A Computational Algorithm for Dynamic Regrasping Using Non-dexterous Robotic End-effectors

Avishai Sintov* and Amir Shapiro
Department of Mechanical Engineering, Ben-Gurion University of the Negev, Beer-Sheva, Israel

1 STAGE OF THE RESEARCH

The presented research addresses the problem of regrasping an object. That is, to switch between grasp configurations according to designated tasks to be done. The state of the art in regrasping methods is mostly quasi-static motions which requires high dexterous end-effectors. This research intends to approach the regrasping problem with dynamic manipulations. That is, utilize the dynamics of the robotic arm to perform the regrasping operation while grasping the object with a simple non-dexterous end-effector.

As will be discussed further, in this research we intend to perform the dynamic regrasping with two strategies, one is the *in-hand spinning* and the other is the *mid-air flipping*. Currently, an algorithm for motion planning of the in-hand spinning strategy was developed. While approaching the problem and its constraints, a method for dynamic motion planning was needed. We surveyed methods which could enable planning under kinematic and dynamic constraints and incorporate rendevouze planning of two systems (arm and manipulated object). First, the RRT (LaValle and Kuffner, 1999) planning method was examined. However, the RRT method has disadvantage in dynamic planning with time constraint. Therefore, we have modified the RRT and presented the Time-Based RRT method (Sintov and Shapiro, 2014b) which could plan a dynamic motion with a time constraint and use it for rendevouze planning. However, this method was not fully compatible for our needs, and we started developing our own dynamic planning algorithm.

The Semi-Stochastic Kinodynamic Planning (SKIP) algorithm was developed. It was originally developed for the regrasping problem, but is given as a generic planning algorithm for any fully actuated dynamic system. A journal paper of the full algorithm was recently submitted (Sintov and Shapiro, 2014a). The SKIP algorithm is a computational algorithm, partially stochastic, for finding a feasible trajectory for a dynamic system to move from an initial state to a goal state under the kinematic and dynamic constraints of the problem. It is proven to find a solution if such exists under a minimal known probability.

The research is currently in an experimental phase. A planar 3R robotic arm was built and positioned over an inclined hockey table. A set of motion capture cameras provides feedback of the manipulated objects state (position and orientation) at all times. The servo motors also provide angle and angular velocity feedback. These feedbacks with the known Computed-Torque control scheme enables motion of the arm to perform a desired regrasping motion according to the output of the SKIP algorithm. Moreover, a model identification computation method and an adaptive computed torque control was implemented to estimate unknown system parameters. The main objective of the experiments is to validate the performance of our dynamic planning algorithm and the regrasping method.

The next stages of the research is the extension of the algorithm for performing mid-air flipping and finally providing a general computation algorithm given an object's geometry and the desired final grasp. That is, the algorithm will independently choose the desired regrasping strategy given a particular object. For example, for a long cylinder the in-hand spinning motion is not practical and mid-air flipping should be able to complete the motion.

2 OUTLINE OF OBJECTIVES

Dynamic regrasping is a new notion in robotics research. The robot will initially spin the grasped object into mid-air, then capture the object back at a new grasp configuration. This challenge will require several key components:

- A consideration of the robot's *full dynamics* in or-

*The work presented in this paper is Avishai Sintov's PhD research under the supervision of Dr. Amir Shapiro.

Sintov A. and Shapiro A..
A Computational Algorithm for Dynamic Regrasping Using Non-dexterous Robotic End-effectors.

der to impose a desired release trajectory of the free flying object.

- A consideration of the *fingers' opening* process to ensure release the object with a desired initial spin.
- A *visual servo system* that will measure the free-flying object trajectory.
- Synthesis of a dynamic *capturing trajectory* for the robot arm and its multi-finger hand.
- A consideration of the *fingers' closing* process in order to achieve object re-capturing.

The project will evaluate two qualitatively distinct dynamic regrasping strategies. The first strategy is termed *in-hand spinning.* The robot will release the object into mid-air with an initial spin while opening the fingers into a partial *cage formation.* The robot arm will then move the caging fingers in tandem with the flying object, but without disturbing the object. When the spinning object will reach a new desired orientation relative to the hand's mechanism, the fingers will capture the object by closing the cage around the flying object. The second strategy is termed *mid-air flipping.* Much like pancake or pizza flipping, the robot arm will initially spin the object high above the robot arm mechanism. This will give the visual servoing system and the robot arm ample time to measure the object's precise trajectory under the influence of gravity. Based on this information, the arm will subsequently move with open fingers in order to capture the flying object at a pre-designated rendezvous location in space. The research will evaluate the relative merits of in-hand spinning and mid-air flipping, with the aim of identifying which type of objects are most suitable for each strategy.

The research will additionally work out practical implementation procedures for the two regrasping strategies. One promising procedure will start with a selection of the robot arm release trajectory (position and velocity), based on optimization criteria such as total travel time, motors effort, trajectory smoothness, and obstacle avoidance. The procedure will then formulate a tracking control law based on classical *inverse dynamics.* The control law will subsequently be modified to include corrective terms that compensate for small modeling errors of the robot's dynamics as well as the small finger placement errors along the object's surface. A similar tracking control law will be synthesized for the re-capturing trajectory. Finally, the work will investigate how to *robustify* the regrasping procedure against small variations in the grasped object parameters (for instance, its shape and mass properties), in order to obtain a practically viable system.

3 RESEARCH PROBLEM

3.1 Given System

Let $\mathcal{Q} \subseteq \mathbb{R}^n$ and $\mathcal{U} \subseteq \mathbb{R}^n$ be the configuration space and the set of all possible torque inputs, respectively of a fully-actuated n-R robotic arm. The robotic arm is equipped with a simple non-dexterous jaw-gripper end-effector. Link i of the robot is with length L_i. We locate the world reference frame O at the base of the manipulator. The configuration space of the manipulator $Q \subseteq \mathbb{R}^n$ is defined to be the set of all configurations of the joint angles. The dynamic motion equations of the manipulator in the configuration space can be written as

$$M(\phi)\ddot{\phi} + C(\phi,\dot{\phi})\dot{\phi} + G(\phi) + \Gamma = \mathbf{u} \qquad (1)$$

where $\phi(t) = (\varphi_1(t) \cdots \varphi_n(t))^T \in Q$ is the configuration of the arm at time t, M is the $n \times n$ inertia matrix, C the $n \times n$ matrix of centrifugal and coriolis acceleration terms, G is the $n \times 1$ vector of gravitational effects, Γ is the $n \times 1$ vector of joints frictional torques and $\mathbf{u}(t) \in \mathcal{U}$ is a vector of control input torques to the joints.

Let $\mathcal{T} \subseteq \mathbb{R}^{m_1} \times \mathbb{S}^{m_2}$ be the task space of the manipulators end-effector where $m_1 = 2, m_2 = 1$ in the planar case ($d = 2$) and $m_1 = 3, m_2 = 3$ in the spatial case ($d = 3$). We denote $\mathbf{p_{ee}}(t) \in \mathcal{T}$ to be the task of the end-effector at time t, that is, its position and orientation with respect to reference frame O. Moreover, we define $\mathbf{x_{ee}}(t) = (\mathbf{p_{ee}}(t)^T \ \dot{\mathbf{p}}_{\mathbf{ee}}(t)^T)^T \in \mathcal{T} \times \mathbb{R}^{m_1+m_2}$ as the state of the end-effector in cartesian coordinates. We set a reference frame $\mathcal{X}_{ee}$ at the center of the end-effector.

Given object $\mathcal{B}$ with mass m. We denote the configuration of the object in the task space of the end-effector at time t by $\mathbf{p_{obj}}(t)$. And therefore, its state is given by $\mathbf{x_{obj}}(t) = (\mathbf{p_{obj}}(t)^T \ \dot{\mathbf{p}}_{\mathbf{obj}}(t)^T)^T \in \mathcal{T} \times \mathbb{R}^{m_1+m_2}$. Further, we define the relative state between the end-effector and object.

Definition 1. *The relative state between the end-effector and object is defined to be*

$$\overline{\mathbf{x}}(t) = \begin{pmatrix} \{\mathbf{p_{obj}}(t)\}_{ee} \\ \dot{\mathbf{p}}_{\mathbf{ee}}(t) - \dot{\mathbf{p}}_{\mathbf{obj}}(t) \end{pmatrix} \qquad (2)$$

where

$$\{\mathbf{p_{obj}}(t)\}_{ee} = R_{ee}^{obj}(\mathbf{p_{ee}}(t)) \cdot \mathbf{p_{obj}}(t) - \mathbf{d}_{\mathbf{ee}}^{\mathbf{obj}}(t) \qquad (3)$$

is the task of the object with respect to $\mathcal{X}_{ee}$, $R_{ee}^{obj} \in SO(d) : \mathcal{T} \to \mathcal{T}$ and $\mathbf{d}_{\mathbf{ee}}^{\mathbf{obj}}(t)$ are the rotation map and relative position, respectively, from reference frame $\mathcal{X}_{obj}$ to reference frame $\mathcal{X}_{ee}$.

The relative state expresses the relative position and velocity between the end-effector and object. At time $t = 0$, the end-effector and object are located at $\mathbf{p}_0$ and at rest, that is, $\mathbf{x_{ee}}(0) = \mathbf{x_{obj}}(0) = [\mathbf{p}_0^T\ \overline{0}]^T$. Therefore, their initial relative state is $\overline{\mathbf{x}}(0) = 0$. At that time, the object is grasped by the end-effector such that the objects reference frame $\mathcal{X}_{obj}$ overlaps the end-effectors reference frame $\mathcal{X}_{ee}$. That is, we define the initial orientation and translation between corresponding reference frames to be zero (Figure 1a) with no relative velocity as well.

3.2 Manipulation Goal

The regrasping problem is defined as follows. Given the initial relative state $\overline{\mathbf{x}}(0)$ and a desired relative state goal $\overline{\mathbf{x}}(t_g) = \zeta$, compute the control input sequence $\mathbf{u}(t) \in \mathcal{U}$, $t \in [0, t_g]$ to form an optimal manipulation of the robotic arm such that the relative goal is reached at some finite time $t_g \in [0, \infty)$. An optimal manipulation is the one forming a feasible trajectory of the arm $\phi(t) \in Q$ such that the relative state goal is reached while minimizing a given cost function $H(\phi(t))$.

Generally speaking, the robotic arm will perform some kind of manipulation to grant the object an initial velocity at release time t_r and catch it again at some time t_g with relative state ζ. The primary goal of this manipulation is to perform a set of motions such that at the final grasp the object will have a relative state $\zeta = (\zeta_{\mathbf{p}}^T\ \zeta_{\dot{\mathbf{p}}}^T)^T$ where $\zeta_{\mathbf{p}} \in \mathcal{T}$ and $\zeta_p \in \mathbb{R}^{m_1+m_2}$. We can divide this goal to two fundamental constraints which must be achieved. The first constraint is demanded by the essence of the regrasping problem, i.e., to achieve a new desired grasp configuration. This constraint states that the end-effector must finally grasp the object with the same relative position and orientation as in $t = 0$ but with an additional desired offset $\zeta_{\mathbf{p}}$. A planar example for this is illustrated in Figure 1b. Here $\zeta_{\mathbf{p}} = (\zeta_x\ \zeta_y\ \zeta_\theta)^T$ where ζ_x and ζ_y are the translation components between coordinate frames $\mathcal{X}_{ee}, \mathcal{X}_{obj}$ and ζ_θ is the relative orientation angle between the two. It should be noted that ζ_x cannot be unequal to zero in a symmetric end-effector but it is parameterized to preserve the generality of the formulation. The second constraint is required for proper manipulation. In order to obtain a soft interception and catch of the object at time t_g, the end-effector and the object must have equal velocities, that is, $\dot{\mathbf{p}}_{\mathbf{ee}}(t_g) = \dot{\mathbf{p}}_{\mathbf{obj}}(t_g)$. In other words, we constrain ζ_p to be

$$\zeta_p = \overline{0}\ . \tag{4}$$

In general, the final goal of the regrasping manipulation would be achieved if at some time $t_g \in [0, \infty)$ the following condition is met:

$$\rho\left(\overline{\mathbf{x}}(t_g) - \zeta\right) \approx 0 \tag{5}$$

where $\rho(\cdot)$ is some metric criterion to be defined. The condition will be met if the metric distance is close enough to zero, that is, the distance is within a tolerance boundary to be defined. Once condition (5) is met (at time t_g), the end-effector will regrasp (jaws closure) the object at the desired relative state. However, the end-effector and the grasped object have relative velocity with respect to coordinate frame O. Therefore, after time t_g where the object should be firmly grasped by the end-effector, the manipulator will gradually decelerate to zero velocity at final time t_f.

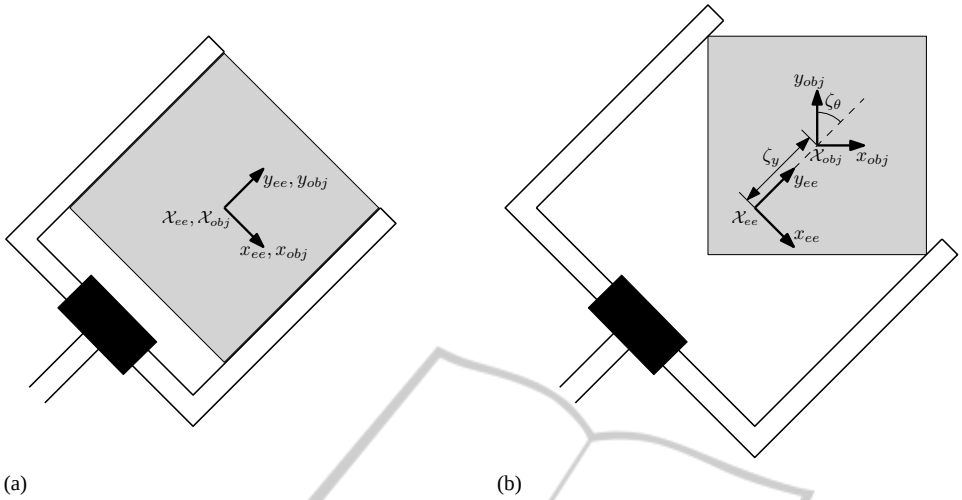
3.3 Constraints

The above desired manipulation should be done under the following constraints:

1. Based on actuators limitations, the allowed subset of control input torques is given by $\mathcal{U}_{al} \subset \mathcal{U}$.
2. The configuration of the arm is constrained to be in $Q_{free} \subseteq Q$ through the whole motion. Meaning, the arms joints are limited within a certain range due to mechanical and physical bounds.
3. Let $\mathcal{V} \subset \mathbb{R}^n$ denote the set of all possible velocities of the arms joints. The allowed joints velocities is defined by the abilities of the actuators and is given by $\mathcal{V}_{al} \subseteq \mathcal{V}$.
4. At time of release t_r, to enable the object to securely leave the end-effector, the orientation of the end-effector must be parallel to its velocity vector. That is,
$$\langle \mathbf{r_{ee}}, \dot{\mathbf{p}}_{\mathbf{ee}} \rangle = 0\ . \tag{6}$$
where $\mathbf{r_{ee}} \in \mathbb{R}^d$ is the vector collinear to the last link fixed with the end-effector.

4 STATE OF THE ART

This paper deals with the regrasping problem in a dynamic manipulation approach. Let us first describe the two regrasping approaches taken by industrial practitioners. The first approach is the *pick and place* method which designates a special work area near each robot arm, where the grasped workpiece can be dropped in a controlled manner, then picked up again at a new grasp configuration (Tournassoud et al., 1987; Lozano-Pérez et al., 1987; Xue et al., 2008). Alternatively, some high-end industrial practitioners resort to ŞunthreadingŤ the regrasping task by



(a) (b)

Figure 1: Initial and goal grasp configurations of the object by the end-effector: (a) both coordinate frames overlap in initial grasp and (b) in the final grasp the relative position and orientation between the coordinate frames are defined by $\boldsymbol{\zeta}_\mathbf{p}$.

placing several (hugely expensive) robot arms along a long production or assembly line, each picking a moving workpiece at a new grasp configuration (Kim and Park, 1995; Levitin et al., 2006). While both regrasping methods work nicely in practice, they consume valuable production time, occupy a substantial work area, and are highly expensive when multiple robot arms are being used.

In the robotics literature, there are three known approaches for regrasping (without picking and placing); First, the use of the end-effectors degrees of freedom to move between contact points while maintaining a force-closure grasp during the entire process (Corves et al., 2011; de Paula Caurin and Felicio, 2006; Roa and Suarez, 2009; Sudsang and Phoka, 2003; Vinayavekhin et al., 2011; Stuheli et al., 2013). This approach is also called *quasi-static finger gaiting* in the robotics literature. However, quasi-static finger gaiting is quite wasteful, as it requires sufficiently many degrees of freedom (requiring highly redundant finger linkages) to manipulate the grasped object between two grasp configurations while maintaining force closure grasps. Most of the finger gaiting algorithms use at least another extra finger that can be lifted at each step (Grosch et al., 2008; Hasegawa et al., 2003; Phoka and Sudsang, 2009; Rapela et al., 2002). Such motion results in a slow quasi-static manipulation, maintaining a state of constant contact with the object. This has analogy to gait, where proceeding is done only with one leg lifted at a time (when dynamic forces are not taken into consideration) so that the other legs can maintain balance. Such process can take valuable amount of time and therefore can be an impractical method. It has been recently suggested that a dual arm robot may be more suitable for object regrasping operations (Balaguer and Carpin, 2012; Harada et al., 2012). While dual arms is a promising approach, it has two significant drawbacks. First, regrasping an object with two cooperating arms require highly dexterous manipulation capabilities from both arms. Second, a dual-arm system is costly and occupies a fairly large work volume.

The second approach is sliding the fingertips on the objects surface without losing contact to reposition them at new contact points (Chen and Zribi, 2000; Cole et al., 1992). This approach has not been widely researched and has low feasibility as it jeopardizes the integrity of the object. Moreover, the approach needs also sufficiently enough degrees of freedom to develop trajectories on the objects surface.

The third approach is much faster and efficient, however more complex, as it uses dynamical manipulations to switch between grasp configurations. In dynamic manipulation, the robotŠs motion is quick and there is no constant contact state. The end-effector lose contact (fully or partly) with the object after releasing it in the air and regains contact by catching it at the final contact points. Such a method has advantages in fast operation. However, most work done in this field use a multi-fingered highly dexterous hand for performing regrasping. Furukawa et al. (Furukawa et al., 2006) proposed a regrasping strategy based on visual feedback of the manipulated object, this with a multi-fingered hand. The work of Tahara (Tahara et al., 2012) introduced a regrasping method using a 3-finger hand with no external sensing for feedback.

There is related work to dynamic regrasping in the field of dynamic manipulations (Senoo et al., 2008; Srinivasa et al., 2005). Dynamic manipulation of

an object enables the change of its position and orientation by tossing, pushing or hitting it. Lynch et al. (Lynch, 1997; Lynch et al., 1998) developed a low degree of freedom robot to perform complex manipulations using rolling, sliding and free-flight, this with no grasping (non-prehensile manipulation). In (Tabata and Aiyama, 2001) a one degree of freedom manipulator was used to toss an object to a goal position. The work by (Higashimori et al., 2009) was done to mimic the manipulation done with a pizza peel. The research states that the pizza peel is controlled with only two degrees of freedom and utilizes vision control. Much work has also been done in pushing manipulations where the object is pushed on a desired trajectory to re-position or re-orient it (Lynch, 1992; Bernheisel and Lynch, 2006; Kopicki et al., 2010). Moreover, some performed manipulation of the grasped objects by dynamic manipulations between fingertips (Garcia-Rodriguez and Diaz-Rodriguez, 2011; Yashima and Yamaguchi, 2002).

Another research area regarding the regrasping is rendezvous planning of two dynamic systems where only one is controlled. That is, matching the states of two dynamic systems within a finite time. Many solutions to this problem were presented; The proportional navigation algorithm is a commonly used method to account for a moving object such as a UAV or a missile (Erer and Merttopçuoglu, 2012; Mehrandezh et al., 2000). The advantage of the algorithm is in its simplicity and is shown to be complete as it will intercept the object in finite time. Many other rendezvous algorithms for optimal trajectories were presented to find the best trajectory in a dynamic environment (Michael et al., 2013; Rybus and Seweryn, 2013). Another method for dynamic planning is the Rapidly-exploring Random Tree (RRT) which is a probabilistic method for trajectory planning in a complex environment taking the dynamics of the system into account (Kothari and Postlethwaite, 2013; Kuwata et al., ; LaValle and Kuffner, 1999; Luders et al., 2010; Shkolnik et al., 2009). The authors have tried to use the RRT method for rendezvous planning and further use it for dynamic regrasping (Sintov and Shapiro, 2014b). However, this method has many difficulties working in real-time due to high complexity and therefore not feasible for our objective.

5 METHODOLOGY

The approach method for solving the above problem can be divided into three parts; First, we parameterize the motion of the arm in its two phases, release phase and free-flight phase. That is, the motion of the arm is described as a vector σ. The next part is the formulation of the problems constraints in terms of time t and the parameter vector σ. The new formulation of the constraints defines a time-varying subspace in σ-space. A feasible motion of the arm under the constraints is a solution vector σ^* which lies within the time-varying subspace for the whole motion. The problem of finding such a feasible solution is termed as the Time-Varying Constraint (TVC) problem. In the third part of the solution, a semi-stochastic algorithm was developed to solve the TVC problem and find a feasible and optimal solution. An optimal solution is the one which minimizes a pre-defined cost function $H(\sigma)$. The following subsections give a short overview of the solution parts.

5.1 Motion Parameterization

As mentioned, the regrasping motion is composed of two main phases: the release phase and the free-flight phase. In this section we parameterize this motion to the parameters which are the DOF of the trajectory. First we parameterize the motion in the free-flight phase. Then, based on the initial task and velocity (release state) at the beginning of the free-flight phase, we parameterize the release phase as a polynomial trajectory from the robots initial state to the release state. That is, we parameterize a polynomial trajectory which could provide the initial velocity at the release position demanded for the free-flight phase.

In this section we present the structure of σ_f and σ_r. Then, we present the formulation of the problems constraints in terms of time and σ.

5.1.1 Free-flight Phase

The free-flight motion of the object depends on its release position and release velocity. That is, granting the object an initial velocity $\dot{\mathbf{p}}_{\mathbf{obj}}(t_r)$ from initial position and orientation (task) $\mathbf{p}_{\mathbf{obj}}(t_r)$ defines its trajectory with no control ability. Once the object is released, its free-flight trajectory is a projectile motion solely under the influence of gravity. Therefore, the definition of motion in this phase includes the initial task and velocity at time t_r.

Moreover, the estimated time for the end-effector and the object to reach the desired relative state, e.g., satisfaction of condition (5), is defined by the gains of the Computed-Torque controller to be used. Therefore, the proportional gains of the controller defines the time of interception t_g with the object.

With the understanding of the parameters which define the free-flight phase, we define the parameters vector σ_f. The form of the parameters vector $\sigma_f \in$

$\mathbb{R}^{d_1}$ in the free-flight phase is defined as follows:

$$\sigma_f = \begin{pmatrix} \mathbf{p}^{\mathbf{r}}_{\mathbf{ee}} & \dot{\mathbf{p}}^{\mathbf{r}}_{\mathbf{ee}} & k_{p_1} & \cdots & k_{p_n} \end{pmatrix}^T . \quad (7)$$

where $\mathbf{p}^{\mathbf{r}}_{\mathbf{ee}} = \mathbf{p}_{\mathbf{obj}}(t_r)$ and $\dot{\mathbf{p}}^{\mathbf{r}}_{\mathbf{ee}} = \dot{\mathbf{p}}_{\mathbf{obj}}(t_r)$ are the task and velocity of the end-effector which would be granted to the object at time t_r, and $d_1 = 2m_1 + 2m_2 + n$. Vector σ_f is a parameterization of the free-flight phase and its determination defines the approximated trajectory and motion time of the arm.

5.1.2 Release Phase

The parameterization of the release phase presented here is the one introduced by (Sintov and Shapiro, 2014a). In this phase we must provide a trajectory from the robots initial state at time $t = 0$ to the desired release state at time t_r presented in the previous subsection. That is, given the robots initial task $\mathbf{p_{ee}}(0) = \mathbf{p^0_{ee}}$ at rest, we need a trajectory $\mathbf{s}(t)$ that will satisfy the following boundary conditions:

$$\begin{cases} \mathbf{s}(0) = \mathbf{p^0_{ee}} \\ s(0) = 0 \\ \mathbf{s}(t_r) = \mathbf{p^r_{ee}} \\ s(t_r) = \dot{\mathbf{p}}^{\mathbf{r}}_{\mathbf{ee}} \end{cases} \quad (8)$$

First we define an optional candidate trajectory which could complete the task.

Definition 2. *A trajectory function $\mathbf{s}(t) \in \mathcal{T}$ is a candidate trajectory if it is differentiable and satisfies some h boundary constraints.*

That is, in our motion planning problem, a trajectory $\mathbf{s}(t)$ is a candidate trajectory if it satisfies the $h = 4m$ boundary constraints in (8) which impose the initial and release states. The following definition describes a candidate trajectory function which is constrained by the problems boundary constraints and has redundant DOF to optimize.

Definition 3. *A candidate trajectory function $\mathbf{s}(t) = f_s(t, \mathbf{w}) \in \mathcal{T}$, where $\mathbf{w} = [w_1 \ \cdots \ w_{m \cdot k}]^T \in \mathbb{R}^{m \cdot k}$ and has h boundary constraints, is redundant if $m \cdot k > h$.*

The four constraints in (8) impose the values for $w_1, ..., w_4$ and leave $k-4$ free parameters for the function. Moreover, the goal time t_r can also be chosen as a free parameter if no time constraint is imposed. Therefore, the redundant parameters are denoted as $\sigma_r = [w_{h+1} \ \cdots \ w_{m \cdot k} \ t_r]^T \in \mathbb{R}^{d_2}$, where $d_2 = k - 4 + 1$. If the goal time t_r is fixed, it should not be included in the parameter vector σ_f.

A redundant trajectory function $\mathbf{s}(t) = f_s(t, \sigma_r)$ is chosen as the desired trajectory from the initial state to the goal state. The function imposes the boundary constraints while providing a desired number of free parameters for motion planning and optimization. Such redundant trajectories could take, for example, polynomial form or be a fourier series. In this work we implement the use of a redundant polynomial function due its simplicity.

5.1.3 Overall Parameterization

The release phase parameters vector σ_r and the free flight parameters vector σ_f can now be generalized to one vector σ which is a parameterization of the whole motion

$$\sigma = \begin{pmatrix} \sigma_r \\ \sigma_f \end{pmatrix} \in \Omega \quad (9)$$

where $\Omega \subseteq \mathbb{R}^{d_1 + d_2}$. The determination of σ will fully define the nominal trajectory of the motion from initial rest position of the arm ($t = 0$), through release of the object ($t = t_r$) to finally regrasping it at the desired relative state ($t = t_g$). The strength of this parameterization is that a single vector defines a specific regrasping motion independent of time.

5.2 Constraints Formulation

In this part we formulate the constraints of the systems motion in terms of the free parameters of the problem. We formulate configuration space constraints which define Q_{free}, other constraints in the task space, and finally velocity and torque constraints imposed by the limitations of the system. For example, due to joint limitations, it is possible that not all the configuration space Q of the arm is accessible. Therefore, we can formulate the free space Q_{free} explicitly as a set of z_1 constraints $\Phi \in \mathbb{R}^{z_1}$

$$\Phi_i(\phi(t)) = \Phi_i(t, \sigma) \leq 0, \ \forall i = 1, ..., z_1. \quad (10)$$

The same could be done to the velocity, torque and workspace constraints. The set of inequalities acquired could be written as

$$\Psi_i(t, \sigma) \leq 0, \ i = 1, ..., z \quad (11)$$

where $\Psi(t, \sigma)$ is a set of z functions which define the systems constraints. Inequality (11) defines the feasible region of the dynamic system in terms of time and the desired trajectory parameters σ. That is, we obtained a set of constraints which defines a time-varying region in Ω. The next section presents the time-varying constraint problem and the search algorithm to find an optimal trajectory satisfying the constraints.

5.3 Time-varying Constraint (TVC) Problem

In the previous section we obtained a set of inequalities depending on t and the parameters vector σ. Re-

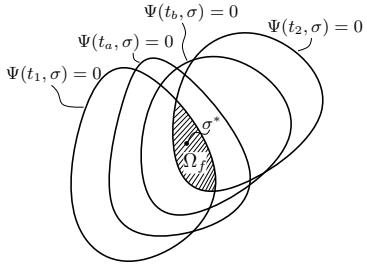


Figure 2: The TVC problem where $t_1 < t_a < t_b < t_2$.

call that the components in σ are independent of the time. Therefore, we would like to find an optimal vector $\sigma^* \in \Omega$ that satisfies the constraints through the regrasping motion and minimizes some cost function. Such an optimal vector will sufficiently define the motion of the system under the kinodynamic constraints. Let $\Sigma \subset \Omega$ be the allowed region for σ. We define the notion of a feasible set.

Definition 4. *A set $\Omega_f \subset \Omega$ is a feasible set in $t \in [t_1,t_2]$ if $\Omega_f \subseteq \Sigma$ and each $\sigma \in \Omega_f$ satisfies inequality (11) for all time $t \in [t_1,t_2]$.*

We now define a feasible vector.

Definition 5. *A vector of trajectory parameters $\sigma \in \mathbb{R}^d$ is said to be feasible in $t \in [t_1,t_2]$ if $\sigma \in \Omega_f$.*

The above two definitions conclude that a vector is feasible if

$$\sigma = \left\{\sigma \in \Omega_f | \sigma \in \Sigma, \Psi_i(t,\sigma) \leq 0,\ \forall t \in [t_1(\sigma), t_2(\sigma)]\right\}, \tag{12}$$

for all $i = 1,...,z$. Notice that the time interval is written in general $[t_1(\sigma), t_2(\sigma)]$ and is a function of σ. This is due to the definition of the free parameters vector σ, which could include parameters that define the operation time. Therefore, the choice of σ determines the boundary time. We now face the problem of finding the feasibility set $\Omega_f \subseteq \Sigma$ where for all vectors within it, inequality (11) is maintained at all times. Formally, the problem is as follows.

Problem 1. *Given the set of constraints in (11) and the set Σ, find the feasibility set $\Omega_f \subseteq \Sigma$.*

Solving the above problem provides the feasible set Ω_f from which the optimal solution is to be chosen. Hence, we define the following minimization problem.

Problem 2. *Find the vector $\sigma^* \in \Omega_f$ where $\Omega_f \subseteq \Sigma$ such that*

$$\begin{aligned} \sigma^* = \underset{\sigma}{arg\min} \quad & H(\sigma) \\ \textit{subject to} \quad & \sigma^* \in \Omega_f \end{aligned} \tag{13}$$

where $H(\sigma)$ is some cost function to minimize.

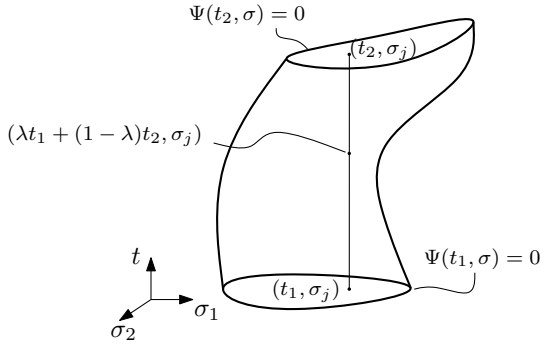


Figure 3: Vector σ satisfying Lemma 1.

In other words, the above general problem is finding an optimal vector σ^* that is feasible and minimizes some cost function $H(\sigma)$ to be determined. Figure 2 illustrates the two problems. The position and volume of $\Psi(t,\sigma)$ in Ω varies in time and therefore, the solution of the problem is in a domain formed by projecting the constraints for time t_1 to t_2 on space Ω. The intersection formed by these projections, if one exists, is the feasibility domain that the optimal solution σ^* should be chosen from.

A search algorithm is now desired to find the set Ω_f of feasible vectors. The domain formed by the set of constraints in inequality (11) is non-linear, non-convex, and not continuous. Therefore, an analytical solution of the reachable set is only possible in rare and simple instances. We present a numerical search algorithm to find a set of vectors satisfying the above constraints. Further, we can choose one vector from the set that best minimizes the cost function. We begin by presenting a simple definition for normalizing the time interval.

Lemma 1. *A vector σ is feasible in $t \in [t_1,t_2]$ if the constraint $\Psi_i(\lambda t_1 + (1-\lambda)t_2, \sigma) \leq 0$ is satisfied for all $0 \leq \lambda \leq 1$ and for all $i = 1,...,z$.*

Proof. For each time instant $t \in [t_1,t_2]$ there exists λ such that $t = \lambda t_1 + (1-\lambda)t_2$ and $0 \leq \lambda \leq 1$. Therefore, if a vector σ satisfies $\Psi_i(t,\sigma) \leq 0$ for all $i = 1,...,z$ and $t_1 \leq t \leq t_2$, it must also satisfy $\Psi_i(\lambda t_1 + (1-\lambda)t_2, \sigma) \leq 0$ for all $i = 1,...,z$ and $0 \leq \lambda \leq 1$. □

Lemma 1 is utilized as a criterion for determining whether a vector σ is feasible. Numerically, for σ we check the constraint for time $t = \lambda t_1 + (1-\lambda)t_2$ with $\lambda = \{0, \Delta\lambda_1, \Delta\lambda_1 + \Delta\lambda_2, ..., 1\}$. The value of the step $\Delta\lambda_j$ will be further defined. Figure 3 illustrates an abstraction of the feasibility problem and the line in time defined by Lemma 1. Without loss of generality, from this point we will address the problem with the time frame $[t_1,t_2] = [0,t_g]$.

The basis of the algorithm's operation is selecting a set of N random points within Σ and checking each

Algorithm 1: *Feasibility_search*$(\Sigma, \Psi, P_{max}, \varepsilon_b)$.

Input: The allowed set Σ, set of constraints Ψ, the probability P_{max}, and tolerance ε_b.
Output: Set of feasible points Ω_f.

```
Calculate number of random points N such that the probability to find a solution is more than 1 − P_max.
Generate the set P = {σ_1, ..., σ_N} of N uniformly distributed random points within Σ.
for i = 1 → N do
    if ¬(Adaptive_Check(σ_i, Ψ, ε_b)) then
        Remove σ_i from P.
    end if
end for
return Ω_f = P = {σ_1, ..., σ_M}    // M ≤ N
```

for its feasibility. The feasibility search algorithm is presented as the *Feasibility_Search*(Σ, Ψ) function in Algorithm 1. The algorithm's input is the allowed set Σ in Ω chosen by the user and the set of constraints of equation (11). The first step of the algorithm is to determine the number of random points N such that the probability to find a solution is more than a user defined probability $1 - P_{max}$. The calculation of N based on the choice of P_{max} will be presented later in the algorithm's analysis. The next step is to sample N random points $\mathcal{P}$ uniformly distributed in Σ. The allowed region formed by Σ is a hyper-rectangle in Ω and therefore we sample points in each axis of Ω within the boundaries defined by Σ. Such sampling provides a Poisson distribution over the volume of Σ. The next step is going over all the N points in $\mathcal{P}$ and filtering out those that are not feasible. The final time t_{g_i} is determined for each point σ_i checked. We check if the constraints are satisfied for time $t = \lambda t_{g_i}$ where $\lambda = \{0, \Delta\lambda_1, \Delta\lambda_1 + \Delta\lambda_2, ..., 1\}$. Those that do not satisfy the constraints are eliminated and the filtered set $\mathcal{P}$ with size $M \leq N$ is outputted.

Scanning the constraint $\Psi(\lambda t_{g_i}, \sigma_i)$ for $\lambda = \{0, \Delta\lambda, 2\Delta\lambda, ..., 1\}$ where $\Delta\lambda$ is a constant value is rather risky. The value of Ψ might ascend over 0 and descend below again within the discretized step size. An example of such is shown in Figure 4 where with step size above 0.03 failure of the constraints might not be discovered. Moreover, too small step sizes could be unnecessary and the price would include very high complexity. Therefore, we present a simple adaptive step size algorithm to fine tune the time steps and diagnose or rule out such scenarios.

Definition 6. *The constraint value of a feasible point σ_i at time λt_{g_i} is defined to be*

$$\tilde{\Psi}_{i,\lambda} = \max_j \left\{ \Psi_j(\lambda t_{g_i}, \sigma_i) \right\} , \qquad (14)$$

where Ψ_j is the j^{th} component of the constraint vector Ψ.

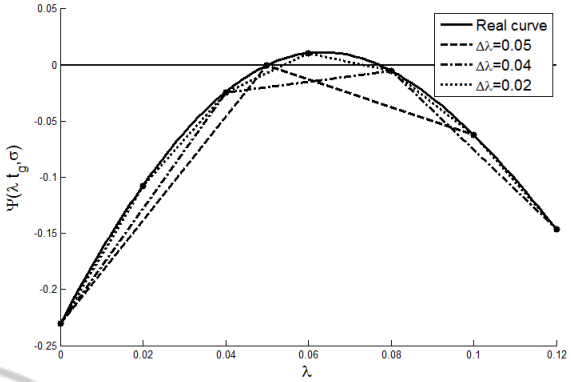


Figure 4: The maximum constraint value along time with change of the step size $\Delta\lambda$.

That is, the constraint value is the shortest distance from point σ_i to the boundary of the closest constraint. Notice that we refer to a distance with a positive value, but the value of $\tilde{\Psi}_{i,\lambda}$ is maintained negative for an indication that σ_i is a feasible point. Assume that the change rate of the constraint value with regard to λ is bounded by

$$\frac{\Delta\tilde{\Psi}_{i,\lambda}}{\Delta\lambda} \leq S_{max}, \ \forall\, 0 \leq \lambda \leq 1 \, . \qquad (15)$$

That is, the maximum slope of the constraint value $\tilde{\Psi}_{i,\lambda}$ is S_{max}. Under this assumption we can say that if at time λt_{g_i} the constraints are satisfied, $\tilde{\Psi}_{i,\lambda} < 0$, then the minimum time for the constraint to reach 0 is $(\lambda + \Delta\lambda_{min}) t_{g_i}$ where $\Delta\lambda_{min} = -\frac{\tilde{\Psi}_{i,\lambda}}{S_{max}}$. Therefore, as we get closer to a boundary of a constraint, we decrease $\Delta\lambda$ such that reaching above the zero line in that time frame is not possible. Figure 5 illustrates the selection of $\Delta\lambda$ as it gets smaller when approaching the zero line and larger when receding. However, in this adaptive approach, even though $\tilde{\Psi}_{i,\lambda}$ passes the zero line, the algorithm will never do so as it will continue to decrease $\Delta\lambda$. Therefore, we bound such that the algorithm will stop checking the current σ_i (and remove it) if $\tilde{\Psi}_{i,\lambda} < \varepsilon_b < 0$, where ε_b is a value that will be defined further in the algorithm's analysis. This also serves as a safety distance, assuring the solution is far enough from the constraints boundary. To calculate S_{max} we differentiate the constraint vector by λ to acquire its slope $\frac{\partial\Psi(\lambda t_g, \sigma)}{\partial\lambda}$, where t_g is the maximum possible goal time based on the allowed time interval given in Σ. S_{max} is the maximum slope of all components over all time and can be computed by the following maximization problem

$$\begin{aligned} S_{max} = \max_{\lambda,\sigma,j} \quad & S_j(\lambda t_g, \sigma) \\ \text{subject to} \quad & 0 \leq \lambda \leq 1 \\ & \sigma \in \Sigma \end{aligned} \qquad (16)$$

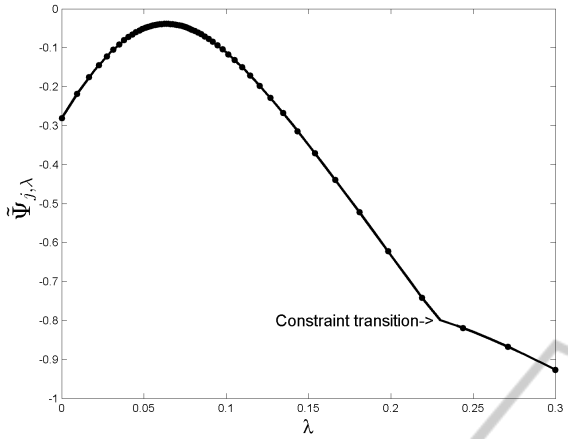


Figure 5: Adaptive step size algorithm.

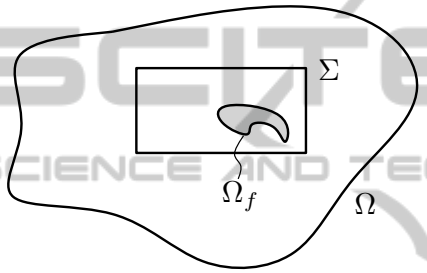


Figure 6: Two-dimensional example of Σ and Ω_f in Ω.

where S_j is the j^{th} component of the constraints derivative $S(\lambda t_g, \sigma) = \frac{\partial \Psi(\lambda t_g, \sigma)}{\partial \lambda}$. This could be computed analytically using Kuhn-Tucker conditions or numerically. The adaptive step size function *Adaptive_Check*(σ_i, Ψ) is presented in Algorithm 2.

Algorithm 2: *Adaptive_Check*$(\sigma_i, \Psi, \varepsilon_b)$.

Input: σ_i, the set of constraints Ψ and the tolerance ε_b.
Output: Boolean: 1 if σ_i is feasible, 0 if not feasible.
1: Set $\lambda = 0$.
2: Extract t_{g_i} from the last component of σ_i.
3: Calculate S_{max}. // using optimization prob. in (16).
4: **while** ($\lambda \leq 1$) **do**
5: Calculate $\tilde{\Psi}_{i,\lambda} = \max_j \{\Psi_j(\lambda t_{g_i}, \sigma_i)\}$.
6: **if** $\neg$($\tilde{\Psi}_{i,\lambda} < \varepsilon_b$) **then**
7: Return 0.
8: **else**
9: Calculate $\Delta\lambda = -\frac{\tilde{\Psi}_{i,\lambda}}{S_{max}}$.
10: $\lambda = \lambda + \Delta\lambda$.
11: **end if**
12: **end while**
13: Return 1.

The final step of the algorithm is selecting the optimal solution among the set of feasible points $\Omega_f = \{\sigma_1, ..., \sigma_M\}$ found in Algorithm 1 and perform local fine-tuning optimization. Given the cost function $H(\sigma)$, the optimal solution σ^* is found according to Algorithm 3. In this algorithm, first a simple naive search is performed on the feasibility set Ω_f to find a feasible point σ_k that best minimizes $H(\sigma)$ (Line 1). Next, we utilize a Gradient Descent (GD) method (**?**) to refine the solution and find a local minimum in the neighborhood of σ_k. The GD method is an iterative algorithm with an update law of the form

$$\sigma^{(i+1)} = \sigma^{(i)} - \gamma \nabla H(\sigma^{(i)}) \tag{17}$$

for some small $\gamma > 0$ (Line 5). In each iteration a check is done to avoid breaking the constraints. Notice that the *Adaptive_Check* function prevents the solution from approaching the constraint boundary with distance less than ε_b. The iterations are terminated if the new point breaks the constraints or if the convergence condition is satisfied (Line 10 of Algorithm 3). Figure 7 presents two examples of local refinement of the optimal solution; one was stopped by the constraint while the other managed to reach the local minimum. The convergence condition checks if the norm of the current descent is smaller than a predefined tolerance ε_d, that is, we have reached a local minimum with ε_d accuracy. The last point of the iteration is the optimal solution and is returned by the algorithm.

Algorithm 3: *Local_Optimization*$(\Omega_f, \varepsilon_d)$.

Input: Feasible set $\Omega_f = \{\sigma_1, ..., \sigma_M\}$ and tolerance ε_d.
Output: Optimal solution σ^*.
1: $k = \arg\min_i H(\sigma_i),\ i = 1, ..., M$
2: Define $\sigma^{(0)} = \sigma_k$.
3: $i = 0$.
4: **repeat**
5: $\sigma^{(i+1)} = \sigma^{(i)} - \gamma \nabla H(\sigma^{(i)})$.
6: **if** $\neg$(*Adaptive_Check*$(\sigma^{(i+1)})$) **then**
7: Return $\sigma^{(i)}$.
8: **end if**
9: $i = i + 1$.
10: **until** $\left\|\nabla H(\sigma^{(i)})\right\|_2 < \varepsilon_d$
11: $\sigma^* = \sigma^{(i)}$.
12: Return σ^*.

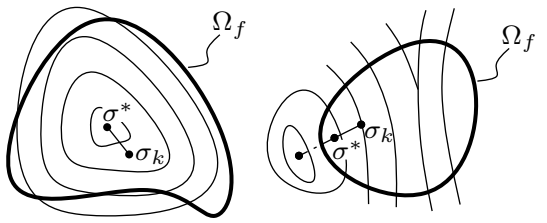


Figure 7: Two examples of the optimal solution refinement; one (left) descends to the local minimum and one (right) is stopped by the constraint boundary.

Statistical analysis which was performed have proven that it is possible to determine the minimal probability to find a solution if such exists. a solution if one exists.

6 EXPECTED OUTCOME

This work addresses the problem of reducing the number of end-effectors needed in the production for grasping a number of parts and performing several tasks on them. Dynamic regrasping approach will be addressed for transition between required grasps to perform designated tasks. The idea for the regrasping operation is to perform it with a simple non-dexterous end-effector and with the dynamics of the robotic arm.

As presented, currently the algorithm has the capabilities for computation of an in-hand spinning motion for regrasping. The next phase of the research is extension of the planning algorithm to compute the motion of the mid-air flipping. These two methods should provide most of the dynamic regrasping needs. The final outcome will be an overall algorithm for computation of a regrasping motion given an object's geometry and the goal relative state. That is, the algorithm will choose the best regrasping strategy according to the objects geometry and desired final grasp.

Looking in a wider application for the regrasping motion. The field of robotic grasping is very wide and serve many civil, industrial and military areas. The regrasping operation is performed whenever the grasp configuration is not compatible to the future task to be done. Many industrial and civilian applications demand change of the grasp configuration to achieve desired tasks. In many applications we demand fast and efficient performance of the tasks. Moreover, we desire simple low cost arms to conduct the tasks. For example, in the industrial world of today, fast and efficient production is highly important. However, achieving so must be accompanied with low costs to ensure a reasonable final product price. Our proposed research will impact the number of robotic arms and end-effectors used in production lines. Currently, each end-effector is specially designed for grasping a specific object and performing a specific task. They are used for assembly or material handling. The more end-effector needed, the more robotic arms, tool changers, and services are needed. Moreover, they demand more operational space in the plant. Using the same robotic arm and end-effector for multiple tasks will drastically reduce the number of robotic arms needed and reduce the demand for technical and engineering services. An additional advantage would be in the production time. The dynamic regrasping algorithm would have great benefit in shortening the material handling time of transferring objects between end-effectors because the object remains in the same end-effector.

REFERENCES

Balaguer, B. and Carpin, S. (2012). Bimanual regrasping from unimanual machine learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3264 –3270.

Bernheisel, J. and Lynch, K. (2006). Stable transport of assemblies by pushing. *IEEE Transactions on Robotics*, 22(4):740 –750.

Chen, J. and Zribi, M. (2000). Control of multifingered robot hands with rolling and sliding contacts. *The International Journal of Advanced Manufacturing Technology*, 16:71–77.

Cole, A., Hsu, P., and Sastry, S. (1992). Dynamic control of sliding by robot hands for regrasping. *IEEE Transactions on Robotics and Automation*, 8(1):42 –52.

Corves, B., Mannheim, T., and Riedel, M. (2011). Regrasping: Improving capability for multi-arm-robot-system by dynamic reconfiguration. In Jeschke, S., Liu, H., and Schilberg, D., editors, *Intelligent Robotics and Applications*, volume 7101 of *Lecture Notes in Computer Science*, pages 132–141. Springer Berlin Heidelberg.

de Paula Caurin, G. A. and Felicio, L. C. (2006). Learning based regrasping applied to an antropomorphic robot hand. ABCM Symposium series in mechatronics.

Erer, K. S. and Merttopçuoglu, O. (2012). Indirect Impact-Angle-Control Against Stationary Targets Using Biased Pure Proportional Navigation. *Journal of Guidance Control Dynamics*, 35:700–704.

Furukawa, N., Namiki, A., Taku, S., and Ishikawa, M. (2006). Dynamic regrasping using a high-speed multifingered hand and a high-speed vision system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 181 –187.

Garcia-Rodriguez, R. and Diaz-Rodriguez, G. (2011). Grasping and dynamic manipulation by soft fingertips without object information. In *Proceedings of the 9th IEEE International Conference on Control and Automation (ICCA)*, pages 766 –771.

Grosch, P., Suarez, R., Carloni, R., and Melchiorri, C. (2008). Planning setpoints for contact force transitions in regrasp tasks of 3d objects. In *Proceedings of the 17th IFAC World Congress*, volume 17, pages 6776–6781, Seoul, Korea. IFAC International Federation of Automatic Control.

Harada, K., Foissotte, T., Tsuji, T., Nagata, K., Yamanobe, N., Nakamura, A., and Kawai, Y. (2012). Pick and place planning for dual-arm manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2281 –2286.

Hasegawa, Y., Higashiura, M., and Fukuda, T. (2003). Simplified generation algorithm of regrasping motion -

performance comparison online-searching approach with ep-based approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1811 – 1816 vol.2.

Higashimori, M., Utsumi, K., Omoto, Y., and Kaneko, M. (2009). Dynamic manipulation inspired by the handling of a pizza peel. *IEEE Transactions on Robotics*, 25(4):829 –838.

Kim, H. and Park, S. (1995). A strong cutting plane algorithm for the robotic assembly line balancing problem. *International Journal of Production Research*, 33(8):2311–2323.

Kopicki, M., Stolkin, R., Zurek, S., Morwald, T., and Wyatt, J. (2010). Predicting workpiece motions under pushing manipulations using the principle of minimum energy. In *Proceedings of the RSS workshop on Representations for Object Grasping and Manipulation in Single and Dual Arm Tasks*. submitted.

Kothari, M. and Postlethwaite, I. (2013). A probabilistically robust path planning algorithm for uavs using rapidly-exploring random trees. *Journal of Intelligent & Robotic Systems*, 71(2):231–253.

Kuwata, Y., Teo, J., Member, S., Fiore, G., Member, S., Karaman, S., Member, S., Frazzoli, E., Member, S., How, J. P., and Member, S. Realtime motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems*, page 2009.

LaValle, S. and Kuffner, J.J., J. (1999). Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 473–479.

Levitin, G., Rubinovitz, J., and Shnits, B. (2006). A genetic algorithm for robotic assembly line balancing. *European Journal of Operational Research*, 168(3):811 – 825.

Lozano-Pérez, T., Jones, J. L., Mazer, E., O'Donnell, P. A., Grimson, W. E. L., Tournassoud, P., and Lanusse, A. (1987). Handey: A robot system that recognizes, plans, and manipulates. In *IEEE International Conference on Robotics and Automation*, pages 843=–849.

Luders, B., Karaman, S., Frazzoli, E., and How, J. (2010). Bounds on tracking error using closed-loop rapidly-exploring random trees. In *American Control Conference (ACC)*, pages 5406–5412.

Lynch, K., Shiroma, N., Arai, H., and Tanie, K. (1998). The roles of shape and motion in dynamic manipulation: the butterfly example. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 1958 –1963 vol.3.

Lynch, K. M. (1992). The mechanics of fine manipulation by pushing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2269–2276.

Lynch, K. M. (1997). Dynamic manipulation with a one joint robot. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 356–366.

Mehrandezh, M., Sela, N., Fenton, R., and Benhabib, B. (2000). Robotic interception of moving objects using an augmented ideal proportional navigation guidance technique. *IEEE Transactions on Systems, Man and Cybernetics*, 30(3):238–250.

Michael, J., Chudej, K., Gerdts, M., and Pannek, J. (2013). Optimal rendezvous path planning to an uncontrolled tumbling target.

Phoka, T. and Sudsang, A. (2009). Contact point clustering approach for 5-fingered regrasp planning. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4174 –4179.

Rapela, D., Rembold, U., and Kuchen, B. (2002). Planning of regrasping operations for a dextrous hand in assembly tasks. *Journal of Intelligent and Robotic Systems*, 33:231–266.

Roa, M. A. and Suarez, R. (2009). Regrasp planning in the grasp space using independent regions. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, pages 1823–1829, Piscataway, NJ, USA. IEEE Press.

Rybus, T. and Seweryn, K. (2013). Trajectory planning and simulations of the manipulator mounted on a free-floating satellite. In SÄĚsiadek, J., editor, *Aerospace Robotics*, pages 61–73. Springer.

Senoo, T., Namiki, A., and Ishikawa, M. (2008). High-speed throwing motion based on kinetic chain approach. In *IROS*, pages 3206–3211.

Shkolnik, E., Walter, M., and Tedrake, R. (2009). Reachability-guided sampling for planning under differential constraints. In *in In Proceedings of the IEEE/RAS International Conference on Robotics and Automation (ICRA)*.

Sintov, A. and Shapiro, A. (2014b). Time-based RRT algorithm for rendezvous planning of two dynamic systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Sintov, A. and Shapiro, A. (Submitted July 2014a). Skip: Semi-stochastic kinodynamic planning algorithm. *The IEEE Transactions on Robotics*.

Srinivasa, S., Erdmann, M., and Mason, M. (2005). Using projected dynamics to plan dynamic contact manipulation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3618 – 3623.

Stuheli, M., Caurin, G., Pedro, L., and Siegwart, R. (2013). Squeezed screw trajectories for smooth regrasping movements of robot fingers. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 35(2):83–92.

Sudsang, A. and Phoka, T. (2003). Regrasp planning for a 4-fingered hand manipulating a polygon. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 2671 – 2676 vol.2.

Tabata, T. and Aiyama, Y. (2001). Tossing manipulation by 1 degree-of-freedom manipulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 132 –137.

Tahara, K., Maruta, K., Kawamura, A., and Yamamoto, M. (2012). Externally sensorless dynamic regrasping and manipulation by a triple-fingered robotic hand with torsional fingertip joints. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3252 –3257.

Tournassoud, P., Lozano-Perez, T., and Mazer, E. (1987). Regrasping. In *Proceedings of the IEEE International*

Conference on Robotics and Automation, volume 4, pages 1924–1928.

Vinayavekhin, P., Kudohf, S., and Ikeuchi, K. (2011). Towards an automatic robot regrasping movement based on human demonstration using tangle topology. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3332 –3339.

Xue, Z., Zollner, J. M., and Dillmann, R. (2008). Planning regrasp operations for a multifingered robotic hand. In *Proceedings of the IEEE Conference on Automation, Science and Engineering*, pages 778–783. IEEE.

Yashima, M. and Yamaguchi, H. (2002). Dynamic motion planning whole arm grasp systems based on switching contact modes. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2492 –2499.