NODE: A Robotic Process Virtual Machine

Author Names Omitted for Anonymous Review. Paper-ID [add your ID here]

***Abstract*—Robots need to pick stuff up. Perception can structure data, structured data facilitates planning, planning allows more principled perception. The tasks of robotic perception, planning, and control will all benefit from a design paradigm which allows them to be jointly programmed. So we contribute an implementation *NODE* which facilitates the design and execution of versatile and scalable MDPs structured in what we call a Von Neumann MDP pattern. We form a closed loop through perception and planning, allowing them to interact by means of structured data. Our system incorporates online, human-in-the-loop algorithms. Non-technical human participants can easily teach and collaborate with the system.**

I. Introduction

There are now many types of robots. A common configuration is one that mimics the human arm and hand system (AHS). Most tasks utilizing an AHS focus on controlling the pose of the end effector, which is determined by 6 degrees of freedom. Thus a hand should have at least 6 degrees of freedom, and typical AH systems have at least 7.

A major problem in robotics research is that, while the basic form of AHS bots has remained similar over time, the systems architectures for controlling those bots tend to be built anew. Therefore, when a lab acquires a new robot, a lot of engineering overhead must be paid in order to get old procedures running on the new bot.

We dramatically increase the portability of robotic procedures by demonstrating a powerful abstraction framework (NODE) which depends only at a very low level upon the robotic platform of execution. Porting code to a new AHS is exactly the act of reconciling a simple programmatic interface with hardware drivers.

Furthermore, NODE is a framework whose structure jointly facilitates perception of and planning in the robot's environment. This allows powerful algorithms to be implemented in a generic, portable environment.

That is to say, NODE is a platform on which we can execute MDPs and related processes.

A Markov decicion process is essentially a state machine with some stochastic planning (reward) sugar on top. It is no surprise, then, that we address the construction of MDPs with a mindset similar to that which we use in the construction of computer programs. Take, for instance, object-oriented MDPs. But there are some things between the state machine of a computer and an object oriented programming language: machine language (pushing the state machine closer to being a true Turing machine), the Von Neumann computer architecture, and its accompanying execution model.

A key point of this architecture is that RAM holds not only program data but also program instructions. Modern implementations also include convenient hierachical cache structures to facilitate fast, "real time" access to many more times data than fits in RAM.

In order to quickly construct MDPs that scale and port like modern software, we provide an archetype system that is to an MDP as a Von Neumann computer is to a Turing machine. In particular, the state of the MDP includes a call stack of instructions which the MDP can modify, as well as data pertaining to various machine learning algorithms which it employs.

Actions in a VNMDP are defined in terms of a basic "machine language".

We used the archetype to construct several MDPs over the Baxter robot. This MDP learns to recognize and manipulate objects which are presented to the bot. We achieve good recognition rates and can learn parameters for complex behaviors such as grasping and throwing. Since its instructions are stored in its main memory, it has the additional ability to learn actions by writing its own code.

II. Related Work

Here we review the standard AI, vision, and ML algorithms we employ. We use the language of ensemble learning to speak about obtaining strong solutions to general problems from weak solutions to specific cases of those problems.

we are bootstrapping proverbially. our philosophy espouses solving weak instances of problems in constrained spaces and using those solutions to create strong solutions in unconstrained spaces. Thereby, we bootstrap the detection process. (To truly be faithful to this approach, we must replace the objectness detector, as it uses an externally trained model. Something like a high sigma DoG or gradient magnitude should do the trick, and it harkens back to Canny.)

We are not stacking our classifiers in the strictest of senses, but k-means could be considered a subordinate classifier so in that sense we are. Logistic regression is a likely candidate for a fusion method when classifying windows of states, and so then we might start to truly stack.

We are using bootstrapping to estimate properties such as table orientation and background color. as it turns out, it may be worth investigating bagged nearest neighbors because we approximating it by resampling the data.

We make heavy use of cascades.

Our training method can be viewed as boosting-motivated because we detect places where the classifier fails and train a sub-nearest neighbors classifier in that neighborhood. Mining hard negatives for kNN is the same as boosting a kNN classifier.

relevant actions are proposed, and activities can then be classified in their own arc. there might be an action happening

at these objects.

the notion of rapids (knn -¿ svm -¿ fit) can be naturally motivated by the visual system, as has been noted before with BoW and SBoW

affordances facilitate cascade arrangement. make sure to talk about viola-jones.

dont forget that k-means here is actually a classifier and so we are technically stacking. introducing the background shunt will make that cascade-like. a shunt is a special kind of cascade step which produces something like a mask.

one approach to teaching the robot to move is to teach it a form for each task. each movement in the form should have parameters corresponding to target points, known obstacles, etc. It should be similar to dealing with character animations in video games.

III. NODE Architecture

NODE is a process virtual machine which implements an MDP with a design patterned after the Von Neumann computer architecture.

One reason to choose this pattern is that it facilitates the organization of states and transitions so that we can build MDPs as easily as we build traditional computer programs.

Another reason to choose the VNMDP design pattern is that the majority of execution platforms (i.e. computers) are modeled after Von Neumann Machines. By sharing that pattern, we can scale MDPs by taking advantage of the natural, efficiently engineered architecture which allows computer programs to operate in real time.

A key property of the Von Neumann computer is that it is a stored instruction computer, meaning that program instructions are stored in its main memory and can be modified by the computer. This pattern yields a direct means for our MDP to learn and describe new behaviours: its instruction set, a language that it interprets during execution.

The NODE CPU (nCPU) consists of a pushdown automaton which utilizes a call stack and a fixed capacity bank of registers which hold data retrieved from the nCACHE. The nCPU has an instruction set, which is the set of atomic operations that we want the MDP to be able to plan with. The instruction set encodes part of the MDP's transition function, is dependent totally upon state, and is invariant to actions taken. The nCPU instructions consist of atomic motor commands (which are passed to and handled by the nARU), basic linear algebra operations to manipulate vector and pose data, synchronization commands to keep subsystems in step and coordinated, sensory monitoring commands (resource control is essential), planning and policy execution, and any other basic calculation or manipulation that we want the MDP to be able to use in its plans.

The NODE Atomic Robot Unit (nARU) implements a set of atomic robotic operations and can be thought of as the hooks connecting NODE to its host robot. It is a component of the nCPU.

The NODE Cache (nCACHE) is local memory associated with the nCPU and nCACHE. It is the medium between the nCPU and the main memory hierarchy.

The NODE RAM (nRAM) consists of object models and procedural data that is relevant to the executing program. Its construction and maintenance are described by the NODE Short Term Memory Model (nSTMM).

The NODE Disk (nDISK) consists of all previously stored memories which have not yet been forgotten. It is described by the NODE Long Term Memory Model (nLTMM) and is structurally isomorphic to the nSTMM but procedurally different.

The NODE GPU (nGPU) consists of a processor structure analogous to the nCPU but asynchronous from it, with instructions suited to the implementation of a set of computer vision algorithms.

******** diagram of pattern ********

IV. Memory Model: nCACHE, nRAM and nDISK

Structured memory facilitates planning and the ability to scale while maintaining high frequency.

Working Memory (nCACHE) What object tokens exist in the world? What are their categories and their names? (names map particular object instances I see now to instances that persist over time, that is, for which I have a history) Where are they in the scene?

Short Term Persistent Memory (nRAM) Structurally identical to long term memory but restricted to the current context. This is much like a cache for long term memory. Short term memories are incorporated into long term memory either online through the day or offline during a sleep phase.

Long Term Memory (nDISK) Intrinsic (Autobiographical) Episodic Memory This is the sensory data that we are collecting. This information cannot be provided by external sources, it comes from the perceptual capabilities of the system. Traditionally this would be thought of as Training Data X. Extrinsic Episodic Memory Labels and names for categories and peristant instances. This information can be provided by another source and will eventually be provided by the entity itself. Traditionally this would be thought of as Training Labels Y. Implicit Memory These are the models, such as SVM or kNN, that are involved in mapping intrinsic memory to the extrinsic memory. Semantic Memory Facts about stuff that can be used to reason. Affordances Procedural Memory Grasps and collision free planning are procedural memory.

each degree of cascade is appropriate for particular regimes. you can switch regimes in real time to accommodate changing environment complexity or resource availability.

at each application, planning includes operator input / real time parameter locking.

a detector cache hierarchy emerges with policies. this immediately induces a dual data hierarchy.

system resources can be bounded by the number of red boxes, which conveniently bounds the number of pose detectors (PDs) necessary to keep around at one time. this is a fixed length PD cache. but if PDs act to break a tie, than PD-cache inherits structure from CD-cache

class detectors can be regenerated based on context. there is a CD-cache from red boxes, a CD-cache from context, and a CD-cache from hard negative classes.

time is segmented into phrases with relative frames (such as boundary frames) saved as observations. most observations are incomplete, but we can tell which examples we can use for which training tasks. it is an affordance.

V. Core Model: nCPU

We use a pushdown automaton with a call stack (CS), several integer state variables (S_i), and an end effector pose cache (EEPC).

******** A subset of the instruction set ********

Description of mapping of registers to MDP state variables.

Examples of how instructions change the state, i.e. how they contribute to the transition function.

In addition to being probabilistically entwined with the external environment, actions of the MDP operate on the internal environment by modifying the program running on the virtual machine.

VI. Robot Model: nARU

The nARU translates Atomic Robotic Instructions invoked by the nCPU into robot driving commands.

VII. Vision Model: nGPU

This should probably be thought of as some number of CPU cores working in conjunction with a GPU.

an appealing approach is to train the SVM only on the examples proposed by the kNN models.

a simpler cascade sliding window on class mesh based pose estimator such as ICP

current cascade objectness knn-bow class label knn pose label

proposed 1.0 cascade objectness class rapids knn-bow class candidates discriminative classifier pose rapids knn pose label candidates discriminative pose estimator

a deep cascade objectness provides evidence; point cloud and other hooks feed in here knn-bow class suggestions form blue labels for blue boxes; planning can place contextual clues here by providing suggestions restricted discriminative classifier (svm or other) forms red label for red boxes; planning can, for instance, fix a red label generative model with volumetric representation classifies pose and accounts objectness evidence, depth information and object labels (in the case of parts and compound objects, this includes dynamics information like constraints)

break the observed space into a block representation, possibly with octree encoding weighted according to scale, and use kNN or hashing to search the space for a similar landscape. then use the navigation schemes learned for the closest neighbors.

then we can begin volumetric modeling of objects. each block has an rgb-a value associated with it, alpha representing the frequency with which depth is observed at that point. we can apply markov methods to regularize these models. then it would be fast to move to modeling the faces of the blocks with independent filters, treating the faces as patches, to capture texture.

actually, red boxes are likely to become "target patches" rather than objects themselves.

VIII. Online Learning With Humans in the Loop

Here we describe the algorithms which we use to build and grow the memory network, which are the second contribution and second half of our framework. Note that performing inference with the framework is actually the same as growing the memory, since the results of inference are recorded in working and short term memory directly and strongly contribute to long term memory.

We learn the affordance "can be grasped" through RL or just trial and error. We can report the success rate over time.

there are certain facets of the world which afford actions that are useful to us during training. the table and the pointer afford labeling the objects with orientations. tables and grounds afford orienting yourself with the world. you could drop an object to find down. the image background affords object segmentation.

our teaching framework incorporates signs with affordances to streamline the process of data collection.

we teach the system by finding its weaknesses and training it to overcome these weaknesses. we provide a framework for structuring that training in an affordable, scalable way. we structure the training, the training structures the models.

I see youre having trouble picking up gyroBowls. Lets think about it differently (initialize generic pose model) and practice gazing a bit (train the pose model).

Also, grounding ambiguous object examples to the right class, retraining the models online.

A. Learning Parameters for Fixed MDP Programs

B. Learning New MDP Programs

IX. Utility Evaluation

Our system can be evaluated in two important ways. Firstly, how effective is the system at facilitating other actions? Secondly, how accessible to users is the system?

Because this system is designed as a component in a complex system, raw recognition rates and pose estimation accuracy would not be very informative even if it were not impractical to obtain them. Besides, we are not trying to extend the state-of-the-art on those tasks. Rather, we are trying to provide an interactive framework which will raise the maximum automated vision available to the average user.

We therefore establish a baseline for performance by training the system in a representative domain specific setting, which tells us how well it can perform on laboratory objects when trained by an expert. This represents the best that the system could be expected to perform.

We then go on to compare the performance of the system when trained to various degrees by naive and technical non-expert users.

A. Baseline Domain Specific Pick and Place

We qualitatively evaluate the performance by picking and placing a respresentative set of objects and reporting the final success rate for each object separately.

We report the performance of the system as a function of user interactions.

We report the performance of the system as a function of program lifetime.

Our representative set consists of a block, a spoon, a bowl, a diaper, and a sippy cup. A *single cut video* showing multiple grasps of all objects is available here.

B. Usability Experiments

We repeat the DSPP experiments with naive users in two settings. In the first, they train laboratory objects. In the second, they provide their own objects.

X. Ludi Metric

Training and evaluating with games.

A. Competitive Pick and Place

Take turns trying to move the object to your opponent's goal while they try to get in your way.

B. Basketball

Toss your projectiles through your opponent's hoop while guarding your own.

[**?**]

XI. Extensions

Right now, NODE runs on Baxter. We will port NODE to PR2 and other AH systems.