Dynamically Feasible Trajectory Generation for Soft Robots

Haley P. Sanders[1] and Marc D. Killpack[1]

***Abstract*— Potential applications for large-scale soft robots include interacting with humans while carrying a heavy load, navigating in clutter, executing impact tasks like hammering a nail into a wall, and so much more. Because of their compliance and lack of fragile gear trains, soft robots are uniquely suited to these tasks. However, we expect that path planning may be more constrained by soft robot kinematics and dynamics than traditional rigid robots. Generating dynamically feasible trajectories for soft robots (especially large-scale soft robots with higher payloads) is critical to the success of low-level controllers tracking reference trajectories. This paper introduces an optimization method to generate task and joint space trajectories for soft robots that satisfy kinematic and dynamic constraints which are unique to large-scale soft robots. The method presented in this paper is an offline trajectory generator that is then fed to a low-level PID joint angle controller. We conduct two experiments to validate this method on a continuum pneumatic soft robot of length 1.19 meters in both simulation and on hardware. We show that this is a viable method of planning trajectories for soft robots with a reported median magnitude of error of 0.032 meters between the planned and actual end effector trajectories.**

I. INTRODUCTION

Robots have the potential to perform a wide variety of tasks such as interacting with humans while lifting heavy objects, navigating in clutter, and even performing explosive or athletic motions like shooting a ball into a hoop or using human tools like hammers. Soft robots have an advantage over traditional rigid robots because of their compliance, and the ability to store and use potential energy in their actuators while performing such tasks. Effective and dynamically feasible trajectory planning is important to enabling soft robots to perform useful tasks while taking advantage of their complicated dynamics. As the size and scale of soft robots increase, effective trajectory planning becomes essential so that soft robots can dynamically reach every point within a planned trajectory [1]. Generating dynamically feasible trajectories is also important so that a soft robot's low-level controller can closely track the planned reference trajectories.

Soft robots are different from rigid robots in both their dynamics and kinematics. This can often make trajectory planning for soft robots more difficult than for traditional rigid robots for a number of reasons. First, the performance and behavior of soft robots is inherently linked to payload and actuation structure, unlike traditional highly-geared robots, resulting sometimes in non-intuitive dynamics. In this case, how can a human define the best or most optimal trajectory a soft robot should take to move an object from point A to point B? With traditional robots, for the most part, we are able to plan a path using kinematics only, but we cannot ignore these important dynamic constraints for soft robots. Additionally, traditional robots have little to no uncertainty in their shape or where their end effector is located in space (making it ideal to path plan using kinematics). However, soft robots have a large amount of uncertainty in both their total shape as well as the kinematics describing their end effector pose. Finally, control for soft robots which are most often underdamped can result in large amounts of variation in their end effector location over time.

This paper presents a new path and trajectory planning method for large-scale and pneumatically-actuated soft robots using both their dynamics and kinematics as constraints in an optimization problem. The optimization is formulated subject to several constraints, including minimum and maximum joint angle constraints, as well as a dynamic torque constraint. We also introduce a new constraint on pressure dynamics that helps our method generate dynamically feasible trajectories for soft robots which include systems with slower actuator dynamics. Additionally, the cost function is formulated using the kinematics of the soft robot. Planning a trajectory in this way solves several problems related to motion planning with soft robots, including ensuring that a feasible trajectory can be planned while not ignoring the complex dynamics of soft robots.

This paper presents the following contributions to the soft robotics community:

1) a formulation that takes a high-level Cartesian space task and distills it into a cost function that can be used in an optimization problem subject to unique soft robot constraints
2) a method to plan a reference trajectory of end effector Cartesian positions and soft robot joint angles that is feasible both kinematically and dynamically
3) a cascade control structure to deliver and execute that reference trajectory on a robot in simulation and hardware

We next provide an outline of the structure of this paper. In section II, we discuss previous work on soft robot motion planning. In section III, we give an overview of the dynamics and kinematics of our soft robot platform. We also introduce the optimization used to generate a trajectory of Cartesian positions and joint angles, and present the algorithm and method used to execute that trajectory on the soft robot. In section IV, we detail the experiments we used to validate our method in both simulation and on hardware. Section V presents the results of the experiments. We conclude in section VI with a discussion of the results and future work.

[1]Robotics and Dynamics Lab, *Brigham Young University*, Provo, USA
marc_killpack@byu.edu

2023 IEEE International Conference on Soft Robotics (RoboSoft) | 979-8-3503-3222-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/ROBOSOFT55895.2023.10122016

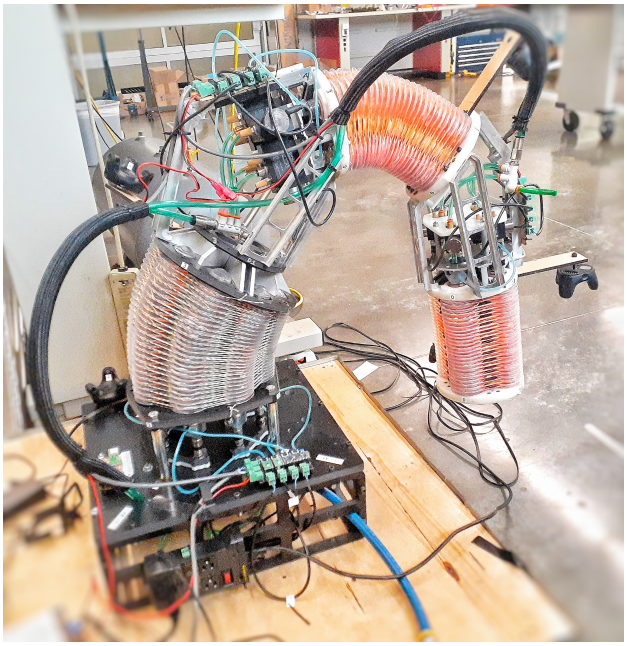

Fig. 1. The pneumatically-actuated continuum soft robot platform used for testing planned trajectories in this paper.

II. Related Work

The related work in the area of soft robot trajectory generation can be divided into topics on control for soft robots, kinematics-only approaches to soft robot trajectory generation, and dynamically constrained, optimization-based approaches to soft robot trajectory generation. We will discuss works related to soft robots in all of these areas, and discuss how our trajectory generation method extends the ideas in those works.

A. Control for Soft Robots

Before a robot can execute a high-level trajectory in task space, there must be a good low-level controller that can track the high-level trajectory. Much of the research in the area of controlling soft robotics has been focused on creating accurate, real-time, low-level controllers that plan a trajectory of inputs to a robotic system, whether they be pressures or torques [2]. Because of the important effects of underdamped and continuum dynamics, many soft robot low-level controllers use model predictive control (MPC) for trajectory following tasks [3]–[5], while others use data-driven and machine learning methods [6]–[9].

The most similar to our approach in this paper is described in [10], where an optimizer performs as a low-level controller. The optimizer plans a trajectory of joint angles and inputs to minimize the error between a reference trajectory of Cartesian locations and the current robot tip position. Despite being somewhat similar to our work, in this paper, we plan trajectories at a higher level by not constraining the tip position to a reference trajectory in time, but instead by letting the optimization find trajectories that are dynamically feasible.

B. Kinematics-Only Trajectory Generation for Soft Robots

Whether using model-based low-level control or not, there is also research related to generating a reference trajectory in task and joint space to be executed by a low-level controller. One approach to task space motion planning for soft robots is a kinematics-only approach. Some kinematics-only methods use Jacobians to plan trajectories [11]. Others use inverse kinematics optimizations [12], [13]. In [14], a whole-arm planar planning algorithm is developed for a soft robot using a constant curvature model as its kinematics. The robot achieves a series of user-defined way points by adjusting its shape along a time trajectory, and the shape is determined using a Sequential Quadratic Programming algorithm. This approach works for lightweight, planar soft robot arms, but our robot is made of rigid and soft segments, with greater size and mass subject to greater gravitational forces and therefore cannot neglect dynamic constraints.

C. Dynamically Constrained Trajectory Generation for Soft Robots

Another approach to soft robot trajectory planning is what the authors in [15] call “Goal Directed Dynamics”. Although the authors do not apply this method explicitly to soft robots, they introduce a method of trajectory generation where an optimization is provided a goal, and the robot’s dynamics and kinematics serve as constraints that drive the robot to the goal state. Many soft robot motion planners use optimizations with a cost function driving the robot tip toward a goal point and constraints on the input dynamics [16]. This method is common in robots that use series elastic actuators (SEAs) to introduce elasticity to their robot. For example, in [17] they present an algorithm that generates a trajectory of motor positions subject to maximum and minimum allowable motor torques. Similarly, [18] enables a small robot arm with SEAs to mimic human motion and drive the robot tip towards a goal point using optimization.

Other works with robotic platforms more similar to ours include [1], where a soft robot hanging straight down tries to drive its end effector toward a goal point. They use direct collocation trajectory optimization to generate a trajectory of inputs that are then translated into manipulator motion to achieve a goal point in Cartesian space. Other related literature presents soft robot motion planners that have the same general optimization structure: a cost function drives the robot’s tip towards a goal point by changing the joint angles and inputs. At each step in the trajectory, the optimization is subject to dynamic constraints whether it be on torques, pressures, or some mapping from input space to state space [19], [20].

In this paper, we seek to build on the concept of motion planning for soft robots using dynamically constrained optimization. However, our method differs from previous works in that our dynamic constraints are on input pressures and pressure derivatives for large, pneumatically-actuated soft robots designed to do complex (i.e. 6 degree-of-freedom) tasks in Cartesian space.

III. Trajectory Planner Formulation

This section outlines the formulation of our high-level trajectory planner for a soft robot. First, the dynamics and kinematics of the robot arm are discussed. Second, we formulate the optimization used to generate task and joint space trajectories. Third, we present the algorithm that utilizes the optimization to generate an offline trajectory. Finally, we present the control cascade structure the robot uses to track the generated trajectory.

A. Dynamics and Kinematics

The soft robot model used in this paper is for a pneumatically driven hybrid robot arm consisting of a series of rigid links and soft joints (see Figure 1). The soft segments are continuum, constant-curvature joints made of blow-molded plastic and driven by four independent pressure controlled chambers. We model the continuum kinematics using six constant curvature joint angles (two per joint) to determine the robot's position in task space. The angle of rotation about each joint's x-axis is denoted by u, and the angle of rotation about the robot's y-axis is denoted by v. See [3], [21] for more details on the arm's kinematic model.

In order to generate a dynamically feasible trajectory that the robot can track, we use the Recursive Newton-Euler formulation of the forward dynamics found in [22]. The dynamics can be summarized as:

$$K_{prs}p = M\ddot{q} + C\dot{q} + G + K_{spring}q + K_d\dot{q} \qquad (1)$$

where q is a vector of joint angles (two per joint), and $\dot{q}$ and $\ddot{q}$ are the time derivatives of q. K_{prs} is a pressure-to-torque mapping, M is the mass matrix, C contains the Coriolis terms, and G is the gravity vector. K_{spring} and K_d are spring coefficients and damping matrices that describe the behavior of the continuum joints. Finally, p is a vector of twelve pressures, one for each pressure chamber (or pair of chambers in the case of the eight-bellows on the proximal or base joint) on the robot arm. This means that there are four control inputs for each joint.

B. The Trajectory Generation Optimization

Many trajectory optimization formulations in robotics generate trajectories of inputs to track a predetermined set of reference joint angles for a robot. In this paper, we seek to generate the reference trajectory itself (of dynamically feasible joint angles), with the goal of accomplishing some task in Cartesian space. In order to demonstrate the difference between the two problems, we highlight several distinctions between optimizing a trajectory of inputs for tracking, and optimizing a trajectory of joint positions that would best accomplish some task in Cartesian space.

The optimization used to simultaneously generate task and joint space trajectories has the same structure as traditional robotic trajectory tracking problems, meaning that it has a cost function, design variables, and constraints. However, in traditional robotic manipulator tracking problems (for both rigid and soft robots), the cost function is often based on the error between a predetermined reference trajectory of commanded joint angles and actual joint angles. The design variables are generally inputs to the manipulator (such as torques for rigid robots and pressures for soft robots), and there are constraints on the state of the robot at every time step defined by the forward dynamics of the robot.

Our optimization differs from the traditional tracking problem in several ways. First, we propose using joint angles instead of inputs as design variables for the optimization problem. Instead of feeding a reference trajectory to the cost function, the cost function will be a quadratic cost on the difference between the current Cartesian location of the robot's end effector and a single goal way point, or a vector of way points placed throughout the robot's workspace. It is important to note that even if the way points define a desired path, we do not expect to define a time-based trajectory for that Cartesian path because doing so may generate a trajectory that is infeasible based on the dynamics and actuation limits of our soft robot. Second, the optimization constraints will include a minimum and maximum pressure limit inside each chamber at the soft robot's joints, defined by the robot's inverse dynamics, as well as minimum and maximum allowable joint angles. We also introduce a new constraint unique to large, pneumatically-actuated soft robots like ours: a constraint on change in pressure, $\dot{p}$. We describe this constraint in more detail below. The optimization problem is formally defined as

$$\begin{aligned}
\min \quad & J = (c_{i+1} - c_e)^T Q (c_{i+1} - c_e) + \\
& (q_{i+1} - q_i)^T R (q_{i+1} - q_i) + \\
& (c_{i+1} - c_i)^T H (c_{i+1} - c_i) \\
\text{w.r.t.} \quad & q_{i+1}, \quad \forall i \in 0, 1, \ldots, T \\
\text{s.t.} \quad & q_{\min} \leq q_{i+1} \leq q_{\max}, \quad \forall i \in 0, 1, \ldots, T \\
& p_{\min} \leq p_{i+1} = K_{prs}^{-1}(M(q_{i+1})\ddot{q}_{i+1} \\
& + C(q_{i+1}, \dot{q}_{i+1})\dot{q}_{i+1} + G(q_{i+1}) + K_{spring}q_{i+1} \\
& + K_d\dot{q}_{i+1}) \leq p_{\max}, \quad \forall i \in 0, 1, \ldots, T \\
& \dot{p}_{\min} \leq \dot{p}_{i+1} = differentiate(p_{i+1}, p_i) \leq \dot{p}_{\max}, \\
& \quad \forall i \in 0, 1, \ldots, T
\end{aligned} \qquad (2)$$

where c_{i+1} is the Cartesian location of the end effector and c_e is the desired Cartesian location. We add additional terms ($(c_{i+1} - c_i)$ and $(q_{i+1} - q_i)$) to the cost function to achieve a smooth trajectory. Q is a weighting matrix on error between our desired and actual Cartesian location, R is a weighting matrix on smoothness in joint space, and H is a weighting matrix on smoothness in Cartesian space. T is a user-defined time horizon over which to optimize the trajectory and i is a timestep in that horizon. p_{max} is the maximum pressure allowed in a single blow-molded chamber based on safety constraints. For our robot, the maximum allowable pressure was 310 KPa. $\dot{p}_{i+1}$ is a constraint on pressure dynamics. The rate of change of pressure is constrained by several soft robot parameters like the current volume and pressure of the individual blow-molded chambers, the maximum allowable diameter of the pressure valves used to fill and vent the chambers, and the electrical current available to open and

close the same valves (in the case of our specific hardware). $\dot{p}_{max}$ may also a be function of q or rather $\dot{q}$, but we have no way of expressing this in the optimization problem with our current dynamic formulation. The combination of these variables means that there is a constraint on how fast the blow-molded chambers can actually reach a commanded pressure.

C. Generating a Trajectory

To compute a trajectory of joint angles that will cause the end effector to travel to some desired location, we present Algorithm 1, which is computed offline. This algorithm starts with the initial state of the robot at q_0 and $\dot{q}_0$. It then enters a while loop that terminates when the predicted robot tip would be within some ϵ distance of c_e.

The optimization described in Equation 2 takes place inside of the "while" loop. The optimization algorithm COBYLA (Constrained Optimization By Linear Approximation) returns $q_{nextTraj}$ (a trajectory for all joint angles over the time horizon T). COBYLA uses Algorithm 2, or $constrFunc$ internally. This serves as the function that enforces constraints on p_{Traj} and $\dot{p}_{Traj}$ (pressure and pressure dot trajectories across the horizon) described in Equation 2. Algorithm 2 shows how we iterate over $q_{nextTraj}$ to calculate p_{Traj} and $\dot{p}_{Traj}$.

Algorithm 1 Trajectory Generator

i = 0
$q_i = q_0$
$\dot{q}_i = \dot{q}_0$
while $||c_{i+1} - c_e||_2 > \epsilon$ **do**
    $q_{nextTraj} \leftarrow COBYLA(T, constrFunc, costFunc)$
    $q_{i+1} = q_{nextTraj}[0]$
    $\dot{q}_{i+1} = differentiate(q_{i+1}, q_i, dt)$
    $c_{i+1} = get\_fk(q_{i+1})$ ▷ forward kinematics
    $q_i = q_{i+1}$
    $\dot{q}_i = \dot{q}_{i+1}$
    i = i + 1
end while

Algorithm 2 Constraint Function

$q_{nextTraj} = optimizer()$ ▷ optimizer chosen joint angle trajectory across horizon
$q_k = qTraj[0]$
for $k = 0 : T$ **do**
    $q_{k+1} = q_{nextTraj}[k+1]$
    $\dot{q}_{k+1} = differentiate(q_{k+1}, q_k)$
    $\ddot{q}_{k+1} = differentiate(\dot{q}_{k+1}, \dot{q}_k)$
    $p_{Traj}[i+1] = dynamics(q_{k+1}, \dot{q}_{k+1}, \ddot{q}_{k+1})$ ▷ Eq 1
    $\dot{p}_{Traj}[i+1] = differentiate(p_{i+1}, p_i)$
end for
return $p_{Traj}, \dot{p}_{Traj}$

COBYLA also uses Algorithm 3, or the $costFunc$. Algorithm 3 calculates the cost function in Equation 2. Inside Algorithm 3, we iterate over $q_{nextTraj}$ and sum up the cost at each timestep in the horizon.

Algorithm 3 Cost Function

$q_{nextTraj} = optimizer()$ ▷ optimizer chosen joint angle trajectory across horizon
$q_j = qTraj[0]$
$cost = 0$
for $j = 0 : T$ **do**
    $q_{j+1} = q_{nextTraj}[j+1]$
    $c_{j+1} = get\_fk(q_{j+1})$ ▷ forward kinematics
    $cost = cost + (c_{j+1} - c_e)^T Q(c_{j+1} - c_e) + (q_{j+1} - q_j)^T R(q_{j+1} - q_j) + (c_{j+1} - cj)^T H(c_{j+1} - c_j)$
end for
return $cost$

After the optimization algorithm has converged on a trajectory of optimal and feasible joint angles, the first set of joint angles in $q_{nextTraj}$ is saved as the next predicted state of the robot. The optimization is performed over and over, each time using the next predicted state of the robot as the first set of joint angles in $q_{nextTraj}$.

This approach of solving for an entire time horizon, but only storing the first time step in $q_{nextTraj}$, before reformulating the optimization and resolving is reminiscent of receding horizon control (or MPC). The main reason for this approach is two-fold. First, we do not know apriori the required time to reach from our initial condition to the final goal pose at the tip of the soft robot in Cartesian space. By optimizing over a horizon, but building the planned trajectory by one step for each iteration, we expect to find a dynamically feasible solution in whatever minimum time is possible. Second, although there is a large potential parameter space relative to the horizon length T, and the number of steps stored from $q_{nextTraj}$ for each iteration, in practice we empirically found an approach that worked best given the problem formulation and optimizer used. This approach involved using a shorter horizon T, and taking only the first set of joint angles from that horizon, while re-optimizing over and over. This allowed the optimizer to more quickly converge on a smoother and faster (i.e. shorter in time) trajectory of joint angles.

D. Cascade Control Structure

Once a trajectory of joint angles has been generated offline, we track that trajectory using the control cascade structure shown in Figure 2.

The trajectory generator box in green consists of the methods described in the last two sections, including Equation 2 and the algorithms that use it. The trajectory is generated offline, although we plan to use the same algorithms to do real-time trajectory planning in the future. The trajectory generator outputs a reference trajectory of joint angles which are then fed to a joint-level PID controller. The PID gains for the joint-level controller were tuned offline to achieve smooth behavior given step inputs. The joint controller calculates a

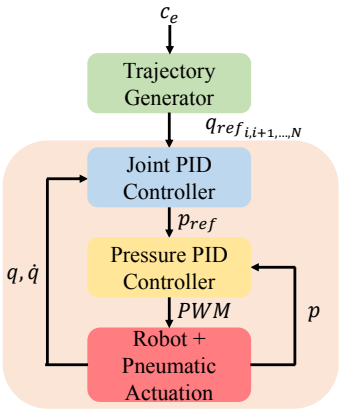

Fig. 2. A visual depiction of the cascade control structure. A single or set of desired Cartesian way points is given to the trajectory generator. A trajectory is computed offline, and the resulting trajectory of joint angles are sent to a joint PID controller. Reference pressures are sent to a pressure PID controller, which sends a PWM signal to control the robot's pneumatic valves to fill or vent air into pneumatic chambers. Everything in orange happens in real time.

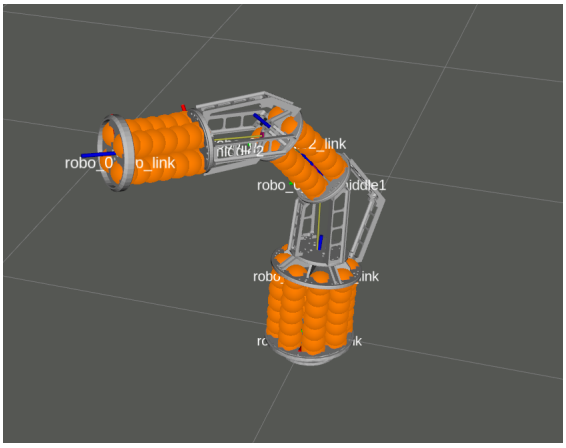

Fig. 3. The robot in a simulation environment.

vector of reference pressures which are then fed to a low-level PID pressure controller. The output of the PID pressure controller (with gains also tuned offline) is a PWM signal, which drives pressure valves onboard the robot that allow air to flow in or out of the pressure chambers. Everything inside the orange box happens in real-time.

IV. Validation Experiment

A. Hardware Description

The robot used to validate our trajectory generator is shown in Figure 1. The arm consists of three pneumatically driven soft continuum segments. As discussed in section III-A, each soft segment is driven by four independent pressures which create a net torque that causes the robot to bend about an inextensible steel cable at the center of the joint. Each soft segment is connected by a rigid link that contains the pneumatic valves and pressure sensors used to fill and vent the chambers. Despite the joints being continuum joints, we model the robot as having a total of six degrees of freedom, or six joint angles. To measure joint angles, we use six HTC Vive Trackers which are attached to the arm at the top and bottom of each link. See [21], [22] for more details on the makeup of each soft segment and the transformations used to convert Vive Tracker data to joint angles.

B. Experiments

We validated our trajectory generator by conducting two different experiments in simulation (see Figure 3) and on the actual robot hardware. In each experiment, way points were fed to the trajectory generator to compute paths of different Cartesian shapes.

In the first experiment, we gave the trajectory generator a single way point, or a single desired end effector location in Cartesian space at $c_e = [-.075, .769, .541]^T$. We chose this location because it is away from any workspace singularities. We chose the starting location to be at $c_{i=0} = [-0.769, 0.075, 0.541]^T$ again to start the robot away from workspace singularities. The optimization horizon was $T = .3$ seconds with a time step of .1 seconds. Because we wanted the trajectory to quickly and smoothly converge on the desired Cartesian location, the weighting matrices, which have a significant impact on the speed and performance of the final trajectory, used in the cost function were $Q = diag(5, 5, 5)$, $R = diag(10, ..., 10)$, and $H = diag(5, 5, 5)$. These weighting matrices were empirically tuned.

For the first experiment, we also compared our trajectory generator to a direct input to the underlying PID controller to show the effect of the planner versus closed-loop control. The same PID gains were used to send the joint angle step command and trajectory of joint angle commands to the robot in simulation.

In the second experiment, we gave the trajectory generator a series of way points that formed a circle of radius 0.15 meters. The top of the circle (or starting position of the end effector) was chosen to be at $c_{i=0} = [-.288, .288, .288]^T$. The cost function used the same weighting matrices as the first experiment. This experiment also had an additional cost on the distance between the current end-effector position and future way points to ensure a smooth trajectory between way points. However, the weighting of this cost compared to the weighting on distance between end-effector and next way point and smoothness was relatively small, and likely could be removed as it had almost no affect on the final generated trajectory. The optimization horizon was also the same as the first experiment.

In the second experiment, we also compared our trajectory generator to sending step commands of joint angles that moved the robot's end-effector to the same series of way points that formed a circle of radius .15 meters.

C. Software Description

Algorithm 1 was written in Python. The optimizer used to solve Equation 2 was the *scipy.optimize.minimize* function. All the trajectories were computed on an AMD Ryzen 9 5900 12-Core Processor. After the trajectories were computed, they were sent to the joint PID controller at a

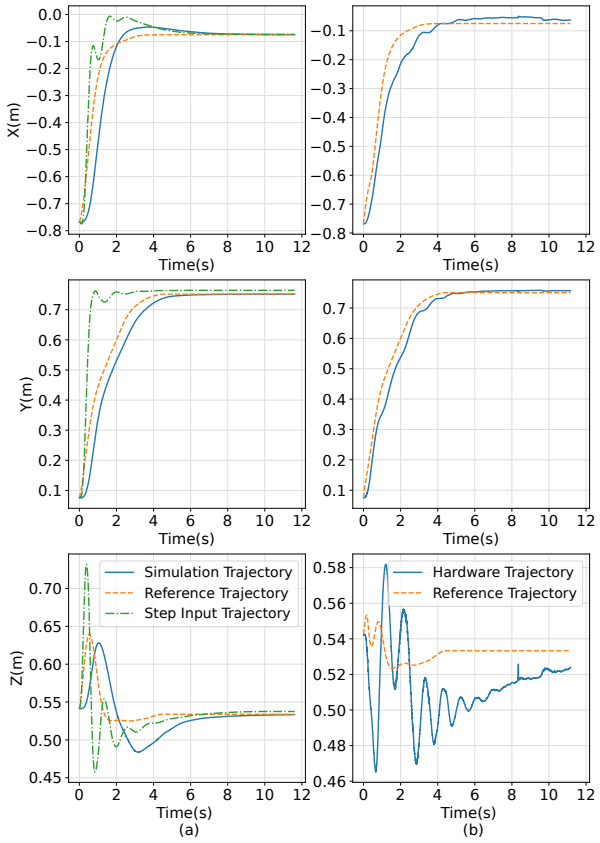

Fig. 4. Graphs showing the trajectories generated given a single way point, and how well the robot tracked the trajectories. Column (a) shows the trajectories in simulation, and column (b) shows the trajectories on hardware. Also shown is the path the robot took given a step input of joint angles.

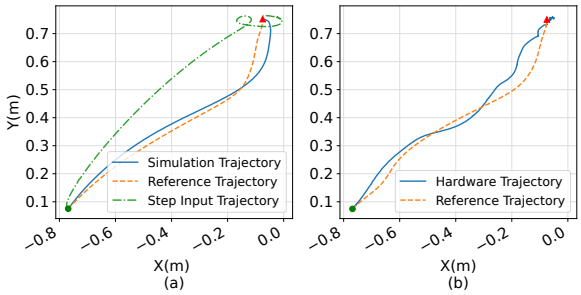

Fig. 5. 2D depictions of the generated trajectories given a single way point, and how well the robot tracked them. (a) shows the simulation trajectories. (b) shows the hardware trajectories. The green dot represents the robot's starting point, and the red triangle represents its ending point. See https://youtu.be/Ew4b-6SRi-Y for a video of the robot executing this trajectory.

TABLE I

ERROR STATISTICS BETWEEN GENERATED REFERENCE TRAJECTORY AND CARTESIAN POSITIONS FOR A LINE.

Cartesian Directions	x	y	z	$Total$
Simulation Mean (m)	0.034	0.027	0.018	0.051
Simulation Median (m)	0.011	0.003	0.008	0.015
Simulation Max (m)	0.279	0.167	0.079	.320
Hardware Mean (m)	0.039	0.023	0.024	0.056
Hardware Median (m)	0.020	0.007	0.022	0.032
Hardware Max (m)	0.159	0.097	0.081	0.183

a rate of 200 Hz. Pressure commands were sent from the joint controller to the pressure PID controller at 475 Hz. The pressure controller ran at 1 kHz.

V. RESULTS

Figure 4 shows the generated Cartesian reference trajectories for the first experiment in dashed lines. Each Cartesian direction is shown plotted versus time. Also shown in Figure 4 is a solid line of the simulation and hardware robots tracking the generated trajectories. The dash-dot line shows the Cartesian path the robot took in simulation when given a step input of joint angles rather than a reference trajectory of joint angles. The simulation trajectory is shown in the left column and the hardware trajectory is shown in the right column.

Figure 5 shows the same trajectories as Figure 4 but in the xy plane, along with the simulation and hardware tracking results. Our trajectory generator computed trajectories that take 4.3 seconds to travel from the starting location to the desired ending location. Table I shows the mean, median, and maximum error between the generated reference trajectories and trajectory the robot took. For reference, our robot's total length is 1.19 meters. As can be seen in Table I, the total mean magnitude of error between the generated trajectory and executed trajectory in simulation was 0.051 meters, and the total median magnitude of error was 0.015 meters. On hardware, the total mean absolute error was 0.056 meters, and the total median absolute error was 0.032 meters.

Figure 6 shows the generated Cartesian reference trajectories for the second experiment in dashed lines (similar to the format for Figure 4, but with a different goal location). Also shown in the dash-dot line is the path the simulation robot took when given a series of joint angle step commands. The simulation trajectory is again shown in the left column and the hardware trajectory is shown in the right column.

Figure 7 shows the same circular trajectories as Figure 6 but in the xy plane. It also shows the simulation and hardware tracking of these trajectories. Our trajectory generator computed trajectories that take 18.9 seconds to travel from the starting location, around the circle, and back to the starting location. Table II shows the mean, median, and maximum error between the generated reference trajectories and trajectories the robot took for both simulation and hardware experiments. As can be seen in Table II, the total mean magnitude of Cartesian error between the generated trajectory and executed trajectory in simulation was 0.017 meters, and the total median magnitude of error was 0.019 meters. On hardware, the total mean absolute error was 0.025 meters, and the total median absolute error was 0.025 meters.

VI. DISCUSSION AND CONCLUSION

Figures 4 through 7 indicate that the soft robot trajectory generation algorithm presented in this paper can generate dynamically feasible and useful paths and trajectories for soft robots to follow. This is supported by statistics in Tables I

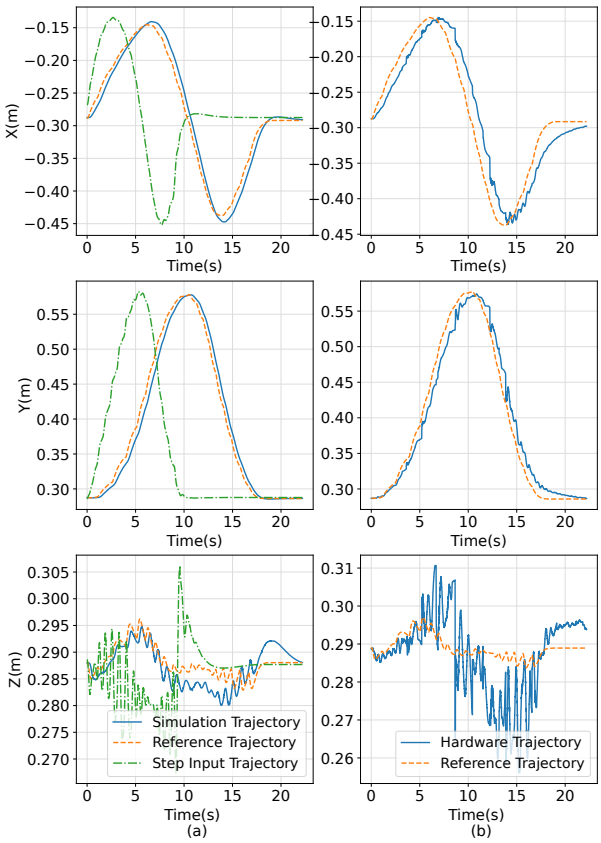

Fig. 6. Graphs showing the trajectories generated to form a circle of .15 meters, and how well the robot tracked the trajectories. Column (a) is the slow trajectory, and column (b) is the fast trajectory. Also shown is the path the robot took given a series of step inputs of joint angles.

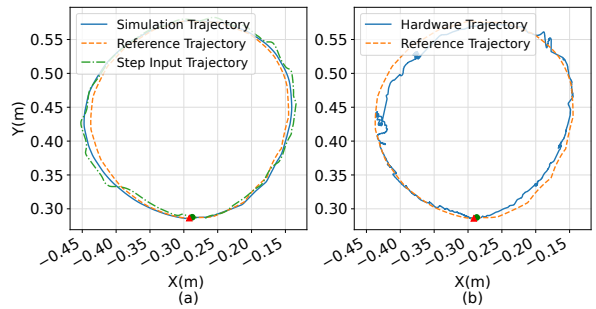

Fig. 7. Circular trajectories generated in both simulation and hardware shown in the xy plane, along with the robot's tracking results. (a) shows the simulation trajectories. (b) shows the hardware trajectories. See https://youtu.be/Ew4b-6SRi-Y for a video of the robot executing this trajectory.

TABLE II

ERROR STATISTICS BETWEEN GENERATED REFERENCE TRAJECTORY AND CARTESIAN POSITIONS FOR A CIRCLE.

Cartesian Directions	x	y	z	$Total$
Simulation Mean (m)	0.011	0.010	0.002	0.017
Simulation Median (m)	0.011	0.009	0.002	0.019
Simulation Max (m)	0.025	0.028	0.006	0.029
Hardware Mean (m)	0.018	0.011	0.006	0.025
Hardware Median (m)	0.016	0.007	0.004	0.025
Hardware Max (m)	0.060	0.042	0.029	0.072

and II where the largest median magnitude of error between generated and executed trajectories in any direction is only 0.032 meters for an arm of length 1.19 meters.

However, it is clear that improvements could be made to the algorithm. The trajectories generated for the physical robot are smooth, but the tracking of these trajectories in hardware show small oscillations around the reference trajectories. This error could stem from a few possible sources. First, the dynamic model used to generate trajectories does not perfectly match the dynamics of the real system. The paper used to mathematically model our soft robot indicates that there are possibly unmodeled dynamics between pressures and joints [22]. Additionally, the commanded pressures from the PID control were never greater than or equal to p_{max}, meaning that our algorithm was not pushing the system to be overly aggressive based on our current cost function tuning. This is possibly due to not having an accurate analytical expression for $\dot{p}$. Despite the fact that our dynamic model does not match the real robot, this is not necessarily an indication of a poorly formulated trajectory planner, but instead a confirmation that improved dynamic models for soft robots are still an open research question and necessary to further improve performance. When our dynamic model used in the trajectory generator matched the simulation's dynamics, the robot was able to closely track the generated trajectories as seen in the previously mentioned figures.

The second source of possible error contributing to the non-smooth motion of the physical robot's end effector is our low-level joint controller. We used a PID controller to drive the joint angles of the real robot towards the desired trajectory. We tuned the gains of the controller to follow the highly dynamic trajectories, but as can be seen in Figures 5 and 7, this often caused the robot's end effector to oscillate around the reference trajectory. Creating a better joint-level soft robot controller is outside of the scope of this paper, but our results could build on existing research in this area (see [3], [4], [6], [10]). We plan to implement one of these controllers in the future for better trajectory tracking.

In conclusion, this paper presents a dynamically feasible trajectory generation algorithm for highly compliant, underactuated soft robots. This algorithm takes a high-level, ambiguous task (such as moving from point A to point B), and distills it down into a mathematical optimization problem that can be used to generate a path in Cartesian space as well as a trajectory of joint angles to follow that path. Future work on this algorithm includes making it real-time and closed loop. By using current end effector positions and re-planning the trajectory in real-time, we can account for uncertainties in the soft robot's shape and dynamics.

ACKNOWLEDGEMENTS

The authors recognize and greatly appreciate the contributions of Curtis C. Johnson with hardware and discussions related to the content of the paper. This work was supported by the National Science Foundation under Grant no. 1935312.

REFERENCES

[1] A. D. Marchese, R. Tedrake, and D. Rus, "Dynamics and trajectory optimization for a soft spatial fluidic elastomer manipulator," *The International Journal of Robotics Research*, vol. 35, no. 8, pp. 1000–1019, July 2016, publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1177/0278364915587926

[2] J. Wang and A. Chortos, "Control Strategies for Soft Robot Systems," *Advanced Intelligent Systems*, vol. 4, no. 5, p. 2100165, 2022, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202100165. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202100165

[3] P. Hyatt, C. C. Johnson, and M. D. Killpack, "Model Reference Predictive Adaptive Control for Large-Scale Soft Robots," *Frontiers in Robotics and AI*, vol. 7, 2020, publisher: Frontiers. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2020.558027/full

[4] P. Hyatt and M. D. Killpack, "Real-Time Nonlinear Model Predictive Control of Robots Using a Graphics Processing Unit," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1468–1475, Apr. 2020, conference Name: IEEE Robotics and Automation Letters.

[5] D. Bruder, X. Fu, R. B. Gillespie, C. D. Remy, and R. Vasudevan, "Data-Driven Control of Soft Robots Using Koopman Operator Theory," *IEEE Transactions on Robotics*, vol. 37, no. 3, pp. 948–961, June 2021, conference Name: IEEE Transactions on Robotics.

[6] C. C. Johnson, T. Quackenbush, T. Sorensen, D. Wingate, and M. D. Killpack, "Using First Principles for Deep Learning and Model-Based Control of Soft Robots," *Frontiers in Robotics and AI*, vol. 8, 2021. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2021.654398

[7] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Learning dynamic models for open loop predictive control of soft robotic manipulators," *Bioinspiration & Biomimetics*, vol. 12, no. 6, p. 066003, Oct. 2017. [Online]. Available: https://iopscience.iop.org/article/10.1088/1748-3190/aa839f

[8] T. G. Thuruthel, E. Falotico, M. Manti, and C. Laschi, "Stable Open Loop Control of Soft Robotic Manipulators," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1292–1298, Apr. 2018, conference Name: IEEE Robotics and Automation Letters.

[9] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Model-Based Reinforcement Learning for Closed-Loop Dynamic Control of Soft Robotic Manipulators," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 124–134, Feb. 2019, conference Name: IEEE Transactions on Robotics.

[10] F. A. Spinelli and R. K. Katzschmann, "A Unified and Modular Model Predictive Control Framework for Soft Continuum Manipulators under Internal and External Constraints," Tech. Rep., Apr. 2022, publication Title: arXiv e-prints ADS Bibcode: 2022arXiv220413710S Type: article. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2022arXiv220413710S

[11] T. Greigarn, N. L. Poirot, X. Xu, and M. C. Çavuşoğlu, "Jacobian-Based Task-Space Motion Planning for MRI-Actuated Continuum Robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 145–152, Jan. 2019, conference Name: IEEE Robotics and Automation Letters.

[12] A. D. Marchese and D. Rus, "Design, kinematics, and control of a soft spatial fluidic elastomer manipulator," *The International Journal of Robotics Research*, vol. 35, no. 7, pp. 840–869, June 2016, publisher: SAGE Publications Ltd STM. [Online]. Available: https://doi.org/10.1177/0278364915587925

[13] D. M. Bodily, "Design Optimization and Motion Planning For Pneumatically-Actuated Manipulators," p. 80.

[14] A. D. Marchese, R. K. Katzschmann, and D. Rus, "Whole arm planning for a soft and highly compliant 2D robotic manipulator," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept. 2014, pp. 554–560, iSSN: 2153-0866.

[15] E. Todorov, "Goal Directed Dynamics," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. Brisbane, QLD: IEEE, May 2018, pp. 2994–3000. [Online]. Available: https://ieeexplore.ieee.org/document/8462904/

[16] C. D. Santina, L. Pallottino, D. Rus, and A. Bicchi, "Exact Task Execution in Highly Under-Actuated Soft Limbs: An Operational Space Based Approach," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2508–2515, July 2019, conference Name: IEEE Robotics and Automation Letters.

[17] A. Palleschi, R. Mengacci, F. Angelini, D. Caporale, L. Pallottino, A. De Luca, and M. Garabini, "Time-Optimal Trajectory Planning for Flexible Joint Robots," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 938–945, Apr. 2020, conference Name: IEEE Robotics and Automation Letters.

[18] F. Angelini, C. Della Santina, M. Garabini, M. Bianchi, and A. Bicchi, "Control Architecture for Human-Like Motion With Applications to Articulated Soft Robots," *Frontiers in Robotics and AI*, vol. 7, 2020. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2020.00117

[19] T. Marcucci, M. Garabini, G. M. Gasparri, A. Artoni, M. Gabiccini, and A. Bicchi, "Parametric Trajectory Libraries for Online Motion Planning with Application to Soft Robots," in *Robotics Research*, ser. Springer Proceedings in Advanced Robotics, N. M. Amato, G. Hager, S. Thomas, and M. Torres-Torriti, Eds. Cham: Springer International Publishing, 2020, pp. 1001–1017.

[20] C. D. Santina, A. Bicchi, and D. Rus, "Dynamic Control of Soft Robots with Internal Constraints in the Presence of Obstacles," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov. 2019, pp. 6622–6629, iSSN: 2153-0866.

[21] P. Hyatt, D. Kraus, V. Sherrod, L. Rupert, N. Day, and M. D. Killpack, "Configuration Estimation for Accurate Position Control of Large-Scale Soft Robots," *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 1, pp. 88–99, Feb. 2019, conference Name: IEEE/ASME Transactions on Mechatronics.

[22] S. W. Jensen, C. C. Johnson, A. M. Lindberg, and M. D. Killpack, "Tractable and Intuitive Dynamic Model for Soft Robots via the Recursive Newton-Euler Algorithm." Edinburgh, Scotland: IEEE.