Adaptation for Active Perception
Robustess from adaptation to context

Andreas Hofmann, Paul Robertson

Dynamic Object Language Labs Inc.
Lexington, Massachusetts, 02421
Email: {andreas, paulr}@dollabs.com

***Abstract*—Existing machine perception systems are brittle and inflexible, and therefore cannot adapt well to environment uncertainty. Natural perception always occurs in support of an activity that provides context. In our approach, we use context to adapt the perceptual aperatus providing robustness and resilience to noise among other benefits. Evidence supports the view that context driven adaptation occurs in natural perception systems including human vision. This *Active Perception* approach prioritizes the system's overall goals, so that perception and situation awareness are well integrated with actions to focus all efforts on these goals in an optimal manner. We use a Partially Observable Markov Decision Process (POMDP) framework, but do not attempt to compute a comprehensive control policy, as this is intractible for practical problems. Instead, we employ *Belief State Planning* to compute point solutions from an initial state to a goal state set. This approach automatically adapts the perception data flow processes, and generates action sequences for sensing operations that reduce uncertainty in the belief state, and ultimately achieve the goal state set. Our early results described in this paper demonstrate the feasibility of this approach using a restriced set of actions.**

Keywords: Active Perception, POMDP, Belief State Planning, Context, Adaptation.

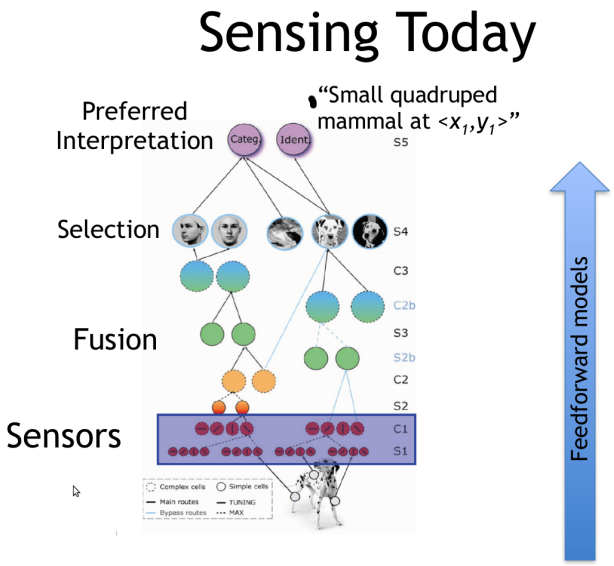


Figure 1. Sensing today is static and largely bottom-up.

I. Introduction

Natural perception systems such as the human visual system operate robustly in a wide variety of situations. Machine vision systems on the other hand have very little flexibility and tend to be very brittle in the face of environmental changes. The human visual system adapts with changing lighting conditions and with changing tasks. Adaptation is the key to this robustness. Machine vision systems work well when the environmental conditions and the task remain fixed but for robots to perceive their environment robustly for a variety of tasks and in ever changing conditions, requires an adaptive approach.

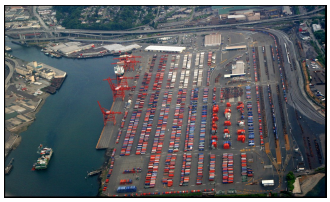
In traditional machine vision systems information flow is bottom up, and generally not guided by knowledge of higher level context and goals, as shown in Figure 1.

If higher level goals, context, or the environment change, the specific conditions for which the static configuration is intended may no longer hold. As a result, the static systems are prone to error because they cannot adapt to the new conditions. With few exceptions, such as [1], [2], they do not reason intelligently about dynamically changing goals, contexts, and conditions, and therefore, are not able to change to a more appropriate process flow configuration, or to change their parameter settings in an intelligent way.

In addition to their inflexibility, existing machine perception systems are often not well integrated into the autonomous and semi-autonomous systems to which they provide information. As a result, they are unaware of the autonomous system's overall goals, and therefore, cannot make intelligent observation prioritization decisions in support of these goals. In particular, it may not be necessary or useful for the perception system to be aware of every aspect of a situation, and it may be detrimental, due to resource contention and time limits, to the overall goal. Examples of autonomous systems operating in dynamic and uncertain environments, and that depend on intelligent perception, are shown in Figure 2. In a cyber attack scenario (sub-figure a), the overall goal of an autonomous system intended to provide security is to defend against the attack. Understanding the details of the attack is a subgoal. In a seaport operation scenario (sub-figure b), the overall goal of an autonomous control system is to operate the seaport safely

and efficiently. Understanding seaport activities is a subgoal. In an auto repair scenario (sub-figure c), the overall goal is to repair the problem. Diagnosing the problem is a subgoal. In all three cases, it may not be necessary to understand every detail of what is happening in order to accomplish the overall goal.

(a) Cyber attack.



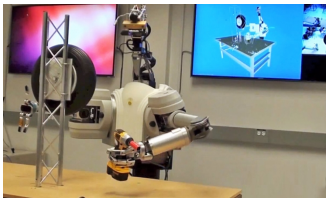
(b) Seaport.

(c) Auto repair.

Figure 2. Three example scenarios with dynamic environments.

Consider the automobile repair problem, or more specifically, the problem of changing a tire. This has been a "textbook" problem for Planning Domain Definition Language (PDDL) generative planning systems [3], and also a challenge problem in the Defense Advanced Research Projects Agency (DARPA) ARM project [4]. There are significant challenges in building an autonomous system that can perform this task entirely, or even one that would just assist with the task (Figure 3). Intelligent machine perception would be needed for both a fully autonomous system (sub-figure a), and a semi-autonomous advisory system (sub-figure b). The latter might include observation drones, and Google glasses [5] to guide a human user in the repair. Such a system would have to be able to determine the vehicle type (Figure 4), whether a tire is actually flat (Figure 5), and what an appropriate sequence of repair steps should be (Figure 6). It would have to be able to solve many subproblems, such as reliably finding a wheel in an image. The system would have to be able to work in many different environments, a great range of lighting conditions, and for a comprehensive set of vehicle types.

We address this challenging problem by using a more dynamic approach in which reasoning about context is used to actively and effectively allocate and focus sensing and



(a) Fully autonomous system using robot.

(b) Semi-autonomous advisory system.

Figure 3. Semi-autonomous and fully-autonomous systems.

(a) Truck

(b) SUV

(c) Sedan

Figure 4. Vehicle type, and ultimately, make and model, are useful things for the perception system to know (or to discover).

(a) Tire is flat.

(b) Tire is under-inflated.

(c) Tire is OK.

Figure 5. The perception system must be able to determine whether a tire is really flat, and which one it is.

(a) Wheel chock.

(b) Remove wheel.

Figure 6. Repair steps include stabilizing the car (a), getting the spare tire and tools, jacking up the car, removing lug nuts and wheel (b), and installing the new wheel.

action resources. This *Active Perception* approach prioritizes the system's overall goals, so that perception and situation awareness are well integrated with actions to focus all efforts on these goals in an optimal manner. Active perception draws on models to inform context-dependent tuning of sensors, direct sensors towards phenomena of greatest interest, to follow up initial alerts from cheap, inaccurate sensors with targeted use of expensive, accurate sensors, and to intelligently combine results from sensors with context information to produce increasingly accurate results (Figure 7). The model-based approach deploys sensors to build structured interpretations of situations to meet mission-centered decision making requirements. Actions are calibrated so that they are proportional to the likelihood of true detection and degree of threat.

We consider, in detail, a subproblem of the tire change problem: reliably finding the wheels of a vehicle in an image. We show how the use of top-down, model-based reasoning

 ISBN: 978-1-61208-391-9

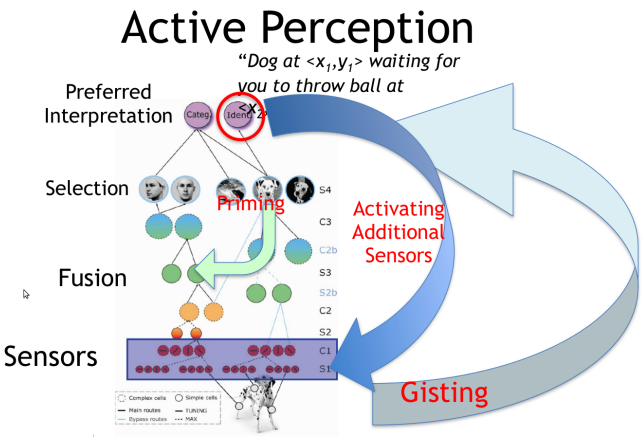


Figure 7. Active Perception uses model-based top-down reasoning, as well as bottom-up computation, to guide sensing actions.

can be used to coordinate the efforts of multiple perception algorithms, resulting in more robust, accurate performance than is achievable through use of individual algorithms operating in a bottom-up way. We also show how our model-based framework can be used to address the entire tire change problem.

The rest of this paper is organized as follows. In Section II, we present informal and formal statements of the problem to be solved. Section III covers related work, and Section IV introduces our approach. Sections V, VI, and VII describe the details of the approach, and Section VIII presents results. We conclude with a discussion in Section IX.

II. PROBLEM STATEMENT

Given one or more *agents* operating in an environment, and given that the agents do not directly know the state of the environment, or even, possibly, parts of their own state, and given a goal state for the environment and agents, the problem to be solved is to compute control actions for the agents such that the goal is achieved. In this case, an agent is a resource capable of changing the environment (and its own) state, by taking action. An agent could be a mobile ground robot, a sensing device, or one of many parallel vision processing algorithms running on a cluster, for example. Given that there is uncertainty in the environment state, because it cannot be measured directly, an agent must estimate this state as best as is possible based on (possibly noisy) observations. Similarly, part of the agent's state, such as the functional status of its components, may not be directly measurable, and must be estimated. Based on the agent's best estimate of the current environment state (and its own state), it should take actions that affect the state in a beneficial way (move the state towards the goal). The actions, themselves have some uncertainty; they do not always achieve the intended effect on the state. The agents must take both state estimate uncertainty, and action uncertainty into account when determining the best course of action. Actions can also have cost. The agents must balance the cost of actions against the reward of reduced uncertainty and progress towards the goal when deciding on actions.

This problem presents significant challenges. First, the overall state space, including environment and agent state, can be very large; there may be many state variables in the representation. Second, the state space is generally hybrid; it includes discrete variables, such as hypotheses for vehicle type, as well as continuous variables, such as position of a wheel. Third, significant parts of the state space may not be directly measurable, and must be estimated based on observations. The observations (usually sensor data) typically have some noise. Therefore, the estimates will have some uncertainty. As a result, the state variables themselves are generally not represented by a single value, but rather, by a probability distribution. These distributions must be updated as part of each state estimation cycle. Fourth, effect of some actions on state may have uncertainty. Fifth, the agents must take many considerations into account when deciding on actions: they must take into account the uncertainty of the state estimate, the uncertainty of the action effect on the state, the cost of the action, and the benefit of the action in terms of reducing uncertainty and making progress towards the goal.

The problem to be solved can be stated formally as follows. Suppose that the state space is represented by $S = \{S_e, S_a\}$, where S_e is the state space associated with the environment, and, S_a, that associated with the agent. Suppose, further, that the agent can perform a set of actions, A, and make a set of observations, O. A state transition model represents state evolution as a function of current state and action: $T : S \times A \times S \mapsto [0,1]$. An observation model represents likelihood of an observation as a function of action and current state: $\Omega : O \times A \times S \mapsto [0,1]$. A reward model represents reward associated with state evolution: $R : S \times A \times S \mapsto \mathbb{R}$. Given an initial state s_0 and goal states s_g, the problem to be solved is to compute an action sequence $a_0, \ldots a_n$ such that $s_n \in s_g$, and R is maximized. The agent bases its action decisions on its estimates of the state, which in turn, is based on the observations. The state estimates must be sufficiently accurate to support good action decisions. Note that this problem formulation, expressed in terms of a single agent, is easily extended to allow for multiple agents.

III. BACKGROUND AND RELATED WORK

A *Partially Observable Markov Decision Process* (POMDP) [6] is a useful framework for formulating problems for autonomous systems where there is uncertainty both in the sensing and in the results of actions taken. A POMDP is a tuple $\langle S, A, O, T, R, \Omega, \gamma \rangle$ where S is a (finite, discrete) set of states, A is a (finite) set of actions, O is a (finite) set of observations, $T : S \times A \times S \mapsto [0,1]$ is the transition model (conditional probabilities for state transitions), $R : S \times A \times S \mapsto \mathbb{R}$ is the reward function associated with the transition function, $\Omega : O \times A \times S \mapsto [0,1]$ is the observation model (conditional probabilities for state given observation and action), and γ is the discount factor on the reward. The *belief state* is a

 ISBN: 978-1-61208-391-9

probability distribution over the state variables, and is updated each control time increment using recursive predictor/corrector equations. First, the *a priori* belief state for the next time increment, $\bar{b}(s_{k+1})$, is based on the *a posteriori* belief state for current time increment.

$$\begin{aligned}\bar{b}(s_{k+1}) &= \sum_{s_k} Pr(s_{k+1}|s_k,a_k)\hat{b}(s_k) \\ &= \sum_{s_k} T(s_k,a_k,s_{k+1})\hat{b}(s_k)\end{aligned} \tag{1}$$

Next, the a posteriori belief state for the next time increment, $\hat{b}(s_{k+1})$, is based on the a priori belief state for next time increment.

$$\begin{aligned}\hat{b}(s_{k+1}) &= \alpha\, Pr(o_{k+1}|s_{k+1})\bar{b}(s_{k+1}) \\ &= \alpha\, \Omega(o_{k+1},s_{k+1})\bar{b}(s_{k+1})\end{aligned} \tag{2}$$

$$\alpha = \frac{1}{Pr(o_{k+1}|o_{1:k},a_{1:k})} \tag{3}$$

These equations work well given that the models are known, and given that the control policy for selecting an action based on current belief state is known. Unfortunately, computing these, particularly the control policy, is a challenging problem. Value iteration [7] is a technique that computes a comprehensive control policy, but it only works for very small problems. An alternative is to abandon computation of a comprehensive control policy, and instead, compute point solutions for a particular initial state and goal state set.

A promising technique for this is *Belief State Planning* [8], which is based on generative planner technology [9]. A key idea in this technique is the use of two basic actions for a robotic agent: move and look. A move action changes the state of the robot and/or environment; it may move the robot in the environment, for example. A look action is intended to improve the robot's situational awareness.

The move action is specified using a PDDL-like description language.

```
Move(lstart, ltarget)
  effect: BLoc(ltarget, eps)
  precondition: BLoc(lstart, moveRegress(eps))
  cost:  1
```

The variables lstart and ltarget denote the initial and final locations of the robot. The effect clause specifies conditions that will result from performing this operation, if the conditions in the precondition clause are true before it is executed. Cost of the action is specified in the cost clause.

The function BLoc(loc, eps) returns the belief that the robot is at location loc with probability at least (1 - eps). The moveRegress function determines the minimum confidence required in the location of the robot on the previous step, in order to guarantee confidence eps in its location at the resulting step:

$$moveRegress(eps) = \frac{eps - p_{fail}}{1 - p_{fail}} \tag{4}$$

where p_{fail} is the probability that the move action will fail.

The look action is specified as follows:

```
Look(ltarget)
  effect: BLoc(ltarget, eps)
  precondition: BLoc(lstart,
                     lookPosRegress(eps))
  cost:  1 - log(posObsProb
                     (lookPosRegress(eps)))
```

The lookPosRegress function takes a value eps and returns a value eps' such that, if the robot is believed with probability at least 1 eps' to be in the target location before a look operation is executed and the operation is successful in detecting ltarget, then it will be believed to be in that location with probability at least 1 eps afterwards.

$$lookPosRegress(eps) = \frac{eps(1-p_{fn})}{eps(1-p_{fn}) + p_{fp}(1-eps)} \tag{5}$$

where p_{fn} and p_{fp} are the false negative and false positive observation probabilities.

In terms of the POMDP belief state update, the move action corresponds to the predictor (1), and the look action corresponds to the corrector (2). It would also be possible to have actions that include both components. In our work, because we are currently focused on the observation aspect, our actions follow the approach of the look action, utilizing the observation model, but not a transition model. Also, the look action presented here implies independence of observations. It supports building confidence by repeatedly applying the look action (by staring). This makes sense in some contexts, but not in ones where a static image is being analyzed by an operator that does not change. Therefore, in our work, we cannot assume independence and handle the belief state update in an alternative manner.

For the actual observation algorithms, we make extensive use of two types of feature detection algorithms: *Speeded Up Robust Features* (SURF) [10], and *Hough Transforms* [11]. We utilize the Mathworks Computer Vision System Toolbox [12] implementation of these algorithms. Neither of these algorithms, used individually, is satisfactory for solving the wheel detection problem robustly. However, when used together, in an Active Perception framework, they beneficially reinforce each others hypotheses, allowing for more reliable performance.

 ISBN: 978-1-61208-391-9

IV. APPROACH

In order to achieve the Active Perception capabilities described in the Introduction, we define a process for Active Perception, as shown in Figure 8. The high-level *Meta-control* component maintains the system's current goals, and manages hypothesis formation and refinement. Hypotheses are maintained in a prioritize pool, and are used to guide *Sensor Fusion* and *Differential Diagnosis and Interpretation* components. These components provide the Meta-control component with an awareness of the current situation by strenghthening or weakening hypotheses, based on evidence from the sensors. The Meta-control component uses this awareness, combined with goals to be achieved, to task sensors in an optimal way, such that the goals will be achieved with greatest likelihood and efficiency.

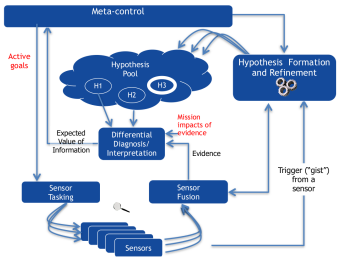


Figure 8. The Active Perception Process uses hypothesis formation and refinement to guide tasking of sensors.

In order to make this more concrete, it is useful to consider what a data flow diagram for the sensors and state estimation components should look like (if a computer vision expert were to solve the specific problem of designing an architecture for finding wheels in an image). Figure 9 shows a network of sensing components that collaborate to achieve the goal of finding wheels in an image with a high level of certainty. Sensing algorithms include feature matching, wheel location prediction, and Hough ellipse detectors. Each such sensing algorithm can use the current belief state as input, and can also adjust belief state as output. Sensing actions perform sensing operations by setting parameters for the sensing algorithms, running the algorithms, and interpreting the results. In particular, the sensing algorithms produce observations, which are used to update belief state.

This organization based on actions, observations, and belief state fits well with the POMDP formalism. Unfortunately, although POMDP's are a standard method of formulating this type of problem, solution of the POMDP can be challenging. In particular, value iteration approaches [7] that attempt to compute a comprehensive control policy are intractible for all but the simplest problems.

A promising alternative is to abandon the attempt to compute a comprehensive control policy, and instead, compute point solutions for a particular initial state and goal state set. This requires significant runtime computation, but is generally far more tractable than value iteration for this type of problem. This approach automatically synthesizes data flow processes such as the one in Figure 9, and generates action sequences for sensing operations that reduce uncertainty in the belief state, and ultimately achieve the goal state set.

We use three main sensing actions: SURF Match, SURF Match Other Wheel, and Hough Ellipse Match. Each of these actions has preconditions (requirements for current belief state), and post conditions (effects on belief state). SURF Match uses the SURF algorithm to perform a preliminary detection of a wheel in an image. SURF Match Other Wheel attempts to find the second wheel, also using SURF, given that the first wheel has been detected. This action also performs vehicle pose estimation, and refines the prediction of where the wheels are. Hough Ellipse Match uses either a Hough Circle or Hough Ellipse transform algorithm to refine the wheel location estimates.

The sequence of actions can be computed using a *generative planner*. A generative planner searches for action sequences that satisfy all the precondition and postcondition constraints, and that maximize the reward function. In order to be usable for this type of problem, two changes from traditional generative planning are necessary. First, traditional generative planners use deterministic operators, but the belief state representation is probabilistic. To solve this problem, we determinize the belief state operators used in the pre and post conditions of the actions [8]. This is accomplished by specifying thresholds for required belief. For example, a precondition for an action might require that a belief state variable be true with probability greater than 0.7. A postcondition might specify that a belief state variable should be true with probability of 0.8. The second change from traditional generative planning is in the overall framework in which planning and control interact. With traditional generative planning, a plan is generated, and is then executed in its entirety. This approach is not suitable for our type of problem, given the high levels of uncertainty, especially for the success of actions. Instead, we adopt a receding horizon control framework in which a plan is generated based on the current belief state, but only the first action of this plan is executed. After the action, the belief state is updated based on observations, and an entirely new plan is generated. The process then repeats with the first action from this new plan being executed. This approach is computationally intensive, but it ensures that all actions are based on the most current belief state.

To implement this receding horizon control framework, we use an architecture consisting of four main components, as shown in Figure 10. The executive manages the receding horizon control process. It maintains the current belief state, and the goal state set. At each control loop iteration, it passes these to the planner. The planner, if successful, returns a plan consisting of actions that transition the system from the current state to a goal state, if there are no disturbances. The executive takes the first action from the plan and executes it by dispatching the appropriate sensor operation. The sensor operation produces an observation, which is used to update the belief state. The process then repeats.

 ISBN: 978-1-61208-391-9

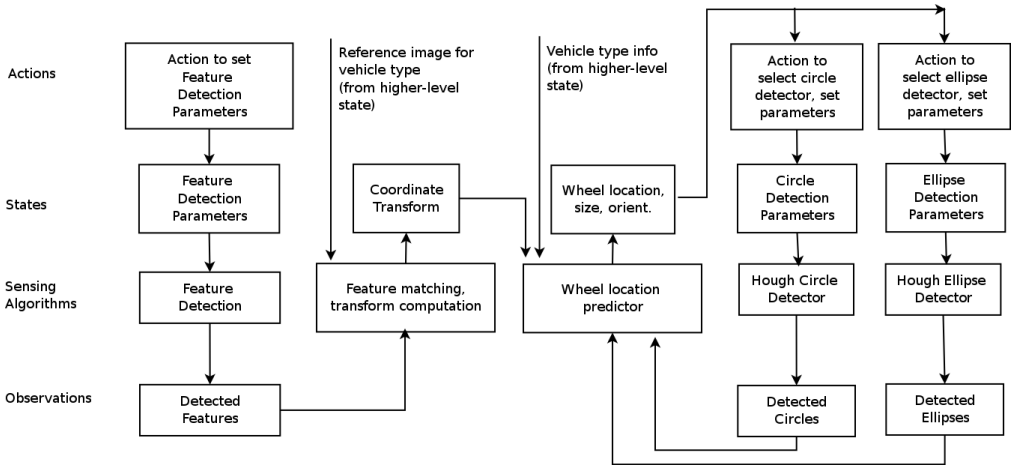


Figure 9. Data flow architecture for wheel finding components.

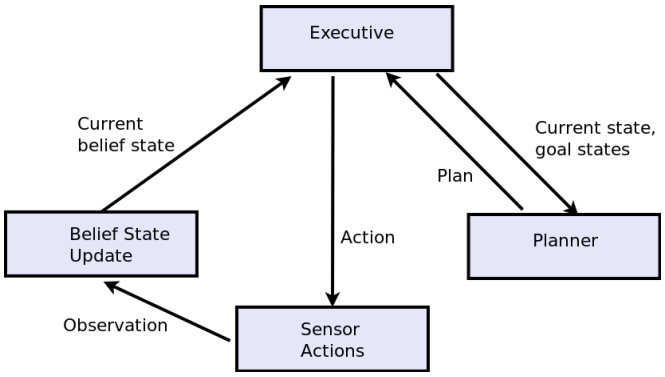


Figure 10. Receding Horizon Control Architecture for Active Perception

The following sections describe each of the components in this architecture in more detail.

V. EXECUTIVE

The Executive component implements the top-level receding horizon control loop, coordinating the activities of the planner and sensor components. Algorithm 1 shows pseudocode. The Executive accepts as input a goal state set, and an a priori belief state. It returns indicating success or failure in achieving a goal state.

The algorithm begins by initializing the belief state according to the a priori probabilities, and performing other initialization (Line 1). Belief state for a discrete state variable is represented as a *Probability Mass Function* (PMF) over the possible values of the probabilistic variable. Belief state for a continuous state variable is represented using a *Gaussian Probability Distribution Function* (Gaussian PDF) with a specified mean and variance. This could be extended to a representation using a mixture of Gaussians, in order to approximate more complex, non-Gaussian PDFs.

The receding horizon control loop begins at Line 5. The first step is to invoke the generative planner in order to determine the next action. A generative planner requires, as one of its inputs, the current state in a deterministic form. The belief state, however, is represented in a probabilistic form, so it first has to be converted to a deterministic form using MostLikelyState (Line 6). The input to the generative planner includes the determinized state, as well as the goal state set. This input is provided in the form of *domain* and *problem* PDDL files, as will be explained in more detail in Section VI. The Executive generates these files, and then executes the planner. The planner generates a result file containing a plan, which the Executive reads. The entire interaction of initializing the planner, running it, and retrieving the results is summarized in Line 7.

The planner may fail to generate a plan, in which case, the algorithm returns failure. If the planner is successful in generating a plan, the executive dispatches the first action (Line 11). The action (sensor operation) generates an observation. The belief state is updated based on this observation (Line 12).

At this point, the algorithm checks whether the goal has been achieved. (values of the state variables in the goal set have a sufficiently high belief). If not, the algorithm checks if the maximum number of allowed iterations have been exceeded (Line 15). If the maximum iterations haven't been exceeded, the algorithm begins a new control cycle, and the process repeats.

The Executive is implemented in Common Lisp. The Sensor Action and Belief State Update components are implemented in Matlab. The Executive communicates with these components using the C to Matlab API provided by Mathworks, combined with the Common Lisp to C foreign function interface provided by the Common Lisp implementation we use [13].

VI. PLANNER

The Planner component is implemented using *Fast Downward* [9], a state of the art generative planner that accepts problems formulated in the PDDL language [14]. A PDDL

 ISBN: 978-1-61208-391-9

Algorithm 2: Algorithm 1: Executive

Input: a-priori-belief-state-probabilities, goal-state-set
Output: goal-achieved?

```
                    /* Perform initialization.  */
1  belief-state ←
   InitializeBeliefState(a-priori-belief-state-probabilities);

2  goal-achieved? ← false;
3  max-iterations ← 1000;
4  iteration ← 0;
                          /* Begin control loop.  */
5  while not goal-achieved? do
6      current-state ← MostLikelyState(belief-state);
7      plan? ← GeneratePlan(current-state, goal-state-set);
8      if not plan? then
9          return ;
10     action ← First(plan?);
11     observation ← Dispatch(action);
12     belief-state ← UpdateBeliefState(observation);
13     goal-achieved?
       ← CheckGoalAchieved(belief-state, goal-state-set);
14     iteration ← iteration + 1;
15     if iteration > max-iterations then
16         return ;
```

problem formulation consists of a *domain* file, and a *problem* file. The domain file specifies types of actions that can be used across a domain of application such as logistics, manufacturing assembly, or in this case, finding wheels in an image. The domain file is fixed; it does not change for different problems within the domain. The problem file, on the other hand, contains problem-specific information such as initial and goal states. The Executive generates this file automatically. The following PDDL domain file fragment shows the definition of the SURF Match action in PDDL.

```
(:action SURF-match
  :parameters
    (?w ?p ?bsv)
  :precondition
    (and (belief-state-variable ?bsv)
         (pose ?p) (wheel ?w)
         (for-wheel ?w ?bsv)
         (at-pose ?p ?bsv)
         (at-belief-level
           belief-level-one ?bsv))
  :effect
    (and (at-belief-level
           belief-level-two ?bsv)
         (increase
           (total-cost)
           (feature-observation-cost ?p))))
```

The precondition clause specifies that the belief state variable value for a particular pose and wheel must be at belief level one in order for this operation to be tried. The effect clause specifies that the belief level increases to belief-level-two if the operation succeeds. The cost for the operation is also added to the total cost. The other sensor actions are specified in the PDDL domain file in a similar manner.

VII. SENSOR ACTIONS

We now describe in more detail the three sensor actions introduced previously: SURF Match, SURF Match Other Wheel, and Hough Ellipse Match.

A. SURF Match

The SURF Match action uses the SURF (Speeded Up Robust Features) algorithm [10] to attempt to identify wheels in the target image. SURF is inspired by SIFT (Scale-invariant Feature Transform), but is several times faster and can also be more robust against different image transformations. SURF is based on sums of 2D Haar wavelet responses and makes an efficient use of integral images.

The SURF Match action uses the SURF algorithm to attempt to match a wheel in a reference image with a wheel in the target image. The SURF algorithm is scale invariant, but is somewhat sensitive to large changes in orientation. Therefore, multiple reference images are used, including ones for different orientations. The orientations in the reference images (see Figure 11) correspond to the orientations of the discrete belief state variables.

(a) +2 (b) +1 (c) 0 (d) -1 (e) -2

Figure 11. Wheel reference images, corresponding to different orientations.

The SURF Match action is not highly reliable; it can miss detecting a wheel, especially if the target image is noisy, it can falsely detect objects that are not wheels. Also, due to the symmetry of wheel images, the SURF Match algorithm does not provide a very accurate estimate of the pose (position and orientation) of the wheel in the target image. Thus, the SURF Match action is used to attempt to achieve a rough initial match, the goal being to move from low to medium confidence estimates, and set the stage for the use of the other sensor actions to improve the estimates.

We used the functions detectSURFFeatures, extractFeatures, matchFeatures, and estimateGeometricTransform from the Mathworks Computer Vision System Toolbox [12] to implement the SURF Match action. Parameters used for these functions are shown in Tables I, II, and III.

 ISBN: 978-1-61208-391-9

TABLE I. PARAMETERS FOR detectSURFFeatures

MetricThreshold	NumOctaves	NumScaleLevels
100	5	5

TABLE II. PARAMETERS FOR matchFeatures

MatchThreshold	MaxRatio
30.0	0.7

In our implementation of this sensor action, we used fixed parameters for these functions, but this could be extended to allow for adaptively adjusting parameters within the active perception framework. For example, the MaxDistance parameter of the estimateGeometricTransformParameters can be beneficially tuned to improve performance. Besides estimating the transform, this function removes outlier matches. If the MaxDistance parameter is too small, it will remove too many points that are not outliers. Conversely, if the value is too large, too many outliers remain. We have set the value to 20.0 (up from its default setting of 1.5), and found this worked well in the tests we performed. However, there may be conditions that we haven't tested under which a lower setting is better.

B. SURF Match Other Wheel

The SURF Match Other Wheel sensor action is similar to the SURF Match action, but assumes that one wheel position estimate already exists (from a previous SURF Match action). It uses this information to try to find the other wheel. This action uses a number of gists. In this case, the gists come from previous observations or assumptions, external to the wheel finder system. In particular, it is assumed that the action is observing the left side of the car, and that the car is on level terrain.

The SURF Match Other Wheel action uses the existing wheel position estimate to crop out that part of the image. This forces the SURF algorithm to look elsewhere for the other wheel.

Additionally, if the SURF Match Other Wheel action is successful in finding the other wheel, then it uses the position estimates of both wheels to estimate the pose of the overall car. Note that the position estimates of the wheel resulting from the SURF algorithm are image position estimates. The sensor action uses projective geometry, combined with a number of additional gists, to estimate the positions of each wheel in the world coordinate frame, given the image position estimates. These gists are: 1) the height of the camera; 2) the focal length of the camera; and 3) the size of the wheel. All of these are reasonable gists; a ground robot or UAV would know the focal length of its camera, as well as its height. Given previous gists for vehicle type, the size of the wheel can be determined from the vehicle type's spec data.

Once the position estimates of each wheel in the world coordinate frame are known, simple trigonometry is used to determine orientation of the car, particularly, its yaw (rotation about the vertical axis). This estimate is the continuous counterpart to the discrete belief state variables for orientation. The continuous and discrete variables inform and reinforce each other as part of the belief state update mechanism.

TABLE III. PARAMETERS FOR estimateGeometricTransform

MaxNumTrials	MaxDistance
20000	20.0

C. Hough Ellipse Match

The Hough Ellipse Match sensor action uses Hough transforms [11] to determine wheel position in an image with a high degree of accuracy. This action is computationally expensive, but has the potential to give very accurate estimates, when supplied with good parameters. Thus, this action is used after the other, less expensive sensor actions have developed a strong hypothesis about where a wheel might be, and what it's orientation is likely to be.

The computational expense of the Hough transform algorithm rises as the number of parameters increases. The variation of the Hough transform for detecting lines is the cheapest; it requires only two parameters. The circle variation is more expensive since a circle is described by three parameters (the x, y position of the center, and the radius). The ellipse variation is still more expensive since an ellipse is described by six parameters. For this reason, the Hough Ellipse Match sensor action first checks whether estimated orientation (yaw angle) of the car is small, indicating that the car side is facing the camera directly. In this case, the sensor action employs the circle variation of the Hough transform, as a special case of an ellipse, since the circle variation is faster. Even so, this algorithm only works well if good bounds can be specified for the parameters (circle center and radius).

For the more general case, we use an efficient Hough Ellipse algorithm [15] that requires good bounds for major axis length, aspect ratio, and rotation angle. This algorithm can take a very long time (30 minutes or more in some cases, depending on image size), and is very sensitive to the parameter bounds. In particular, it can easily incorrectly match non-wheel objects as ellipses if its search area and parameters are not tightly constrained. For this reason, we crop the image based on the current estimated wheel position in the image, to reduce the area where the algorithm has to search. Additionally, we use orientation and scale information computed by the other sensor actions to constrain the major axis, aspect ratio, and rotation parameters.

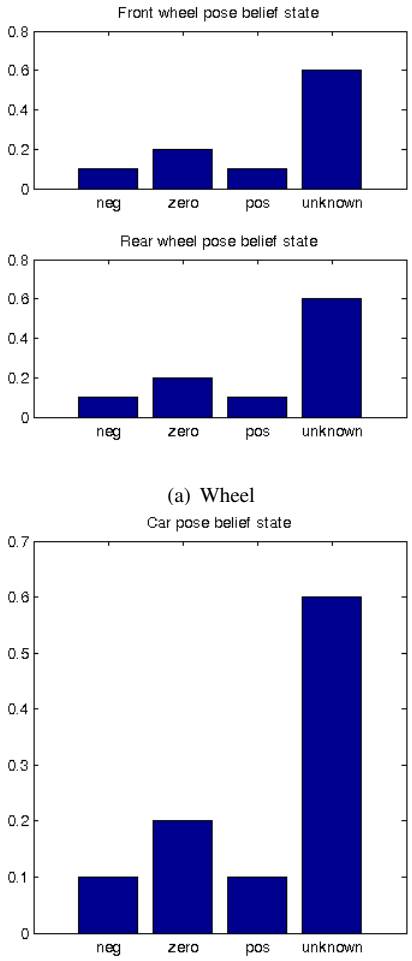
VIII. RESULTS

A. Results

In this section, we provide an end-to-end example run showing adaptation to the sensors as knowledge is learned from the image.

The a priori belief state is shown in Figure 12. This indicates that the poses are largely unknown, with a slight bias to the zero pose.

 ISBN: 978-1-61208-391-9



(a) Wheel

(b) Car

Figure 12. Wheel and car pose, a priori belief state.

The first control step iteration, based on this belief state, yields the following plan from the PDDL planner.

```
1. SURF Match (front-wheel pose-zero)
2. SURF Match Other Wheel (front-wheel pose-zero)
3. Hough Ellipse Match (rear-wheel pose-zero)
```

The Executive performs the first of these actions, yielding a successful match, as shown in Figure 13.

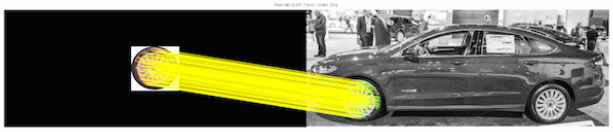


Figure 13. Successful SURF match to front wheel.

Based on this, wheel pose estimates are updated as shown in Figure 14; the hypothesis for zero pose for the front wheel has been strengthened over the a priori estimate shown in Figure 12.

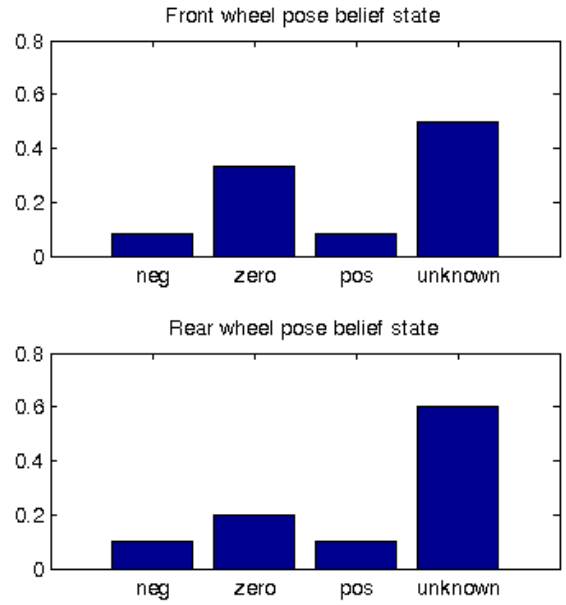


Figure 14. Wheel pose belief state after initial SURF match.

The second control step iteration, based on this updated belief state, yields the following plan from the PDDL planner.

```
1. SURF Match Other Wheel (front-wheel pose-zero)
2. Hough Ellipse Match (rear-wheel pose-zero)
```

The Executive performs the first of these actions, yielding a successful match, as shown in Figure 15. Note that the front wheel has been cropped out to focus the sensing action on the other (rear) wheel.

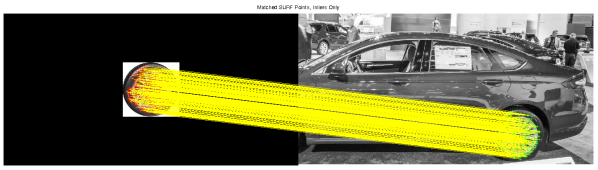


Figure 15. Successful SURF match to other (rear) wheel.

Based on this, wheel and car pose estimates are updated (see Figure 16), the hypotheses for zero pose for the wheels and car have been strengthened.

The third control step iteration, based on this updated belief state, yields the following plan from the PDDL planner.

```
1. Hough Ellipse Match (rear-wheel pose-zero)
```

The Executive performs the first of these actions, yielding a successful match, as shown in Figure 17.

The final wheel belief state estimates are shown in Figure 18.

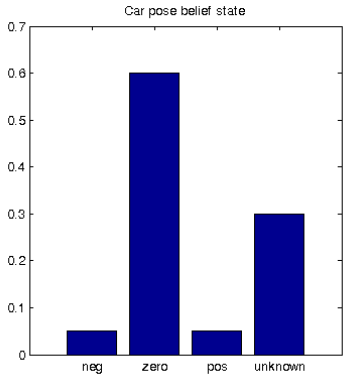


Figure 16. Car pose belief state, after SURF Match Other Wheel action.

Figure 17. Successful Hough ellipse match to rear wheel (match highlighted in green).

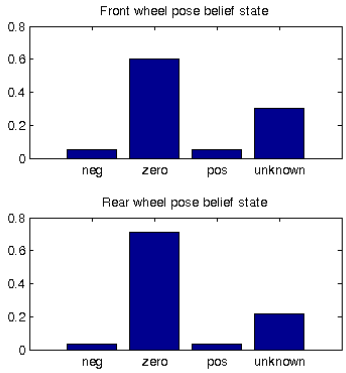


Figure 18. Final wheel pose belief state.

IX. DISCUSSION

The focus of our efforts thus far has been on the sub-problem of finding a wheel in an image. This has led to an emphasis on "look" actions, but no incorporation of "move" actions (actions that change the state of the environment). As stated previously, this is only part of the larger problem of autonomously or semi-autonomously repairing a car tire. We believe that the approach we have developed is well suited for extension to the larger problem, and problems like it. In particular, it is well suited to incorporating move as well as look actions, with the generative planning component intelligently combining both types of actions. This would allow for testing with more general kinds of problems, where the goal is more than purely a perception goal, but rather, involves changing the environment state to achieve an environment goal.

X. CONCLUSIONS

We have demonstrated the approach of using a PDDL planner with receding horzon planning to produce a context driven feature extraction capability. We are working now to integrate this capability within a framework that encompasses real actions that lead to a problem solving activity being performed (changing a tire) and in which many more observation actions are possible. To handle the increased state space size we are moving to a monte-carlo planner.

ACKNOWLEDGEMENTS

This research was developed with funding from DARPA. The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] G. Hoelzl, M. Kurz, and A. Ferscha, "Goal processing and semantic matchmaking in opportunistic activity and context recognition systems," in The 9th International Conference on Autonomic and Autonomous Systems (ICAS2013), March 24 - 29, Lisbon, Portugal, textbfBest Paper Award, March 2013, pp. 33–39.

[2] ——, "Goal oriented recognition of composed activities for reliable and adaptable intelligence systems," Journal of Ambient Intelligence and Humanized Computing (JAIHC), vol. 5, no. 3, July 2013, pp. 357–368.

[3] M. P. Fourman, "Propositional planning," in Proceedings of AIPS-00 Workshop on Model-Theoretic Approaches to Planning, 2000, pp. 10–17.

[4] P. Hebert et al., "Combined shape, appearance and silhouette for simultaneous manipulator and object tracking," in Robotics and Automation (ICRA), 2012 IEEE International Conference on. IEEE, 2012, pp. 2405–2412.

[5] N. Bilton, "Behind the google goggles, virtual reality," New York Times, vol. 22, 2012.

[6] G. E. Monahan, "State of the arta survey of partially observable markov decision processes: Theory, models, and algorithms," Management Science, vol. 28, no. 1, 1982, pp. 1–16.

[7] N. L. Zhang and W. Zhang, "Speeding up the convergence of value iteration in partially observable markov decision processes," arXiv preprint arXiv:1106.0251, 2011.

[8] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," The International Journal of Robotics Research, 2013, p. 0278364913484072.

[9] M. Helmert, "The fast downward planning system." J. Artif. Intell. Res.(JAIR), vol. 26, 2006, pp. 191–246.

[10] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," Computer vision and image understanding, vol. 110, no. 3, 2008, pp. 346–359.

[11] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," Communications of the ACM, vol. 15, no. 1, 1972, pp. 11–15.

[12] P. Corke, "Matlab toolboxes: robotics and vision for students and teachers," IEEE Robotics & automation magazine, vol. 14, no. 4, 2007, pp. 16–17.

[13] H. Sexton, "Foreign functions and common lisp," ACM SIGPLAN Lisp Pointers, vol. 1, no. 5, 1987, pp. 11–23.

[14] D. McDermott et al., "Pddl-the planning domain definition language," 1998.

[15] M. Simonovsky, "Ellipse detection using 1d hough transform," http://www.mathworks.com/matlabcentral/fileexchange/33970-ellipse-detection-using-1d-hough-transform retrieved: January, 2015.

 ISBN: 978-1-61208-391-9