PERCEPTION ENGINE FOR ACTIVITY RECOGNITION AND LOGGING USING MANUAL PROCEDURE INSTRUCTIONS

Patrick Beeson[1], **Nicholas Barrash**[1], **and Brian Burns**[2]

[1]*TRACLabs Inc., Houston, TX, USA*
[2]*SRI International, Artificial Intelligence Center, Menlo Park, CA, USA*

ABSTRACT

TRACLabs and SRI International are designing a vision recognition system to detect and track crew members, human activities, human-vehicle interaction, and team social interactions for ISS, EVA, and future missions like Deep Space Habitat. We call our proposed system PEARL (Perception Engine for Activity Recognition and Logging). In this paper, we present PEARL using Manual Procedure Instructions. For PEARL-MPI, we integrate a visual recognition system with a procedure execution framework and an ontology that stores the known objects, actions, cameras, and observed object locations in the environment. Using formal procedures constrains the type, location, and duration of possible activities, which improves recognition performance. It also provides a way to demonstrate cooperative human/machine logging of complex, multi-step scenarios in the visual scene.

1. INTRODUCTION

Thousands of hours of video streams exist from ISS, and countless more hours will be streamed and/or logged as various spacecraft continue to orbit the Earth and explore the solar system. These videos embed immeasurable amounts of useful data about crew social interactions, crew task performance, and crew-vehicle interaction. Currently, videos must be searched and indexed by hand to look for specific items of interest. A more suitable solution would be to automate the analysis of video streams—detecting, modeling, quantifying, and logging various significant events, both in existing footage and via real-time analysis of live cameras. Automating this process has a number of potential benefits including: reducing the man power, automatically logging recognized activities for later retrieval via queries, and providing a history of the precise locations of crew members without the need for special badges.

With support from NASA Ames, we are designing a system that detects and tracks humans, human activity, human-station interaction, and team interactions using existing videos and cameras onboard ISS. We call our proposed system *PEARL* (*Perception Engine for Activity Recognition and Logging*). We envision PEARL as a video analysis toolbox that executes at multiple levels of autonomy. First, it should allow ad hoc queries on camera feeds or video logs via a sophisticated user interface. These queries might be rather straightforward ("Find person X."). We assume many queries will be more complex, such as looking for events in a specified region of the environment ("How many times did crew member A enter area B in the last two weeks?"), which may involve on-the-fly video analysis, or even requesting video snippets of known, logged activities ("Show me the video from the last time anyone executed Procedure 1.308."). Any such logged activities would result from an autonomous layer of PEARL that is continuously analyzing video streams for specific behaviors or events and logging all relevant information.

The objectives of PEARL can be achieved by developing a suite of algorithms that can handle a variety of abstractions: tracking an individual's pose and velocity with respect to the camera, a fixed world frame of reference, or other crew members; detection and logging of events in a selected region, like near a console; detection of crew interactions; logging of events for later retrieval via a database query; either autonomous operation, or interactive operation as a video analysis toolbox. This project has several research challenges, including: robust analysis of unconstrained video content, extraction of functional descriptions of complex human events, execution of ad hoc queries involving video content, and operation at sub-real-time speeds.

The envisioned PEARL system will require the development and integration of many complex software modules, including state-of-the-art video analysis and novel user-interfaces. For our initial investigations, we are focusing on a specific application of activity recognition during *procedure* execution. Procedures are pervasive in spacecraft operations and are used to control every aspect of human spaceflight. Both ground controllers and crew members use procedures on a daily basis. Procedures are the accepted means of commanding spacecraft. They encode the operational knowledge of a system as derived from system experts, testing, training and experience. NASA has tens of thousands of procedures for the International Space Station (ISS) covering all aspects of operation, including medical operations. Most ISS procedures are executed by ground controllers who send commands to ISS and receive telemetry from ISS, but certain procedures must be run by crew members because they involve *manual instructions* (cleaning a filter, sampling noise levels, performing medical tasks, and EVA operations).

We present here *PEARL-MPI* (*PEARL using Manual Procedure Instructions*). PEARL-MPI is a subset of the overall PEARL system functionality that integrates activity recognition algorithms with the execution of procedures that contain manual instructions. We decided to focus on PEARL-MPI as our first demonstration of the potential benefits of visual activity recognition for several reasons.

1. In order to ensure schedules are on track, NASA would like to log all procedure activity, both human and automated. PEARL-MPI can supplement the human or terminal logging of certain instructions in a procedure.
2. The vast majority of crew members' scheduled time is spent executing procedures; thus, there is a large potential impact in demonstrating this particular functionality.
3. Focusing on manual instructions in procedures vastly constrains the types of activities that a deployed activity recognition system must be able to identify, allowing simpler video analysis techniques to be used during prototyping.

Section 2 provides a brief overview of the architecture for PEARL-MPI and describes relevant background information on the various components used. Section 3 then describes our initial test scenarios and illustrates preliminary results. Section 4 discusses the short term future goals of PEARL-MPI.

2. OVERVIEW OF PEARL-MPI

Figure 1 illustrates the current PEARL-MPI architecture, which has three main components that can be run on separate, networked machines. They are connected by socket interfaces and/or well-known web interfaces. Section 2.1 discusses the Procedure Management component. Section 2.2 discusses the Manual Instruction Recognition component. User Interfaces are discussed as needed and are used to show preliminary results in Section 3.3.

2.1. Procedure Management

The Procedure Management component of PEARL-MPI is responsible for keeping track of all procedures being viewed or being run by a user. Additionally, all information pertaining to currently running procedures, like telemetry values, individual instruction completion, aborted procedures, etc., get logged with appropriate timestamps by the Procedure Management component. Logs of previously executed procedures can be replayed, queried, and compiled into reports.

PEARL-MPI relies on procedures encoded in a machine-readable format. All NASA procedures are currently written in word processing software. In previous work, TRACLabs Inc. developed a *Procedure Representation Language* (*PRL*), which is an XML schema that defines a variety of tags that can be used to describe a procedure [6]. Many ISS, EVA, and medical procedures have already been converted into PRL. PRL procedures can be authored using TRACLabs' PRIDE Author tool, which uses an ontology that includes all devices (and their procedure relevant properties) on ISS. This allows quick drag-and-drop procedure authoring. Manual instructions can be entered as free-form text in PRL.

Encoding procedures in PRL allows procedure assistance software to execute automated procedure instructions in-line with a human user [1]. Additionally, PRL allows software to provide more human-friendly, interactive visualizations of procedures. TRACLabs' PRIDE View software translates procedures into HTML5 that can be rendered by any web browser on any platform.

PRIDE View uses a RESTful web interface [3] to provide a client browser with a list of procedures or the content of requested procedures in a platform-independent fashion. In addition, the RESTful interface allows connected clients to start running a procedure, to send telemetry data or entered values, to mark individual instructions as completed, to make comments on a running procedure, to log when a procedure loads another procedure, to log the activity of the user "focus" bar, etc. All information about currently running procedures is saved by PRIDE View into a MySQL database. This database can be used to replay procedures in the browser, make reports, etc.

In PEARL-MPI, we use the PRIDE View browser interface to start a procedure in the Procedure Management component. This, in turn, sends the PRL of the newly running procedure to the Manual Instruction Recognition component (see Section 2.2). During procedure execution, the Procedure Management component keeps track of all instructions marked complete by either the user, the Manual Instruction Recognition component, or (in the future) any other automated procedure execution module. When procedure update messages are received, the Procedure Management component logs the updates, along with who was responsible for the information (human user or PEARL). The human user sees all PEARL activity in the browser interface, and at any time, the user can revise incorrect completion information, make comments to be logged, or fill-in instructions missed by the PEARL activity recognition software. Similarly, any user changes to the running procedure get reported to the Manual Instruction Recognition component.

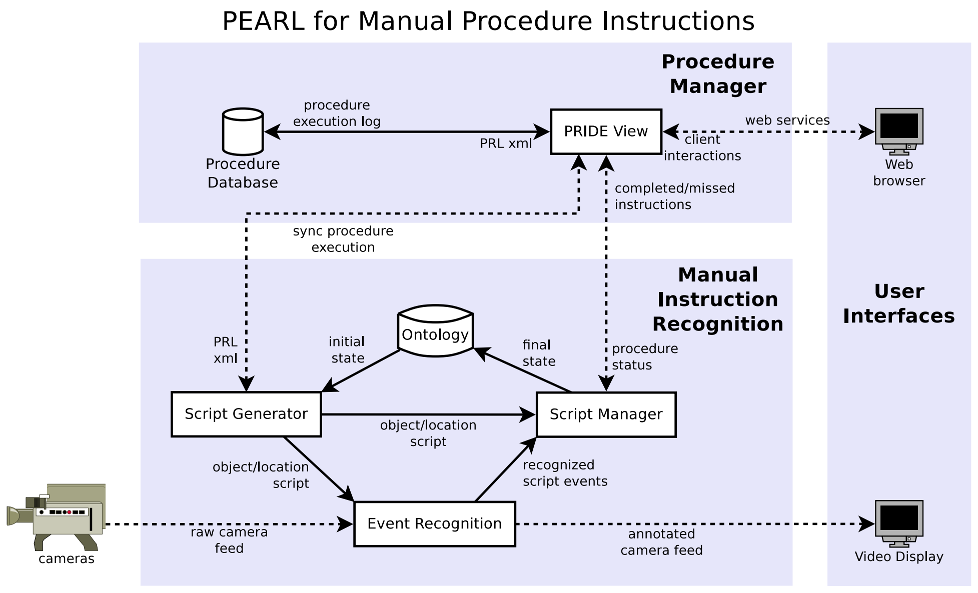


Figure 1. Architecture diagram for the PEARL-MPI prototype. There are three main components of the integrated system. Dotted lines represent data passed via various network connections, while solid lines represent data passed via shared memory.

2.2. Manual Instruction Recognition

The Manual Instruction Recognition component of PEARL-MPI is responsible for generating a complex script of expected *events* in the video stream (each event corresponds to a single manual instruction in a procedure), recognizing when these events occur, and reporting the beginning and ending times of procedure instructions associated with each event to Procedure Management (for logging and update of the user display). We use the term "events" here to distinguish the manual instruction behaviors that we expect to recognize from the types of unconstrained activities that PEARL should reason about in the future.

There are multiple, concurrently running modules in the Manual Instruction Recognize component that make all this happen. First, is the Script Generator. This module queries the Procedure Manager component for all procedures that a user is executing. If the user begins a new procedure, the Script Generator requests the raw PRL, and, if the procedure contains any manual instructions that should be recognized, it generates an *event script* from the procedure. An event script defines an event for each instruction in a procedure, which includes the actions, objects, and locations listed for manual instructions—non-manual instructions are listed in the script to ensure proper execution ordering occurs, but are not meant to be recognizable events.

Events in the script also have preconditions and postconditions pertaining to objects, locations, and cameras in the environment. This information is inferred by using an ontology that contains all the objects, actions, and locations of the manual procedures that we intend to recognize in the video stream. Figure 2 shows the current ontology, which contains translative actions that change an object's location and non-translative objects which do not change an object's location. The 3D volumes that define "locations" in the environment are known to be seen by certain fixed cameras, and (based on each camera's intrinsic calibration parameters) the appropriate u, v pixel range of each location for each camera can be derived and stored in the ontology. Additionally, some objects can contain other objects, which also allows the script generator to infer starting locations of objects that are known to be stored inside containers.

In order to map manual procedure instructions to items in the ontology, all manual instructions used by PEARL-MPI *must* be authored in PRL the using actions, objects, and locations that are present in the ontology—Natural Language Processing of free-form manual instructions is not currently an expected avenue of research for PEARL-MPI. Long-term, we expect to extend the current PRIDE Author ontology, which already contains many locations and objects of ISS, with 1) camera information, 2) a set of valid actions that might be recognizable from video analysis, and 3) the initial expected locations of all objects manipulated by manual instructions. Currently, we use the small testbed ontology described below in Section 3.1.

The event script, which describes how ordered behaviors by the user should change the locations of objects in the environment, is passed to both the Script Manager and to the Event Recognition module. Our system handles multiples procedures at a time, by creating multiple instances of the Script Manager and Event Recognition modules to be spawned—one to track each running procedure that has manual instructions with recognizable activities.

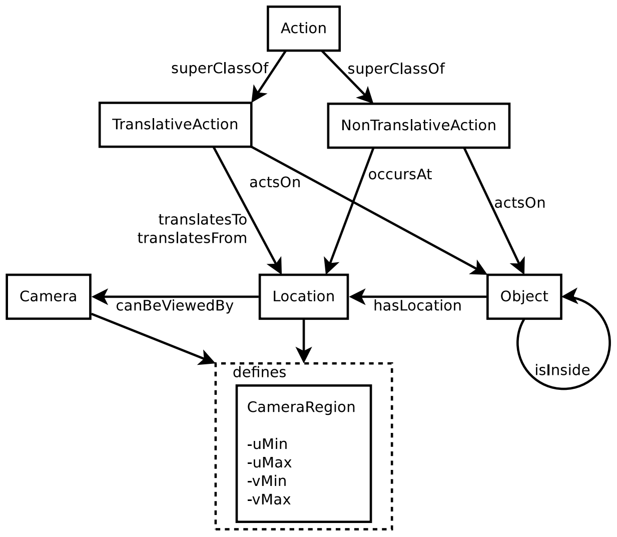


Figure 2. We augmented the traditional procedure authoring ontology of objects and locations with actions that can be performed on objects, containment for objects, fixed cameras that can view locations, and camera calibration parameters. This ontology is used to create event scripts for manual instructions that seed the event recognition system with types of objects that will be used and their expected starting and ending locations.

The Event Recognition module uses the associated u, v regions of interest for each camera stream, along with motion profiles associated with the recognizable activities to watch for recognizable events. Whenever an event is thought to have occurred, the event id, the recognition confidence, and the start/end times of the event are passed to the Script Manager. The algorithms used here will result in the largest area of future research on PEARL. Section 3.2 details the current algorithms used in our initial prototype.

The Script manager has three roles, first it receives recognized events from the Event Recognition module and any status updates about the procedure execution (like user completed instructions) from the Procedure Manager. Using these two sources of information, the Script Manager next decides whether the event really did occur and, if so, it sends this information to the Procedure Manager. How exactly the Script Manager should combine script order, procedure state, and confidence-weighted event recognition events is ongoing research. Currently, the Script Manager simply believes any recognized event that corresponds to a future instruction in the procedure, regardless of confidence. It does, however, inform the Procedure Manager of skipped instructions, which in turn show up as comments in the users' visualization of the procedure. These comments are saved to the database by the Procedure Manger as part of the procedure execution log. Finally, given the assumption that an event recognized by the Event Recognition module properly matches all pre-conditions and post-conditions in the event script, updated object locations are stored in the ontology.

- TranslativeAction
  - *moveObject*
  - *remove*
- NonTranslativeAction
  - *turnOn*
- Object
  - *box1*
  - *console1*
  - *object1*
- Location
  - *boxFloorLocation1*
  - *consoleLocation1*
  - *worktable1*
- Camera
  - *camera1*
  - *camera2*
- CameraRegion
  - CameraRegion(camera1, boxFloorLocation1) == *camera1boxFloorLocation1*
  - CameraRegion(camera1, consoleLocation1) == *camera1consoleLocation1*
  - CameraRegion(camera1, worktable1) == *camera1worktable1*
  - CameraRegion(camera2, boxFloorLocation1) == *camera2boxFloorLocation1*
  - CameraRegion(camera2, consoleLocation1) == *camera2consoleLocation1*
  - CameraRegion(camera2, worktable1) == *camera2worktable1*
- *Starting values*
  - hasLocation(box1) == boxStoredLocation1
  - hasLocation(console1) == consoleLocation1
  - isInside(object1, box1) $\rightarrow$ (hasLocation(object1) == hasLocation(box1))

Table 1. Our testing ontology.

3. CURRENT RESULTS

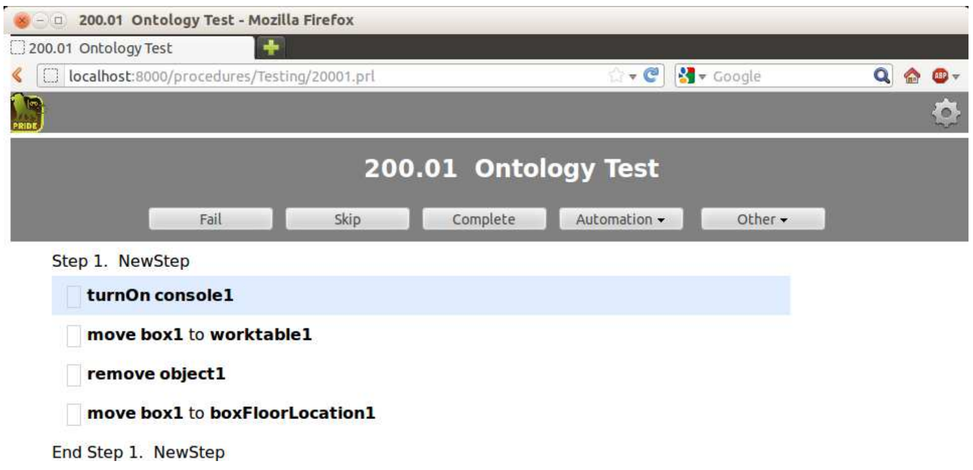
We demonstrate preliminary results of the integration work discussed above. Video demonstrating PEARL-MPI testing can be found at `http://www.traclabs.com/%7Epbeeson/PEARL/`.

3.1. A Simple Manual Procedure

We created a preliminary test procedure that contains all manual instructions. The procedure deals with three objects, three actions, and three locations. The instantiated ontology, based on the layout illustrated in Figure 2 is shown in Table 1. Using this ontology, and the PRIDE Author tool, we created a simple PRL-based procedure. The PRIDE View browser visualization of this simple procedure is shown in Figure 3.

3.2. Event Recognition

Our prototype system attempts to observe events that can be physically defined as an object or a person moving be-


200.01 Ontology Test - Mozilla Firefox
200.01 Ontology Test
localhost:8000/procedures/Testing/20001.prl
Google
PRIDE

200.01 Ontology Test

Fail | Skip | Complete | Automation ▾ | Other ▾

Step 1. NewStep

- turnOn console1
- move box1 to worktable1
- remove object1
- move box1 to boxFloorLocation1

End Step 1. NewStep

Figure 3. Our test procedure. This procedure involves three actions on three objects.

tween regions of interest. This is a general tool that can be used to acquire evidence of activities like people entering and exiting a room, an object being moved from storage to a work area, an object being placed in a container, or someone reaching for something (for example, switching on a machine). In this last example, the system would observe something moving from the body region to the on-switch region. Other event detection methods, such as recognizing learned spatiotemporal patterns associated with specific actions (walking, hammering, sweeping) [7, 5, 12] will expand our current capability, but these methods require extensive example collection and training. Additional events could be covered by observing specific types of state changes, such as lights being switched on or doors being opened.

To observe an object moving from one region to another, it is important to acquire evidence of two things: 1) that the *same* object has traveled from the first region to the second one (not simply one object disappearing in the first region and some other appearing in the second), and 2) that the object is not just passing through these regions, but is actually starting from the first region and coming to rest in the second one (otherwise, traffic in the room will be constantly triggering event detections). We determine that the appropriate motion has taken place by tracking points on the objects. The initial algorithm is as follows.

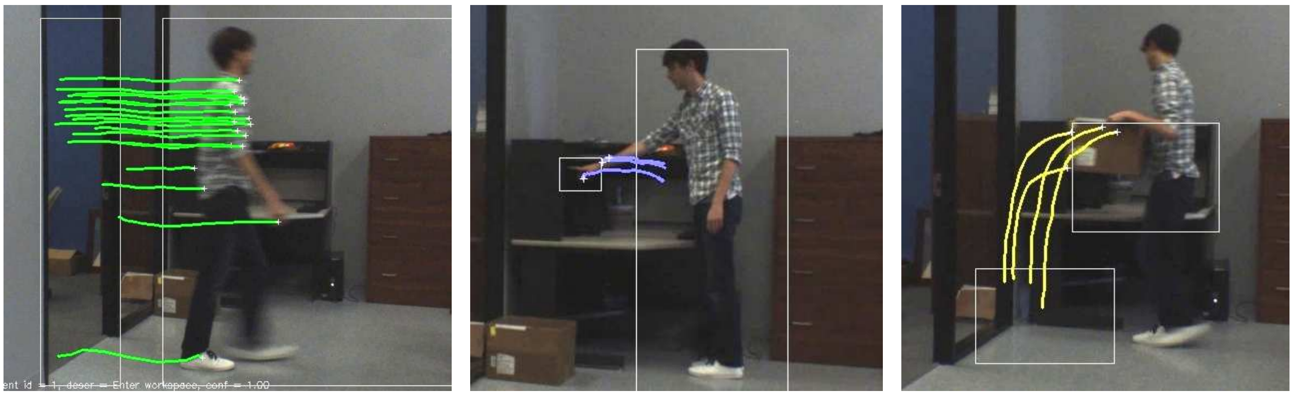
1. Interest points that are both stable for tracking and densely cover the scene are detected. This can be done efficiently using the FAST corner operator [9].
2. Points are tracked across time using the Lucas-Kanade tracker [2]. To reduce spurious tracks, tracking is performed forwards and backwards. Inconsistent tracks are removed.
3. Point tracks that are consistent with motion between two regions of interest and have similar trajectories are grouped together.
4. Any point track bundle that starts near the first region and stops near the second, without overshooting either region significantly, is considered evidence of an object moving between the regions.

This algorithm is relatively straightforward, yet works remarkably well. We have observed no false positives in our current set of sample videos. We can see false negatives when users remove objects from containers and place them behind the original container. This is due to the container occluding the view of the objects placement. Figure 4 shows examples of position-changing events detected by our system. They include a person entering a room, a hand reaching to turn on a console, and a box being picked-up and placed on a work station.

3.3. Procedure Execution

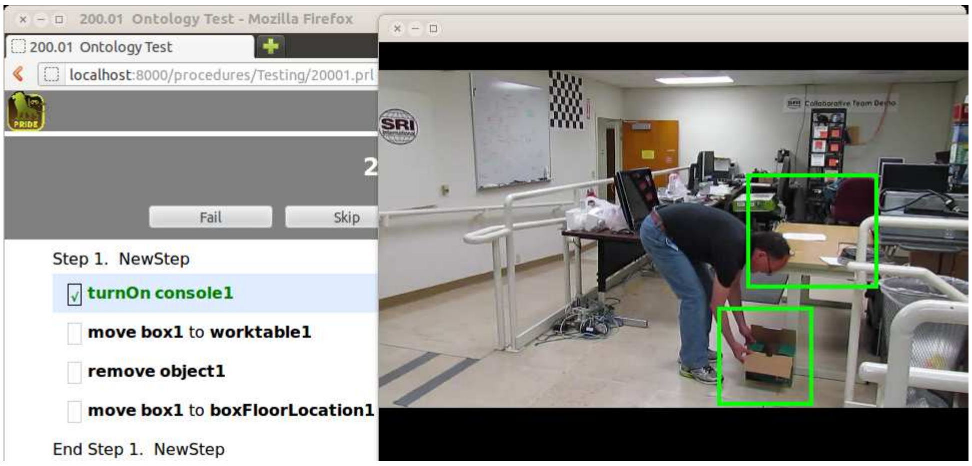
As the procedure executes, Procedure Management gets occasional messages from the Manual Instruction Recognition component saying that certain manual instructions have completed. Figure 5 shows a snapshot near the beginning of the procedure execution based on video taken at SRI. Here, the Event Recognition software has already correctly identified that the turnOn console1 instruction completed (this event is triggered when the users arm moves into the location at the front of console1 and then moves away). The annotated video feed shows the pre- and post-locations of the next expected event in the script, which is the move box1 instruction.

Sometimes, events might be skipped. Figure 6 shows the visualization at the end of this procedure when the Event Recognition module does not recognize the remove event. The Script Manager detects that an event (thus manual instruction) was skipped and alerts the user of this via comment to the Procedure Manager. This comment is also logged in the MySQL database pertaining to this procedure execution instance. The user can manually mark this instruction as completed if they know the manual instruction was indeed performed. Alternatively, the user can simply mark the procedure as completed/aborted without making any changes to the instruction. If this happens the procedure is logged as completed/aborted, but that instruction remains marked as possibly skipped in the execution logs.



(a) User entering work area (b) Turning on a console (c) Moving a box from floor to a work area

Figure 4. Examples of current events detected by the Event Recognition module. Users entering and exiting the work area are implicit and not explicit in procedures.



200.01 Ontology Test - Mozilla Firefox

200.01 Ontology Test

localhost:8000/procedures/Testing/20001.prl

Fail Skip

Step 1. NewStep

√ **turnOn console1**

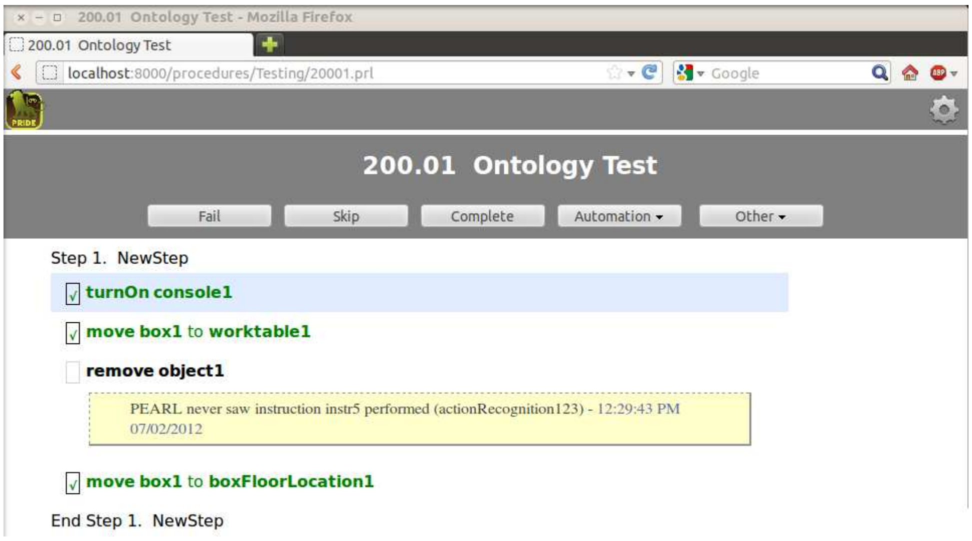
move box1 to **worktable1**

remove object1

move box1 to **boxFloorLocation1**

End Step 1. NewStep

Figure 5. Snapshot of progress in recognizing manual procedure instructions using camera1 *from the ontology. At this point, the Event Recognition module has correctly identified that the* turnOn *action was performed, and is showing the user the areas of the video it expects to see change during the next scheduled instruction.*



200.01 Ontology Test - Mozilla Firefox

200.01 Ontology Test

localhost:8000/procedures/Testing/20001.prl

200.01 Ontology Test

Fail Skip Complete Automation ▾ Other ▾

Step 1. NewStep

√ **turnOn console1**

√ **move box1** to **worktable1**

remove object1

PEARL never saw instruction instr5 performed (actionRecognition123) - 12:29:43 PM 07/02/2012

√ **move box1** to **boxFloorLocation1**

End Step 1. NewStep

Figure 6. View of the procedure execution after PEARL-MPI detects that the last instruction was completed. Upon seeing a future event occur, the Script Manager logs a comment on the skipped instructions to alert the user or ground control that an event it believes the Event Recognition system should identify was not seen.

Figure 7 shows the summary screen where procedures can be launched, where all running procedures are summarized, and where completed/aborted procedures are shown. Completed procedure logs can be replayed in the PRIDE View visualization, with all appropriate instructions marked off at the relative times and all telemetry data (where appropriate) filled in. Similarly, logged video streams are replayed for procedures examined by PEARL-MPI.

3.4. Multiple Cameras

On ISS, there may be multiple cameras that have overlapping fields of view. In such scenarios, the Event Recognition system, should use multiple camera stream to obtain the highest confidence recognition possible. Currently, we only use a single image stream at a time; however, by using the ontology, PEARL-MPI does know what cameras can view different locations. For now, we allow the user to choose what camera feed to use by popping up a window on the User Interface computer. The Script Generator handles this, as the script has different location values for different cameras. This allows us to test multiple videos of people in different locations performing the same procedure. Figure 8 shows a video of a user at TRACLabs performing our same simple procedure, but using different cameras, lighting, and directions of motion/bearing. Due to our limited vocabulary of recognizable events, we get the same results as above—all instructions except the remove (where the containing object occludes placement of the removed object) are correctly observed.

4. FUTURE WORK

We have presented an initial prototype of PEARL-MPI that connects all the relevant components shown in Figure 1. This prototype demonstrates how automated recognition of manual procedure instructions, where the user performing the procedure is away from any console, can enhance procedure logging accuracy and potentially alert the user to incorrectly performed procedures, including improperly ordered or skipped instructions.

Now that the full integration of PEARL-MPI modules is complete, we expect to fill out some of the algorithms that need more research. In particular, this includes extending the Event Recognition module and the Script Manager 1) to handle a larger vocabulary of events, 2) to reason about potential false positives/false negatives given the current procedure state, and 3) to handle user reactions to PEARL alerts, like retrying an instruction or denoting any false positives/negatives that may still be present in the system.

Specifically, we expect to augment the current algorithms used for Event Recognition. In addition to person tracking and looking for motion into and out of specific regions of the video stream for object tracking, we plan to track objects based on appearance models [10]. Also, adaptive background subtraction [4, 8] will be used in order to filter out the kinds of repetitive or non-important motions that are often seen in ISS and other zero-G videos. Finally, because PEARL-MPI has both objects that are manipulated by people and an action language associated with the objects, we expect to leverage recent work that exploits language to learn improved activity recognition categorization [11].

ACKNOWLEDGMENTS

This research is funded by NASA contract NNX12CD37P. The authors would like to thank Dr. Bob Bolles of SRI International for his past and future help in designing an event recognition system for PEARL. Also, TRACLabs' programmer Scott Bell and intern Brian Lemke were responsible for much of the PRIDE View programming needed for the PEARL-MPI prototype.

REFERENCES

[1] S. Bell and D. Kortenkamp. Embedding procedure assistance into mission control tools. In *IJCAI Workshop on Artificial Intelligence in Space*, 2011.

[2] J.-Y. Bouguet. Pyramidal implementation of the lucas kanade feature tracker description of the algorithm, 2000.

[3] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[4] P. KadewTraKuPong and R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In *Proceedings of the European Workshop on Advanced Video-Based Surveillance Systems*, 2001.

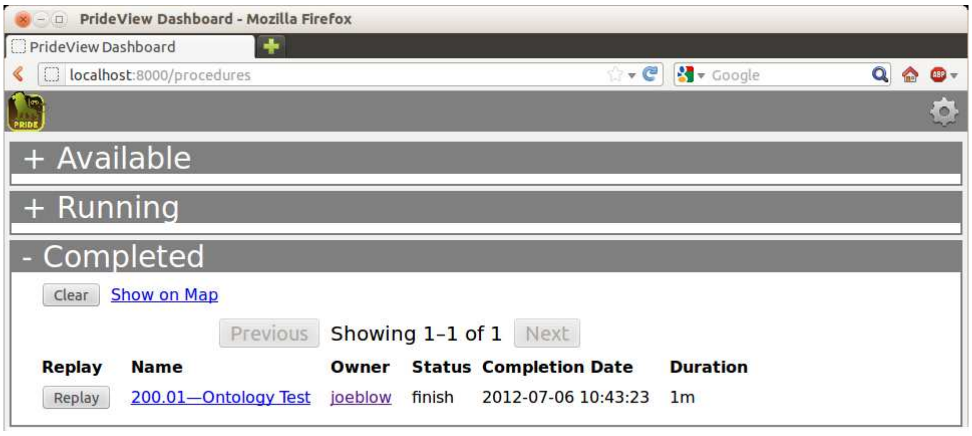
[5] A. Kläser, M. Marszałek, and C. Schmid. A spatio-temporal descriptor based on 3D-gradients. In *British Machine Vision Conference*, 2008.

[6] D. Kortenkamp, R. P. Bonasso, and D. Schreckenghost. A procedure representation language for human spaceflight operations. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2008.

[7] I. Laptev and P. Pérez. Retrieving actions in movies. In *IEEE International Conference on Computer Vision*, 2007.

[8] L. Li, W. Huang, I. Y. H. Gu, and Q. Tian. Foreground object detection from videos containing complex background. In *Proceedings of the ACM International Conference on Multimedia*, 2003.

[9] E. Rosten, R. Porter, and T. Drummond. FASTER and better: A machine learning approach to corner detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:105–119, 2010.


PrideView Dashboard - Mozilla Firefox

PrideView Dashboard

localhost:8000/procedures

+ Available

+ Running

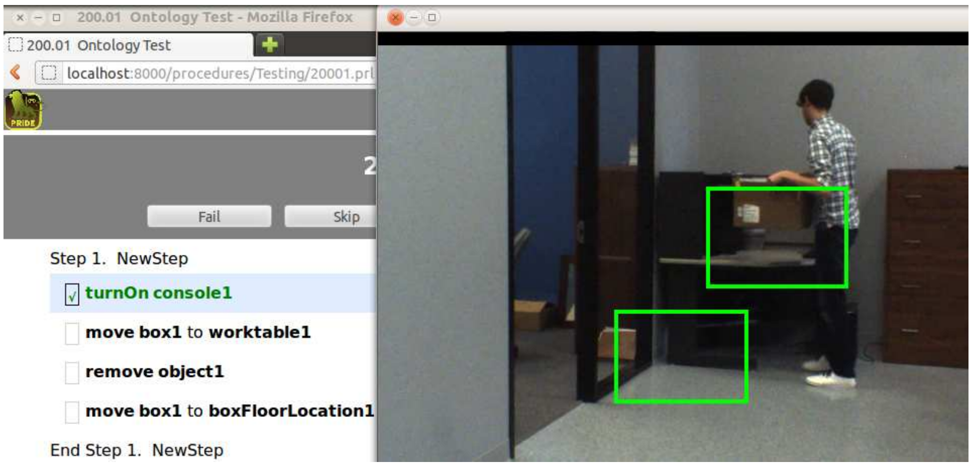
- Completed

Clear Show on Map

Previous Showing 1–1 of 1 Next

Replay	Name	Owner	Status	Completion Date	Duration
Replay	200.01—Ontology Test	joeblow	finish	2012-07-06 10:43:23	1m

Figure 7. The PRIDE View summary screen shows the current changes to the procedure execution database that the user has made.


200.01 Ontology Test - Mozilla Firefox

200.01 Ontology Test

localhost:8000/procedures/Testing/20001.prl

Fail Skip

Step 1. NewStep

✓ turnOn console1

move box1 to worktable1

remove object1

move box1 to boxFloorLocation1

End Step 1. NewStep

Figure 8. Another snapshot from execution of the same procedure but using a different camera in a new location. This video corresponds to camera2 *in the ontology.*

[10] P. M. Roth and M. Winter. Survey of appearance-based methods for object recognition. Technical report, Institute for Computer Graphics and Vision at Graz University of Technology, Austria, 2008.

[11] C. L. Teo, Y. Yang, H. D. III, C. Fermüller, and Y. Aloimonos. Towards a Watson that sees: Language-guided action recognition for robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2012.

[12] M. yu Chen and A. Hauptmann. MoSIFT: Recognizing human actions in surveillance videos. Technical Report CMU-CS-09-161, School of Computer Science, Carnegie Mellon University, 2009.