2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Estimating Manipulability of Unknown Obstacles for Navigation in Indoor Environments

Christopher Clingerman[1], Daniel D. Lee[2]

Abstract— **The challenging task of navigating in cluttered environments has been studied extensively with indoor autonomous mobile robots. However, few approaches attempt to estimate real-valued costs for manipulating said obstacles with no prior knowledge of the environment. Our approach not only estimates these costs but also models the uncertainty inherent in making such estimates. We present an algorithm that, with no prior knowledge of the environment, allows a mobile robot to determine which obstacles are movable and which are not while navigating a cluttered environment. The algorithm also applies this knowledge of manipulability to obstacles encountered in the future that are similar in appearance to ones previously seen. Using our approach, a mobile robot can act intelligently about uncertain information as well as successfully navigate initially unknown indoor environments without relying on human-provided information.**

I. INTRODUCTION

Cluttered indoor environments present a difficult problem for autonomous navigation by mobile robots. Traditional navigation algorithms simply assume that if the robot avoids all detectable obstacles, it will safely reach its destination. However, many objects found in typical indoor environments are easily manipulable, several even by simple pushing. By eliminating the assumption that all obstacles are immovable, we enable the possibility of finding feasible paths for navigating in environments where otherwise no solution exists. It is still necessary to determine if an obstacle is indeed immovable or difficult to manipulate, but by testing and estimating the manipulability of these objects, the robot can determine which obstacles are movable and which are not, as well as to what degree movable obstacles are movable. There exists inherent uncertainty in determining an object's manipulability, but through careful estimation we can assign a cost and relative certainty to future attempts at manipulating the obstacle. The robot can then apply this knowledge to future encounters with similar obstacles, enabling it to navigate successfully while saving time. Through this approach, with no prior knowledge of its environment, the robot can find feasible paths through cluttered rooms and hallways and in doing so successfully navigate in situations that would typically prove problematic.

In this paper we present an algorithm for estimating manipulability of obstacles in unknown indoor environments, allowing a mobile robotics platform with no initial knowledge of the environment to successfully navigate through a room or hallway filled with various objects. The algorithm incorporates an online, self-supervised learning procedure to record visual features of encountered obstacles and identify them again in the future. We use a 2-D discretized model of the environment where each grid cell has associated with it a real-valued manipulation cost. These manipulation costs are represented in the robot's internal knowledge by independent, one-dimensional Gaussian random variables. Samples of the true manipulation cost are calculated from measurements of the robot's inverse speed and are used to update estimates of manipulability of cells in front of the robot. An appropriate planner, such as A*, is then used to calculate a minimal-cost path through the costmap. The purpose of this implementation is to optimize the time it takes for the robot to reach the goal. Our contribution is an algorithm for indoor mobile robots that minimizes the expected time to goal while accounting for uncertainty in the costs associated with manipulating obstacles as well as applying learned costs to future encountered obstacles.

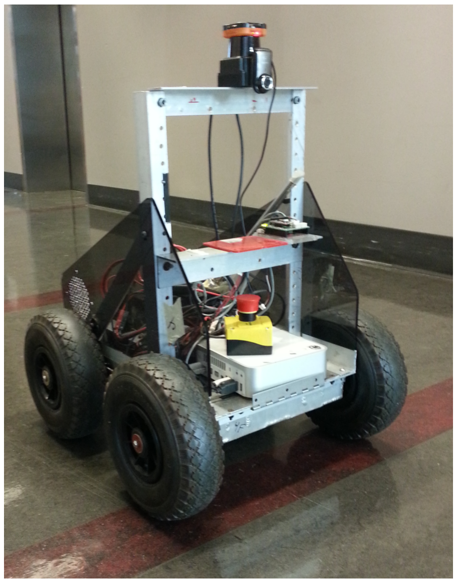
Fig. 1: The University of Pennsylvania MAGIC 2010 competition robot.

II. RELATED WORK

Our manipulability estimation approach draws techniques from several different areas of robotics research. One of these areas is traversability estimation in outdoor robotics. [1] and [2] use self-supervised or unsupervised online visual learning algorithms to detect different types of terrain. This

[12]C. Clingerman and D. D. Lee are with the GRASP Lab at the University of Pennsylvania, 3330 Walnut Street, Philadelphia, PA 19104, USA `chcl -at- seas.upenn.edu`, `ddlee -at- seas.upenn.edu`

allows the robot to associate learned visual features with traversability estimates. [3] and [4] use Gaussian Processes to model probabilistic costmaps for outdoor navigation planning. [3] uses these models to classify visual information from overhead imagery as different types of terrain. [4] goes one step further and uses a Gaussian Process regressor to associate overhead imagery with slip estimates from the robot. [5] discusses how these probabilistic costmaps can be used to precompute probabilistic heuristics for search-based planning. The advantage of doing so is a trade-off between the risk of suboptimality and lower planning times.

Like [1] and [2], and unlike [3], [4], [5], our approach continually forms and updates its understanding of the environment as the robot explores. We are not given overhead maps but must rather use a robot-mounted forward-facing camera to gain visual information about the environment. However, unlike [1], our algorithm uses real-valued costs and does not simply label the environment with binary values of “yes” or “no” when asked about manipulability. Also, given that our approach is focused on indoor environments, we are more interested in the obstacles presented rather than in modeling the ground surface [2].

In [6] and [7] the authors use what are known as object affordances, which are different attributes of objects that describe possible actions that can be performed on them. The authors use probabilistic graphical models to represent mappings between sensor data and visual object categories, from which they can determine likely object affordances. The experiments in [6] were performed using a manually driven robot as well as with manual segmentation of images, while our approach involves no human interaction. The authors in [7] determine through their experiments that their attribute model did not perform better than much simpler models when it came to the task of affordance prediction for novel object classes.

While our algorithm does not solve the Navigation Among Movable Obstacles (NAMO) problem, it is important to recognize this distinct area of research, which involves planning both navigation and manipulation that will allow the robot to reach a goal location in the presence of movable obstacles. Most papers only approach the NAMO problem using complete (or mostly complete) information about the environment or its obstacles [8], [9]. [10] presents NAMO algorithms that can plan given initially unknown environments and partial object information, but these algorithms rely on the assumption that all obstacles are axis-aligned rectangles. [11] implements a NAMO algorithm based on MDPs that considers uncertainty in object dynamics. The robot's knowledge of these obstacles' dynamics is then updated when the robot attempts to move them. However, similarly to [10], [11] only presents results in a 2-D simulator, whereas our implementation runs successfully on a real robot. [12] implements a NAMO algorithm on a humanoid robotics platform in unknown environments where the robot is given no previous knowledge of the objects in its environment, but, unlike our algorithm, the authors predetermined the specific direction and distance of manipulation for a given test object and only consider objects as manipulable or not manipulable.

III. Modeling Uncertainty in the Costmap

A. Costmap as a Gaussian Process

As in [4] and [3], the probabilistic costmap can be modeled using a multi-dimensional Gaussian Process regressor, where cell costs are jointly estimated based on the robot's perception of its environment. In our implementation, the robot learns about its environment through a combination of pushing and visual identification of objects. The formulation for a Gaussian Process assumes that observed data is modeled by an unknown latent function of feature vectors with the addition of noise. These feature vectors and observed values are used as training data. If the function has a Gaussian Process prior, then the probability of the test values, conditioned on their corresponding feature vectors, is a Gaussian distribution with a mean vector of the dimension of the test set and a full covariance matrix.

To simplify our implementation, as well as to reduce computation time during each iteration of our algorithm, we instead choose to model each cell in the costmap with individual one-dimensional Gaussian random variables. Our model is essentially a Gaussian Process with a diagonal covariance matrix. Although we do not explicitly model any correlation between cells, the robot's visual updates (assigning costs to objects that look similar to ones previously seen, based on similarity between visual feature vectors) does in a sense provide some correlation in the costmap.

B. Modeling Single-Cell Cost Uncertainty

As previously mentioned, the manipulation cost of a cell in the costmap can be modeled using a one-dimensional Gaussian random variable. This random variable represents the cumulative knowledge of prior observations made of the true cost, which is hidden from the robot. Following a derivation similar to that seen in [13], let c_t, the cost of a cell at time t, be the hidden variable of an unobserved Markov process, and let y_t be the real-world observation of that process. We are observing a system where $c_{t+1} = f(c_t)$ and $y_t = g(c_t)$. More specifically, assume

$$c_{t+1} = c_t + \epsilon_t \tag{1}$$

$$y_t = c_t + \eta_t \tag{2}$$

where ϵ_t is process noise and η_t is observation noise. Both noise variables are zero-mean Gaussian distributed. At each time step t, we want to estimate $P(c_t|y_1, y_2, \ldots, y_t)$, the probability of the cost c_t given all observations up until time t. If Q is the variance of the process noise, R is the variance of the observation noise, $\hat{c}_t$ is our estimate of the cost at time t, and P_t is the variance of the estimated distribution for the cost at time t, we also know

$$\begin{aligned} P(c_{t+1} \mid c_t) &= \mathcal{N}(c_t, Q) \\ P(y_t \mid c_t) &= \mathcal{N}(c_t, R) \\ P(c_{t-1} \mid y_1, y_2, \ldots, y_{t-1}) &= \mathcal{N}(\hat{c}_{t-1}, P_{t-1}) \end{aligned}$$

1) Cost Prediction and Estimate Update: The previous definitions lead us to the standard filter "Predict" and "Update" steps. In the "Predict" step, we evaluate $P(c_t \mid y_1, y_2, \ldots, y_{t-1})$, which, given the Markov assumption, can be computed as

$$P(c_t \mid y_1, y_2, \ldots, y_{t-1}) = \mathcal{N}(\hat{c}_{t-1}, P_{t-1} + Q). \quad (3)$$

The "Update" step is now

$$P(c_t \mid y_1, y_2, \ldots, y_t) = \frac{P(y_t \mid c_t) P(c_t \mid y_1, y_2, \ldots, y_{t-1})}{P(y_t \mid y_1, y_2, \ldots, y_{t-1})}, \quad (4)$$

where

$$P(y_t \mid y_1, y_2, \ldots, y_{t-1}) = \mathcal{N}(\hat{c}_{t-1}, P_{t-1} + Q + R) \quad (5)$$

is a normalization term.

Now we need to compute the left-hand side of Equation 4. Using Theorem 2 in [13], the term $P(y_t \mid c_t)P(c_t \mid y_1, y_2, \ldots, y_{t-1}) = \mathcal{N}(c_t, R) * \mathcal{N}(\hat{c}_{t-1}, P_{t-1} + Q)$ can be reformulated as the product of two Gaussians, one that contains the term c_t and another that does not. This results in

$$\begin{aligned} P(y_t \mid c_t) P(c_t \mid y_1, y_2, \ldots, y_{t-1}) &= \\ = \mathcal{N}\left(\hat{c}_t, P_t\right) * P(y_t \mid y_1, y_2, \ldots, y_{t-1}), \quad (6) \end{aligned}$$

where

$$\begin{aligned} \hat{c}_t &= \frac{R\hat{c}_{t-1} + (P_{t-1} + Q) y_t}{R + P_{t-1} + Q} \\ P_t &= \frac{R(P_{t-1} + Q)}{R + P_{t-1} + Q}. \end{aligned}$$

This new Gaussian distribution is a Gaussian over c_t. Since $P(y_t \mid y_1, y_2, \ldots, y_{t-1})$ is the normalization term, $P(c_t \mid y_1, y_2, \ldots, y_t)$ becomes simply

$$P(c_t \mid y_1, y_2, \ldots, y_t) = \mathcal{N}\left(\hat{c}_t, P_t\right), \quad (7)$$

which is our new estimate for the manipulation cost and its corresponding uncertainty.

2) Analysis of Cost Convergence: Now, let us look at Equation 7 in more detail. Given enough samples, we want our cost estimate to converge. Using the above definition for P_t, if we set $P_t = P_{t-1}$ we derive a quadratic equation in P_{t-1}. The positive solution to this quadratic is

$$P_{t-1} = \frac{-Q + \sqrt{Q^2 + 4RQ}}{2}.$$

Therefore, this value of P_{t-1} is the value at which P_t will be equal to P_{t-1}. In practice, the value of P_t will converge to this value as t goes to infinity. With enough manipulation cost samples, we should see P_t converge.

Now let us consider the cost estimate $\hat{c}_t$. If we set $\hat{c}_t = \hat{c}_{t-1}$, we derive a linear equation that simplifies to $\hat{c}_{t-1} = y_t$. Therefore, in order for $\hat{c}_t$ to equal $\hat{c}_{t-1}$, we need our estimate of the manipulation cost to match the observations from the environment. This is the necessary condition for the cost estimate to converge, which should be satisfied by using a simple filter.

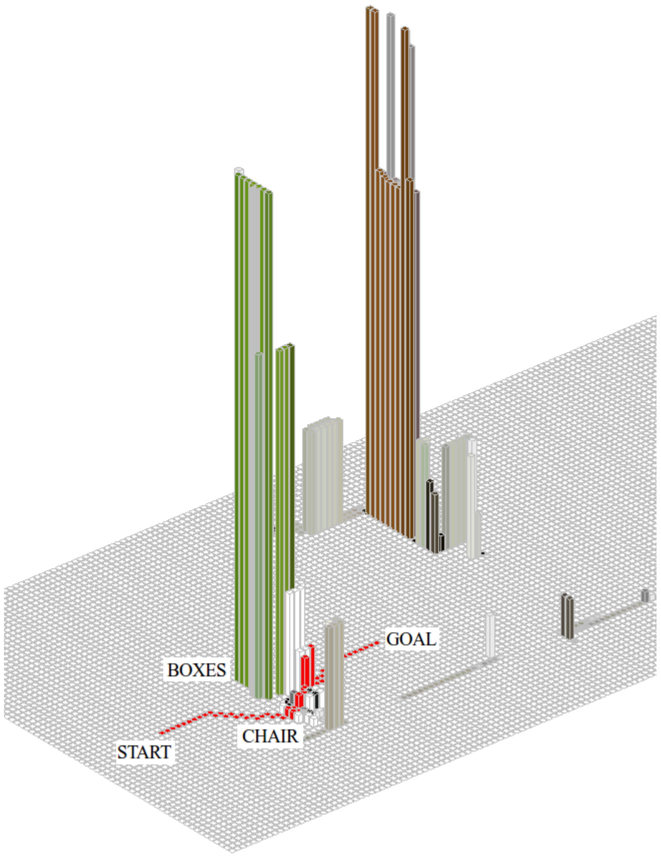

Fig. 2: Fused cost and color from the robot's costmap after a trial. Costs are shown on a logarithmic scale. Red cells indicate the robot's actual path.

C. Planning with Uncertainty

Traditional navigation planning using a discretized costmap makes use of graph search algorithms such as A*. Rather than utilizing our probabilistic cell costs to derive a probabilistic heuristic as in [5], we instead choose to form a deterministic costmap derived from the distributions modeled for each cell (similar to [3] and [4]). For instance, a simple algorithm could take $E[c_t]$ at time t (i.e. the mean $\hat{c}_t$) and do a graph search over those values. A deterministic heuristic, such as the often-used Euclidean distance metric, may not provide the fastest search, but it is still admissible in this case. This is the basis of our "agnostic" algorithm, so named because it is neither optimistic nor pessimistic about the costs. It simply uses the best estimate we can provide. While this algorithm will suit the purposes of most applications, it ignores a significant part of our uncertainty model: the cell cost variance. Therefore we create a second algorithm that uses this information to provide a more optimistic estimate of the cell costs.

The second algorithm we implemented deviates from the approaches discussed in Section II by calculating costs based on the idea of an Upper Confidence Bound [14] estimate. The idea behind the Upper Confidence Bound comes from the multi-arm bandit problem. The multi-arm bandit presents several choices for a potential reward. However, the person playing the game does not know *a priori* the reward values or probabilities of receiving the rewards. At every round, it is up to the player to choose which arm to pull, with the objective being to maximize the player's cumulative reward. Given limited information, it is potentially in the player's best interest to not always pull the arm with the highest estimated reward, as there may be an arm that offers an even higher reward. Therefore, it is beneficial to sometimes "explore" the bandit by pulling arms whose rewards are currently uncertain, rather than always "exploit" the arm with the highest believed reward. In our case, it is not always optimal to take the path with the lowest cost but to explore areas of the map with high uncertainty in the hope of finding a path with even lower cost.

Since we are calculating the cost of a path rather than a cumulative reward, we instead use the "lower confidence bound," which we define by

$$\mathrm{LCB}(i,j) = \hat{c}_t(i,j) - \sqrt{\frac{P_t(i,j)}{\log n}}. \tag{8}$$

The value n is the number of times a cell's cost has been estimated, either by pushing or visually. In the case where $n \in \{0,1\}$, the term $\log n$ is replaced with $\log 2$. Using the lower confidence bound instead of the mean for cell costs leads to a more optimistic or exploratory navigation plan, which we believe will be beneficial in terms of learning the association between environment perception and manipulation costs.

IV. IMPLEMENTATION

Our implementation utilizes several simplifying assumptions. To optimally navigate an environment, we want to minimize the time it takes for the robot to reach the goal position. Initially assuming all obstacles are manipulable by being pushed by the robot in its current direction of movement (unless otherwise estimated), the planned path minimizing execution time is initially a straight line from the start position to the goal position. This assumption is suitable for navigating through offices or hallways, and any navigation goal requiring turning a corner can be executed either by learning that the wall is an immovable obstacle or by splitting the navigation goal into multiple sub-goals. Also, if an obstacle can (or cannot) be pushed in a given direction, we assume that it can (or cannot) be pushed in any direction. Such an assumption is realistic for objects not placed against immovable obstacles such as walls or closed doors. If placed in the middle of a room, an office chair with wheels will freely move in any direction, whereas a heavy box will not move in any direction. Finally, we do not anticipate situations where one possibly pushable obstacle is made immovable by being pushed against an immovable obstacle. The robot will eventually think the movable obstacle is immovable and will associate a high manipulation cost with similar obstacles in the future. Our implementation focuses on environmentally based estimates of manipulability and does not segment or discriminate between multiple objects.

A. Environment Model

We model the robot's environment using a discretized costmap. Each grid cell $\vec{g}_t$ in the costmap is represented by

$$\vec{g}_t(i,j) = \Big[c_t(i,j) \sim \mathcal{N}\left(\hat{c}_t(i,j), P_t(i,j)\right), \vec{h}_t(i,j)\Big] \tag{9}$$

where $c_t(i,j)$ is the manipulation cost associated with the grid cell $\vec{g}_t(i,j)$ located at cell index (i,j) in the costmap, and $\vec{h}_t(i,j)$ is a 64-element vector histogram (4x4x4 3-D histogram) containing counts of color samples taken of the objects inhabiting the grid cell. (While all of these values are written with respect to time t, not every grid cell $\vec{g}_t(i,j)$ is updated for every time step t.) Thus each grid cell in the costmap has not only a cost estimate of the manipulability of the potential obstacles occupying this cell but also a color histogram representing the visual features of those obstacles. Updates to the distribution of the cost $c_t(i,j)$ are estimated using a simple Kalman filter (which was derived in accordance with the Bayesian estimation and analysis in Section III-B), where $\hat{c}_0$ and P_0 are the same for every cell (see Algorithm 1). We initially assume that every grid cell is unoccupied and easily traversable, so the value of $\hat{c}_0$ is 1. $c_t(i,j)$ can be sampled from the distribution associated with $\vec{g}_t(i,j)$ (assuming negative costs are not allowed), but as described in Section III-C we use either $E[c_t(i,j)] = \hat{c}_t(i,j)$ or $\mathrm{LCB}(i,j)$ in our implementation. The manipulation costs are used with the ARA* planning algorithm [15] to minimize the time it takes for the robot to reach the goal.

```
Result: Cost distribution at $\vec{g}_t(i,j)$ is updated with a
        new manipulability estimate.
Update($\hat{c}_t(i,j)$,$P_t(i,j)$,$y_t$):
begin
    //Predict:
    $\hat{c}_t = \hat{c}_t(i,j)$;
    $P_t = P_t(i,j) + Q$;

    //Update:
    $\tilde{y}_t = y_t - \hat{c}_t$;
    $S_t = P_t + R$;
    $K_t = P_t / S_t$;
    $\hat{c}_{t+1} = \hat{c}_t + K_t * \tilde{y}_t$;
    $P_{t+1} = (1 - K_t) * P_t$;

    //Set new values:
    $\hat{c}_{t+1}(i,j) = \hat{c}_{t+1}$;
    $P_{t+1}(i,j) = P_{t+1}$;
end
```

Algorithm 1: Updating the costmap with a cost sample

The robot uses a forward-facing webcam to gather color samples from objects in its way. Laser hits that are within 10.0 meters of the LIDAR sensor, 3-D points represented by (x_l, y_l, z_l), are first converted into the camera's reference frame and then mapped to pixel coordinates (x_c, y_c) in the sampled image. This is done through equations derived from the camera projection matrix P, specifically

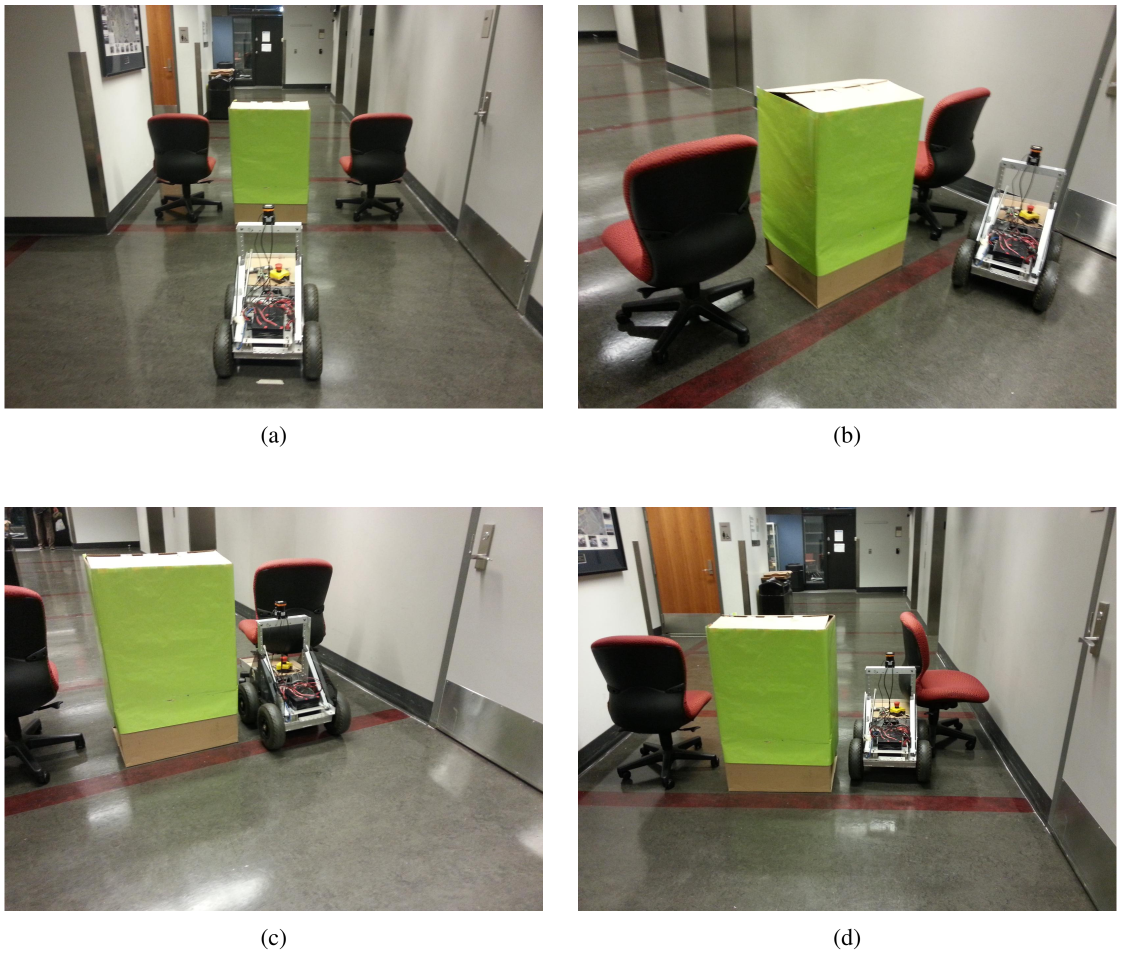

Fig. 3: The robot during a typical trial.

$$x_c = \frac{f'_x * x_l + c'_x * z_l}{z_l} \tag{10}$$

$$y_c = \frac{f'_y * y_l + c'_y * z_l}{z_l} \tag{11}$$

where f'_x is the rectified focal length in the x-dimension, f'_y is the rectified focal length in the y-dimension, and (c'_x, c'_y) is the center of the rectified image. The image coordinates (x_c, y_c) are used to find pixel samples in the original RGB color channels of the camera. We convert the RGB sample into the HSV color space and add the sample to the histogram of the costmap cell directly beneath the sampled pixel.

B. Cost Calculation

Since our robot attempts to manipulate obstacles by pushing them, our algorithm attempts to estimate manipulation cost by measuring the robot's inverse speed through its environment. Cost samples are calculated by the ratio between actual measured inverse speed and an idealized commanded inverse speed, determined by the robot's uninhibited forward velocity. A SLAM algorithm is used in addition to programmed pauses in motion to ensure accurate pose estimates. Manipulation cost is calculated by

$$y_t = \frac{v^{-1}_{\text{actual}}}{v^{-1}_{\text{commanded}}} = \frac{\Delta t_a / \Delta d_a}{\Delta t_c / \Delta d_c} \tag{12}$$

However, if we enforce that the robot takes roughly the same amount of time to execute a motion goal as the commanded ideal time $\Delta t_c \approx \Delta t_a$, Equation 12 simplifies to

$$y_t = \frac{\Delta d_c}{\Delta d_a} \tag{13}$$

Therefore, our manipulation cost calculation simply depends on the ratio between distance commanded and the actual distance the robot was able to travel (within a reasonable amount of time.) Cost samples are generated after a cumulative commanded distance of at least 0.2 meter, and the robot applies this cost update to a small radial window immediately in front of the robot. Cells detected by the LIDAR sensor to be obstacle cells receive the update as calculated, whereas cells detected as free cells receive an update of $y_t = 1.0$, the lowest possible cost. The obstacle cells that receive the update are further labeled as "training cells" and are placed in a separate list to be used in the "learning" phase of the algorithm.

C. Learning

The formulation described in previous sections allows us to determine an association between visual information of environmental obstacles and estimates of manipulability where those obstacles exist. Any cell that contains an obstacle and receives a cost update as the result of the robot making a manipulability estimation is added to a list of grid cells used for training a nearest-neighbor classifier. After each motion command is executed by the robot, we iterate over every grid cell in the costmap and find its nearest neighbor in the training list using the standard Euclidean distance metric. If the nearest neighbor has a distance less than a specified threshold, then the "knowledge" of the cost of that training cell is applied to this test cell. In this way the robot can plan ahead based on the visual information about obstacles in its environment and apply knowledge to objects that are similar in appearance to ones encountered before. The training grid cells are also evaluated in this list of test cells, but they should always find themselves as their own nearest neighbor (or another grid cell with the exact same histogram.) See Algorithm 2.

```
Result: Manipulation cost knowledge is applied to cells
        with color histogram data.
foreach $(i, j) \in \text{Costmap}$ do
    $d_{\text{min}} = \infty$;
    $(i_{\text{min}}, j_{\text{min}}) = \emptyset$;
    foreach $(i', j') \in \text{TrainingCells}$ do
        $d = \|\vec{h}_t(i, j) - \vec{h}_t(i', j')\|_2$;
        if $d < d_{\text{min}}$ then
            $(i_{\text{min}}, j_{\text{min}}) = (i', j')$;
            $d_{\text{min}} = d$;
        end
    end
    if $d_{\text{min}} < d_{\text{thresh}}$ then
        $c_{\text{sample}} = \hat{c}_t(i_{\text{min}}, j_{\text{min}})$;
        $\{\hat{c}_{t+1}(i, j), P_{t+1}(i, j)\} = \ldots$
                $\text{Update}(\hat{c}_t(i, j), P_t(i, j), c_{\text{sample}})$;
    end
end
```

Algorithm 2: Applying manipulation cost knowledge to the costmap, where "Update()" is detailed in Algorithm 1

The distance threshold that is used to consider if a cell's nearest neighbor is in fact "close enough" can be adjusted to suit the needs of the application. A lower value will cause the robot to be more selective about spreading cost information among cells and require that visual features match closely. A higher value will promote the sharing of cost estimates among obstacles that may appear somewhat different from each other. By setting the value lower, the robot will be more likely to try to push obstacles and ignore prior information, whereas by setting the value higher, it will be more likely to use knowledge discovered from previous estimations.

V. RESULTS

A. Platform

The robot we use in our experiments is a custom platform first developed at the University of Pennsylvania for the international MAGIC 2010 competition (Figure 1.) It is a four-wheeled skid-steer robot with a horizontal LIDAR sensor and a forward-facing webcam. Odometry is provided by wheel encoders and an IMU sensor, and the robot's pose is determined by a ROS SLAM library that uses the odometry estimation as an initial guess. The robot also has a static bumper on its front in order to push against obstacles and prevent them from blocking the sensors. All computation is performed on an onboard dual-core Mac Mini, and we communicate with the onboard computer using a wireless connection to the robot's wireless access point.

B. Experiments

Our implementation was evaluated in an experiment involving a heavy stack of boxes and two light, wheeled office chairs. The boxes were filled with enough weight to make them practically impossible for the robot to move, while the chairs were normal office chairs with no modifications. The chairs were placed facing away from the robot so it was easier for the robot to detect that it was pushing against them, and the boxes were wrapped with green paper to assist the robot in recognizing them. Figure 3 illustrates the configuration of the experiment. While a SLAM algorithm was used for localization, odometry and SLAM pose estimates were reset at the beginning of each trial.

Each experiment took place in two trials, a "learning" trial and a "test" trial. In the learning trial, the robot was given no prior knowledge of the box or chairs and was directed to navigate to a goal point 4 meters straight ahead. The robot then plans an optimal path from start to goal given no prior knowledge of the environment or obstacles, which in this experiment is a straight line. The robot travels along this path until it pushes against the stack of boxes. After attempting to move on a straight-line path through the boxes, the robot detects that its manipulability for the last commanded motion was low and subsequently assigns a high cost to the obstacle cells in front of it. As mentioned in Section IV-B, this cost is derived from the manipulability estimate. The high cost for cells along the formerly optimal path forces the robot to plan around the stack of boxes through either of the two chairs. The robot attempts to navigate through the space occupied by one of the chairs and associates high manipulability/low cost with these cells. After the robot pushes through the space formerly occupied by the chair, it continues to the goal. In the test trial, the robot is given the knowledge of color histograms and associated costs that it generated in the learning trial (but no map.) The robot is presented with the same configuration: a heavy stack of boxes between two light chairs. The robot detects the color of the green paper on the boxes, assigns a high cost with the cells the boxes occupy, and chooses to navigate through the area occupied by one of the chairs. Finally, the robot continues to the goal position.

TABLE I: Average values for experiment statistics.

	Mean Costs, Learn	Mean Costs, Test	LCB Costs, Learn	LCB Costs, Test
Execution Time (s)	236.93±193.26	192.04±186.65	294.89±156.49	176.82±56.80
Travelled Distance (m)	5.54±1.93	5.01±1.40	5.17±0.59	4.52±0.23
Final Optimal Path Cost ($\sum \hat{c}_f$)	52.63±1.89	58.77±15.35	54.58±2.72	55.19±3.31
Final Optimal Path Var. ($\sum P_f$)	578.23±40.45	744.39±291.64	721.30±71.01	691.76±94.65
Actual Path Cost ($\sum \hat{c}_f$)	71.85±33.42	78.60±51.56	73.28±18.36	67.30±6.94
Actual Path Var. ($\sum P_f$)	688.57±246.33	674.90±190.58	684.13±120.35	610.95±36.87
Regret (Eq. 14)	19.22±31.68	19.83±36.32	18.70±15.81	12.11±8.21

This experiment was designed to test the basic functionality of our algorithm. The experiment was designed so that the robot learns that the stacks of boxes are heavy while the office chairs are light. The robot is given the opportunity to navigate through grid cells occupied by the objects and estimate their manipulability. The experiment then forces the robot to apply its knowledge in the same scenario, except this time the robot should not need to push against the box to know it is heavy. Figure 2 shows both the cost and color of cells in the robot's costmap after a trial run in the experiment setup.

While this experiment was used to test our algorithm's functionality, it also provided a way to compare results of paths generated using the cell means as costs versus using LCB estimates. The literature for confidence bounds and multi-arm bandits presents the idea of regret, which is defined as the difference between choosing the best reward (or lowest cost) at each time step versus the choices that are actually made. In the case of navigation planning for mobile robots, we consider the regret to be the difference in path cost between the optimal path given the robot's knowledge of its environment versus the path the robot actually took. More formally, the regret function we use is

$$\text{Regret}(P) = \sum_{(i,j)\in P} E[c_f(i,j)] - \sum_{(i',j')\in P'} E[c_f(i',j')], \tag{14}$$

where P' represents the optimal path and P represents the robot's actual path. $E[c_f(i,j)]$ is our estimate of the costs when the robot has reached the goal. Here, we sum up our estimates of the manipulation costs $\hat{c}_f$ at the end of the trial for both the actual path the robot took and for the optimal path from the starting position. Given the knowledge gained from the trial, we want to see what the optimal path is over the learned costmap. By comparing the total manipulation cost of the optimal path with the path the robot actually took, we gain a measure of how the robot performed. Obviously, the regret in all cases should be a positive value, as by definition the robot cannot provide a better path than the optimal path. By using this metric in our evaluation, we are able to determine which version of the algorithm provides more complete and correct information about manipulation costs in the environment as well as which version deviates less from the optimal path in execution.

In the analysis of our algorithm we ran five experiments (five learning/test trial pairs) using the means as costs and five experiments using LCB values as costs. In all but one of the test trials for the means experiments and one of the learning trials for the LCB experiments the robot was able to autonomously navigate to the goal. In the unsuccessful trials the robot became stuck on the side of the boxes and misassociated a high cost for the black back of the chairs. In these instances, after a significant amount of time, the robot was manually driven through the troublesome high-cost region and allowed to autonomously continue to the goal. In all but one test trial for the LCB experiments the robot was able to successfully identify the stack of boxes and associate it with a high manipulability cost. In the unsuccessful trial, the robot pushed against the boxes and relearned that the boxes were heavy before navigating to the goal. This failure can most likely be attributed to moderate differences between the histograms detected during the test trial and those acquired during the learning trial.

Table I compares statistics generated by our experiment for the two variations of cell cost functions. Path means and variances were computed by summing up the values of $\hat{c}_f(i,j)$ and $P_f(i,j)$ for all cells (i,j) that comprised the given path. As can be seen by the high standard deviations for "Actual Path Cost" and "Regret" values, one or two experiments with poor performance moderately skewed the average values, even though at first glance it may appear that the Test experiments for LCB handily outperformed the mean Test experiments, at least based on regret. These outliers were included in the calculation of the averages as these cases indicated failures in the overall performance of the particular algorithm being used. The only set of experiments without a significant outlier (and a significant failure) was the Test case using LCB. The LCB algorithm took slightly longer in the learning phase to assign appropriate costs to cells, which most likely gave it an advantage of having better learned histograms and more accurate costs. When incorporating all of the collected data, we see that the LCB algorithm slightly outperformed the mean algorithm in the Test case. Figure 4 compares the average regret calculated for each experiment category.

VI. FUTURE WORK

The next step for validating our algorithm and comparing different cell cost functions is to test in more complex scenarios. The preliminary experiment is fairly simple, and we hope to demonstrate the ability to navigate more difficult environments than the one presented here. As in [4] and [3], we can also model our costmap as a Gaussian Process with full covariance matrices. The learning phase of our algorithm

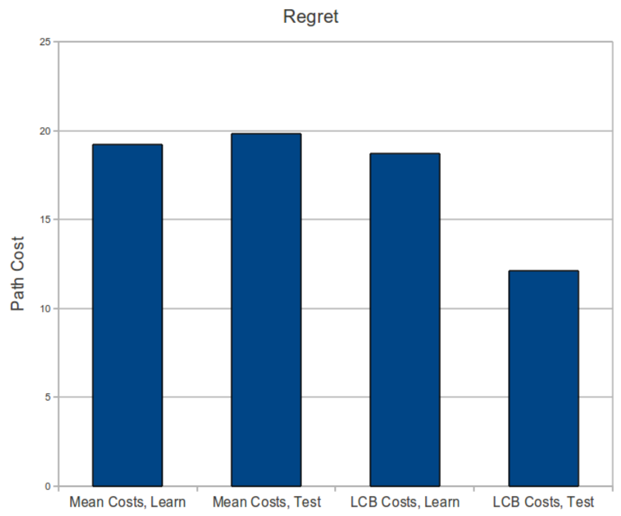

Fig. 4: A visualization of the regret calculated from our experiments.

might take more time, but the additional representational power of Gaussian Processes could possibly lead to better cost estimation results. Additionally, we plan to replace our simple color histograms with more sophisticated visual features. The features produced via a SIFT filter will most likely suit our needs for obstacle identification much better than color alone. Finally, with the addition of a small robot arm or other considerations, we can allow the robot to manipulate objects in more complicated ways than simply pushing.

VII. CONCLUSION

Through the application of manipulability estimation of obstacles, we have implemented and tested an algorithm for autonomous mobile robot navigation in unknown indoor environments. Our algorithm allows a mobile robot to successfully assess the manipulability of objects in its surroundings and properly model the uncertainty that naturally arises from such estimations. We formulated our approach, implemented it on a physical robot, and verified its functionality in a simple yet informative experiment. While our approach has proven successful so far, we will continue to test our algorithm in real-world experiments and further verify its significance.

REFERENCES

[1] D. Kim, J. Sun, S. M. Oh, J. Rehg, and A. Bobick, “Traversability classification using unsupervised on-line visual learning for outdoor robot navigation,” in *IEEE International Conference on Robotics and Automation (ICRA 2006)*, 2006, pp. 518–525.

[2] S. Zhou, J. Xi, M. W. McDaniel, T. Nishihata, P. Salesses, and K. Iagnemma, “Self-supervised learning to visually detect terrain surfaces for autonomous robots operating in forested terrain,” *J. Field Robot.*, vol. 29, no. 2, pp. 277–297, Mar. 2012.

[3] L. Murphy and P. Newman, “Planning most-likely paths from overhead imagery,” in *IEEE International Conference on Robotics and Automation (ICRA 2010)*, 2010, pp. 3059–3064.

[4] L. Murphy, S. Martin, and P. Corke, “Creating and using probabilistic costmaps from vehicle experience,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2012)*, 2012, pp. 4689–4694.

[5] L. Murphy and P. Newman, “Risky planning: Path planning over costmaps with a probabilistically bounded speed-accuracy tradeoff,” in *IEEE International Conference on Robotics and Automation (ICRA 2011)*, 2011, pp. 3727–3732.

[6] J. Sun, J. L. Moore, A. Bobick, and J. M. Rehg, “Learning visual object categories for robot affordance prediction,” *The International Journal of Robotics Research*, vol. 29, no. 2-3, pp. 174–197, 2010.

[7] T. Hermans, J. M. Rehg, and A. Bobick, “Affordance Prediction via Learned Object Attributes,” in *IEEE International Conference on Robotics and Automation (ICRA 2011): Workshop on Semantic Perception, Mapping, and Exploration*, May 2011.

[8] M. Stilman and J. Kuffner, “Navigation among movable obstacles: Real-time reasoning in complex environments,” in *IEEE/RAS International Conference on Humanoid Robotics (Humanoids 2004)*, 2004, pp. 322–341.

[9] M. Stilman, K. Nishiwaki, S. Kagami, and J. Kuffner, “Planning and executing navigation among movable obstacles,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, 2006, pp. 820–826.

[10] H.-N. Wu, M. Levihn, and M. Stilman, “Navigation among movable obstacles in unknown environments,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, 2010, pp. 1433–1438.

[11] M. Levihn, J. Scholz, and M. Stilman, “Planning with movable obstacles in continuous environments with uncertain dynamics,” in *IEEE International Conference on Robotics and Automation (ICRA 2013)*, 2013, pp. 3832–3838.

[12] Y. Kakiuchi, R. Ueda, K. Kobayashi, K. Okada, and M. Inaba, “Working with movable obstacles using on-line environment perception reconstruction using active sensing and color range sensor,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, 2010, pp. 1696 –1701.

[13] A. L. Barker, D. E. Brown, and W. N. Martin, “Bayesian estimation and the kalman filter,” *Computers Math. Applic*, vol. 30, pp. 55–77, 1994.

[14] P. Auer, “Using confidence bounds for exploitation-exploration tradeoffs,” *J. Mach. Learn. Res.*, vol. 3, pp. 397–422, Mar. 2003.

[15] M. Likhachev, G. Gordon, and S. Thrun, “Ara*: Anytime a* with provable bounds on sub-optimality,” in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 16: PROCEEDINGS OF THE 2003 CONFERENCE (NIPS 2003)*. MIT Press, 2004.