A Unified Framework for Coordinated Multi-Arm Motion Planning

Seyed Sina Mirrazavi Salehian*, Nadia Figueroa* and Aude Billard*

Abstract
Coordination is essential in the design of dynamic control strategies for multi-arm robotic systems. Given the complexity of the task and dexterity of the system, coordination constraints can emerge from different levels of planning and control. Primarily, one must consider task-space coordination, where the robots must coordinate with each other, with an object or with a target of interest. Coordination is also necessary in joint-space, as the robots should avoid self-collisions at any time. We provide such joint-space coordination by introducing a centralized inverse kinematics (IK) solver under self-collision avoidance constraints; formulated as a quadratic program (QP) and solved in real-time. The space of free motion is modeled through a sparse non-linear kernel classification method in a data-driven learning approach. Moreover, we provide multi-arm task-space coordination for both synchronous or asynchronous behaviors. We define a synchronous behavior as that in which the robot arms must coordinate with each other and with a moving object such that they reach for it in synchrony. In contrast, an asynchronous behavior allows for each robot to perform independent point-to-point reaching motions. To transition smoothly from asynchronous to synchronous behaviors and conversely, we introduce the notion of synchronization allocation. We show how this allocation can be controlled through an external variable, such as the location of the object to be manipulated. Both behaviors and their synchronization allocation are encoded in a single dynamical system. We validate our framework on a dual-arm robotic system and demonstrate that the robots can re-synchronize and adapt the motion of each arm while avoiding self-collision within milliseconds. The speed of control is exploited to intercept fast moving objects whose motion cannot be predicted accurately.

Keywords
Multi-arm motion planning, coordination, dynamical systems, self-collision avoidance

1 Introduction

The use of multi-arm robotic systems allows for highly complex manipulation of heavy or bulky objects that would otherwise be infeasible for a single-arm robot. One can envision a plethora of applications in smart-factories or homes, that would benefit from such extended workspace and capabilities. Examples include, lifting, grabbing, catching, manipulating objects with multiple arms which could be either traveling on a cart or a running conveyor belt, carried by humans or even flying towards the multi-arm robot system, as depicted in Fig. 1a. Moreover, a multi-arm system could provide not only synchronous behaviors, as the ones mentioned above, but also asynchronous behaviors, where each robot follows its own goal-oriented task (Fig. 1a). Multi-arm control strategies endowed with these capabilities can pave the way for the flexible manufacturing systems of the future.

The challenge is then to control these robots in a coordinated manner, in order to safely and efficiently achieve the desired manipulation task. Due to the technological difficulties that entail coordinating multiple arms with a dynamic object in a computationally efficient way, these applications have yet to be explored in the robotics community. Most effort in the field of multi-arm control has focused primarily on devising strategies for coordinated manipulation of static objects that are partially or fully grasped by the multi-arm system (Caccavale and Uchiyama 2016). Seldom work has focused on developing coordinated strategies that a multi-arm system can use to reach and grab moving objects synchronously; while also being capable of reaching for independent targets or objects asynchronously (Vahrenkamp et al. 2012, 2010).

In this work, we draw inspiration from the field of human coordination dynamics, in order to devise safe and coordinated motions in dual/multi-arm robotic systems. Humans have a remarkable way of controlling their bi-manual movements in everyday life. They are capable of coordinating both arms and hands in synchronous and asynchronous tasks, in uni-manual and bi-manual tasks, and they transition smoothly between all of these behaviors. Interestingly, the motion of the hands, and accordingly of the arms, is generated in a smooth and efficient manner, all the while avoiding self-collisions between their limbs. Seminal works from the field of human coordination dynamics suggest that human inter-limb coordination is governed

---

* These authors contributed equally to this work.

1 Learning Algorithms and Systems Laboratory (LASA), Swiss Federal Institute of Technology, Lausanne (EPFL) CH-1015 Lausanne, Switzerland.

Corresponding author:
Seyed Sina Mirrazavi Salehian, EPFL-STI-IMT-LASA, ME A3 424 (Bâtiment ME) Station 9 CH-1015 Lausanne, Switzerland
Email: sina.mirrazavi@epfl.ch

Prepared using sagej.cls (Version: 2015/06/09 v1.01)
Transitioning between

(a) Transitioning between **Synchronous** and **Asynchronous**
Mult-Arm Task-Space Behavior

(b) Self-Collision Avoidance in Joint-Space

**Figure 1.** Illustration of Multi-arm Task-Space Coordination and Joint-Space Collision Avoidance. (a) **Synchronous** and **Asynchronous** task-space behaviors, where the robots coordinate with each other to simultaneously reach-for a moving object or each robot has its own target and is endowed with an independent stable DS to generate desired motions, respectively. (b) Self-Collision Avoidance (SCA) in Joint-Space for both task-space behaviors.

by a strongly coupled underlying nonlinear dynamical system (DS) (Kelso et al. 1979; Kelso 1984). These studies involved transitioning between in-phase and anti-phase rhythmic movements, which led to the Haken-Kelso-Bunz (HKB) model of coordination (Haken et al. 1985). (Swinnen 2002) highlights strong limb-coupling in terms of the **spatial** constraints that govern coordination, which can be **egocentric** (based on mirror symmetry in muscle groups) or **allocentric** (following the same direction in extrinsic space). Later on, (Calvin and Jirsa 2011) introduced a generalized description of such coordination dynamics, as a combination of intrinsic dynamics (coordination types) and a strong coupling between them. Showcasing that, coordination can also arise with discrete (i.e. non-rythmic) bi-manual movements. More compelling evidence of the strong bi-manual coupling in humans can be seen when executing independent tasks. In the study of (Franz et al. 1991), the subjects were asked to perform two **independent** discrete movements with each hand, drawing a circle and a vertical straight line, respectively. The resulting shapes were **vertical** ellipses, elucidating the strong **ego-centric** constraints in bi-manual movements.

Humans also display an underlying coordination with external agents, for example when manipulating objects. Tasks such as lifting, carrying and reaching for large or heavy objects rely on bi-manual reaching behaviors which require not only **spatial** but also **temporal** constraints (Coats and Wann 2012). In a bi-manual reach, each hand has to adjust to the orientation, shape and size of the object while reaching for it. Moreover, the action of grabbing the object (i.e. closing the hands on the object) must be timed prior to rather than as a reaction to intercepting the object. Hence, bi-manual reaching requires to solve simultaneously **spatial** and **temporal** coordination constraints to move toward the object in coordination and to intercept the object (Vernon et al. 2011). Furthermore, when humans reach to different targets, their behavior might not exhibit spatial constraints, but the **temporal** constraints are still enforced. They smoothly adapt their respective speed to reach the targets simultaneously. Meaning that, there is an apparent timing synchronization even in asynchronous behaviors.

In our previous works (Salehian et al. 2016a, 2017), we offered a dynamical system (DS) based controller for **coordinated** multi-arm motion planning. The approach consists of a virtual object-DS control law that generates autonomous and synchronized motions for a multi-arm robot system. We use the notion of a virtual object to both **coordinate** the motion of the multiple robots with each other and with a moving object, such that the robots reach the dynamic object in **synchrony** (Figure 1a). Such a dynamical system emulates the strongly coupled human coordination strategies exhibited in the previously mentioned studies, providing predictable motions for humans.

In this paper, we improve upon our previous work, by tackling two main challenges that were not addressed in (Salehian et al. 2016a). First, we extend the virtual object-DS, such that it can generate two types of behaviors: (i) multi-arm **asynchronous** task-space behaviors, where each robot has its own target or desired motion (Figure 1a) and (ii) multi-arm **synchronous** task-space behaviors, where the robots’ task is to **coordinate** with each other to simultaneously reach-for a moving object (Figure 1a). To provide a smooth transitioning between these two behaviors, we introduce the notion of synchronization allocation. Given the motion of the object and the joint workspace of the multi-arm system, each arm is being continuously allocated to a desired behavior. While being allocated to the synchronous behavior, control of the robots is taken over by the virtual
object-DS. While allocated to the asynchronous behavior, the robots are controlled independently, each with their own goal-directed stable DS. The proposed multi-arm-DS is expressed as a Linear Parameter Varying (LPV) system subject to stability constraints, that ensure convergence to the object or targets. Second, to ensure self-collision avoidance at the joint-level, we propose a centralized IK solver, formulated as a constrained convex optimization problem subject to data-driven self-collision avoidance (SCA) constraints. These SCA constraints are introduced as linear inequality constraints in the optimization problem in the form of a continuous “SCA Boundary function” and its gradient. We then propose to efficiently encode this SCA boundary function through a sparse kernel Support Vector Machine (SVM) (Joachims and Yu 2009), learned a priori from a simulated dataset of feasible configurations of the multi-arm system. The contributions of this paper, compared to our previous work (Salehian et al. 2016a), are thus threefold:

1. Unification of synchronous and asynchronous multi-arm behaviors in a single dynamical system.
2. A Self-Collision Avoidance (SCA) IK solver formulated as a convex QP problem that can be solved in real-time.
3. A data-driven learning approach to model the SCA constraints in a computationally efficient way.

This paper is organized as follows. Section 2 highlights related work on multi-arm control and SCA strategies. In Section 3, we present the multi-arm-DS, including a formalization and convergence proof of the LPV DS. Section 4 introduces a centralized inverse kinematic solver which can handle SCA constraints. In Section 5, we present the learning approach to approximate the safe manipulation regions fed to the SCA-IK solver. The proposed method is then validated with a dual-arm platform for several coordination and reaching scenarios in Section 6. Discussions and future work are presented in Section 7.

2 Related Work

Coordinating multiple robotic arms for object manipulation has been extensively studied in the robotics literature, one can find comprehensive surveys on dual/multi-arm motion planning in Wimböck et al. (2012); Wimböck and Ott (2012); Smith et al. (2012); Caccavale and Uchiyama (2016). However, the problem of planning the reach to grasp motion for a moving object with a multi-arm robotic system, while keeping coordination constraints, is a new field of research. The sole work that tackles a similar problem is that of Vahrenkamp et al. (2012, 2010), who proposed a motion synchronization controller to coordinate the end-effectors of dual-arm system, where the slave arm is synchronized with the master arm through a coupling guided by position and velocity feedback errors. Although computationally efficient, this strategy assumes a fixed master-slave relationship, which, when dealing with moving objects, may adversely affect performance if the arms need to switch responsibility to perform the task on-line. In (Bai and Wen 2010), by using a velocity feedback and force feed-forward strategy, a decentralized controller is proposed for transporting a flexible payload at a constants speed with multiple arms. In this approach master/slave roles are not assigned. However, as it assumes that each robot is a point-mass system it could not readily be applied to manipulating a moving object with unpredictable dynamics, as considered in our approach.

2.1 De-Centralized Multi-Arm Control Strategies

In de-centralized control architectures, the robots are controlled separately by their own local controllers (Liu and Arimoto 1998). In early approaches, the coordination between a dual-arm system is achieved by categorizing them into two categories; namely a master and a slave. The motion of the master robot is assumed known whereas the slave robot must follow the master’s motion while satisfying the closed-chain geometrical constraints (Luh and Zheng 1987). Similarly, (Gams et al. 2015) proposed a control architecture to perform a task of lifting an unknown object with a dual-arm system, where the slave arm is synchronized with the master arm through a coupling guided by position and velocity feedback errors. Although computationally efficient, this strategy assumes a fixed master-slave relationship, which, when dealing with moving objects, may adversely affect performance if the arms need to switch responsibility to perform the task on-line. In (Bai and Wen 2010), by using a velocity feedback and force feed-forward strategy, a decentralized controller is proposed for transporting a flexible payload at a constants speed with multiple arms. In this approach master/slave roles are not assigned. However, as it assumes that each robot is a point-mass system it could not readily be applied to manipulating a moving object with unpredictable dynamics, as considered in our approach.

2.2 Centralized Multi-Arm Control Strategies

Some of the shortcomings of the de-centralized control architectures can be addressed by centralized control strategies (Aghili 2013; Suda et al. 2003; Wang et al. 2015). These strategies consider the robots and the manipulated object as a closed kinematic chain. In this line, an impedance control architecture for dual-arm manipulation is proposed in (Himpel and Ott 2012), where the two end-effectors and a virtual frame, which is a function of the end-effectors’ poses, are coupled via spatial springs. (Zhu 2005) proposed a motion synchronization controller to coordinate the end-effectors of dual-arm system. By concatenating the kinematic of two arms and the object, they introduced an augmented kinematic chain. The corresponding Jacobian is calculated to control the augmented kinematic chain by solving the inverse kinematics problem at the velocity level. By exploiting advantages of centralized and de-centralized impedance control strategies, (Caccavale et al. 2008) proposed a control architecture to achieve a desired impedance at both the object and the end-effector levels. Similarly, (Chiachi and Chiaverini 1998) proposed a two
level control architecture. Initially, the desired task variables are transformed into the corresponding joint-space motions by solving a centralized inverse kinematic problem. Then, the desired joint motions are fed to a decentralized joint-space controller.

All previously mentioned works assume that the object is firmly attached to the robots and modeled via a virtual object frame or by closing the kinematic chain. In this work, we leverage the idea of the virtual object to address the problem of coordination. The term virtual object is mostly used in robotics literature to represent the internal forces of a grasping task (Williams and Khatib 1993; Winbod et al. 2008). In this paper, however, this term is used to achieve coordination at two levels. In the first level, the motion of each robot is coordinated with all robotic arms. In the second level, the resultant motion of the arms are coordinated with that of the object to satisfy the coordination constraints (i.e. for synchronous behavior).

2.3 Multi-Arm Self-Collision Avoidance

Self-collision avoidance is one of the main challenges in multi-arm manipulation. It is particularly relevant in the humanoid robot community and hence, has been extensively studied. Throughout the years, the approaches for solving collision avoidance for manipulation or locomotion in humanoids can be categorized into two types: (i) planning methods which generate feasible collision-free trajectories of known/quasi-static environments (Gharbi et al. 2009; Vahrenkamp et al. 2012; Escande et al. 2007; Vahrenkamp et al. 2010; Escande et al. 2014; Chrétien et al. 2016) and (ii) reactive approaches which solve collision-avoidance through the IK problem online (Ge and Cui 2000; Santis et al. 2007; Sugiura et al. 2007; Fang et al. 2015). For a comprehensive review on collision avoidance strategies for bi-manual systems refer to Petrić et al. (2015).

The main disadvantage of (i) planning approaches is their computational cost. Although much progress has been achieved in this regard, most planning approaches are still restricted to static scenarios. For example, in a dynamic scenario, where the robots must adapt to fast external perturbations, solving for collision avoidance must be in a matter of $1-2\text{ms}$. Typical computation times for efficient solvers in humanoid scenarios, lie between $\approx 700\text{ms} \rightarrow \approx 1\text{s}$ (Kanehiro et al. 2012; Orthey and Stasse 2013). Recently, (Chrétien et al. 2016) showed that the computation time of one iteration (for their collision-avoidance humanoid trajectory solver) is $2715\text{ms}$ on a single thread CPU. They prove, however, that with the use of parallel computing, their computation time can be improved significantly; i.e. to $54\text{ms}$ on a GPU. Though significant, this improvement is far from the requirements of dynamic scenarios. The sole approach which has proven to achieve dynamic adaptation at $<1\text{ms}$ is that of (Murray et al. 2016), where an FPGA robot-specific chip is designed for dynamic motion planning. Although promising, this approach is limited by the need of computing a probabilistic roadmap a priori and its dependence on a specific robot configuration.

Reactive approaches (ii), on the other hand, are not hindered by computational inefficiency. In the works of (Santis et al. 2007; Sugiura et al. 2007) repulsion forces are computed from colliding segments to generate self-collision avoidance motions. (Fang et al. 2015) proposed a hierarchical-based algorithm to find in-danger points and solve the IK problem such that the distance between these points is maximized. These approaches, although fast, suffer from the same shortcoming as generic potential-field obstacle avoidance algorithms; i.e. the stability of the generated motion is not guaranteed, as the robot might get stuck at local minima. Moreover, potential-field approaches suffer from providing no passages between closely spaced obstacles and the possibility of oscillations in presence of obstacles (Ge and Cui 2000).

All of the previously mentioned works, be it (i) planning-based or (ii) reactive, have a common methodology: they rely on computing minimum distances between links/joints/segments/objects (represented as sphere/swept-spheres/polygons) to detect/avoid collisions. The use of minimum distances for collision avoidance inherently introduces non-linear and non-convex constraints to an otherwise convex optimization problem (Ratliff et al. 2015). This, in fact, is the main reason (i) planning algorithms rely on computationally inefficient global optimization or trajectory optimization methods and that (ii) reactive methods tend to get stuck in local minima. To this end, approaches based on signed distance fields have been successful in encoding proximity of obstacles as continuous costs in local trajectory optimization frameworks; by either providing explicit cost gradients (Ratliff et al. 2009; Zucker et al. 2013) or through derivative-free stochastic optimization methods (Kalakrishnan et al. 2011). Such approaches, however, fail to recover when solving for optimization problems that become ill-defined due to particular shapes of obstacles or many local minima. To alleviate this, (Ratliff et al. 2015) provide a general motion optimization framework which exploits the Riemannian geometry of the workspace to represent costs for obstacle avoidance, kinematic limits, etc, by warping the workspace via Riemannian metrics and their gradients. These approaches, although promising, are still limited by their computational efficiency, as they all present optimization times of $\approx 0.5s$.

In this work, we focus on solving the self-collision avoidance problem efficiently; i.e. in $<2\text{ms}$. We work-around the limitations of the previously mentioned approaches, by learning a continuous and continuously differentiable function $\Gamma(\cdot)$ from a dataset of “collided” and “non-collided” multi-arm configurations. $\Gamma(\cdot)$ represents the region of feasible and infeasible robot configurations, implicitly encoding a distance in feature space, of the current robot configuration to a “collided” configuration. By formulating $\Gamma(\cdot)$ as the prediction rule of a kernel Support Vector Machine (SVM) (Scholkopf and Smola 2001) we can compute a continuous $\text{VT}(\cdot)$ on-line. We hence propose a centralized IK solver as a convex QP optimization problem subject to linear inequality constraints imposed by $\Gamma(\cdot)$ and $\text{VT}(\cdot)$, avoiding (i) the computation of $\text{min}[[\cdot],[\cdot]]$ pair-wise distances between joints/links/segments and (ii) the need for trajectory optimization. Through the use of a sparse Support Vector Machine approximation (Joachims and Yu 2009) we learn an efficient representation of $\Gamma(\cdot)$ that enables us to solve the QP optimization problem in less than $2\text{ms}$, implemented on a single threaded CPU.
3 Coordinated Multi-Arm Motion Planning

In order to achieve the envisioned scenario; i.e. smoothly transition between asynchronous and synchronous behaviors depending on the object’s motion, two main challenges should be addressed:

1. Prediction of the object’s trajectory and computation of its feasible intercept points.
2. Planning stable multi-arm motions towards their corresponding object intercept points (when allocated to synchronous behavior), or individual targets (when allocated to asynchronous behavior).

The proposed solutions to these challenges are described in sub-sections 3.1 and 3.2, respectively. Once the multiple end-effector motions are generated, these are mapped to desired joint-space configurations by a centralized IK solver which uses a learned model of collision-free regions and the robots’ kinematic constraints, this is described in Sections 4 and 5. For simplicity and practicality, we summarize the most relevant notations used throughout the paper in Table 4 and 5. For simplicity and practicality, we summarize the.

![Diagram](image)

**Figure 2.** An illustration of the variables for \( N_R = 2 \). The reachable areas are feasible areas for grasping the object. For except for \( \xi^{O}_{2} \) and \( \xi^{O}_{1} \), the variables are expressed in the reference frame located on the desired intercept point; i.e. \( \xi^{O}(T^{*}) = [0 \cdots 0]^T \).

Euclidean space position, to the robot end-effectors. Refer to (Salehian et al. 2016a) for details of this procedure.

### 3.1 Object Trajectory and Intercept-Point Prediction

Once the main object starts moving towards the robots, a linear model predicts its progress ahead of time and determines a point along its trajectory where the object will become reachable by all robotic arms. We do not assume a known model of the dynamics of the object. The sole knowledge about the object is the location of its reaching points. These correspond to user-defined position and orientation of the arms at the reaching point (see Fig. 2).

To find the feasible intercept point, which is the point where the object can be reached by all robots, at its predefined reaching points, we model the workspace of each robot via a probabilistic classification scheme. The reachable workspace of each robot, \( p(j|\xi, \Theta^{R}) \) \( \forall j \in \{1, \ldots, N_R\} \) for \( N_R \) robots, is modeled through a Gaussian Mixture Model (Bishop 2007). If \( p(j|\xi, \Theta^{R}) > \delta_{1} \), where \( \delta_{1} \) is a minimum likelihood threshold, then \( j^{*} \), i.e. the \( i \)-th reaching position on the object (sub-script), is classified as a feasible position for the \( j \)-th robot to reach. As the reachable workspaces of each robot are statistically independent from each other, we can calculate the joint distribution of all workspaces by computing the product of distributions, as follows:

\[
p(\{\xi_{1}^{O}, \ldots, \xi_{N_R}^{O}\}; \Theta^{W}) = \prod_{j=1}^{N_R} \prod_{i=1}^{N_R} p_{j}(\xi_{j}^{O}; \Theta_{j}^{W})
\]

where \( \Theta^{W} = \{\Theta_{1}^{W}, \ldots, \Theta_{N_R}^{W}\} \) is the set of parameters for all robot workspaces and \( \{\xi_{1}^{O}, \ldots, \xi_{N_R}^{O}\} \) are the reaching points in each robot’s reference frame. The minimum joint likelihood threshold is \( \delta = \prod_{j=1}^{N_R} \delta_{j} \) if \( \exists T^{*} : \delta < \prod_{j=1}^{N_R} p_{j}(\xi_{j}^{O}(T^{*}); \Theta_{j}^{W}) \). The object at \( T^{*} (\xi^{O}(T^{*}) = \frac{1}{N_R} \sum_{j=1}^{N_R} \xi_{j}^{O}(T^{*})) \) is classified as the feasible intercept point. If more than one point on the predicted trajectory is classified as the feasible intercept point, we select the closest one, in
from kinematically feasible demonstrations. The advantage of this technique is that it inherently results in normalized scheduling parameters; i.e. \( 0 < \theta_k \leq 1 \), \( \sum_{k=1}^{N_R} \theta_k = 1 \), \( \forall i \in \{1, \ldots, N_R\} \), refer to (Salehian et al. 2016b) for further details on this approximation approach.

**Theorem 1.** The dynamical systems given by (2) asymptotically converge to \( \tau_c, x^V_i + (1 - \tau_c)x^d_i \); i.e.

\[
\lim_{t \to \infty} \| x_i^R(t) - \tau_c(t) x_i^V(t) + (\tau_c(t) - 1)x_i^d \| = 0
\]

if there exist \( P^R, Q^R \) such that:

\[
\begin{cases}
0 < P^R_i & 0 < Q^R_i \\
P^RA_{ik} + A_{ik}^TP^R_i = -Q^R_i & \forall k \in \{1, \ldots, d_i\}
\end{cases}
\]

Moreover, by taking \( \dot{\tau}_c(x_i^V - x_i^d) + \tau_c \dot{x}_i^V - A_i(\theta(x_i^V))(x_i^d + \tau_c(x_i^V - x_i^d)) \) as the input and \( x_i^d(t) \) as the output of the dynamical system (2), (2) is passive if (5) is satisfied.

**Proof:** see Appendix B and C.

In (5), \( P^R_i \) and \( Q^R_i \) are auxiliary matrices which are used in a Lyapunov stability and convergence proof provided in Appendix B. It is important to remark that the Lyapunov function used in these stability proofs and hence, the matrices \( P^R_i \) and \( Q^R_i \), are never evaluated, since it is merely their existence that is required. In (2), \( \tau_c \) determines the level of synchronization between the \( i \)-th robot and the virtual object, see Fig. 3. Assuming that \( \tau_c = 0 \), (2) yields an asynchronous DS for reaching towards individual targets:

\[
x_i^R = A_i(\theta)(x_i^R - x_i^d) \rightarrow \{\lim_{t \to \infty} \| x_i^R - x_i^d \| = 0 \}
\]

(6)

Similarly, when \( \tau_c = 1 \), (2) results in a perfect tracking DS of the \( i \)-th reaching point on the virtual object:

\[
x_i^R = \dot{x}_i^V + A_i(\theta)(x_i^R - x_i^V) \rightarrow \{\lim_{t \to \infty} \| x_i^R - x_i^V \| = 0 \}, \lim_{t \to \infty} \| \dot{x}_i^R - \dot{x}_i^V \| = 0
\]

(7)

To smoothly transition between these behaviors, one could calculate \( \tau_c \), \( \forall i \in \{1, \ldots, N_R\} \) with a continuous logistic function. In this paper, however, we propose the following DS which varies \( \tau_c \), \( \forall i \in \{1, \ldots, N_R\} \) such that \( \tau_c \rightarrow 1 \) when the object moves towards the robots and \( \tau_c \rightarrow 0 \) when it moves away:

\[
\begin{align*}
\tau_c(t) &= \frac{\tau_c(1 - \tau_c)G(x_i^O(t), \xi_i^O(t))}{k} \\
\tau_c(0) &= e, \forall i \in \{1, \ldots, N_R\} \\
G(.) &= \frac{\xi_i^O(t)}{\| \xi_i^O(t) - \xi_i^O(0) \|}
\end{align*}
\]

Where, \( 0 < \varepsilon < 1 \) is a small positive value, \( k \in \mathbb{R}_{>0} \) is a positive constant that controls for the steepness of the increase or decrease of the parameter.\(^2\) As the initial value of \( \tau_c \) is positive \( < 1 \), (8) is a bounded dynamical system between \([0, 1]\). \( G(.) \) is a function that coordinates the robots with the virtual object, such that if the real object moves toward the workspaces, the robots perform the synchronous behavior, otherwise they fall back to the asynchronous behavior. The main advantage of the proposed criterion is its adaptability, \( \text{sgn}(\tau_c) \) changes with respect to the direction of the object’s motion; when the object approaches the robots, \( \text{sgn}(\tau_c) \rightarrow (+) \), otherwise, \( \text{sgn}(\tau_c) \rightarrow (-) \). Consequently, if the object is moving towards the robots, they are synchronized with the virtual object. Otherwise, they perform the asynchronous behavior.

As the synchronization allocation parameters vary over time, the virtual object-DS proposed in (Salehian et al. 2016a), which generates the motion of the virtual object, is no longer applicable. To appropriately consider the effects of the synchronization parameters on the motion of the virtual object, and consequently of the robots, the following DS is proposed to generate the motion of the virtual object.

\[
x_i^V(t) = \frac{1}{1 + \sum_{i=1}^{N_R} \tau_c} \left( \gamma x_i^O + \gamma x_i^A + A_i(x_i^V - x_i^O) + \sum_{i=1}^{N_R} U_i \right)
\]

Where, \( x_i^V(t) = [\xi_i^V(t) \ \dot{\xi}_i^V(t)] \) is the state of the virtual object. \( 0 < \gamma < 1 \) is the coordination parameter and is of class \( \gamma^1 \). \( U_i \) is the interaction effect of the motion of the \( i \)-th effector on the virtual object, based on (2) and (9):

\[
U_i = x_i^R - A_i(\theta)(x_i^R - x_i^V - \tau_c(x_i^V - x_i^d))
\]

(10)

By substituting, (2) and (10) into (9), we have:

\[
x_i^V(t) = \frac{1}{1 + \sum_{i=1}^{N_R} \tau_c} \left( \gamma x_i^O + \gamma x_i^A + A_i(x_i^V - x_i^O) + \sum_{i=1}^{N_R} \tau_c x_i^d \right)
\]

(11)

**Theorem 2.** The dynamical system given by (11) asymptotically converges to [\( \gamma(t) \xi_i^O(t) \ \gamma(t) \dot{\xi}_i^O(t) \ + \gamma(t) \dot{\xi}_i^O(t) \)] \( t \) i.e.

\[
\lim_{t \to \infty} \| \xi_i^V(t) - \gamma(t) \xi_i^O(0) \| = 0
\]

\[
\lim_{t \to \infty} \| \dot{x}_i^V(t) - (\gamma(t) \dot{\xi}_i^O(t) + \gamma(t) \dot{\xi}_i^O(t)) \| = 0
\]

if there exit \( P^V_i, Q^V_i \) such that:

\[
\begin{cases}
0 < P^V_i \\
0 < Q^V_i
\end{cases}
\]

\[
P^V A_i + A^T_i P^V < -Q^V
\]

(13)

Moreover, by taking \( \gamma x_i^O + \gamma x_i^A + A_i(x_i^V - x_i^O) \) as the input and \( x_i^d \) as the output of the dynamical system (11), (11) is passive if (13) is satisfied.

**Proof:** see Appendix D, E.
The C++ implementation of this approach is provided in
virtual object
guarantee collision-avoidance between end-effectors, via the
feasible reaching positions
motion in face of inaccuracies in the object’s motion
that addresses self-collision avoidance at all times.
section we present a centralized inverse kinematics solver,

\[ \mathbf{J}(\mathbf{q}) = \frac{q^T \mathbf{W} q}{2} \] (17a)

Minimize expenditure
Subject to:
\[ \mathbf{J}(\mathbf{q}) = \frac{q^T \mathbf{W} q}{2} \] (17b)
Satisfy the desired end-effector motion
\[ \dot{\theta} \leq \mathbf{q} \leq \dot{\theta}^+ \] (17c)
Satisfy the kinematic constraints
\[ -\nabla \Gamma^i(q^{ij})^T \dot{q}^{ij} \leq \log(\Gamma^i(q^{ij}) - 1) \] (17d)
Do not penetrate the collision boundary

4 Self-Collision Avoidance (SCA)

To avoid collisions between the joints of the arms, we need to
device a control algorithm to ensure that none of the robots’ body parts collide with each other.
To achieve this objective, the IK solver must consider not
only the kinematic constraints of each robot, but also self-
collision constraints. Given that the robots’ bases are fixed wrt.
each other, we can explore the joint workspace of the
robots, in order to model the regions that may lead to collision.
Since the space of joint configurations is continuous, we must
approximate the regions of collisions by building a continuous map of the feasible (safe) and
infeasible (collided) configurations. Assuming that the infeasible regions can be bounded through a continuous and
continuously differentiable function \( \Gamma(q^{ij}) : \mathbb{R}^{N_d^j} \to \mathbb{R} \),
where \( q^{ij} = [q^i,q^j]^T \in \mathbb{R}^{N_d^i+N_d^j} \) are the joint angles of the \( i \)

and \( j \)
th robots, respectively. We define \( \Gamma(\cdot) \) such that:

**Collided configurations:** \( \Gamma(q^{ij}) < 1 \)

**Boundary configurations:** \( \Gamma(q^{ij}) = 1 \) (16)

**Free configurations:** \( \Gamma(q^{ij}) > 1 \)

A data-driven approach for building \( \Gamma(q^{ij}) \) is proposed in
Section 5. Fig. 4 illustrates a \( \Gamma(\cdot) \) for a toy 2D example. (16)
provides constraints that must be taken into account when solving the inverse kinematics (IK) problem. We propose the following quadratic program to solve the IK:

\[ \arg \min_{\mathbf{q}} \frac{q^T \mathbf{W} q}{2} \] (17a)

Minimize expenditure
Subject to:
\[ \mathbf{J}(\mathbf{q}) = \frac{q^T \mathbf{W} q}{2} \] (17b)
Satisfy the desired end-effector motion
\[ \dot{\theta} \leq \mathbf{q} \leq \dot{\theta}^+ \] (17c)
Satisfy the kinematic constraints
\[ -\nabla \Gamma^i(q^{ij})^T \dot{q}^{ij} \leq \log(\Gamma^i(q^{ij}) - 1) \] (17d)
Do not penetrate the collision boundary

Where, \( \mathbf{q} = [q^1, \ldots, q^{N_R}]^T \in \mathbb{R}^{N_R} \), \( \mathbf{d}_q = \sum_{i=1}^{N_R} d_i \mathbf{q} \). \( \mathbf{W} \) is a
block diagonal matrix of positive definite matrices. \( \mathbf{J} = diag(J_1, \ldots, J_{N_R}) \) is block diagonal matrix of
the Jacobian matrices.

\[ \mathbf{J}(\mathbf{q}) = \frac{q^T \mathbf{W} q}{2} \] (17b)
Satisfy the desired end-effector motion
\[ \dot{\theta} \leq \mathbf{q} \leq \dot{\theta}^+ \] (17c)
Satisfy the kinematic constraints
\[ -\nabla \Gamma^i(q^{ij})^T \dot{q}^{ij} \leq \log(\Gamma^i(q^{ij}) - 1) \] (17d)
Do not penetrate the collision boundary

Where, \( \mathbf{q} = [q^1, \ldots, q^{N_R}]^T \in \mathbb{R}^{N_R} \), \( \mathbf{d}_q = \sum_{i=1}^{N_R} d_i \mathbf{q} \). W is a
block diagonal matrix of positive definite matrices. J =
block diagonal matrix of the Jacobian matrices.

\[ \mathbf{J} = \begin{bmatrix} J_1 & \cdots & J_{N_R} \end{bmatrix} \] (17b)
Satisfy the desired end-effector motion
\[ \dot{\theta} \leq \mathbf{q} \leq \dot{\theta}^+ \] (17c)
Satisfy the kinematic constraints
\[ -\nabla \Gamma^i(q^{ij})^T \dot{q}^{ij} \leq \log(\Gamma^i(q^{ij}) - 1) \] (17d)
Do not penetrate the collision boundary

\[ \dot{\theta}_i := \max \left( \mu(q_i - q_i), q_i^{-} \right) \] (18)
\[ \dot{\theta}_i := \min \left( \mu(q_i - q_i), q_i^{+} \right) \] (18)
With $q_i^\pm, \dot{q}_i^\pm, \ddot{q}_i$ as the conservative lower and upper bounds on the joints’ positions and velocities. The intensity coefficients, $0 < \mu_i$, determine the magnitude of decelerations. This should be defined such that the feasible region of $\dot{\theta}$ is greater than the joint velocity limits.

While the robots are far from the boundary configurations, the value of $\log(\Gamma^\theta(q^\theta) - 1)$ is negative which relaxes the inequality constraints; i.e. the robots accurately follow the desired end-effector trajectory. When they are near the boundary configurations, the value of $\log(\Gamma^\theta(q^\theta) - 1)$ is negative. Therefore, constraint (17d) forces the joint angles to move away from the boundary as they approach it. Since satisfying the collision avoidance and the kinematic constraints is of higher priority than following the desired end-effector motion, we give higher penalty to (17c) and (17e) as the solutions to problem with equality and inequality constraints, hence, $P_{\Omega}^i$ and $P_{\Omega}^j$ are the solutions to the QP.

Equation (17) is a convex quadratic programming (QP) problem with equality and inequality constraints, hence, there is no closed form solution for it. As the solutions to such linear optimization are solver-dependent, in terms of computation cost, we compare three approaches to solve (17). The first approach formulates (17) as a system of piecewise-linear equations and uses a DS-based approach to solve them (Xia and Wang 2000; Zhang et al. 2004; Zhang 2005). The second approach uses Nlopt, a standard nonlinear programming solver (Johnson 2016). The third approach uses a solver specifically designed for constrained convex problems; the solver which we use is called CVXGEN, introduced in (Mattingly and Boyd 2012), which generates C codes, tailored for the specific formulation of (17). As the second and third approaches are ready to use interfaces, in the rest of this chapter we introduce the first approach.\(^5\)

**Lemma 1.** Linear quadratic programming (17) is equivalent to the following system of piecewise-linear equations.

$$P_{\Omega}^i(u - (Mu + b)) - u = 0$$ (19)

Moreover, the following dynamical system is asymptotically stable to $u^*$, where $u^* \in \mathbb{R}^d_a$ is the solution of (19).

$$\ddot{u} = (I + MT)(P_{\Omega}^i(u - (Mu + b)) - u)$$ (20)

where

$$M = \begin{bmatrix} W & -\dot{J}(q)^T & -\dot{\Gamma}(q) \\ J(q) & 0 & 0 \\ \dot{\Gamma}(q)^T & 0 & 0 \end{bmatrix}$$ (21a)

$$b = \begin{bmatrix} 0 \\ -\dot{\zeta}_R \\ -\log(\Gamma(q) - 1) \end{bmatrix}$$ (21b)

$u = [q \quad \eta \quad v]^T \in \mathbb{R}^{d_a}$; $d_a = d_q + d_u + 1$. As $0 \leq W, M$ is also positive semi-definite. $\eta \in \mathbb{R}^{d_\eta}$ and $v \in \mathbb{R}$ are the dual decision vectors. $P_{\Omega}^i(h) = [P_{\Omega}^i(h_1) \ldots P_{\Omega}^i(h_a)]$ is the element-wise $\Omega-$projection operator defined as

$$P_{\Omega}^i(h_i) = \begin{cases} u_i^- & h_i < u_i^- \\ h_i & u_i^- \leq h_i \leq u_i^+ \\ u_i^+ & u_i^- < h_i \end{cases} \quad \forall i \in \{1, \ldots, a\}.$$ (22)

$u^+$ and $u^-$ are the bounds of the primal-dual decision vector $u$ defined as

$$u^- = [\theta^-]^{T} \quad u^+ = [\theta^+]^{T}$$ (23)

and $\Omega = \{u \in \mathbb{R}^d_a | u^- \leq u \leq u^+\}$.

**Proof:** Refer to (Zhang 2005) and (Xia and Wang 2000).

**Theorem 3.** By taking $u$ and $((I + MT)P_{\Omega}^i(u - (Mu + b))$ as the output and the input of the system (20), respectively, (20) is passive.

**Proof:** See Appendix F.

**Remark 2.** Theorems 1, 2 and 3 show that all the proposed dynamical systems are passive. Hence, if the robots and the low level torque controllers are passive, the proposed framework for coordinated Multi-Arm system is passive and stable as it is a feedback system of passive elements.

The C++ implementation of the proposed centralized inverse kinematic solver is provided by authors in $QP\_IK\_solver$, see Table 3.

### 5 Learning a Self-Collision Avoidance (SCA) boundary

In this section, we introduce a data-driven approach to approximate the self-collision avoidance (SCA) boundary function $\Gamma(q^\theta)$, used in (17) as a constraint for the IK solver. As per (16), $\Gamma(q^\theta)$ should be of class $C^0$ and $C^1$, and, interestingly, can be formulated as a binary classification problem, $y \leftarrow \text{sgn} (\Gamma(q^\theta))$ for $y \in \{+1, -1\}$, where “collided” joint configurations belong to the negative class (i.e. $y = -1$) and “non-collided” configurations belong to the positive class (i.e. $y = +1$). When $\Gamma(q^\theta) = 1$, $q^\theta$ is at the boundary of the positive class; i.e. the self-collision boundary (Figure 4).

To recall, the configuration space of a multi-arm system is an $n$-dimensional torus ($S^1 \times S^1 \times \cdots \times S^1 (n - \text{times}) = \mathcal{F}^n$) for $n$ DOF (LaValle 2006). Given two robots with 7DOF each, the manifold in which the multi-arm joint-angle vector lies in is $q^\theta \in \mathcal{F}^{14}$. Employing $q^\theta$ as the feature vector for a classification problem can be problematic for several reasons. Firstly, many machine learning algorithms rely on computing distances/norms in Euclidean space, assuming the features are i.i.d. from an underlying distribution in $\mathbb{R}^N$. Hence, a Euclidean norm applied on $q^\theta \in \mathcal{F}^N$, is merely an approximation of the actual distance in the $\mathcal{F}^N$ manifold. In fact, a proper distance metric for joint-angles, i.e. $d(q^\theta_1, q^\theta_2)$ where $q^\theta \in \mathcal{F}^N$, is non-existent. For this reason, most trajectory optimization planning algorithms rely on mapping joint-space configurations to task-space via...
The green data-points represent “collision-free” robot configurations ($y = +1$), while the red data-points represent “collided” robot configurations ($y = -1$). The background colors represent the values of $\Gamma(q_{ij})$; refer to colorbar for exact values, where the blue area corresponds to collision-free robot configurations ($\Gamma(q_{ij}) > 1$), the red area to collided configurations ($\Gamma(q_{ij}) < 1$). The Arrows inside the collision-free region denote $\nabla \Gamma(q_{ij})$. We can see how $\nabla \Gamma(q_{ij})$ pushes the robot configurations away from the self-collision boundary.

For such reasons, and in-line with the trajectory optimization literature, instead of learning our self-collision avoidance (SCA) decision boundary function $\Gamma(.)$ on the joint-angle data $q_{ij}$, we learn $\Gamma(.)$ on the 3D Cartesian representation of the joint-angles $f(q_{ij})$. As illustrated in Figure 5, $f(q_{ij})$ is a vector composed of the 3D Cartesian positions of all joints for the $i$-th and $j$-th robot, computed via forward kinematics. The feature vector for a dual-arm robotic system is thus $f(q_{ij}) \in \mathbb{R}^{3d_1 + 3d_2}$. We posit that, by using $f(q_{ij})$ instead of $q_{ij}$, we can achieve a better trade-off between model complexity and error rate. Moreover, since the output of $\Gamma(.)$ is expected to be a scalar (16), no extra computation is necessary, as $\Gamma(q_{ij}) \equiv \Gamma(f(q_{ij}))$. The linear inequality constraints in (17d) require $\nabla \Gamma(.)$ to be $\kappa'$, this can also be provided by either SVMs or NNs.

In this work, we favor the use of SVMs, for two main reasons: (i) Learning a SVM is a convex optimization problem; hence, we can always reach a global optimum, whereas NNs rely on heavy parameter tuning and multiple initializations in order to avoid local minimum solutions. (ii) SVMs yield sparser models than NNs for high-dimensional non-linear classification problems, leading to better runtimes at the prediction stage. In the following sub-sections, we describe how we learn $\Gamma(f(q_{ij}))$ through the SVM formulation from a simulated dataset of “collided” and “non-collided” robot configurations. We present our method for constructing such a dataset, motivate and propose to use a sparse SVM learning algorithm (Joachims and Yu 2009), to achieve runtime limitations imposed by the robot control loop, as discussed later on.
5.1 $\mathcal{G}^0$ and $\mathcal{G}^1$ Self-Collision Avoidance (SCA) Boundary via SVM

We follow the kernel Support Vector Machine (SVM) formulation and propose to encode $\Gamma(f(q_i))$ as the SVM decision rule. By omitting the sign function and using the RBF kernel $k(f(q_i^j), f(q_i^j)) = e^{-\frac{1}{2\sigma^2}||f(q_i^j) - f(q_i^j)||^2}$, for a kernel width $\sigma$, $\Gamma(f(q_i))$ has the following form,\(^7\)

$$
\Gamma(f(q_i^j)) = \sum_{n=1}^{N_v} \alpha_n y_n f(q_i^j, f(q_i^j)) + b = \sum_{n=1}^{N_v} \alpha_n y_n e^{-\frac{1}{2\sigma^2}||f(q_i^j) - f(q_i^j)||^2} + b,
$$

for $N_v$, support vectors, where $y_i \in \{-1, +1\}$ are the positive/negative labels corresponding to non-collided/collided configurations, $0 \leq \alpha \leq C$ are the weights for the support vectors which must yield $\sum_{n=1}^{N_v} \alpha_n y_n = 0$ and $b \in \mathbb{R}$ is the bias for the decision rule. $C \in \mathbb{R}$ is a penalty factor used to trade-off between maximizing the margin and minimizing classification errors. Given $C$ and $\sigma$, $\alpha_n$’s and $b$ are estimated by solving the dual optimization problem for the soft-margin kernel SVM (Scholkopf and Smola 2001). Moreover, (24) naturally yields a continuous gradient as follows,

$$
\nabla \Gamma(f(q_i^j)) = \sum_{n=1}^{N_v} \alpha_n y_n \frac{\partial k(f(q_i^j), f(q_i^j))}{\partial f(q_i^j)}
$$

$$
= \sum_{n=1}^{N_v} \frac{1}{\sigma^2} \alpha_n y_n e^{-\frac{1}{2\sigma^2}||f(q_i^j) - f(q_i^j)||^2} (f(q_i^j) - f(q_i^j))
$$

for the first term is equivalent to (25) and the second term is in fact the Jacobian of each 6D joint position wrt. each joint angle $J(q_i^j) = \frac{\partial f(q_i^j)}{\partial q_i^j}$ for which we have a closed-form solution.

5.2 Self-Collision Avoidance (SCA) Dataset Construction

In order to learn $\Gamma(f(q_i^j))$, we must initially generate a dataset capable of identifying the so-called self-collision boundary. We begin by describing our simplified geometric representation of the robot’s kinematic configuration used to identify “collided” and “non-collided” configurations.

For simplicity, let’s assume a dual-arm setting, with each arm being a KUKA 7DOF (Figure 7). Similar to (Zucker et al., 2013), we simplify the representation of the robot’s structure by fitting spheres to each joint and its adjoining physical structure as shown in Figure 6. By doing so, we generate a discrete representation of the multi-arm robotic system as a set of spheres $S^j = \{s^j_1, \ldots, s^j_7\}$ for a KUKA IIWA 7DOF robot arm. $S^j$ is the geometric representation of $i$-th robot with a set of spheres corresponding to each joint used to construct the SCA dataset. If a tool is attached to the arm, the last sphere is enlarged such that it encapsulates its extremities.

By using spheres as a geometric representation of a joint, we simplify the distance computation between joints. As the distance from any point in a sphere to the nearest obstacle is lower-bounded to $d(c_i - r)$, where $c_i$ is the center of the sphere and $r$ its corresponding radius (Ratliff et al. 2009). Further, the lower-bound between two spheres is the distance between their centers ($c_i^j$) minus the sum of their respective radii ($r_i^j$), for the $k$-th spheres of the $i$-th robot. For example, given $s_1^j$ and $s_2^j$ the lower-bounded distance between them can be computed as $d(s_1^j, s_2^j) = d(c_1^j, c_2^j) - (r_1^j + r_2^j)$.

To identify collision in the dual-arm system, we compute the pairwise distances of the centers of the set of spheres of the $i$-th robot ($S^i$) wrt. the set of spheres of the $j$-th robot ($S^j$) and find the minimum distance $\min[d(c_i^j, c_i^j)]$. We then define a label for each robot configuration $S^j$ as follows,

$$
y(S^j) = \begin{cases} 
-1 & \text{if } \min[d(c_i^j, c_i^j)] < (r_1^j + r_2^j) \\
1 & \text{if } b_- \leq \min[d(c_i^j, c_i^j)] \leq b_+ \\
0 & \text{if } \min[d(c_i^j, c_i^j)] > b_+
\end{cases}
$$

where $r_i^j$ corresponds to the radius of the $k$-th sphere, and $b_-b_+$ correspond to minimum/maximum distances of the “safe” boundary. Specifically, a joint configuration is “collided”, i.e. labeled as $y = -1$, when the $\min[d(c_i^j, c_i^j)]$ between the centers of the closest spheres is less than the sum of the radii of the corresponding spheres, i.e. $(r_1^j + r_2^j)$. In practice, we set the spheres to a fixed radius of 10cm, hence $(r_1^j + r_2^j) = 20cm$. Given that virtually any robot configuration where $\min[d(c_i^j, c_i^j)] > (r_1^j + r_2^j)$ can be considered “non-collided” configurations, we would end up with a heavily un-balanced dataset of “collided”/“non-collided” data-points. We thus, introduce a decomposition of the “non-collided” robot configurations into “boundary”, labeled as $y = +1$, and “safe” configurations, which are not labeled $y = \emptyset$. If $\min[d(c_i^j, c_i^j)]$ lies within a safety margin, denoted by $b_-b_+$, the robots are very close to each other but still safe, see Figure 7. We empirically found $b_- = 30cm$ and $b_+ = 33cm$ to be

---

\(^7\)\(\)
safe boundaries for our dual-arm setting. Hence, a “non-collided” configuration is in fact a “boundary” configuration, as all of the “safe” configurations are filtered out. This has a geometric meaning, rather than finding the margin between “collided” and “safe” configurations, our boundary function will model the tighter margin between “non-collided” and “boundary” configurations. From herein, we consider “boundary” configurations as the “non-collided” configurations.

To generate the positive \( y(S^i) = +1 \) and negative samples \( y(S^j) = -1 \) for our SCA dataset, we sample from all the possible motions of the robots in their respective workspaces and apply (27) to each configuration. To explore all possible joint configurations \( q^i \), we systematically displace all of the joints of both robots by 20\(^\circ\) each. Joints \( q_1^i \), \( q_2^i \), \( q_4^i \), \( q_7^i \) have a range of ±170\(^\circ\), whereas joints \( q_2^i \), \( q_4^i \) and \( q_6^i \) have a range of ±120\(^\circ\). Given the 20\(^\circ\) sampling resolution, this leads to 18 samples for the former group and 13 for the latter. One can see from Figure 6 that sampling joint \( q_7^i \) has no effect on the configuration of the spheres (even when considering a tool attached to it). Hence, the total number of possible configurations is \( 18^3 \times 12^3 \). C++ code to efficiently generate such dataset is provide in the SCA\_boundary\_construction package (Table 3), the user needs only to specify the \( X_{off} \) between robot bases and the DH parameters of each arm. For our dual-arm setting we gathered a dataset of approximately \( 5.4 \) million data-points, \( \approx 2.4 \) million belonging to the “collided” configuration class \( y = -1 \) and the rest to the “non-collided” configurations \( y = +1 \). Due to our systematic sampling of “collided” and “boundary” robot configurations, we can generate such balanced datasets, which is desirable for any learning algorithm.

### 5.3 Efficient SVMs for Large Datasets

Training time of a kernel SVM has a complexity of \( O(N_M^2D) \), where \( N_M \) is the number of samples and \( D \) is the dimension of the data-points. Prediction time, on the other hand, depends on the number of support vectors \( N_sv \) learned through training. In practice, the \( N_sv \) tends to increase linearly with the amount of training data \( N_M \) (Bakir et al. 2004). More specifically, for a kernel SVM \( N_M/N_M \rightarrow B_k \), where \( B_k \) is the smallest achievable classification error by the kernel \( k \) (Steinwart 2003); i.e. in a non-separable classification scenario, to achieve 5% error, at least 5% of the training points must become support vectors. This comes as a nuisance when large training sets are involved, as is the case for our application. A \( N_M \gg \) signifies a dense solution for representing the hyper-plane of the classifier margin \( w = \sum_{y=1}^{N_M} a_y y f(q^{(i)}) \). Naturally, the denser the solutions, the more computationally expensive they are at run-time. This makes dense SVMs infeasible for real-time robot control. In order to achieve fast adaptation for both the desired end-effector positions and self-collisions, the IK solver must run
Figure 9. Performance Comparison of learning exact SVM models on randomly sub-sampled datasets vs sparse SVM models on larger chunks of the dataset. Each model was evaluated on the test set, which contains 2.7 million unseen sample robot configurations. We present accuracy (ACC), F-1 Score (F1), False Positive Rate (plotted as 1-FPR), True Positive Rate (TPR) and $N_{sv}$, (left) With the random sub-sampling method, using the 2nd model ($N_{sv} = 2.7 k$), one can achieve $FPR \approx 2.4\%$ and $TPR \approx 96.19\%$ within the desired $2ms$ runtime limit. (right) With a sparse SVM model trained on $540k$ points we can achieve $FPR \approx 1.45\%$ and $TPR \approx 97.4\%$ $k_{max} = 3000$.

(at most) at a rate of $2ms$. During this cycle, prior to solving (17), both (24) and (26) must be evaluated.

Given the desired control rate (2ms), the specific hardware used to control the robots (i.e. 3.4-GHz i7 PC with 8GB RAM) and the kinematic specifications of each robot, we can define a computational budget for our Self-Collision Avoidance (SCA) Boundary function. This budget translates to, defining a limit of the maximum allowable $N_{sv}$ for our SVM representation of $\Gamma(f(q^{i}))$. In Figure 8, we show a plot of different computation times $^8$ for the evaluation of (24) and (26) for the dual-arm setting shown in Figure 7. We omit computation time of the IK solver as this is presented in detail in Section 6 Figure 16b. These computation times evaluate the implementation of $\Gamma(f(q^{i}))$ and $\Gamma(f(q^{j}))$ from the C++ SVMGrad (Table 3) package, provided by the authors.

According to Figure 8, in order to comply with the $2ms$ runtime requirement, we have a computational budget of $N_{sv} \leq 3k$. Given the size of our dataset, it is not feasible to train SVM models that typically optimize for the dual through a variant of Sequential Minimal Optimization (SMO) (Platt 1999) or SMO-type decomposition methods (Joachims 1999; Bordes et al. 2005; Chang and Lin 2011). The fact that SVM learning algorithms tend to produce dense solutions has been recognized as one of its main weaknesses. To this end, several approaches have been proposed in order to solve the problem of finding sparser solutions to $w$. These can be categorized into: i) post-processing approximations and ii) objective function or optimization strategy modification. The former approaches rely on approximating a sparse solution to an initially dense SVM (through the exact solution). The latter approaches either modify the SVM objective function by imposing sparsity constraints or propose a modified optimization algorithm for with sparsity considerations.

In this work, we choose one of the prevailing approaches which reformulates the SVM optimization problem, namely the Cutting Plane Subspace Pursuit (CPSP) method introduced by (Joachims and Yu 2009); as it directly estimates a solution to the hyper-plane with a strict bound on the number of support vectors $k_{max}$. A short motivation for selecting this method is discussed in Appendix H. In short, the CPSP method approximates a sparse hyper-plane by expressing it in terms of a set $B = \{ b_1, \ldots, b_{k_{max}} \}$ of basis vectors $b_i \in \mathbb{R}^{d_{sv}}$ ($k_{max}$) (not necessarily training points) as follows,

$$w = \sum_{i=1}^{k_{max}} \alpha_i \Phi(b_i).$$

The optimization algorithm to estimate (28) then focuses on pursuing such a subspace through the fixed-point iteration approach for RBF kernels (Scholkopf and Smola 2001). The learned basis vectors $B$ and $\alpha_i$’s can be directly used in (24) and (25). We direct the interested reader to (Joachims and Yu 2009) for theoretical equivalence proofs and implementation details of this learning approach.

Learning Performance: We begin our $\Gamma(f(q^{i}))$ learning performance analysis by presenting results from learning exact SVMs from small sub-samples of the 5.4m point dataset. To generate such comparison, the libSVM library (Chang and Lin 2011)$^9$ was used for learning the SVM models, cross-validation was performed with routines from ML toolbox$^{10}$ to find the optimal hyper-parameters. A 50% split for training/validation/testing datasets was used to generate such evaluations. We evaluate 5 models with increasing complexity $N_{sv} = \{ 1.7 k, 2.7 k, 3.7 k, 4.7 k, 9.7 k \}$. These were learned from using $\{0.5\%, 1\%, 1.5\%, 2\%, 5\% \}$ of the training set (i.e. 2.7 million data-points)$^{11}$. For each model, a 10-fold Cross-Validation was performed to find the optimal hyper-parameters $C$ and $\sigma$ which yield the best trade-off between $N_{sv}/N_M$ and classification accuracy. The search space of each hyper-parameter was log-spaced in the following ranges $C = \{ 10^{-3}, 10^4 \}$ and $\sigma = [0.2, 2]^2$. In our application, we care about correctly classifying the negative class (i.e. “collided” configurations), for this we have two objectives:

- Minimize False Positive Rate (FPR): $FPR = \frac{FP}{TP+FP}$, otherwise known as fall-out error, quantifies the probability of negative samples ($y = -1$) being classified as positive ($y = +1$). This is equivalent to maximizing
the True Negative Rate ($TNR = 1 - FPR$). Classifying “collided” configurations ($y = -1$) as “non-collided” configurations ($y = +1$) yields a False Positive ($FP$). This error is critical as it would cause the IK solver to move the robots into an infeasible region, leading to collision and possibly permanent damage.

- **Maximize True Positive Rate ($TPR$):** The $TPR = \frac{TP}{TP+FN}$, otherwise known as Recall or Sensitivity, quantifies the probability of positive samples ($y = +1$) being classified as positive ($y = +1$). This is equivalent to minimizing the False Negative Rate ($FNR = 1 - TPR$). Classifying “collided” configurations ($y = +1$) as “non-collided” ($y = -1$) yields a False Negative ($FN$). This error is not as critical as the former, but it has an effect on the performance of the IK solver, as classifying “non-collided” configuration as “collided” would restrict the IK solver to move the robots into regions that are indeed feasible.

For reference, we also present accuracy $ACC = \frac{TP+TN}{TP+TN+FP+FN}$ and F1-Score $F1 = \frac{2TP}{2TP+FP+FN}$. As can be seen from Figure 9, we can achieve optimal error rates on the testing set of $FPR \approx 1.3\%$ and $TPR \approx 97.54\%$, with 5% of the training dataset, albeit surpassing the $N_S$ limit. One might argue that, with such high performance of models trained on a minuscule amount of data (relative to the complete dataset), perhaps such a large dataset is not necessary. This is related to the SCA dataset construction procedure (Section 5.2), where we set the joint sampling interval to 20\(^{\circ}\). An analysis of the performance of SVM models trained on datasets with lower sampling resolutions is provided in Appendix G. In short, as we increase the joint sampling resolution, less “collided” configurations are seen, resulting in drastic increases in $FPR$.

From Figure 9, we can see that with the 2nd model (i.e. 1% of training data), we achieve error rates of: $FPR \approx 2.4\%$ and $TPR \approx 96.19\%$ withing the computational budget. This is quite acceptable performance, however, due to the delicacy of our application we seek to achieve the best solution possible, i.e. at least $FPR \approx 1\%$. In Figure 9, we present the results of using the CPSP SVM learning approach on different sub-sets of our training data limited to a support vector budget of $k_{max} = 3000$, specifically $\{2.5\%, 5\%, 10\%, 20\%\}$. As can be seen, for the models learned on datasets with the same size as the exact SVM solutions, the results are marginally lower. However, as the number of training-points increases the error rates improve as much as $FPR = 1.5\%$ and $TPR = 97.4\%$ for a training set of 540k points. By using this sparse learning method we have proven that optimal error rate can be achieved with minimal model complexity.

### 6 Empirical Validation

The performance of the proposed framework is implemented on two different real dual-arm platforms. On the first platform, the coordination between the arms and with the object is evaluated. The second experimental set up is designed to evaluate the performance of dual-behavior and the self-collision avoidance.

#### 6.1 First experimental set-up

The proposed framework is implemented on a real dual-arm platform, consisting of two 7 DOF robotic arms, namely a KUKA LWR 4+ and a KUKA IIWA mounted with a 4 DOF Barrett hand and a 16 DOF Allegro hand. As the results from the first set-up were presented in our previous work, we only summarize the main points. For more information, the readers are referred to (Salehian et al. 2016a).

The empirical validation is divided into two parts that demonstrate the controller’s ability: (i) coordinate the multi-arm systems; (ii) rapidly adapt bi-manual coordination to intercept a flying object, without using a pre-defined model of the object’s dynamics. As the synchronized behavior is the only desired behavior in this section, the value of synchronization parameters are manually set to one.

#### 6.1.1 Coordination Capabilities

The first scenario is designed to illustrate the coordination capabilities of the arms with each other through the virtual object. As the human operator perturbs one of the robot arms, the virtual object is perturbed as well, resulting in a stable synchronous motion of the other unperturbed arm (Fig. 10). Since we offer a centralized controller based on the virtual object’s motion, there is no master/slave arm; thus, when any of the robots are perturbed, the others will synchronize their motions accordingly.

#### 6.1.2 Reaching for Fast Flying Objects

The second scenario is designed to show the coordination between the robots and a fast moving object, where a rod (150 x 1 cm) is thrown to the robots from 2.5 m away, resulting in approx. 0.56 s flying time. Due to inaccurate prediction of the object trajectory, the feasible intercept points need to be updated and redefined during the motion execution. The new feasible intercept point is chosen in the vicinity of the previous one to minimize the convergence time. As the motion of the object is fast and the predicted reaching points are not accurate, the initial values of $\gamma$ in (15) are set to 0.5. This decreases the convergence duration of the robots to the real object. Snapshots of the real robot experiments are shown in Fig. 11. Visual inspection of the data and video confirmed that the robots coordinately follow the motion of the object and intercept it at the vicinity of the predicted feasible intercept point.

#### 6.2 Second experimental set-up

The proposed framework is implemented on a dual-arm platform, consisting of two 7 DOF KUKA IIWA robotic arms mounted with a 2 finger Robotiq gripper and a 16 DOF Allegro hand. The robots are controlled via Fast Research Interface (FRI) at joint impedance mode. The fingers are controlled with joint angle position controllers in two states: Open, Close. All the hardware involved (e.g. arms and hands) are controlled by one 3.4-GHz i7 PC. The position of the feasible reaching points of the objects are captured by an Optitrack motion capture system from Natural point at 240 Hz. As the outputs of the vision system are noisy, a Savitzky-Golay filter is used to smooth the position of the object and estimate velocity and acceleration from these position measurements.
6.2.1 Dual Behavior Capabilities. The first scenario is designed to illustrate the dual-behavior capabilities of the arms. The asynchronous behavior of each robot is to reach a fixed target (see Fig.12) or follow the hands of operator 1 who stands between the arms (see Fig.13). The synchronized behavior is to coordinately reach an object brought by operator 2. The vision inspection of the data shows that when operator 2 moves the object toward the robots, based on (8), the value of \( \tau_c, \forall i \in \{1, 2\} \) smoothly increases to one. Hence, a smooth transition from the unsynchronized behavior to the synchronized behavior is achieved; see Fig. 12 and Fig. 13. As there is no full coordination between the arms while the value of \( \tau_c, \forall i \in \{1, 2\} \) is less than one, perturbing one arm does not affect the motion of other arm, see Figure 13(a),(b),(c),(d). While the arms are allocated to the synchronized behavior, due to (15), the arms successfully track, coordinate with and intercept the object and with each other, see Figure 13(g),(h).

6.2.2 Self-Collision Avoidance. Even though, the self-collision avoidance is successfully tested in all the experiments, we designed two different scenarios to highlight the performance of the inverse kinematic solver with this constraint. As the motion of the fingers are not considered in the generated data-set, see Section 5.2, we removed the hands in these two scenarios. In the first scenario, both robots move on straight lines to reach fixed and predefined targets. The targets are picked such that the arms surely collide with each other if the collision avoidance constraint is not considered. Figure 14c shows the initial and the final configurations. As it can been seen from Figures14a the value of \( \Gamma(.) \) is coordinately updated during the motion execution with respect to the robot configurations and when it is less than 2, based on (17d), the robots are moved away from each other to increase their distance. The minimum distance between the robots’ segments are shown in Figure 14b. In the second scenario, the end-effectors are following the hands of an operator. Visual inspection of the data and video confirmed that the robots follow the targets while any collision between them are avoided. A partial result of this experiment is presented in the accompanying video, see Multimedia Extension 1.

6.2.3 Systematic assessment

Coordination and Adaptation Assessment. To systematically assess the performance of the proposed framework. We design a handover scenario, where an operator holds an object and moves toward the arms and hand overs the object to the robots. The robots’ hands are triggered to close when the distance between the arms and the object is less than 1cm. The success rates of our experiments are measured by defining a Boolean metric; i.e. success or
Figure 12. Snapshots of the video illustrating of dual behavior capabilities. The target of synchronous and asynchronous behaviors are highlighted in (a) by the green square and the blue circles, respectively. Initially, the robots are allocated to the asynchronous behavior. Hence, the robots moves toward the asynchronous targets in (b) and (c). In (d), $\gamma_i \approx 1 \ \forall i \in \{1, 2\}$ as the operator moves the object toward the arms; i.e. the synchronous behavior. Consequently, the robots coordinately and simultaneously reach and intercept the object at the desired reaching points.

Figure 13. Snapshots of the video illustrating of dual behavior capabilities. The asynchronous behavior is to follow the hands of the operator who is inside of the robot workspaces. When the operator moves the object away from or toward the arms, the synchronization parameter smoothly goes to 0 and 1, respectively.

Figure 14. The distance between the bases of the arms are $[0.0 \ -1.3 \ 0.14]^T m$. The targets are $[0.0 \ -0.6 \ 0.55]^T m$ for the arms of the left and right, respectively. In (a), the initial and the final configurations (transparent one) are depicted. (b) and (c) show the minimum distance between the arms and the value of $\Gamma$ during the motion execution. As it is shown $\Gamma$ never goes bellow 1 which indicates that the motion of the arms are safe.
The overall success rate of the experiment is 86.7%. Failures are mostly due to inaccuracies in the measurement of the object’s state. To find the position of the reaching areas and the orientation of the object, all the five markers must be visible to the cameras. The objects’ tracking markers were occluded partly when the objects were covered by the robotic arms or the operator. In 5 out of 8 cases, one or two out of the five markers were not detected accurately when the object was close to the robots, hence either the robots converged to a wrong position or the synchronization parameters were changing undesirably. These two cases can be detected easily. In the first case, the hands are closed where there is no object. In the second case, the robots rapidly move back and forth. In 2 out of 8 cases, the robots started moving very late as the object predicted motion was completely wrong. In this case, the object was inside of the robots’ workspaces when the robots started moving. As the motions of the objects was not extremely fast, the IK solver was able to accurately generate the joint space trajectory and only in 1 out of 8 cases it failed to track the desired end-effectors’ motions. In this case, the operator suddenly changed the object’s orientation when it was about to be intercepted by the robots. Hence, the reaching points on the object became kinematically infeasible for the robots to reach.

To systematically study the performance of the IK solvers and sensitivity of the framework to unmeasured object positions, two sets of simulations were designed to reach for

---

**Table 1.** The details of the systematically assessment experiment. All the positions are expressed with respect to the base of the KUKA iiwa 7 robot. The starting positions are randomly chosen by the operator. The robots do not move till the first intercept point is calculated, we call the position of the object at this time the first point. The first 0.4m of the objects’ motions in x direction are used to initialize the object prediction trajectory, see Section 3.1.

<table>
<thead>
<tr>
<th>Weight</th>
<th>Material</th>
<th>The bases distance</th>
<th>Initial position (m)</th>
<th>First point (m)</th>
<th>Duration (s)</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bumper</td>
<td>2.2</td>
<td>Plastic</td>
<td>0.2 ± 1.4 0.1</td>
<td>-3.6 ± 0.2 -0.9 ± 0.2 0.6 ± 0.2</td>
<td>3 ± 1.6</td>
<td>85%</td>
</tr>
<tr>
<td>Fender</td>
<td>2.4</td>
<td>Metallic</td>
<td>0.0 ± 1.3 0.1</td>
<td>-3.8 ± 0.5 -0.7 ± 0.3 0.6 ± 0.1</td>
<td>2.3 ± 1.2</td>
<td>90%</td>
</tr>
</tbody>
</table>

---

**Figure 15.** The car parts which are used for the systematic assessment of the framework.

**Figure 16.** The results of performance of the solvers for (17) in terms of computation time and the smoothness of the motion. DS stands for the dynamical system (20).
Figure 17. Spatial Variation of final intercept points. For clarity, only ten runs (i.e. trajectories) of the object and robots’ motions are shown. For each run a different object trajectory and final intercept points were observed. Experimental results verified that the robots intercept the object in synchrony.

Figure 18. The position of the end-effector, the virtual object generated by (2) and (11), respectively. These trajectories are illustrated from the first point till the stop point. As expected, both arms intercept the reaching positions on the object at the same time. In order to avoid any internal forces, the robots are stopped once the fingers are closed on the object.

a moving box. The size of the box is same as the size of the bumper. In both scenarios, the object is moving toward the robots on a straight line. The simulations are conducted in the kuka-rviz environment, see Table 3.
higher than other two approaches. The smoothness of the standard deviation of the computation time of (20) is much higher. As initialization of the dynamical system (20) plays important role in the convergence duration, the computation time of (17) is set to 10\(^{-4}\). The computation time of each solver is illustrated in Fig.16a. As it was expected, CVXgen is the fastest solver and it takes about 0.000358s for it to solve (17) in average. The performance of the our implementation of (20) takes approximately 0.00092s to solve (17). As initialization of the dynamical system (20) plays important role in the convergence duration, the standard deviation of the computation time of (20) is much higher than other two approaches. The smoothness of the trajectory \(L\) is assessed by \(\mathcal{S} = \frac{\text{std}(L)\cdot \text{std}(\dot{L})}{\text{mean}(L)\cdot \text{mean}(\dot{L})}\), where a smaller value of it indicates a smoother motion. As it is shown in Fig.16b, the result of (20) is much smoother than the other methods. It was expected as (20) calculates the desired motion at the acceleration level. Hence, the output of (20) can directly be transmitted to the robots, but the outputs of either Nlopt or CVXgen need to be filtered. As the computation power was the main criterion for choosing the IK solver for us, we mostly used CVXGEN during the experiments.

**IK Solver Performance.** In the first set of simulations, the performance of the three solvers of (17) (CVXgen, Nlopt and the dynamical system (20)) are assessed in terms of computation time and the smoothness of the generated joint motions. The initial velocity of the object is fixed but the initial position is randomly chosen within the range of \([-3.5 \pm 0.05, -0.45 \pm 0.05, 0.8 \pm 0.05]\) m. The simulation is repeated 5 times for each solver which results in more than 5 * 35000 data points. The termination tolerance of the solvers is set to 10\(^{-4}\). The computation time of each solver is illustrated in Fig.16a. As it was expected, CVXgen is the fastest solver and it takes about 0.000358s for it to solve (17) in average. The performance of the our implementation of (20) takes approximately 0.00092s to solve (17). As initialization of the dynamical system (20) plays important role in the convergence duration, the standard deviation of the computation time of (20) is much higher than other two approaches. The smoothness of the trajectory \(L\) is assessed by \(\mathcal{S} = \frac{\text{std}(L)\cdot \text{std}(\dot{L})}{\text{mean}(L)\cdot \text{mean}(\dot{L})}\), where a smaller value of it indicates a smoother motion. As it is shown in Fig.16b, the result of (20) is much smoother than the other methods. It was expected as (20) calculates the desired motion at the acceleration level. Hence, the output of (20) can directly be transmitted to the robots, but the outputs of either Nlopt or CVXgen need to be filtered. As the computation power was the main criterion for choosing the IK solver for us, we mostly used CVXGEN during the experiments.

**Sensitivity to noise.** In the second set of simulations, the robustness of the framework to noise and unmeasured object position is assessed. The simulation is repeated 165 times in total for three different object velocities; i.e. 0.25\(\frac{m}{s}\), 1.25\(\frac{m}{s}\) and 3.75\(\frac{m}{s}\). Results from this evaluation indicate that the interception error is directly correlated to the percentage of unmeasured object points and the velocity of the moving object (see Fig. 20). Thus, the faster the object, the more sensitive the system is to tracking inaccuracies.

7 Summary and Discussion

In this paper, we proposed a dynamical system based unified framework for generating motion of multi robotic arms to either coordinately reach a moving object or reach stably to a desired point. If provided with robotic arms that can travel fast enough, our algorithm can select the most feasible robotic arms to intercept the object in coordination and with the desired velocity aligned to the object while any collision between the arms are avoided. For selecting the most feasible robotic arms, we define a parameter (i.e. the \(\text{synchronization}\) parameter) to assign the robots to the appropriate behavior; i.e. the \(\text{asynchronous}\) or \(\text{synchronous}\) behaviors. The \(\text{synchronization}\) parameter varies between zero and one based on the feasibility criteria.

The stability and convergence of the proposed dynamical system depends on ensuring conditions (13). As there are no constraints on the magnitude of the eigenvalues \(|\lambda_{ij}|\) of \(A_{ij}\) \(\forall(i,j) \in \{(1,1),(1,2),\ldots,(N_R,d_{IN})\}\), there is no analytical proof that (11) is fast enough to converge to \(\gamma_0\) in time. To address this challenge, a potential direction would be to estimate the parameters of (11) with respect to the stability and the convergence rate constraints. One way to calculate the convergence rate of a dynamical system is to prove that it is exponentially stable (Khalil 2002).

To solve the quadratic programming problem, we used three different approaches. First a dynamical system based approach, Eq.(20). Second, nonlinear programming solver (Johnson 2016). The third approach was CVXGEN, introduced in (Mattingley and Boyd 2012). Each of these approaches has its own advantages and disadvantages. Using the first approach is advantageous in the way that the passivity of the dynamical system can be proven. Hence, the unified framework stays passive and stable as long as the robots are passive. In addition, the first approach result in smoother joint motions. The main advantage of the second approach is its interface. Nlopt is very user friendly and it is possible to test several solvers, but it is computationally costly. The main advantage of the third approach is the computational cost. As it has been shown in (Mattingley
when the percentage of unmeasured position is 85\% ± speeds, three noise powers and four percentages of unmeasured positions. We only consider trials when the box passes through the vicinity of this point to avoid high accelerated motions.

arms to extract the intercept posture. However, the proposed feasible posture extraction (Kim et al. 2014) for multiple predictable, one can simply generalize a single-robot arm the convergence. If we assume that the object trajectory is γ is reachable, much faster than the rate of update. Second, when the object the vicinity of the previous ones; i.e. the convergence rate is evaluated or the simulations.

The interception error is the average of the minimum distance between the box and the end-effectors. The initial positions of the box are randomly chosen within the range of $[-3.5 \pm 0.1 \ -0.45.0 \pm 0.1 \ 0.8 \pm 0.1]^{T}$ m. The distance between the arms and the size of the box are same as the bumper scenario. The simulations are repeated for each combination of three object’s speeds, three noise powers and four percentages of unmeasured positions. We only consider trials when the box passes through the robots workspaces. The measurement noise is simulated with pseudo-random values within the range of ±0.005, ±0.005 and ±0.0005m. The results of the worse case are not illustrated as the robots were not able to follow the object; i.e. the worse case is when the percentage of unmeasured position is 85\% and the box’s velocity is 3.73 m s$^{-1}$.

and Boyd 2012), CVXGEN is computationally very efficient. The main shortcoming of the third approach is the stability of the closed loop system which can not be proven; however we have not seen any unstable behaviors during the real world evaluations or the simulations.

Throughout the proofs, we assume that the intercept point is a fixed attractor. However, due to the imperfect prediction of the object trajectory, the feasible intercept postures need to be iteratively updated. Nevertheless, this does not affect the convergence of the system for two main reasons. First, when γ < 1 the new feasible intercept point is chosen in the vicinity of the previous ones; i.e. the convergence rate is much faster than the rate of update. Second, when the object is reachable, γ = 1, the virtual object converges to the real object and the position of the intercept point does not affect the convergence. If we assume that the object trajectory is predictable, one can simply generalize a single-robot arm feasible posture extraction (Kim et al. 2014) for multiple arms to extract the intercept posture. However, the proposed algorithm is not restricted to this assumption. In this work, the feasible intercept point is calculated to make sure that, firstly, the object is passing through the robots’ reachable space and, secondly, to move the robots in advance to the vicinity of this point to avoid high accelerated motions.

In this paper, we control the motion of the arm from initial condition (palm open, robots far from the object) to the point when the arms reach the object and the fingers are about to close on the object. Hence, there are no interaction forces (which would arise once in contact with the object). Once the fingers close on the object, the robots-object system become a closed kinematic chain. In this case, devising an appropriate force controller is necessary to coordinate the robots. Future work in multi-arm manipulation will be directed to address the challenges of devising an appropriate force controller and the transition between the position and force controller.

One of the main advantages of the proposed framework is the computational cost. The implementation shows that the overall computation is rapid, thus enabling us to not only compute the feasible intercept point, the reaching motion and solve centralized IK problem, but also to control a 33 (7 + 7 + 16 + 1) DOF system with one 3.4-GHz i7 PC.

Finally, we are currently working on improving the performance of (15), by learning its parameters via a convex optimization problem with respect to the workspace constraint of the robots. With this approach, we could ensure that the performance of the dynamical system is optimal and the generated motion is not infeasible for the robots to follow.

Acknowledgements
Research leading to this work was supported by the EU project Cogimon H2020 – ICT – 23 – 2014. The implementation and evaluation presented in this work has been partly supported by KUKA in the scope of the KUKA Innovation Award 2017. The authors kindly thank Leonardo Urbano for his help during the experiments.
Notes
1. The motion generator is fully observable.
2. We set $0 \ll k$ as we are mainly interested to have only two behaviors.
3. We assume that the dynamical system (11) is fast enough to converge to an acceptable neighborhood around the desired trajectory $\gamma^\tau (T^*)^T$ before $T^*$; i.e. $\|\mathbf{z}^\tau (T^*) - \gamma^\tau (T^*)\| \leq \varepsilon$
4. As an example, $N_k = 2$, $q = q^{12}$. Based on our experience, due to hardware limitations, it is not possible to construct a data set of collision boundaries for more than two 7-DOF KUKA arms at once; e.g. for 3 arms, the size of the data set approximately is $(3\times7\times3) \times 1000^2$ while it is $(3\times7\times2) \times 1000^2$ for 2 arms.
6. This fact has been implicitly proven in (Fang et al. 2015). The authors learned SVMs on joint-angle data $q^{ij}$ for limb-pairs of a humanoid robot, to classify safe/dangerous configuration regions. The reported misclassification rate was between 15 – 35%, they thus relied on a Quadratic Lagrangian Interpolation (QLI) function to improve the classification error on-line. Since “collided” and “non-collided” configurations can differ by $< 5$deg rotation of 1 joint angle, the underlying binary classification problem is that of an extremely overlapped dataset. For an SVM to yield high-accuracy the model must be highly complex, i.e. large number of support vectors. (Fang et al. 2015) reported a training set size of $\approx 6k$ points. We believe they had such a small training set in order to keep the complexity of the SVM low for runtime, at the cost of low prediction accuracy.
7. For sake of simplicity, the superscripts $(ij)$ of $\alpha^{ij}$, $N^{ij}_d$, $y^{ij}$ and $\sigma^{ij}$ are dropped in the paper.
8. This runtime includes the construction of the feature vector $f(q^{ij})$ as well as the construction of $J(q^{ij})$ which must be multiplied by (25), this involves vector and matrix constructions as well as matrix multiplication. The https://eigen.tuxfamily.org/index.php?title=Main_Page is used for such operation, which has underlying dynamic allocation strategies, this induces the std. seen in Figure 8.
9. libSVM: https://www.csie.ntu.edu.tw/~cjlin/libsvm/
10. ML_toolbox: https://github.com/epfl-lasa/ML_toolbox
11. Due to implementation constraints, one of the arms are connect to another PC. Apart from this connection, no computation is done on the other PC.
12. The range of $\sigma$ was determined by the values of tails generated by a fitted Gaussian distribution on the pair-wise Euclidean norm of all data-points.
13. It is important to clarify that each DOF of the hand is separately controlled. For each state (open/close) a target joint configuration is defined. Once the states are triggered, we use a PD-joint position controller to guide the fingers to the desired configurations.
14. Due to implementation constraints, one of the arms are connect to another PC. Apart from this connection, no computation is done on the other PC.
15. References

---

Acknowledgments

The authors wish to thank Prof. M. Aghili for his support and encouragement.\n
---

Appendix A: Proof of Theorem 1

---

Appendix B: Detailed Equations

---

Appendix C: Additional Experiments

---

Appendix D: Conclusion

---

Appendix E: Future Work

---

Appendix F: References

---

Appendix G: Acknowledgments

---

Appendix H: Abbreviations

---

Appendix I: Supplementary Material

---

Appendix J: Supplementary Figures

---

Appendix K: Supplementary Tables

---

Appendix L: Supplementary Code

---

Appendix M: Supplementary Videos

---

Appendix N: Supplementary Data

---

Appendix O: Supplementary Notes

---

Appendix P: Supplementary References

---

Appendix Q: Supplementary Figures

---

Appendix R: Supplementary Tables

---

Appendix S: Supplementary Code

---

Appendix T: Supplementary Videos

---

Appendix U: Supplementary Data

---

Appendix V: Supplementary Notes

---

Appendix W: Supplementary References

---

Appendix X: Supplementary Figures

---

Appendix Y: Supplementary Tables

---

Appendix Z: Supplementary Code

---

Appendix AA: Supplementary Videos

---

Appendix BB: Supplementary Data

---

Appendix CC: Supplementary Notes

---

Appendix DD: Supplementary References

---

Appendix EE: Supplementary Figures

---

Appendix FF: Supplementary Tables

---

Appendix GG: Supplementary Code

---

Appendix HH: Supplementary Videos

---

Appendix II: Supplementary Data

---

Appendix JJ: Supplementary Notes

---

Appendix KK: Supplementary References

---

Appendix LL: Supplementary Figures

---

Appendix MM: Supplementary Tables

---

Appendix NN: Supplementary Code

---

Appendix OO: Supplementary Videos

---

Appendix PP: Supplementary Data

---

Appendix QQ: Supplementary Notes

---

Appendix RR: Supplementary References

---

Appendix SS: Supplementary Figures

---

Appendix TT: Supplementary Tables

---

Appendix UU: Supplementary Code

---

Appendix VV: Supplementary Videos

---

Appendix WW: Supplementary Data

---

Appendix XX: Supplementary Notes

---

Appendix YY: Supplementary References

---

Appendix ZZ: Supplementary Figures

---

Appendix AAA: Supplementary Tables

---

Appendix BBB: Supplementary Code

---

Appendix CCC: Supplementary Videos

---

Appendix DDD: Supplementary Data

---

Appendix EEE: Supplementary Notes

---

Appendix FFF: Supplementary References

---

Appendix GGG: Supplementary Figures

---

Appendix HHH: Supplementary Tables

---

Appendix I: Supplementary Material

---

Appendix J: Supplementary Figures

---

Appendix K: Supplementary Tables

---

Appendix L: Supplementary Code

---

Appendix M: Supplementary Videos

---

Appendix N: Supplementary Data

---

Appendix O: Supplementary Notes

---

Appendix P: Supplementary References

---

Appendix Q: Supplementary Figures

---

Appendix R: Supplementary Tables

---

Appendix S: Supplementary Code

---

Appendix T: Supplementary Videos

---

Appendix U: Supplementary Data

---

Appendix V: Supplementary Notes

---

Appendix W: Supplementary References

---

Appendix X: Supplementary Figures

---

Appendix Y: Supplementary Tables

---

Appendix Z: Supplementary Code

---

Appendix AA: Supplementary Videos


$V$ is positive definite, radially unbounded, continuous, and continuously differentiable. The derivative of $V$ with respect to time is

$$\frac{dV}{dt} = \frac{1}{2} \left( (x^R_i(t) - \tau_i(t))x_i^V(t) - (\tau_i(t) - 1)x_i^R(t) + \tau_i(t)x_i^R + \tau_i(t)x_i^R \right) + \frac{1}{2} \left( (\tau_i(t) - 1)x_i^R(t) + (x_i^R(t) - \tau_i(t)x_i^R(t) + (\tau_i(t) - 1)x_i^R(t) - \tau_i(t)x_i^R(t) + \tau_i(t)x_i^R(t) ) \right)$$

By substituting (2) into (30), we have:

$$\frac{dV}{dt} = \frac{1}{2} \left( (A_i(\theta_i))(x_i^R - \tau_i(x_i^R - x_i^V)) \right)^T P_i^R \left( x_i^R(t) - \tau_i(t)x_i^R(t) + (\tau_i(t) - 1)x_i^R(t) + (\tau_i(t) - 1)x_i^R(t) - \tau_i(t)x_i^R(t) + \tau_i(t)x_i^R(t) \right)$$

$$+ \frac{1}{2} \left( \sum_{k=1}^{d_i} \theta_{ik} A_{ik}^T P_i^R (x_i^R(t) - \tau_i(t)x_i^R(t) + (\tau_i(t) - 1)x_i^R(t) - \tau_i(t)x_i^R(t) + \tau_i(t)x_i^R(t) \right)$$

Hence, $\psi(.) = -(x_i^R)^T \sum_{k=1}^{d_i} \theta_{ik} (A_{ik}^T P_i^R + P_i^R A_{ik}) x_i^R$. Hence (34) is satisfied. Furthermore, as the memoryless system $x_i^R = (x_i^R)^{-1} Y_i$ is passive, the dynamical system given by (2) is passive when $\tau_i(x_i^R - x_i^V) + \tau_i x_i^V - A_i(\theta_i)(x_i^R(t) + \tau_i(t)x_i^R(t) - x_i^V))$ and $x_i^R(t)$ are the input the output, respectively.

D Proof of Theorem 2, Part A

As $\dot{x}_i^V = x_i^V, \forall i \in \{1, \ldots, N_R\}$, (11) can be written as:

$$\dot{x}_i^V(t) = -\gamma_i x_i^O + \chi_i A^V(x_i^V - x_i^O)$$

We propose a Lyapunov function as follows:

$$V = \frac{1}{2} (x_i^V(t) - \gamma_i x_i^O)^T P_i^V (x_i^V(t) - \gamma_i x_i^O)$$

$V$ is positive definite, radially unbounded, continuous and continuously differentiable. Substituting, (36) into the derivative of $V$ with respect to time results in:

$$\frac{dV}{dt} = \frac{1}{2} \left( (x_i^V(t) - \gamma_i x_i^O)^T P_i^V A_i x_i^V(t) - \gamma_i o \right) + (x_i^V(t) - \gamma_i x_i^O)^T A_i^T P_i^V (x_i^V(t) - \gamma_i x_i^O)$$

$$\leq 0$$

Therefore, dynamical system (36) and (11) are globally stable; i.e. $x_i^V$ and $x_i^O$ are bounded as $\gamma_i x_i^O$ and $\gamma_i x_i^O$ are bounded. Since $V$ is finite, Barbalat’s lemma (Khalil 2002) indicates that the attractor is globally asymptotically stable;
\[ \lim_{t \to 0} \| x^V(t) - \gamma(t)x^O(t) \| = 0 \]  
\[ \lim_{t \to 0} \| \dot{x}^V(t) - (\gamma(t)x^O(t) + \gamma(t)x^O(t)) \| = 0 \]  
\[ \text{c.q.f.d.} \]

**E  Proof of Theorem 2, Part B**

Consider the following storage function:

\[ V = \frac{1}{2}(x^V)^T P^V (x^V) \]  
\[ (40) \]

Clearly (40) is positive definite, radially unbounded, continuous and continuously differentiable. To simplify the notations, in this section, we consider \( Y = P^V x^V \) and \( U = \gamma x^O + \gamma x^O - A^V \gamma x^O \) as the output and the input of (11). To prove the passivity of (40), we need to show that

\[ \frac{dV}{dt} + \psi(.) \leq U^T Y \quad \exists \psi(.), 0 \leq \psi(.) \]  
\[ (41) \]

The derivative of \( V \) with respect to time is as follows:

\[ \dot{V} = \frac{1}{2}(x^V)^T P^V (x^V) + \frac{1}{2}(x^V)^T P^V (x^V) 
\]
\[ = \frac{1}{2} \left( \gamma x^O + \gamma x^O + A^V (x^V - \gamma x^O) \right)^T P^V x^V + 
\]
\[ \frac{1}{2}(x^V)^T P^V \left( \gamma x^O + \gamma x^O + A^V (x^V - \gamma x^O) \right) 
\]
\[ = (x^V)^T \left( (P^V A^V + (A^V)^T P^V) x^V + U^T Y \right) \]
\[ \leq 0 \]  
\[ (42) \]

Hence, \( \psi(.) = -(x^V)^T (P^V A^V + (A^V)^T P^V) x^V \). Furthermore, as the memoryless system \( x^V = (P^V)^{-1} Y \) is passive, the dynamical system given by (36) and consequently (11) are passive when \( \gamma x^O + \gamma x^O - A^V \gamma x^O \) and \( x^V \) are the input the output, respectively. \[ \text{c.q.f.d.} \]

**F  Proof of Theorem 3**

Consider the following storage function:

\[ V = \frac{1}{2}u^T u \]  
\[ (43) \]

(43) is positive definite, radially unbounded, continuous and continuously differentiable. To simplify the notations, in this section, we consider \( V = u \) and \( U = (I + M^T)P_{21}(u - (Mu + b)) \) as the output and the input of (20). To prove the passivity of (20), we need to show that

\[ \frac{dV}{dt} + \psi(u) \leq Y^T U \quad \exists \psi(u), 0 \leq \psi(u) \]  
\[ (44) \]

The derivative of \( V \) with respect to time is as follows:

\[ \dot{V} = u^T \dot{u} \]
\[ V = u^T (I + M^T)(P_{21}(u - (Mu + b)) - u) \]  
\[ \dot{V} + u^T (I + M^T)u = u^T (I + M^T)P_{21}(u - (Mu + b)) \]
\[ (45) \]

Hence, \( \psi(u) = u^T (I + M^T)u \), which indicates the passivity of (20). \[ \text{c.q.f.d.} \]

**G  Analysis of Joint Sampling Interval Resolution**

In Figure 21, we plot the performance of optimal SVM models trained on datasets with lower sampling resolutions \{20deg, 22deg, 25deg, 30deg, 45deg\}. This led to datasets of the following sizes \{5.4m, 2.3m, 870k, 240k, 8.9k\}, respectively. For each dataset, we performed the same CV procedure to find the optimal parameters \( C \) and \( \sigma \), that yield the best error rates. The error rates reported in Figure 21, are from testing each model on the original 20deg-sampled dataset. Clearly, as the joint sampling interval increases, the dataset size decreases dramatically, this leads to a same trend in performance. The most critical evidence of the fact that, randomly sampling a small percentage from the 20deg-sampled dataset is not equivalent to increasing the joint sampling interval, is the constant decrease in FPR. For every sampling interval \( > 20 \text{deg} \), the optimal SVMs cannot or marginally reach the error-rates achieved by the sub-sampled datasets (Figure 9). The rationale behind this evidence can be easily explained geometrically. With higher joint sampling intervals, less “collided” regions are explored in the robot configuration search procedure, hence, the negative class is not spanned properly and FPR decreases dramatically. On the other hand, by sampling joint configurations with a lower interval, we manage to represent each class properly, and a small randomly sampled percentage of the dataset is capable of exhibiting the underlying class distribution.

![Figure 21. Joint sampling granularity tests. We learn optimal models (concerning classification error) on different datasets created with increasing joint sampling intervals: {20deg, 22deg, 25deg, 30deg, 45deg}). As the sampling interval increases, the dataset size decreases dramatically (from millions to thousands in two steps) as well as the FPR. \( N_M \) represents dataset size. The reported error rates are from the testing set of the 20deg-sampled dataset.](image)

**H  Selection of CPSP method**

The two state-of-the-art sparse SVM approaches are: (i) that of ( Cotter and Serrebro 2013 ), which we will name the Sub-Gradient (SG) method and (ii) the Cutting Plane Subspace Pursuit (CPSP) method introduced by (Joachims and Yu 2009), respectively. The SG approach is a very simple, yet theoretically motivated method to learn a sparse approximation \( \hat{w} \) of \( w \), ensuring an empirical 0/1 error
bound. From an initially dense estimation of $\mathbf{w}$, (Cotter and Srebro 2013) use a randomized classification rule to search for a $\hat{\mathbf{w}}$ by minimizing a loss function $f(\hat{\mathbf{w}})$ through sub-gradient descent. This approximation is bounded by $4\|\mathbf{w}\|^2$, hence it will select the optimal set of support vectors that comply with this bound. This method achieves the best approximation in terms of accuracy compared to other methods to date (Cotter and Srebro 2013). However, it will select as many support vectors as needed to achieve this accuracy, yielding a sparse but not bounded solution. Moreover, as mentioned earlier, the size of our dataset limits our capability of learning a dense solution from all the datapoints. Fortunately, the Cutting Plane Subspace Pursuit (CPSP) algorithm (Joachims and Yu 2009), specifically tackles these problems. Instead of finding the best sparse approximation ($\hat{\mathbf{w}}$) from a dense one ($\mathbf{w}$), it directly estimates a solution to $\mathbf{w}$ with a strict bound on the number of support vectors, by reformulating the SVM optimization problem.

I Notation table, framework schematic and source codes
Figure 22. Block diagram for coordinated multi-arm motion planning for reaching a large moving object. Where $N_R$ represents the total number of robot arms. $T$ represents the motion prediction duration. In this paper, we assume that the low-level controller of the robot is a perfect tracking controller.
Table 2. Nomenclature

<table>
<thead>
<tr>
<th>Variable</th>
<th>Domain</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\varepsilon$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>A small positive number.</td>
</tr>
<tr>
<td>$k$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>A large constant positive number.</td>
</tr>
<tr>
<td>$\mathbf{k}$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>A constant positive number.</td>
</tr>
<tr>
<td>$t$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>Time.</td>
</tr>
<tr>
<td>$\delta_j$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>Minimum likelihood threshold of $j$th robot’s workspace.</td>
</tr>
<tr>
<td>$\delta$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>Minimum joint likelihood threshold.</td>
</tr>
<tr>
<td>$T^*$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>Time when the object is kinematically reachable.</td>
</tr>
<tr>
<td>$N_A$</td>
<td>$\mathbb{N}$</td>
<td>Number of the available robot arms.</td>
</tr>
<tr>
<td>$N_{sv}$</td>
<td>$\mathbb{N}$</td>
<td>Number of the support vectors.</td>
</tr>
<tr>
<td>$N_d$</td>
<td>$\mathbb{N}$</td>
<td>Number of the samples points.</td>
</tr>
<tr>
<td>$n_i$</td>
<td>$\mathbb{R}_{(0,1)}$</td>
<td>Synchronization allocation parameter of $i$th robot.</td>
</tr>
<tr>
<td>$n_i$</td>
<td>$\mathbb{R}_{(0,1)}$</td>
<td>Coordination parameter.</td>
</tr>
<tr>
<td>$d_s$</td>
<td>$\mathbb{N}$</td>
<td>Dimension of the states of the virtual/real object or one robot.</td>
</tr>
<tr>
<td>$d_s$</td>
<td>$\mathbb{N}$</td>
<td>Dimension of the states of all the robots in total.</td>
</tr>
<tr>
<td>$d_o$</td>
<td>$\mathbb{N}$</td>
<td>Number of scheduling parameters of $i$th robot.</td>
</tr>
<tr>
<td>$d_o$</td>
<td>$\mathbb{N}$</td>
<td>Number of the joints of $i$th robot.</td>
</tr>
<tr>
<td>$d_o$</td>
<td>$\mathbb{N}$</td>
<td>Number of the joints of all the robots.</td>
</tr>
<tr>
<td>$d_s$</td>
<td>$\mathbb{N}$</td>
<td>Number of the scheduling parameters of $i$th robot.</td>
</tr>
<tr>
<td>$d_s$</td>
<td>$\mathbb{N}$</td>
<td>Number of the joints of all the robots.</td>
</tr>
<tr>
<td>$q_j$</td>
<td>$\mathbb{R}^{\nu_{i,j}}$</td>
<td>Joint angles of the $i$th robot.</td>
</tr>
<tr>
<td>$q_j$</td>
<td>$\mathbb{R}^{\nu_{i,j}}$</td>
<td>Joint angles of the $i$th and $j$th robots.</td>
</tr>
<tr>
<td>$q_j^+$</td>
<td>$\mathbb{R}$</td>
<td>Angle of $i$th joint of $i$th robot.</td>
</tr>
<tr>
<td>$f(q_j)$</td>
<td>$\mathbb{R}^3$</td>
<td>Cartesian position of the $i$th joint of $i$th robot with respect to the world frame.</td>
</tr>
<tr>
<td>$J_i$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>Jacobi matrix of $i$th robot.</td>
</tr>
<tr>
<td>$J_i^0$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>Position of the $i$th end-effector.</td>
</tr>
<tr>
<td>$J_i^j$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>Position of the $i$th reaching point on the virtual object.</td>
</tr>
<tr>
<td>$\xi$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>Position of the virtual object.</td>
</tr>
<tr>
<td>$\xi$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>Position of the real object.</td>
</tr>
<tr>
<td>$\xi^n$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>$i$th feasible reaching point on the real object.</td>
</tr>
<tr>
<td>$\xi^n$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>$i$th feasible reaching point on the real object.</td>
</tr>
<tr>
<td>$\xi^n$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>Static target of the asynchronous behavior of $i$th robot.</td>
</tr>
<tr>
<td>$\xi^n$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>Static target of the asynchronous behavior of $i$th robot.</td>
</tr>
<tr>
<td>$\alpha_i^n$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>Set of GMM parameters of the workspace model of $i$th robot.</td>
</tr>
<tr>
<td>$\alpha_i^n$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>Set of GMM parameters of the workspace model of all the robots.</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>States of the virtual object’s dynamical system.</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>States of the virtual end-effector.</td>
</tr>
<tr>
<td>$x_i$</td>
<td>$\mathbb{R}^{7 \times 4_d}$</td>
<td>States of the static target of the asynchronous behavior of $i$th robot.</td>
</tr>
<tr>
<td>$\theta_i$</td>
<td>$\mathbb{R}^{7 \times 1}$</td>
<td>Scheduling parameters for pos/orient. dynamics for $i$th robot.</td>
</tr>
<tr>
<td>$A_{g,i}, A_i(\theta_i)$</td>
<td>$\mathbb{R}^{7 \times 7 \times 4_d}$</td>
<td>Affine dependent state-space matrices for $i$th robot.</td>
</tr>
<tr>
<td>$P_{d,i}, Q_i, R_i, Q_{d,i}$</td>
<td>$\mathbb{R}^{4_d \times 4_d}$</td>
<td>Auxiliary matrices which are used in the stability and convergence proofs.</td>
</tr>
<tr>
<td>$\Gamma(\cdot)$</td>
<td>$\mathbb{R}$</td>
<td>Self-collision boundary.</td>
</tr>
<tr>
<td>$\theta_i^+ \theta^-$</td>
<td>$\mathbb{R}^{7 \times 1}$</td>
<td>Conservative lower and upper bounds of the joint limits of $i$th robot.</td>
</tr>
<tr>
<td>$\mu_i$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>Intensity coefficient in the IK solver.</td>
</tr>
<tr>
<td>$\eta$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>Dual decision vector.</td>
</tr>
<tr>
<td>$\nu$</td>
<td>$\mathbb{R}$</td>
<td>Dual decision variable.</td>
</tr>
<tr>
<td>$k(\cdot)$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>RBF kernel.</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>$\mathbb{R}_{&gt;0}$</td>
<td>Kernel width.</td>
</tr>
<tr>
<td>$\chi_i$</td>
<td>${-1,1}$</td>
<td>Positive/negative labels.</td>
</tr>
<tr>
<td>$\alpha_i$</td>
<td>$\mathbb{R}_{(0,1)}$</td>
<td>Weights of the support vectors.</td>
</tr>
<tr>
<td>$C$</td>
<td>$\mathbb{R}$</td>
<td>Penalty factor used in learning the SCA boundary.</td>
</tr>
<tr>
<td>$c^+$</td>
<td>$\mathbb{R}$</td>
<td>Center of the sphere on the $i$th joint of the $i$th robot.</td>
</tr>
<tr>
<td>$r^+$</td>
<td>$\mathbb{R}$</td>
<td>Minimum/Maximum distance of the safe boundary.</td>
</tr>
</tbody>
</table>

Table 3. The implementation toolboxes which are provided by the authors.

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
<th>Language</th>
<th>Uniform Resource Locator (URL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multarm_ds</td>
<td>Centralized motion generator</td>
<td>C++</td>
<td><a href="https://github.com/sinamr66/Multarm_ds">https://github.com/sinamr66/Multarm_ds</a></td>
</tr>
<tr>
<td>QP_IK_solver</td>
<td>Centralized IK solver</td>
<td>C++</td>
<td><a href="https://github.com/sinamr66/QP_IK_solver">https://github.com/sinamr66/QP_IK_solver</a></td>
</tr>
<tr>
<td>SESODS_lib</td>
<td>Estimating parameters of second order LPV based DSs</td>
<td>Matlab</td>
<td><a href="https://github.com/sinamr66/SESODS_lib">https://github.com/sinamr66/SESODS_lib</a></td>
</tr>
<tr>
<td>SVMGrad</td>
<td>RBF kernel SVM and its Gradient</td>
<td>C++</td>
<td><a href="https://github.com/nbfigueroa/SVMGrad">https://github.com/nbfigueroa/SVMGrad</a></td>
</tr>
<tr>
<td>kuka-rviz-simulation</td>
<td>Simulation of the KUKA</td>
<td>C++</td>
<td><a href="https://github.com/epfl-lasa/kuka-rviz-simulation.de">https://github.com/epfl-lasa/kuka-rviz-simulation.de</a></td>
</tr>
</tbody>
</table>