Local Pursuit as a Bio-Inspired Computational Optimal Control Tool

by Cheng Shao, Dimitrios Hristu-Varsakelis

CDCSS TR 2005-1
(ISR TR 2005-85)

Local Pursuit as a Bio-Inspired Computational Optimal Control Tool [⋆]

Cheng Shao [a], Dimitrios Hristu-Varsakelis [b,∗]

[a] *Department of Mechanical Engineering and Institute for Systems Research University of Maryland, College Park, MD 20742 USA*

[b] *Department of Applied Informatics, University of Macedonia, Thessaloniki 54006, Greece*

Abstract

This paper explores the use of a bio-inspired control algorithm, termed “local pursuit”, as a numerical tool for computing optimal control-trajectory pairs in settings where analytical solutions are difficult to obtain. Inspired by the foraging activities of ant colonies, local pursuit has been the focus of recent work on cooperative optimization. It allows a group of agents to solve a broad class of optimal control problems (including fixed final time, partially-constrained final state problems) and affords certain benefits with respect to the amount of information (description of the environment, coordinate systems, etc.) required to solve the problem. Here, we present a numerical optimization method that combines local pursuit with the well-known technique of multiple shooting, and compare the computational efficiency and capabilities of the two approaches. The proposed method method can overcome some important limitations of multiple shooting by solving an optimal control problem “in small pieces”. Specifically, the use of local pursuit increases the size of the problem that can be handled under a fixed set of computational resources. Furthermore, local pursuit can be effective in some situations where multiple shooting leads to an ill-conditioned nonlinear programming problem. The trade-off is an increase in computation time. We compare our pursuit-based method with direct multiple shooting using an example that involves optimal orbit transfer of a simple satellite.

Key words: Co-operative control, Optimization, Agents, Group work, Trajectories, Numerical methods, Nonlinear programming

1 Introduction

Over the past decade, researchers have been increasingly looking to cooperative strategies as a means for addressing systems problems which are difficult to solve by single systems [18,5]. The push for cooperation is partly due to the maturation of technology that makes it possible to develop meaningful cooperation among, say, groups of wheeled or flying vehicles, and partly because of the obvious success of biological cooperative systems which have apparently evolved to “perform as more than the sum of their parts” [6,13]. Notable examples include worker honey bees, which share information by “dancing” and distribute themselves among different 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[4,2,9] as well as other work on cooperative control strategies, from distributed collective covering and searching [18,14], to estimating by groups [15,11], and biologically-motivated optimization [5,3,8].

In particular, [2] presented a decentralized organizing principle, inspired by the foraging behaviors of ant colonies, that allowed a group to optimize an initial path between two locations on $\mathbb{R}^2$, by having a sequence of group members travel from one location to the other while each member points its velocity vector towards its predecessor. This so-called “local pursuit” rule was generalized to non-Euclidean environments [7], and

⋆ This work was supported by the National Science Foundation under Grant No. EIA0088081 and by ARO ODDR&E MURI01 Grant No. DAAD19-01-1-0465, (Center for Communicating Networked Control Systems, through Boston University).

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

Email addresses: cshao@glue.umd.edu (Cheng Shao), dcv@uom.gr (Dimitrios Hristu-Varsakelis).

later to a pair of pursuit-based algorithms that applied to a much broader class of optimal control problems and to systems with non-trivial dynamics[8,16]. These algorithms require each agent in the collective to solve a series of locally optimal control problems within small neighborhoods that are defined by its current state and the state of its predecessor. Doing so defines a control/trajectory sequence that, under certain conditions, converges to the optimum.

Local pursuit was originally conceived as a means of solving optimal control problems in settings where mapping, communication and sensing capabilities were severely limited. The algorithm manages to avoid the need for global information by breaking up the problem into many pieces, each to be optimized by leader-follower agent pairs. Its key feature is a reduction in the range (measured by time, distance or other metric) over which computations must take place, by paying a price in terms of the number of agents that are necessary to carry out the algorithm. For example, in order for the collective to solve an optimal control problem, it is not necessary to have available a map of the environment, an agreed-upon coordinate system, or even the coordinates of the target state; only an initial feasible control is required.

Up to now, discussions of local pursuit have focused on broadening the domain of applicability of the algorithm and on the limiting properties of the agents' trajectories. However, its limited information requirements make it a potentially useful tool in numerical trajectory optimization, where there are often similar trade-offs to be made. In particular, given a control system which must be steered between two states in a fixed amount of time, one typically proceeds by "sampling" the system trajectory at pre-determined times, and solving a non-linear programming (NLP) problem to determine the best placement of the samples in the statespace, subject to constraints given by the equations of motion and continuity, among others. In that setting, there is a balance that must be struck between the number of trajectory samples and the size of the time intervals that separate them. On one hand, we would like to keep the number of segments small, so that the associated NLP problem has reasonable storage requirements. If the number of segments is too small however, then 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. If, on the other hand, the trajectory is sampled densely, then there may not be adequate memory to solve the associated nonlinear programming (NLP) problem, which at the same time is prone to a more significant accumulation of numerical errors.

The purpose of this paper is to explore the application of local pursuit as a computational tool in order to address the shortcomings outlined above. We introduce a numerical optimization method, titled "pursuit-based multiple shooting" (PBMS), that combines a recently reported local pursuit strategy with established numerical methods. In particular, we propose a modified version of the well-known multiple-shooting (MS) method that uses local pursuit to overcome some of the limitations of the original MS technique. The PBMS method that is proposed here involves simulating group of agents which cooperate to solve (numerically) an optimal control problem. Agents use standard MS to optimize their short-term behavior.

A comparison of the computational complexity and size of the resulting problems formulated via PBMS versus standard MS, reveals that cooperation among group members can overcome some important limitations of MS. Our pursuit-based formulation allows one to always operate in the domain of manageable-sized problems, while still being able to treat problems with very large numbers of segments. As a result, it enlarges the maximum size of problems that can be handled given a fixed amount of storage space. In addition, it can reduce computational errors due to ill-conditioning of the non-linear programming problem that must be solved when MS is used. The trade-off is that, for well-conditioned problems, demands more running time than MS in order to converge.

The remainder of this paper is organized as follows: Section 2 defines a local pursuit algorithm which is applicable to optimal control problems with fixed final time and partially-constrained final states, and gives the main results concerning the behavior of a collective under that algorithm. Section 3 begins with a brief introduction to numerical optimization and goes on to discuss the potential advantages of modifying multiple shooting (one of the best-known and widely used methods for numerical optimal control) to include local pursuit. An illustrative example is presented in Section 4 where we compare the performance of multiple shooting to that of puruit-based multiple shooting in a satellite orbit transfer problem.

2 A Bio-inspired Algorithm for Optimal Control

Local pursuit begins with a group of cooperating "agents", where the term "agent" refers to a copy of a dynamical system:

$$\dot{x}_k = f(x_k, u_k),\ x_k(t) \in \mathbb{R}^n, u_k(t) \in \Omega \subset \mathbb{R}^m \qquad (1)$$

for $k = 0, 1, 2 \ldots$. The problem of interest is as follows:

Problem 1 *Find a trajectory $x^*(t)$, and a final state $x^*(T)$, (T fixed) that minimize*

$$J(x, \dot{x}, t_0, T) = \int_{t_0}^{t_0+T} g(x, \dot{x})dt + G(x(t_0 + T)) \qquad (2)$$

subject to the constraints $x(t_0) = x_0$, and $Q(x(t_0+T)) = 0$.

Here it is assumed that $g(x(t), \dot{x}(t)) \geq 0$, $G(x(t_0+T)) \geq 0$ and that $Q(\cdot)$ is an algebraic function of the state.

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

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

For any pair of fixed states $a, b \in \mathbb{D} \subset \mathbb{R}^n$, suppose the optimal trajectory from a to b with fixed final time (minimizing J with respect to x only) is denoted by $x^*(t)$. Then the cost of following $x^*(\cdot)$ is denoted by:

$$\eta(a, b, t_0, T) \triangleq \int_{t_0}^{t_0+T} g(x^*(t), \dot{x}^*(t))dt \tag{3}$$

subject to $x(t_0) = a,\ x(t_0 + T) = b$.

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

$$\begin{aligned}\eta_Q(a, t_0, T) &\triangleq \int_{t_0}^{t_0+T} g(x^*, \dot{x}^*)dt + G(x^*(t_0 + T)) \\ &= \min_x J(x, \dot{x}, t_0, T) \end{aligned}\tag{4}$$

subject to $x(t_0) = a, Q(x(t_0 + T)) = 0$.

2.1 Algorithm

In previous works [8,16] we have proposed two classes of algorithms, one for problems with fixed boundary conditions (final time and final states), and another for problems with partially-constrained boundary conditions. We will briefly present a "sampled" version local pursuit for trajectory optimization problems with fixed final time and partially-constrained final states (based on algorithms from [8,16]).

We assume that there is an available initial (but sub-optimal) feasible control/trajectory pair $(u_{feas}(t), x_{feas}(t))$ for (1), which could have been obtained through exploration or from a-priori knowledge. Agents are scheduled to leave the starting state x_0 sequentially, separated by a pursuit interval of Δ time units, i.e., if the first (initial) agent leaves at time t_0, the k^{th} agent will leave at $t_k = t_0 + k\Delta, k = 0, 1, 2, \ldots$. The first agent is required to follow x_{feas} from x_0 to S_Q. The next agent attempts to intercept its predecessor along an optimal trajectories defined by (3), as long as the predecessor has not yet reached the target set S_Q. If the predecessor is already in S_Q then the follower moves along the optimal trajectory defined by (4) (see Fig. 1 for illustration). Agents do not monitor their predecessors continuously, but instead updates their trajectories with the sampling rate of $1/\delta$. The precise rules that govern the movement of each agent are:

Algorithm 1 *(Sampled Local Pursuit): Identify the starting state x_0 on $\mathbb{D}$ and the constraint set S_Q. Let $x_0(t)$ ($t \in [0, T_0]$) be an initial trajectory satisfying (1) with $x_0(0) = x_0,\ Q(x_0(T_0)) = 0$. Choose the pursuit interval Δ and updating interval δ such that $0 < \delta < \Delta \leq T_0$.*

(1) For $k = 1, 2, 3\ldots$, let $t_k = k\Delta$ be the starting time of the k^{th} agent, i.e. $u_k(t) = 0,\ x_k(t) = x_0$ for $0 \leq t \leq t_k$.
(2) When $t = t_k + i\delta, i = 0, 1, 2, 3, \ldots$, calculate the control $u_t^*(\tau)$ that achieves (subj. to (1)):

$$\begin{cases} \eta(x_k(t), x_{k-1}(t), t, \Delta), & \text{if } x_{k-1}(t) \notin S_Q \\ \qquad (\tau \in [t, t + \Delta]) & \\ \eta_Q(x_k(t), t, T - (t - t_k)), & \text{if } x_{k-1}(t) \in S_Q \\ \qquad (\tau \in [t, t_k + T]) & \end{cases}$$

(3) Apply $u_k(t) = u^*_{t_k+i\delta}(t)$ to the k^{th} agent for $t \in [t_k + i\delta, t_k + (i + 1)\delta)$ if $\Delta + i\delta < T$, or for $t \in [t_k + i\delta, t_k + T)$ otherwise.
(4) Repeat from step 2 until the k^{th} agent reaches S_Q.

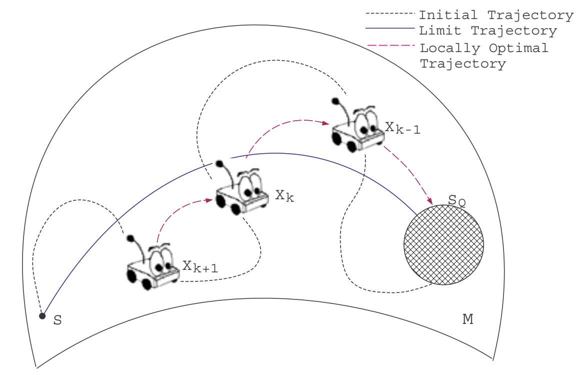

Fig. 1. Illustration of local pursuit on a manifold. Each agent except the first one is trying to "catch up" its predecessor before the predecessor reaches S_Q, and trying to reach S_Q via a locally optimal trajectory otherwise.

The two adjustable parameters in sampled local pursuit (SLP) algorithm are the *pursuit interval* Δ, which defines how frequent agents depart from x_0 (consequently how far away agents are separated), and the *updating interval* δ, which defines the frequency with which each agent updates its trajectory. We will refer to two successive agents as a "pursuit pair". Within a pair, the $(k-1)^{th}$ agent will be known as the "leader", and the k^{th} agent as the "follower". We will refer to the times $t_k^i = t_k + i\delta, i = 0, 1, 2, 3\ldots$ as the "updating times". The SLP algorithm employs two types of pursuit – "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. This is different from the version of sampled local pursuit in [8], which involves only "catching up" stages, and because δ is fixed, rather than determined on-line (as is the case in [16]), the total number of updates performed by each agent is pre-determined.

We now state the main result concerning the limiting behavior of the SLP algorithm. Before proceeding to the main theorem, we must require that the cost of (2) changes "little" for small changes to the endpoints of a trajectory:

Condition 1 *Assume for a generic trajectory $x_1(t)$ there exists an $\varepsilon > 0$ such that for all $a, b_1, b_2 \in \mathbb{D}$ and all $\Delta > 0$, there exists a trajectory $x_2(t)$ such that the cost $C(x_1, 0, T)$ $(x_1(0) = a, x_1(T) = b_1)$ from a to b_1 and cost $C(x_2, 0, T)$ $(x_2(0) = a, x_2(T) = b_2)$ from a to b_2 satisfy*

$$\begin{aligned}&\|b_1 - b_2\|_\infty < \varepsilon \\ \Rightarrow &\|C(x_1, 0, T) - C(x_2, 0, T)\|_\infty < \mathcal{L}\Delta\end{aligned}$$

for some constant $\mathcal{L}$ independent of Δ.

Theorem 1 *Suppose a group of agents evolve under sampled local pursuit, and that at every updating time t_k^i, the locally optimal trajectories from follower to leader are unique. If the updating interval δ and pursuit interval Δ satisfy $0 < \delta < \Delta$ and Condition 1 holds, then the limiting trajectory obtained is unique and locally optimal. It is also smooth if the locally optimal trajectories calculated at every updating time are smooth.*

PROOF. See [8,16].

3 Local Pursuit as a Numerical Computational Tool

Multiple-shooting, together with its various improved forms, could be considered a workhorse of numerical optimization. However, the large storage requirements associated with MS limit the so-called "permitting size" of problems which can be solved on a digital computer [1]. In the following, we briefly review the technique known as multiple shooting (MS) before showing how it improves when combined with SLP.

3.1 A Brief Review of Multiple Shooting

In principle, MS is a type of NLP method[1]. Suppose, for now, that we are interested in minimizing the scalar cost function $F(x) : \mathbb{R}^n \to \mathbb{R}$, subject to m $(m \le n)$ equality constraints $c(x) = 0$, where $c(x)$ is an $(m \times 1)$ vector. To do this, we first introduce the Lagrangian:

$$L(x, \lambda) = F(x) + \lambda^T c(x) \tag{5}$$

The necessary conditions for the point (x^*, λ^*) to be an optimum are satisfied by the stationary points of the Lagrangian [1]:

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

and

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

The equations (6)∼(7) are usually solved via Newton's method. Proceeding formally we obtain the following Karush-Kuhn-Tucker system:

$$\begin{bmatrix} H_L & \nabla_x c(x)^T \\ \nabla_x c(x) & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} -\nabla_x F(x) - \nabla_x c(x)^T \lambda \\ -c(x) \end{bmatrix} \tag{8}$$

where Δx is the "search-direction" and H_L is the Hessian of the Lagrangian

$$H_L = \nabla_x^2 F(x) + \sum_{i=1}^{m} \lambda_i \nabla_x^2 c_i \tag{9}$$

For convenience, we define

$$H = \begin{bmatrix} H_L & \nabla_x c(x)^T \\ \nabla_x c(x) & 0 \end{bmatrix} \tag{10}$$

Suppose now that we are interested in optimizing (2) subject to the dynamics

$$\dot{x} = f(x(t), u(t)). \tag{11}$$

[1] Here, we use the notation:

$$\nabla_x A(x) = \begin{bmatrix} \frac{\partial a_1}{\partial x_1} & \frac{\partial a_1}{\partial x_2} & \cdots & \frac{\partial a_1}{\partial x_n} \\ \frac{\partial a_2}{\partial x_1} & \frac{\partial a_2}{\partial x_2} & \cdots & \frac{\partial a_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial a_n}{\partial x_1} & \frac{\partial a_n}{\partial x_2} & \cdots & \frac{\partial a_n}{\partial x_n} \end{bmatrix}$$

Doing so with MS (here we mainly refer to direct MS) involves breaking up the trajectory into "shorter" pieces [1] by partitioning the time domain $[t_0, t_f] = \cup_{i=1}^{N-1}[t_i, t_{i+1}),\ t_0 = t_1 < \cdots < t_N = t_f$. Each subinterval $[t_i, t_{i+1})$ is called a "segment".

Multiple shooting uses NLP to find the optimal trajectory (i.e. to minimize a scalar function along a trajectory), by defining the NLP variables $\nu = [\nu_1, \ldots, \nu_N]$ to be the arguments – usually they are concatenations of the states and the corresponding controls – at times $t_1, \ldots, t_N$ along a trajectory. The stationary points $[\nu_1^*, \ldots, \nu_N^*]$ obtained via NLP will approach points on the optimal trajectory. Without loss of generality, we choose to sample the state and control vectors at the segment endpoints, and define the NLP variables

$$\nu = \{x_1, u_1, x_2, u_2, \ldots, x_N, u_N\} \tag{12}$$

Notice that the dimension of x_i and u_i, $i = 1, \ldots, N$ is M_x and M_u, respectively.

As the NLP variables are adjusted, one must ensure that they satisfy the the system dynamics (11), and that sequential trajectory segments obey a matching condition at their boundaries. This means that the evolution of (11) from x_i at t_i with input u_i, should steer the system to x_{i+1} at t_{i+1}. This suggests that the following constraints are necessary:

$$c(x) = \begin{bmatrix} x_2 - \bar{x}_1 \\ x_3 - \bar{x}_2 \\ \vdots \\ x_N - \bar{x}_{N-1} \\ \phi_0(x_1, t_0) \\ \phi_T(x_N, t_f) \end{bmatrix} = 0 \tag{13}$$

where the functions $\phi_0(x_1, t_0)$ and $\phi_T(x_N, t_f)$ represent the initial and final condition constraints, respectively, and $\bar{x}_i$ are to be calculated by integrating the differential equation (11) from t_i to t_{i+1}.

In practice, one often computes $\bar{x}_i$ only approximately, instead of integrating the complete equations of motion. For example, Euler's method provides a first-order approximation to the integration of (11):

$$\bar{x}_{i+1} = x_i + hf(x_i, u_i) \tag{14}$$

where h is the time step of each segment. The choice of approximation should be based on the following considerations: First, we have sampled the controls at the segment boundaries (the samples are to be optimized) instead of considering the continuous-time control on the intervals $[t_i, t_{i+1}]$; therefore, we can not compute the precise state evolution by integration. Second, using linear approximation reduces computational burden; on the other hand, higher-order approximations are likely to produce more precise results, but they will also give rise to more complicated equations in (6)∼(13) and the resulting NLP problem will require more time to solve.

One shortcoming of linear approximation is that the time separation between neighboring points must be limited to a small time step h. If h is "too large", then $\bar{x}_{i+1}$ will not be a good estimate of the true state of (11) as it evolves from x_i to x_{i+1} under u_i, and NLP will produce erroneous results. The estimation error is of $o(h^n)$, where $n \in \mathbb{N}$ is the order of the approximation methods [10]. It is clear that a smaller h results in better approximation. For that reason, keeping the segment size small is desirable and is a key feature of MS compared with other single-step shooting methods [1].

Let us assume that we can fix an acceptable h, i.e., one that is small enough to lead to convergence for the associated NLP. Then the permitting size of problems that can be solved by MS depends solely on the number of steps N required to cover the time span $[0, T]$, with $T = (N-1)h$. Thus, solving large-scale problems (with large time spans), requires a large number of segments $N = Th + 1$. For problems with varying dynamics but fixed time span, the requirement for N could be large because nonlinearities in the dynamics make it necessary to use small time steps to ensure that the estimates $\bar{x}_i$ are precise enough and that the associated NLP will convergence.

Remark 2 *For any given set of dynamics and choice of NLP method, the time step h is upper bounded. The permitting size in MS is proportional to the number of segments N, if h is pre-determined.*

Suppose now that the dimensions of system state x and control u are M_x and M_u respectively, and the number of segments is N, then the dimension of NLP variables for a multiple shooting method is $n_x = (M_x + M_u)N$, and the NLP has at least $M_x N$ constraints (the number of constraints could be larger if the final states are fixed). The Hessian in (9) is of dimension $n_x \times n_x = (M_x + M_u)^2 N^2$. To proceed with NLP, one must obtain the solution of (8), which requires finding the solution to a linear system of algebraic equations with dimension at least $(M_u + 2M_x)^2 N^2$. Here we have seen that the number of segments involved in the calculation affects the "degree of labor-consumption" with $O(N^2)$.

3.2 Numerical Optimization by Local Pursuit

As we have seen, the number of segments used in MS affects the dimension of the associated NLP variables and the computational complexity of the NLP problem to be solved. For that reason, it would be desirable to use fewer segments with the MS method. However, because the time step h is upper bounded for a fixed set of dynamics and NLP algorithm, decreasing the number of segments means that NLP process can only deal with trajectories spanning shorter time intervals. This limitation can be circumvented by combining local pursuit with direct MS, to obtain what we refer to as PBMS. Specifically, one can introduce a sequence of simulated agents that pursue each other using the algorithm given in Sec. 2. Each agent will use MS to compute the optimal trajectory from its own state to that of its leader, giving rise to a series of smaller NLP problems, whose time span is limited by the pursuit interval Δ. Although there will be more of these NLP problems to solve, their lower dimension will make it possible to handle larger optimization problems overall.

3.2.1 Decreasing the size of the NLP problems during computation

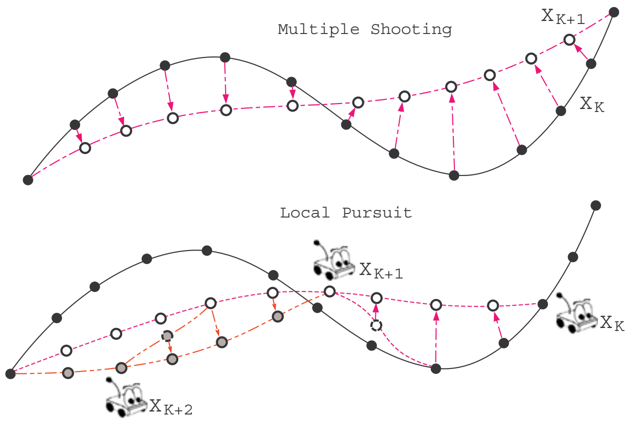

Fig. 2. Local pursuit could decrease the problem size involved in calculation at every updating step. The number of variables calculated in multiple shooting is $N = 13$, and the number of variables calculated by each agent in local pursuit is $N_\Delta = 5$.

For simplicity, fix $h = T/(N-1)$ and select the pursuit interval $\Delta = (N_\Delta - 1)h,\ N_\Delta \in \mathbb{N}$, where N_Δ is the number of segments within Δ. Usually we will have $N_\Delta << N$. For convenience, we also choose the updating interval in the SLP algorithm to be an integer multiple of segment size, $\delta = N_\delta h,\ \ N_\delta \in \mathbb{N}$ so that the agents are always updating their trajectories at the times t_i which we chose to sample the trajectory of (11). At each updating step $t = i\delta$, each agent is solving a NLP over N_Δ segments, with a time span of $(N_\Delta - 1)h$ instead of $(N-1)h$, as illustrated in Fig. 2. Because the computational complexity of MS is related to the square of the number of segments, using $N_\Delta << N$ will significantly decrease the computational burden for each agent. Table 1 shows the dimensions of the NLP vector, ν, the constraints $c(x)$, and the matrix H in Eq. (10), respectively, for MS and PBMS. The dimension of the associated Karush-Kuhn-Tucker system is on the order of N^2

Table 1
Comparing the dimensions of the NLP problem variables when using Multiple Shooting (MS) vs. Pursuit-based Multiple Shooting (PBMS).

	MS	**PBMS**
$\dim(\nu)$	$(M_x + M_u)N$	$(M_x + M_u)N_\Delta$
$\dim(c(x))$	$M_x N$	$M_x N_\Delta$
$\dim(H)$	$((M_u + 2M_x)N)^2$	$((M_u + 2M_x)N_\Delta)^2$

for MS, vs. N_Δ^2 for PBMS. Operating on large matrices consumes large amounts of memory, especially when using non-iterative methods, e.g., the memory of a typical desktop PC can be easily used up by a matrix with dimension of 5000×5000 in Matlab when using 64 bits of digital precision. However, under PBMS, the matrix size is scaled down by a factor of $(N_\Delta/N) \times (N_\Delta/N)$ and hardware requirements can be decreased significantly for any given problem.

We note that under PBMS, each agent needs to solve $(N - N_\Delta)/N_\delta$ "smaller" MS problems in order to reach the target set S_Q from the initial state x_0, but the time needed to do so – denoted by T_a – will generally be less than the total iterative time in multiple shooting, which we will denote by T_{MS}. Of course, the total running time for PBMS – denoted by T_{LP} – may be greater than T_{MS} because local pursuit relies on multiple agents to converge to the optimum. Our experience in prior work on local pursuit and in various numerical experiments (including the example in next Section) has been that, for well-conditioned problems, the convergence rate of PBMS is usually slower than that of MS. As expected, decreasing N_Δ leads to slower convergence for PBMS. However, the added running time comes with the benefit of lower memory requirements, allowing us to handle problems with larger state vectors and longer time horizons. At the same time, if N_Δ is decreased and the available storage is fixed, one can afford to also 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.

3.2.2 Reducing numerical error when H is ill-conditioned

Besides maximum problem size, another important consideration in numerical optimal control, 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 non-convergence. In such settings, correction algorithms, such as Tikhonov regularization, can be applied [10,12]; 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 H in (10) is ill-conditioned, then the numerical solution of (8) introduces small errors which accumulate towards the final segment N. If the problem's time horizon is long, the accumulated error may lead to erroneous results, or prevent MS from converging. 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 failed to converge. Furthermore, if every locally optimal trajectory satisfies the convergence criteria of the numerical method used to solve the "short-range" MS problems between leader-follower pairs, then the convergence of the agents' trajectory sequence is guaranteed by the local pursuit algorithm itself.

3.2.3 Remarks

In summary, the combination of MS with local pursuit can increase the permitting size of problems that can be handled with fixed storage, because at every updating step, each agent deals with a problem with "reduced size". The development of NLP algorithms (and of MS) has followed the growth of the digital computer. The size of a typical application in the early 1960s was $n, m \approx 10$, while in the 1970s and early 1980s most application were of size as $n, m < 100$. With subsequent advances in linear algebra techniques, such as matrix sparsity, and ongoing progress in the semiconductor industry, the permitting size in late 1990s was $n, m \approx 10,000$ [1]. However, when using a fixed time partition, PBMS involves solving problems of size N_Δ instead of N. Therefore, we can address much larger problems under the limits imposed by the hardware. Although it requires more running time, PBMS does provide a feasible solution when the traditional formulation exceeds those limits, making it impossible to proceed.

In cases where MS fails to converge because the errors introduced by the approximation to (11) are too great, PBMS may succeed by reducing the segments N_Δ, thereby reducing the accumulated error over the trajectory of a single agent. In next section we present an example of MS stagnancy caused by an ill-conditioned matrix H and how PBMS can avoid the problem.

4 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. For simplicity, we only consider the effect of the Earth and restrict the problem to a plane, as illustrated in in Fig. 3.

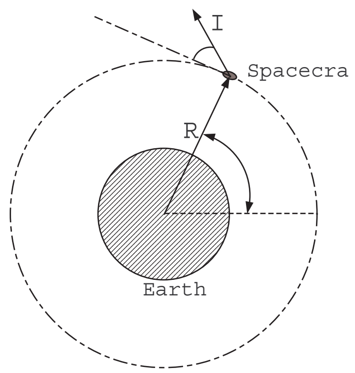

Fig. 3. A planar spacecraft in orbit around the Earth.

The dynamics of this system are

$$\begin{aligned}
\ddot{r} &= \frac{\dot{\theta}^2}{r} - \frac{u_E}{r^2} + \frac{Pu_1}{mg} \\
\ddot{\theta} &= -\frac{2\dot{\theta}\dot{r}}{r} + \frac{Pu_2}{mgr} \\
\dot{m} &= -\frac{P(u_1^2 + u_2^2)}{g^2} \\
u_1 &= I\sin(\varphi) \\
u_2 &= I\cos(\varphi)
\end{aligned} \tag{15}$$

where r is the distance between the spacecraft and the center of Earth, θ is its longitude with respect to the horizontal line, m is the mass of the spacecraft, u_E is the gravitational parameter of Earth, P is a constant concerning engine power, and g is the acceleration of gravity at sea level [17]. The control inputs, u_1 and u_2, are functions of I and φ, the thrust force and thrust steering angle with respect to the tangent of the local orbit.

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

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

[2] The term "stable orbit" means that the spacecraft will remain on this orbit if no external force (other than gravity) is applied, i.e. $\dot{\theta} = \sqrt{u_E/r^3}$.

while steering the system (15) from the initial condition $r(0) = R_0, \dot{r}(0) = 0, \dot{\theta}(0) = \sqrt{u_E/R_0^3}, m(0) = M_0$ to the final condition $r(T) = R_T, \dot{r}(T) = 0, \dot{\theta}(T) = \sqrt{u_E/R_T^3}$, where T is fixed. For simplicity, we assumed there was no upper bound of the thrust force, thus there is no restriction for the control u_1 and u_2.

4.1 Applying MS vs. PBMS

To solve the problem by MS, we used Euler's method to estimate the piecewise integration of Eq. (11) and imposed the following constraints:

$$
\begin{aligned}
0 &= r_{j+1} - \bar{r}_j \\
&= r_{j+1} - (r_j + h_j \dot{r}_j) \\
0 &= \dot{r}_{j+1} - \bar{\dot{r}}_j \\
&= \dot{r}_{j+1} - (\dot{r}_j + h_j(\frac{\dot{\theta}_j^2}{r_j} - \frac{u_E}{r_j^2} + \frac{Pu_{1j}}{m_j g})) \\
0 &= \dot{\theta}_{j+1} - \bar{\dot{\theta}}_j \\
&= \dot{\theta}_{j+1} - (\dot{\theta}_j + h_j(-\frac{2\dot{\theta}_j \dot{r}_j}{r_j} + \frac{Pu_{2j}}{m_j g r_j})) \\
0 &= m_{j+1} - \bar{m}_j \\
&= m_{j+1} - (m_j - h_j \frac{P(u_1^2 + u_2^2)}{g^2}) \\
&\quad \text{for } j = 1, 2, 3 \ldots, N-1 \text{ and} \\
0 &= r_1 - R_0 \\
0 &= \dot{r}_1 - \dot{R}_0 \\
0 &= \dot{\theta}_1 - \dot{\theta}_0 \\
0 &= m_1 - M_0 \\
0 &= r_T - R_T \\
0 &= \dot{r}_T - \dot{R}_T \\
0 &= \dot{\theta}_T - \dot{\theta}_T
\end{aligned} \tag{16}
$$

where $r_i = r((i-1)h), \dot{r}_i = \dot{r}((i-1)h), \ldots$ and $h = T/(N-1)$ was the time step. The dimension of constraints $c(x)$ (m as we denoted before) was $(4 \times (N-1) + 7)$. The NLP variable

$$
\begin{aligned}
\nu = \{& r_1, \ldots, r_N, \dot{r}_1, \ldots, \dot{r}_N, \dot{\theta}_1, \ldots, \dot{\theta}_N, m_1, \ldots, m_N \\
&, u_{11}, \ldots, u_{1N-1}, u_{21}, \ldots, u_{2N-1}\}
\end{aligned} \tag{17}
$$

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

When applying local pursuit, the N should be replaced by N_Δ in the above equations. The dimension of the constraint set and the NLP variable were $(4 \times (N_\Delta - 1) + 7)$ and $(6 \times N_\Delta - 2)$, respectively.

4.2 Results

We solved the problem both by PBMS and standard MS. The criteria for convergence were $\|m(T)_{i+1} - m(T)_i\| \leq 1E-8$ (to guarantee little improvement with future iterations) and $\|c(x)\| \leq (1E-15) \times m$, where m was the dimension of constraints (to guarantee that the trajectory satisfies the system dynamics). The total time T was fixed to 300 minutes. The numerical properties of the problem – and consequently the performance of the two methods – dependeded on the selection of the total number of segments N.

4.2.1 Well-conditioned case

With $N = 101$, the matrix in Eq. (8) was well conditioned (can be verified by its condition number and the fast convergence rate for MS shown below). The performance of MS and PBMS are summarized in Table 2, where c(H) was the condition number of matrix H; The "iterations" column lists the iteration numbers for convergence in multiple shooting, and number of agents needed for local pursuit to converge, respectively.

Table 2
Comparison between Multiple Shooting and Local Pursuit in well-conditioned case

N=101	**MS**	**PBMS** ($N_\Delta = 30$, $N_\delta = 16$)	**PBMS** ($N_\Delta = 60$, $N_\delta = 32$)
Comp. Time	66.8594	793.8594	430.2344
Iterations	14	277	41
$m(T)$	0.524246124	0.524245714	0.524246100
$\|c(x)\|$	3.7144E-14	9.2499E-15	9.9170E-15
Ave(c(H))	1.5931E+6	4.8919E+5	5.3612E+5

We can see that both methods were successful and that the convergence rate of PBMS was slower than that of MS. Increasing N_Δ resulted in increased need for storage and decreased running time. If $N_\Delta = N$, then PBMS reduces to MS. The final trajectories obtained by both methods are shown in Fig. 4.

4.2.2 Ill-conditioned case

When the segment size was halved, i.e., $N \geq 201$, MS became stagnant because the matrix H in (10) was ill-conditioned. The error generated by solving Karush-Kuhn-Tucker system became so large that the NLP algorithm (using Newton's method) was not able to converge. The large condition number of H (our criteria of ill-condition) was due to the large number of segments (in fact H's condition number increased with the size of segments, see Table 3). For PBMS, the reduction in the number of segments for the sub-problems solved by pursuing agents meant a reduction in the condition number of H when using small N_Δ. The results from both methods are summarized in Table 3.

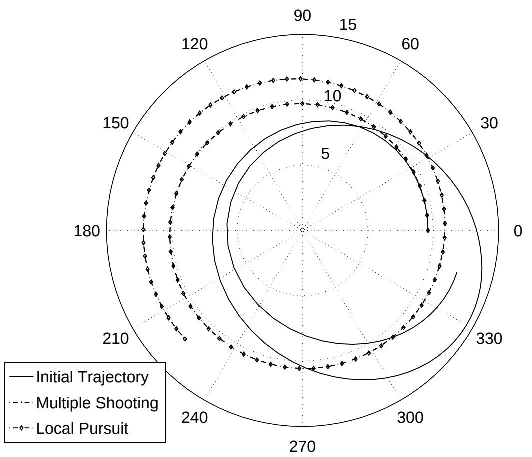

Fig. 4. Trajectories of both methods under well-conditioned case with $N = 101, N_\Delta = 30, N_\delta = 16$. The trajectories obtained from both methods were virtually identical.

Table 3
Comparison between Multiple Shooting and Local Pursuit in ill-conditioned case

N=201	**MS**	**PBMS** ($N_\Delta = 30$, $N_\delta = 16$)	**PBMS** ($N_\Delta = 60$, $N_\delta = 32$)
Time	≥100000	32911.8750	10676.9844
Iteration	≥3000	7223	603
$m(T)$	0.523948746	0.524560874	0.524571236
$\|c(x)\|$	0.01709873	6.0790E-14	1.4019E-14
Ave(c(H))	1.8263E+10	5.0576E+5	4.7881E+5
Max(c(H))	4.3377E+12	6.1142E+5	5.6897E+5
N=301	**MS**	**PBMS** ($N_\Delta = 30$, $N_\delta = 16$)	**PBMS** ($N_\Delta = 60$, $N_\delta = 32$)
Time	≥360000	239601.4219	81264.3906
Iteration	≥3200	30722	2845
$m(T)$	0.527347984	0.524643889	0.524700485
$\|c(x)\|$	0.04135162	2.6051E-13	1.2922E-13
Ave(c(H))	4.5106E+10	5.3032E+5	4.8184E+5
Max(c(H))	7.3201E+10	6.5692E+5	6.5833E+5

In this case, MS could not converge (the values of the constraint residues $c(x)$ did not become sufficiently small), and the iterative process became stagnant. On the other hand, PBMS was effective in producing the optimal trajectory. The final trajectories of both methods are shown in Fig. 5. By comparing to the trajectories generated in the well-conditioned case, it is obvious that the trajectory obtained from multiple shooting was sub-optimal.

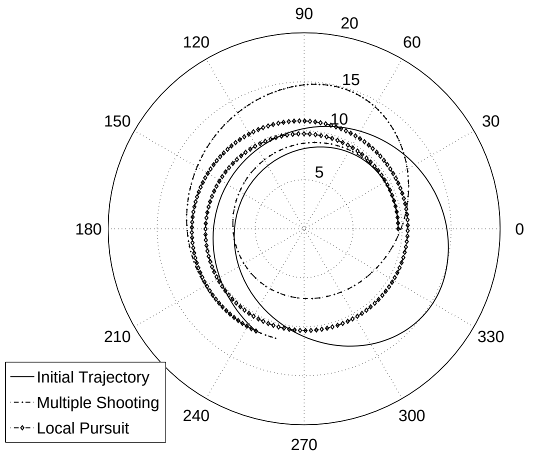

Fig. 5. Trajectories under ill-conditioned case with selection of $N = 201, N_\Delta = 30, N_\delta = 16$. The trajectory obtained from local pursuit was essentially optimal, while the one obtained from multiple shooting was far away from optimum.

4.2.3 Large number of segments

When $N \geq 600$ the orbit transfer problem could not be solved a PC with 1Gb of RAM using MS [3] . On the other hand, PBMS's lower memory requirements meant that the algorithm was able to operate and converge to the optimum. Here we used $N_\Delta = 60, N_\delta = 32$; the final trajectory is shown in Fig. 6.

5 Conclusions and ongoing work

This paper explored the use of a bio-inspired cooperative strategy termed "local pursuit" for solving a class of numerical optimal control problems. We discussed a pursuit algorithm appropriate for 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.

[3] This includes the memory requirements of the function used to compute condition numbers. If we did not want to record condition numbers and only used Gaussian elimination method to solve the linear system, then N could be increased a bit further.

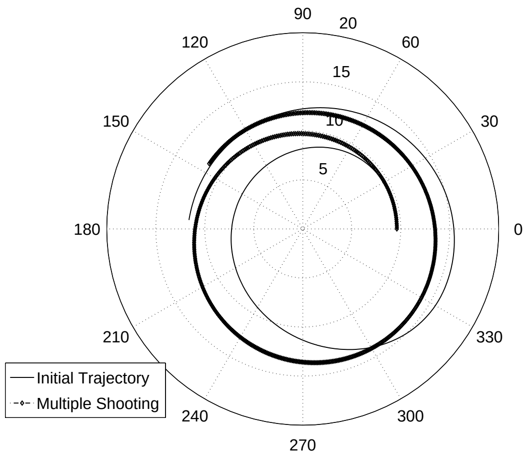

Fig. 6. Spacecraft trajectory in the case of large number of segments, $N = 601, N_\Delta = 60, N_\delta = 32$.

The new method, termed pursuit-based multiple shooting (PBMS) involves a kind of "breaking down" of the original problem down to smaller segments, to be optimized (using MS) by a group of (simulated) agents. PBMS has slower convergence than pure MS but can exceed the limitations of MS due to memory requirements by taking advantage of the cooperation among a group, and of the nonlinear relationship between a problem's time horizon and the memory required to solve it by MS. Furthermore, for an optimal control problem with a given time horizon, PBMS gives one the option to increase the number of intermediate points along the solution, increasing the solution's accuracy and performing better in cases where MS becomes ill-conditioned. These properties of PBMS, and a comparison with MS were illustrated using an example involving orbit-transfer of a spacecraft.

6 Acknowledgment

The authors would like to thank Prof. P. S. Krishnaprasad and Prof. B. Balachandran for their helpful discussion on the subject of numerical computation.

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, April 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, Princeton, New Jersey 08540, 2001.

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

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

[7] D. Hristu-Varsakelis. Robot formations: Learning minimum-length paths on uneven terrain. In *Proceedings of the 8th IEEE Mediterranean Conference on control and Automation*, 2000.

[8] D. Hristu-Varsakelis and C. Shao. Biologically-inspired optimal control: learning from social insects. *the International Journal of Control*, 77(18):1545–1566, Dec. 2004.

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

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

[11] R. Kurazume and S. Hirose. Study on cooperative positioning system: optimum moving strategies for cps-iii. In *Proceeding of the 1998 IEEE International Conference in Robotics and Automation*, volume 4, pages 2896–2903, Leuven, Belgium, May 1998.

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

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

[14] 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.

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

[16] C. Shao and D. Hristu-Varsakelis. A local pursuit strategy for bio-inspired optimal control with partially-constrained final state. Technical Report TR 2005-76, Institute for Systems Research, University of Maryland, College Park, MD 20742, 2005.

[17] 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.

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