SYNTHESIZING MODULAR MANIPULATORS FOR TASKS WITH TIME, OBSTACLE, AND TORQUE CONSTRAINTS

Thais Campos
Sibley School of Mechanical and Aerospace Engineering
Cornell University
Ithaca, New York 14850
Email: tcd58@cornell.edu

Hadas Kress-Gazit
Sibley School of Mechanical and Aerospace Engineering
Cornell University
Ithaca, New York 14850
Email: hadaskg@cornell.edu

ABSTRACT

Modular robots can be tailored to achieve specific tasks and rearranged to achieve previously infeasible ones. The challenge is choosing an appropriate design from a large search space. In this work, we describe a framework that automatically synthesizes the design and controls for a serial chain modular manipulator given a task description. The task includes points to be reached in the 3D space, time constraints, a load to be sustained at the end-effector, and obstacles to be avoided in the environment. These specifications are encoded as a constrained optimization in the robot’s kinematics and dynamics and, if a solution is found, the formulation returns the specific design and controls to perform the task. Finally, we demonstrate our approach on a complex specification in which the robot navigates a constrained environment while holding an object.

1 Introduction

Figure 1: From task definition to execution. (A) Points to be reached with their times stamps, spherical obstacles in green, planar obstacles in yellow, constraint on the origin of the manipulator in blue; (B) Connection between the task points after RRT*; (C) Trajectory over time after a polynomial fit; (D) Simulation of the resultant manipulator; (E) Hardware demonstration.

*Corresponding author.
Typical manipulators can perform a set of tasks; however, they are limited by their fixed structure and actuators specifications. Modular robots, on the other hand, can be designed to achieve a specific task and adjusted to handle tasks that otherwise would be infeasible, making them more versatile and easier to repurpose. This is only possible because modular robots consist of individual components that can be reorganized or replaced to create different designs, each with its own functionality. However, manually selecting a manipulator’s structure and its controls can be a difficult and error-prone process, as it involves analyzing kinematic and dynamic equations as well as deciding on discrete and continuous properties, such as number of degrees-of-freedom (DOF) and link lengths.

In this work, we automate the design process for modular manipulators: we introduce an end-to-end system that automatically synthesizes both structure and controls of modular manipulators from high-level task specifications. Here, the task (depicted in Fig. 1-(A)) is a sequence of points in 3D space that the manipulator’s end-effector (EF) must reach at specific time instants while sustaining a load applied to the EF, a set of planar and spherical obstacles, and constraints on the position of the base, depicted as yellow rectangles, green spheres, and a blue rectangular region, respectively. Our system outputs the robot’s structure, which is a serial chain manipulator with a fixed base composed of links sequentially connected by one DOF modular actuators, as seen in Fig. 1-(E), where the red components are HEBI X-series actuators [1]. It also outputs the actuator control commands for the robot to be able to perform the task.

In our approach, the synthesis problem is divided into three steps: trajectory selection, constrained optimization and verification. In the trajectory selection step, we first use the sampling-based motion planner RRT* [2] to select a path, defined as a sequence of waypoints, in 3D space connecting all the points in the original task (Fig. 1-(B)). Then, we construct a trajectory from the selected path using polynomial interpolation (Fig. 1-(C)). The trajectory defines the position of the robot’s end-effector over time, thus specifying its velocities and accelerations. In the second step, we solve a constrained optimization problem in the robot’s kinematics and dynamics, so the manipulator can reach the points on the path individually at their respective time instants. Finally, in the verification step, we check if the previously found robot design is able to follow the trajectory while avoiding obstacles and satisfying the dynamic constraints. If our approach is able to find a solution, then it is guaranteed to be correct. However, the approach is not complete; there might exist solutions for a given task specification that our algorithm was not able to find in the allotted time.

Related work: In [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23], the authors analyzed kinematic requirements, such as dexterity [4, 6], manipulability [10] and maximization of reachable space [5], in [11, 12, 23, 13, 15, 16, 17, 18, 19, 20, 19, 21, 22, 14, 20], the authors also incorporated torque specifications in their formulation.

In [15], the authors used a priority rating system in the robot’s metric specifications, in which high priority items were more likely to be satisfied by the selected robot structure. The metrics included task completion, average torque applied in the actuators and completion time, ranked from most important to least. Given the prioritization formulation and the average torque calculation, the actuators torque limits might not be satisfied for the total duration of the task.

In [18, 16, 17, 19], the authors included static torque requirements in their approach. The total torque applied to the robot’s actuator is composed of static and dynamic components. While the static torque is the one required for the manipulator to hold a position against gravity, the dynamic torque refers to the one necessary for the manipulator to perform a pre-determined motion. If for a given task, such as motion from point A to point B, the robot does not need to follow a schedule, then it can perform this motion very slowly, and the torques required in each actuator are close to the static ones. However, some tasks require the robot to reach points at certain time instants, for example when coordinating with other robots, or when moving payloads. In these cases, the robot motion must adhere to specific velocity and acceleration profiles while respecting torque limits. Thus, the dynamic component of the torque must be considered as well. In our approach, by specifying the EF trajectory before attempting to find a manipulator structure, we include dynamics constraints in our formulation guaranteeing that, when applying the controls to the selected design, it will reach the task points at the times specified while respecting the actuators torque limits.

Both static and dynamic components of the torque were incorporated in [13, 12, 14]. However, the authors only examined environments without obstacles, whose presence restricts the robot’s collision-free reachable space. This restriction causes two challenges not addressed in their formulation: first, the task might require a more complex robot with more DOFs in order to avoid collisions; second, the path generated might be longer due to the obstacles, therefore to maintain the original timing, the robot might require higher torques than are supported by its actuator.

Similar to our task definition, kinematic and dynamic specifications are also shown in [23, 21, 20, 22], where the total torque is considered when finding a design for a manipulator in an environment with obstacles. However, in [23, 21, 20], the authors fix the number of actuators in the manipulator design beforehand and only allow conventional values of twist angle. These restrictions create a smaller search space that might lead to no solutions in a highly
constrained environment. Moreover, this approach might select complex designs with a large number of modules when a simpler solution is sufficient and preferred. In [22] and in our approach, these restrictions are not imposed, creating structures more suitable and less complex for the task at hand. However, in [22], the authors do not consider the robot’s final trajectory in their formulation, but only the task points. When considering only certain poses for the robot in a constrained environment, the synthesis process might create structures with disconnected reachable space, as seen in [3], resulting in an infeasible task. In our approach, by simultaneously searching for a robot’s design and a path that connects the task points, we guarantee that the solution is able to complete the task while satisfying all kinematics and dynamics constraints.

The main contributions of this paper are: (i) a framework for automated design of a manipulator structure and its controls for a task with specific timing and workspace constraints, that (ii) guarantees that the actuators do not exceed torque constraints while executing the task. We demonstrate our approach with HEBI actuators for a physical demonstration involving a complex task.

2 Definitions

The notation $a,b$ refers to a parameter $b$ of variable $a$. For example, $o.rad$ refers to the radius $rad$ of a spherical obstacle $o$. Continuous and discrete intervals are denoted as $[a,b]$ and $\{a_1,\ldots,b\}$, respectively. Line segment, vector, and vector’s length whose endpoints are $a,v$ are denoted as $\overrightarrow{av}$, $\overrightarrow{uv}$, and $||\overrightarrow{uv}||$, respectively. The minimum distance between two curves $h_1$ and $h_2$ or a curve $h_1$ and a point $p$ is referred to as $||h_1,h_2||$ and $||h_1,p||$, respectively.

2.1 Task

The workspace of the manipulator may include a set of spherical and planar obstacles. The set of $n_{sph}$ spherical obstacles is represented by $Sph = \{o_1,\ldots,o_{n_{sph}}\}$, where $o$ is defined by its center $o.c \in \mathbb{R}^3$ and radius $o.rad \in \mathbb{R}^+$. The planar obstacles are rectangles defined by their vertices’ coordinates and represented by $Walls = \{w_1,\ldots,w_{n_{walls}}\}$, with a total of $n_{walls}$ obstacles. Fig. 1(A) shows the spherical obstacles in green and the planar ones in yellow.

The set of points that the robot’s $EF$ must reach at their specific time instants while sustaining an external load is defined by $P = \{(p_1, t_1, F_1), \ldots, (p_n, t_n, F_n)\}$, where $(p_i, t_i, F_i)$ is a tuple in which $i \in \{1,\ldots,n\}, n$ is the total number of task points, $p_i \in \mathbb{R}^3$ is the task point whose coordinates are defined with respect to the global frame, $t_i \in \mathbb{R}^+$ is the time stamp, and $F_i \in \mathbb{R}^6$ is the vector containing the forces $[f_x, f_y, f_z]$ and moments $[M_x, M_y, M_z]$ acting on the $EF$. The first task point $p_1$ defines the start of the task; thus, $t_1 = 0$. Fig. 1(A) shows the task points in black dots with their respective time stamps.

The set $B$ captures the possible positions of the origin $O_0$ of the manipulator. $B$ is parallel to the x-y plane and is specified by its diagonal coordinates, $(B.x_{min},B.y_{min}), (B.x_{max},B.y_{max})$, and height $B.z$. Fig. 1(A) shows the constraints in the origin as a rectangular blue region.

Here we define the task as $T = (P, Sph, Walls, B)$.

2.2 Robot Structure

We design the robot as a serial open-chain manipulator composed of modules. A module $m_k$, where $k \in \{1,\ldots,n_{DOF}\}$ and $n_{DOF}$ is the total number of modules, is composed of an actuator and a link attached to it (see Fig. 2). The actuator provides one rotational degree-of-freedom (DOF) to the structure, while the link is a rigid tube with fixed length $m_k.r$, or $r_k$ (we contract any $m_k.var$ variable to $var_k$), and diameter $\Delta$ that connects its module to the next actuator or $EF$.

We attach a reference frame to each $m_k$, with origin $O_{k-1}$, $z_{k-1}$-axis in the direction of the actuator’s rotation, and $x_{k-1}$ and $y_{k-1}$-axis directions defined according to the Denavit-Hartenberg (DH) convention [24]. Since the joints are revolute, the configuration of $m_k$ is defined by the joint angle $\theta_k$. The other DH parameters are link offset $o.k$, which is the offset along $z_{k-1}$-axis between $O_{k-1}$ and $O_k$, defined by the actuators dimensions, and the link twist $\alpha_k$, which is the angle between the rotation axis of two consecutive actuators. $z_0$ is constrained to be parallel to the global z-axis $z_G$, for easier construction of the physical robot.

We define a structure $D$ by $O_0$, the total number of modules $n_{DOF}$, the DH parameters $\alpha_k \in [0, 2\pi]$ and $r_k \in [r_0, r_{max}]$ $\forall k \in \{1,\ldots,n_{DOF}\}$, where $r_0$ is the minimum length and $r_{max}$ the maximum. A configuration $q_i$ of the manipulator is the list of joint angles $q_i = \{\theta_1,\ldots,\theta_{n_{DOF}}\}$, that uniquely defines the position of all points in the system such that $EF$ reaches $p_i$. Applying the forward kinematics $(FK)$ equations in $q_i$ returns the state of the manipulator $FK(D,q_i) = S_i = \{S_0, S_1,\ldots,S_{2n_{DOF}}\}$. $S_i$ is the global coordinates of the $2n_{DOF}$ points that define the endpoints of all the links in $D$ plus the origin $O_0 \equiv s_0$, where $s_{2n_{DOF}} = EF$ as seen in Fig. 3. Two consecutive points in $S_i$
form a line segment that defines the spatial position of either an actuator or a link. For example, as seen in Fig. 2, the actuator component of \( m_1 \) is represented by \( s_{0s_1} \), or \( \text{act}_1 \). Similarly, the first link can be represented by \( s_{s_1s_2} \), or \( \text{link}_1 \).

To clarify our symbol definition, \( S_i \), \( \text{EF} \) is the end-effector position given that the manipulator state is \( S_i \).

2.3 Controls

We define the controls to be applied to \( D \) such that \( \text{EF} \) reaches \( p_i \) at time \( t_i \) while sustaining external load \( F_i, i \in \{1, \ldots, n\} \) as \( \Theta_i = (q_i, \dot{q}_i, T_i) \), where \( q_i = \{\theta_1, \ldots, \theta_{\text{nDOF}}\} \), and \( T_i = \{\tau_1, \ldots, \tau_{\text{nDOF}}\} \), are the velocities and torques, respectively, applied to each actuator.

Figure 2: Scheme of a 2 DOF modular manipulator. The red cylinders represent the actuators while the black ones, the links.

2.4 Problem Statement

In this work, we automatically attempt to find a structure \( D \) and a sequence of controls \( C = \{\Theta_1, \ldots, \Theta_n\} \) such that a robotic manipulator is able to perform task \( T \) while satisfying kinematic and dynamic constraints.

3 Approach

Fig. 3 shows our overall approach. We start by using a sampling-based motion planner, RRT* (Section 3.1) to find a collision-free path for the \( \text{EF} \) in the form of waypoints. Then, through polynomial interpolation, we construct a trajectory for the robot’s end-effector to follow (Section 3.2). This step is essential because we extract, from the trajectory, \( \text{EF} \) velocities and accelerations that are needed for calculating the torques applied to the actuators.

Since the existence and quality of constrained optimization solutions is highly dependent on the initial guess, we solve a simpler version of the problem with only kinematic requirements (Section 3.3) to generate a suitable initial guess for the full constrained optimization (Section 3.6). We then check if the solution provided by the kinematic optimization is able to satisfy the task while meeting all kinematic (Section 3.3) and dynamic (Section 3.4) constraints. If it is, we proceed to the verification step (Section 3.7). Otherwise, we continue with the full constrained optimization. In the verification step, we certify that the candidate design is not only able to reach the task points but also able to follow the previously defined trajectory while complying with the torque requirements and avoiding collisions.

Finally, if the verification is successful, the process is finished and the solution returned, otherwise, we restart the process either from the kinematic optimization using a randomly selected initial guess or from the RRT*. This restart process might be repeated until the maximum number of iterations for each step, specified by the user, is reached.

To decide on the number of actuators, we iterate over a finite set of possible \( n_{\text{DOF}} \), starting from the smallest value and adding DOFs as needed, similar to what we did in [3, 25].

3.1 RRT*

We start our approach with the sampling-based motion planner RRT* \([2]\) to select an \( \text{EF} \) path that passes through all of the task points. RRT* is asymptotically optimal, meaning that as the number of samples grows, the path will approach the optimal Shortest2020 path. This is important for our problem since longer trajectories between two task points require higher actuator velocities and torques, which could potentially saturate the actuators.

The output of the RRT*, if successful, is a list \( p^{\text{RRT*}} \) of \( n^* \) points in the environment, \( n^* \geq n \), that when consecutively connected by line segments, produces a collision-free path for the \( \text{EF} \) in Cartesian space, as shown in red in Fig. 1(B).
3.2 Polynomial Fitting

For each new point in $p^\text{RRT}^*$, we assign a time instant based on its distance to the original task points, and an external load to be sustained by the $EF$ equals to the previous load. For example, if a new point found by the RRT* is halfway between $p_i$ and $p_{i+1}$, then it is assigned the time instant $(t_i + t_{i+1})/2$, and the load $F_i$. This process constructs a new list, $P^*$, $P \subset P^*$. From $P^*$, we construct a trajectory that is smooth and guarantees the actuators’ velocity and acceleration continuity by interpolating piecewise polynomials over time for each coordinate, creating the trajectory $\text{Traj} = \langle x(t), y(t), z(t) \rangle$ (see Fig. 1-(C)). We enforce the following boundary conditions: (i) the first and second derivatives of two consecutive polynomials must be equal at the waypoint connecting them, and (ii) at the beginning ($t_1$) and at the end ($t_n$) of the motion, the derivatives of the polynomials are zero, so that the $EF$ starts and ends at rest.

When interpolating the polynomials, there are no guarantees that they do not collide with obstacles, therefore before following to the next step of the approach, we certify that $\text{Traj}$ is collision-free. We discretize it finely and check if any point on $\text{Traj}$ is inside a spherical obstacle or any two consecutive points on $\text{Traj}$ are on opposite sides of any planar obstacle. If there are collisions, we rerun the RRT* for a different path.

3.3 Kinematic Requirements

For the kinematic constrained optimization problem, we enforce the following constraints:

3.3.1 Reachability

The distance between the manipulator’s $EF$ when at state $S_i$ and the point $p_i$ is shown in Eq. 1. For the manipulator to successfully reach $p_i$, $h_i^R = 0$.

$$\forall p_i \in p^\text{RRT}^*, h_i^R = ||S_i, EF, p_i||$$  (1)

3.3.2 Collision Avoidance

Eq. 2 calculates the minimum distance between the spherical obstacles and the robot’s components (links and actuators), where $\Delta$ is the components’ thickness.

$$\forall i \in \{1, ..., n^*\}, \forall o \in Sph, \forall k \in \{1, ..., n_{DOF}\},
\quad g_{i,k}^S = \min_{a,k} \{||S_i, act_k, o.c|| - (o.rad + \Delta),
\quad ||S_i, link_k, o.c|| - (o.rad + \Delta)\}$$  (2)

Similarly, Eq. 3 detects if there is collision between any planar obstacles and the robot’s components, where the function Intersect returns True (or 1) if two points are on opposite sides of any wall and False (or 0) if they are not.

$$\forall i \in \{1, ..., n^*\}, \forall w \in Walls, \forall k \in \{1, ..., n_{DOF}\},
\quad h_i^W = \max_{w,k} \{\text{Intersect}(S_i, act_k, w), \text{Intersect}(S_i, link_k, w)\}$$  (3)

Finally, Eq. 4 refers to self-collision avoidance; it calculates the minimum distance between two non-consecutive links/modules minus their thickness.

Figure 3: Overview of the approach.
∀i ∈ {1,...,n∗}, ∀k ∈ {1,...,nDOF},
∀k′ ∈ {k + 1,...,2nDOF − 1},
g^{SC}_i = \min_{k,k′}(\|S_i{s^k_k{k−1}}s_k{s_{k−1}k+1}\| − 2\Delta) \tag{4}

For the robot to successfully avoid any collisions when reaching \(p_i\), we require \(g^S_i ≥ 0, h^W_i = 0\), and \(g^{SC}_i ≥ 0\).

3.3.3 Origin

The manipulator’s origin position is bounded by \(B\), as shown in Eq. 5

\[B.x_{\text{min}} ≤ O_0.x ≤ B.x_{\text{max}}, B.y_{\text{min}} ≤ O_0.y ≤ B.y_{\text{max}}, \]
\[O_0.z = B.z\]  \tag{5}

3.3.4 Link Length

Eq. 6 enforces that the previous link length is always equal to or smaller than the next. This constraint might contradict
the usual manipulator design, in which longer links are positioned closer to the base together with heavier and more powerful actuators. However, here we are considering identical modules. Smaller link lengths closer to the base reduces
the distance from the actuators closer to the \(EF\) to the actuators closer to the base. Since the actuators are the heaviest
components (except, possibly, for the load), reducing the link lengths close to the base reduces the torque required there.
Moreover, shorter arms contribute for better accuracy and stability and, thus, improving the robot’s behavior. If the
modules are not identical, this constraint can be removed.

\[
\forall k ∈ \{1,...,nDOF\}, r_{k−1} ≤ r_k ≤ r_{\text{max}} \tag{6}
\]

3.4 Dynamic Requirements

We use the Recursive Newton-Euler Algorithm (RNEA) [26] to calculate the joint torques required for a serial
manipulator to perform a trajectory. Next, we present how we compute the necessary parameters to perform the RNEA.

3.4.1 EF Velocity and Acceleration

We calculate the velocity \(v_i\) and acceleration \(a_i\) at the \(EF\), with respect to the base frame (located at \(O_0\)), required
when reaching \(p_i \in P^∗\) by taking the first and second derivative of \(Traj\) and evaluating them at time stamp \(t_i\),
\(v_i = [\dot{x}(t_i); \dot{y}(t_i); \dot{z}(t_i)]\) and \(a_i = [\ddot{x}(t_i); \ddot{y}(t_i); \ddot{z}(t_i)]\). Since we do not control the orientation of the \(EF\), we only
consider the linear components of the robot’s spatial velocity. Therefore, we eliminated the angular velocity (usually,
referred to as \(ω_x, ω_y, ω_z\)) and removed the corresponding rows in the jacobian matrix.

3.4.2 Actuators’ Velocity and Acceleration

The relation between \(v_i, a_i, \dot{q}_i\), and \(\ddot{q}_i\) is described by the manipulator’s geometric Jacobian \(J_i \in \mathbb{R}^{3 \times nDOF}\) [27], and
shown in Eqs. 7 and 8, where \(J_i^{-1}\) is the pseudo-inverse matrix of \(J_i\).

\[
\dot{q}_i = J_i^{-1} v_i \tag{7}
\]

Taking the time derivative of Eq. 7 yields Eq. 8

\[
\ddot{q}_i = J_i^{-1} (a_i − J_i \dot{q}_i) \tag{8}
\]

3.4.3 Inertia Tensor

The module \(m_k, k \in \{1,...,nDOF\}\), is a compound object composed of an actuator and a link. To calculate its inertia
tensor \(I_k \in \mathbb{R}^{3 \times 3}\), we model the actuator as a solid uniform cylinder whose mass, height, and radius are approximated
by the measurements provided by the manufacturer. The link is a hollow cylinder with length \(r_k\), density \(ρ\), and external
and internal radius \(rad_2\) and \(rad_1\), respectively. \(I_k\) is, then, the sum of the actuator and the link inertia tensors calculated
with respect to the frame attached to \(m_k\), whose position and axis are defined following DH convention (Section 2.2).
3.4.4 RNEA

The RNEA has two steps: a forward and a backward recursion. In the forward recursion, we progressively calculate the modules’ velocities and accelerations with respect to the base frame, using the previously calculated parameters (actuators poses, velocities, and accelerations), as well as the velocity and acceleration of the base frame, which are referred to as initial conditions. In our formulation, we consider that the base frame is static.

Then, in the backward step, we use the results from the forward recursion, the modules’ inertia tensors, and the force and moment exerted at \( EF \) \((F_i)\) to recursively compute the forces and torques required for each actuator. For a detailed explanation of the algorithm, refer to [27]. Finally, with the torques required for all actuators at time stamps \( t_i, \forall i \in \{1, ..., n^*\} \), we define Eq. [9] which calculates the minimum difference between the maximum torque \( \tau_{k,i}^{max} \) allowed and the torque required in the actuators at specific time instants \( \tau_{k,i} \). If \( g_i^k \geq 0 \), then the actuators satisfy the torque limits. In Eq. [9] we add \( \beta > 1 \) as a safety factor used to protect the actuators from saturation. From our experience with the physical modules, nonlinearities such as overshoot and sample time may cause slight deviations from the desired trajectories. Therefore we add \( \beta \) to create structures and controls that are more constrained and thus more robust to such unmodeled physical phenomena.

\[
\forall i \in \{1, ..., n^*\}, \forall k \in \{1, ..., n_{DOF}\}, \\
g_i^k = \min_k (\tau_{k,i}^{max} - |\beta \tau_{k,i}|) 
\]  \( \text{(9)} \)

3.5 Kinematics Constrained Optimization

The kinematics constrained optimization is defined in Eq. [10]. The cost function \( f^{kin} \) includes collision avoidance objectives (Eqs. [2][4][14] and the constraints include reaching the task points, origin placement, and link lengths constraints (Eqs. [1][5][6]). Since we view the kinematic optimization as providing an initial solution, we encode the collision avoidance as soft constraints by placing them in the cost function. This makes the optimization problem easier to solve. This might result in solutions that are not collision free; however, the design returned might still be able to reach the task points using different configurations that are collision free. We decided to maintain the reachability requirement as a hard constraint so that the structure is guaranteed to reach the task points. Given a solution, we first check if any configuration returned collides with an obstacle. If it does, we then search for a new configuration that is collision-free using the inverse kinematics (IK) equations. If one is found, we replace it in the solution.

While one can choose to add weights to the different components in the cost function, we set all weights equal to \( \pm 1 \) because they are all in comparable ranges and have the same importance (a solution with self collisions is as inadequate as one that collides with an obstacle).

\[
\min f^{kin} = \sum_{i=1}^{n^*} (-g_i^S + h_i^W - g_i^{SC}), \\
\text{Subject to: } h_i^R = 0, \text{ and Eqs. } [5][6] 
\]  \( \text{(10)} \)

For some tasks, the kinematic optimization is enough to yield a solution that also satisfies the dynamic requirements. For example, for tasks in which there are no loads applied in the \( EF \) and the robot design is compact (small link lengths and small number of DOFs), the torques required for the actuators may be small. Therefore, we first check whether the solution provided by this optimization satisfies both kinematic and dynamic requirements by calculating if \( \forall i \in \{1, ..., n^*\}, h_i^R = 0, h_i^W = 0, g_i^S \geq 0, g_i^{SC} \geq 0, \) and \( g_i^T \geq 0 \). If it does, we continue to the verification step; otherwise, this solution is used as initial guess for the full constrained optimization.

3.6 Constrained Optimization

The full constrained optimization guarantees that the design selected reaches all task points while avoiding collisions and satisfying torque requirements, as shown in Eq. [11]. We use a different cost function than in Eq. [10] in order to modify solutions returned by the kinematic optimization which might not satisfy the dynamic requirements.

\[
\min f' = \sum_{i=1}^{n^*} -g_i^T, \\
\text{Subject to: } h_i^R = 0, h_i^W = 0, \\
g_i^S \geq 0, g_i^{SC} \geq 0, g_i^T \geq 0, \text{ and Eqs. } [5][6] 
\]  \( \text{(11)} \)
3.7 Verification

The output of the optimization is a design \( D \) and its controls \( \theta_i, \forall i \in \{1, ..., n^*\} \), required to reach the points on the path at the time instants in \( P^* \). In the verification step, we check if \( D \) can not only reach these points but can also follow \( T_{ra}j \) while satisfying all the constraints. For that, we first further discretize \( T_{ra}j \) into \( n^\dagger > n^* \) points \( p_i^\dagger \). Then, we solve the inverse kinematics (IK) for each point \( p_i^\dagger, \forall i^\dagger \in \{1, ..., n^\dagger\} \), to find the joint angles such that \( EF \) reaches the desired point while avoiding collisions. If the IK provides a solution, we check if the dynamic constraints are satisfied and if the continuity of the motion is maintained by applying Eq. 12, where \( q_{i^\dagger} \) and \( q_{i^\dagger+1} \) are consecutive configurations and \( \epsilon \) is a small threshold. The continuity constraint is necessary because there might exist several solutions for the IK problem and we are looking for solutions in which \( q_{i^\dagger+1} \) is similar to \( q_{i^\dagger} \) to ensure that the robot’s \( EF \) does not deviate from the trajectory. The value we use for \( \epsilon \) is \( \max(\dot{q}_{i^\dagger}, \dot{q}_{i^\dagger+1})dt \), where \( dt \) is the time interval and \( \dot{q}_{i^\dagger} \) and \( \dot{q}_{i^\dagger+1} \) are calculated from Eq. 7, since it constraints the maximum possible distance (in radians) traveled by the actuator during that interval.

\[
\forall i^\dagger \in \{1, ..., n^\dagger\}, g_{i^\dagger} \geq 0 \land |q_{i^\dagger} - q_{i^\dagger+1}| \leq \epsilon
\]  

4 Results

Figure 4: Tasks descriptions used for comparison. The green spheres are the obstacles and the black dots the task points. The origin of the manipulator is constrained to be at the origin of the global frame.

Figure 5: Three different task definitions (i-A to i-C), and the corresponding returned manipulator design (ii-A to ii-C).

We implemented our approach in MATLAB R2020b and solved the constrained optimization problem using Sequential Quadratic Programming (SQP) with the \textit{fmincon} function. We used the RNEA algorithm from The Robotics Toolbox for MATLAB (RTB) [28]. To calculate the distances between line segments and the collision check between line segment and planar obstacles, we used the algorithms in [29] and [30], respectively. We implemented the RRT* algorithm based on the one in [31].

For all the simulations presented in this section, our approach took a few hours (from around 2 for tasks in Fig. 5 to 5 for task in Fig. 7) to return a solution. Since we do not fix the number of DOFs beforehand, our approach might take a
Table 1: Comparison of the number of solutions obtained by our approach and by the kinematic-only approach for the tasks shown in Fig. 4.

| DOF | Example A # Success (Fig. 4-A) | Example B # Success (Fig. 4-B) |
|-----|--------------------------------|--------------------------------|
|     | Kinematics req. opt. - Trajectory search | Our approach | Kinematics req. opt. - Trajectory search | Our approach |
| 2   | 22 - 0                          | 0                | 0 - -                                    | 0             |
| 3   | 43 - 33                         | 8                | 39 - 26                                  | 8             |
| 4   | 61 - 47                         | 65               | 44 - 35                                  | 42            |

longer time to return a solution compared to approaches in which the number of components is fixed, as in \([23, 21, 32]\). However, we might return less complex and more compact structures that are easier to construct and control.

4.1 Comparison With Other Approaches

To enable the joint synthesis of design and control from tasks with dynamic constraints, we must create a trajectory over time for the robot’s end-effector to follow before choosing a robot design. Moreover, the candidate design needs to be able to follow the entire trajectory (checked in the Verification step), for it to be a valid solution. On the other hand, approaches that only consider kinematic or static dynamic requirements might find a trajectory for the robot to follow after deciding on a design, by applying an off-the-shelf motion planner.

In this section, we compare our formulation to our prior work with only kinematic requirements \([3]\). Here, we are interested in comparing the total number of solutions provided by each approach for the same task specification, and the running time. In order to make this analysis more adequate, we eliminate the torque constraints from our formulation. Thus, we are only evaluating how constructing the EF trajectory before (this paper) or after (prior work) choosing a robot design affects the results.

For the comparison, we implemented a kinematic-only approach adapted from our own kinematic constrained optimization in Section 3.3 and inspired by \([3]\). After a solution is found by the optimization, we run the RRT* in joint space to find a collision-free trajectory for the manipulator. The task descriptions used for comparison are shown in Fig. 4 and extracted from \([3]\), where the kinematic-only approach synthesized a manipulator’s structure with a disconnected workspace, making it impossible for the RRT* to find a trajectory for the robot to follow.

The strategy used in \([3]\) to produce a robot design in such cases was to ask for human input. The approach required the user to add a new task point, forcing the constrained optimization to select a design whose workspace was fully connected. In our formulation, by including the sequence of waypoints returned by the RRT* into our optimization, we can select robots that are more likely to have a fully connected workspace, not requiring user involvement. Here, we do not request user input for any of the approaches.

We ran both approaches 100 times for each task description shown in Fig. 4 and for each number of actuators in \(n_{DOF} = \{2, 3, 4\}\). The initial guesses were randomly selected but identical for both approaches.

The number of solutions provided by each formulation is shown in Table 1. For both tasks A and B, for a \(n_{DOF} = 2\), the kinematic-only optimization is able to synthesize a design (22 of 100) for task A. However, the RRT* was unable to find a trajectory in the allotted time which indicates that the workspace may be disconnected. This corroborates the findings in \([3]\). Our approach is also unable to find a solution for \(n_{DOF} = 2\), which is expected since a 2 DOF robot is underactuated and, thus, unlikely to be able to follow an arbitrary trajectory like the ones returned by RRT*. As \(n_{DOF}\) increases, our approach is able to find more solutions than the kinematic-only approach because our optimization returns more designs with a connected workspace that are able to follow the required trajectories.

Our approach takes on average 2.6x longer to provide a solution (19.05s compared to 7.30s) for the tasks studied. This is expected, since we include all \(n^* \geq n\) waypoints in \(p^{RRT*}\) in the optimization, thus creating more constraints.

4.2 Tasks and Designs

In this section we show how a similar task with slight modifications results in different designs, showcasing the advantage of using such design automation techniques for a variety of tasks.

We examined three slightly different task descriptions, shown in Fig. 5. The manipulator’s goal is to reach a point above the middle shelf inside the cabinet at \(t_2 = 2s\), grab a payload of 33g and carry it to the bottom shelf without colliding with any wall, shelf or spherical obstacle. The task ends at \(t_3 = 6s\). For \(n_{DOF} = 2\), we require \(r_{max} = 0.6m\), for
Synthesizing Modular Manipulators For Tasks With Time, Obstacle, And Torque Constraints

Figure 6: Simulation (A) and physical robot (B) synthesized from the task shown in Fig. 5-i-(C).

$n_{DOF} = 3$, $r_{max} = 0.4m$, and for $n_{DOF} = 4$, $r_{max} = 0.3m$, thus maintaining a maximum span of 1.2m, regardless of the number of actuators. In the task presented in Fig. 5-i-(A), the cabinet does not have the middle shelf, which was added in task i-(B). In task i-(C), the task point to be reached at $t_2 = 2s$ is moved to be behind one of the spherical obstacles. Even with a similar set of tasks, our approach was able to synthesize a diverse set of designs as seen in Fig. 5-ii-(A) to (C). For task i-(A), the environment is less constrained without the middle shelf, allowing a robot with $n_{DOF} = 3$ to fulfill the task; however for tasks containing the added obstacle, i-(B) and i-(C), an additional DOF is needed. Moreover, for task i-(C), the second point to be reached is more difficult to reach when compared to task i-(B). Thus, the resultant manipulator needs a longer span, as seen in Fig. 5-ii-(B) and ii-(C).

4.3 Simulation and Demonstration

Figure 7: Simulation results for a task in a highly constrained environment (A) at the beginning, (B) middle, and (C) end of the task. The original task points (black dots) and the timings are also shown.

To show that our approach is able to handle highly constrained environments and produce solutions with varied DOFs, we specified the task shown in Fig. 7, in which the manipulator must travel through narrow passages, while reaching certain points at specific times (see Fig. 7(A)). Given the constrained environment, the resultant manipulator is redundant, having 6 DOFs, which gives it the required dexterity to navigate between the walls.

In order to demonstrate the feasibility of the physical implementation of our approach, we implemented the solution found for the task in Fig. 7-i-(C). The constructed robot is shown in Fig. 6-(B). The links are aluminum tubes with 31.75 mm diameter and length according to the synthesis results. Mounting brackets manufactured by HEBI Robotics (black devices showed in Fig. 6-(B)) fix the actuators to the links. Fig. 6-(A) also shows the simulated robot. We used an open-loop controller commanding simultaneous position, velocity and torque control for the physical robot and a magnet as the end-effector to attract and secure the payload (a fork) at $t_2 = 2s$. Fig. 8 shows the absolute value of the torque $\tau_2$ calculated and commanded to the module $k = 2$, as well as $|\beta \tau_k|$ with $\beta = 2$, and the torque sensed at the actuator during the whole trajectory. We chose to plot the torques of this actuator because it endures a higher applied torque than the others as it carries more load against gravity. We considered $\tau_{max} = 8N/m$. As seen in Fig. 8, the torque sensed in the actuator (orange curve) is similar to the torque commanded (dashed black curve), even though it shows some jittering possibly due to the nonlinearities inherent in physical systems, such as overshoot and hysteresis. Due to these discrepancies, we can see that around $t = 4.5s$, when the torque assume small values, the sensed torque
Surpass $|\beta r_2|$ by a negligible amount showing the need of the safety factor for more robust structure and controls. A video accompanying this paper shows the manipulator performing the task.

Figure 8: Absolute value of torque $\tau_2$ calculated and commanded to module $k = 2$ (dashed black curve), as well as, $|\beta r_2|$ with $\beta = 2$ (black curve), and absolute value of torque sensed in the actuator during the duration of the task (orange). For the satisfaction of the torque constraint, the curve $|\beta r_2|$ needs to be below the $\tau_{\text{max}}$, which is assumed to be 8 N/m (black thick line).

5 Conclusions and Future Work

Summary. In this work, we presented a framework for automated synthesis of a serial chain modular manipulators’ design and controls based on a task, introducing an end-to-end method from task description to physical implementation. Our task formulation includes a set of points in 3D space that must be reached at predefined time instants while the robot’s end-effector sustains a specified load. We demonstrated that our approach synthesizes structures tailored to the task at hand and are, therefore, less complex and more compact than commercial prefabricated manipulators that contain a fixed number of DOFs. Finally, we show the feasibility of our solutions by physically implementing a manipulator whose task involved securing an object and traveling inside a cabinet, a highly constrained environment, while complying with the actuators’ torque limits.

Practical considerations: While the physical robot successfully achieved the desired task, there might be differences between the synthesized trajectory and the performed one. This is due to several factors. First, small errors in the robot construction and placement modifies its kinematics and dynamics and, therefore, its behavior. Second, loss of accuracy characteristic of open-loop controllers (like the one used here) introduces small errors in the system that builds up with time. Checking the robot’s design parameters after construction and introducing a feedback loop, with integral and derivative terms into the controller might reduce these discrepancies and improve the robots’ behavior.

Future Work. There are a few directions we would like to explore in future work. First, if our approach provides a solution, the physical robot might only accomplish the task if its manufacturing errors, and its actuators’ accuracy are within an acceptable range. Thus, we will explore techniques for the robot’s self-verification, as in [23], so it can check if its design parameters are similar enough to the ones from the nominal results. If they are different, it will recalculate the controls, if possible, so the task is still feasible. Second, our formulation assumes the robot structure to be a serial chain with fixed origin, which constrains the family of functions it can perform. We will extend our
work to include more complex structures, such as parallel manipulators and robots that combine both locomotion and
manipulation. Finally, we will include feedback to the user for infeasible tasks, as in [3]. For example, this feedback
could be suggestions to eliminate a task point or to modify some specifications, such as to increase the time instants to
allow for smaller torques applied to the actuators.

Acknowledgments

This work was funded by NSF CNS-1837506.

References

[1] Hebi robotics. Jun. 17, 2021. [Online].
[2] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. The international
journal of robotic research, 30(7):846–894, 2011.
[3] Thais Campos, Jeevana Priya Inala, Armando Solar-Lezama, and Hadas Kress-Gazit. Task-based design of ad-hoc
modular manipulators. In 2019 International Conference on Robotics and Automation (ICRA), pages 6058–6064. IEEE, 2019.
[4] EJ Van Henten, DA Van’t Slot, CWJ Hol, and LG Van Willigenburg. Optimal manipulator design for a cucumber
harvesting robot. Computers and electronics in agriculture, 65(2):247–257, 2009.
[5] Cenk Baykal and Ron Alterovitz. Asymptotically optimal design of piecewise cylindrical robots using motion
planning. In Robotics: Science and Systems, 2017.
[6] Olivier Chocron. Evolutionary design of modular robotic arms. Robotica, 26(3):323–330, 2008.
[7] Guilin Yang and I-Ming Chen. Task-based optimization of modular robot configurations: minimized degree-of-
freedom approach. Mechanism and machine theory, 35(4):517–540, 2000.
[8] I-Ming Chen and Joel W Burdick. Determining task optimal modular robot assembly configurations. In
proceedings of 1995 IEEE International Conference on Robotics and Automation, volume 1, pages 132–137. IEEE, 1995.
[9] NA Stravopodis and VC Moulianitis. Rectilinear tasks optimization of a modular serial metamorphic manipulator.
Journal of Mechanisms and Robotics, 13(1), 2021.
[10] Wan Kyun Chung, Jeongheon Han, Youn Gil Youm, and SH Kim. Task based design of modular robot manipulator
using efficient genetic algorithm. In Proceedings of International Conference on Robotics and Automation, volume 1, pages 507–512. IEEE, 1997.
[11] Sarosh Patel and Tarek Sobh. Task based synthesis of serial manipulators. Journal of advanced research,
6(3):479–492, 2015.
[12] Saleh Tabandeh, William Melek, Mohammad Biglarbegian, Seong-hoon Peter Won, and Chris Clark. A memetic
algorithm approach for solving the task-based configuration optimization problem in serial modular and reconfigurable
robots. Robotica, 34(9):1979–2008, 2016.
[13] Anna Valente. Reconfigurable industrial robots: A stochastic programming approach for designing and assembling
robotic arms. Robotics and Computer-Integrated Manufacturing, 41:115–126, 2016.
[14] Anubhav Dogra, Srikanth Sekhar Padhee, and Ekta Singla. An optimal architectural design for unconventional
modular reconfigurable manipulation system. Journal of Mechanisms and Robotics, 143(6), 2021.
[15] Chris Leger and John Bares. Automated task-based synthesis and optimization of field robots. 1999.
[16] Julian Whitman, Raunaq Bhirangi, Matthew J Travers, and Howie Choset. Modular robot design synthesis with
deep reinforcement learning. In AAAI, pages 10418–10425, 2020.
[17] Esra Icer, Heba A Hassan, Khaled El-Ayat, and Matthias Althoff. Evolutionary cost-optimal composition synthesis
of modular robots considering a given task. In 2017 IEEE/RSJ International Conference on Intelligent Robots and
Systems (IROS), pages 3562–3568. IEEE, 2017.
[18] Julian Whitman and Howie Choset. Task-specific manipulator design and trajectory synthesis. IEEE Robotics and
Automation Letters, 4(2):301–308, 2018.
[19] J-O Kim and Pradeep K Khosla. A formulation for task based design of robot manipulators. In Proceedings of
1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS’93), volume 3, pages
2310–2317. IEEE, 1993.
[20] Stefan B Liu and Matthias Althoff. Optimizing performance in automation through modular robots. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4044–4050. IEEE, 2020.

[21] Hatem Al-Dois, AK Jha, and RB Mishra. Task-based design optimization of serial robot manipulators. *Engineering Optimization*, 45(6):647–658, 2013.

[22] Satwinder Singh, Ashish Singla, and Ekta Singla. Modular manipulators for cluttered environments: A task-based configuration design approach. *Journal of Mechanisms and Robotics*, 10(5), 2018.

[23] M Althoff, A Giusti, SB Liu, and A Pereira. Effortless creation of safe robots from modules through self-programming and self-verification. *Science Robotics*, 4(31):eaaw1924, 2019.

[24] Peter I Corke. A simple and systematic approach to assigning denavit–hartenberg parameters. *IEEE transactions on robotics*, 23(3):590–594, 2007.

[25] Thais Campos, Samhita Marri, and Hadas Kress-Gazit. Automated synthesis of modular manipulators’ structure and control for continuous tasks around obstacles. In *Robotics Science and Systems*, 2020.

[26] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[27] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: modelling, planning and control*. Springer Science & Business Media, 2010.

[28] Peter Corke. *Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised*, volume 118. Springer, 2017.

[29] Nick. Shortest2020 distance between two line segments. MATLAB Central File Exchange, Dec. 16 2020. [Online].

[30] David Legland. geom3d2020. MATLAB Central File Exchange, Dec. 16 2020. [Online].

[31] Adnan Munawar. Matlab implementation of rrt variants. [https://github.com/adnammunawar/matlab-rrt-variants](https://github.com/adnammunawar/matlab-rrt-variants) 2019.

[32] Xin-Jun Liu, Jinsong Wang, and Feng Gao. Performance atlases of the workspace for planar 3-dof parallel manipulators. *Robotica*, 18(5):563–568, 2000.