Grasp Planning of 3D Objects Using Genetic Algorithm

Zichen Zhang
Department of Electrical and Computer Engineering
Dalhousie University
Halifax, NS, B3J 2X4, Canada
zichen.zhang@dal.ca

Jason Gu
Department of Electrical and Computer Engineering
Dalhousie University
Halifax, NS, B3J 2X4, Canada
Jason.Gu@dal.ca

***Abstract*—In this paper, we apply genetic algorithm (GA) to the optimization problem in grasp planning. This method can be used to find "pregrasps" for 3D objects in arbitrary shape and different dexterous hands, which serve as the first step of a complete grasping action. Each component of the GA planner is discussed in detail. The proposed algorithm is implemented in *GraspIt!* simulator [1]. It is tested on different hand-object combinations and the result shows that genetic algorithm is effective in finding high-quality pregrasps.**

***Index Terms*—Robot Grasping, Multifingered Hands, Genetic Algorithm**

I. Introduction

Robot grasping has been an active field of research in the past two decades. It allows a robot to have more impact on the environment rather than just sensing it. Tactile and vision sensors can be used to obtain information about the target object and possible obstacles. In this paper, we focus on the grasp planning problem based on the geometrical information of objects, which comes from pure vision feedback only. A stable grasp can be defined in the contact space of object(the fingertip placement on the object), or in the configuration space of the hand(the position and posture of the hand). Grasp planning therefore can be approached as an optimization problem: to find a better (more stable) grasp from this space. The difficulty of this problem is threefold. First, this space is usually too large to be searched effectively. Second, it is a discontinuous space since there are some contact points or hand configurations we want to avoid. Third, the quality metrics used as the objective function are usually non-linear.

As a general-purpose optimization method, GA is widely considered to be efficient and stable in finding the global optimum. Since it is suitable for solving nonlinear, discontinuous and high-dimensional problem. it has been applied in the domain of grasp planning. S. Mannepalli *et al.* [2] proposed using GA to find good fingertip placements in 2D space of prismatic objects. Sangkhavijit, C. *et al.* [3] applied GA in finding 4-fingered stable grasps from surface points of 3D objects. Mesgari, H. *et al.* [4] worked on a problem where a robotic arm system grasps a planar object with only one-point contact. GA was adopted to find the best grasping point which maximizes a performance index called MAG.

The above methods can be classified as *synthesis approach* [5]. The goal is to find the contact points on the surface of the object which will yield stable grasps. It relies on precise models of the object which is usually hard to obtain in the real world. *Heuristic grasp planning* [5] is another approach where optimization is performed in the configuration space of the hand. It usually contains two stages. The first step is to find a good pre-grasp [6], a hand posture that mimics the way humans preshape the hand before the actual grasp. Considering the large number of DOFs of dexterous hand, *eigengrasp* [7] can be used to to reduce the dimensionality of the search space. The pre-grasp is then searched in a low-dimensional space with sufficient information to form a good grasp. In the second stage, the hand is moved toward the object with the approach direction specified by the pre-grasp and then fingers are closed to conform to the surface of the object to complete the whole grasp action. J.J Fernandez *et al.* has proposed using GA in the *heuristic grasp planning* for three-fingered robot hands [8]. Their work had some limitations. Firstly, they only tested on three-fingered hands. Second, only a certain hand-object placement was allowed. Lastly, only some pre-defined 3D objects with regular shapes were tested. No other work has been found in applying GA to this approach.

In this paper, we will investigate the feasibility of applying GA on heuristic grasp planning of 3D objects in arbitrary shapes and with different hands. The selection of operators and parameters of the GA planner will be discussed in detail. The GA planner will be tested with different parameters and a variety of hand-object combinations. Quantitative results will be presented to evaluate the performance of the GA.

This paper is structured as follows. Section II introduces the optimization problem in grasp planning and the mathematical description is given. The design of the GA planner will be discussed in detail in Section III. In Section IV, the performance of the GA planner is examined with extensive tests and quantitative analysis results will be presented. Section V is the conclusion and some future direction.

TABLE I: Variable List

Property	Name	Definition	Range
Position	Tx	x-coordinate	[-250,250]
	Ty	y-coordinate	[-250,250]
	Tz	z-coordinate	[-250,350]
Orientation	θ	angle between the z-axis and the axis	[0,π]
	ϕ	angle between the projection of the axis on x-y plane and x-axis	[-π,π]
	α	rotation angle around the axis	[0,π]
Eigengrasp	EG[0,..,b]	amplitude along the eigen-grasp dimension	[-4,4]

II. Problem Formulation

We use the quality metric proposed in [7] to evaluate the quality of a *pre-grasp*:

$$Q = \sum_{all\ contacts} \delta_i \tag{1}$$

where δ_i is a measure of the distance between the desired contact locations on the hand and the object. For details, the reader is referred to [7].

The lower the Q, the better the pregrasp is. The optimization goal is to find a pregrasp that can minimize this quality measure. Therefore, the grasp planning problem in the configuration space of the hand can be represented as [7]:

$$\underset{p,w}{\operatorname{argmin}} Q(p,w), \text{ subject to} : p \in \Re^b, w \in \Re^6 \tag{2}$$

where $Q(p,w) : \Re^d \mapsto \Re$ is the objective function to be minimized over the variable space of dimension $d = b+6$. b is dimension of the eigengrasp space, p is a vector representing the hand posture, and w is a vector of the position and orientation of the wrist.

In our case, we will use Barrett Hand as a demonstration first. $b = 2$, i.e., the DOFs of the hand are mapped to a two-dimensional subspace. Axis-Angle representation is used for the 3D rotation. Totally we have 8 variables, as listed in Table I.

III. Genetic Algorithm on Grasp Planning Optimization

In this section, we will apply GA on grasp planning. The selection of operators and parameters are investigated in detail.

There are a variety of variations of genetic algorithms. A generalized flowchart of the genetic algorithm can be illustrated in Fig 1. The genetic algorithm used in our implementation is laid out in Algorithm 1. The hand state is encoded into a chromosome. A population of chromosomes evolve at each generation by applying *crossover* and *mutation* operators to the parents from the mating pool, which keeps the parents selected by the *parent selection* operator. All the

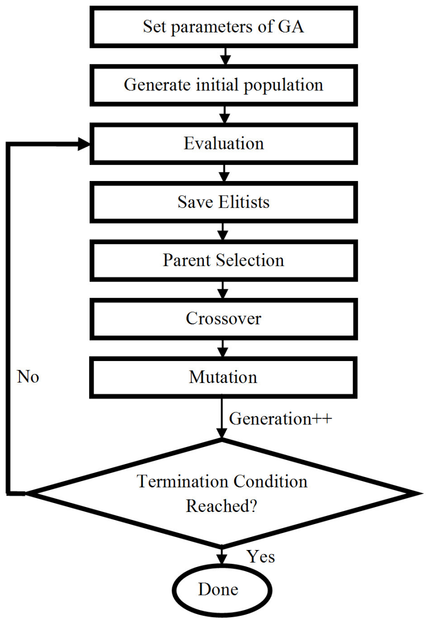


Fig. 1: A generalized flow chart of genetic algorithm

parents are saved in the mating pool. The quality of each chromosome is calculated by 1. n stands for the population size. The extra two slots in the population are used to keep Elitists. The best solutions over generations are saved in the "BestChromList". $flip$ is a function with a boolean return value. It returns $true$ with the specified probability. The implementation of this algorithm is inspired by [9]. We will discuss the details of each component of the GA planner in the following sections.

A. Operator Selection

First we need to decide on the encoding for a solution. There are several types of representation for a chromosome. As discussed in the previous section, the variables in this problem take real value. We naturally adopt real-valued representation. GA that uses floating-point encoding is called Real-Coded Genetic Algorithm (RCGA). It provides higher accuracy and faster speed as opposed to binary-coded GA.

Each chromosome contains 8 genes:

$$chromosome = < Tx, Ty, Tz, \theta, \phi, \alpha, EG_0, EG_1 > \tag{3}$$

The next step is to choose the *parent selection* technique, i.e, which chromosome will be chosen from current population to be saved to the mating pool to go through the rest operators. The most popular two selection methods are Roulette Wheel Selection and Tournament Selection.

In Roulette Wheel Selection, the probability of a chromosome of being chosen is proportional to their fitness. The main disadvantage of this method is that chromosomes with low fitness score rarely get the chance to be selected. The much better solutions will dominate the population

Algorithm 1 Genetic Algorithm on Grasp Planning

```
{Generate initial population with $(n+2)$ chromosomes}
for all chromosome do
    A random number for each gene within the range
end for
Evaluate the quality of each chromosome
Save two Elitists
Generation = 1

while Generation $\neq$ MaxGeneration do
    clear newPop
    {select $(n+2)$ parents from currentPop, save into the mating pool }
    $tournamentSelectionWithoutReplacement$(currentPop)
    for $i \leq n$ do
        parent1 = currentPop.mat(i)
        parent2 = currentPop.mat(i+1)
        {crossover}
        if $flip$(crossoverprobability) then
            $crossover$(parent1,parent2)
        end if
        {mutation}
        for all Genes in parent1,parent2 do
            if flip(mutationprobability) then
                Gaussian Mutation
            end if
        end for
        Evaluate the quality of parent1,parent2
        Add parent1,parent2 to the newPop
        $i = i+2$
    end for
    {Add Elitists to newPop}
    newPopElitist1 = best individual of the newPop
    newPopElitist2 = second best of the newPop
    if Q(newPopElitist1) < Q(currentElitist1) then
        Add newPopElitist1 to the newPop
        newPopElitist2 = newPopElitist1
    else
        Add currentElitist1 to the newPop
    end if
    if Q(newPopElitist2) < Q(currentElitist2) then
        Add newPopElitist2 to the newPop
    else
        Add currentElitist2 to the newPop
    end if

    {Elitists are the last two individuals in the newPop}
    newPopElitists = newPop($n$),newPop($n+1$)
    currentPopulation = newPopulation
    currentElitists = newPopElitists
    if Q(currentElitists) < Q(worst_BestPool) then
        insert newPopElitists into BestChromList
        worst_BestPool= worst quality in BestChromList
    end if
    Generation ++
end while
```

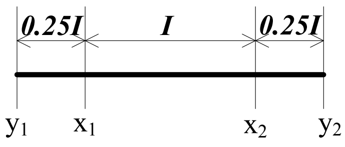


Fig. 2: Intermediate recombination

very fast which leads to premature convergence. Another disadvantage is that it can be very slow to move toward a better chromosome when the entire population have very similar fitness [10]. In Tournament Selection, T_s individuals are chosen to compete with each other and the one with better fitness score wins. T_s stands for tournament size. There are two slightly different implementations: Tournament Selection Without replacement (TSWOR) and Tournament Selection With Replacement (TSWR). In TSWR, candidates for each tournament are randomly chosen, while in TSWOR, an individual does not enter a tournament if it has already been chosen. The best chromosome in a population may have many copies or no copies in the mating pool using TSWR, while the best is guaranteed to have exactly T_s copies in the mating pool in TSWOR [11]. TSWOR is chosen in our implementation.

For crossover operator, we employed intermediate recombination operator proposed in [12], which is defined as:

$$\begin{aligned} child1 &= a_i \cdot chrom^{p_1} + (1-a_i) \cdot chrom^{p_2} \\ child2 &= b_i \cdot chrom^{p_2} + (1-b_i) \cdot chrom^{p_1} \end{aligned} \tag{4}$$

where $a_i, b_i \in [-0.25, 1.25]$,uniformly at random and a_i, b_i is different for each gene, and $chrom^{p_1}, chrom^{p_2}$ stands for the chromosome of parent1 and parent2. The solution space of this operator is illustrated in Fig. 2. The children generated using this operator span a slightly larger space than the parents, which from a statistical perspective ensures that the solution space does not shrink over the generations [13].

Gaussian Mutation is applied as the mutation operator. A new gene value is obtained by adding to the current value a number drawn from a Gaussian distribution $N(0, \sigma)$, where σ is a user-specified parameter [10]. We want to select the parameter σ to make sure that the possible solution can cover the full range. Since the value range for a Gaussian distribution is approximately equal to 6σ, we choose

$$\sigma_i = \frac{1}{5} \cdot r_i \tag{5}$$

where r_i is the range of the value of $gene_i$.

Elitism is often used to prevent the loss of the best chromosome in a population. In our case, two best chromosomes (Elitists) up to the present generation are kept and added to the next population. At each generation, the offspring are compared to the elitists. The elitists are replaced with the best two of the offspring if they have better quality and

TABLE II: Operator List

Operators	Method
Representation	Floating-point Numbers, 8 genes
Parent Selection	TournamentSelectionWithoutReplacement, $T_s = 2$
Crossover	Intermediate Recombination
Mutation	Gaussian Mutation
Elitism	Two Elitists, added to the next population
Survivor Selection	Generational Model

are kept otherwise. In addition, we adopt the *generational model* for survivor selection, i.e., the entire population are replaced by their offspring at each generation. To summarize, the operators are listed in Table II.

B. Parameter Tuning

Given particular GA operators, parameters tuning is crucial for achieving good performance. Three parameters are typically considered:

- n : Population size
- p_c : Probability of crossover
- p_m : Probability of mutation

It is suggested that good solutions can be obtained at a high crossover probability, a low mutation probability, and a moderate population size [14]. However, there is no general theory on parameter selection. The optimal parameters are generally problem-dependent. It may differ with the type of problem and operators. And the importance of crossover versus mutation has always been debated, directly determining the range of the parameter.

To simplify the tuning process we will start with some general recommendations of parameters: $n = 30-100$, $p_c = 0.6-0.9$ and $p_m = 0.01-0.1$. On the other hand, in order to gain a better understanding of this optimization problem in grasp planning, we are not restricted to these recommendations: some different sets of parameters will be tested, with a higher mutation probability p_m. A loose-grid search will be carried out first to obtain a rough picture of the solution space and locate the better area, followed by a fine-grid search near the more promising parameter sets.

IV. Results and Discussion

A. Test Platform

The algorithm is implemented in C++ under a modified version of *GraspIt!* which runs in ROS framework [15]. All the tests are performed in this modified *GraspIt!* simulator, on a desktop computer with a AMD Athlon II Quad-core 2.9Ghz CPU and 6G RAM.

B. Procedure and Performance Measure

The objects used for the tests are imported from the *Household Objects and Grasps Data Set* [16]. To tune the parameters, the program is executed with a Barrett Hand grasping a glass, shown in Fig. 3. We assume that there

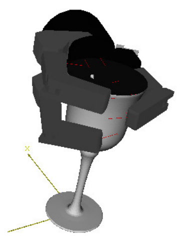


Fig. 3: Barrett Hand grasping a glass

is no obstacles obstructing the hand from grasping. The optimization with each set of parameters is performed over 10,000 generations and the best grasp found is saved. To account for the stochastic nature of GA, each test is repeated five times. The best pre-grasp qualities are taken as the average from the five runs. The on-line performance and off-line performance proposed by De Jong [14] are used to monitor the evolution of the quality of the grasps over generations and evaluate the convergence performance of GA. On-line performance at generation t is an average of the best from each generation in the past. Off-line performance keeps track of the best solution $Q(best)$ up to each generation and is calculated by taking an average of $Q(best)$ of the past generation at generation t.

C. Results

The quality of the best pre-grasps (lowest Q) found at different parameters are shown in Table. III. And the data is illustrated with the plots in Fig. 4.

As can be seen from the table and plots, the overall trend suggests the best grasp can be found at around $p_c = 0.8, p_m = 0.1$ and $n = 100$. p_m higher than 0.1 is more likely to have a negative impact on the performance. This is expected, as even though a higher p_m helps to keep the diversity of the population, it makes GA resemble a random search, opposing the original intention of applying GA. Note that even though the stochastic part of GA has been accounted for by taking the average from five runs, the result is somewhat stochastic. There are several "outliers", such as $Q(p_c = 0.75, p_m = 0.05, n = 50) = 14.591$ and $Q(p_c = 0.85, p_m = 0.08, n = 50) = 20.743$. Premature convergence is the possible reason for these outliers which leads to a performance largely dependent on the initial population.

We select $p_c = 0.8, p_m = 0.1$, and $n = 100$ as the parameter of GA. The on-line and off-line performance over 10,000 generations as well as the elitists and the $Q(best)$ of each generation are presented in Fig. 5. In this figure, we also include the on-line and off-line performance of a simple random planner. For a fair comparison, the random planner is ran for $10,000 * 100 = 1,000,000$ function evaluations. And the results for both planners are presented over function evaluations rather than generations. The figure shows that

TABLE III: Statistics of the best pre-grasps found at different parameters

			p_m							
			0.01	0.05	0.08	0.10	0.12	0.2	0.3	0.4
p_c	0.60	$n = 50$	28.98	20.772	18.656	14.468	20.735	15.397	18.515	23.025
		$n = 100$	26.816	22.951	20.678	19.585	20.905	18.133	17.961	16.266
	0.65	$n = 50$	25.289	22.411	15.865	15.011	18.933	16.767	17.456	21.931
		$n = 100$	35.441	25.16	14.418	22.188	16.519	25.072	19.14	18.849
	0.70	$n = 50$	32.286	27.814	19.391	15.88	15.984	15.705	17.831	17.284
		$n = 100$	27.405	15.186	14.538	14.799	22.398	17.996	21.956	18.521
	0.75	$n = 50$	28.754	14.591	23.099	21.483	19.47	18.672	22.203	16.796
		$n = 100$	23.464	21.947	15.882	14.707	16.448	17.24	19.319	16.744
	0.80	$n = 50$	29.733	19.703	14.309	15.471	15.362	15.512	20.16	22.201
		$n = 100$	29.401	18.420	15.452	14.041	15.629	17.328	17.355	19.713
	0.85	$n = 50$	21.965	16.046	20.473	15.774	14.256	17.083	16.721	17.146
		$n = 100$	28.01	18.193	16.09	15.313	17.141	16.599	18.202	20.303
	0.90	$n = 50$	27.231	22.774	18.658	17.217	20.575	20.436	17.642	17.911
		$n = 100$	32.975	16.766	18.607	18.455	21.034	23.766	22.472	17.881

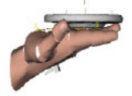
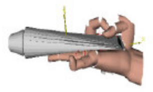
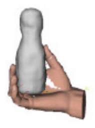
Hand	Barrett				Human			
Object	Glass	Saucer	Toothpaste	Bottle	Glass	Saucer	Toothpaste	Bottle
Average pregrasp quality	14.041	10.302	16.308	13.182	18.540	8.107	14.275	11.951
Best pregrasp found								

Fig. 6: Best pre-grasps found by GA planner for different hand-object combinations

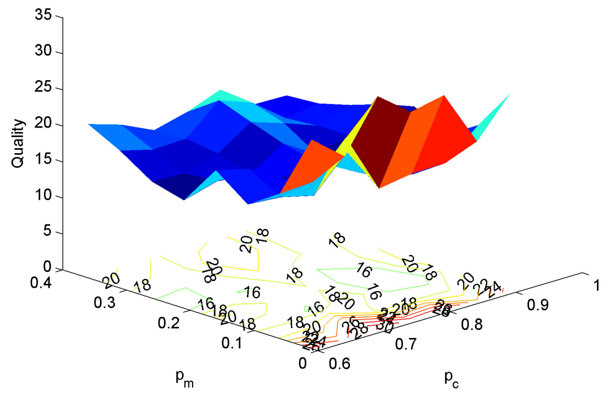
the population evolves rapidly over the first 5,000 generations and grows slowly afterwards. In the later stage, the population reaches a stable state. Most individuals are within a small space around the elitist. During this time period, crossover hardly produces new individuals and and thusly is not efficient in the search of the solution space. However, the diversity is maintained by mutation so that the rest of the configuration space still gets the chance to be searched over, which is indicated in the on-line performance. Compared with the simple random planner, we can see that GA is far more efficient. That is the reason we do not want to use a high mutation probability for GA, since that will make GA act like a random algorithm. It should be noted that in the parameter tuning process, the best parameters we got was for Barrett hand grasping the glass only. If either the hand or the object was changed, that set of parameters may not bring the best performance because the solution space would be changed. This addresses a very important characteristic of this grasp optimization problem. It requires the algorithm to be robust to different search space. To further evaluate the performance and the robustness of this proposed GA planner, we test it on different hand-object combinations. The Barrett Hand and the Human Hand model released with the *GraspIt!* simulator are employed in this study. The DOFs and the eigengrasp (EG) space of the hand models are listed in Table. IV. The result is shown in Fig. 6. The average best pregrasp quality for each hand-object combination is calculated out of five runs. And the best pregrasps found were included in the figure.

TABLE IV: Dimension of the search space

Hand Model	DOFs and Eigengrasp Space	GA Encoding Length
Barrett	4DOFs $\mapsto$ 2 EG	8 genes
Human Hand	20DOFs $\mapsto$ 6 EG	12 genes

V. Conclusion

In this paper, we applied genetic algorithm to grasp planning of 3D objects and dexterous hand. The concepts essential in building a genetic algorithm based grasp planner are introduced. The operators and parameters of genetic algorithm are known to have a crucial impact on the performance and may depend on specific problems. We implemented this algorithm using *GraspIt!* simulator and performed extensive tests to choose the appropriate parameters for grasp planning. The ones that give the best performance are chosen for further tests. The quantitative results on a number of hand-object


(a) population size $n = 50$

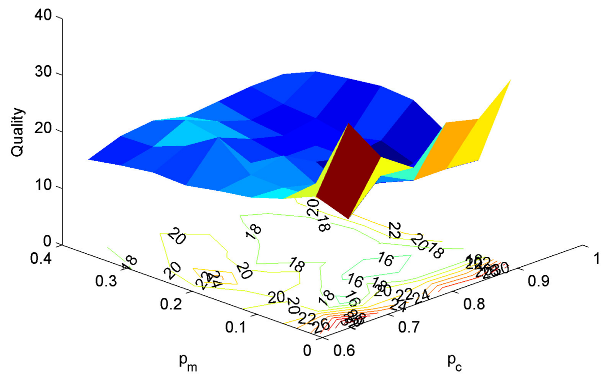

(b) population size $n = 100$

Fig. 4: The best quality of the pre-grasps

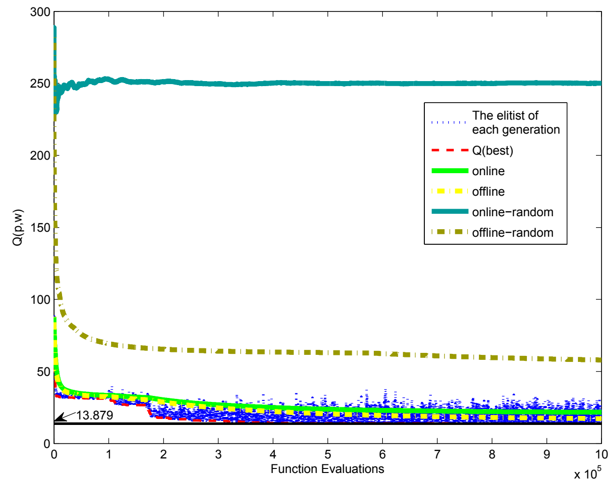


Fig. 5: The on-line and off-line performance of the genetic algorithm

combinations indicate that genetic algorithm can be used to obtain a good pre-grasp efficiently and it is robust to different search space. For the future work, it will be interesting to apply adaptive method in GA to improve its robustness. The quality metric we used is for selecting a good pregrasp, but cannot be used as a quality score for the final grasp. We will try using GA with other quality measures for planning final grasps.

References

[1] A. T. Miller, "Graspit!: A versatile simulator for robotic grasping," Ph.D. dissertation, Columbia University, New York, USA, june 2001.

[2] S. Mannepalli, A. Dutta, and A. Saxena, "A multi-objective ga based algorithm for 2d form and force closure grasp of prismatic objects," *I. J. Robotics and Automation*, vol. 25, no. 2, 2010.

[3] C. Sangkhavijit, N. Niparnan, and P. Chongstitvatana, "Computing 4-fingered force-closure grasps from surface points using genetic algorithm," in *Robotics, Automation and Mechatronics, 2006 IEEE Conference on*, dec. 2006, pp. 1 –5.

[4] H. Mesgari, F. Cheraghpour, and S. Moosavian, "Application of mag index for optimal grasp planning," in *Mechatronics and Automation (ICMA), 2011 International Conference on*, aug. 2011, pp. 2171 –2176.

[5] D. Bowers and R. Lumia, "Manipulation of unmodeled objects using intelligent grasping schemes," *Fuzzy Systems, IEEE Transactions on*, vol. 11, no. 3, pp. 320 – 330, june 2003.

[6] A. Miller, S. Knoop, H. Christensen, and P. Allen, "Automatic grasp planning using shape primitives," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 2, sept. 2003, pp. 1824 – 1829 vol.2.

[7] M. Ciocarlie and P. Allen, "Hand posture subspaces for dexterous robotic grasping," *The International Journal of Robotics Research*, vol. 28, pp. 851–867, 07/2009 2009.

[8] J. Fernandez and I. Walker, "Biologically inspired robot grasping using genetic programming," in *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, vol. 4, may 1998, pp. 3032 –3039 vol.4.

[9] K. Sastry, "Single and multiobjective genetic algorithm toolbox in c++," University of Illinois at Urbana-Champaign, Urbana IL, Tech. Rep. IlliGAL Report No. 2007016, 2007.

[10] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. SpringerVerlag, 2003.

[11] K. Sastry and D. E. Goldberg, "Modeling tournament selection with replacement using apparent added noise," in *Intelligent Engineering Systems Through Artificial Neural Networks. Proceedings of the Conference ANNIE 2001*, vol. 2, 2001, pp. 129–134.

[12] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm i. continuous parameter optimization," *Evol. Comput.*, vol. 1, no. 1, pp. 25–49, Mar. 1993.

[13] H. Pohlheim, "GEATbx - genetic and evolutionary algorithm toolbox for use with matlab. http://www.geatbx.com/,."

[14] K. A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems." Ph.D. dissertation, Ann Arbor, MI, USA, 1975, aAI7609381.

[15] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[16] M. Ciocarlie, C. Pantofaru, K. Hsiao, G. Bradski, P. Brook, and E. Dreyfuss, "A side of data with my robot: Three datasets for mobile manipulation in human environments," *IEEE Robotics & Automation Magazine, Special Issue: Towards a WWW for Robots*, vol. 18, 06/2011 2011.