Stable, Concurrent Controller Composition for Multi-Objective Robotic Tasks

Anqi Li†, Ching-An Cheng†, Byron Boots†, and Magnus Egerstedt†

***Abstract*— Robotic systems often need to consider multiple tasks concurrently. This challenge calls for controller synthesis algorithms that fulfill multiple control specifications while maintaining the stability of the overall system. In this paper, we decompose multi-objective tasks into subtasks, where individual subtask controllers are designed independently and then combined to generate the overall control policy. In particular, we adopt Riemannian Motion Policies (RMPs), a recently proposed controller structure in robotics, and, RMPflow, its associated computational framework for combining RMP controllers. We re-establish and extend the stability results of RMPflow through a rigorous Control Lyapunov Function (CLF) treatment. We then show that RMPflow can stably combine individually designed subtask controllers that satisfy certain CLF constraints. This new insight leads to an efficient CLF-based computational framework to generate stable controllers that consider all the subtasks simultaneously. Compared with the original usage of RMPflow, our framework provides users the flexibility to incorporate design heuristics through nominal controllers for the subtasks. We validate the proposed computational framework through numerical simulation and robotic implementation.**

I. Introduction

Multi-objective tasks are often involved in the control of robotic systems [1]–[4]. For example, a group of robots may be tasked with achieving a certain formation, moving toward a goal region, while avoiding collisions with each other and obstacles [2]. These types of problems call for algorithms that can systematically generate a stable controller capable of fulfilling multiple control specifications simultaneously.

A classic strategy is to first design a controller for each individual control specification, and then provide a high-level rule to *switch* among them. This idea has been frequently exploited in robotics [5]. For example, it is common practice to switch to a collision avoidance controller when the robot risks colliding with obstacles [5]. The stability of switching systems has been thoroughly investigated, e.g. by finding a common or switched Lyapunov function for the systems among all designed controllers [6]–[9]. However, a fundamental limitation shared by these switching approaches is that only a single controller is active at a time and hence only a subset of the control specifications is considered. If not designed properly, some controllers for secondary tasks might take over the operation for most of the time. For example, when a robot navigates in a cluttered environment, the collision avoidance controller can dominate other controllers and the primary tasks may never be considered [2]. Therefore, it may be more desirable to *blend* controllers rather than impose a hard *switch* between them, so that all tasks can be considered simultaneously.

In robotics, the strategy of weighting controllers for different tasks has been explored in potential field methods [5], [10]. While easy to implement such schemes, it can be difficult to provide formal stability guarantees for the overall "blended" system, especially when the weights are state-dependent. In some cases, the stability of the overall system has been shown through a common Lyapunov function [7], [8], but the existence of a common Lyapunov function is not guaranteed. Finding a common Lyapunov function can be particularly challenging for robotics applications because the tasks can potentially conflict, e.g. the robot may need to move through a cluttered environment to go to the goal.

The framework of null-space or hierarchical control handles this problem by assigning priorities to the tasks, and hence to the controllers [1], [11]. The performance of the high-priority tasks can be guaranteed by forcing the lower-priority controllers to act on the null space of high-priority tasks. However, several problems surface as the number of tasks increases. One problem is the algorithmic singularities introduced by the usage of multiple levels of projections [1], [11]. Most algorithms are designed under the assumption of singular-free conditions. But this assumption is unlikely to hold in practice, especially when there are a large number of tasks, and the system can easily become unstable if the algorithmic singularities occur. In addition, similar to the switching scheme, it is possible that secondary controllers, e.g. collision avoidance controllers, become the ones with high-priorities and the primary task can not be achieved. While several heuristics [12], [13] have been proposed to shift the control priorities dynamically, whether such systems can be globally stabilized in presence of the algorithmic singularities is still an open question [14].

Control Lyapunov functions (CLFs) and control barrier functions (CBFs) constitute another class of methods to encode multiple control specifications [2], [4], [15]. In the CLF and CBF frameworks, the control specifications are encoded as constraints on the time derivatives of Lyapunov or barrier function candidates, and a control input that satisfies all the constraints is solved through a constrained optimization problem. However, in the case of conflicting specifications, the CLF and CBF frameworks suffer from feasibility problems [16], i.e. there does not exist any controller that satisfies all the control specifications. Although the CLF constraints

*This work was sponsored in part by Grant No. W911NF-17-2-0181 from the U.S. Army Research Laboratory DCIST CRA.

†Anqi Li, Ching-An Cheng, Byron Boots, and Magnus Egerstedt are with the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332, USA. Email: {anqi.li, cacheng, magnus}@gatech.edu, bboots@cc.gatech.edu

can be relaxed through slack variables [15], they also add a new set of hyperparameters to trade off the importance of different specifications; care must be taken in tuning these hyperparameters in order to achieve desired performance properties and maintain stability. Finally, it can be hard to encode certain high-dimensional control specifications, such as damping behaviors, as CLF or CBF constraints.

In this paper, we focus on *weighting* individual controllers. We aim to address two interrelated questions:

> How can controllers be composed while guaranteeing system stability?
>
> How should individual controllers be designed so that they can be easily combined?

Although ensuring stability is challenging for arbitrary blending schemes, we design a systematic process to combine controllers so that the stability of the overall system is guaranteed. Our framework considers all control specifications simultaneously, while providing the flexibility to vary the importance of different controllers based on the robot state. Moreover, instead of considering specifications in the configuration space, we allow for controllers defined directly on different spaces or manifolds[1] for different specifications.

This separation can largely simplify the design and computation of each individual controller, because it only concerns a possibly lower-dimensional manifold that is directly relevant to a particular control specification. For example, controllers for different links of a robot manipulator can be designed in their corresponding (possibly non-Euclidan) workspaces. We leverage a recent approach to controller synthesis in robotics, Riemannian Motion Policies (RMPs) [3] and RMPflow [17], which have been successfully deployed on robot manipulators [3], [17] and multi-robot systems [18]. An RMP is a mathematical object that is designed to describe a controller on a manifold, and RMPflow is a computational framework for combining RMPs designed on different task manifolds into a controller for the entire system. A particular feature of RMPflow is the use of state-dependent importance weightings of controllers based on the properties of the corresponding manifolds. It is show in [17] that when RMPs are generated from Geometric Dynamical Systems (GDSs), the combined controller is Lyapunov-stable.

RMPs and RMPflow were initially studied in terms of the geometric structure of second-order differential equations [17], where Riemannian metrics on manifolds (of GDSs) naturally provide a geometrically-consistent notion of task importance and hence a mechanism to combine controllers (i.e. RMPflow). While differential geometry provides a mathematical interpretation of RMPflow, in practice, the restriction to GDSs for control specifications could limit performance and make controller design difficult.

To overcome this limitation, we revisit RMPflow with a rigorous CLF treatment and show that the existing computational framework of RMPflow actually ensures stability for a *larger* class of systems than GDSs. This discovery is made possible by an alternative stability analysis of RMPflow and an induction lemma that characterizes how the stability of individual controllers is propagated to the combined controller in terms of CLF constraints. Hence, we can reuse RMPflow to stably combine a range of controllers, not limited to the ones consistent with GDSs. To demonstrate, we introduce a computational framework called *RMPflow–CLF*, where we augment RMPflow with CLF constraints to generate a stable controller given user-specified nominal controllers for each of the control specifications. This allows users to incorporate additional design knowledge given by, e.g. heuristics, motion planners, and human demonstrations, without worrying about the geometric properties of the associated manifolds. RMPflow–CLF can be viewed as a soft version of the QP–CLF framework [4] that guarantees the stability of the overall system, while ensuring feasibility even when control specifications are conflicting.

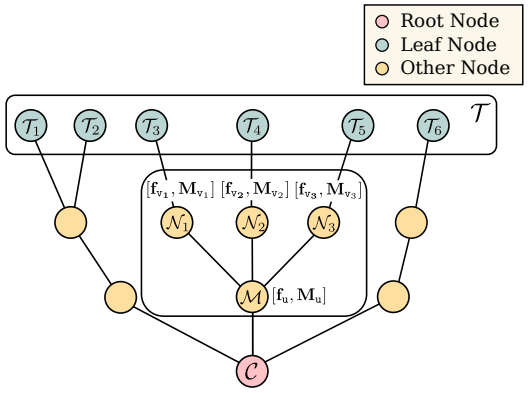


Fig. 1: An example of an RMP-tree. See text for details.

II. Background

We first review RMPflow [3], [17] and CLFs [4], [15], which are different ways to combine control specifications.

A. Riemannian Motion Policies (RMPs) and RMPflow

Consider a robot with configuration space $\mathcal{C}$ which is a smooth d-dimensional manifold. We assume that $\mathcal{C}$ admits a global *generalized coordinate* $\mathbf{q} : \mathcal{C} \to \mathbb{R}^d$ and follow the assumption in [17] that the system can be feedback linearized in such a way that it can be controlled directly through the generalized acceleration[2], i.e. $\ddot{\mathbf{q}} = \mathbf{u}(\mathbf{q}, \dot{\mathbf{q}})$. We call $\mathbf{u}$ a *control policy* or a *controller*, and $(\mathbf{q}, \dot{\mathbf{q}})$ the *state*.

The task is often defined on a different manifold from $\mathcal{C}$ called the *task space*, denoted $\mathcal{T}$. A task may admit further structure as a composition of *subtasks* (e.g. reaching a goal, avoiding collision with obstacles, etc.). In this case, we can treat the task space as a collection of multiple lower-dimensional *subtask spaces*, each of which is a manifold. In other words, each subtask space is associated with a control specification and together the task space $\mathcal{T}$ describes the overall multi-objective control problem.

[1]Specifications defined on non-Euclidean manifolds are common in robotics; for example, in obstacle avoidance, obstacles become holes in the space and the geodesics flow around them [3].

[2]This setup can be extended to torque controls as in [1].

Ratliff et al. [3] propose *Riemannian Motion Policies* (RMPs) to represent control policies on manifolds. Consider an m-dimensional manifold M with a global coordinate $x \in \mathbb{R}^m$. An RMP on M can be represented by two forms, its *canonical form* $(a, M)^M$ and its *natural form* $[f, M]^M$, where $a : (x, \dot{x}) \mapsto a(x, \dot{x})$ is the desired acceleration, $M : (x, \dot{x}) \mapsto M(x, \dot{x}) \in \mathbb{R}^{m \times m}_+$ is the inertial matrix, and $f = Ma$ is the desired force. It is important to note that M and f do not necessarily correspond to physical quantities; M defines the importance of an RMP when combined with other RMPs, and f is proposed for computational efficiency.

RMPflow [17] is a recursive algorithm to generate control policies on the configuration space given the RMPs of subtasks. It introduces: 1) a data structure, the *RMP-tree*, for computational efficiency; and 2) a set of operators, the *RMP-algebra*, to propagate information across the RMP-tree.

An RMP-tree is a directed tree, which encodes the computational structure of the task map from C to T (see Fig. 1). In the RMP-tree, a node is associated with the state and the RMP on a manifold, and an edge is augmented with a smooth map from a parent-node manifold to a child-node manifold. In particular, the root node r is associated with the state of the robot $(q, \dot{q})$ and its control policy on the configuration space $(a_r, M_r)^C$, and each leaf node l_k is associated with the RMP $(a_{l_k}, M_{l_k})^{T_k}$ for a subtask, where T_k is a subtask manifold. Recall the collection $\{T_k\}_{k=1}^K$ is the task space T, where K is the number of tasks.

To illustrate how the RMP-algebra operates, consider a node u with N child nodes $\{v_j\}_{j=1}^N$. Let e_j denote the edge from u to v_j and let ψ_{e_j} be the associated smooth map. Suppose that u is associated with an RMP $[f_u, M_u]^M$ on a manifold M with coordinate x, and v_j is associated with an RMP $[f_{v_j}, M_{v_j}]^{N_j}$ on a manifold N_j with coordinate y_j. (Note that here we represent the RMPs in their natural form.) The RMP-algebra consists of the following three operators:

1) `pushforward` is the operator to forward propagate the *state* from the parent node u to its child nodes $\{v_j\}_{j=1}^N$. Given the state $(x, \dot{x})$ from u, it computes $(y_j, \dot{y}_j) = (\psi_{e_j}(x), J_{e_j}(x)\dot{x})$ for each child node v_j, where $J_{e_j} = \partial_x \psi_{e_j}$ is the Jacobian matrix of ψ_{e_j}.
2) `pullback` is the operator to backward propagate the RMPs from the child nodes to the parent node. Given $\{[f_{v_j}, M_{v_j}]^{N_j}\}_{j=1}^N$ from the child nodes, the RMP $[f_u, M_u]^M$ for the parent node u is computed as,
$$f_u = \sum_{j=1}^N J_{e_j}^\top (f_{v_j} - M_{v_j} J_{e_j} \dot{x}), \quad M_u = \sum_{j=1}^N J_{e_j}^\top M_{v_j} J_{e_j}.$$
3) `resolve` maps an RMP from its natural form to its canonical form. Given $[f_u, M_u]^M$, it outputs $(a_u, M_u)^M$ with $a_u = M^\dagger f_u$, where $\dagger$ denotes Moore-Penrose inverse.

RMPflow performs control policy generation through running the RMP-algebra on the RMP-tree. It first performs a forward pass, by recursively calling `pushforward` from the root node to the leaf nodes to update the state associated with each node on the RMP-tree. Second, every leaf node l_k *evaluates* its natural form RMP $f(f_{l_k}, M_{l_k})^{T_{l_k}}\}_{k=1}^K$ given its associated state. Then, RMPflow performs a backward pass, by recursively calling `pullback` from the leaf nodes to the root node to back propagate the RMPs in the natural form. Finally, `resolve` is applied to the root node to transform the RMP $[f_r, M_r]^C$ into its canonical form $(a_r, M_r)^C$ and set the control policy as $u = a_r$.

RMPflow was originally analyzed based on a differential geometric interpretation. Cheng et al. [17] consider the inertial matrix M generated by a Riemannian metric on the *tangent bundle* of the manifold M (denoted as TM). Let $G : (x, \dot{x}) \mapsto G(x, \dot{x}) \in \mathbb{R}^{m \times m}_+$ be a (projected) Riemannian metric and define the *curvature terms*

$$\begin{aligned} \Gamma_G(x, \dot{x}) &:= \frac{1}{2} \sum_{i=1}^m \dot{x}_i \partial_x g_i(x, \dot{x}), \\ \Psi_G(x, \dot{x}) &:= \dot{G}(x, \dot{x})\dot{x} - \frac{1}{2} \nabla_x (\dot{x}^\top G(x, \dot{x}) \dot{x}), \end{aligned} \tag{1}$$

where $\dot{G}(x, \dot{x}) := [\partial_x g_i(x, \dot{x})\dot{x}]_{i=1}^m$, $g_i(x, \dot{x})$ is the ith column of $G(x, \dot{x})$, and x_i is the ith component of x. The inertial matrix $M(x, \dot{x})$ is then related to $G(x, \dot{x})$ through,

$$M(x, \dot{x}) = G(x, \dot{x}) + \Gamma_G(x, \dot{x}). \tag{2}$$

Under this geometric interpretation, RMPs on a manifold M can be (but not necessarily) generated from a class of systems called *Geometric Dynamical Systems* (GDSs) [17], whose dynamics are on the form of

$$M(x, \dot{x})\ddot{x} + \Gamma_G(x, \dot{x}) = -\nabla_x \Phi(x) - B(x, \dot{x})\dot{x}, \tag{3}$$

where $B : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^{m \times m}_+$ is the *damping matrix*, and $\Phi : \mathbb{R}^m \to \mathbb{R}$ is the *potential function*. When $G(x, \dot{x}) = G(x)$, the GDSs reduce to the widely studied *Simple Mechanical Systems* [19].

The stability properties of RMPflow is analyzed in [17] under the assumption that *every* leaf-node RMP is specified as a GDS (3). Before stating the stability theorem, let us define the metric, damping matrix, and potential function for every node in the RMP-tree: For a leaf node, its metric, damping matrix, and potential are defined naturally by its underlying GDS. For a non-leaf node u with N children $\{v_j\}_{j=1}^N$, these terms are defined recursively by the relationship,

$$G_u = \sum_{j=1}^N J_{e_j}^\top G_{v_j} J_{e_j}, \; B_u = \sum_{j=1}^N J_{e_j}^\top B_{v_j} J_{e_j}, \quad \Phi_u = \sum_{j=1}^N \Phi_{v_j} \circ \psi_{e_j}, \tag{4}$$

where G_{v_j}, B_{v_j} and Φ_{v_j} are the metric, damping matrix, and potential function for the jth child. The stability results for RMPflow are stated below.

Theorem II.1. [17] *Let G_r, B_r, and Φ_r be the metric, damping matrix, and potential function of the root node defined in (4). If each leaf node is given by a GDS, $G_r, B_r \succ 0$, and M_r is non-singular, then the system converges to a forward invariant set $\mathcal{C}_1 := \{(q, \dot{q}) : \nabla_q \Phi_r = 0, \dot{q} = 0\}$.*

B. Control Lyapunov Functions (CLFs)

Control Lyapunov Function (CLF) methods [4], [15], [20] encode control specifications as Lyapunov function candidates. In these methods, controllers are designed to satisfy the inequality constraints on the time derivative of the Lyapunov function candidates.

Consider a dynamical system in control-affine form,

$$\dot{x} = f(x) + g(x)u, \tag{5}$$

where $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ are the state and control input for the system. We assume that f and g are locally Lipschitz continuous, and the system (5) is forward complete, i.e. $x(t)$ is defined for all $t \geq 0$. For second-order systems considered by RMPflow, we have $x = [y^\top \dot{y}^\top]^\top$,

$$f(x) = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \quad g(x) = \begin{bmatrix} 0 \\ I \end{bmatrix}. \tag{6}$$

Suppose that a Lypapunov function candidate $V(x)$ is designed for a control specification. The control input is then required to satisfy a CLF constraint, e.g. $\dot{V} \leq \alpha(V)$, where $\alpha : \mathbb{R}_+ \to \mathbb{R}_+$ is a locally Lipschitz class $\mathcal{K}$ function [21] (i.e. α is strictly increasing and $\alpha(0) = 0$). In the case of control-affine system, the CLF constraint becomes a linear inequality constraint on control input u given state x,

$$L_g V(x)u \leq -L_f V(x) - \alpha(V(x)), \tag{7}$$

where $L_f V$ and $L_g V$ are the *Lie derivatives* of V along f and g, respectively.

When there are multiple control specifications, one can design Lyapunov function candidates $\{V_k\}_{k=1}^K$ separately. Then the controller synthesis problem becomes finding a controller that satisfies all the linear inequalities given by the Lyapunov function candidates. Morris et al. [4] propose a computational framework, QP–CLF, that solves for the controller through a Quadratic programming (QP) problem that augments the constraints with a quadratic objective:

$$\begin{aligned} \min_{u} \quad & \frac{1}{2} u^\top H(x) u + F(x)^\top u \\ \text{s.t.} \quad & L_g V_k(x) u \leq -L_f V_k(x) - \alpha_k(V_k(x)), \\ & \forall k \in \{1, \dots, K\}. \end{aligned} \tag{8}$$

However, when the specifications are conflicting, it may not be possible to enforce the CLF constraints for all $\{V_k\}_{k=1}^K$ since the optimization problem (8) can become infeasible [4]. In [15], Ames et al. introduce *slack variables* $\{\epsilon_k\}_{k=1}^K$ so that the optimization problem is always feasible. Let $u = [u^\top \epsilon_1 \cdots \epsilon_K]^\top$ denote all decision variables. Then the relaxed optimization problem becomes,

$$\begin{aligned} \min_{u} \quad & \frac{1}{2} u^\top H(x) u + F(x)^\top u \\ \text{s.t.} \quad & L_g V_k(x) u \leq -L_f V_k(x) - \alpha_k(V_k(x)) + \epsilon_k, \\ & \forall k \in \{1, \dots, K\}, \end{aligned} \tag{9}$$

where $H(x)$ and $F(x)$ encode how the original objective function and the CLF constraints are balanced. However, care must be taken in tuning $H(x)$ and $F(x)$ to achieve desired performance properties and maintain stability.

III. The CLF Interpretation of RMPflow

The goal of this paper is to combine control policies specified for subtask manifolds into a control policy for the robot with stability guarantees. RMPflow provides a favorable computational framework but its original analysis is limited to subtask control policies generated by GDSs. This assumption is rather unsatisfying, as the users need to encode the control specifications as GDS behaviors. Further, this restriction can potentially limit the performance of the subtasks and result in unnecessary energy consumption by the system. Although empirically RMPflow has been shown to work with non-GDS leaf policies [17], it is unclear if the overall system is still stable.

In this section, we show that the RMP-algebra actually preserves the stability of a wider range of leaf-node control policies than GDSs. We relax the original GDS assumption in [17] to a more general CLF constraint on each leaf node, and provide a novel stability analysis of RMPflow. These results allow us to reuse RMPflow for combining a more general class of control policies, which we will demonstrate by combining controllers based on CLF constraints.

A. An Induction Lemma

In order to establish CLF constraints on leaf nodes that guarantee stability, we first need to understand how the RMP-algebra, especially the `pullback` operator, connects the stability results of the child nodes to the parent node.

Again, let us consider a node u with N child nodes $\{v_j\}_{j=1}^N$, in which u is associated with a manifold M with coordinate x, and v is associated with a manifold N_j with coordinate y_j. In addition, let $\psi_{e_j} : x \mapsto y_j$ be the smooth map between manifolds M and N_j. We furthur assume that ψ_{e_j} is surjective, i.e., $N_j = \psi_{e_j}(M)$.

Let us associate each child node v_j with a proper, continuously differentiable and lower-bounded potential Ψ_{v_j} on its manifold N_j along with a continuously differentiable Riemannian metric G_{v_j} on its tangent bundle TN_j. Then, for node v_j, there is a natural Lyapunov function candidate,

$$V_{v_j}(y_j, \dot{y}_j) = \frac{1}{2} \dot{y}_j^\top G_{v_j}(y_j, \dot{y}_j) \dot{y}_j + \Psi_{v_j}(y_j), \tag{10}$$

and an associated natural-formed RMP $[f_{v_j}, M_{v_j}]^{N_j}$, where f_{v_j} is the force policy, and M_{v_j} is defined by G_{v_j} as in (2). We shall further assume that M_{v_j} is locally Lipschitz continuous for the ease of later analysis. By construction of the RMP-algebra, these Lyapunov function candidates and RMPs of the child nodes $\{v_j\}_{j=1}^N$ define, for the parent node u, a Lyapunov function candidate

$$V_u(x, \dot{x}) = \frac{1}{2} \dot{x}^\top G_u(x, \dot{x}) \dot{x} + \Psi_u(x). \tag{11}$$

where G_u and Ψ_u are given in (4).

The following lemma states how the decay-rate of V_u is connected to the decay-rates of $\{V_{v_j}\}_{j=1}^N$ via `pullback`.

Lemma III.1. *For each child node v_j, assume that $f_{v_j} = M_{v_j} \ddot{y}_j$ renders $\dot{V}_{v_j}(y_j, \dot{y}_j) \leq -\alpha_{V_{v_j}}(V_{v_j}(y_j, \dot{y}_j))$ for V_{v_j} in (10). If the parent node u follows dynamics $f_u = M_u(x, \dot{x})\ddot{x}$,*

where f_u and M_u are given by pullback , then $V_u(x, \dot{x}) = \sum_{j=1}^{N} U_{v_j}(y_j, \dot{y}_j)$ for V_u in (11).

Proof. For notational convenience, we suppress the arguments of functions. First, note that,

$$V_u = \frac{1}{2} \sum_{j=1}^{N} \dot{x}^\top J_{e_j}^\top G_{v_j} J_{e_j} \dot{x} + \sum_{j=1}^{N} {}_{v_j} = \sum_{j=1}^{N} V_{v_j}: \quad (12)$$

As G_{v_j} is a function in both y and $\dot{y}$, following a similar derivation as in [17], we can show

$$\dot{V}_u = \sum_{j=1}^{N} \dot{y}_j^\top M_{v_j} \ddot{y}_j + \frac{1}{2} \dot{y}_j^\top G_{v_j}^{y_j} \dot{y}_j + \dot{y}_j^\top r_{y_j \quad v_j}; \quad (13)$$

where M_{v_j} and $G_{v_j}^{y_j}$ are the inertial matrix and the curvature term de ned in Section II-A.

Note that $\dot{y}_j = J_{e_j} \dot{x} + \dot{J}_{e_j} \dot{x}$, where $\dot{x}$ is given by the RMP $[f_u, M_u]^M$. Hence, the rst term can be rewritten as

$$\begin{aligned}
&\sum_{j=1}^{N} \dot{y}_j^\top M_{v_j} \ddot{y}_j = \dot{x}^\top \sum_{j=1}^{N} J_{e_j}^\top M_{v_j} \left(J_{e_j} \ddot{x} + \dot{J}_{e_j} \dot{x} \right) \\
&= \dot{x}^\top f_u + \dot{x}^\top \sum_{j=1}^{N} J_{e_j}^\top M_{v_j} \dot{J}_{e_j} \dot{x} \\
&= \dot{x}^\top \sum_{j=1}^{N} J_{e_j}^\top \left(f_{v_j} - M_{v_j} \dot{J}_{e_j} \dot{x} \right) + \dot{x}^\top \sum_{j=1}^{N} J_{e_j}^\top M_{v_j} \dot{J}_{e_j} \dot{x} \\
&= \dot{x}^\top \sum_{j=1}^{N} J_{e_j}^\top f_{v_j} = \sum_{j=1}^{N} \dot{y}_j^\top f_{v_j}:
\end{aligned}$$

The time-derivative of V_u can then be simpli ed as

$$\dot{V}_u = \sum_{j=1}^{N} \dot{y}_j^\top f_{v_j} + \frac{1}{2} \dot{y}_j^\top G_{v_j}^{y_j} \dot{y}_j + \dot{y}_j^\top r_{y_j \quad v_j}: \quad (14)$$

By assumption on v_j , we also have

$$\dot{y}_j^\top f_{v_j} + \frac{1}{2} \dot{y}_j^\top G_{v_j}^{y_j} \dot{y}_j + \dot{y}_j^\top r_{y_j \quad v_j} = \dot{U}_{v_j}(y_j, \dot{y}_j): \quad (15)$$

The statement follows then from the two equalities.

We can use Lemma III.1 to infer the overall stability of RMP ow. For an RMP-tree with K leaf nodes, let leaf node l_k be de ned on task space T_k with coordinates z_k and has a natural Lyapunov function candidate

$$V_{l_k}(z_k, \dot{z}_k) = \frac{1}{2} \dot{z}_k^\top G_{l_k}(z_k, \dot{z}_k) \dot{z}_k + {}_{l_k}(z_k): \quad (16)$$

for some potential function $_{l_k}$ and positive-de nite Riemannian metric G_{l_k} de ned as above. By Lemma III.1, if each leaf node l_k satis es a CLF constraint,

$$\dot{V}_{l_k}(z_k, \dot{z}_k) = \quad U_{l_k}(z_k, \dot{z}_k); \quad (17)$$

then a similar constraint is satis ed by the root node. This observation is summarized below without proof.

Proposition III.2. For each leaf node l_k, assume that $f_{l_k} = M_{l_k} \ddot{z}_{l_k}$ renders (17) for (16). Consider the Lyapunov function candidate at the root node $V_r(q, \dot{q})$ de ned through (11). Then, for the root node control policy of RMP ow, it holds $\dot{V}_r(q, \dot{q}) = \sum_{k=1}^{K} \dot{U}_{l_k}({}_{r! l_k}(q), J_{r! l_k} \dot{q})$, where ${}_{r! l_k}$ is the map from C to T_k, which can be obtained through composing maps from the root node to the leaf node l_k on the RMP-tree, and $J_{r! l_k} = \partial {}_{r! l_k}$.

Note that Proposition III.2 provides an alternative way to show the stability results of RMP ow.

Corollary III.2.1. For each leaf node l_k, assume that f_{l_k} is given by a GDS $(T_k, G_{l_k}, B_{l_k}, {}_{l_k})$. Consider the Lyapunov function candidate at the root node $V_r(q, \dot{q})$ de ned recursively through (11). Then we have $\dot{V}_r(q, \dot{q}) = \dot{q}^\top B_r(q, \dot{q}) \dot{q}$ under the resulting control policies from RMP ow, where B_r is de ned recursively through (4).

With Corollary III.2.1, we then can show Theorem II.1 by invoking LaSalle's invariance principle [21].

B. Global Stability Properties

More importantly, by Proposition III.2, we can nd how CLF constraints are propagated from the leaf nodes to the root node through pullback .

Proposition III.3. For each leaf node l_k, assume that $f_{l_k} = M_{l_k} \ddot{z}_{l_k}$ renders, for V_{l_k} in (16),

$$\dot{V}_{l_k}(z_k, \dot{z}_k) \quad {}_k(k\dot{z}_k k); \quad (18)$$

where ${}_k$ is a locally Lipschitz continuous class K functions [21]. Consider the Lyapunov function candidate at the root node $V_r(q, \dot{q})$ de ned recursively through (11). Then

$$\dot{V}_r(q, \dot{q}) \quad \sum_{k=1}^{K} {}_k(kJ_{r! l_k} \dot{q} k) \quad (19)$$

under the resulting control policies from RMP ow.

With this insight, we state a new stability theorem of RMP ow by applying LaSalle's invariance principle. We assume that the inertia matrix at the root node M_r is nonsingular for simplicity, so that the actual control input can be solved through the resolve operation; a suf cient condition for M_r being nonsingular is provided in [17].

Theorem III.4. For each leaf node l_k, assume that $f_{l_k} = M_{l_k} \ddot{z}_{l_k}$ renders (18). Suppose that M_r is nonsingular, and the task space T is an immersion of the con guration space C. Then the control policy generated by RMP ow renders the system converging to the forward invariant set

$$C_1 := \left\{ (q, \dot{q}) : \dot{q} = 0, \sum_{j=1}^{K} J_{r! l_k}^\top f_{l_k} = 0 \right\}: \quad (20)$$

Further if, for all leaf nodes, $f_{l_k} = r_{z_k \quad l_k}(z_k)$ when $\dot{z}_k = 0$, the system converges to the forward invariant set

$$C_1 := \{ (q, \dot{q}) : r_{q \quad r}(q) = 0, \dot{q} = 0 \}; \quad (21)$$

where ${}_r$ is the potential in V_r de ned recursively in (4).

Proof. By assumption, V_r is proper, continuously differentiable and lower bounded. Hence, the system converges to the largest invariant set in $\{(q, \dot{q}) : \dot{V}_r(q, \dot{q}) = 0\}$ by LaSalle's invariance principle [21]. By (19) in Proposition III.3, $\dot{V}_r = 0$

if and only if $J_{r!l_k}\dot{q} = 0$ for all $k = 1, \dots, K$. Since $\mathcal{C}$ is immersed in $\mathcal{T}$, we have $\dot{q} = 0$. Hence, the system converges to a forward invariant set $\mathcal{C}_1 := \{(q, \dot{q}) : \dot{q} = 0\}$. Any forward invariant set in $\mathcal{C}_1$ must have $\ddot{q} = 0$, which implies that $f_r = 0$ as M_r is nonsingular. Note that f_r is given by the pullback operation recursively, hence,

$$0 = f_r = \sum_{k=1}^{K} J_{r!l_k}^{\top}(f_{l_k} - M_{l_k} J_{r!l_k}\dot{q}) = \sum_{k=1}^{K} J_{r!l_k}^{\top} f_{l_k}$$

where the last equality follows from $\dot{q} = 0$. Thus, the system converges to the forward invariant set in (20).

Now, assume that $f_{l_k} = r_{z_k l_k}(z_k)$ when $\dot{z}_k = 0$ (which is implied by $\dot{q} = 0$). Notice that by the definition of ψ_r in (4), $\psi_r(q) = \sum_{k=1}^{K} \psi_{l_k}(z_k)$. By the chain rule,

$$r_{q\,r}(q) = \sum_{k=1}^{K} J_{r!l_k}^{\top} r_{z_k l_k}(z_k) = \sum_{k=1}^{K} J_{r!l_k}^{\top} f_{l_k}.$$

Hence $\sum_{k=1}^{K} J_{r!l_k}^{\top} f_{l_k} = 0$ implies $r_{q\,r}(q) = 0$. Thus, the system converges forwardly to (21).

Theorem III.4 implies that subtask controllers satisfying CLF constraints (18) can be stably combined by RMP-flow.

IV. A Computational Framework for RMP-flow with CLF Constraints

We introduce a computational framework for controller synthesis based on the stability results presented in Section III. The main idea is to leverage Proposition III.3, which says that RMP-flow is capable of preserving CLF constraints in certain form. Recall that for leaf node l_k, the constraint on the time-derivative of the Lyapunov function in Proposition III.3 is $\dot{V}_{l_k}(z_k, \dot{z}_k) \le -\alpha_{l_k}(\|\dot{z}_k\|)$. Combined with the particular choice of leaf-node Lyapunov function candidate in (16), this yields a CLF constraint[3]

$$\dot{z}_k^{\top} f_{l_k} \le \dot{z}_k^{\top} r_{z_k l_k} + \alpha_{G_{l_k}}(\|\dot{z}_k\|), \qquad (22)$$

where G_{l_k} is defined in (1). Proposition III.3 shows that, when the leaf-node control policies satisfy (22), RMP-flow will yield a stable controller under suitable conditions. This provides a constructive principle to synthesize controllers.

A. Algorithm Details

Assume that some nominal controller $u_{l_k}^d$ is provided by the specification of subtask k. We design the leaf-node controller as a minimally invasive controller that modifies the nominal controller as little as possible while satisfying the CLF constraint (22):

$$f_{l_k} = \arg\min_{f_{l_k}} \|f_{l_k} - M_{l_k} u_{l_k}^d\|_{P_{l_k}}^2 \qquad (23)$$
$$\text{s.t.: } \dot{z}_k^{\top} f_{l_k} \le \dot{z}_k^{\top} r_{z_k l_k} + \alpha_{G_{l_k}}(\|\dot{z}_k\|)$$

where $P_{l_k} \succ 0$ and M_{l_k} is given by G_{l_k} through (2). Possible choices of P_{l_k} include the identity matrix I and the inverse of the inertial matrix $M_{l_k}^{-1}$. In particular, $P_{l_k} = M_{l_k}^{-1}$ yields an objective function equivalent to $\|a_{l_k} - u_{l_k}^d\|_{M_{l_k}}^2$, where a_{l_k} is the acceleration policy of the node.

We combine this minimally invasive controller design with RMP-flow as a new computational framework for controller synthesis, called RMP-flow–CLF. RMP-flow–CLF follows the same procedure as the original RMP-flow [17] as is discussed in Section II-A. The difference is that the leaf nodes solve for the RMPs based on the optimization problem (23) during the evaluation step. Note that (23) is a QP problem with a single linear constraint, so it can be solved analytically by projecting $M_{l_k} u_{l_k}^d$ onto the half-plane given by the constraint.

B. Stability Properties

The form of (23) together with Theorem III.4 and the results of [4] yields the following theorem:

Theorem IV.1. Under the assumptions in Theorem III.4, if $\{u_{l_k}^d\}_{k=1}^K$, $\{M_{l_k}\}_{k=1}^K$, all edge Jacobians and their derivatives are locally Lipschitz continuous, then the control policy generated by RMP-flow–CLF is locally Lipschitz continuous and renders the system converging forwardly to (20).

Proof. By Theorem III.4, the system converges to (20). By [4], for all $k \in \{1, \dots, K\}$, f_{l_k} is locally Lipschitz. Since under the assumption pullback and resolve preserves Lipschitz continuity; the statement follows.

Note that RMP-flow can be interpreted as a soft version of the QP–CLF formulation [4] that enforces the decay-rates of all Lyapunov function candidates (8). Meanwhile, compared with the QP–CLF framework with slack variables [15] that requires the users to design the objective function trade-off between control specifications, RMP-flow provides a structured way to implicitly generate such an objective function so that the system is always stable.

It should be noted that the system can also be stabilized by directly enforcing a single constraint on the time-derivative of the combined Lyapunov function candidate (19) at the root node, rather than enforcing the CLF constraint at every leaf node (18). However, this can be less desirable: although the stability can be guaranteed for the resulting controller, the behavior of each individual subtask is no longer explicitly regulated. By contrast, the approach with leaf-node CLF constraints allows the users to design and test the controllers from (23) independently. This allows for designing controllers that can be applied to robots with different kinematic structures, which is a significant feature of RMP-flow [17].

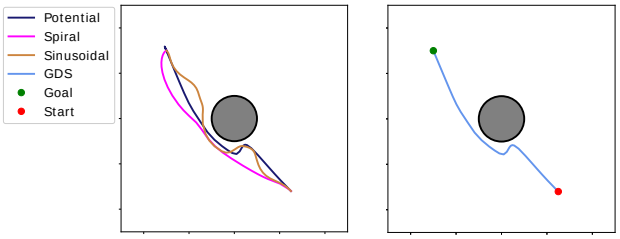
V. Experimental Results

In this section, we compare the proposed RMP-flow–CLF framework with the original RMP-flow framework [17]. A video of the experimental results can be found at https://youtu.be/eU_x8Yklv-4. The original RMP-flow framework [17] is referred to as RMP-flow–GDS to differentiate it from RMP-flow–CLF.

A. Simulation Results

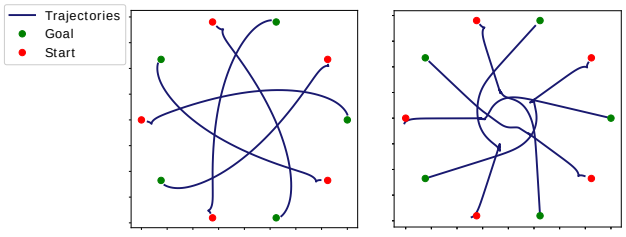
We present two simulated examples: a 2-dimensional goal reaching task and a multi-robot goal reaching example.

[3] This a linear constraint with respect to f_{l_k}. When $\dot{z}_k = 0$, the constraint (22) holds trivially because both sides equal 0.



(a) RMPflow–CLF (b) RMPflow–GDS [17]

Fig. 2: 2D goal reaching task with a circular obstacle (grey). (a) RMPflow–CLF with three choices of nominal controllers, resulting in different goal reaching behaviors. (b) RMPflow–GDS with the goal attractor given by a GDS. The behavior is limited by the choice of the metric and the potential function.



(a) RMPflow–CLF (b) RMPflow–GDS [17]

Fig. 3: Multi-robot goal reaching task. (a) RMPflow–CLF with spiral nominal controllers. The robots move to their goal smoothly. (b) RMPflow–GDS with the goal attractor given by a GDS. Due to the symmetry of the configuration, the system suffers from deadlock when the robots are near the center: the robots oscillate around the deadlock configuration.

1) 2D Goal Reaching: We first consider the 2D goal reaching task presented in [17]. In this example, a planar robot with double-integrator dynamics is expected to move to a goal position without colliding into a circular obstacle. As is in [17], the RMP-tree has a collision avoidance leaf-node RMP and a goal attractor leaf-node RMP. For the RMPflow–CLF framework, we use the collision avoidance RMP in [17] and keep the choice of metrics and potential functions for the goal attractor RMP consistent with [17]. For the goal attractor RMP, we present several nominal controllers: (i) a pure potential-based nominal controller $\mathbf{f}^d_{pt} = \mathbf{M}\mathbf{u}^d_{pt} = r$; (ii) a spiral nominal controller $\mathbf{f}^d_{sp} = r + k\underline{z}k\,\mathbf{v}$, where $\mathbf{v}$ is the potential-based controller rotated by $=2$, i.e. $\mathbf{v} = R(=2)\,r$ with $R(\,)$ being the rotation matrix; and (iii) a sinusoidal controller $\mathbf{f}^d_{sn} = r + \sin(t{=}4)\,k\underline{z}k\,\mathbf{v}$. For the minimally invasive controller, we use $\mathbf{P} = \mathbf{I}$ to minimize the Euclidean distance between the nominal controller and the solution to the optimization problem (23). We implement the RMPflow–GDS framework with the same choice of parameters as [17]. The trajectories under different nominal controllers are shown in Fig. 2. Although it is possible that similar behaviors can be realized with the RMPflow–GDS framework with a careful redesign of the metric and potential function, the RMPflow–CLF framework can produce a rich class of behaviors without being concerned with the geometric properties of the subtask manifold.

2) Multi-Robot Goal Reaching: RMPflow–CLF guarantees system stability even when the nominal controllers are not inherently stable or asymptotically stable. Therefore, the user can incorporate design knowledge given by, e.g. motion planners, human demonstrations or even heuristics, into the nominal controllers. To illustrate this, we consider a multi-robot goal reaching task, where the robots are tasked with moving to the opposite side of the arena without colliding. If the robots move in straight lines, their trajectories would intersect near the center of the arena. Due to the symmetric configuration, the system can easily deadlock with robots moving very slowly or stopping near the center to avoid collisions. This problem can be fixed if the symmetry is broken. One possible solution is to design nominal controllers for the goal attractors so that the robots move along curves.

We compare the spiral goal attractor RMP with the GDS goal attractor RMP presented in [18]. In both cases, an RMP-tree structure similar to the centralized RMP-tree structure in [18] is used. We define collision avoidance for pairs of robots in the RMP-tree with the same choice of parameters. The trajectories of the robots under the spiral nominal controllers are shown in Fig. 3a. The spiral controllers produce smooth motion, whereas the GDS goal attractors produce jerky motion when the robots are near the center due to deadlock caused by the symmetric configuration (Fig. 3b).

B. Robotic Implementation

We present an experiment conducted on the Robotarium [22], a remotely accessible swarm robotics platform. In the experiment, five robots are tasked with preserving a regular pentagon formation while the leader has an additional task of reaching a goal. We use the same RMP-tree structure and parameters for most leaf-node RMPs as described in the formation preservation experiment in [18]. The only difference is that we replace the GDS goal attractor in [18] with the spiral nominal controller augmented with the CLF condition (23). Fig. 4 presents the snapshots from the formation preservation experiment. In Fig. 4a–Fig. 4c, we see that the leader approaches the goal with a spiral trajectory specified by the nominal controller, while other subtask controllers preserve distances and avoid collision. This shows the efficacy of our controller synthesis framework. By contrast, the robot moves in straight lines under the goal attractor given by the GDS (see Fig. 4d-Fig. 4f). Although it could be possible to redesign the subtask manifold such that there exists a GDS that produces similar behaviors, the RMPflow–CLF framework provides the user additional flexibility to shape the behaviors without worrying about the geometric properties of the subtask manifolds.

VI. Conclusions

We consider robot control with multiple control specifications by adopting Riemannian Motion Policies (RMPs), a recent concept in robotics for describing control policies on manifolds, and RMPflow, the computational structure to combine these controllers. The stability results of RMPflow is re-established and extended through a rigorous