Abstract—In this project I attempt to learn open loop grasping parameters by implementing a two stage Gaussian Process (GP) based Upper Confidence Bound bandit solving algorithm. Through the course of the project, parametrized grasping strategies were developed, grasp verification using soft fingers via tandem grasp was developed, the bandit algorithm to exploit minimal number of robot interactions and learn parameters was developed, and rudimentary block detection algorithm was developed.

I. INTRODUCTION

Uncertainties in grasping objects by a manipulator arise due to (a) imprecise self localization: The manipulator’s estimate of its location is incorrect due to cable extension, link bending, incorrect encoder calibration or soft fingers (As is our case with the Kinova manipulator), (b) imprecise object localization: The robot’s estimate of the object to be grasped is incorrect due to errors in visual or depth processing. One solution to this problem is in using closed loop grasping. In such a method of grasping information such as tactile feedback from the manipulator’s fingers [1] or visual feedback by using a camera’s data [2] can be used to reduce uncertainty. However, tactile sensors are quite expensive and often require frequent calibration. Visual feedback from a stationary camera is often plagued by occlusions from the robot and on-robot cameras still have to deal with pesky extrinsic parameter calibration.

Open loop grasping, on the other hand, is immune to external sensor calibration, at the cost of taking more conservative and time intensive grasping actions. An example of such a grasping strategy is the push grasp presented by Dogar et al. in [3]. Here at the cost of a longer, drawn out grasp motion, the authors are able to grasp objects with considerable uncertainty. In the project presented in this report, I look at open loop grasping with longer drawn out motions in localized task space that involve only wrist and finger joint motions. By parametrizing such a grasp policy we can optimize the policy for a given object. For the purpose of finding good grasp policies, the problem of grasping is framed as a multi-arm bandit with continuous arms.

In our formulation of the multi-arm bandit problem [4], a particular arm corresponds to a set of grasp parameters. The reward obtained is determined by the robot’s success in grasping the object. Hence the evaluation of a specific arm (or grasp policy) involves executing a real grasp on the robot. Executing simulated grasps would be a valid strategy if the fingers of the robot could be modelled effectively; However we are working with a robot with soft fingers. This provides the robot with better grasping ability at the cost of complicated dynamics [5].

The quandary of exploration versus exploitation is even more significant in our problem since the robot must execute a given grasp policy to evaluate it. Upper Confidence Bound (UCB) algorithms have been shown to be effective in such scenarios [6]. These algorithms however require estimates of the variance in parameter prediction, which can be done by carefully chosen heuristics. Gaussian Processes (GP) [7] are powerful generative regressors that provide an efficient way to model both the fitness predictions and variance of a given grasp policy in domains with small training datasets.

GP regression is dependent on hyperparameters that define the priors on the fitness distribution. These hyperparameters can be be optimized via cross validation or via expectation maximization [8]. However, optimization cannot be performed during the initial iterations of policy evaluation since the amount of training data is insufficient for effective optimization. So I propose a two stage GP-UCB algorithm that is purely explorative in the first stage and then follows standard UCB in the second stage.

GPs have previously been used in learning optimal control strategies in walking robots [9], which motivates using GPs for this project. There also exist no regret bounds on the GP-UCB algorithm [10].

II. THEORY & PRELIMINARIES

Multi Arm Bandits: The n-armed bandit problem [11], involves the selection of 1 action among n possible actions at time t in order to maximize the reward. The relation of the reward to a specific action been taken is stochastic in a general setting. Hence for every action we associate an expected reward, which is termed value. Knowing the value for every action solves the problem trivially by choosing the action with the maximum value. In a general case, the values of an action aren’t known with certainty.

A. Robot, Grasping System and Vision System

Exploitation vs Exploration: At any time step, an estimate of the action values is maintained. If the action with the maximum estimated value (greedy action) is chosen, the action is said to be exploitative. However if a non greedy action is chosen, the action is said to be explorative. Exploration enables improving the estimate of the non greedy action by evaluating it. Ideally, a good algorithm will start of being explorative and then, once the estimates of the values have low variance, be exploitative.

Universal First strategy: In this approximate solution to the bandit problem, a pure exploration stage is followed by a pure
covariance between the sets $X$ the predicted targets $y$ of which any finite subset have a joint gaussian distribution. The Gaussian Process (GP) is a collection of random variables, a distribution over functions and inference is performed in this function space. The GP is a useful tool for regression problems where the data comes from a process that is unknown or complex. The upper confidence bound (UCB) strategy is as follows:

$$A_t = \arg\max_{a_t} [Q_t(a) + \beta \sigma_t(a)]$$

where $A_t$ is the action selected to be evaluated at time $t$, $Q_t(a)$ is the estimated value of the action $a$. $\sigma_t(a)^2$ is the estimated variance in the estimation of the value of the action and $\beta$ is the importance given to the variance. For appropriate values of $\beta$, the quantity being maximized provides an estimate of the upper bound on the action value, hence the name Upper Confidence Bound.

Gaussian Process (GP): A GP [7] can be viewed as a distribution over functions and inference is performed in this function space. The GP is a collection of random variables, of which any finite subset have a joint gaussian distribution. $m(x)$ is defined as the mean of the process $f(x)$, while $k(x, x')$ defines the covariance function. We assume that $m(x)$ is zero. Given a training set $(X, Y)$ and test data $X_T$, the predicted targets $y_{pred}$ is jointly normal with $y$ as:

$$\begin{bmatrix} y \\ y_{pred} \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} 0 \\ \mathbf{K}(X, X) \end{bmatrix}, \begin{bmatrix} \mathbf{K}(X, X_T) \\ \mathbf{K}(X_T, X) & \mathbf{K}(X_T, X_T) \end{bmatrix} \right)$$ (2)

Here $\mathbf{K}(X, X^*)$ is the kernel matrix that defines the covariance between the sets $X$ and $X^*$, where

$$K_{i,j} = k(X_i, X_j)$$ (3)

The posterior distribution is also gaussian and is given by

$$y_{pred} \sim \mathcal{N}(\mathbf{m}_{pred}, \mathbf{S}_{pred})$$

$$\mathbf{m}_{pred} = \mathbf{K}(X_T, X)\mathbf{K}(X, X)^{-1}y$$

$$\mathbf{S} = \mathbf{K}(X_T, X_T) - \mathbf{K}(X_T, X)\mathbf{K}(X, X)^{-1}\mathbf{K}(X, X_T)$$ (4)

$m_{pred}$ is the estimated values for actions in $X_T$, while the diagonal of $\mathbf{S}$ contains the estimated variance information for the corresponding action values.

Radial Basis Function (RBF) kernel: I use the RBF kernel in the creation of the kernel matrix $K$. The RBF kernel for is given by

$$k(x, x^*) = \sigma_f^2 \exp\left( -\frac{d^T M d}{2l^2} \right) + \sigma_n^2 \delta_{kr}(x, x^*)$$ (5)

$\sigma_f^2, l$ and $\sigma_n^2$ are kernel hyper parameters [12] that signify the signal variance, length scale and the noise variance respectively. Here $d = x - x^*$ and $M$ is an appropriate scaling factor that can also be estimated as a hyperparameter. However since we normalize the data before passing it through the GP regressor, we set $M$ to the identity matrix.

III. APPROACH/METHOD

A. Robot and Vision system

Experiments are carried out on the 6 continuous DOF Kinova robot arm. The arm is equipped with 2 soft gripper fingers which can be individually controlled. The objects being grasped are small cubical blocks. A visualization of the system can be seen in Fig.2.

The robot’s vision system includes a depth camera (Structure sensor from Occipital, Inc.). The recognition of the blocks is done via hole rectification followed by depth thresholding. A blob detector is then used to cluster the segmented image to get the block positions. The block
recognition method works for multiple blocks. The system is visualized in Fig. 1.

NOTE: Due to mechanical failures in mounting, the vision system wasn’t used during the policy evaluation.

B. Parametrized Grasp Policy

In this section we explore parametrized open loop grasp policies. Since the robot we are operating with has continuous wrist rotation, we exploit this fact to create grasp policies that only include the wrist and finger motions. Furthermore, we use symmetric finger actuation, hence there are effectively 2 degrees of freedom being controlled by the grasp policy. Considering only the wrist and the fingers ensures that if the robot is able to plan to a specific point to grasp in task space, the grasp policy can always be executed. An exception to this is when the grasp is so close to obstacles that rotations of the wrist cause collisions between the fingers and the obstacle. Two such parametrizations are discussed below.

1) One Parameter Policy: The simplest possible grasp strategy would be constant motion in the wrist \((w)\) and a linear relationship between the finger \((f)\) actuator and the wrist. The policy is described in Algorithm 1.

Algorithm 1 One parameter grasp policy

1: Go to initial grasp position. Open fingers \(f(t=0)=0\).
2: Initialize trajectory object \(T\).
3: while \(f(t)\) not equals \(JointLimits(f)\) do ▷ Closing the gripper
4: \(t = t + \delta\)
5: \(w(t) = kt\)
6: \(f(t) = \alpha w(t)\)
7: \(T.push(t,w(t),f(t))\)
end while
8: ExecuteTrajectory(T)
9: return \(T\)

This policy is parametrized by \(\alpha\). Decreasing the value \(\alpha\) creates a longer drawn out motion that sweeps a greater area in task space. During experiments, it was noted that this policy was good for block positions that had small amounts of displacements. However it failed and pushed blocks away if the displacements were a little larger. We hence explore a more complex policy.

2) Three Parameter Policy: The intuition behind this method is that by executing multiple open-close finger motions at different wrist positions, the uncertainty of the block position can be reduced to a much smaller region. The policy is described in Algorithm 2.

The parameter \(\alpha\) controls the speed of finger closure and opening. Parameter \(\beta\), also called back factor controls the ratio of speed of opening with the speed of closing. Ideally \(\beta > 1\) since ideally, we want the gripper to open as fast as possible. \(NoI\) controls the number of grasp open-close cycles. Graphs that depict the variation of policy with the parameters can be seen in Fig. 3.

Algorithm 2 Three parameter grasp policy

1: Go to initial grasp position. Open fingers \(f(t=0)=0\).
2: Initialize trajectory object \(T\) \(\text{iteration\_count} = 0\).
3: while \(iteration\_count < NoI\) do ▷ Closing the gripper
4: \(t = t + \delta\)
5: \(w(t) = kt\)
6: \(f(t) = \alpha w(t)\)
7: \(T.push(t,w(t),f(t))\)
end while
8: while \(f(t)\) not equals 0 do ▷ Opening the Gripper
9: \(t = t + \delta\)
10: \(w(t) = kt\)
11: \(f(t) = -\beta \alpha w(t)\)
12: \(T.push(t,w(t),f(t))\)
end while
15: end for
16: ExecuteTrajectory(T)
17: return \(T\)

C. Grasp Verification

For effective grasp policy evaluation, we require a method to verify if a grasp is successful. This is challenging since using tactile or visual feedback is essentially violating our assumptions for the need of open loop grasping. Using the finger’s joint actuator information may provide insights into the grasp configuration. However due to its compliant nature, the actuator data from the soft fingers provide no distinction between grasping the block vs grasp failure.

One way to circumvent this problem is to accentuate the grasp by controlling the finger actuators independently. In this method one finger is first commanded to hold a close position followed by the second finger. This tandem nature of finger grasps creates a spike in the finger joint actuator’s effort (Fig. 3) that can be used to verify the grasp.
Fig. 4. Sequence of finger motions to verify grasp.

Fig. 5. Finger effort profile generated during the execution of grasp verification (Algorithm 3). A clear spike can be seen which is detected by a Butterworth filter

The algorithm is described in Algorithm 3. Visually, the verification can be seen in Fig. 4.

D. Two Stage GP UCB algorithm

The idea of this upper confidence bound (UCB) action selection is that the variance in the estimate is considered. So we want to evaluate points that have high variance to get better estimates. We however also want to search in good regions. So we sample points in regions that have a combination of high posterior reward and a high variance.

Since we are using GP regressors as bandit fitness estimators, we also have to think about optimizing hyper parameters. This is the crux of why we two steps. Optimization (EM style) is only possible when we have at least a few evaluations. So the first step is pure explorative in regions with high variance. The second step is a mixture of explorative and exploitative which gets more exploitative as the variance decreases. The complete algorithm is described in Algorithm 4.

The samples under consideration for evaluation are uniformly sampled in the parameter space. In Stage 1, samples that have the largest variance are chosen. This is akin to setting $\beta = \infty$ in Eqn. 1 which over $\mathcal{N}$ iterations reduce the overall variance in the GP regressor. In stage two, samples with maximal UCB are chosen to be evaluated with $\beta = 3$, since the bulk of uncertainty lies under the $+3\sigma$ bound. For the final parameter, regression is performed over a more densely sampled space to improve the prediction.

### Algorithm 3 Grasp Verification

1. Go to initial grasp position. Open fingers $f_1(t = 0) = 0, f_2(t = 0) = 0$. Initialize trajectory object $T$, $\text{graspSuccess} = \text{False}$
2. while $f_1(t)$ not equals $\text{JointLimits}(f_1)$ do $\triangleright$ Closing finger 1
   3. $t = t + \delta$
   4. $f_1(t) = k t$
   5. $f_2(t) = 0$
   6. $T$.push($t, f_1(t), f_2(t)$)
3. end while
4. $t_f_2 = t$
5. while $f_2(t)$ not equals $\text{JointLimits}(f_2)$ do $\triangleright$ Closing finger 2
   6. $t = t + \delta$
   7. $f_1(t) = \text{JointLimits}(f_1)$
   8. $f_2(t) = k(t - t_f_2)$
   9. $T$.push($t, f_1(t), f_2(t)$)
10. end while
11. while ExecuteTrajectory($T$) do
12. if Spike detected in finger 1 effort then
13. $\text{graspSuccess} = \text{True}$
14. end if
15. end while
16. return $\text{graspSuccess}$

E. Grasp Evaluation

The formulation of policy search as a regressor allows us to use more interesting fitness functions that encapsulate desirable properties in the grasp. The grasp evaluator used in this project is described in Algorithm 5. The method penalizes policies that take a long time to execute.

IV. RESULTS

A. 2 Stage GP-UCB evaluation

I evaluate the proposed GP-UCB algorithm on two functions defined on $x \in [0, 1]^3$, since the policy to be learnt is 3 dimensional:

1) $f_{\text{easy}}(x) = x[0]x[1] + \sin(3x[2])$
2) $f_{\text{tough}}(x) = \cos(25 \sin(5x[0]) \exp(15x[1])) + \sin(\exp(x[0]) \sin(20x[2])) \sin(3x[1])) + \cos(\exp(\cos(25x[2])))x[2]^{3}$
Two Stage GP-UCB

Algorithm 4 Two Stage GP-UCB

1: Given: N number of steps and epsilon ratio $\epsilon$; $D \subset \mathbb{R}^n$ is the space of possible parameters; $Eval(x)$ is the fitness of the n dimensional parameter $x \in D$; $GPR(X,y,X^*)$ provides the Gaussian Process estimates of the mean and variance of the value for action $X^*$ according to Eqn[4]

2: procedure INITIALIZATION
3: $X = \emptyset$, $y = \emptyset$
4: $X_{rest} = uniformGrid(D,R)$ \,$\triangleright$ \,R defines resolution
5: $x_{seed} = random(D)$ \,$\triangleright$ \,Start with random parameter
6: $y_{seed} = Eval(x_{seed})$ \,$\triangleright$ \,Evaluate $x$ by executing policy
7: $X.push(x_{seed})$, $y.push(y_{seed})$
8: end procedure
9: procedure STAGE 1: EXPLORATIVE
10: for n in 1 : cN do
11: $mean, std = GPR(X,y,X_{rest})$
12: $x_{sample} = argmax(std)$
13: $y_{sample} = Eval(x_{sample})$
14: $X.push(x_{sample})$, $y.push(y_{sample})$
15: end for
16: end procedure
17: procedure STAGE 2: UCB
18: for n in cN : $(1-\epsilon)N$ do
19: $mean, std = GPR(X,y,X_{rest})$
20: $x_{sample} = argmax(mean + \beta * std)$
21: $y_{sample} = Eval(x_{sample})$
22: $X.push(x_{sample})$, $y.push(y_{sample})$
23: end for
24: end procedure
25: procedure SELECT BEST PARAMETER
26: $X_{out} = uniformGrid(D,R^*)$ \,$\triangleright$ \,Finer resolution
27: $mean, std = GPR(X,y,X_{out})$
28: return argmax($mean$)
29: end procedure

Algorithm 5 Grasp Evaluation

1: Input : Parameter vector $x$ of length 3; set $(\alpha, \beta, NoI) \leftarrow x$; set maxExecTime=100s.
2: $T \leftarrow ExecutePolicy(\alpha, \beta, NoI)$ \,$\triangleright$ \,Algorithm 2
3: $graspSuccess \leftarrow GraspVerify()$ \,$\triangleright$ \,Algorithm 3
4: $t \leftarrow T.getExecutionTime()$
5: if $graspSuccess = False OR t > maxExecTime$ then
6: return $-10$
7: else
8: return $1 + \frac{maxExecTime-t}{maxExecTime}$
9: end if

By plotting the difference between the best estimated action value and the actual best action value, we can understand the effects of the parameters N and $\epsilon$ in Algorithm 4. These plots in Fig. 6 show that we need greater exploration for functions with multiple local minima.

B. GP-UCB on the robot

I run the proposed algorithm on the robot with N = 20 and $\epsilon = 0.2$. This is followed by 5 more iterations with the estimated best policy (Algorithm 4). The robot successfully finds a grasp policy that grasps objects 5/5 times in pure exploitation. A graph of the evaluated reward (Algorithm 5) can be seen in

C. Open loop grasp policy

The estimated best action corresponds to $(\alpha = 4.77, \beta = 9.55, NoI = 7)$, which takes 78 seconds to execute. On a different of the algorithm $(\alpha = 1.85, \beta = 10, NoI = 3)$ were found to be the best parameters that executed in 28 seconds, however the policy didn't work 100% of the time in the pure exploitation phase.

Fig. 8 demonstrates the robot executing this grasp policy. It can be noted how initial grasps fail, but towards the end of the policy, the robot localises the object to the center of the finger gripping position.
Fig. 7. Graph of the Evaluated reward with the iteration number. It can be seen that in the end we find a good policy. The GP-UCB is run with $N = 20$ and $\epsilon = 0.2$ followed by 5 iterations of pure exploitation.

REFERENCES


