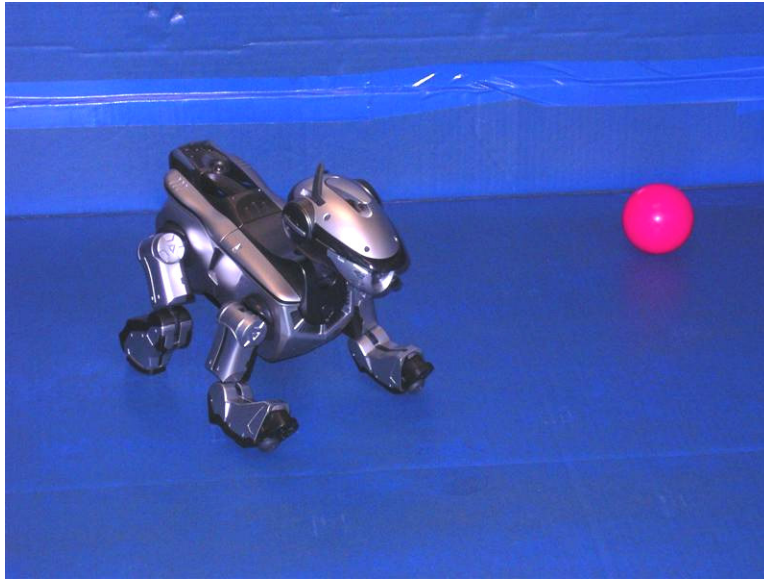


Sensory Graph Planning With a Physical Agent

Daniel T. Casner

Advisor: Kurt Krebsbach



Abstract

Planning for agents working in simulated worlds where sensing and actuation are perfect are well understood but planning for a physically embodied agent who has to deal with the imperfections of the real world is more difficult. This paper presents an implementation of deliberative intelligence for a physical agent in a domain with uncertain initial conditions imperfect sensing and actuation. Replanning when the world state does not match the planner's model is addressed.

Introduction

People have wanted to construct genuinely intelligent machines to accomplish tasks for countless years and science fiction writers have been generating ideas of what those machines might be like for almost as long. Robots being built today seem to be getting

closer and closer to this goal but still have many problems. They can now walk on two legs, manipulate objects, speak and understand human language but they are still far from having anything like human intelligence.

Two main philosophies of robot intelligence are in use today, reactive intelligence and high level planning. In the simplest model of reactive intelligence, the robot's intelligence consists of a set of functions which take input as arguments and produce outputs with perhaps limited memory at best. The only domain knowledge is that of the programmer who creates the functions. On the opposite end of the spectrum are completely deliberative robots which use domain knowledge to formulate a series of outputs to transform the current world state to some desired world state. Most actual robots aren't purely reactive or deliberative

but use a combination of techniques to achieve intelligent behavior.

Much research has been done in artificial intelligence planning with simulated agents that exist, plan and act in a simulated world and there are many well understood algorithms [4]. Comparatively little work has been done with actual robots that interact with the real world. Operating in the real world presents several challenges which simulated agents don't have to deal with such as imperfect sensing of the environment and uncertainty of actuation results. My work was on this problem of using high level planning with a physical robot and addressing some of the challenges related to it.

Background

The STRIPS Representation

Many current AI planning algorithms use the STRIPS assumption [4] to represent to state and actions. The STRIPS assumption is a way to simplify the representation of the world state and actions in the domain. We represent the domain as a conjunction of positive literals such as "robot is in the green room" or "alive." Anything which is not specified as true is assumed to be either false or something which we don't care about (at least at the present time). Actions specify a conjunction of literals which get added to the list along with the literals which are deleted from the list as a result of the action being taken. Because we reduce the representation of everything to either true or false / don't care, the domain representation becomes much simpler and by extension so does writing a planner to solve problems in the domain.

Sensory Graphplan

Sensory Graphplan (SGP) [1,6] is a complete and fairly efficient planner based on the earlier Graphplan algorithm [2]. It extends Graphplan with several features useful for a physical agent.

- First, uncertainty in the initial world state, this enables us to start the robot without giving it complete information about the initial world state and allow it determine necessary information as part of the plan.
- Second, sensing actions. Since the agent doesn't know what the initial state is, it needs some way to determine at least a minimum level of information about the world state to proceed. SGP allows actions to have among their effects not only additions and deletions from the STRIPS state representation, but also observation of the state of a previously uncertain predicate.
- Third, conditional effects of actions. That is, actions may have different effects depending on the state when the action was taken. For example in my domain there are two rooms, one green and one blue, taking the "go to other room" action obviously has one effect the robot starts in the green room and the reverse effect if it started in the blue room. The conditional affects of this particular action are even more complex than this and I will return to it later.

Because of these features SGP was chosen as the planning algorithm to use for this project.

For the sake of comparison with other planning algorithms which are used with robots, two features that SGP lacks are worth noting:

- There is no representation of the relative cost of actions in SGP. All actions effectively cost the same amount of time

and resources to enact. Further, even assuming equal cost, SGP does not guarantee an optimal plan (i.e. one that requires the minimum number of actions), it guarantees a plan which is complete given the uncertainty of the world (i.e. one which will result in the goal state when enacted regardless of which of the possible initial states we started in).

- We cannot parallelize planning and execution. Other planning algorithms such as decision theoretic planning [4] return a best next step which can be carried out immediately as planning for future steps continues. Using SGP there is no plan until the complete plan is generated. My domain was small enough that generating a complete plan took less than a minute¹ so generating a complete plan before acting was acceptable. Further, since a plan consist of a series of actions which lead from the start state to the goal state, we *ideally* only need to generate a plan once. In reality this is not usually the case but the trade off is still acceptable for this domain.

Details of the implementation of SGP beyond its interface exceed the scope of this paper and so are omitted. The operation of SGP is well described in a series of papers by Weld et. al. [1,2,6] and the reader is referred there for more information.

Domain

Time and resource constraints limited me to a very small domain but one which nevertheless contains a number of the challenges which are significant to creating

¹ Running the University of Washington implementation of SGP [1] in Allegro Common Lisp [3] on a 1.5GHz, 512Mb RAM laptop computer using the Windows XP operating system.

a deliberative robot. It consists of a green and blue room, a ball and the robot. The robot and ball are placed somewhere in the world and the robot must figure out how to reach the goal state of both the ball and itself being in the green room.

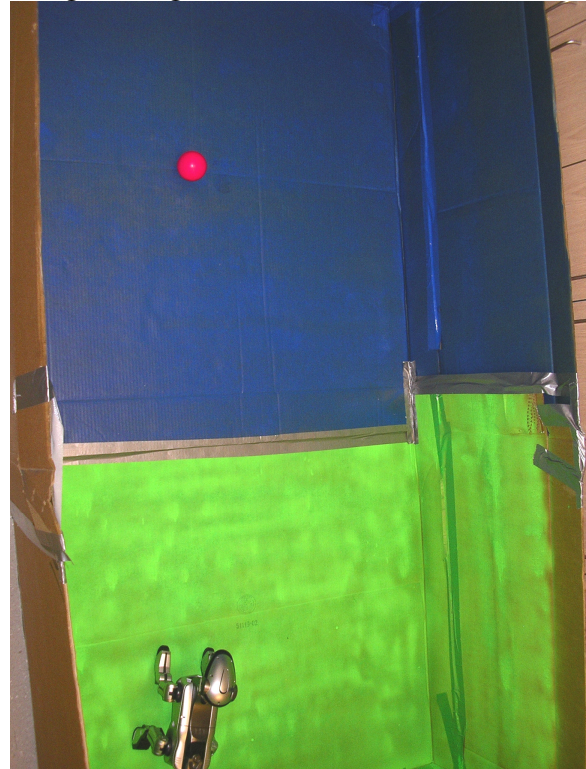


Figure 1: the domain, green and blue rooms, ball and robot.

Real World

In the real world the domain consists of a cardboard box 8 feet by 4 feet with 3 foot high walls. One half of the box is painted blue and the other half green. There is a single pink plastic ball in the box which the robot can push around. The robot itself is approximately a foot long, capable of sensing its surroundings and moving around the box. It is programmed with a set of actions which it performs as directed by the planner.

Representation

The domain is encoded in planning domain definition language (PDDL) [4] as a set of actions with preconditions and conditional

effects. This combined with a problem definition consisting of a set of possible initial states and a goal state is the input to SGP. PDDL is a convenient language to use because it easily represents the actions and domain using the STRIPS assumption in a human readable fashion as seen in figure 2.

Whereas in the real world the ball and robot have continuous positions and the robot has a heading, in the representation for the planner the state of the world is greatly simplified to 5 Boolean values.

- (robot) in green room
- ball in green room
- holding the ball
- facing the ball
- facing the destination

The last two values are inconsequential to the planning problem itself but are necessary to keep track of as preconditions for the grab-ball and go-destination actions respectively. These literals are deleted from the world state by every action which repositions the robot's head except those whose purpose is to add them to the world state because when the robot's head is repositioned for some other purpose, we can't be sure that we can still see the ball or destination anymore even if we could before.

The planner's domain knowledge is very specific. It knows what will happen if it tells the robot to take any of the actions under any of the possible conditions. On the other hand it doesn't know anything about what a room or a ball is. So while it can generate a valid plan to get to a goal state from an unknown start state, it can't make intuitive assumptions such as left the field of view to the left, it's probably somewhere to the robots left now.

```
(define (domain aibo-ball)
  (:requirements :conditional-effects :sensing :uncertainty)

  (:action check-room
    :effect (and (observes (in-green-rm))
                 (not (facing-ball))
                 (not (facing-dest))))
  )

  (:action check-holding
    :effect (and (observes (holding))
                 (not (facing-ball))
                 (not (facing-dest))))
  )

  (:action locate-ball
    :effect (and (not (facing-dest))
                 (when (not (holding))
                      (and (observes (ball-in-g-rm)) (facing-ball))))
  )

  (:action grab-ball
    :precondition (and (facing-ball) (not (holding)))
    :effect (and (holding)
                 (not (facing-ball))
                 (not (facing-dest))
                 (when (ball-in-g-rm) (in-green-rm))
                 (when (not (ball-in-g-rm))
                      (not (in-green-rm))))
  )

  (:action release-ball
    :effect (and (not (facing-dest))
                 (when (holding)
                      (and (not (holding)) (facing-ball)))
                 (when (not (holding)) (not (facing-ball))))
  )

  (:action face-dest
    :effect (and (facing-dest) (not (facing-ball)))
  )

  (:action go-dest
    :precondition (facing-dest)
    :effect (and (when (and (in-green-rm) (holding))
                   (and (not (in-green-rm))
                        (not (ball-in-g-rm))
                        (not (facing-dest))))
                 (when (and (in-green-rm) (not (holding)))
                   (and (not (in-green-rm))
                        (not (facing-dest))
                        (not (facing-ball))))
                 (when (and (not (in-green-rm)) (holding))
                   (and (in-green-rm)
                        (ball-in-g-rm)
                        (not (facing-dest))))
                 (when (and (not (in-green-rm)) (not (holding)))
                   (and (in-green-rm)
                        (not (facing-dest))
                        (not (facing-ball))))
  )
)
```

Figure 2: PDDL definition representation of the domain

Actions

Since all of the robot's behavior is carried out through its pallet of actions. Each action is programmed into the robot and started by a command from the planner. When the action finishes it either returns a finished flag, or if it was a sensing action, a true or false value depending on the result of the sensor.

1. *check-room*: is the first sensing action. It is not conditional and has no preconditions. Executing the action causes the robot to point its head at the floor to check if it is green or blue. It returns true if the floor is green and false if the floor is blue. As described above, because the robot's head is pointed downward by this action, it also deletes the *facing-ball* and *facing-dest* literals from world state since the robot's head is now no-longer looking at the ball or destination even if it was before.
2. *check-holding*: Like *check-room* it has no preconditions and is not conditional. Again it points the robot's head downward to see if the ball is between its front feet.
3. *locate-ball*: When called the robot starts turning in place scanning its head up and down looking for the ball. It stops turning when it sees the ball and points the head to center the ball in its field of view. True is returned if the ball is in the green room (as determined by its background) and false returned if it is in the blue room. The representation is partially conditional. In all cases the robot will not be looking at the destination. However, the action is only guaranteed to work if the robot is not already holding the ball so the other effects are contingent on that aspect of the world state. *locate-ball* also adds *facing-ball* to the world state since it causes the robot to face the ball at the end of the action.
4. *grab-ball*: The first action with a precondition, *grab-ball* requires that we be facing the ball to begin with (i.e. we can see it right now) and that we aren't already holding the ball. One post condition of this action is that the robot is holding the ball. The complexity come from the fact that the robot moves to the ball in the course of grabbing it so after the action is taken, which room the robot is in depends on which room the ball was in to begin with. Physically *grab-ball* makes the robot track toward the ball raising its head at the last moment, taking a couple more steps forward then stopping and lowering its head to capture the ball.
5. *release-ball*: after this action the robot is not holding the ball. It is conditional because if the robot was holding the ball before, it should now be facing it whereas if the robot wasn't holding the ball this action does not cause it to be facing the ball.
6. *face-dest*: when this action is executed the robot lifts its head and scans it's surroundings to find the room of the other color than it is in and returns the head is facing that room. This is the only action which adds *facing-dest* to the world state.
7. *go-dest*: finally the most complex conditional action. The only precondition is that the robot is facing the destination so it can track to it visually. The first condition rises from the fact that if the robot is in the green room, going to the other room will put it in the blue room and vice versa. The second conditional aspect is that if the robot is holding the ball when it moves, the ball moves with it.

Implementation

Hardware

I used a Sony ERS-220 AIBO robot for this project. For small robotics experiments like this the AIBO is an extremely convenient platform to work with. It is a quadruped about 9 inches tall and a little over a foot long. Processing is provided by a 384 MHz MIPS processor with 32MB of RAM. The robot's head contains a color camera and inferred range finder and can be pointed on pan tilt roll axis. Communication with a PC or other robots is accomplished via 802.11b wireless Ethernet. Programming for the AIBO is done in C++.

Due to the lack of a LISP interpreter for the AIBO, its lack of processing power and memory, the planner itself had to be run on a PC. Commands from the planner were then sent via wireless Ethernet to the robot which returned the results of the actions taken.

Tekkotsu framework

To speed up the development process I used the Tekkotsu development framework for AIBO robots [5] developed at Carnegie Mellon University. Tekkotsu provides midlevel functionality to the programmer. By using it, I started programming with functions on the level of walk forward at 10 centimeters per second and what pixel is the center of the ball at, instead of having to control each servo individually and develop the vision system from scratch.

Vision System

The vision system itself bears mentioning. Tekkotsu processes video on the ERS-220 at 25 frames a second through a chain of processing stages. Initially frames are raw camera data in YUV space as shown in the center image of figure 3. Vision processing in YUV space is easier than the RGB (red green blue, top image of figure 3) space

humans are used to working in because by ignoring the brightness channel (Y), a given color looks the same under any illumination level. YUV is also the native output of the AIBO's camera so it takes less processing to use.

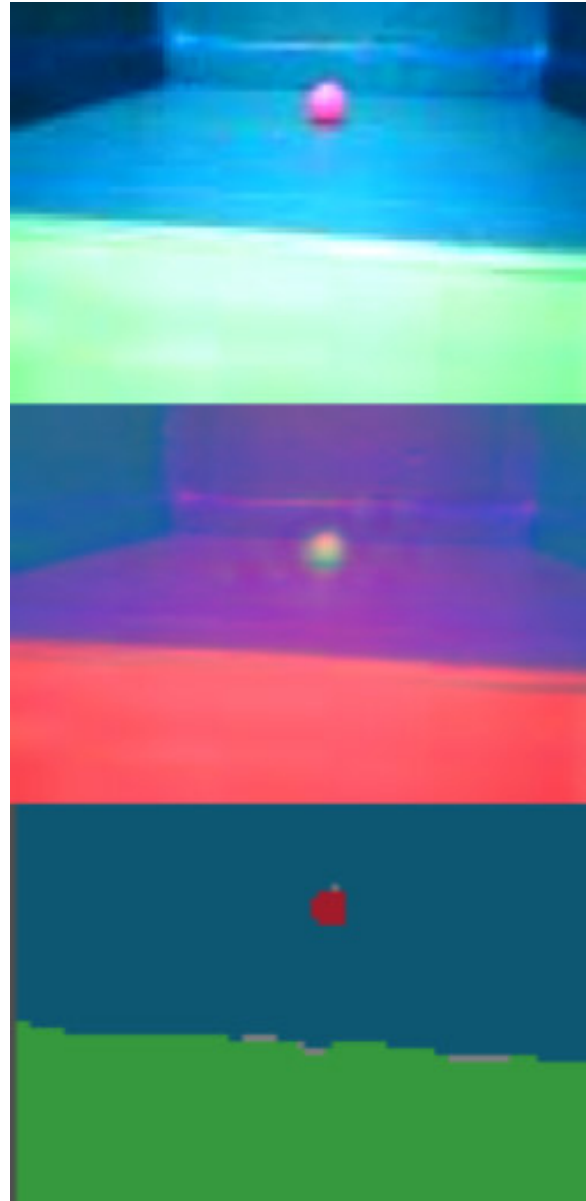


Figure 3: Tekkotsu vision processing chain. All three images are of the pink ball in the blue room as seen by the robot standing in the green room. At the top is a RGB color space camera image as humans are used to interpreting. Second is the same image in YUV color space which is what the robot actually uses. Bottom is the color segmented version of the image after processing is finished.

Pixels are classified using a hand generated threshold file into red, green, blue and unclassified. Regions are formed by joining contiguous pixels of the same classification and absorbing noise within some threshold to produce the segmented image at the bottom of figure 3. The ball is detected as the largest red blob in the image. When locating the ball room the ball is in is determined by whether green or blue has the larger occurrence when the ball is centered in the robot's view. Because the walls and floor both are colored, this simple scheme determines the ball's room correctly. For checking which room the robot is in it similarly checks which color has more area when it looks down.

Action Implementation

In the Tekkotsu framework, individual programs are referred to as behaviors. Each one of the actions used by the planner is implemented as an individual behavior. A single manager behavior takes care of communication with the planner on the PC and starts the other behaviors as requested by the planner.

While the overall behavior of the robot is deliberative there is a combination of deliberative and reactive intelligence. Each one of the actions constitutes a purely reactive program. For example, the grab ball action moves to the ball by first attempting to center the ball in the robot's field of view by moving the head. It then tries to center the neck joint by turning the body. The robot's speed is a function of the remaining distance to the ball until it gets very close at which point it lifts its head steps forward and lowers its head again to capture the ball. The combination of reactive with deliberative action is required for the robot to be able to respond to the motion of the ball, and its own motion, quickly.

The Planner

Planning is accomplished using the University of Washington Sensory Graphplan LISP code [1]. The code is run under Allegro Common Lisp 7.0 [3] and wrapped in a set of functions I created to manage communication with the robot. The domain is specified in PDDL as in figure 2 but the problem definition is dynamically generated based on what information the robot has about the world state at the time. Since SGP supports uncertainty the problem can be stated with several possible initial worlds if the robot has incomplete information as is usually the case.

When the program is run a plan is generated then broken down and sent to the robot. Plans returned by SGP are conditional partial order plans. Some of the steps in the plan should be executed under all conditions while others should not be executed depending on what the initial world state was. Sensing actions narrow down the possibilities of what the initial state actually was. To ensure that we don't execute actions which we're not supposed to given the actual initial world state, my SGP wrapper functions maintain a list of the possible worlds that the robot might have been started in. As results from sensing actions are returned, these are used to prune possibilities out of this list. At each plan step if one of the possible worlds is one of the worlds which that action should not be taken in, that action is skipped and the planner moves on to the next step in the partial order plan.

Replanning

Unlike for the simulated agent, the actions our robot takes are not guaranteed success. It is possible and in some cases quite likely that after taking an action the world state

will not have been transformed the way the planner expects. Thus we need the capacity to detect when events haven't occurred as expected and replan from where we are now.

There are many possible policies for checking action effects and replanning which one might implement. Lack of time limited me to trying only one. The planner maintains variables internally with what it expects the current world state to be based on its (partial) knowledge of the initial state and the actions it has taken so far. After certain actions it performs extra sensing actions (not specified in the SGP plan) to determine if the actual state matches its expectation. The *grab-ball* and *go-dest* actions have the most significant effect on the world state and known *a priori* to be the most likely to fail (i.e. not have the expected result). Thus in the planner I implement effect checking and possible replan triggering after those two actions.

Effect Checking

Effects are checked in an order designed to have the minimum cost and indicate a failure of the action as quickly as possible. As soon as even one predicate differs from the expected value it stops checking and requests a replan using the sensor data it has just collected to specify the initial state.

The order predicates are checked in is as follows:

1. Holding the ball, low cost to check and it quickly indicates a failure in many cases. Also, if the robot is holding the ball then when checking the robot's location also determines the balls location by extension.
2. Robot's location, similarly cheap to check since it only involves moving the head down.
3. The ball's location, this is expensive to check since it involves scanning the

whole world. However, in practice, this is never done during effect checking because we know the ball's location from the fact that we are holding it and we know our own, or we have already determined that the actions has failed before this check is reached.

Generating a New Plan

If the current world state turns out not to match expectation when effect checking is done, the planner defines a new PDDL problem based on the new sensory values just set by effect checking. For this problem definition we will always know at least one predicate value, since we had to sense at least one of them to detect the failure, and it is possible all three will be known in which case the new plan will not need to contain any additional sensing actions.

After the new problem has been defined SGP is called on it and whatever was left of the previous plan is replaced by the new plan. Execution then starts on this new plan. Because LISP is innately recursive, the cycle of planning, executing and replanning can continue indefinitely until the robot succeeds at reaching the goal state.

Difficulties Encountered

Programming challenges from this project can be split into three groups: programming the actions to run on the robot, interfacing the robot with the planner and programming the planner.

Programming Actions

While implementing the actions to run on the AIBO I encountered some of the common difficulties in robot programming such as calibrating the sensors, physical limitations of the robot and the difficulty of debugging code running on a remote

machine. The greatest challenge was in designing actions which would reliably manipulate the ball the way the planner expected it to. Since the AIBO has no actual gripper it can only “hold” the ball by trying to corral it between its front legs, chest and chin. Keeping the ball in this position while still being able to use the head to look around and having the robot move around the domain required a long iterative process to develop the action behaviors.

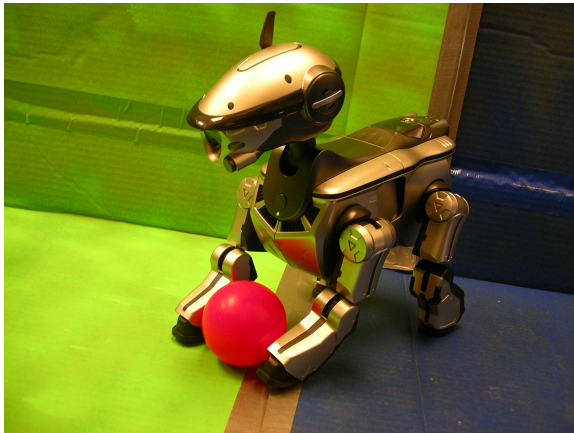


Figure 4: Robot corraling the ball between its front legs while using its head to locate destination.

Interfacing the Robot and Planner

The first challenge in interfacing is the difference in data types between LISP for the planner and C++ for the robot. Strings turn out to be the most easily consistent between the C++ and lisp processes. From the robot’s perspective, the interface is simply a socket from which it receives line terminated strings to indicate which action it is to take and to which it writes single characters either Y for true or N for false. On the PC the planner opens a socket to the robot and sends command strings to the robot and waits after each command for a one character reply resulting from the action. Synchronization turned out to be a small problem at first since it is essential that the robot is executing the action which the planner is expecting a result from and that

the planner doesn’t send any additional commands until the action has completed.

Programming the Planner

There were two main aspects to writing the planner. First representing the domain and problem in PDDL so that SGP could be used to solve the problem and second writing appropriate wrapper functions to execute the plan and replan as necessary.

Writing the PDDL domain definition is effectively programming the robot’s behavior because the domain definition determines the plan which is generated and the plan is the robots behavior. However, PDDL is a declarative language not a procedural one. Writing a program in PDDL consists of declaring various properties and truths about the world instead of writing functions or logic chains to control the robot. The difficulty arises from needing to encode all of the assumptions which the human programmer makes without realizing it but the planner doesn’t know about until specifically told.

For example, early on my domain definition included the statement that the effect of the *release-ball* action was not *holding* and *facing-ball*. Based on this the plans generated would start with *release-ball*, *grab-ball* since according to the domain definition, executing *release-ball* would satisfy the *facing-ball* precondition so the robot could grab the ball. The assumption I was making was that you only release the ball when you’re holding the ball but I had not included this implicit precondition in the PDDL code so the planner had no such assumption. Writing and fine tuning the PDDL definition consisted of realizing and encoding many such assumptions which resulted in the definition in figure 2.

One other difficulty of encoding the domain was that SGP does not want sensing actions to have preconditions or conditional effects but *locate-ball* as implemented on the robot only works when we aren't already holding the ball. The definition of *locate-ball* in figure 2 is the best compromise I could achieve which would work with SGP to generate competent behavior.

The second part of programming the planner was writing wrapper functions for SGP to dynamically create a problem definition, generate a plan, and then recurse through the plan. Of these the first was actually the most difficult. Fortunately this is one of the places where LISP is uniquely convenient. Macros allow programs to effectively generate their own code very easily via the programs as data principle.

In two places I added code to modify the plan returned by SGP. The first was adding effect checking as described above to initiate replanning if necessary. Second I added checking to prune out immediately repeated actions. Because of the way SGP generate conditional-partial-ordered plans, all branches must be the same length. In order to make them all the same length when some branches don't actually require as many steps, the planner repeats an action multiple times as an effective non-action (in this case *locate-ball* which if repeated immediately after being called doesn't change the world state). This repetition costs time in the physical world so the plan execution code will simply skip an action if it is the same action it executed last. In general it is not acceptable to skip an action simply because we have just executed it since some actions may have different effects when executed repeatedly but SGP only repeats actions as no-ops if they actually don't change anything. Another place this pruning might cause problems is in replanning since the

last action executed by the old plan might also need to be the first action in the replan. This isn't a problem either though because the effect checking which would prompt the replan involves at least one sensing action in between the old and new plans.

Experimental Results

After completing the implementation of the actions on the robot and the planner I tested the robots ability to function intelligently in its domain. I started the robot and planner with the following conditions:

1. Robot and ball in the green room
2. Robot in the green room and ball in the blue room
3. Ball in the green room and robot in the blue room
4. Both the robot and the ball in the blue room.

In all cases the planner would generate the appropriate plan to reach the goal state of the robot and ball being in the green room.

Intriguingly, in case number 1, which is already the goal, the agent does not just conduct sensing actions and then realize its done but instead locates the ball then grabs it. Analyzing this plan it turns out that under an equal cost assumption it is on average cheaper than locating the ball then checking the robot's location. The latter would require two more actions to locate and go to the green room if the robot turned out to be in the blue room thus giving an average cost of 3 actions. Instead just grabbing the ball if the ball is in the green room is guaranteed to have the robot in the green room (since the robot ends the *grab-ball* action in the same room as the ball) and has a cost of only 2 actions.

It also turns out that the plans for cases 1 and 3 and for cases 2 and 4 are identical. In

1 and 3 the robot will be in the green room after grabbing the ball irrespective of where it started. While in cases 2 and 4, the robot again locates then grabs the ball after which it is in the blue room in both cases and the problem is the same from then on.

In the course of executing the plan actions did occasionally fail on their own, the robot loses control of the ball somewhere through transporting it to the destination or just fails to capture it in the first place. When this occurred, the agent did in reasonably short order identify that it had failed and generate a new plan. The replanning process would continue as many times as necessary until the robot succeeded in transforming the world to the goal state.

The Tortured Agent

After demonstrating that the agent could solve the problem intelligently without interference, I started deliberately causing actions to have unexpected effects. For instance, removing the ball from the robot's grasp, pausing the robot's motion midway through an action so that it couldn't move temporarily, moving the ball to the other room after the robot had decided which room it was in. Even when interfered with repeatedly in this manner, the agent would continue detecting the discrepancy between the real world state and its expectation and generating new plans based on the new world state and attempting to execute them until it either succeeded or I placed it in the goal state.

An Extension

A somewhat tangential extension to this work which I will describe in brief was creating an agent which planned in largely the same manner but did not start out with a definition of its actions. Whereas for my main project the domain definition for SGP

was fixed but the problem definition changed depending on the world state, for this extension, both the problem definition and the domain definition were dynamically defined. Because the agent began without knowledge of action preconditions or effects it had to determine them by trial and error. Two main simplifications were involved in this extension. First instead of having preconditions all actions were conditional with the post-conditions in some states turning out to be the same state. SGP actually converts all actions with preconditions into this format anyway so this just makes the conversion explicit. Second sensing actions were removed from the problem, before and after each action the robot would always conduct enough sensing to determine the current world state.

The basic algorithm for this trial and error planning agent was as follows:

1. Check current state to see if it is goal
2. Attempt to generate a plan with existing knowledge.
3. Try an action we don't know the effect of from the current state and bind the result.
4. Take a random action to change the state
5. Loop till until #1 is true.

Extension Results

In this very small domain, an agent who starts with no domain knowledge can actually discover enough information about the domain to reach the goal state. Quite often the agent would stumble upon the goal state while conducting its trial and error exploration of the domain. If placed back in a state from which it had reached the goal and restarted, however, it would remember the domain knowledge it had gained and successfully generate and execute a plan to reach the goal state.

Conclusions

For a domain of this size, deliberative planning intelligence can successfully guide a physical agent through a series of actions from an uncertain initial state to the goal state. This paper has described the basic challenges and solutions to using Sensory Graphplan to generate a plan for the physical agent, have the physical agent execute the plan and deal with replanning issues when real world uncertainty and imperfection cause the world state to differ from that the plan indicates it should be. The problem of planning and replanning with a physical agent is tractable but not trivial to solve.

Future Work

There are a number of areas where this work could be extended to improve the robot's performance or simply explore new aspects of the fundamental problem.

- A larger domain, multiple balls to move or a physically larger and / or more complex area to navigate would make the problem more interesting. The initial formulation of this domain was as an extremely simplified version of a robot locating bringing a disaster survivor (the ball) to safety (the green room from the unsafe blue room). Making the domain more complex to start making it more like that real world problem would be an extremely interesting area for further work. A more complex domain would also likely bring out the limitations in the current implementation which aren't problems now because of the simplicity of the domain.
- More general replanning techniques. The current mechanism for determining that

a new plan is needed is based mainly on *a priori* knowledge of the domain and is not easily generalized to new domains. Ideally we would like the planner itself to have a mechanism to decide intelligently what additional sensing action it needs to take to determine if it needs to replan and when and how often to do it. Given more time this would be the most likely place for immediate continuation of this work.

- More efficient replanning. The current system completely discards the old plan when it becomes clear that it is no longer valid. In many cases the remaining part of the plan is still nearly valid and the agent shouldn't have to generate a whole new plan which is computationally expensive. Thus a planner which could repair the remaining plan instead of replacing it would be desirable for efficiency. Doing so would require a different planning algorithm than SGP for the repair component at least since SGP is a state space planner not a plan space system and cannot repair a nearly correct plan.

References

- [1] Anderson, C. "Sensory Graphplan." Washington University Computer Science & Engineering.
<www.cs.washington.edu/ai/sgp.html>
- [2] Anderson, C., Smith, D. and Weld, D 1998. Conditional Effects in Graphplan. in *AIPS-98*
- [3] Franz Inc. "Allegro CL®" 2006
<www.franz.com/products/allegrocl/>
- [4] Russle, S and Norvig, P. Artificial Intelligence: A Modern Approach. New Helhi, Prentice-Hall, 2005.

[5] “Tekkotsu: an open source project
created & maintained at Carnegie Mellon
University”
2005-07-20 <www.cs.cmu.edu/~tekkotsu/>

[6] Weld, D., Anderson, C. and Smith, D
1998. Extending Graphplan to Handle
Uncertainty & Sensing Actions. In *AAAI-98*