Dynamical Models for Instruction Completion and Error Recognition for NASA Physical Procedures

Steven Johnson
Department of Computer Sciences
University of Wisconsin–Madison
sjj@cs.wisc.edu

Ronak Mehta
Department of Computer Sciences
University of Wisconsin-Madison
ronakrm@cs.wisc.edu

John Cabaj
Department of Electrical and Computer Engineering
University of Wisconsin-Madison
cabaj@wisc.edu

Abstract

1. Introduction

Procedures are the accepted means to operate a spacecraft system or systems to perform specific functions, and consequently are at the heart of all NASA human spaceflight operations [9]. A procedure is a detailed set of instructions specifying how a piece of equipment is operated or a task is performed [7]. They are often written to be very general and to cover numerous contingencies. Procedures to operate a class of equipment (e.g., smoke detector) will differ based on make, while procedures to operate a piece of equipment will have conditional or optional steps based on configuration. As an additional complication, constraints of some procedures may be highly conditional, discretionary, or unordered. At the same time, there may be external constraints that limit how a procedure must be executed, and these constraints are not made explicit. The outcomes of NASA missions rely on crew members properly executing a multitude of these complex procedures, making procedure execution support and monitoring a critical factor that can determine success or failure measured both in terms of monetary costs as well as preventing loss of life.

There is a body of prior NASA work focused on monitoring the progress of procedures that are not physical. For instance, when instructions to systems of the ISS are sent from ground, the application ThinLayer highlights commands as they are executed to show procedure progress [7]. IPV itself also allows for manually tracking procedure progress for a crew person onboard ISS. However, to date there is little work from NASA in the realm of tracking execution status of physical procedures where crew members are manually manipulating physical objects, such as during maintenance tasks. Our goal with this work is to develop a method to computationally model a procedure to enable tracking of the execution of its steps and detection of crew errors during physical execution.

2. Related Work

Significant work has been done in the field of human action and activity recognition. Turaga et al. [14] presents a comprehensive overview of this work in detail. In most work, activity recognition is identified as the sum of *actions* performed in a temporal ordering. Parametric models, particularly Hidden Markov Models, have been used with success in many action recognition applications. Yamato et al. [17] employ them to identify whole-body tennis swings. Using background subtraction, they are able to identify the actor in the scene, and learn a model based on how the actor alone is moving over time.

The particular advantage of HMMs in the field of vision lies in their efficiency in modeling time-sequential data. Because the underlying representation of an HMM essentially encodes a state machine, they lend themselves nicely to modeling actions. Actions are composed of sequential time-varying video frames, and the model creates a representation with ”probabilistic jumps” from one state to another [14].

There has been some work in developing HMM formalisms that incorporate domain knowledge as well. Moore et al. [10] created a framework combining both HMMs and object detection modules to extract interaction information HMMs alone could not identify. In the work of Chen et al. [3] HMMs are used in conjunction with specific feature extraction methods to recognize hand gestures.

While the above works have been significantly expanded upon, all of that effort has been under the assumption of a third-person view of the action, where the camera is removed from the scene. In our setup, our observation of the underlying scene occurs from an egocentric viewpoint. Egocentric cameras present their own unique affordances, and come with their own unique problems. Technological advances have only recently begun to make common use of egocentric cameras a reality, and as such their is a limited existing body of work to use as a foundation.

Though this does prove to be an interesting challenge, the task of instruction completion and error recognition provides us with significant domain knowledge. Rather than having a classification problem in which an unknown task is to be identified, we have significant knowledge of what has already been completed, what the correct next action should look like, and what error states look like. It is this knowledge that we hope to incorporate into our model.

In this way, we employ the concept of *activity* recognition, coupled with our action recognition. We model the entire activity as a set of actions completed in a specific order. To this end, Petri Nets appear to apply quite well [13]. The graphical model format, along with the unique properties of sequencing, concurrency, synchronization, and resource sharing [4] that Petri Nets provide allow for a complete high-level representation of the activity. Incorporating the entire procedure in the form of a Petri Net, coupled with the lower-level HMM, we hope to create a system which will allow for effective instruction completion and error recognition.

3. Feature-Action-Activity Pipeline

Because of our novel combination of a variety of vision and machine learning methods, a pipeline was developed to afford fast and efficient processing of raw video data directly through to the learning process.

During the training stage, low-level features are extracted from training videos and binned. These features are then clustered to create symbol representations, and sequences of these symbols are used as training data for the Hidden Markov Models.

3.1. Feature Representation

In recent years, dense trajectories have proven to be the most accurate feature representation for action recognition [15]. While their combined representation leads to strong classification results, the downside is the immense computational complexity. In order to provide reasonable notification of an erroneous action, our system needs to be as close to real-time as possible using reasonable hardware. While much of the computation can be offloaded to a server, in environments similar to the ISS such computation can still be costly, making dense trajectories feasible for this context of use.

With this motivation, we solely make use of optical flow as our low-level feature representation. Wang et al. [15] show strong results using only optical flow, and there is some heritage in using it for action recognition [2].

To obtain the keypoints in the video frames, Lucas-Kanade feature tracking was used to obtain the flow between frames. Bouguet's pyramidal implementation allows for features to be tracked across scales, while still being reasonably fast in its extraction [1].

Features were tracked over half-second periods. Given our domain knowledge, we expect an action to take time on the order of 10 seconds to 1 minute, and as such expect half-second features will provide adequate information to describe the entire period.

Once features have been extracted, Histograms of Oriented Optical Flow (HOOF) were generated for each half-second interval. We might expect subactions over these intervals to range from being relatively obvious to fairly subtle, and as such performed a grid search over the number of bins.

3.2. Codebook Generation

During the training phase, a codebook was generated to represent subactions from which an action could be comprised. Using the binned feature representations from above, k-means clustering was performed on all features extracted from all training datasets. These clusters were used as the symbols passed forward to the Hidden Markov Model.

During testing, each half-second feature was queried for the cluster closest to it, and that cluster ID was used as the symbol for state estimation. Because our domain implies similar movements across multiple instances of a single action, we expect that this method should result in reasonably accurate cluster assignments.

3.3. Hidden Markov Models

Classification of the assigned task required that each of the actions that made up the tasks must be recognized to determine whether or not instructions had been completely incrementally. To achieve classification of individual actions, a Hidden Markov Model (HMM) implementation was used. Markov Models are essentially a graph structure that represent states of an input sequence probabilistically. Given a known set of possible observed symbols and states, the model can determine the probability of a transition from one state to another given an input sequence.

Given that the state sequences of the optical flow features could not be determined in advance, a Hidden Markov Model was trained. A HMM has no knowledge of the state topology that was used to generate an output sequence.

However, a HMM can be trained based on instances of observed sequences using an Expectation-Maximization algorithm. The most commonly used is the Baum-Welch algorithm [16]. To simplify the HMM implementation, discrete output sequences were assumed and achieved by means of the previously described codebook generation. When a new HOOF feature is generated from optical flow, the nearest cluster center from the codebook classifies each flow. This cluster center roughly maps to a subaction of the activity being performed. An activity is made up of many optical flows over many frames, thus finding the closest cluster centers provides a symbol of roughly which subaction occurred in the activity as a whole. Using such a method allowed for discrete HMMs to be used.

For action recognition, one HMM will model a single action. As the task used for testing had eight activities, eight HMMs had to be trained for action recognition. All eight models were initialized with random parameters for the transition probabilities and emission states. Training data was segmented into only instances of a single activity for HMM training. The Baum-Welch algorithm was then used to train a single HMM model based on all observed sequences of an activity using the training data. The likelihood of the model representing the observed sequences is calculated in every iteration until the likelihood converges. This occurs for all eight HMMs used to classify all eight actions until the models parameters have converged and the HMMs are built.

For classification given observed sequences s of testing data, the likelihood $L(M)$ of each HMM M given an input sequence is calculated. Activity classification is then based on which of the eight HMMs gives the highest likelihood.

$$\hat{a} = \arg_{a \in A} \max L(M(s)) \tag{1}$$

The HMM that achieves the highest likelihood is deemed the most likely action and we classify it as such. We used an implementation of the HMM Toolbox from Kevin Murphy [11].

3.4. Petri Network

A Petri Network was used as a means of modeling the task as a series of actions to be completed to achieve the classification of the task as a whole. Petri Networks were first introduced by Carl Adam Petri [12]. Petri Networks are represented by a bipartite graph that represent states that you can be in, called places, as well as arcs from place to place, known as transitions. When in a given state, there are any number of tokens in that state representing that the place is the current place. When a state can transition to another, the transition is said to fire, whereupon we advance to a different place and another set of transitions can be enabled.

In this standard definition of Petri Networks, one can model behavior similar to a Finite State Machine. Given the nature of an assigned task, with domain knowledge, this scales with our model very well as we have several known activities that must be completed before the task as a whole can be deemed complete. For our purposes, each node represents the fact that we are currently waiting for a specific activity to be recognized, while a transition represents that a transition from one place to another indicates that an activitiy was detected, and the task has progressed to the next stage overall.

In this basic set up, we simply defined the layout of our Petri Network as shown in Figure 1. Using the known structure of the Petri Network, the activity classifications are passed off from the HMM models' outputs. When the likelihood of an action is greater than a threshold, we can say that the activity was observed, and the Petri Network should advance accordingly. When all necessary transitions to perform the task are satisfied, we can say with confidence that the task was completed based on the series of activities observed. Petri Network modeling was achieved using GPenSIM [5].

4. Experimental Evaluation

We evaluated our proposed Feature-Action-Activity pipeline in a number of experiments. Below we describe our dataset, which was constructed based off of an actual NASA procedure and related constraints. Additionally, we describe the acquisition of the dataset following by the results of classification experiments with and without use of the Petri Network.

4.1. Dataset

Frank et al. [8] describe several physical procedures frequently conducted on the International Space Station (ISS). One of these procedures is the Interim Resistive Exercise Device (iRED) Inspection and Cleaning task. This procedure involves crew disassembling the piece of exercise equipment, inspecting and cleaning the inner parts, capturing and sending inspection photos to ground control, and then reassembling the device. We based the task we created on this procedure.

The piece of exercise equipment we used for our task is shown on the left in Figure 2 with the necessary tools shown on the right. The complete disassembly, photographing, and reassembly task consists of 66 actions. However, we used only the first eight actions of the disassembly portion of the task in our dataset. This task therefore consisted of 1) removing an electronic display, 2 and 3) removing two resistance cords, 4) attaching the proper drill bit to an electronic screwdriver, 5 and 6) removing the two steps, 7) unscrewing by hand a resistance knob, and 8) detaching a wheel. This task involves a mixture of tool use and hand manipulation of components of a variety of parts on the exercise device. It therefore requires a large amount of participant mobility.

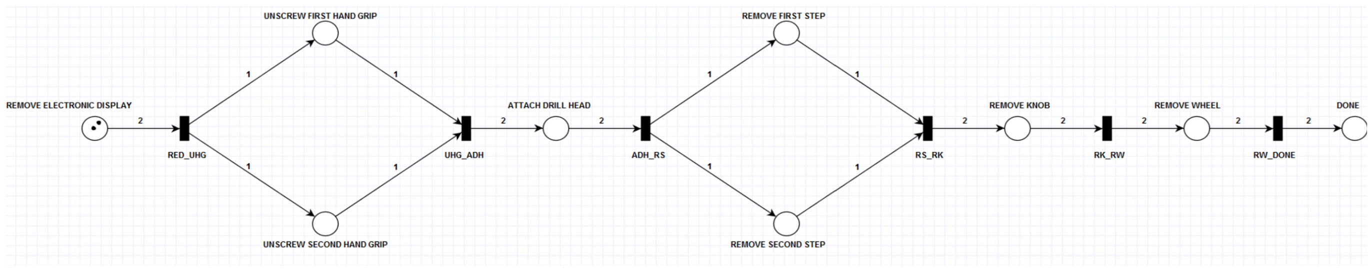

Figure 1. The Petri Net structure for the eight actions used in the disassembly task.

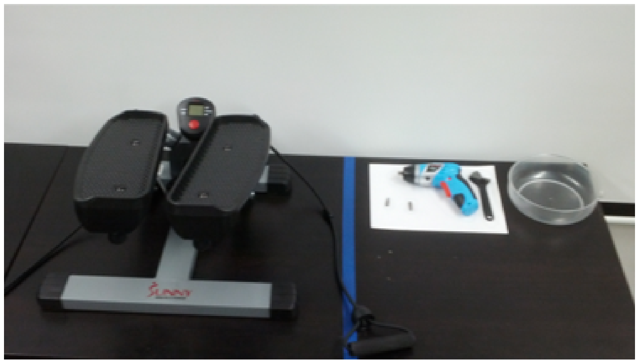

Figure 2. The piece of exercise equipment and tools we used in the task to create our dataset.

For the data collection, each participant executed the disassembly procedure with the instructions for the procedure delivered via a prototype task support system developed on the Google Glass[1] Platform, a lightweight head-mounted display device. These types of task guidance systems are being developed in other work, but this system provides a good platform for the acquisition of egocentric video information captured from the Glass device's camera, so it was selected for this purpose. The device was modified slightly to better capture the salient visual content of the task execution. The modified device is shown in Figure 3. We attached a mirror at an angle of approximately 25° from level to the device with a 3D printed mount. When manipulating objects with the hands in front of themselves, most people both tilt their heads downward slightly and look downward with their eyes. The head-mounted camera will move with the head tilt, and the mirror helps to correct for the additional downward direction contributed by eye movements. A sample frame captured from the egocentric camera and mirror during the task is shown in Figure 4.

Each of the authors executed the eight actions of the exercise procedure ten times, resulting in our dataset of 30 egocentric videos of the same task. The authors had varying degrees of prior knowledge of the procedure, so the data consists of procedure executors of multiple levels of skill, resulting in a variety of techniques and times needed to execute the procedure. For example, one author had no knowledge of the procedure, and therefore initially relied completely on the task guidance delivered by the Glass device to preform the procedure. By the end of his tenth execution, he had internalized the procedure, eliminating the need for the assistance device, and was significantly faster than before. His set of ten videos, therefore, has him performing at a continuum of expertise levels as he increased in familiarity with the procedure. Thus, while there were only three participants included in the dataset, we believe there is still ample variety in the dataset and that testing with this data will generalize to other executors of the same procedure. Each of the actions in the video was segmented by hand to

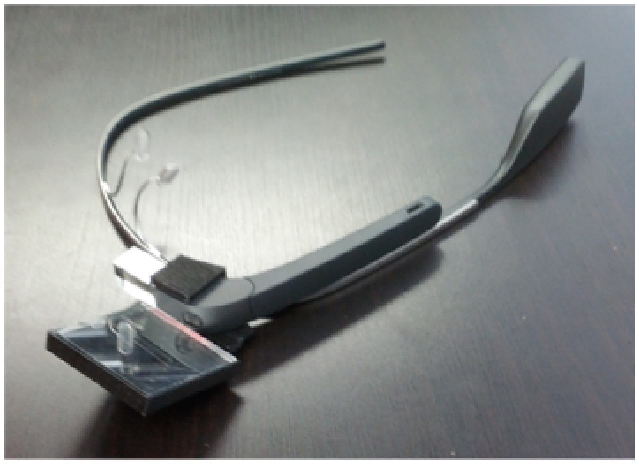

Figure 3. The modified Google Glass device we used to capture the video for our dataset. The attached mirror helps to capture manipulations of objects directly in front of the procedure executor.

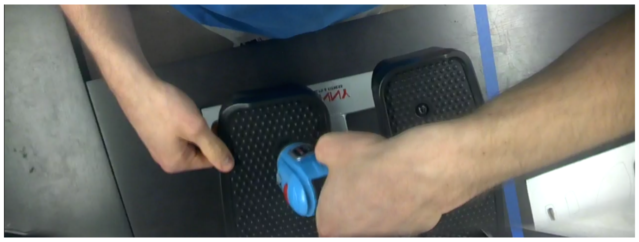

Figure 4. A sample frame captured during the disassembly task from the Glass device camera using the mirror attachment.

[1] http://en.wikipedia.org/wiki/Google_Glass

use for training and testing purposes in the classification.

4.2. Results

We evaluated our pipeline in a number of experiments using our exercise equipment disassembly dataset. In the following paragraphs we discuss the methodology used in our grid search of the pipeline parameters and the resulting accuracies.

We performed a grid search manipulating the following parameters: 1) the number of bins used in the HOOF histogram binning, 2) the size of the the codebook of HOOF features, and 3) the number of states in the HMMs. To evaluate each parameter configuration, we used a leave-one-out cross validation technique where 29 of the 30 videos were used to generate the codebook and train the HMMs. The actions of the last video were then each passed through the pipeline and the predicted action was the one which had the highest probability from the set of HMMs. The results using the Petri Network of the task used a weighting factor of 0.98. In all cases, random number generation was seeded such that k-means and the HMMs will produce repeatable results. Additionally, both of these processes were run with enough iterations to ensure convergence. To evaluate the accuracies of predicting each action for each configuration, we averaged (by action) these 30 accuracy results to determine a configuration accuracy. The results for some of the parameter configurations are shown in Table **??**.

While these accuracies are not spectacular, they are still significant in that random chance would put the accuracy at 0.125. Additionally, these results are comparable to accuracies in prior work that also use an egocentric camera [**?**]. In the next section we discuss the challenges we discovered with our technique and the implications for future methods to improve the results of real-time prediction of actions from egocentric cameras.

5. Discussion and Future Work

While we were able to see the results of our pipeline, we expect there may be strong reasons for which our classification accuracy is not high. Due to time limitations, we were not able to further explore these issues, but they may prove to be valuable as future work.

5.1. Diversity and Quantity of Training Data

Though we were able to simulate the training data that may be obtained in an actual application of our work, during our testing we found misclassification issues that could be directly attributed to our training data. In many cases, the cluster sequences chosen for a particular test action were completely dissimilar to sequences that that action's HMM was trained on. While in a real-world streamed-data setting we might expect this, in our controlled experiment it seems to indicate our model is very susceptible to slight variations.

For this reason, we think our training data may be too similar. More diverse training data may allow for less overfitting, and create looser feature-cluster associations. In the same vein we note the small size of our dataset. Because of the requirement that we have a formal model of the activity being performed, we were required to create our own. Many learning methods require a large amount of data to create accurate predictions. Future work might include expanding and diversifying the training data, in a way that will allow the learning models to better generalize from training to testing.

5.2. Low-Level Feature Representation

While results from using Histograms of Oriented Optical Flow have been significant in the previous work cited above, we were not able to reach similar results even when attempting a one-vs.-all classification for each action. While the training data limitations would affect this, we expect an issue with our feature representation.

In our unique setup with an egocentric camera, we found that head movement tends to dominate any flow features found within the scene itself. Head movement was not controlled for at all during data acquisition, and as such head movement varied significantly across actions and across videos. Because these flows were primarily capturing this movement, we may have been learning models for head movement rather than the action in the scene.

For this reason we expect more work needs to be done in feature extraction in this domain. Theoretically our learning methods are sound, and given the right representation we expect much higher results. Some work has been done by Fathi et al. [6] in the realm of egocentric video, but their implementations prove to be too slow for our activity guidance task. Future work might include incorporating object detection into the model, and using only the flow of tracked objects. Background subtraction and head movement compensation could also be explored as a means to increase the signal of the scene relative to head movement.

6. Conclusion

References

[1] J.-Y. Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5:1–10, 2001. 2

[2] R. Chaudhry, A. Ravichandran, G. Hager, and R. Vidal. Histograms of oriented optical flow and binet-cauchy kernels on nonlinear dynamical systems for the recognition of human actions. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1932–1939. IEEE, 2009. 2

[3] F.-S. Chen, C.-M. Fu, and C.-L. Huang. Hand gesture recognition using a real-time tracking method and hidden markov

models. *Image and vision computing*, 21(8):745–758, 2003. 1

[4] R. David and H. Alla. Petri nets for modeling of dynamic systems: A survey. *Automatica*, 30(2):175–202, 1994. 2

[5] R. Davidrajuh. *GPenSIM: A New Petri Net Simulator*. INTECH Open Access Publisher, 2010. 3

[6] A. Fathi, A. Farhadi, and J. M. Rehg. Understanding egocentric activities. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 407–414. IEEE, 2011. 5

[7] J. Frank. When plans are executed by mice and men. In *Aerospace Conference, 2010 IEEE*, pages 1–14. IEEE, 2010. 1

[8] J. Frank, L. Spirkovska, R. McCann, L. Wang, K. Pohlkamp, and L. Morin. Autonomous mission operations. In *Aerospace Conference, 2013 IEEE*, pages 1–20. IEEE, 2013. 3

[9] D. Kortenkamp, R. P. Bonasso, D. Schreckenghost, K. M. Dalal, V. Verma, and L. Wang. A procedure representation language for human spaceflight operations. In *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-08)*, 2008. 1

[10] D. J. Moore, I. A. Essa, and M. H. Hayes III. Exploiting human actions and object context for recognition tasks. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 80–86. IEEE, 1999. 1

[11] K. Murphy. Hidden Markov Model (HMM) toolbox for matlab. 1998. 3

[12] C. A. Petri. Kommunikation mit automaten. 1962. 3

[13] C. A. Petri. Communication with automata. 1966. 2

[14] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea. Machine recognition of human activities: A survey. *Circuits and Systems for Video Technology, IEEE Transactions on*, 18(11):1473–1488, 2008. 1

[15] H. Wang, A. Klaser, C. Schmid, and C.-L. Liu. Action recognition by dense trajectories. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3169–3176. IEEE, 2011. 2

[16] L. R. Welch. Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, 53(4):10–13, 2003. 3

[17] J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden markov model. In *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*, pages 379–385. IEEE, 1992. 1