Caging in Time: A Framework for Robust Object Manipulation under Uncertainties and Limited Robot Perception

Gaotian Wang[*1], **Kejia Ren**[*1], **Andrew S. Morgan**[2], **and Kaiyu Hang**[1]

[1]**Department of Computer Science, Rice University, Houston, TX 77005, USA**
[2]**The AI Institute, Cambridge, MA 02142, USA**
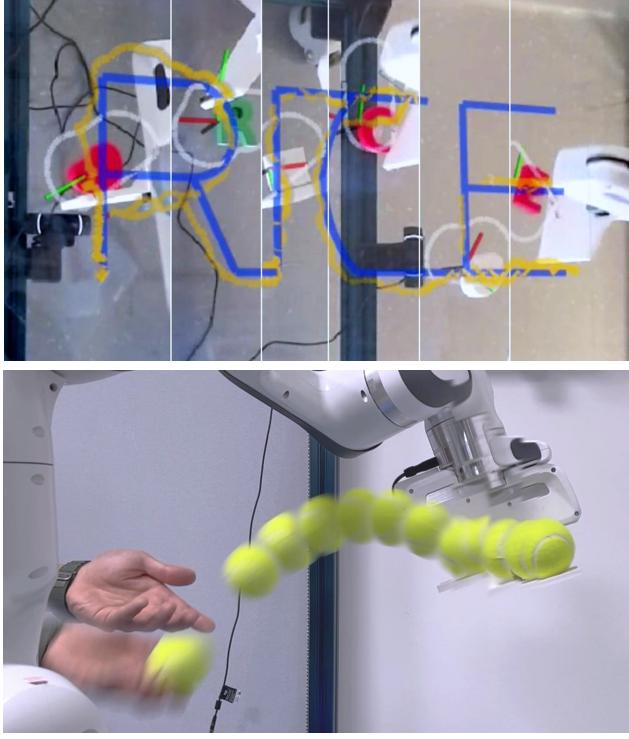[*]**Equal Contribution**

Abstract

Real-world object manipulation has been commonly challenged by physical uncertainties and perception limitations. Being an effective strategy, while caging configuration-based manipulation frameworks have successfully provided robust solutions, they are not broadly applicable due to their strict requirements for the availability of multiple robots, widely distributed contacts, or specific geometries of the robots or the objects. To this end, this work proposes a novel concept, termed *Caging in Time*, to allow caging configurations to be formed even if there is just one robot engaged in a task. This novel concept can be explained by an insight that even if a caging configuration is needed to constrain the motion of an object, only a small portion of the cage is actively manipulating at a time. As such, we can switch the configuration of the robot strategically so that by collapsing its configuration *in time*, we will see a cage formed and its necessary portion active whenever needed. We instantiate our *Caging in Time* theory on challenging quasi-static and dynamic manipulation tasks, showing that *Caging in Time* can be achieved in general state spaces including geometry-based and energy-based spaces. With extensive experiments, we show robust and accurate manipulation, in an open-loop manner, without requiring detailed knowledge of the object geometry or physical properties, nor real-time accurate feedback on the manipulation states. In addition to being an effective and robust open-loop manipulation solution, the proposed theory can be a supplementary strategy to other manipulation systems affected by uncertain or limited robot perception.
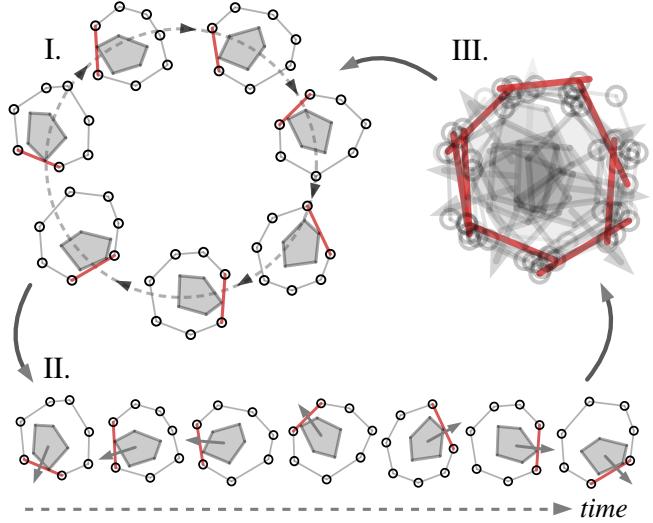
1 Introduction

Object manipulation is a fundamental ability for robots to engage themselves in tasks where physical interactions are expected (Billard and Kragic 2019). While research in this field has seen significant advancements with data-driven frameworks and sensing-enhanced systems (Kaelbling 2020; Yuan et al. 2017; Lee et al. 2019), we still do not see many robot systems working robustly around us. Among others, two major reasons are typically observed for this challenge: 1) real-world physical uncertainties are significantly more complex than any lab environment, often rendering certain modeling assumptions invalid (Rodriguez 2021), and 2) perception or sensing is almost never good enough for real-world robot deployments. Gaps between modeling assumptions and reality, domain variations, and random occlusions often worsen the perception performance, or even cause a perception system to completely lose track of target objects (Bohg et al. 2017). As such, approaches fully relying on closed-loop control for robot manipulation have been constantly challenged in real-world applications.

Different from approaches that focus on contacts, caging configuration-based manipulation has provided a novel paradigm to significantly reduce perception requirements. More importantly, although it does not aim at accurate control, caging configurations can robustly work by fully ignoring the effect of physical uncertainties, such as seen in grasping, in-hand manipulation, and multi-robot coordination tasks (Rodriguez et al. 2012; Song et al. 2021; Bircher et al. 2021). Concretely, a caging configuration aims at completely constraining all possible configurations of the target object within a known region (cage). The target object is manipulated as the cage moves or deforms, while the configuration of the target object is guaranteed to follow the cage to complete the task. However, it has been a challenge to apply this idea to general manipulation tasks, since forming a caging configuration has very strict requirements on the hardware, such as multiple robots to coordinate, widely distributed contacts, or specific geometries of the robot or the object to allow such configurations to exist (Makita and Wan 2018).

This work proposes a novel concept, termed *Caging in Time*, to extend caging configuration-based manipulation to more general problems without hardware-specific assumptions. Example applications are shown in Fig. 1. The high-level idea of this new theory can be explained as follows. In an extreme situation, let us assume we have an object to manipulate and a robot is able to make an infinite number of contacts everywhere on the object. As such, the object is fully caged, and arbitrary

Figure 1. Example object manipulation tasks via *Caging in Time*. *Top*: Object planar pushing to trace "RICE". Without sensing feedback, unknown objects were randomly replaced during the manipulation process. The recordings were taken at different times and concatenated to show all objects. *Bottom*: Ball catching on a flat end-effector without any sensing feedback.

Figure 2. The theory of *Caging in Time* visualized through an example planar pushing task. **I**: A virtual cage, formed by line-shaped bars (grey), robustly pushes an object through a circular trajectory. **II**: Along the time dimension, we can see that only one bar (red) is effectively making contacts at a time, while other bars seem to be unnecessary. **III**: If we collapse all configurations of the effective bars (red) through time, a cage is formed, which can be unrolled into time to achieve the task in I with only one red bar at a time.

manipulation can be achieved by moving all contacts simultaneously. In another extreme situation, let us assume a robot can make one contact with the object at a time, but it is able to switch the contact to other locations infinitely fast so that virtually there are contacts everywhere on the object. Equivalently to the former case, arbitrary manipulation can also be achieved with this virtual cage. Our idea of *Caging in Time* exploits the possibilities between these two extremes: We assume a robot can make one or a few contacts at a time, and it can switch to other contacts fast enough as needed so that, *in time*, it makes a cage. This idea is visualized in Fig. 2 via a planar pushing task, where an object is pushed by a virtual cage (grey) with multiple bars through a circular trajectory. Note that, physically and in time, only one bar (red) is effectively needed at a time to complete the task. Furthermore, by collapsing the configurations of the effective bars through time, a cage is formed and can be unrolled in time to complete the task as if a complete cage has always been there.

We instantiated the *Caging in Time* theory on both quasi-static and dynamic manipulation tasks with a real robot. It is worthwhile to note that, the example instantiations showed that *Caging in Time* can be applied on general state spaces, including geometry-based and energy-based spaces. Without any sensing feedback, *Caging in Time* showed guaranteed task success on all experimented tasks, even when the manipulation is physically affected by in-task perturbations and unknown object shape variations. In comparison with a baseline closed-loop control approach, we show that our framework is similarly accurate, while being significantly more robust against perception uncertainties, as enabled by zero reliance on precise sensing feedback. *Contributions:* The proposed *Caging in Time* concept makes three theoretical contributions: 1) providing a planning paradigm for robust robot manipulation that can entirely remove the effect of perception uncertainties; 2) broadening the traditionally narrowly scoped caging configuration-based manipulation to a more general manipulation framework as enabled by strategic sequential robot motions; and 3) offering a possibility to precisely control robot manipulation without requiring sensing feedback, for the first time, to support robust manipulation in scenarios where perception is not reliable, e.g., object tracking in real-world is typically noisy and can lose track once in a while due to occlusions. *Limitations:* The work reported in this paper is the first step towards general applications of the *Caging in Time* concept. With an emphasis on the derivation of the foundations of the theory, this work does not develop algorithms for addressing general manipulation problems via *Caging in Time* skills. Specifically, we identify the following major limitations of the current scope of this work and state them before the details of the proposed theory: 1) as example instantiations of the proposed theory, the reported manipulation algorithms were designed for implementing *Caging in Time* on specific tasks only; and 2) manipulation physics or dynamics models are analytically derived for the purpose of explicitly verifying the theory, although simulation or learning-based models can be more efficient. Nevertheless, we underscore that the proposed *Caging in Time* framework is a complete theory for general manipulation problems as discussed in Sec. 4.

2 Related Works

Robot Manipulation: Contact properties, object geometrical and physical properties, and perfect object perception, are typically assumed to be sufficiently provided for modeling robot manipulation systems (Suomalainen et al. 2022; Bütepage et al. 2019). As an alternative, learning-based approaches assume that the data used for training the manipulation models are sufficiently covering all potential variations in relevant task domains, and that the perception system used in the task is the same or similar to the ones used in the training set, e.g., similar camera poses (Lee et al. 2019; Kaelbling 2020; Andrychowicz et al. 2020; Kroemer et al. 2021). Along the other line, interactive perception has significantly reduced the requirements on certain prior knowledge and direct perception in manipulation systems (Bohg et al. 2017; Hang et al. 2021), while still assuming reliable perception through some channels to be available for robots to iteratively estimate the state of the manipulation system. Although many manipulation approaches have shown excellent performance, there is always a real-world challenge of unreliable perception, such as noise in object tracking or object occlusions, that will significantly affect the robustness of most manipulation systems. Different from most existing works, this work aims to use caging configurations along the time dimension to eliminate the reliance on, hence the negative effect of, unreliable perception. Note that the goal of this work is not about replacing other planning or control methods for manipulation with *Caging in Time*. Rather, we aim to provide an alternative or supplementary approach alongside others to improve the robustness of robot manipulation under unreliable perception.

Caging Configuration-based Manipulation: Caging configuration was proposed originally as a tool for a team of robots in $SE(2)$ to constrain the motion of a target object, so that the motion of the robot team will determine the motion of the caged object (Pereira et al. 2004; Sudsang and Ponce 2000). Such a concept was later extended into the problem of robot grasping in 2D, for which the goal was to fix the target object within a small region (Rodriguez et al. 2012). By further modeling the topological relationships between robots and objects, hooking or latching-based manipulation was developed based on caging in topological spaces (Stork et al. 2013). More recently, a specific type of planar dexterous manipulation has been enabled by in-hand caging configurations so that the object is manipulated through controlled energy states inside the cage that deform over time (Bircher et al. 2021). With a mobile robot actively trying to escape from a team of other robots, caging configurations have also enabled formation-based motion planning to herd such a robot to a goal region (Song et al. 2021). Essentially, all existing frameworks aim at first constructing a cage to enclose the target object, and then maintain the cage while moving or deforming it to achieve a manipulation task. Lately, caging configuration-based manipulation has been extended to incorporate robot motions to relax the requirements on specific geometric properties of robots or objects (Dong and Pokorny 2024; Dong et al. 2024). This line of research is focused on designing metrics to predict the manipulation robustness, while autonomous caging planning and control are left as open questions.

Challenges with Traditional Caging: The most fundamental limitation of traditional caging is that multiple robots (ZhiDong and Kumar 2002), widely distributed contacts (Rodriguez et al. 2012), or specific geometries of the robots or objects (Varava et al. 2016) are required, making caging a concept not applicable to more general task setups. In this work, with the proposed framework of *Caging in Time*, we show that even with just one robot, we will be able to virtually construct cages in time to satisfy the aforementioned requirements. Additionally, traditional methods require very complex algorithms to verify caging configurations (Varava et al. 2021), which can be even harder for partial cages (Makita and Nagata 2015). As will be seen with *Caging in Time*, since we can, in theory, virtually have our robots or contacts everywhere as needed, caging verification is generally a much simpler problem as we only need to make sure that the object does not penetrate the predefined cage boundaries.

3 Preliminaries

In this section, we first introduce notations in Sec. 3.1 for traditional caging definitions, and then in Sec. 3.2 extend the notations to more general task spaces to enable the derivation of our proposed *Caging in Time* framework.

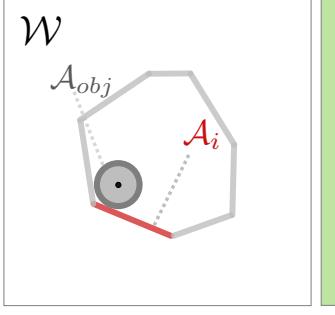### 3.1 *Traditional Caging as Object Closure*

In the scenario of an object caged by multiple robots, we denote the configuration space (C-space) of an object by $\mathcal{C}_{obj}$ and the configuration of an object (e.g., position and orientation) by $c_{obj} \in \mathcal{C}_{obj}$. The C-space and the configuration of the i-th robot are denoted by $\mathcal{C}_i$ and $c_i \in \mathcal{C}_i$, respectively. The object and all the robots share a common workspace $\mathcal{W}$, where $\mathcal{A}_{obj}, \mathcal{A}_i \subset \mathcal{W}$ are the geometries of the object and of the i-th robot in the workspace. As such, the free C-space of the object $\mathcal{C}_{free} \subset \mathcal{C}_{obj}$ can be defined as the set of all possible states where the object does not collide with any robot:

$$\mathcal{C}_{free} := \{c_{obj} \in \mathcal{C}_{obj} \mid \forall i : \mathcal{A}_{obj}(c_{obj}) \cap \mathcal{A}_i(c_i) = \emptyset\} \tag{1}$$
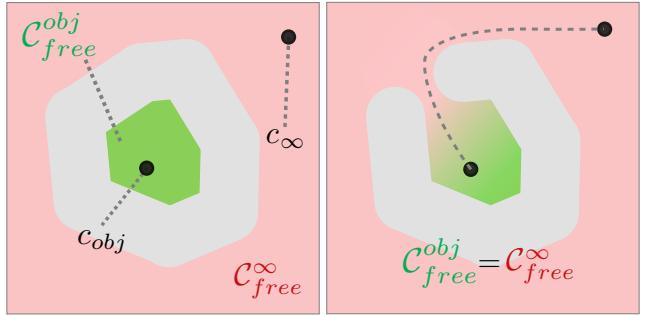
An illustration of the notations above is given in Fig. 3.

The traditional definition of caging, initially termed object closure, was first introduced by ZhiDong and Kumar (2002) as a condition for an object to be trapped by robots. In other words, the caging condition is met when there is no viable path for the object to move from its current configuration to a configuration infinitely far away.

As depicted in Fig. 4, given a point infinitely far away from the current c_{obj} in the C-space $c_\infty \in \mathcal{C}_{obj}$, two distinct sets $\mathcal{C}_{free}^{obj}, \mathcal{C}_{free}^{\infty} \subset \mathcal{C}_{free}$ are defined as follows. $\mathcal{C}_{free}^{obj}$ is the largest subset of $\mathcal{C}_{free}$ within which every configuration c is reachable from the current configuration c_{obj} through a collision-free path; similarly, $\mathcal{C}_{free}^{\infty}$ is the largest subset of $\mathcal{C}_{free}$

Figure 3. An illustration of the workspace $\mathcal{W}$ (left) and the C-space $\mathcal{C}_{obj}$ (right) of a circular object surrounded by seven line robots. *Left*: The red line is the geometry of the i-th robot $\mathcal{A}_i$ and the solid grey circle is the geometry of the object $\mathcal{A}_{obj}$. *Right*: The red region is all the object's configurations at which the object collides with a robot and the green region is the object's free C-space $\mathcal{C}_{free} \subset \mathcal{C}_{obj}$.

Figure 4. An illustration of the traditional caging condition. *Left*: The green region and the red region represent $\mathcal{C}_{free}^{obj}$ and $\mathcal{C}_{free}^{\infty}$, respectively. The object is caged in this case since $\mathcal{C}_{free}^{obj}$ and $\mathcal{C}_{free}^{\infty}$ are not connected. *Right*: A non-caging scenario where $\mathcal{C}_{free}^{obj}$ and $\mathcal{C}_{free}^{\infty}$ become connected. The object may escape following the trajectory shown by the dashed line.

within which every c can reach c_∞:

$$
\begin{aligned}
\mathcal{C}_{free}^{obj} &= \{c \in \mathcal{C}_{free} \mid \text{connected}\,(c, c_{obj})\}, \\
\mathcal{C}_{free}^{\infty} &= \{c \in \mathcal{C}_{free} \mid \text{connected}\,(c, c_{\infty})\}
\end{aligned}
\tag{2}
$$

The caging condition is met if and only if:

$$
\begin{cases}
\mathcal{C}_{free}^{obj} \neq \emptyset \\
\mathcal{C}_{free}^{obj} \cap \mathcal{C}_{free}^{\infty} = \emptyset
\end{cases}
\tag{3}
$$

where the first criterion ensures the existence of a caging configuration, and the second condition guarantees no feasible path connects $\mathcal{C}_{free}^{obj}$ and $\mathcal{C}_{free}^{\infty}$. This traditional concept of caging serves as the foundation for our proposed *Caging in Time* framework.

3.2 Generalized Representations

To enable the derivation of the *Caging in Time* theory, we now generalize the representations and notations from configuration spaces to state spaces. This allows for more comprehensive descriptions of manipulation tasks and accommodates various dynamic scenarios and uncertainties, as opposed to the traditional quasi-static configuration-based caging definitions.

We denote the state space of an object by $\mathcal{S}_{obj}$, which encompasses a wider range of properties than $\mathcal{C}_{obj}$. The state of an object at time t is represented by $\mathbf{q}_t \in \mathcal{S}_{obj}$. Depending on the specific manipulation task, $\mathbf{q}_t$ may include not only configuration (e.g., position and orientation) but also velocity, acceleration, or other relevant properties.

Furthermore, often in real-world systems, the state of the object at time t is not exactly known due to perception uncertainties or modeling simplifications. To account for such uncertainties, we define the **Potential State Set** (PSS) of an object to be the set consisting of all possible states of the object at time t, denoted by $\mathcal{Q}_t \subset \mathcal{S}_{obj}$.
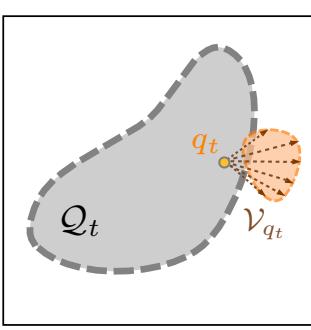
4 Caging in Time

In this section, we will formally introduce the definition of our proposed *Caging in Time* theory. With as few as only a single robot interacting with the object, *Caging in Time* verifies that the object's state remains caged while being manipulated, thereby enabling open-loop manipulation and guaranteeing the desired movement of the manipulated object. As an essential component of our proposed theory, we need to predict the bounded motions of the object, which involves propagating the PSS over time, as will be introduced in Sec. 4.1. Then, we will formally define *Caging in Time* in Sec. 4.2.

For more intuitive illustrations of the proposed concepts, figures in this section are sketched in 2D spaces. However, it is important to emphasize that $\mathbf{q}_t$ can take any form and dimension as required by the specific manipulation task. The 2D visualizations are chosen for visual simplicity and do not limit the generality of the state space concept.
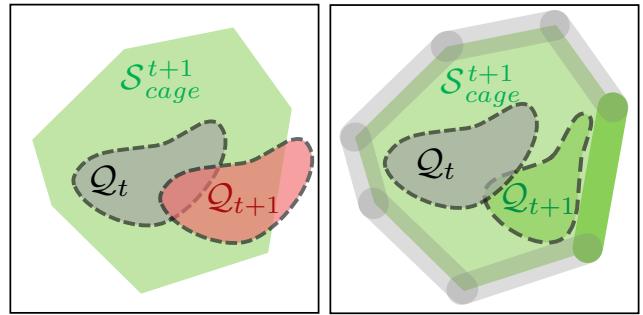
4.1 Propagation of PSS

At time t, for each specific state of the object in the PSS $\mathbf{q}_t \in \mathcal{Q}_t$, we denote the set of all allowed motions that transit the object's state by $\mathcal{V}_{\mathbf{q}_t}$. We define the notation $\mathrm{T}_x \mathcal{M}$ to represent the tangent space of a manifold $\mathcal{M}$ at a point $x \in \mathcal{M}$ in this manifold. Note that $\mathcal{V}_{\mathbf{q}_t} \subset \mathrm{T}_{\mathbf{q}_t} \mathcal{S}_{obj}$, i.e., $\mathcal{V}_{\mathbf{q}_t}$ lies in the tangent space of the object's state space at the state $\mathbf{q}_t$. Furthermore, based on the tangent bundle theory (Jost 2008), we define the motion bundle at time t in Eq. (4), which is the set consisting of all possible state-motion pairs of the object:

$$
\mathcal{QV}_t := \{(\mathbf{q}_t, \mathbf{v}_t) \mid \mathbf{q}_t \in \mathcal{Q}_t, \mathbf{v}_t \in \mathcal{V}_{\mathbf{q}_t}\}
\tag{4}
$$

Figure 5. An illustration of PSS propagation. *Left*: The propagation for a single state $\mathbf{q}_t \in \mathcal{Q}_t \subset \mathcal{S}_{obj}$. With a set of possible motions $\mathcal{V}_{\mathbf{q}_t}$, $\mathbf{q}_t$ can be propagated to a set of different states (the orange region). *Right*: The propagation from the entire PSS $\mathcal{Q}_t$ to $\mathcal{Q}_{t+1}$ by propagating all points in $\mathcal{Q}_t$. The different colors of regions in $\mathcal{Q}_{t+1}$ illustrate the propagation from multiple different $\mathbf{q}_t \in \mathcal{Q}_t$.

Figure 6. A 2D pushing example showing how the PSS propagation is influenced by a robot action. *Left*: A scenario where $\mathcal{Q}_t$ propagates to $\mathcal{Q}_{t+1}$ without robot intervention. $\mathcal{Q}_{t+1}$ will exceed the cage region $\mathcal{S}_{cage}^{t+1}$. *Right*: With a well-selected robot push starting from the pose shown by the green bar towards the center of $\mathcal{S}_{cage}^{t+1}$, $\mathcal{Q}_{t+1}$ will deform to a different shape and stay inside $\mathcal{S}_{cage}^{t+1}$. In this example, the workspace and the object's state space (i.e., 2D position of the object) share the same space $\mathbb{R}^2$, so we overlay the robot pushes (grey bars) onto the object's state space for visualization.

We need to define a propagation function $\pi : \mathcal{QV}_t \mapsto \mathcal{S}_{obj}$ to predict the object's state $\mathbf{q}_{t+1}$ at the next time step given a possible state $\mathbf{q}_t$ and a motion $\mathbf{v}_t$ of the object, i.e., $\mathbf{q}_{t+1} = \pi(\mathbf{q}_t, \mathbf{v}_t)$.

However, the above propagation function only predicts a single state for the next time step. In our *Caging in Time* framework, we need to know all the possible states propagated from the previous step, i.e., the PSS $\mathcal{Q}_{t+1}$. To this end, we extend the propagation function π to another function $\Pi : 2^{\mathcal{QV}_t} \mapsto 2^{\mathcal{S}_{obj}}$ whose domain is the power set of the motion bundle $\mathcal{QV}_t$. As illustrated in Fig. 5, it is used to obtain the entire set of all possible states at time $t + 1$ by propagating every possible $\mathbf{q}_t$:

$$\begin{aligned} \mathcal{Q}_{t+1} &= \Pi(\mathcal{QV}_t) \\ &= \{\mathbf{q}_{t+1} = \pi(\mathbf{q}_t, \mathbf{v}_t) \mid (\mathbf{q}_t, \mathbf{v}_t) \in \mathcal{QV}_t\} \end{aligned} \tag{5}$$

4.2 Caging in Time

Unlike the traditional caging introduced in Sec. 3.1, our *Caging in Time* framework only requires one robot to manipulate the object. The cage is formed over time by switching the single robot to a different configuration and interacting with the object differently at each time step. Fig. 6 shows an example where a robot pusher can push an object from various possible initial locations relative to the object (marked by the grey bars in the right figure). At each time step, by predicting the object's possible motion via the propagation function Π defined in Sec. 4.1, the robot needs to figure out which candidate push to select (e.g., the green bar in the right figure of Fig. 6), to prevent the object from escaping. As such, the object can always be confined to a bounded region as the robot pusher switches to different positions over time and pushes the object in different directions.

Specifically, for every time step t, we want to confine the object's state to be always inside a region $\mathcal{S}_{cage}^t \subset \mathcal{S}_{obj}$ in its state space, and we term this region as the "cage region". The "cage region" $\mathcal{S}_{cage}^t$ does not have to be stationary; it can deform and displace depending on the manipulation task it entails. Moreover, a set of candidate robot actions $\mathcal{U}$ needs to be pre-defined based on the context of the manipulation task. For example, in the planar pushing manipulation scenario shown in Fig. 6, $\mathcal{U}$ can be a set of robot-pushing actions defined by different starting positions and pushing directions of the robot pusher. Such robot actions will affect the possible motion of the object. For example, while the robot is interacting with the object through contact, the object will not be able to move in the directions that penetrate the robot. In other words, for a specific object state $\mathbf{q}_t \in \mathcal{S}_{obj}$, the allowed object motions $\mathcal{V}_{\mathbf{q}_t}$ will be influenced by the robot action $u \in \mathcal{U}$. We use a function U to represent such influence by

$$\mathcal{V}_{\mathbf{q}_t} = U(\mathbf{q}_t, u) \subset \mathrm{T}_{\mathbf{q}_t}\mathcal{S}_{obj} \tag{6}$$

Applying a robot action u may cause the object to move in a direction towards the outside of the cage region $\mathcal{S}_{cage}^{t+1}$ at the next time step. To guarantee that the object's state is always inside $\mathcal{S}_{cage}^t$, for each time step $t = 1, \cdots, T$, there must exist some robot action $u_t \in \mathcal{U}$ that causes all possible states of the object to be contained in $\mathcal{S}_{cage}^{t+1}$ after the robot action is executed. As such, the definition of *Caging in Time* is formally given as follows.

Definition 4.1. *For a time-varying cage $\mathcal{S}_{cage}^t$ in an object's state space where $t = 1, \cdots, T$,* Caging in Time *is achieved if and only if: $\forall t = 0, 1, \cdots, T - 1$, $\exists u_t \in \mathcal{U}$ such that*

$$\begin{aligned} \mathcal{QV}_t &= \{(\mathbf{q}_t, \mathbf{v}_t) \mid \mathbf{q}_t \in \mathcal{Q}_t, \mathbf{v}_t \in U(\mathbf{q}_t, u_t)\}, \\ \mathcal{Q}_{t+1} &= \Pi(\mathcal{QV}_t) \subset \mathcal{S}_{cage}^{t+1} \end{aligned} \tag{7}$$

Algorithm 1: The Verification of *Caging in Time*

Input: The time-varying cage region $\mathcal{S}_{cage}^t \subset \mathcal{S}_{obj}$, a sequence of robot actions $\{u_t \in \mathcal{U}\}$
Output: The object is caged in time or not

1   $\mathcal{Q}_0 \leftarrow$ initial PSS of the object
2   **for** $t = 0, 1, \cdots, T-1$ **do**
3   **if** *not* FEASIBLE(u_t) **then**
4   **return** *false*
5   **end**
6     $\mathcal{QV}_t \leftarrow \{(\mathbf{q}_t, \mathbf{v}_t) \mid \mathbf{q}_t \in \mathcal{Q}_t, \mathbf{v}_t \in U(\mathbf{q}_t, u_t)\}$ ▷ Construct the Motion Bundle
7
8     $\mathcal{Q}_{t+1} \leftarrow \Pi(\mathcal{QV}_t)$ ▷ Propagation of PSS via Π
9   **if** $\mathcal{Q}_{t+1} \not\subset \mathcal{S}_{cage}^{t+1}$ **then**
10   **return** *false*
11   **end**
12 **end**
13 **return** *true*

It is worth noting that *Caging in Time* does not guarantee the existence of such robot actions $\{u_t\}_{t=0}^{T-1}$ to cage the object. However, if a sequence of robot actions $\{u_t\}_{t=0}^{T-1}$ is verified to meet the *Caging in Time* condition, it is guaranteed that the object's state always lies inside the time-varying cage $\mathcal{S}_{cage}^t$.

The process of determining whether *Caging in Time* is achieved is detailed in Alg. 1. When the *Caging in Time* condition is met, the object is guaranteed to be caged inside a cage region $\mathcal{C}_{cage}^t$ through a certain sequence of robot actions in an open-loop manner. If we apply this sequence of robot actions, the object will never escape even without any sensing feedback. As aforementioned, the cage region $\mathcal{S}_{cage}^t$ does not have to be static, it is allowed to change over time. For example, if $\mathcal{S}_{cage}^t$ is moved to follow a certain trajectory in the state space of the object over time, the object being caged in time will be able to follow the same trajectory of $\mathcal{S}_{cage}^t$.

Verification of Action Feasibility: In practice, the robot action u_t may not be realizable due to the speed limit or reachability of the robot, which varies on different robot platforms. For example, between the executions of consecutive actions u_{t-1} and u_t, the object may keep moving if without certain assumptions, such as quasi-static motions. This requires the robot to switch to the next action execution fast enough, which may exceed the hardware's speed limit. To this end, at Line 3 of Alg. 1, to achieve *Caging in Time* under a real-world setting, it is necessary to verify the feasibility of the action u_t by taking into account the capability of the actual hardware.

5   Quasi-static Tasks

In this section, we instantiate the *Caging in Time* theory on a quasi-static planar pushing problem. To facilitate the instantiation, we also develop relevant tools for propagating the PSS of the object in a 2D space and generating open-loop robot pushing actions to cage the object. With the *Caging in Time* framework applied to the planar pushing problem, we can enable a single robot to push an object of unknown shape to follow certain trajectories in an open-loop manner, without requiring sensory feedback, exact geometric information, or any other physical properties of the object. This is an instantiation of *Caging in Time* in a geometry-based state space.
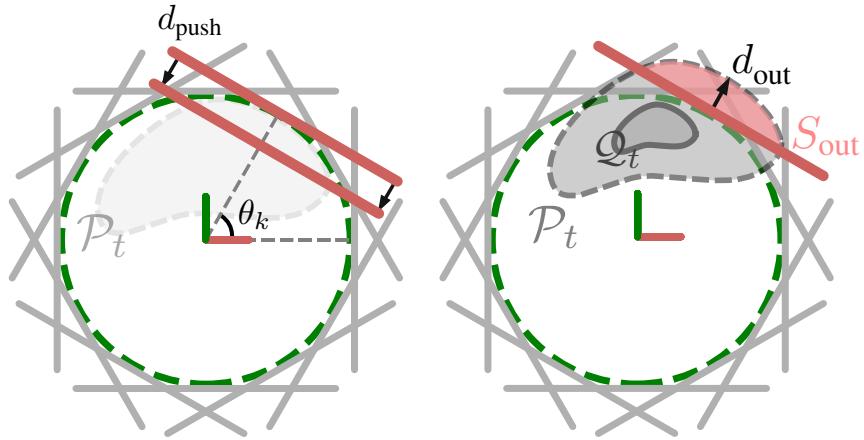
5.1   Problem Statement

The robot is tasked to push an object on a 2D plane. To generalize over different shapes of the object without requiring exact geometric modeling, we represent the object's geometry by a simplified bounding circle with radius r, i.e., $\mathcal{A}_{obj}(\mathbf{q}) = \{\mathbf{p} \in \mathbb{R}^2 \mid \|\mathbf{p} - \mathbf{q}\| \leq r\}$ where $\mathbf{q}$ denotes the position of the object. The radius r is chosen based on the size of the object such that the bounding circle covers the actual shape of the object. The robot is equipped with a fixed-length line pusher, represented by a line segment $\mathcal{L}$ in the 2D space. The object is considered to be in collision with the line pusher when $\mathcal{L}$ intersects the bounding circle of the object, i.e., the distance from $\mathbf{q}$ to the line segment $\mathcal{L}$ is not greater than the radius r.

While the configuration space of a quasi-static 2D object is normally $SE(2)$, requiring both the position and orientation of the object, our use of a bounding circle allows us to simplify the object's state space to its position only: $\mathcal{S}_{obj} \subset \mathbb{R}^2$. As such, the state of the object becomes $\mathbf{q}_t = (x_t, y_t) \in \mathcal{S}_{obj}$. Therefore, both the PSS $\mathcal{Q}_t$ and the cage $\mathcal{S}_{cage}^t$ can be conceptualized as regions within this 2D plane. In this specific planar pushing task, we further define the **Potential Object Area** (POA), $\mathcal{P}_t \subset \mathbb{R}^2$, to be the set consisting of all the points in the workspace that are possibly occupied by the object's geometric shape:

$$\mathcal{P}_t := \bigcup_{\mathbf{q}_t \in \mathcal{Q}_t} \mathcal{A}_{obj}(\mathbf{q}_t) \tag{8}$$

The "cage region" $\mathcal{S}_{cage}^t \subset \mathcal{S}_{obj}$ is then given in order to cage the object's geometric coverage $\mathcal{P}_t$. Specifically, at each time t, we require the object's state to be always caged such that the object's geometric shape (i.e., the bounding circle),

Figure 7. Representation of robot actions and heuristic evaluation for planar pushing. *Left*: The green dashed circle is $\mathcal{S}_{\mathcal{A}}^t$. Each of the $K = 12$ candidate actions (grey bars) is characterized by an angle θ_k. The robot places the line pusher $\mathcal{L}$ (red) at θ_k and pushes towards the circle center with a distance d_{push}. *Right*: Heuristic evaluation, where S_{out} (pink area) is the part of $\mathcal{P}_t$ (the POA) behind the pusher, and d_{out} (black arrow) is the maximum distance to the boundary of $\mathcal{P}_t$.

Algorithm 2: Pushing by Caging in Time

Input: a trajectory as a sequence of waypoints $\mathcal{T} = \{(x_1^d, x_1^d), \cdots, (x_t^d, y_t^d), \cdots, (x_T^d, y_T^d)\}$
Output: The task finished with success or not

1   $\mathcal{Q}_0 \leftarrow \{q_0\}$ ▷ Observe the Object's Position
2   **for** $t = 0, \cdots, T - 1$ **do**
3      $(x_{t+1}^c, y_{t+1}^c) \leftarrow (x_{t+1}^d, y_{t+1}^d)$ ▷ Center of Cage
4      $\mathcal{S}_{cage}^{t+1} = \{(x, y) \mid (x - x_{t+1}^c)^2 + (y - y_{t+1}^c)^2 \le (R - r)^2\}$
5   ▷ Construct the Circular Cage
6      $u_t \leftarrow \text{FindPush}(\mathcal{Q}_t, \mathcal{U}, \mathcal{S}_{cage}^{t+1})$ ▷ Alg. 4
7      $\mathcal{Q}_{t+1} \leftarrow \text{Propagate}(\mathcal{Q}_t, u_t)$ ▷ Alg. 3
8   **if** $\mathcal{Q}_{t+1} \not\subset \mathcal{S}_{cage}^{t+1}$ **then**
9   **return** *false*
10   **end**
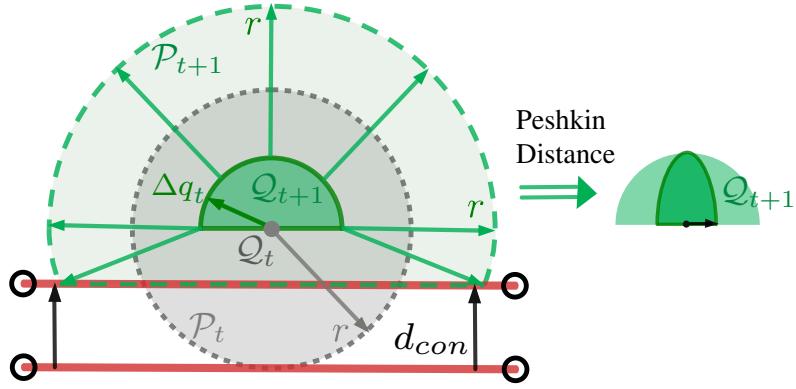11 **end**
12 **return** *true*

---

when at this state, is inside a manually set circle $\mathcal{S}_{\mathcal{A}}^t$. The center of the circle $\mathcal{S}_{\mathcal{A}}^t$ is at (x_t^c, y_t^c) with a constant radius R. As such, the "cage region" $\mathcal{S}_{cage}^t$ in the object's state space is also a circle that shares the same center with $\mathcal{S}_{\mathcal{A}}^t$ and has a radius of $R - r$, i.e., $\mathcal{S}_{cage}^t = \{(x, y) \in \mathbb{R}^2 \mid (x - x_t^c)^2 + (y - y_t^c)^2 \le (R - r)^2\}$. The radius of $\mathcal{S}_{cage}^t$, which is $R - r$, is called the size of the cage.

The robot action is characterized by a scalar angle θ, as shown in the left of Fig. 7. The robot will first place the center of the line pusher at a starting position determined by θ. This starting position is always on the boundary of the circle $\mathcal{S}_{\mathcal{A}}^t$ and can be calculated by $(x_t^c + R \cos \theta, y_t^c + R \sin \theta)$. The line pusher will be oriented to be tangent to the boundary circle of $\mathcal{S}_{\mathcal{A}}^t$. Then, the robot will move the pusher in the direction towards the center of $\mathcal{S}_{\mathcal{A}}^t$ with a fixed distance d_{push} to push the object. We assume there are K candidate robot actions, i.e., $\mathcal{U} = \{\theta_k\}_{k=1}^K$. The value of each action is obtained through $\theta_k = 2k\pi / K$ such that the starting positions of these K candidate actions will evenly surround $\mathcal{S}_{\mathcal{A}}^t$.

As defined in Sec. 4, we require the object's state to be always confined inside the circular cage region $\mathcal{S}_{cage}^t$, i.e., the PSS of the object $\mathcal{Q}_t \subset \mathcal{S}_{cage}^t$. Then, for the object to follow a certain trajectory $\mathcal{T} = \{(x_1^d, x_1^d), \cdots, (x_t^d, y_t^d), \cdots, (x_T^d, y_T^d)\}$ represented by a sequence of waypoints (i.e., desired positions) in the object's state space, we just need to move the center of $\mathcal{S}_{cage}^t$, which is (x_t^c, y_t^c), along the same trajectory. As the object is always confined inside the moving $\mathcal{S}_{cage}^t$ for every time step $t = 1, \cdots, T$, the object's position is guaranteed to follow the trajectory $\mathcal{T}$ with a positional error no greater than the cage size $R - r$.

Next, we develop an algorithm for solving the planar pushing problem based on the proposed *Caging in Time* theory. A high-level description of the developed algorithm is detailed in Alg. 2. At each time step t, an open-loop robot action $u_t \in \mathcal{U}$ to cage the object in time will be found, as will be detailed in Sec. 5.3; and then the PSS of the object will be propagated according to the robot action u_t, which will be discussed in Sec. 5.2. If the propagated PSS is always contained in the cage region $\mathcal{S}_{cage}^t$, the object being pushed will follow the desired trajectory along with the moving cage; otherwise, it is possible that the object is outside the cage, indicating a failure of the task.

Figure 8. The illustration of the bounds for the object's displacement. The maximum displacement of the object equals d_{con}. Therefore, $\mathcal{Q}_t$ containing only the central grey point can propagate into a semi-circle $\mathcal{Q}_{t+1}$ (the semi-circle region in dark green) with a radius equal to d_{con} (i.e., $\|\Delta q_t\| = d_{con}$). Correspondingly, the POA $\mathcal{P}_t$ (the grey bounding circle) with radius r will evolve into the larger translucent green area $\mathcal{P}_{t+1}$. With analysis based on Peshkin Distance, the possible displacement of the object will be further constrained such that Q_{t+1} becomes a semi-ellipse, as shown on the right side.

5.2 Unknown Object Shape and Its Motion

For planar pushing, we represent the object's motion by the displacement of the object's position between adjacent time steps, which we denote as $\mathbf{v}_t = \Delta \mathbf{q}_t = (\Delta x_t, \Delta y_t)$. Then the propagation function π, initially defined in Sec. 4.1, for a single object's configuration and motion as inputs becomes $\pi(\mathbf{q}_t, \mathbf{v}_t) = \pi(\mathbf{q}_t, \Delta \mathbf{q}_t) = \mathbf{q}_t + \Delta \mathbf{q}_t$, which is used to evaluate the propagated object's configuration q_{t+1} at the next step.
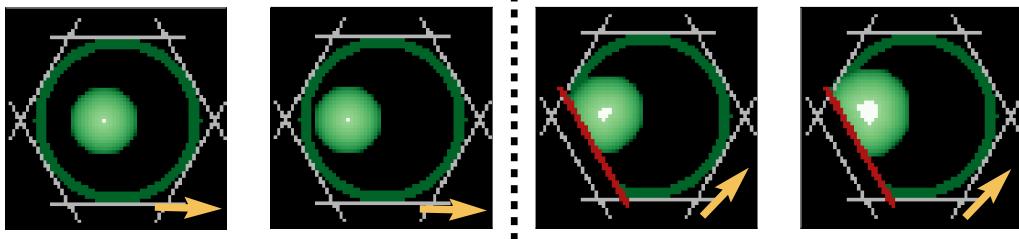
For a specific object's configuration $\mathbf{q}_t$, the set of all possible motions $\mathcal{V}_{\mathbf{q}_t}$ is modeled by the interaction between the object and the robot's line pusher $\mathcal{L}$ while the action u_t is being executed, as represented by the function U defined in Eq. (6). The line pusher approaches the object with a straight-line distance d_{push} while u_t is being executed. If the distance between the object's configuration and the line pusher $\mathbf{d}(\mathbf{q}_t, \mathcal{L})$ is greater than $r + d_{push}$, the object's geometric shape (i.e., the bounding circle) will not collide with the line pusher. The object will not move at all, and thus $\mathcal{V}_{\mathbf{q}_t} = \{(0,0)\}$. However, if the object's bounding circle collides with the line pusher, which indicates the object is likely displaced by the line pusher, we denote $d_{con} \leq d_{push}$ as the moving distance of the line pusher after contacting the bounding circle of the object. As shown in Fig. 8, we can derive a bound for this displacement using the friction theory and Peshkin Distance Peshkin and Sanderson (1988) as follows:

$$
\begin{aligned}
&\mathcal{V}_{\mathbf{q}_t} = U(\mathbf{q}_t, u_t) = \\
&\text{if } \mathbf{d}(\mathbf{q}_t, \mathcal{L}) > r + d_{push} \text{ or } u_t \text{ is } \textit{None}: \\
&\quad \{(0,0)\} \\
&\text{otherwise:} \\
&\quad \left\{ \mathbf{v}_t = R(u_t) \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \,\middle|\, \frac{(\Delta x)^2}{d_{con}^2} + \frac{(\Delta y)^2}{\left(\frac{d_{con}}{2}\right)^2} \leq 1, \Delta x \leq 0 \right\}
\end{aligned}
\tag{9}
$$

where $R(u_t) \in SO(2)$ is a rotation matrix of the angle u_t. The details of the derivation are given in the Appendix. 8.1.

The PSS $\mathcal{Q}_t$ and the set of potential motions $\mathcal{V}_{\mathbf{q}_t}$ can both include an infinite number of points in the continuous space, making the propagation of PSS intractable. To this end, we implement the propagation of PSS in a discretized manner, similar to processing a binary image. Specifically, we construct a binary image $\mathcal{I}_t \in \mathbb{R}^{H \times W}$ at each time step t with height H and width W, centered at (x_t^c, y_t^c), i.e., the center of the cage. We use $\mathcal{I}_t(x, y)$ to denote the intensity value of the pixel corresponding to the point (x, y). An intensity value of 1 signifies the presence of the point (x, y) in the PSS of the object, and 0 indicates the absence, i.e., $\mathcal{I}_t(x, y) = 1 \iff (x, y) \in \mathcal{Q}_t$ and $\mathcal{I}_t(x, y) = 0 \iff (x, y) \notin \mathcal{Q}_t$. Fig. 9 shows several example images of $\mathcal{P}_t$ enhanced with other features including visuals of the robot actions, the cage, the object's POA, etc. In the figure, the white pixels (surrounded by the green pixels which are the POA of the object) correspond to the object's PSS, and the pixels in other colors indicate the absence of PSS.

With this discretized representation, the PSS of the object is implemented by a finite set of pixels on $\mathcal{I}_t$. We also do a similar discretization for the possible motion set $\mathcal{V}_{\mathbf{q}_t}$. As such, to propagate the PSS on the discretized space, we just enumerately propagate every pixel in the PSS with every motion in the discretized finite set of $\mathcal{V}_{\mathbf{q}_t}$. As mentioned, $\mathcal{I}_t$ is always centered with the moving cage. So we need to deal with an offset of (x_t^c, y_t^c) whenever we convert between a pixel location on $\mathcal{I}_t$ and its actual coordinates in the original continuous space. Alg. 3 outlines the discretized implementation of the PSS propagation.

Figure 9. An example showing how the PSS (white region) and the POA (green region) evolve in the discretized space relative to a moving cage. The moving direction of the cage is shown by the yellow arrows. *Left*: Without robot intervention, the PSS and the POA remain unchanged. However, they will translate to the left since the cage is moving towards the right. *Right*: If the POA is predicted to be moving outside of $\mathcal{S}_A^t$ (the dark green circle), a selected line pusher (red) will interact with the POA from the left bottom to deform it to a different shape.

Algorithm 3: Propagate($\cdot$)

Input: The PSS $\mathcal{Q}_t$, a robot action $u_t \in \mathcal{U}$
Output: The PSS at the next time step $\mathcal{Q}_{t+1}$

1   $(x_{t+1}^c, y_{t+1}^c) \leftarrow$ the center of the cage $\mathcal{S}_{cage}^{t+1}$
2   $\mathcal{Q}_{t+1} \leftarrow \{\}$
3   $\mathcal{I}_{t+1} \leftarrow$ initialized with zeros
4   **for** $\mathbf{q}_t \in \mathcal{Q}_t$ **do**
5     $\mathcal{V}_{\mathbf{q}_t} \leftarrow U(\mathbf{q}_t, u_t)$ $\triangleright$ Eq. (9)
6   **for** $\mathbf{v}_t = (\Delta x, \Delta y) \in \text{DISCRETIZED}(\mathcal{V}_{\mathbf{q}_t})$ **do**
7       $\mathcal{I}_{t+1}(x_t + \Delta x_t - x_{t+1}^c, y_t + \Delta y_t - y_{t+1}^c) \leftarrow 1$
8       $\mathcal{Q}_{t+1} \leftarrow \mathcal{Q}_{t+1} \cup \{(x_t + \Delta x_t, y_t + \Delta y_t)\}$
9   **end**
10 **end**
11 **return** $\mathcal{Q}_{t+1}$

Algorithm 4: FindPush($\cdot$)

Input: The PSS $\mathcal{Q}_t$, the set of all candidate robot actions $\mathcal{U}$, the cage region $\mathcal{S}_{cage}^{t+1}$
Output: The selected robot action $u_t \in \mathcal{U}$ to be taken

1   **if** $\mathcal{Q}_t \subset \mathcal{S}_{cage}^{t+1}$ **then**
2   **return** *None*
3   **end**
4   **for** $k = 1, \cdots, K$ **do**
5     $h^k \leftarrow \lambda_1 S_{out}^k + \lambda_2 \left(d_{out}^k\right)^2$ $\triangleright$ Evaluate Heuristics
6   **end**
7   $\mathcal{U}' \leftarrow \{\text{top five } \theta_k \text{ with greatest } h^k\}$
8   $u_t \leftarrow \arg\min_{\theta \in \mathcal{U}'} |\theta - u_{t-1}|$ $\triangleright$ Closest to u_{t-1}
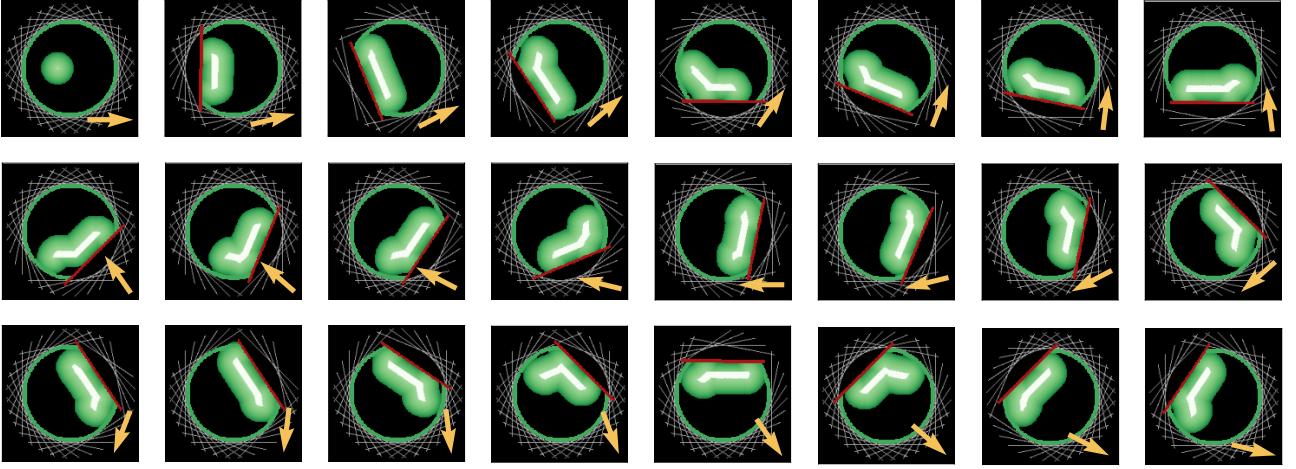9   **return** u_t

---

5.3 Cage the Push in Time

At each time step t, we need to select one robot pushing action u_t from all the candidates $\mathcal{U}$. A naive approach is simulating every action in $\mathcal{U}$, propagating the PSS for every action, and then selecting one of the actions that keep the PSS inside the cage $\mathcal{S}_{cage}^{t+1}$. However, propagating the PSS can be computationally expensive, especially when the number of candidate actions K is large. Therefore, we propose a heuristic-based strategy without requiring the PSS propagation for every action being considered, as detailed in Alg. 4. It is worth mentioning that the heuristics-based strategy is developed to make the action selection more efficient, alleviating the burden of brute-force search over all possible actions. It does not conflict with the completeness of the definition of *Caging in Time*. If the heuristically selected pushing actions are verified to meet the *Caging in Time* condition, the object's state is still guaranteed to be always caged. However, this work does not focus on developing an optimal strategy for action selection, and we admit a better strategy might be developed in the future.
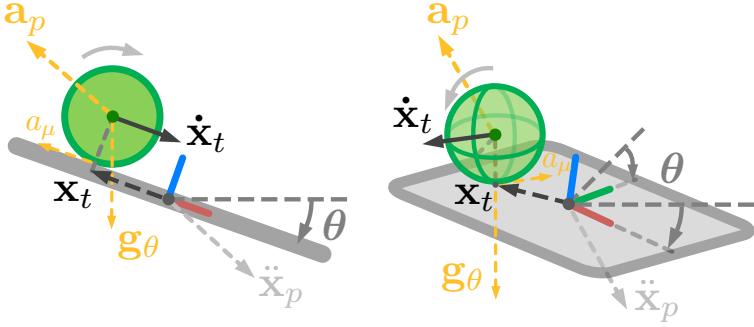
As the cage moves from $\mathcal{S}_{cage}^t$ in the previous time step to $\mathcal{S}_{cage}^{t+1}$ along with the desired trajectory, if the PSS is still inside the moving cage, i.e., $\mathcal{Q}_t \subset \mathcal{S}_{cage}^{t+1}$, no robot action will be needed and Alg. 4 will return a *None*. However, if $\mathcal{Q}_t \not\subset \mathcal{S}_{cage}^{t+1}$, it indicates that the PSS of the object will go outside the cage if no robot action is taken. In this case, we have to select one robot action to move and deform the PSS to keep the object being caged.

Recall that in Sec. 5.1, each candidate action in $\mathcal{U}$ is represented by an angle θ_k, which determines the starting pose of the line pusher. For each candidate action in $\mathcal{U}$ with an index $k = 1, \cdots, K$, we will evaluate a heuristic score by $h^k = \lambda_1 S_{out}^k + \lambda_2 \left(d_{out}^k\right)^2$ where λ_1 and λ_2 are two weighting factors. As shown in the right of Fig. 7, S_{out}^k is the POA area behind the line pusher when we place the pusher at a starting pose determined by the angle θ_k, and d_{out}^k is the furthest distance from behind the line pusher to the POA boundary. It is worth noting that calculating the heuristics h^k for each action does not require the propagation of PSS, i.e., the PSS remains $\mathcal{Q}_t$. Intuitively, a large h^k indicates that a large portion of the PSS will go outside the cage if the k-th action is not taken.

The top five candidate actions with the highest heuristic scores, which compose a set $\mathcal{U}'$, will be further considered for selecting the best one. From these five candidates in $\mathcal{U}'$, the one closest to the previous action u_{t-1} (i.e., the angle of the last push) will be chosen to propagate the PSS to the next step. This mechanism reduces the reorientation of the line pusher at adjacent time steps, which can largely facilitate motion efficiency when implemented on a real robot manipulator. Under the quasi-static setting, we know that the object does not move when not interacting with the robot pusher. This assumption ensures that the physical robot can switch to the subsequent pushing action in time, without requiring a sufficiently fast speed of operation. Therefore, the verification of action feasibility (Line 3 of Alg. 1) is always satisfied. Fig. 10 demonstrates an

Figure 10. The propagation of PSS (white region) and POA (green region) when the cage is following a circular path. Each image is centered at the center of the cage. The yellow arrow in each image represents the motion direction of the cage. In this example, $K = 32$ candidate actions are used, represented by the grey bars. With a different pushing action selected (red bar) for each step, the object is always caged in time, and the POA always stays inside $\mathcal{C}_\mathcal{A}^t$ (the dark green circle).

Figure 11. Illustration of the dynamic ball balancing system for plate dimension $n = 1$ and $n = 2$. *Left*: Physical setup when $n = 1$. $\mathbf{x}_t$ and $\dot{\mathbf{x}}_t$ represent the ball's position and velocity respectively. The plate's tilt angle is denoted by θ. Accelerations acting on the ball include gravity $\mathbf{g}_\theta$, $\mathbf{a}_p$ induced by the plate's acceleration $\ddot{\mathbf{x}}_p$, and rolling friction a_μ. *Right*: Physical setup when $n = 2$, where the notations are the same with $n = 1$ case. Note that $\boldsymbol{\theta}$ here is the tilt angle vector that contains two angles.

overall process of how the PSS and POA evolve as the cage moves, with strategically selected pushing actions ensuring the object remains caged throughout the task.

6 Dynamic Tasks

In this section, we extend our *Caging in Time* theory from quasi-static tasks to dynamic scenarios where we instantiate the same framework on a challenging dynamic ball balancing problem. Tools are developed for propagating the ball's PSS of its position and velocity, as well as for generating open-loop robot actions to keep the ball balanced while the robot end-effector follows different trajectories without requiring any sensory feedback. This is an instantiation of *Caging in Time* in an energy-based state space.

6.1 Problem Statement

The robot is tasked with balancing a rolling object on a tilting plate (end-effector) while the plate follows different trajectories in the workspace. The plate is an n-dimensional flat surface in a workspace of dimension $n + 1$, where $n = 1$ or 2. Each dimension of the plate has a size ranging from $-l$ to l. The object has mass m, radius r_b, an estimated rolling friction coefficient μ_r, and an estimated moment of inertia I_b. We assume that the object only rolls on the plate without slipping. Fig. 11 illustrates example setups where $n = 1$ and 2. The plate's position is denoted as $\mathbf{x}_p \in \mathbb{R}^{n+1}$ and its translational acceleration as $\ddot{\mathbf{x}}_p \in \mathbb{R}^{n+1}$. Its orientation is characterized by a tilt angle vector $\boldsymbol{\theta} \in \mathbb{R}^n$. While only $\boldsymbol{\theta}$ and $\ddot{\mathbf{x}}_p$ are initially defined in the world frame (inertial), all other definitions and models are made in the plate frame (non-inertial). In this case, we introduce the ball's non-inertial acceleration $\mathbf{a}_p \in \mathbb{R}^{n+1}$ and gravity $\mathbf{g}_\theta \in \mathbb{R}^{n+1}$ induced by $\ddot{\mathbf{x}}_p$ and $\boldsymbol{\theta}$ so that all the calculations can be done in the plate frame, as shown in Fig. 11.

The state of the object at time t is defined as $\mathbf{q}_t = (\mathbf{x}_t, \dot{\mathbf{x}}_t) \in \mathbb{R}^{2n}$, where $\mathbf{x}_t \in \mathbb{R}^n$ is the position vector of the object relative to the plate center, and $\dot{\mathbf{x}}_t \in \mathbb{R}^n$ is its velocity vector. In this dynamic system, the state space is denoted as $\mathcal{S}_{obj} \subset \mathbb{R}^{2n}$, which encompasses all the possible states that the object can have in the plate frame.

To represent the distribution of states in a continuous way, we define the probability density of the object being in state $\mathbf{q}_t$ as $p(\mathbf{q}_t) = p(\mathbf{x}_t, \dot{\mathbf{x}}_t)$. The PSS $\mathcal{Q}_t$ is represented as the set of all possible states with non-zero probability density:

$$\mathcal{Q}_t = \{\mathbf{q}_t \in \mathcal{S}_{obj} | p(\mathbf{q}_t) > 0\} \tag{10}$$

After defining PSS, a "cage region" is needed to constrain these states. Unlike quasi-static tasks where the cage is defined geometrically (Sec. 5.1), dynamic systems require a different approach because their state space encompasses both configuration and velocity. For such systems, energy provides a powerful and intuitive framework for representing and constraining motion-inclusive states. Here we define the cage region in the state space based on system energy:

$$\mathcal{S}_{cage}^t = \{\mathbf{q}_t \in \mathbb{R}^{2n} | E(\mathbf{q}_t, \mathbf{g}_\theta, \mathbf{a}_p) < E_{max}(\mathbf{g}_\theta, \mathbf{a}_p)\} \tag{11}$$

The total energy of the system $E(\mathbf{q}_t, \mathbf{g}_\theta, \mathbf{a}_p)$, for a given state $\mathbf{q}_t = (\mathbf{x}_t, \dot{\mathbf{x}}_t)$ is expressed as:

$$E(\mathbf{q}_t, \mathbf{g}_\theta, \mathbf{a}_p) = E_k(\dot{\mathbf{x}}_t) + E_{ve}(\mathbf{x}_t) + E_p(\mathbf{x}_t, \mathbf{g}_\theta, \mathbf{a}_p) \tag{12}$$

where E_k, E_{ve}, and E_p represent the kinetic energy, virtual elastic energy, and potential energy in the plate frame.

To establish an upper bound on the allowable energy that ensures the object cannot "escape" from the plate, we consider an extreme scenario where the object is statically positioned at the edge of the plate as viewed from the plate's frame. This maximum allowable energy, E_{max}, is defined as:

$$E_{max}(\mathbf{g}_\theta, \mathbf{a}_p) = E_{ve}(l) + E_p(l, \mathbf{g}_\theta, \mathbf{a}_p) \tag{13}$$

If the object's total energy exceeds E_{max} at any time step, it would have enough energy to move beyond the plate's boundaries, even if its current position is within the plate. By maintaining the object's energy under the condition $E(\mathbf{q}_t, \mathbf{g}_\theta, \mathbf{a}_p) < E_{max}(\mathbf{g}_\theta, \mathbf{a}_p)$, we ensure it remains within the plate for all possibilities throughout the task, effectively creating an energy-based "cage".

This energy-based definition of the cage region can capture both position and velocity constraints and establish boundaries that prevent the object from leaving the plate to effectively cage it in the state space and enable robust dynamic object manipulation.

The control action of the robot is defined by adjusting the tilt angle vector $\boldsymbol{\theta}$ of the plate by a continuous variable $\mathbf{u}_t = d\boldsymbol{\theta}(t) \in \mathbb{R}^n$, representing the change rate of the tilt angles. The translation motion of the plate is guided by a predetermined trajectory $\mathcal{T} = \{\mathbf{x}_p(t)\}_{t=0}^T$ in the world frame, where $\mathbf{x}_p(t) \in \mathbb{R}^{n+1}$ represents the position of the plate's center in the workspace. The plate's acceleration trajectory $\mathcal{T}_a = \{\ddot{\mathbf{x}}_p(t)\}_{t=0}^T$ is then calculated using numerical differentiation of the velocity, which in turn is derived from the position trajectory $\mathcal{T}$. According to the theory defined in Sec. 4, our objective is to maintain the object's state within the energy-constrained cage $\mathcal{S}_{cage}^t$ while the plate follows its prescribed trajectory $\mathcal{T}$, ensuring that the PSS of the object satisfies $\mathcal{Q}_t \subset \mathcal{S}_{cage}^t$ at all times.

Next, we develop an algorithm for solving the dynamic ball balancing problem based on the proposed *Caging in Time* theory. At each time step t, we compute the maximum allowable energy, calculate the weighted average energy of the current PSS, and solve an optimization problem to find the optimal control input $\mathbf{u}_t = d\boldsymbol{\theta}(t)$ that keeps the ball caged in time. We then propagate the PSS according to the system dynamics and check if the energy constraint and position constraint are satisfied. If either constraint is violated at any time step, the task is considered failed.

6.2 Ball Dynamics and PSS Propagation

As aforementioned, the system state at time t is denoted as $\mathbf{q}_t = (\mathbf{x}_t, \dot{\mathbf{x}}_t)$ with its corresponding probability density $p(\mathbf{q}_t)$. We represent the system's motion between adjacent time steps as $\mathbf{v}_t = \Delta\mathbf{q}_t = (\dot{\mathbf{x}}_t \Delta t, \ddot{\mathbf{x}}_t \Delta t)$, with associated probability $p(\mathbf{v}_t)$. As defined in Eq. (6), the set of potential motions $\mathcal{V}_{\mathbf{q}_t}$ at a certain $\mathbf{q}_t$ can be given as:
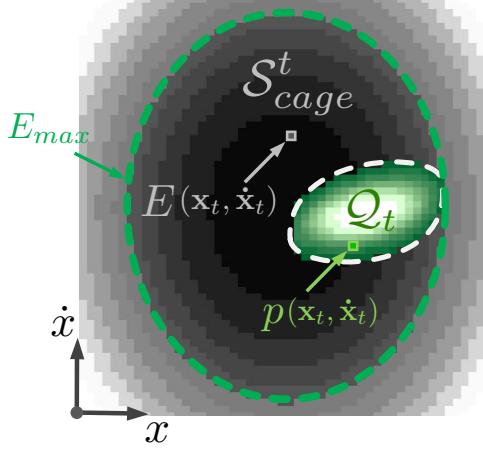
$$\mathcal{V}_{\mathbf{q}_t} = U(\mathbf{q}_t, u_t) = \{\mathbf{v}_t \mid \mathbf{v}_t = (\dot{\mathbf{x}}_t \Delta t, \ddot{\mathbf{x}}_t \Delta t)\} \tag{14}$$

The propagation function π that propagate from $\mathbf{q}_t$ to $\mathbf{q}_{t+1}$ for a single state-motion pair $(\mathbf{q}_t, \mathbf{v}_t)$ is given with the joint probability $p(\mathbf{q}_t, \mathbf{v}_t)$ by:

$$\pi(\mathbf{q}_t, \mathbf{v}_t) = (\mathbf{x}_t + \dot{\mathbf{x}}_t \Delta t, \dot{\mathbf{x}}_t + \ddot{\mathbf{x}}_t \Delta t) \tag{15}$$

Note that π is a deterministic function for state transition. Unlike Sec. 5.2, here we propagate the PSS by updating its probability distribution. Since the joint probability of the state-motion pair can be expressed as $p(\mathbf{q}_t, \mathbf{v}_t) = p(\mathbf{q}_t) \cdot p(\mathbf{v}_t | \mathbf{q}_t)$, the probability $p(\mathbf{q}_{t+1})$ can then be derived as:

$$
\begin{aligned}
p(\mathbf{q}_{t+1}) &= \iint_{\mathbf{q}_t \in \mathcal{Q}_t, \mathbf{v}_t \in \mathcal{V}_{\mathbf{q}_t}} p(\mathbf{q}_t, \mathbf{v}_t) \mathbb{1}(\pi(\mathbf{q}_t, \mathbf{v}_t) = \mathbf{q}_{t+1}) d\mathbf{v}_t d\mathbf{q}_t \\
&= \int_{\mathbf{q}_t \in \mathcal{Q}_t} p(\mathbf{q}_t) \int_{\mathbf{v}_t \in \mathcal{V}_{\mathbf{q}_t}} p(\mathbf{v}_t | \mathbf{q}_t) \mathbb{1}(\pi(\mathbf{q}_t, \mathbf{v}_t) = \mathbf{q}_{t+1}) d\mathbf{v}_t d\mathbf{q}_t
\end{aligned} \tag{16}
$$

Figure 12. Discretized representation of the state space $\mathcal{S}_{obj}$ with energy map. Each pixel of the discretized map represent a state $(\mathbf{x}_t, \dot{\mathbf{x}}_t)$ of the ball. The cage region $\mathcal{S}_{cage}^t$ is bounded by the maximum energy E_{max}. The gradient gray area shows energy value $E(\mathbf{x}_t, \dot{\mathbf{x}}_t)$ at different states. And the gradient green area shows PSS $\mathcal{Q}_t$ with the ball's probability density $p(\mathbf{x}_t, \dot{\mathbf{x}}_t)$ at different states.

where the decomposition of $p(\mathbf{v}_t|\mathbf{q}_t)$ can be given as:

$$
\begin{aligned}
p(\mathbf{v}_t|\mathbf{q}_t) &= p(\dot{\mathbf{x}}_t|\mathbf{q}_t) \cdot p(\ddot{\mathbf{x}}_t|\mathbf{q}_t) \\
&= p(\ddot{\mathbf{x}}_t|\mathbf{q}_t) \\
&= p(\ddot{\mathbf{x}}_t)
\end{aligned}
\tag{17}
$$

where $p(\dot{\mathbf{x}}_t|\mathbf{q}_t) = 1$ since $\dot{\mathbf{x}}_t$ is fully determined by $\mathbf{q}_t$. To calculate $p(\ddot{\mathbf{x}}_t)$, we need to consider the system dynamics. Based on the physical properties of the ball mentioned in Sec. 6.1, we can define the set of potential accelerations $\mathcal{A}_{\mathbf{q}_t}$ for a given state $\mathbf{q}_t$ and control input u_t as:

$$
\begin{aligned}
\mathcal{A}_{\mathbf{q}_t} = \Big\{ \ddot{\mathbf{x}}_t \,|\, &\ddot{\mathbf{x}}_t = f(\mathbf{q}_t, u_t, \eta_m, \boldsymbol{\eta}_p, \eta_\mu), \\
&\eta_m \sim \mathcal{N}(0, \sigma_m^2), \\
&\boldsymbol{\eta}_p \sim \mathcal{N}(\mathbf{0}, \Sigma_p), \\
&\eta_\mu \sim \mathcal{N}(0, \sigma_\mu^2) \Big\}
\end{aligned}
\tag{18}
$$

where $f(\cdot)$ represents the system dynamics equation, detailed in the Appendix 8.2.1. The terms η_m, $\boldsymbol{\eta}_p$, and η_μ represent uncertainties in mass, plate acceleration, and friction coefficient, respectively.

Leveraging the closure property of Gaussian distributions under linear operations, we can conclude that the acceleration $\ddot{\mathbf{x}}_t$, being a linear combination of Gaussian-distributed uncertainties, also follows a Gaussian distribution:

$$
\ddot{\mathbf{x}}_t \sim \mathcal{N}(\boldsymbol{\mu}_{\ddot{\mathbf{x}}}, \boldsymbol{\Sigma}_{\ddot{\mathbf{x}}})
\tag{19}
$$

where $\boldsymbol{\mu}_{\ddot{\mathbf{x}}}$ and $\boldsymbol{\Sigma}_{\ddot{\mathbf{x}}}$ are derived from the system dynamics equation and the distributions of the uncertainty terms.

Given this Gaussian distribution of $\ddot{\mathbf{x}}_t$, we can now compute $p(\ddot{\mathbf{x}}_t)$ for any $\ddot{\mathbf{x}}_t \in \mathcal{A}_{\mathbf{q}_t}$, which allows us to further calculate $p(\mathbf{q}_{t+1})$ based on Eq. (16) and Eq. (17).

As the probability of $p(\mathbf{q}_{t+1})$ is propagated from all the states in $\mathcal{Q}_t$, we can get the PSS at next time step in the same way as Eq. (10):

$$
\mathcal{Q}_{t+1} = \{\mathbf{q}_{t+1} \in \mathcal{S}_{obj} | p(\mathbf{q}_{t+1}) > 0\}
\tag{20}
$$

To numerically implement the PSS propagation described by Eq. (20) and Eq. (16), we employ a discretized representation of the state space, similar to the approach used in the quasi-static case (Sec. 5.2). However, instead of binary values, we use continuous values to represent the probability distribution of the PSS in the state space.

We construct a 2n-dimensional array $\mathcal{I}_t \in \mathbb{R}^{N^{2n}}$ at each time step t, where N is the grid size for each dimension of the state. The array $\mathcal{I}_t$ is centered at the origin in the state space, with $\mathbf{x} \in [-\mathbf{x}_{max}, \mathbf{x}_{max}]$ and $\dot{\mathbf{x}} \in [-\dot{\mathbf{x}}_{max}, \dot{\mathbf{x}}_{max}]$. The state space is discretized uniformly in each dimension, with appropriate grid spacings for position and velocity components. Each element $\mathcal{I}_t(\mathbf{q}_t)$ represents the probability of the occurrence of the discretized version of state $\mathbf{q}_t$, derived from $p(\mathbf{q}_t)$. Fig. 12 shows a visual illustration under the above numerical representation when $n = 1$.

The PSS propagation is implemented in Alg. 5, which discretizes the continuous integrals in Eq. (16). For each $\mathbf{q}_t$ in the current PSS $\mathcal{Q}_t$, we compute the set of potential motions $\mathcal{V}_{\mathbf{q}_t}$ using Eq. (14). We then iterate over discretized versions of these motions, updating the probability of the resulting states $\mathbf{q}_{t+1}$ according to Eq. (17):
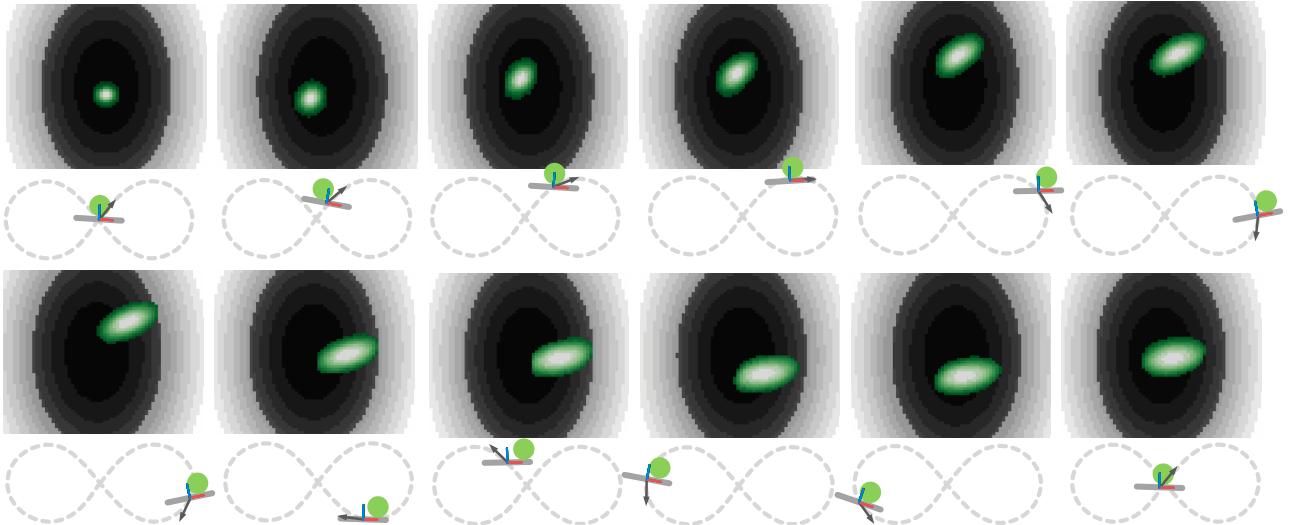

Algorithm 5: DynamicPropagate(·)

Input: Current PSS $\mathcal{Q}_t$, a robot action $u_t \in \mathcal{U}$, plate acceleration $\ddot{\mathbf{x}}_p$, time step Δt
Output: The PSS at next time step $\mathcal{Q}_{t+1}$

1 $\mathcal{Q}_{t+1} \leftarrow \{\}$
2 $\mathcal{I}_{t+1} \leftarrow$ initialized with zeros
3 **for** $q_t \in \mathcal{Q}_t$ **do**
4      $\mathcal{V}_{\mathbf{q}_t} \leftarrow U(\mathbf{q}_t, u_t)$ ▷ Eq. (14)
5   **for** $\mathbf{v}_t = (\dot{\mathbf{x}}_t \Delta t, \ddot{\mathbf{x}}_t \Delta t) \in \text{DISCRETIZED}(\mathcal{V}_{\mathbf{q}_t})$ **do**
6          $\mathbf{q}_{t+1} \leftarrow \text{DISCRETIZED}(\pi(\mathbf{q}_t, \mathbf{v}_t))$
7   **if** $\mathbf{q}_{t+1} \in \mathcal{S}_{obj}$ **then**
8              $P_{q,v} \leftarrow \text{CALCPROB}(\mathbf{q}_t, \mathbf{v}_t)$ ▷ Eq. (17)
9              $\mathcal{I}_{t+1}(\mathbf{q}_{t+1}) \leftarrow \mathcal{I}_{t+1}(\mathbf{q}_{t+1}) + P_{q,v}$
10   **end**
11   **end**
12 **end**
13 $\mathcal{I}_{t+1} \leftarrow \text{NORMALIZE}(\mathcal{I}_{t+1})$
14 **return** $\mathcal{I}_{t+1}$

Figure 13. Sequence of the ball's PSS evolution during dynamic balancing where the plate dimension $n = 1$. *Top rows*: State space representation of the PSS $\mathcal{Q}_t$ (green) within the cage region $\mathcal{S}_{cage}^t$ (dark background), as detailed in Fig. 12. *Bottom rows*: Corresponding configurations of the system in the world frame. The dark gray arrow shows the direction of the plate's translational motion.

$$\mathcal{I}_{t+1}(\mathbf{q}_{t+1}) = \mathcal{I}_{t+1}(\mathbf{q}_{t+1}) + \mathcal{I}_t(\mathbf{q}_t) \cdot p(\mathbf{v}_t | \mathbf{q}_t) \tag{21}$$

where $p(\mathbf{v}_t | \mathbf{q}_t)$ is computed as in Eq. (17). After updating all probabilities, we normalize it to ensure the values of the grid sum to 1 as a valid distribution. For computational efficiency, we apply a threshold to $\mathcal{I}_{t+1}$, setting probabilities below a certain value (e.g., 10^{-3}) to zero. This pixel-based approach provides a computationally efficient representation of the PSS in the state space, avoiding the exponential increase in computational cost associated with sampling methods used in previous caging work in long-horizon tasks (Welle et al. 2021; Makapunyo et al. 2012). Fig. 13 illustrates the propagation and caging of the PSS in the state space during the tracing of an ∞-shaped trajectory for a one-dimensional plate ($n = 1$). The PSS initially manifests as a compact Gaussian distribution, subsequently expanding before converging to a stable size caged by the energy boundary.

6.3 Cage the Ball in Time

To achieve *Caging in Time* for the dynamic ball balancing problem, we employ a control optimization strategy that combines Control Barrier Function (Ames et al. 2019) and Control Lyapunov Function (Blanchini 1995; Anand et al. 2021) to keep the ball remain on the plate while following the desired trajectory.

In our case, Control Barrier Function (CBF) is utilized to implement the concept of "caging" in dynamic setting. Just as a physical cage constraining an object within a defined space, the CBF establishes an energy-based cage that can be characterized by the margin between the maximum energy attainable within the PSS and the predefined energy boundary:

$$h(\mathcal{Q}_t) = E_{max}(\mathbf{g}_\theta, \mathbf{a}_p) - \max_{\mathbf{q}_t \in \mathcal{Q}_t} E(\mathbf{q}_t, \mathbf{g}_\theta, \mathbf{a}_p) \tag{22}$$

This creates an energy barrier and acts as the mathematical representation of our cage. By ensuring that $h(\mathcal{Q}_t) > 0$ at all times, we guarantee that the ball's energy, $E(\mathbf{q}_t, \mathbf{g}_\theta, \mathbf{a}_p)$ at state $\mathbf{q}_t$ as calculated in Eq. (12), never exceeds the maximum allowable energy E_{max} defined in Eq. (13).

To further ensure our control actions maintain this energy barrier, we enforce a CBF condition. This provides a forward-looking constraint that considers not only the current state but also the system's future behavior, rather than simply enforcing $h(\mathcal{Q}_t) > 0$.

$$L_f h(\mathcal{Q}_t) + L_g h(\mathcal{Q}_t) d\boldsymbol{\theta} + \alpha(h(\mathcal{Q}_t)) \geq 0 \tag{23}$$

In this condition, $L_f h(\mathcal{Q}_t)$ and $L_g h(\mathcal{Q}_t)$ are the Lie derivatives of the CBF $h(\mathcal{Q}_t)$ along the vector fields f and g respectively, where f and g are defined in the system dynamics equation $\ddot{\mathbf{x}}_t = f + gu$, detailed in Appendix 8.2.2. $L_f h(\mathcal{Q}_t)$ represents the change in the barrier function due to the natural dynamics of the system, while $L_g h(\mathcal{Q}_t) d\boldsymbol{\theta}$ represents the change due to the control input. The term $\alpha(\cdot)$ is a class $\mathcal{K}$ function that shapes the convergence behavior of the barrier function, ensuring that it increases more rapidly as the system approaches the boundary of the safe set.

Control Lyapunov Functions (CLFs), on the other hand, are used to optimize the control performance by driving the system towards a desired state. In our context, we use a CLF to encourage the ball to stay near the center of the plate and maintain a well-distributed probability state, with both the system's energy and entropy considered:

$$V(\mathcal{Q}_t) = \sum_{\mathcal{I}_t(\mathbf{q}_t)>0} \mathcal{I}_t(\mathbf{q}_t) E(\mathbf{q}_t, \mathbf{g}_\theta, \mathbf{a}_p) - k_S S(\mathcal{Q}_t) \tag{24}$$

where $\mathcal{I}_t(\mathbf{q}_t)$ is the probability value at state $\mathbf{q}_t$ in the discretized probability distribution, $E(\mathbf{q}_t, \mathbf{g}_\theta, \mathbf{a}_p)$ is the system energy at the state $\mathbf{q}_t$, k_S is a weighting factor, and $S(\mathcal{Q}_t)$ is the system entropy.

This CLF balances two objectives: minimizing the expected energy of the system, which tends to keep the ball near the center of the plate, and maximizing the entropy of the probability distribution, which maintains a well-distributed probability state, ensuring both stability and robustness in the face of uncertainties. By minimizing this CLF, we drive the system towards a state where the ball is likely to be near the center of the plate, but with some uncertainty to handle unexpected disturbances.

Similar to CBF condition, we enforce the CLF condition to ensure the CLF decreases over time:

$$L_f V(\mathcal{Q}_t) + L_g V(\mathcal{Q}_t) d\boldsymbol{\theta} + cV(\mathcal{Q}_t) \leq \delta \tag{25}$$

where $L_f V(\mathcal{Q}_t)$ and $L_g V(\mathcal{Q}_t)$ are Lie derivatives of the CLF along the system dynamics and control input respectively, c is the convergence rate, and δ is a relaxation variable. This condition ensures that our control input $d\boldsymbol{\theta}$ drives the system towards the desired behavior at a sufficient rate, while the relaxation variable δ allows for temporary violations of the decrease condition when necessary to satisfy the CBF safety constraint.

By combining the CBF and CLF conditions, we formulate a Quadratic Program (QP) that not only keeps the ball on the plate but also tries to keep it centered and stable.

$$\begin{aligned} \min_{d\boldsymbol{\theta}, \delta} \quad & (d\boldsymbol{\theta})^2 + \lambda\delta^2 \\ \text{s.t.} \quad & L_f h(\mathcal{Q}_t) + L_g h(\mathcal{Q}_t) d\boldsymbol{\theta} + \alpha(h(\mathcal{Q}_t)) \geq 0 \\ & L_f V(\mathcal{Q}_t) + L_g V(\mathcal{Q}_t) d\boldsymbol{\theta} + c(V(\mathcal{Q}_t)) \leq \delta \\ & d\boldsymbol{\theta}_{\min} \leq d\boldsymbol{\theta} \leq d\boldsymbol{\theta}_{\max} \end{aligned} \tag{26}$$

Here, $\lambda > 0$ is a weighting factor that balances the trade-off between minimizing the control effort $(d\boldsymbol{\theta})^2$ and the CLF constraint violation δ^2. The values of $d\boldsymbol{\theta}_{\min}$ and $d\boldsymbol{\theta}_{\max}$ are derived from the hardware torque limits of the robot arm's motors. We utilize the OSQP (Operator Splitting Quadratic Program) solver to efficiently solve this optimization problem at each time step.
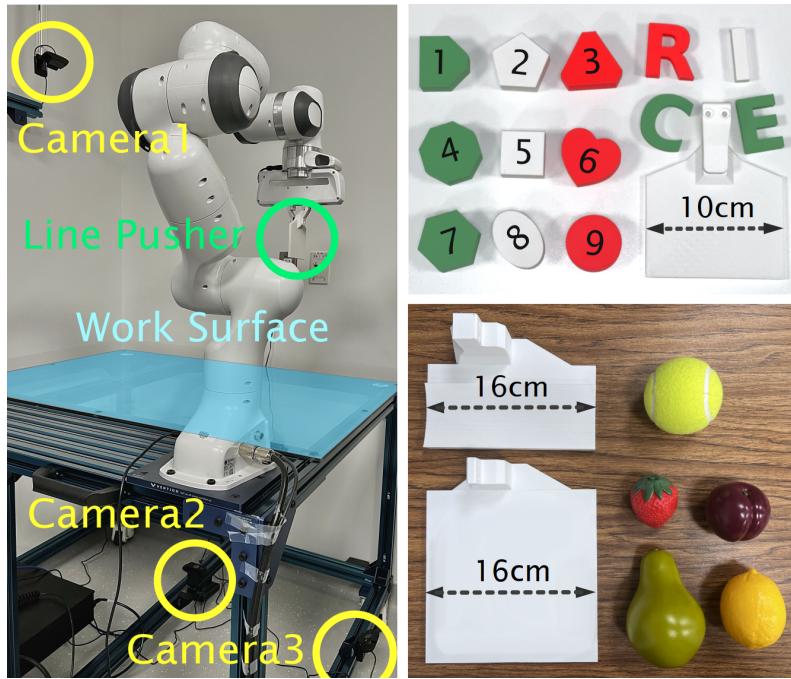
By solving this QP at each time step, we obtain the optimal control input that maintains the ball within the energy-based cage while following the desired trajectory. The detailed implementation is presented in the Appendix 8.2.2.

This approach allows us to cage the ball in time under dynamic conditions on a tilting plate, accounting for the ball's dynamics, uncertainties, and energy constraints. By solving the QP at each time step, we generate a sequence of actions that keeps the ball within the energy-based cage region with the plate following the desired trajectory.

7 Experiments

We conducted extensive experiments to evaluate our *Caging in Time* framework using a Franka Emika Panda robot arm for both quasi-static pushing tasks and dynamic ball balancing tasks with different end-effectors shown in Fig. 14. All algorithms were implemented in Python and executed on a single thread of a 3.4 GHz AMD Ryzen 9 5950X CPU.

Our experimental setup utilized multiple cameras for recording: cameras shown on the left of Fig. 14 for quasi-static tasks, with an additional camera for dynamic tasks. For quasi-static experiments, object tracking was achieved using AprilTags

Figure 14. Experiment setup. *Left*: Robot and camera setup. The robot performs the experiment on a transparent surface (blue) with a pre-installed line pusher (green). We used cameras 2 and 3 below the surface to record (not to track) the trajectories of the objects through April-Tag. Camera 1 is used to record the experiment from an upper view. *Top Right*: The 3D-printed objects used in the experiments. 1. Square with two trimmed tips, 2. Pentagon, 3. Triangle with trimmed tips, 4. Octagon, 5. Square, 6. Heart, 7. Hexagon, 8. Ellipse, 9. Circle, and letter-shaped objects: R, I, C, E. *Bottom Right*: 3D printed plates, tennis ball and plastic fruits for dynamic tasks. The square plate is for ball balancing when dimension $n = 2$, and the thinner one is for ball balancing and catching when dimension $n = 1$, which has a guide rail to constrain the motion of the ball only on the required dimension. The plastic fruits are strawberry (YCB #12), plum (YCB #18), pear (YCB #16) and lemon (YCB #14).

(Olson 2011), while in dynamic scenarios, OpenCV was employed to track the ball's motion. It's important to note that object tracking served only for trajectory visualization and accuracy evaluation and all experiments were conducted in an open-loop manner.

To quantify manipulation precision, we adopted the Mean Absolute Error (MAE) as our primary metric. This was calculated by averaging the absolute distances between the actual and reference trajectories throughout the manipulation process. Through these experiments, we aimed to thoroughly assess the performance, robustness, and versatility of our *Caging in Time* framework across a range of quasi-static and dynamic manipulation tasks.
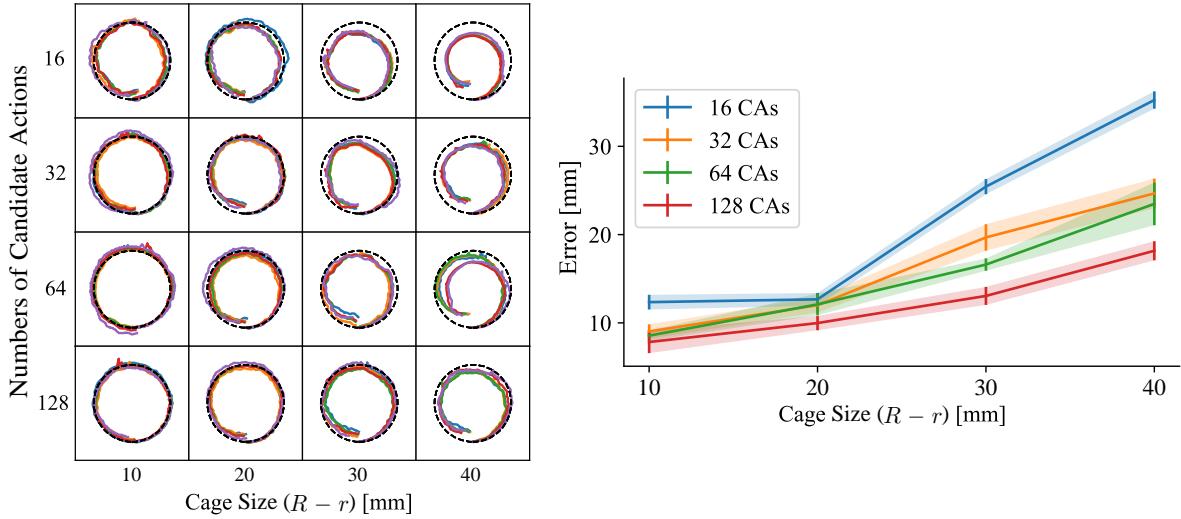
7.1   Quasi-static Tasks

Through quasi-static experiments, we would like to investigate the performance of our framework from three aspects: 1) Without any sensing feedback besides the size and initial position of the bounding circle, can robust planar manipulation be achieved by the proposed *Caging in Time*? 2) If so, what precision can be achieved, and how is the robustness affected by different settings of the framework and outer disturbances? 3) Under imperfect perceptions like positional noise and network lag, can *Caging in Time* surpass closed-loop methods?
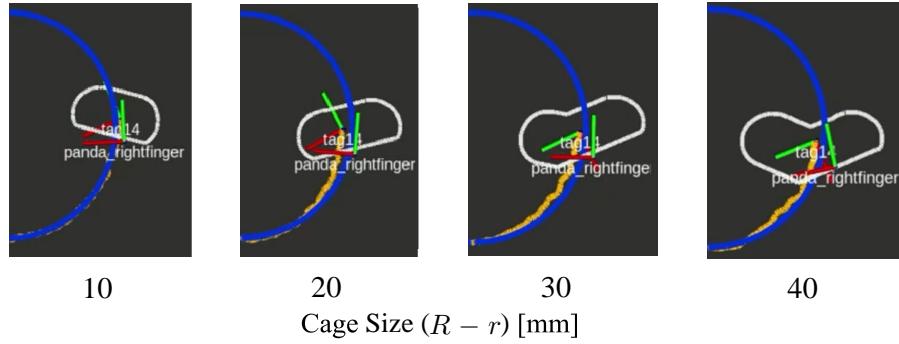
As displayed in Fig. 14, The robot in this task used a 100 mm line pusher, and objects of various shapes were 3D-printed in a similar size, fitting within a bounding circle of radius $r = 25$ mm. The pushing distance of each action was set as $d_{push} = 20$ mm.

7.1.1   Evaluation of Cage Settings The task of this experiment is to push the object through a predefined circular trajectory. This experiment focused on evaluating the precision of our framework by varying two key parameters: $R - r$, the cage size, and K, the number of candidate actions. We selected values for $R - r$ and K from the sets $R - r = \{10, 20, 30, 40\}$ mm and $K = \{16, 32, 64, 128\}$. For each combination of $R - r$ and K, five trials were conducted using five different objects (Object 1-5, see Fig. 14). We generated only one robot action sequence per $(R - r, K)$ setting, following Alg. 2 and applied it to different objects in an open-loop manner. The MAE across these trials was averaged and presented in Fig. 15, which also includes the actual trajectories recorded from all experiments with different settings and different objects.

From the results, we can see that the performance follows two major trends. One is that a larger cage radius tends to increase the average error. The other trend is that an increase in the number of candidate actions generally leads to a decrease in error. As shown in Fig. 16, with larger cage sizes, the POA will grow larger, hence increasing the range the object could possibly deviate from the reference trajectory. Also with more candidate actions, it increases the possibility of finding the optimal action in Alg. 4.

Figure 15. Evaluation results in terms of the cage size $(R - r)$ and number of candidate actions. *Top*: All trajectories (nine trajectories of different objects are overlaid in the plot per circle) *Bottom*: Mean Absolute Error of the shown trajectories under different cage settings. "CA" stands for candidate action.

Figure 16. Visualization of POAs (the white boundaries) in real pushing experiment recordings under various cage sizes.
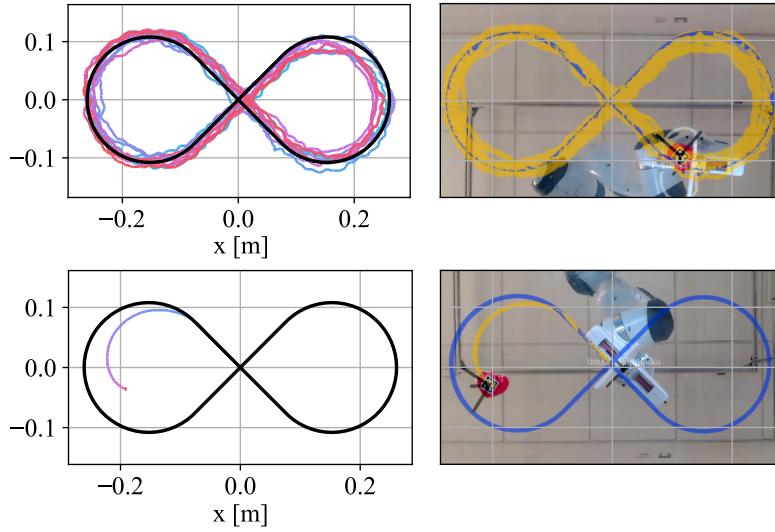
Examining the trajectories in Fig. 15, we observed that all paths remained within their respective cage regions. Notably, trajectories generated by the same action sequence showed similar patterns across different objects. For example, trajectories with 16 candidate actions and a 40 mm cage size $(R - r)$ mostly moved to the left of the reference path, indicating that the action sequence plays a more dominant role in the performance than the object shape.

7.1.2 Why Caging in Time While the line pusher might be perceived as a stable method for pushing an object in an open-loop way, this stability is often compromised by the unpredictability of the object's physical properties and shape, which necessitates the implementation of the *Caging in Time*.
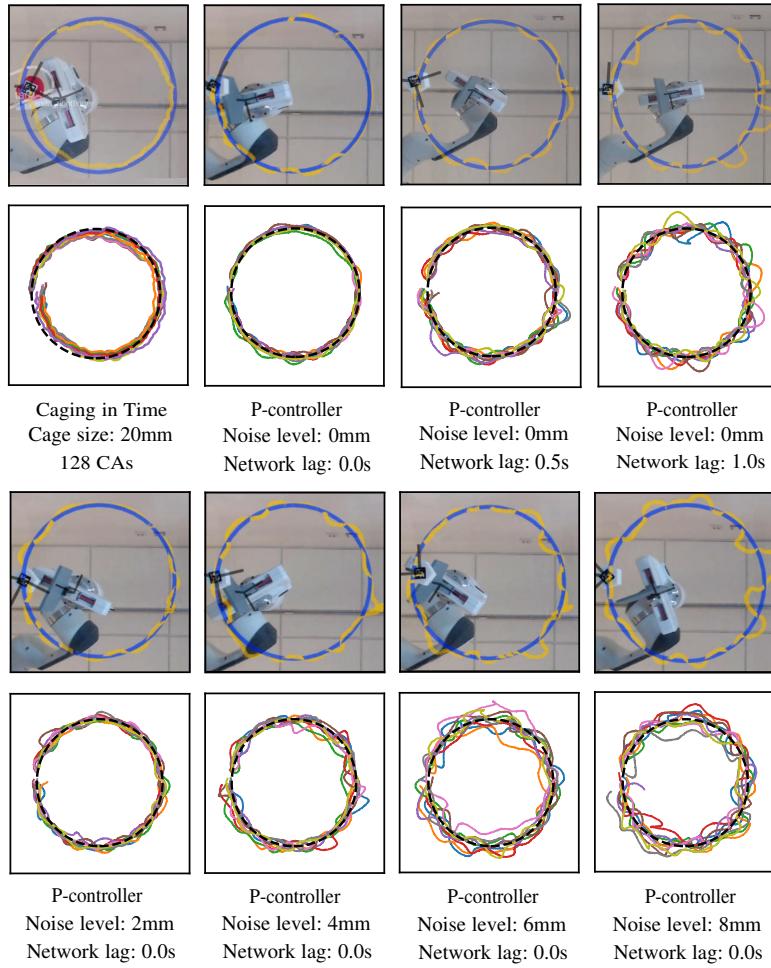
To validate the necessity and robustness of our approach, an object was tasked to navigate a ∞-shaped trajectory repeatedly for 10 cycles. A comparative analysis was conducted against a naïve pushing strategy, where the pusher simply followed the target trajectory with its orientation parallel to the trajectory direction. As shown in Fig. 17, in contrast to the naïve method, where the pusher lost control of the object after a few steps, our approach successfully maintained control over the object after 10 rounds with an average error of 10.09 mm.

7.1.3 Comparison with a Closed-loop Method To further validate the effectiveness of our framework, we conducted a comparative study against a closed-loop pushing method: a proportional controller (P-controller) implemented under the quasi-static assumption. The P-controller works by calculating an error vector from the object's current position to the nearest waypoint on the reference trajectory. This error vector is then scaled by the P-gain (set to 0.5) to determine the direction and magnitude of the pushing action. Specifically, the pushing direction is aligned with the scaled error vector, while the pushing distance is capped at 20 mm for each action, consistent with the d_{push} used in our *Caging in Time* experiments.

We introduced two types of disturbances to simulate real-world scenarios with imperfect perception and communication delays. Firstly, positional noise was added as Gaussian noise with a standard deviation of the noise level in both x and y directions, simulating the effect of imperfect perception, such as that from an RGBD camera under dim lighting conditions. Secondly, to emulate potential packet loss during data transmission or sudden obstruction of the object from the camera's view, we introduced a network lag that for each execution step, there was a 50% probability that the controller would receive the object's position from 0.5 or 1 second ago.
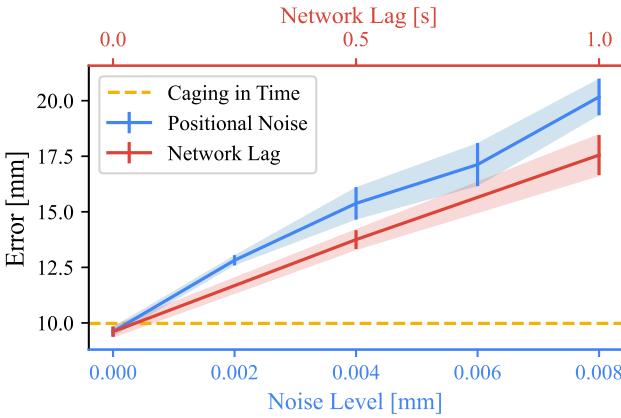
Figure 17. Robustness validation with a ∞-shaped trajectory. This experiment was conducted with the heart-shaped object (9), recorded by camera 2. *Top*: Ten loops conducted by the *Caging in Time* framework. *Bottom*: A failed attempt by a naïve pushing strategy.

Figure 18. Trajectory comparison between *Caging in Time* with 20mm cage size and 128 candidate actions and the P-controller under varying positional noise levels and network lags. *Odd rows*: Representative recordings of a single object. *Even rows*: Overall trajectories from all nine test objects.

The experimental results, as shown in Fig. 18 and Fig. 19, demonstrate that our *Caging in Time* approach achieves accuracy comparable to that of the P-controller with perfect perception. Moreover, the accuracy of the P-controller deteriorates significantly under the influence of perception noise or network lag. As shown in Fig. 18, when the Gaussian noise is large,

Figure 19. Evaluation results of the P-controller in terms of the noise level (blue) and network lag (red) compared to the *Caging in Time* framework (yellow) under 20mm cage size and 128 candidate actions.

Figure 20. Trajectory following with in-task perturbation using six different objects (*4, 7, 9, R, C, E*) for tracing "RICE". The manipulated objects are shown at the start point of each segment after the switch, with the color of the objects matched with their trajectories. The real recording is shown in Fig. 1.

the resultant trajectory tends to exhibit random motion patterns, whereas network lag induces classic sinusoidal fluctuations in the trajectory.

These findings underscore the robustness of our *Caging in Time* framework, particularly in real-world scenarios where perception and communication are often imperfect. The open-loop nature of our approach eliminates the need for continuous feedback, making it inherently resilient to perception noise and network delays that can significantly impair the performance of closed-loop methods.

7.1.4 In-task Perturbations The robustness of *Caging in Time* was further tested by maneuvering objects along specially designed trajectories shaped by the letters "RICE", as shown in Fig. 20. Notably, during this task, objects were intermittently randomly replaced by a human operator to test the system's ability to adapt to new shapes and position perturbations while maintaining trajectory precision in one single task. The results have further shown the robustness of our proposed *Caging in Time* framework: as long as the new object was positioned within the current PSS $\mathcal{Q}_t$, the robot was able to consistently cage and manipulate it along the reference trajectory, demonstrating the method's robustness and adaptability.
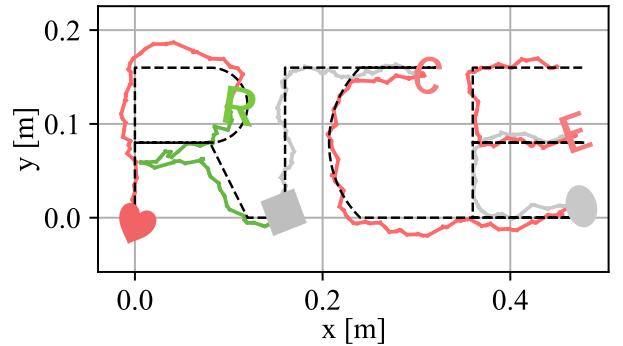
7.2 Dynamic Tasks

To evaluate the capabilities and limitations of *Caging in Time* in handling various dynamic tasks, we conducted a series of experiments using a tennis ball of diameter 6.6cm with end-effector plates in different sizes, as shown in Fig. 14.

Like quasi-static tasks, dynamic experiments also focused on three key aspects: 1) Can *Caging in Time* achieve robust ball balancing on a moving plate along different trajectories without real-time sensing feedback? 2) How well does the framework perform in more extreme dynamic scenarios, such as catching a ball in an open-loop manner? 3) How robust is *Caging in Time* when handling uncertainties higher-dimensional state spaces? In the following experiments, the plate followed a predefined translational trajectory $\mathcal{T} = \{\mathbf{x}_p(t)\}_{t=0}^T$. While the trajectory of the plate was preset, the tilt angle of the plate at each time step is computed by our framework. A Cartesian space motion control is then used to control the robot's end-effector to follow this trajectory in a pure open-loop manner. For all dynamic experiments, we also used MAE to quantify the performance. Since the plate accurately tracks the target trajectory, the MAE is calculated as the average distance between the ball and the center of the plate across all time steps and all trials.
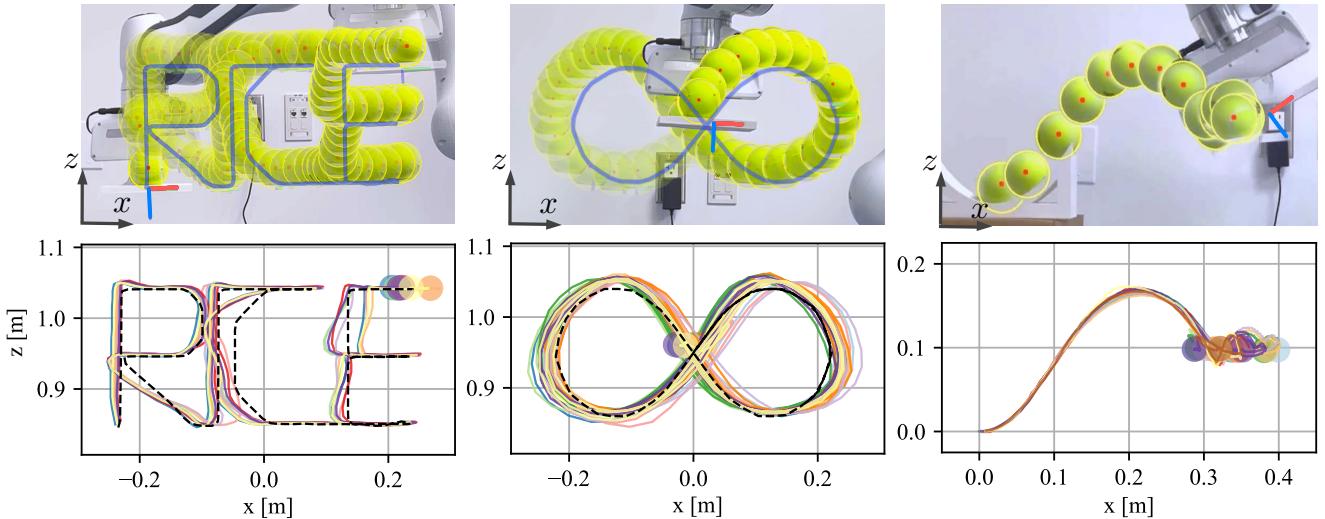
7.2.1 Ball Balancing We first tested ball balancing with dimension $n = 1$ on two distinct trajectories, as shown in Fig. 21. The plate used in this task is 16cm long and has a guide rail to constrain the ball's rolling to the x-axis only, as shown on the bottom right of Fig. 14. The "RICE" trajectory, featuring sharp turns, resulted in an average error of 20.12 ± 5.66 mm. The ∞-shaped trajectory, repeated four times to test smooth recurring curves, yielded an average error of 26.59 ± 7.54 mm. Both experiments were repeated 20 times, with the ball consistently remaining on the plate, demonstrating the robustness and stability of *Caging in Time* for long-horizon dynamic tasks.

We observed that sharp turns in the plate's trajectory led to sudden changes in $\ddot{\mathbf{x}}_p$, increasing the control difficulty and causing larger errors in the ball's trajectory. Additionally, when tracking the ∞-shaped trajectory, $\ddot{\mathbf{x}}_p$ was non-zero for most of times, resulting in a persistent non-zero tracking error. This led to a higher average error for the ∞-shaped trajectory compared to the "RICE" trajectory, even though the maximum positional error of the ball's trajectory was relatively smaller.
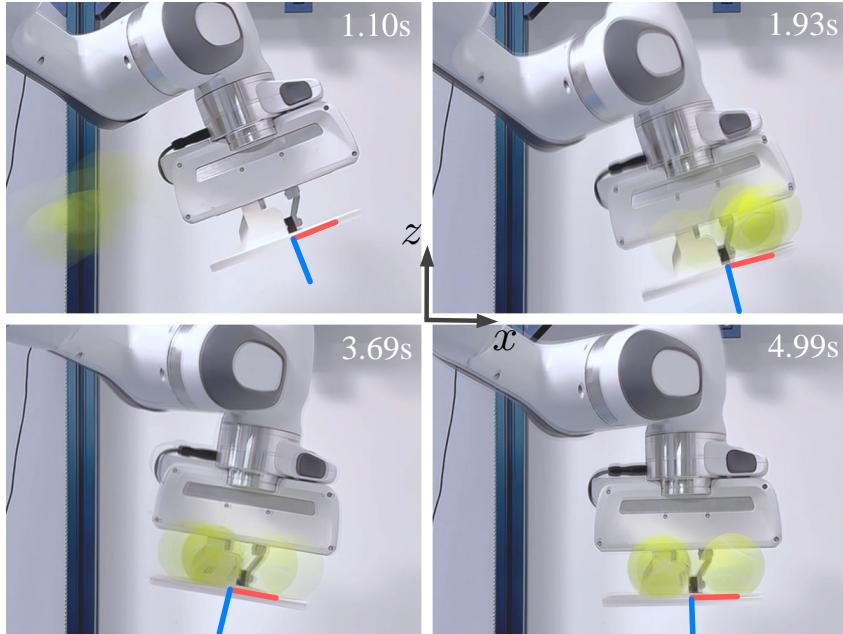
7.2.2 Ball Catching To further challenge the capability of *Caging in Time* in handling complex dynamic tasks, we implemented an open-loop ball catching experiment using the same plate with a 1D track constraint where dimension $n = 1$. In actual implementation, to ensure appropriate timing for initiating the action sequence, we utilized OpenCV to detect when the ball entered the camera frame. The open-loop action was triggered upon detection, with a manually tuned fixed time offset. Within the *Caging in Time* framework, we considered the task to start when the ball made contact with the plate,
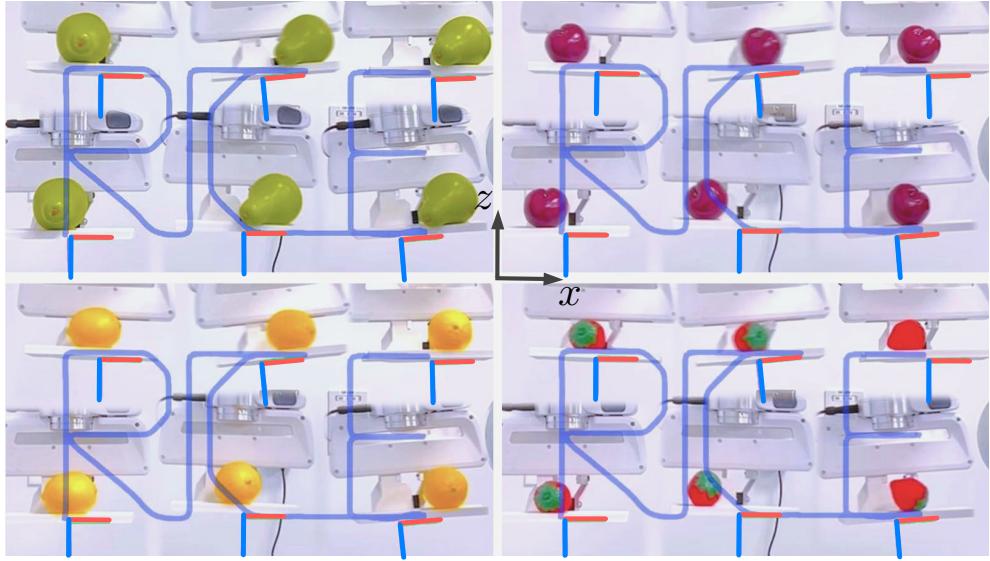
Figure 21. Experimental validation of *Caging in Time* for dynamic manipulation tasks using a tennis ball and a support surface where dimension $n = 1$ (See Fig. 14). *Top row*: Representative frames from video recordings, with overlaid transparent balls showing the ball's position at different time points. *Bottom row*: Trajectory plots of 10 repeated trials for each task. *Left*: Ball balancing while tracing "RICE". *Middle*: Ball balancing along an ∞-shaped trajectory. *Right*: Ball catching with the balls rolling down from the same height on a slope to achieve consistent initial tossing velocity.
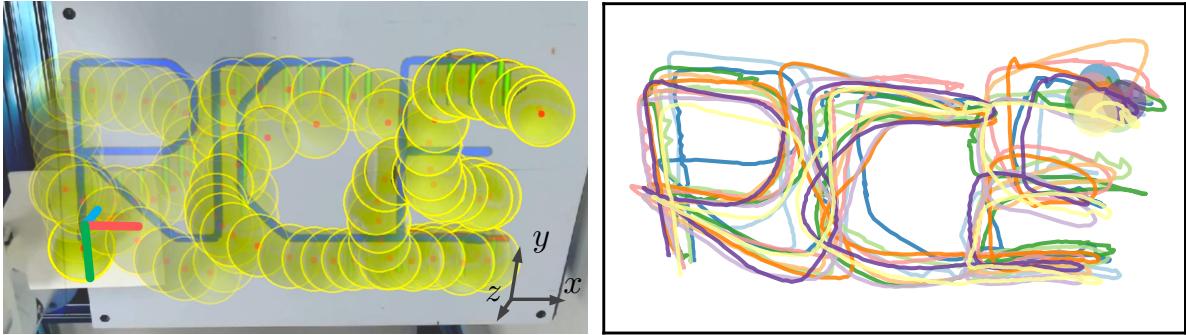

Figure 22. Demonstration of ball catching with human tossing using representative frames of 9 overlayed trials with time labels. Results along with Fig. 1 show that *Caging in Time* is able to catch the ball tossed inaccurately by a human using the same open-loop action sequence in Fig. 21.

ignoring bouncing effects. Additionally, we incorporated a hard-coded translational retreat motion to minimize ball rebounds on the plate.

We conducted two sets of experiments. In the first experiment, balls were released from a fixed height on an inclined slope, ensuring consistent initial position and approximately same velocity of the ball upon leaving the slope. Out of 20 trials, the system successfully caught the ball in all 20 attempts, as shown in Fig. 21 (right). This demonstrates the reliability of *Caging in Time* under well-defined initial conditions. In the second experiment, we introduced greater uncertainty by having a human manually toss the ball onto the plate, as illustrated in Fig. 22 and Fig. 1. Due to the inherent difficulty in controlling the landing position of hand-tossed balls, we only recorded trials where the ball successfully landed on the plate. Out of 20 such successful tosses, the system was able to catch and stabilize the ball in 13 trials. This partial success rate demonstrates the framework's robustness to velocity uncertainties within a certain range. The partial success rate in catching human-tossed balls demonstrates the limitations of the *Caging in Time* framework's pre-configured error tolerance. This tolerance is constrained to ensure feasible actions within the robot's hardware torque limits, and some human tosses evidently introduced velocity variations beyond this tolerance range.

Figure 23. Demonstration of object-agnostic robustness in *Caging in Time*. Balancing of diversely shaped YCB fruits from Fig. 14 where dimension $n = 1$ using the same open-loop action sequence for the tennis ball. Each figure shows multiple instances of the same fruit, representing its position at different time points during the balancing task.

Figure 24. Ball balancing using *Caging in Time* with the dimension $n = 2$ while tracing "RICE". *Left*: Representative frames from video recordings. The overlaid transparent balls show the ball's position at different time points during the task. *Right*: Qualitative trajectory plots of 10 repeated trials for each task.

7.2.3 Caging with Uncertainty To test the framework's robustness to shape uncertainty, we balanced various fruits in YCB dataset shown in Fig. 14 along the "RICE" trajectory with dimension $n = 1$, as in Fig. 23. Though the shape uncertainty brought higher uncertainties to the system dynamics, the framework could still make sure the object motion stays in the caged PSS. In 10 trials, all objects remained on the plate, demonstrating *Caging in Time*'s adaptability to shape variations.

To note that, The success of these tasks can be partially attributed to the nature of irregular shapes, which are less likely to roll freely due to their inherent energy traps created by their non-uniform geometry. This characteristic actually aids in maintaining stability, as the objects tend to settle into local energy minima, complementing the caging strategy of our framework.

7.2.4 Caging in Higher Dimensions Lastly, we extended our experiments to higher-dimensional spaces using a 16cm×16cm plate, where the absence of the guide rail allows the ball to move freely in both X and Y directions, significantly increasing the complexity of the balancing task. As shown in Fig. 24, we performed ball balancing with the dimension $n = 2$, where the ball's state is represented in a 4D state space $(x_t, y_t, \dot{x}_t, \dot{y}_t)$, two dimensions higher than all previous tasks.

In 10 trials, the ball consistently remained on the plate, showcasing *Caging in Time*'s applicability to higher-dimensional scenarios. The trajectories shown in Fig. 24 are for illustrative purposes due to non-orthogonal camera placement and may not represent exact quantitative performance, where we can visibly tell the trajectories exhibit larger deviations compared to the previous balancing experiments. This increased error is attributed to the unrestricted rolling direction in this setup, which introduces greater uncertainties and control challenges.

8 Conclusion

In this work, we proposed and evaluated *Caging in Time*, a novel theory for robust object manipulation. Our framework demonstrated high precision in both quasi-static and dynamic tasks without requiring detailed object information or real-time feedback. Rigorous evaluations highlighted the framework's resilience and adaptability to various objects and dynamic scenarios. The *Caging in Time* approach proved effective in handling new objects, positional perturbations, and challenging dynamic tasks, showcasing its potential for reliable manipulation in uncertain environments.

While the current *Caging in Time* framework shows promising results, it is important to acknowledge its limitations. Presently, the framework requires manual definition of task-specific parameters and the PSS propagation function. This reliance on human expertise may limit its generalizability to a wider range of manipulation tasks. Additionally, the current approach may not fully capture the complexity of certain real-world scenarios where object interactions and environmental factors are highly unpredictable.

Looking forward, we aim to address these limitations and expand the horizons of *Caging in Time*. A key direction for future research is the integration of learning techniques and Large Language Models (LLMs) to autonomously discover and learn *Caging in Time* tasks from everyday scenarios,which could enable the framework to automatically derive appropriate propagation functions and strategies. We also plan to explore its application to multi-object scenarios and more complex cases, potentially leveraging machine learning to handle increased task complexity and environmental variability.

As the *Caging in Time* theory develops with more generalizable algorithms to handle increasingly complex task dynamics, we anticipate enabling many interesting and robust real-world applications. These may include in-hand manipulation with compliant robot hands, robot picking from clutter by pushing against environmental constraints, and dynamic transportation of large objects in robot hands without secured grasps. The framework's demonstrated ability to handle both quasi-static and dynamic tasks opens up new possibilities for robust, open-loop control strategies in various robotic manipulation scenarios.

Acknowledgements

References

Ames AD, Coogan S, Egerstedt M, Notomista G, Sreenath K and Tabuada P (2019) Control barrier functions: Theory and applications. In: *2019 18th European Control Conference (ECC)*.

Anand A, Seel K, Gjærum V, Håkansson A, Robinson H and Saad A (2021) Safe learning for control using control lyapunov functions and control barrier functions: A review. *Procedia Computer Science* 192: 3987–3997.

Andrychowicz OM, Baker B, Chociej M, Jozefowicz R, McGrew B, Pachocki J, Petron A, Plappert M, Powell G, Ray A et al. (2020) Learning dexterous in-hand manipulation. *The International Journal of Robotics Research* 39(1): 3–20.

Billard A and Kragic D (2019) Trends and challenges in robot manipulation. *Science* 364(6446).

Bircher WG, Morgan AS and Dollar AM (2021) Complex manipulation with a simple robotic hand through contact breaking and caging. *Science Robotics* 6(54): eabd2666.

Blanchini F (1995) Nonquadratic lyapunov functions for robust control. *Automatica* 31(3): 451–461.

Bohg J, Hausman K, Sankaran B, Brock O, Kragic D, Schaal S and Sukhatme GS (2017) Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics* 33(6): 1273–1291.

Bütepage J, Cruciani S, Kokic M, Welle M and Kragic D (2019) From visual understanding to complex object manipulation. *Annual Review of Control, Robotics, and Autonomous Systems* 2: 161–179.

Dong Y, Cheng X and Pokorny FT (2024) Characterizing manipulation robustness through energy margin and caging analysis. *IEEE Robotics and Automation Letters* 9(9): 7525–7532.

Dong Y and Pokorny FT (2024) Quasi-static soft fixture analysis of rigid and deformable objects. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 6513–6520.

Hang K, Bircher WG, Morgan AS and Dollar AM (2021) Manipulation for self-identification, and self-identification for better manipulation. *Science Robotics* 6(54): eabe1321.

Jost J (2008) *Riemannian geometry and geometric analysis*, volume 42005. Springer.

Kaelbling LP (2020) The foundation of efficient robot learning. *Science* 369(6506): 915–916.

Kroemer O, Niekum S and Konidaris G (2021) A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of machine learning research* 22(30): 1–82.

Lee MA, Zhu Y, Srinivasan K, Shah P, Savarese S, Fei-Fei L, Garg A and Bohg J (2019) Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In: *IEEE International Conference on Robotics and Automation (ICRA)*. pp. 8943–8950.

Makapunyo T, Phoka T, Pipattanasomporn P, Niparnan N and Sudsang A (2012) Measurement framework of partial cage quality. In: *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. pp. 1812–1816.

Makita S and Nagata K (2015) Evaluation of finger configuration for partial caging. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. pp. 4897–4903.

Makita S and Wan W (2018) A survey of robotic caging and its applications. *Advanced Robotics* 31(19): 1071–1085.

Olson E (2011) Apriltag: A robust and flexible visual fiducial system. In: *IEEE International Conference on Robotics and Automation (ICRA)*. pp. 3400–3407.

Pereira GAS, Campos MFM and Kumar V (2004) Decentralized algorithms for multi-robot manipulation via caging. *The International Journal of Robotics Research* 23(7-8): 783–795.

Peshkin M and Sanderson A (1988) The motion of a pushed, sliding workpiece. *IEEE Journal on Robotics and Automation* 4(6): 569–598.

Rodriguez A (2021) The unstable queen: Uncertainty, mechanics, and tactile feedback. *Science Robotics* 6(54): 4667.

Rodriguez A, Mason MT and Ferry S (2012) From caging to grasping. *The International Journal of Robotics Research* 31(7): 886–900.

Song H, Varava A, Kravchenko O, Kragic D, Wang MY, Pokorny FT and Hang K (2021) Herding by caging: a formation-based motion planning framework for guiding mobile agents. *Autonomous Robots* 45(5): 613–631.

Stork JA, Pokorny FT and Kragic D (2013) A topology-based object representation for clasping, latching and hooking. In: *IEEE International Conference on Humanoid Robots (HUMANOIDS)*. pp. 138–145.

Sudsang A and Ponce J (2000) A new approach to motion planning for disc-shaped robots manipulating a polygonal object in the plane. In: *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2. pp. 1068–1075.

Suomalainen M, Karayiannidis Y and Kyrki V (2022) A survey of robot manipulation in contact. *Robotics and Autonomous Systems* 156: 104224.

Varava A, Carvalho JF, Kragic D and Pokorny FT (2021) Free space of rigid objects: caging, path non-existence, and narrow passage detection. *The International Journal of Robotics Research* 40(10): 1049–1067.

Varava A, Kragic D and Pokorny FT (2016) Caging grasps of rigid and partially deformable 3-d objects with double fork and neck features. *IEEE Transactions on Robotics* 32(6): 1479–1497.

Welle MC, Varava A, Mahler J, Goldberg K, Kragic D and Pokorny FT (2021) Partial caging: a clearance-based definition, datasets, and deep learning. *Autonomous Robots* 45(5): 647–664.

Yuan W, Dong S and Adelson E (2017) Gelsight: High-resolution robot tactile sensors for estimating geometry and force. *Sensors* 17(12): 2762.

ZhiDong W and Kumar V (2002) Object closure and manipulation by multiple cooperating mobile robots. In: *IEEE International Conference on Robotics and Automation (ICRA)*. pp. 394–399.
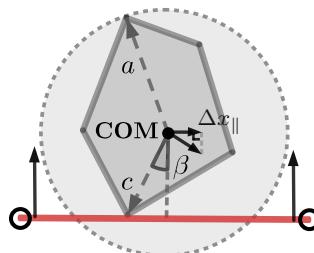
Appendix

8.1 PSS Propagation for Quasi-static Pushing

As shown in Fig. 8, assuming the line pusher is sufficiently long, we denote $d_{con} \leq d_{push}$ as the moving distance of the line pusher after contacting the bounding circle of the object. With the assumption that the motion of the object is quasi-static, we can apply the principles of energy conservation and force equilibrium to bound the displacement of the object. The force equilibrium condition yields that the frictional force is equal to the pushing force, i.e., $F_{\text{friction}} = F_{\text{push}}$. The work done by the push W_{push} is partially absorbed by the translational friction energy E_{loss}, therefore, we have $W_{\text{push}} = F_{\text{push}} \cdot d_{con} \geq F_{\text{friction}} \cdot \|\Delta q_t\| = E_{\text{loss}}$. This gives a bound for the magnitude of the object's displacement $\|\Delta q_t\| \leq d_{con}$.

Additionally, through further analysis using the Peshkin Distance (Peshkin and Sanderson 1988), we can further bound the side displacement of the object $\|\Delta q_{t\|}\|$, which is the displacement of the object parallel to the line pusher.

8.1.1 The Peshkin Distance

The Peshkin distance defines the minimal translational displacement of a fence necessary to achieve alignment of an object's edge with the fence. This distance is a function of the object's rotation center, corresponding to the sticking condition that results in the minimum rotation speed. For a reorientation process, as shown in Fig. 25, given the sticking-slowest rotation scenario, the Peshkin distance m is determined by:

$$m = \frac{a^2 + c^2}{2c}\left(\ln\left|\frac{1-\cos\beta_1}{1+\cos\beta_1}\right| - \ln\left|\frac{1-\cos\beta_0}{1+\cos\beta_0}\right|\right), \qquad (27)$$

Figure 25. The illustration of notations used for Peshkin Distance calculation. The Peshkin Distance bounds the side displacement of the object by $m/2$, where m denotes the pushing distance (same as d_{con} in Sec. 5.2).

where a represents the radius of the object's circumscribing circle centered at the mass center, c is the distance from the mass center to the contact point, and β is the angle subtended by the motion direction and the line connecting the contact point with the mass center. The angles β_0 and β_1 correspond to the initial and final orientation angles for this rotation.

To reverse engineer this process and compute the maximum change in β (denoted as $\Delta\beta$) for a fixed push distance m (i.e., d_{con} in Sec. 5.2), the following derivation is used:

$$e^{\frac{cm}{a^2+c^2}} = \frac{\tan\frac{\beta_1}{2}}{\tan\frac{\beta_0}{2}} = \frac{\tan\frac{\beta_0+\Delta\beta}{2}}{\tan\frac{\beta_0}{2}} \approx 1 + \csc\beta_0\Delta\beta \tag{28}$$

Assuming that the push distance m is much smaller than both a and c, we approximate:

$$e^{\frac{cm}{a^2+c^2}} \approx 1 + \frac{cm}{a^2+c^2} \tag{29}$$

Consequently, the maximum change in β can be approximated by:

$$\Delta\beta = \frac{c\sin\beta_0}{a^2+c^2}m \tag{30}$$

These derivations provide a mathematical basis for predicting the rotational effect of a planar pushing interaction on an object, under the assumption of no slip at the contact point.

We can further get the bound of the side-displacement $\Delta x_\parallel$ (corresponding to $\|\Delta q_{t\parallel}\|$):

$$\begin{aligned}
\Delta x_\parallel &= c\Delta\beta_0\cos\beta_0 \\
&= \frac{mc^2}{a^2+c^2}\sin\beta_0\cos\beta_0 \\
&\leq m\sin\beta_0\cos\beta_0 \leq \frac{m}{2}
\end{aligned} \tag{31}$$

Therefore, the side displacement $\|\Delta q_{t\parallel}\|$ is bounded by $\|\Delta q_{t\parallel}\| \leq d_{con}/2$. This bound plus the other bound $\|\Delta q_t\| \leq d_{con}$ constrain all the possible displacements of the object $\mathcal{V}_{q_t}$ to be inside a semi-ellipse shown by the dark green region on the right side of Fig. 8, for which the mathematical expression is given in Eq. (32).

In summary, combining both the cases where the bounding circle of the object collides with the pusher or not, the set of all possible motions for a specific configuration q_t is given by the following equations, i.e., an explicit expression of the abstract function U defined in Eq. (6).

$$\begin{aligned}
&\mathcal{V}_{q_t} = U(q_t, u_t) = \\
&\text{if } \mathbf{d}(q_t, \mathcal{L}) > r + d_{push} \text{ or } u_t \text{ is } None: \\
&\quad \{(0,0)\} \\
&\text{otherwise:} \\
&\quad \left\{ v_t = R(u_t)\begin{pmatrix}\Delta x\\\Delta y\end{pmatrix} \,\middle|\, \frac{(\Delta x)^2}{d_{con}^2} + \frac{(\Delta y)^2}{\left(\frac{d_{con}}{2}\right)^2} \leq 1, \Delta x \leq 0 \right\}
\end{aligned} \tag{32}$$

In the equation, $R(u_t) \in SO(2)$ is a rotation matrix of u_t.

8.2 Dynamic Control Implementation Details

Here we provide additional details on the implementation of the dynamic control strategy.

8.2.1 System Dynamics
The system dynamics of the ball on the tilting plate can be described by the following equation:

$$\begin{aligned}
\ddot{\mathbf{x}}_t &= f(\mathbf{q}_t, u_t, \eta_m, \boldsymbol{\eta}_p, \eta_\mu) \\
&= \frac{m(1+\eta_m)}{m + \frac{I_b}{r_b^2}}\left((I_{n+1} - \mathbf{n}(\boldsymbol{\theta})\mathbf{n}(\boldsymbol{\theta})^T) \times (\mathbf{g}_\theta + \mathbf{a}_p + \boldsymbol{\eta}_p)\right) - (\mu_r + \eta_\mu)\dot{\mathbf{x}}_t
\end{aligned} \tag{33}$$

where I_{n+1} is the identity matrix, μ_r is the estimated rolling friction coefficient, and $\mathbf{n}(\boldsymbol{\theta}) \in \mathbb{R}^{n+1}$ is the normal vector of the plate as a function of $\boldsymbol{\theta}$. The parameters m, I_b, and r_b represent the mass, moment of inertia, and radius of the ball respectively. $\mathbf{g}_\theta$ is the gravitational acceleration vector in the plate frame, and $\mathbf{a}_p$ is the acceleration of the plate.

The terms $\eta_m \in \mathbb{R}$, $\boldsymbol{\eta}_p \in \mathbb{R}^{n+1}$, and $\eta_\mu \in \mathbb{R}$ represent random distributions accounting for uncertainties in mass, plate acceleration, and friction coefficient respectively. These uncertainties follow Gaussian distributions:

$$\begin{aligned}
\eta_i &\sim \mathcal{N}(0, \sigma_i^2), \quad i \in \{m, \mu\} \\
\boldsymbol{\eta}_p &\sim \mathcal{N}(\mathbf{0}, \Sigma_p)
\end{aligned} \tag{34}$$

where $\Sigma_p \in \mathbb{R}^{(n+1)\times(n+1)}$ is the covariance matrix.

To derive the probability distribution of the ball's acceleration, $p(\ddot{\mathbf{x}}_t)$, we consider the system dynamics equation as a function of the random variables η_m, $\boldsymbol{\eta}_p$, and η_μ. Given that these uncertainties follow Gaussian distributions, $\ddot{\mathbf{x}}_t$ can be approximated as a Gaussian distribution as well. We can express this distribution as:

$$\ddot{\mathbf{x}}_t \sim \mathcal{N}(\boldsymbol{\mu}_{\ddot{\mathbf{x}}}, \boldsymbol{\Sigma}_{\ddot{\mathbf{x}}}) \tag{35}$$

Algorithm 6: DynamicControl($\cdot$)

Input: Initial state $\mathcal{Q}_0$, desired trajectory $\mathcal{T} = \{\mathbf{x}_p(t)\}_{t=0}^T$, time step Δt
Output: Success or failure of the task

1 **for** $t = 0, \ldots, T - 1$ **do**
2     $S(\mathcal{Q}_t) \leftarrow \text{EstimateEntropy}(\mathcal{Q}_t)$ $\triangleright$ Eq. (39)
3   Compute $E_{max}(\mathbf{g}_{\theta_t}, \mathbf{a}_p)$ and $E(\mathbf{q}_t, \mathbf{g}_{\theta_t}, \mathbf{a}_p)$ for all states in $\mathcal{Q}_t$ $\triangleright$ Eq. (11) (12)
4   Calculate $L_f h, L_g h, L_f V, L_g V$ $\triangleright$ Eq. (40)
5     $d\boldsymbol{\theta} \leftarrow \text{SolveQP}(L_f h, L_g h, L_f V, L_g V)$ $\triangleright$ Eq. (26)
6     $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + d\boldsymbol{\theta}$
7     $\mathcal{Q}_{t+1} \leftarrow f(\mathcal{Q}_t, \boldsymbol{\theta}_{t+1})$ $\triangleright$ Eq. (16)
8   **if** $\max_{\mathbf{q}_t} E(\mathbf{q}_t, \mathbf{g}_{\theta_{t+1}}, \mathbf{a}_p) > E_{max}(\mathbf{g}_{\theta_{t+1}}, \mathbf{a}_p)$ **then**
9   **return** *false*
10   **end**
11 **end**
12 **return** *true*

---

where $\boldsymbol{\mu}_{\ddot{\mathbf{x}}}$ is the mean acceleration vector and $\boldsymbol{\Sigma}_{\ddot{\mathbf{x}}}$ is the covariance matrix. The mean $\boldsymbol{\mu}_{\ddot{\mathbf{x}}}$ is obtained by setting all uncertainties to their expected values (zero) in the system dynamics equation.

For the special case where $n = 1$, we can reformulate the system dynamics into a control-affine form to facilitate the application of Control Barrier Functions (CBFs) and Control Lyapunov Functions (CLFs):

$$\ddot{x}_t = f_t + g_t u \tag{36}$$

where:

$$f_t = \frac{m}{m + \frac{I_b}{r_b^2}}(1 + \eta_m)((g + (\ddot{z}_p + \eta_z))\sin\theta + (\ddot{x}_p + \eta_x)\cos\theta) - (\mu_r + \eta_\mu)\dot{x}_t$$
$$g_t = \frac{m}{m + \frac{I_b}{r_b^2}}(1 + \eta_m)((g + (\ddot{z}_p + \eta_z))\cos\theta - (\ddot{x}_p + \eta_x)\sin\theta) \tag{37}$$

Here, $\eta = \{\eta_m, \eta_x, \eta_z, \eta_\mu\}$ represents random distributions accounting for uncertainties in the system. This representation separates the dynamics into a drift term f_t (system behavior without control input) and a control term $g_t u$.

This form simplifies the calculation of Lie derivatives for CBFs and CLFs. It allows for explicit representation of uncertainties in different system components through the η terms, and decouples the control input $u = d\theta$, facilitating analysis of how control changes affect the system dynamics.

In practice, the variances of these uncertainties are manually determined. Larger variances indicate higher tolerance for errors, but also increase the difficulty of finding feasible actions.

8.2.2 CBF and CLF The CLF is defined as:

$$V(\mathbf{x}) = \sum_{\mathbf{q}_t} \mathcal{I}_t(\mathbf{q}_t) E(\mathbf{q}_t, \mathbf{g}_{\theta_t}, \mathbf{a}_p) - k_S S(\mathbf{x}) \tag{38}$$

where $\mathcal{I}_t(i, j)$ is the probability at grid point (i, j) in the discretized probability distribution, and $S(\mathbf{x})$ is the system entropy. The CLF is chosen as a weighted sum of the system's energy and negative entropy to balance the goals of minimizing energy expenditure and maintaining a well-distributed probability state, ensuring both stability and robustness in the face of uncertainties.

The entropy $S(\mathcal{Q}_t)$ is calculated as:

$$S(\mathcal{Q}_t) = -\sum_{\mathbf{q}_t} \mathcal{I}_t(\mathbf{q}_t) \log \mathcal{I}_t(\mathbf{q}_t) \tag{39}$$

The Lie derivatives for the CBF and CLF are calculated as follows:

$$L_f h(\mathbf{x}) = \frac{\partial h}{\partial \mathbf{x}} f(\mathbf{x}), L_g h(\mathbf{x}) = \frac{\partial h}{\partial \mathbf{x}} g(\mathbf{x})$$
$$L_f V(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} f(\mathbf{x}), L_g V(\mathbf{x}) = \frac{\partial V}{\partial \mathbf{x}} g(\mathbf{x}) \tag{40}$$

where $\frac{\partial h}{\partial \mathbf{x}}$ and $\frac{\partial V}{\partial \mathbf{x}}$ are the gradients of h and V with respect to the state $\mathbf{x}$, respectively.

In practice, these derivatives are approximated numerically due to the complexity of the expressions for h and V.

8.2.3 Numerical Implementation The numerical implementation of the control strategy is outlined in Alg. 6.

This algorithm implements the control strategy by iteratively solving the QP and updating the system state. It ensures that the ball remains within the energy-based cage while following the desired trajectory.