2012 IEEE International Conference on Robotics and Automation
RiverCentre, Saint Paul, Minnesota, USA
May 14-18, 2012

Bimanual Regrasping from Unimanual Machine Learning

Benjamin Balaguer and Stefano Carpin

Abstract— **While unimanual regrasping has been studied extensively, either by regrasping in-hand or by placing the object on a surface, bimanual regrasping has seen little attention. The recent popularity of simple end-effectors and dual-manipulator platforms makes bimanual regrasping an important behavior for service robots to possess. We solve the challenge of bimanual regrasping by casting it as an optimization problem, where the objective is to minimize execution time. The optimization problem is supplemented by image processing and a unimanual grasping algorithm based on machine learning that jointly identify two good grasping points on the object and the proper orientations for each end-effector. The optimization algorithm exploits this data by finding the proper regrasp location and orientation to minimize execution time. Influenced by human bimanual manipulation, the algorithm only requires a single stereo image as input. The efficacy of the method we propose is demonstrated on a dual manipulator torso equipped with Barrett WAM arms and Barrett Hands.**

I. INTRODUCTION

Recent progresses in service robotics have allowed humanoid robots to perform various household tasks requiring two manipulators working in cooperation (e.g., handling a wok [20]). Historically, the simpler pick-and-place actions have been mostly solved using unimanual grasping, where a single manipulator is used to perform the task. Scenarios requiring an object to be placed in a location out of the robot's reach have typically been managed by exploiting mobile manipulation, where the robot's mobile platform (e.g., legs, wheels) is used to move the robot into an adequate position. Humans are, however, very adept at using both of their arms to efficiently interact with objects. The goals of these human bimanual object interactions vary from changing an object's configuration to repositioning it to a more efficiently accessed location. We note that humans exploit these bimanual repositioning actions for efficiency (e.g., it would be inefficient to transfer a stack of plates one by one by walking when the final location is within reach of an arm). Evidently, possessing this bimanual regrasping skill is essential for humanoids to successfully enter the realm of home robotics, not only as an efficient pick-and-place solution but also as a way to introduce more human-like robotic coworkers.

Compared to other topics, regrasping has seen little attention from the robotics community and can generally be divided into three approaches. The first, in-hand regrasping, has experienced more interest than the others and consists of relying on one end-effector to regrasp the object. In-hand regrasping does not solve the problem of getting the object into a location reachable by only one arm, but rather addresses the problem of changing the object's configuration (e.g., rotating a pen in your hand). This approach is not only dependent on the robot's end-effector, but also requires a dexterous hand with many degrees of freedom and, as such, cannot be used with simple or under-actuated end-effectors. The second, which we call on-surface regrasping, consists of running a unimanual grasping algorithm two or more times, depositing the object on a surface (e.g., table) between each grasp. Even though on-surface regrasping works in practice, it requires an inefficient number of manipulator movements and multiple computationally-intensive calls to a unimanual grasper, resulting in a long and ineffective process that defeats the efficient purpose behind human regrasping. The third method, which we coin in-air regrasping, is the one used most frequently by humans where the object is regrasped in the air using both arms. In-air regrasping is exploited when an object in one of the manipulator's reachability subspace needs to be moved to a location in the other manipulator's reachability subspace (e.g., putting a cup from a table to a cupboard). Even though in-air regrasping has seen little attention, it is a crucially beneficial behavior for service robots since it not only saves time performing certain pick-and-place tasks but also mimics human behavior. Additionally, robotic in-air regrasping does not require complex end-effectors, nor does it impose restrictions on the manipulators being used.

We specifically solve the problem of in-air regrasping with an emphasis on minimizing execution time and propose an algorithm that exploits image processing, machine learning, and optimization. The manuscript's contributions are:

- the enhancement of a state-of-the-art image processing algorithm by reducing its computation time by 96%;
- the introduction of a novel optimization framework, validated against other algorithms;
- an end-to-end algorithm, with modular components, that is manipulator- and end-effector-independent;
- the reduction of the algorithm's computation time, resulting in real-time performance;
- the evidence that the algorithm works well in practice by running it on a real robotic platform.

The rest of the paper is organized as follows. Section II highlights previous work on regrasping, both from a robotics and human perspective. We continue by giving a high-level overview of the full algorithm in Section III. The details of the algorithm are given in Section IV, followed, in Section V, by off- and on-line experiments. We conclude the paper with final remarks and future work in Section VI.

School of Engineering, University of California, Merced, CA, USA, {bbalaguer,scarpin}@ucmerced.edu.

II. Related Work

One of the earliest works attempting to solve on-surface regrasping is presented by Yournassoud et al. [19]. The regrasping operation is divided into two components. The Grasp Space defines the space where the object is being carried by the manipulator. The Placement Space defines objects' locations on the table. The regrasping problem is then cast as the problem of finding the right transitions between Grasp and Placement Spaces given a start and goal object configuration. Although the algorithm lays the theoretical foundation for most regrasping algorithms, it does not generalize well (only three-dimensional polyhedron objects are considered) and does not provide experimental results. Building upon Yournassoud et al. early work, Koga et al. address the motion planning problem of in-air regrasping in [12]. Both works are related in that they utilize two similar regrasping phases (called transfer and transit phases in [12]), but Koga et al. exploit a second manipulator and remove the on-surface grasping condition. The algorithm operates in the object's configuration space to find a path from the initial to final object configuration. For each configuration in the path, all possible ways of grasping the object can be determined, and pruned according to a set of metrics and constraints. Unfortunately, the problem is solved from a computer animation perspective, which provides simplifications that would be invalid in a real-world robotics scenario (e.g., the entire environment's geometry, the initial object's configuration, and a set of valid grasps are known apriori). Our work is, in some ways, a reversal of this process since we utilize the manipulator's configuration space (as opposed to the object's configuration space), from which we can determine the object configuration when grasped by the manipulator.

Kawamura et al. approach the problem of regrasping using dual manipulators without relying on real-time external sensing [11]. Their solution is not general, since they solve it specifically for two 2-link planar arms and assume rectangular objects. A previously-published grasp algorithm [1] is rendered computationally efficient by empirically determining, using numerical simulations, a quasi-linear relationship between the object's orientation and the manipulator's initial contact positions. This relationship is then exploited to calculate regrasping phases that will change the object's orientation. The work is entirely theoretical in nature, and many assumptions make it inflexible for a real robotic scenario. Finding the linear relationship has merit, however, and the process can be thought of as machine learning, which we also exploit in our paper to determine grasps. In a more practical paper, Berenson et al. investigate the problem of finding valid grasps in cluttered environments, supplemented by in-air regrasping scenarios [4]. They rely on a database of pre-computed grasps for a set of known objects, along with motion capture, to calculate the best grasp's quality based on forces, friction, and contact points. Evidently this approach does not generalize since the robot will only be able to grasp objects that are part of the database and requires an expensive motion capture system - problems we circumvent by using image processing and feature-based grasping. Differently from the typical robot regrasping papers, Edsinger et al. consider human-robot regrasping [7]. This complex interaction is solved using a set of pre-defined behaviors (e.g., detect a person, hand object to a person, etc...) that are exploited when necessary. The authors extend this behavior database in a subsequent paper [8], allowing for bimanual manipulation of two objects. A potential problem with the generation of robotic behaviors is that they are constrained to the tasks and robots for which they are originally designed and might not generalize to different tasks, robots, or end-effectors. Additionally, the behaviors assume that the human-in-the-loop will understand cues given by the robot. These cues effectively place the burden of learning the environment and understanding the tasks on a human rather than the robot. The authors' works are interesting, however, since using human intuitions is a valid strategy in scenarios involving human-robot interactions. We also believe in taking advantage of humans to advance robotics research and specifically use them to acquire good training examples for our machine learning component.

With the inherent connection between service robots and humans, a series of human studies have influenced our algorithm. Churchill et al. investigate the kinematic behaviors of unimanual and bimanual human grasps [6]. No significant kinematic difference was found between human unimanual and bimanual manipulation. Specifically, one- and two-effector tasks pose the same constraints, share the same control structures, and are achieved in similar fashion by human test subjects. An earlier work by Castiello [5], with fewer experiments and data, also pointed out this phenomenon. These findings have influenced our decision to build our bimanual regrasper from a unimanual grasping algorithm. Simoneau et al. and Spencer et al. study the importance of vision in human bimanual tasks in [17] and [18], respectively. Both works show that there is little effect from complete vision loss during bimanual tasks. These findings suggest that humans possess spatial awareness independent of sensory feedback. We exploit these findings by only relying on vision to start the regrasping procedure (i.e., no sensory feedback is used during the physical regrasping phase), allowing for significant savings in computation time. In [9], Gribova studies bimanual coordination from a neural perspective, finding that bimanual movements are internally handled by the brain as a single process. In other words, bimanual movements are not a serialization of unimanual ones. Robotic on-surface regrasping violates this finding.

III. Algorithm Overview

Our problem setting is as follows. Given an object in the right manipulator's reachability subspace and out of the left manipulator's reach, transfer the object into an area only accessible by the left manipulator using in-air regrasping. Considering the nature of human regrasping, we are primarily concerned with efficiency, both from a computation and execution perspective, the potential for generalization, and the replication of human-like motions. Throughout the paper, we describe our algorithm and provide examples for the case when the object is reachable by the right manipulator and needs to be transferred to the left manipulator. We purposely present our work this way in an attempt to simplify and shorten the discussion, but we note that the algorithm works regardless of how the object needs to be transferred (i.e., independent of the starting location and transfer direction).

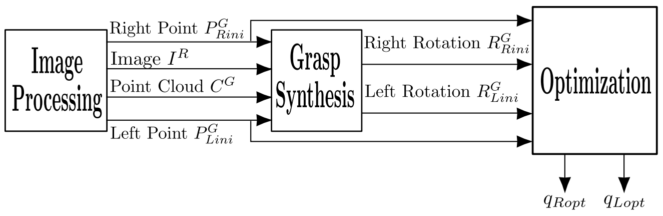


Fig. 1. High-level overview of proposed in-air regrasping algorithm.

As shown in Figure 1, the algorithm is composed of three components: Image Processing, Grasp Synthesis, and Optimization. The Image Processing, the purpose of which is to find two good grasping points in image space, exploits a stereo camera along with a state-of-the-art machine learning algorithm that we further improve to solve the regrasping challenge. The two points correspond to the points that each manipulator will use during the regrasping phase (i.e., when both manipulators hold the object simultaneously). We use a single stereo image as input, I^R, from which we can calculate the corresponding point cloud, C^G, thanks to stereo vision. Using the image, a machine learning algorithm assigns two good grasping points to the right and left manipulators, and converts them to initial Cartesian coordinates, P^G_{Rini} and P^G_{Lini}, respectively. The Grasp Synthesis component takes I^R, C^G, P^G_{Rini}, and P^G_{Lini} as input and outputs appropriate orientations for the right and left end-effectors, R^G_{Rini} and R^G_{Lini}, to grasp the object at the points P^G_{Rini} and P^G_{Lini}. This process is based on an efficient supervised learning unimanual grasping algorithm that is extended for bimanual grasps. Last but not least, the Optimization component searches the reachability subspaces of the arms to find the most efficient transfer configuration and outputs the right and left manipulators' configuration, q_{Ropt} and q_{Lopt}, to achieve the regrasping phase.

In some sense, we first find, using the Image Processing and Grasp Synthesis components, a couple of appropriate regrasping holds as if the object were in both manipulators' reachability subspaces. Then, in the Optimization component, we seek a transfer point minimizing the execution time required to put the object in both manipulators' reachability subspaces. We note that the algorithm is modular by design since each component can be swapped with different algorithms, potentially relying on different sensors that provide the same outputs. For example, the Image Processing and Grasp Synthesis algorithms could be replaced by a model-based grasping algorithm driven by a laser range finder. The components presented are designed, however, to be extremely efficient and result in real-time performance.

IV. Algorithm Details

A. Image Processing

For the Image Processing component, we utilize and improve a state-of-the-art feature-based algorithm originally proposed by Saxena et al. [15]. The algorithm assigns every pixel a probability of being a good grasping point by calculating a feature vector for each pixel and using supervised learning. The feature vector is acquired by applying a set of convolution filters at and around each pixel to be classified. The probabilistic binary classifier uses logistic regression:

$$P(z_i) = P(z_i|x_i;\theta) = \frac{1}{1+e^{-x_i^T\theta}}$$

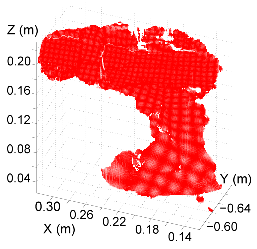
where z_i is a good grasping point, x_i the feature vector for pixel p_i, and θ the logistic regression coefficients. The logistic regression coefficients θ are learned through maximum likelihood estimation from a set of labeled positive and negative training examples.

Even though this image processing algorithm works well in practice, it is extremely time consuming and cannot run in real time when using typical image resolution (e.g., 320×240 or 640×480). We have found, however, that the bottleneck of the algorithm comes from the computation of the convolution filters. We additionally discovered, using a feature selection process based on principal component analysis, that edge filters were the primary contributors to the pixels' classification [2]. Given this information, we improve the algorithm's speed by reducing the number of pixels to classify, the process of which is highlighted in Figure 2. We start with our 640×480 camera image (Figure 2(a)) and compute its corresponding point cloud using stereo vision. The object in the point cloud is isolated (Figure 2(b)) using orthogonal distance regression [16] to remove surface planes (e.g., table) and quartile range outlier detection [13] to remove noise. The isolated point cloud, C^G, is used to crop the image around the object (Figure 2(c)), yielding I^R. We further reduce the search space by applying a Canny edge detector to the cropped image and only classifying the pixels that are in a 3×3 window around each pixel detected as an edge, resulting in a search region $\mathcal{R}$. We finally select our two good grasping points by using the formula

$$\arg\max_{i,j}\left(\frac{|P(z_i)+P(z_j)|}{2}\|p_i-p_j\|\right) \quad \forall i,j \in \mathcal{R}$$

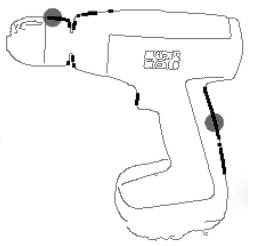
such that $P(z_i)$,$P(z_j) > 0.90$ and $P_i^G(z)$,$P_j^G(z) \geq$ 5cm, where P_i^G is the Cartesian coordinate for pixel p_i. This criterion chooses two grasping points that have a classification rate higher than 90% and for which points have a 5cm clearance from the table. The distance term, $\|p_i - p_j\|$, is introduced to make sure as much spacing as possible exists between the two end-effectors, to stay away from colliding solutions. The pixel selection process is shown in Figure 2(d). Finally, out of the two good grasping points converted to Cartesian coordinates, P_1^G and P_2^G we assign the point further away from the left manipulator to the right manipulator and vice-versa, yielding the points P^G_{Rini} and P^G_{Lini}. This relatively simple selection process works extremely well in practice while being very efficient.

(a)



(b)

(c)



(d)

Fig. 2. Step by step example of the Image Processing component. From a camera image (Fig. 2(a)), we isolate the object from the point cloud (Fig. 2(b)) and the image (Fig. 2(c)). Fig. 2(d) shows results from the Canny edge detection (grey pixels), the good grasping points found (black pixels), and the two points selected (circles).

The procedure results in a dramatic reduction of the pixel search space. Indeed, an original search space of 307,200 pixels is reduced, on average, to 38,339 pixels by cropping the object and to 12,429 pixels by searching the 3×3 window around each pixel labeled as an edge. Since the principal bottleneck of Saxena's algorithm lies in the feature vector computation, which needs to be performed for each pixel, the aforementioned search space reduction procedure lowers the average computation time from 3.25 seconds to 130 milliseconds (i.e., 96% reduction). We note that these results are conditioned on our application and that running the algorithm for images with multiple objects will evidently take longer. In those cases, our approach can nevertheless be applied, still providing a time reduction, although less significant. The time reduction allows the algorithm to process full-resolution images, which we have experimentally found to provide better results than with downscaled images.

B. Grasp Synthesis

Having found a grasping point for each manipulator, P^G_{Rini} and P^G_{Lini}, along with cropped versions of the object's image and point clouds, I^R and C^G, the Grasp Synthesis computes appropriate end-effector orientations, R^G_{Rini} and R^G_{Lini}, to correctly grasp the object. This component is based on our formerly-developed unimanual grasping algorithm [3] that we modify to accommodate bimanual grasping. As can be seen in Figure 3, the algorithm, which builds upon the Image Processing results, is comprised of three parts, each dependent on training data acquired by human subjects on a diverse set of objects. First, the Classification stage classifies the object to be grasped into one of N classes (acquired during training) using a one-against-one support vector machine classifier [10] with a parameter vector derived from the point cloud C^G. In the second stage, we determine the object's orientation relying on orthogonal distance regression [16] (using C^G) and image moments (using I^R) and comparing them to our training data. Knowing the object type and its orientation, we finally perform a Nearest Neighbor Search in the training data to compute the best orientations R^G_{Rini} and R^G_{Lini}, given the grasping points P^G_{Rini} and P^G_{Lini}.

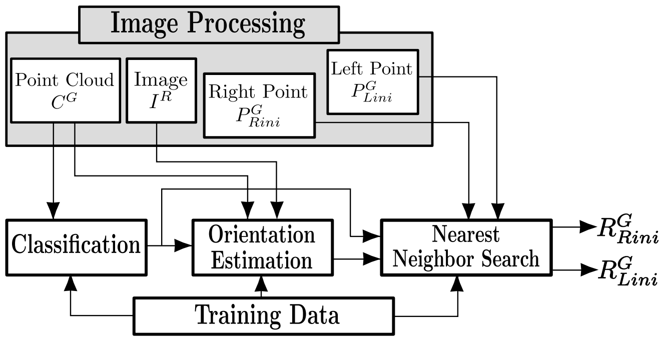


Fig. 3. Flowchart of the Grasp Synthesis component (bold), with inputs from the Image Processing (grey).

Interested readers are referred to the original publication [3] for more technical details. We simply note that the algorithm can generalize to objects that were not part of the training data. In that case, the unknown object will be classified as the closest one from our training database and a grasping orientation will be found based on that classification, a process which works 76.66% of the time (as opposed to 81.66% when using the same objects from the training data). We conclude this section by mentioning that the extension from unimanual to bimanual grasping is very efficient since the two most time consuming processes, the Classification and Orientation Estimation (101ms), only need to run once, whereas the most efficient process, the Nearest Neighbor Search (6ms), needs to run twice.

C. Optimization

By providing grasping information for both manipulators, in the form of P^G_{Rini}, P^G_{Lini}, R^G_{Rini}, and R^G_{Lini}, the Image Processing and Grasp Synthesis components have essentially found a regrasping phase for the object, as if it was located in both manipulators' reachability subspaces. Since, however, the object is out of the left manipulator's reach, we need to find the best object configuration, located in both manipulators' reachability subspaces, for the regrasping phase to occur. The object configuration will be attained by being grasped and moved by the right manipulator. We note that, since we are dealing with rigid objects, the object's configuration, when grasped, can be determined by the right manipulator's configuration. Consequently, we search the configuration space of the right manipulator. The Optimization process can be thought of as trying the regrasping phase found in the Image Processing and Grasp Synthesis components under different right manipulator configurations (and, consequently, object configurations) until an optimized result is found. We define an optimized result as one that minimizes the execution time of the regrasping task.

We exploit the Nelder-Mead optimization algorithm [14], whose pseudo-code is shown in Algorithm 1, because of its beneficial properties. Indeed, the algorithm does not require an objective function with a corresponding derivative (i.e., only a cost function is needed) and is computationally efficient. The algorithm works on a multi-dimensional triangle, a simplex, with each vertex corresponding to a potential solution to the optimization problem. For an n-dimensional optimization problem, the vertices are labeled x_1, x_2, …, x_{n+1}, where $x_i \in \mathbb{R}^n$. A function f is used to calculate the vertices' cost (line 2), which are manipulated thanks to a series of four geometrical operations: reflection (line 4), expansion (line 6), contraction (line 11), and deflation (line 13). The reflection operation, which yields a new vertex x_r, mirrors the worst vertex, x_{n+1}, across the centroid of the n best vertices, $\hat{x}$ (line 3). The expansion operation finds a new vertex, x_e, that is farther than the reflection vertex, x_r, but in the same direction. Alternatively, the contraction operation finds a point, x_c, between $\hat{x}$ and x_{n+1}, still along the same direction. The deflation operation is performed when everything else fails and shrinks all the simplex vertices by a factor of 2 towards the best solution, x_1. The reflection, expansion, and contraction operations are influenced by a set of coefficients (γ_r, γ_e, and γ_c, respectively) that dictate the operations' influence on the simplex. We empirically determined that $\gamma_r = 1$, $\gamma_e = 2$, and $\gamma_c = 0.5$ provide the best results, a finding corroborated by the authors of the original algorithm [14]. At each iteration, the algorithm tries to replace the worst vertex (line 7,8,10,12,15), in terms of cost, with one of the results from the geometrical operations. The geometrical operation used is based on simple comparisons of cost values (line 5,7,9,10,12). The algorithm iterates until the average distance from the geometrical center of the simplex to all its vertices falls below threshold $\varepsilon = 0.01$ (line 17). When the stopping condition is met, the best solution, x_1, is returned (line 18).

Algorithm 1 Optimization(x_1, x_2, …, x_{n+1})

```
1: repeat
2:     Order Vertices: $f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{n+1})$
3:     $\hat{x} \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$
4:     $x_r \leftarrow (1+\gamma_r)\hat{x} - \gamma_r x_{n+1}$
5:     if $f(x_r) < f(x_1)$ then
6:         $x_e \leftarrow (1-\gamma_e)\hat{x} + \gamma_e x_r$
7:         if $f(x_e) < f(x_1)$ then $x_{n+1} \leftarrow x_e$
8:         else $x_{n+1} \leftarrow x_r$
9:     else if $f(x_r) > f(x_n)$ then
10:        if $f(x_r) \leq f(x_{n+1})$ then $x_{n+1} \leftarrow x_r$
11:        $x_c \leftarrow (1-\gamma_c)\hat{x} + \gamma_c x_{n+1}$
12:        if $f(x_c) \leq f(x_{n+1})$ then $x_{n+1} \leftarrow x_c$
13:        else $x_i \leftarrow \frac{1}{2}(x_i - x_1)\ \forall_i$
14:    else
15:        $x_{n+1} \leftarrow x_r$
16:    end if
17: until $\frac{1}{(n+1)}\sum_{i=1}^{n+1} \|x_i - \bar{x}\| < \epsilon$
18: return $x_1$
```

The Optimization algorithm runs in the 6-dimensional optimization space of the right manipulator configuration defined by x_i. Specifically, x_i encompasses the end-effector's Roll (φ), Pitch (ϑ), Yaw (ψ), and X, Y, and Z coordinates, which accounts for the minimum representation in $SE(3)$. We note that the conversion from x_i to a position vector, P^G_{Ropt}, and a rotation matrix, R^G_{Ropt}, is straightforward; as is the conversion to an arm configuration q_{Ropt}. With a 6-dimensional optimization space, we need a set of 7 vertices $x_1, x_2, \ldots, x_7$ to bootstrap the algorithm. The first vertex is set as a guess estimate, x_g, with the six remaining vertices encompassing offset values of the guess estimate, which we set to 5 degrees for φ, ϑ, ψ, and 5 centimeters for X, Y, Z. In other words, $x_1 = x_g = [\varphi_g, \vartheta_g, \psi_g, X_g, Y_g, Z_g]$, $x_2 = [\varphi_g + 5, \vartheta_g, \psi_g, X_g, Y_g, Z_g]$, $x_3 = [\varphi_g, \vartheta_g + 5, \psi_g, X_g, Y_g, Z_g], \ldots,$ $x_7 = [\varphi_g, \vartheta_g, \psi_g, X_g, Y_g, Z_g + 5]$. Evidently, a good result is dependent on a good starting seed, which is itself dependent on our initial guess estimate. We create an automated guess estimate selection process using a Nearest Neighbor Search on trained data that is acquired offline. Given a set of manually selected example regrasping phases for various objects, the training data is obtained by running a sparse grid search on each example to determine the best configuration, as dictated by the cost function f. In order to find a good guess estimate, we can then make a quick Euclidean-based Nearest Neighbor Search query in the space of regrasping phases. This approach, which can be thought of as a simplistic learning algorithm works very well in practice and can be extended to any optimization algorithm for which it is easy to get training data.

Our cost function, f, minimizes the manipulators' execution time by minimizing the amount of joint movements that the manipulators undertake. In order to compute the cost, we first need to compute the arms' configurations, q_{Ropt} and q_{Lopt}. A visualization of the process is shown in Figure 4. We have already found, in the Image Processing and Grasp Synthesis components, a regrasping phase for the initial object's configuration, defined by P^G_{Rini}, P^G_{Lini}, R^G_{Rini}, and R^G_{Lini}. For each vertex x_i of the optimization's simplex, we straightforwardly determine P^G_{Ropt} and R^G_{Ropt}, from which we can determine, using Inverse Kinematics, q_{Ropt}. Given this information, we calculate the left arm configuration (q_{Lopt}) from its transformation (P^G_{Lopt} and R^G_{Lopt}):

$$P^{Rini}_{Lini} = (R^G_{Rini})^T (P^G_{Lini} - P^G_{Rini}) \quad (1)$$

$$V^G = (R^G_{Ropt})(P^{Rini}_{Lini}) \quad (2)$$

$$P^G_{Lopt} = P^G_{Ropt} + V^G \quad (3)$$

$$R^G_{Lopt} = (R^G_{Ropt})[(R^G_{Rini})^T (R^G_{Lini})] \quad (4)$$

The cost, c, of the function, f, is determined by finding, for each manipulator, the joint that experiences the maximum movement, which dictates the speed of the manipulator (see Equation 5). We define q_{Lstr} as the left arm's configuration before a regrasping phase is initiated.

$$f(x) = c = \max(|q_{Rini} - q_{Ropt}|) + \max(|q_{Lstr} - q_{Lopt}|) \quad (5)$$

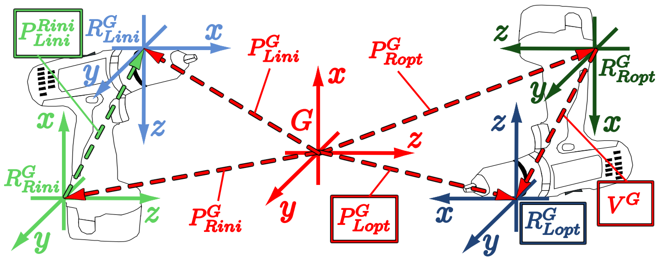


Fig. 4. Diagram representation showing the geometrical computation of P^G_{Lopt} and R^G_{Lopt}, given P^G_{Rini}, P^G_{Lini}, R^G_{Rini}, and R^G_{Lini}. The unknown parameters calculated from Equations 1, 2, 3, and 4 are boxed.

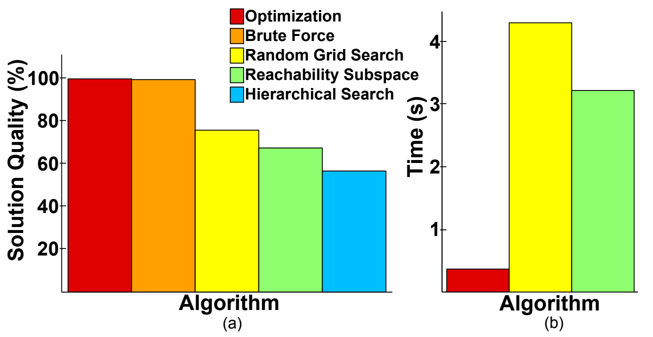


Fig. 5. Comparison of Solution Quality (a) and Computation Time (b) across different algorithms.

V. Experiments

In our first set of experiments, we compare the optimization portion of our algorithm against potential substitutes. These experiments are performed offline, since the quality of a solution is inversely proportional to the cost function $f(x)$ given in Equation 5 (i.e., the lower $f(x)$, the higher the solution's quality). Specifically, as is done for the Optimization component, each algorithm performs a search over a 6-dimensional grid in the manipulators' reachability subspace with parameters φ, ϑ, ψ, X, Y, and Z. The grid resolution is set to 10 degrees for the angles and 5 centimeters for the Cartesian coordinates. The algorithms are as follows.

Brute Force: an exhaustive search where each cell is explored successively. This algorithm is complete with respect to the grid resolution.

Random Grid Search: an anytime algorithm that randomly picks cells from the aforementioned grid. We stop the algorithm at multiple iterations and present in this section results for the iteration that yields the highest quality to computation time ratio. Due to the random nature of this process, we average the results over 10 runs.

Reachability Subspace: we use the robot's reachability subspace, where the points in the grid are ranked in decreasing order based on the number of manipulator configurations that can reach them. The idea behind this algorithm is that a position on the grid with many ways to get there by the manipulator will have more chances of stumbling upon a good solution. This is another anytime algorithm, the results of which are presented for the iteration yielding the highest quality to computation time ratio.

Hierarchical Search: a three-layer grid search, with each layer representing a smaller grid size. For each layer, the best solution is found and the area around that solution is explored, using a finer grid resolution. The first layer's grid size is set to 60 degrees for the angles and 25 centimeters for the Cartesian coordinates. The grid size is divided by 2 for each subsequent level.

The algorithms are run on 10 varying configurations of 4 different objects (a water bottle, a spray bottle, a coffee can, and a drill), the results of which are shown in Figure 5 in terms of the solution's quality and algorithm's computation time. The Brute Force algorithm is strictly used as a baseline for comparison and is not a viable solution because it takes 72.5 hours on average. Similarly, the Hierarchical Search terminates in an average of 66 seconds, which is much too long for our application. Given these observations, we omit these two algorithms in Figure 5(b). In Figure 5(a), the solutions' qualities are normalized and displayed in terms of percentages (i.e., the best solution, with the lowest cost, is set as 100% and the remaining solutions are normalized accordingly). It is clear that the optimization algorithm is not only extremely efficient, providing solutions more than 6 times faster than the second-fastest algorithm, but also competitive with the Brute Force approach. In fact, the Optimization algorithm provides better solutions, on average, than Brute Force (only by about 1%). This seemingly-surprising observation is however easily explained by the fact that the Brute Force algorithm searches a discrete grid, whereas the Optimization operates in continuous space. Specifically, the Optimization algorithm finds a better solution than Brute Force 84.61% of the time. For the remaining 25.39%, the optimization algorithm is within 4.08% of the grid search solution on average. The optimization algorithm finds better solutions than the other algorithms 100% of the time.

The same 10 varying configurations of the 4 objects were executed on our robotic platform, consisting of a static torso with two 7 degrees of freedom Barrett WAM arms and 4 degrees of freedom Barrett hands. We approach the object using the direction dictated by the orthogonal vector to the best-fit plane [16] near the grasping point and the orientation dictated by the algorithm. Additionally, we utilize a sparse roadmap, along with a collision detector, to guide the arm through collision-free paths. We note that this motion planning works for the experiments we present, but a better planner, based on rapidly-exploring random trees or probabilistic roadmaps, should be utilized for more complex scenarios comprised of more objects or furniture. In our first experimental setup, we manually dictate the grasping positions and orientations of the manipulators, consequently removing potential errors from the Image Processing and Grasp Synthesis components and investigating the Optimization component on its own. Being successful 87.5% of the time, with the cause for every failure being positional errors from the manipulator, it is clear that the Optimization component yields valid results. We then analyze the end-to-end algorithm by incorporating the Image Processing and Grasp Synthesis components back into the algorithm, a few snapshots of which are shown in Figure 6. In addition to the snapshots, the accompanying video shows successful examples of regrasping. Overall, the end-to-end algorithm performed very well, successfully completing 75% of the experiments. The majority of the errors were attributed to the Grasp Synthesis component failing to provide good orientations for one of the manipulators. It is worthwhile mentioning that the results are on par with the unimanual grasping algorithm [3].

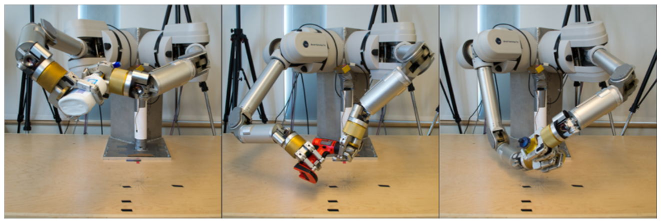


Fig. 6. Screenshots of our robot performing a regrasping phase for a spray bottle (left), drill (center), and water bottle (right).

We conclude this section by providing, in Table I, a categorized decomposition of the computation time spent by the end-to-end algorithm. As can clearly be seen, the algorithm is very fast, being capable of running in real-time on a standard 2.6GHz computer.

Component	Part	Time(ms)
Image Processing	Acquisition	40
	Denoising	42
	Pixel Selection	130
Grasp Synthesis	Classification	80
	Orientation Estimation	21
	Nearest Neighbor	16
Optimization	-	366
Total		**695**

TABLE I
ALGORITHM COMPUTATION TIME, DIVIDED BY PARTS.

VI. CONCLUSION

We have presented a bimanual grasping algorithm specifically designed to efficiently solve the problem of in-air regrasping. The algorithm possesses important properties such as its computational speed, small sensory requirements, generalization, modularity, manipulator-independence, and relevance to under-actuated end-effectors. As shown in the experimental section, we were unable to find a close rival algorithm both in terms of computational speed or solution quality. An extensive set of experiments have shown the algorithm's applicability to a real world platform composed of standard manipulators and under-actuated hands.

A few interesting directions can be taken to extend this work. While relatively straightforward, allowing the algorithm to regrasp the object multiple times would be a useful addition, requiring slight modifications to the presented components. Although the heuristic nature of the Image Processing's good grasping point selection worked well, it might be improved by some kind of learning or optimization, with the potential detriment of increased computational time. Similarly, the process to acquire the initial guess for the optimization algorithm could be improved with a supervised learning algorithm. Lastly, in order for the algorithm to operate in more complex environments, a better motion planning algorithm should be exploited to find appropriate paths for the manipulators.

ACKNOWLEDGMENT

This work is partially supported by the National Science Foundation under grant BCS-0821766. [1]

REFERENCES

[1] S. Arimoto, K. Tahara, J. Bae, and M. Yoshida. A stability theory of a manifold: Concurrent realization of grasp and orientation control of an object by a pair of robot fingers. *Robotica*, 21(2):163–178, 2003.

[2] B. Balaguer and S. Carpin. Efficient grasping of novel objects through dimensionality reduction. In *IEEE International Conference on Robotics and Automation*, pages 1279–1285, 2010.

[3] B. Balaguer and S. Carpin. Learning end-effector orientations for novel object grasping tasks. In *IEEE/RAS International Conference on Humanoid Robots*, pages 302–307, 2010.

[4] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner. Grasp planning in complex scenes. In *IEEE/RAS International Conference on Humanoid Robots*, pages 42–48, 2007.

[5] U. Castiello. Arm and mouth coordination during the eating action in humans: a kinematic analysis. *Experimental Brain Research*, 115(3):552–556, 1997.

[6] A. Churchill, S. Vogt, and B. Hopkins. The coordination of two-effector actions: Spoon-feeding and intermanual prehension. *British Journal of Psychology*, 90(2):271–290, 1999.

[7] A. Edsinger and C. Kemp. Human-robot interaction for cooperative manipulation: Handing objects to one another. In *IEEE International Symposium on Robot and Human interactive Communication*, pages 1167–1172, 2007.

[8] A. Edsinger and C. Kemp. Two arms are better than one: a behavior-based control system for assistive bimanual manipulation. In *International Conference on Advanced Robotics*, 2007.

[9] A. Gribova. *Bimanual Coordination: Electrophysiological and Psychophysical Study*. PhD thesis, Hebrew University of Jerusalem, 2001.

[10] A. Karatzoglou, D. Meyer, and K. Hornik. Support vector machines in R. *Journal of Statistical Software*, 15(9):1–28, 2006.

[11] A. Kawamura, K. Tahara, R. Kurazume, and T. Hasegawa. Simple orientation control of an object by regrasping using a dual-arm manipulator with multi-fingered hands. In *International Conference on Advanced Robotics*, pages 1–6, 2009.

[12] Y. Koga, K. Kondo, J. Kuffner, and J.C. Latombe. Planning motions with intentions. In *21st annual conference on Computer graphics and interactive techniques*, pages 395–408, 1994.

[13] J. Laurikkala, M. Juhola, and E. Kentala. Informal identification of outliers in medical data. In *Workshop on "Intelligent Data Analysis in Medicine and Pharmacology" at ECAI*, 2000.

[14] J. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[15] A. Saxena, J. Driemeyer, and A.Y. Ng. Robotic grasping of novel objects using vision. *International Journal of Robotics Research*, 27(2):157–174, 2008.

[16] C. Shakarji. Least-squares fitting algorithms of the NIST algorithm testing system. *Journal of Research of the National Institute of Standards and Technology*, 103(6):633–641, 1998.

[17] M. Simoneau, J. Paillard, C. Bard, N. Teasdale, O. Martin, M. Fleury, and Y. Lamarre. Role of the feedforward command and reafferent information in the coordination of a passing prehension task. *Experimental Brain Research*, 128(1–2):236–242, 1999.

[18] R. Spencer, R. Ivry, D. Cattaert, and A. Semjen. Bimanual coordination during rhythmic movements in the absence of somatosensory feedback. *Experimental Brain Research*, 94(4):2901–2910, 2005.

[19] P. Tournassoud, T. Lozano-Perez, and E. Mazer. Regrasping. In *IEEE International Conference on Robotics and Automation*, pages 1924–1928, 1987.

[20] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R Dillmann. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2464–2470, 2009.

[1]The views and conclusions in this article are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation.