(t) = \begin{cases} 90(1.5 + \sin(\frac{6\pi t}{T})) \cdot 10^{-4}, & 0 \le t < \frac{T}{2} \\ 75 \cdot 10^{-4}, & \frac{T}{2} \le t \le T, \end{cases}$$
$$\phi(t) = \begin{cases} \pi/2, & 0 \le t < \frac{T}{2} \\ 0.4417\pi, & \frac{T}{2} \le t \le T, \end{cases} \quad (20)$$

we produced a new trajectory which steered the satellite from its initial state to an altitude of $r(T) = 23.642 \times 10^6\ m$ (roughly twice as high as in the previous case), in $T = 300\ min$, using $103\ kg$ of fuel.

In this case, MS was unable to optimize the transition to a stable orbit at $r(T)$, for $N$ ranging from 101 to 701 in increments of 100 steps. In each case, the method stagnated, with large condition numbers (typically $\sim 10^{10}$), and was unable to reduce the norm of the constraint vector $c(x)$ (typical values were in the order of 1 to $1/100$, several orders of magnitude above tolerance). The larger thrust magnitude $I(t)$ in (20) meant that the spacecraft traveled further during each time step $h$, thus more segments were needed to maintain the same spatial sampling density as in the case of the lower-thrust input (16). Although we cannot say with certainty, we speculate based on numerical experiments that for small $N$ the failure of MS was in part due to the large errors introduced by Euler's approximation to the dynamics (1), while for large $N$, the problem was mainly the accumulation of numerical (e.g., truncation) errors when solving the KKT equations. In comparison, PBMS converged every time. For example, with $N = 501$, with $N_\Delta = 60$, $N_\delta = 32$, the optimal trajectory (shown in Fig. 5) was computed in 127620 sec, using 2892 agents. The optimal final mass was $\mu(T) = 516.327792kg$ ($8.672kg$ of fuel used).

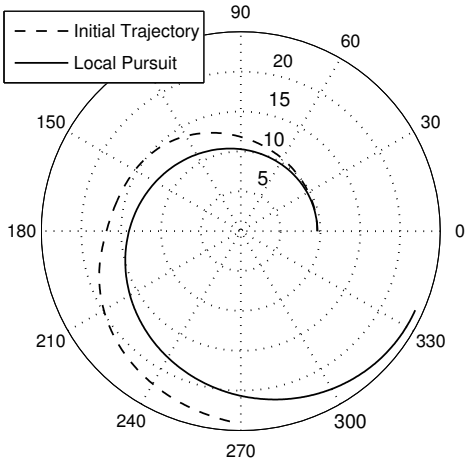


Fig. 5. Spacecraft trajectory obtained by PBMS with $N = 501, N_\Delta = 30, N_\delta = 16$; $Avg(cond(H)) = 3.349 \times 10^5$, $\|c(x)\| = 3.7 \times 10^{-14}$. The optimal fuel expenditure was $8.672\ kg$. The NLP associated with MS was ill-conditioned and that method did not converge to a meaningful solution.

*3) Large number of segments:* As we have already mentioned, the numerical integration of the equations of motion (1) during optimization introduces errors which may accumulate towards the final state. That error is $O(h^q)$ if (1) is approximated up to order $q$ (in our case $q = 1$). Thus, assuming the approximation method is fixed, one must increase the number of trajectory segments in order to obtain a sufficiently accurate solution.

To gauge the "true" evolution of the spacecraft dynamics that would take place if the (piecewise constant) optimal inputs produced by MS or PBMS were to be applied, we

simulated the trajectory of (15) using a fourth order Runge-Kutta solver (MATLAB's `ode45` command) under optimal control. We then compared this "reference" trajectory to those produced by MS and PBMS. In each case tested, PBMS produced trajectories which were closer to the reference (both the $L_2$ and $L_\infty$ sense) than those computed via MS.

For the orbit transfer to $r(T) = 12.778 \times 10^6 m$ (Sec. VI-B.1) with $N = 301$ sampling points, the reference trajectory reached a final altitude which was $7.9km$ higher than desired, with the spacecraft approximately $23km$ away from the reference final position. For the higher-altitude orbit transfer to $r(T) = 23.642 \times 10^6 m$ (Sec. VI-B.2) with $N = 501$, the spacecraft's final position was approximately $557.3km$ higher than planned, with a radial velocity of $66.67m/s$ instead of zero. These differences — though small ($\sim 2\%$) in relative terms — are clearly significant and can be reduced by increasing $N$.

On our hardware platform, MS could not be executed with $N > 1000$. On the other hand, PBMS's lower memory requirements meant that the optimization problem could be solved for sampling densities well above that number. For the orbit transfer problem of Sec. VI-B.2 with $N = 2001$, PBMS converged in 304 hours, using 12402 agents ($N_\Delta = 100$, $N_\delta = 81$) and approximately 18 Mb of memory (MS would have required approximately 6 Gb). The final spacecraft mass was $\mu(T) = 516.183951kg$, with $\|c(x)\| = 2.4 \times 10^{-14}$ and $Avg(cond(H)) = 4.481 \times 10^5$. The fuel expenditure was $144gr$ less than that obtained with $N = 501$ samples. As a result of the higher sampling density, the difference between the desired altitude ($12.778 \times 10^3 km$) and the one achieved in the reference trajectory (computed via Runge-Kutta) was reduced to $10.5km$; the spacecraft's final radial velocity was approximately $8.1\ m/s$.

## VII. Conclusions and ongoing work

This paper explored the use of a bio-inspired cooperative strategy known as *local pursuit* for solving a class of numerical optimal control problems. We discussed a pursuit algorithm which is suited to problems with fixed final time and partially-constrained final states. The algorithm mimics the foraging behavior of ant colonies and allows a collective to "discover" optimal controls, starting from an initial suboptimal solution and using only local, pair-wise interactions.

We proposed combining local pursuit with multiple shooting (MS) as a way to overcome some of the computational and storage limitations of the latter method. The new method, termed pursuit-based multiple shooting (PBMS) involves a kind of "breaking down" of the original problem to smaller segments, each to be optimized (using MS) by a group of simulated agents. PBMS convergences more slowly than pure MS but can achieve very low memory requirements, by taking advantage of the cooperation among agents. Furthermore, for a fixed time horizon, PBMS gives us the option to sample the solution more densely, resulting in increased computational accuracy and better performance in cases where MS is ill-conditioned.

## References


[1] J.T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control and Dynamics*, 21(2):193–207, Mar.–Apr. 1998.

[2] A. M. Bruckstein. Why the ant trails look so straight and nice. *The Mathematical Intelligencer*, 15(2):59–62, 1993.

[3] A. M. Bruckstein, C. L. Mallows, and I. A. Wagner. Probabilistic pursuits on the grid. *The American Mathematical Monthly*, 104(4):323–343, Apr. 1997.

[4] S. Camazine, J.-L. Deneubourg, N. R. Franks, J. Sneyd, G. Theraulaz, and E. Bonabeau. *Self-Organization in Biological Systems*. Princeton University Press, 2001.

[5] Y. Chen and A. L. Schwartz. Riots95 - a Matlab toolbox for solving general optimal control problems and its applications to chemical processes. In R. Luus, editor, *Recent developments in optimization and optimal control in chemical engineering*, pages 229–252. Transworld Research Pub., 2002.

[6] M. Dorigo, V. Maniezzo, and A. Colorni. Ant systems: Optimization by a colony of cooperating agents. *IEEE Trans. on Systems, Man and Cybernetics, Part B*, 26(1):29–41, 1996.

[7] P. E. Gill, W. Murray, M. A. Saunderss, and M. H. Wright. User's guide for NPSOL (v.5.0): A FORTRAN package for nonlinear programming. Technical Report 98-2, Dept. of Mathematics, U.C. San Diego, 1998.

[8] D.M. Gordon. *Ants at work*. The Free Press, New York, 1999.

[9] D. Hristu-Varsakelis and C. Shao. Biologically-inspired optimal control: Learning from social insects. *Int'l Journal of Control*, 77(18):1545–66, Dec. 2004.

[10] D. Hristu-Varsakelis and C. Shao. Corrections to: Biologically-inspired optimal control: Learning from social insects. *Int'l Journal of Control*, 78(2):157, Jan. 2005.

[11] D. Hristu-Varsakelis and C. Shao. A bio-inspired pursuit strategy for optimal control with partially-constrained final state. *Automatica*, (to appear), 2007.

[12] A. Jadbabaie, J. Lin, and A.S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Trans. Automatic Control*, 48(6), Jun. 2003.

[13] R. Kress. *Numerical Analysis*. Springer-Verlag New York Inc, New York, 1998.

[14] R. Kurazume and S. Hirose. Study on cooperative positioning system: optimum moving strategies for cps-iii. In *Proc. 1998 IEEE Int'l Conf. in Robotics and Automation*, volume 4, pages 2896–2903, Leuven, Belgium, May 1998.

[15] A. Neumaier. Solving ill-conditioned and singular linear systems: a tutorial on regularization. *SIAM Review*, 40(3):636–666, Sep. 1998.

[16] J.K. Parrish and W.M. Hammer. *Animal groups in three dimensions*. Cambridge, U.K., 1997.

[17] K. Passino, M. Polycarpou, D. Jacques, M. Pachter, Y. Liu, Y. Yang, M. Flint, and M. Baum. Cooperative control for autonomous air vehicles. In R. Murphey and P.M. Pardalos, editors, *Cooperative control and optimization*, pages 233–272. Kluwer Academic Publishers, 2002.

[18] S.I. Roumeliotis and G.A. Bekey. Distributed multi-robot localization. *IEEE Trans. Robotics and Automation*, 18(5):781–795, 2002.

[19] C. Shao and D. Hristu-Varsakelis. Bio-inspired cooperative optimal control with partially-constrained final state. Technical Report 2005-76, Institute for Systems Research, University of Maryland, College Park, MD., 2005. Available online at: http://techreports.isr.umd.edu/reports/2005/TR_2005-76.pdf.

[20] C. Shao and D. Hristu-Varsakelis. Optimal control through biologically inspired pursuit. In *Proc. of the 16th IFAC World Congress*, Prague, Jul. 2005.

[21] S.R. Vadali and R. Nah. Fuel-optimal planar earth-mars trajectories using low-thrust exhaust-modulated propulsion. *Journal of Guidance, Control, and Dynamics*, 23(3):476–482, May–Jun. 2000.

[22] I. A. Wagner, M. Lindenbaum, and A. M. Bruckstein. Distributed covering by ant-robots using evaporating traces. *IEEE Trans. Robotics and Automation*, 15(5):918–933, 1999.