Project 16:
Da Vinci Intelligent Surgical Assistance
Final Report

Chris Paxton (cpaxton3@jhu.edu)
Mentors: Jonathan Bohren, Kelleher Guerin, Prof. Greg Hager

May 9, 2014

1 Introduction

The purpose of this project is to develop the tools and systems necessary to experiment on methods for learning from demonstration that can be applied to real-world robotic and human/robot collaborative tasks. It is increasingly clear that the workers of the future will perform their jobs alongside specialized robotic systems. This extends to medical tasks such as surgery: the success of Intuitive Surgical's Da Vinci is evidence of the potential of applied robotics. However, many current systems are purely teleoperated, meaning that the users have to explicitly control the motions of the robot. This wastes some of the potential value of bringing robots into the workplace, since it means that repetitive or well-understood segments of tasks cannot be automated.

This technology would produce valuable results that could improve surgical performance by decreasing the cognitive load on the surgeon. The Da Vinci is normally operated by a trained surgeon who controls two manipulator arms at a time, each of which can be outfitted with specialized tools. The operator can clutch to control either the camera or a third manipulator arm. This system offers plenty of potential for partial task automation. If the operator is suturing shut an incision, they may use one manipulator arm to drive the needle through the tissue and another to grab the end of the needle and pull away from the patient. While driving the needle through tissue is a difficult task that depends on the particular circumstances of the task, pulling the needle away is relatively simple and thus might be well suited for automation; this would decrease the necessary concentration and allow the surgeon to focus on the high level task. Another example is in knot tying, wherein the surgeon needs to use a third tool to snip a thread after a knot has been tied.

An ordinary procedure might look like this:

1. Surgeon ties knot with arms 1 and 2.
2. Surgeon switches control to arm 3.
3. Surgeon moves arm 3
4. Surgeon cuts the thread with arm 3.
5. Surgeon switches back to controlling arms 1 and 2.
6. Surgeon moves on to the next task.

This procedure may have to happen many times within a single surgery. It breaks up the flow of the surgical performance, and takes the surgeon's mind off of the task and off of the patient. A partially automated version would be much simpler:

1. Surgeon ties knot with grippers 1 and 2.
2. Surgeon indicates where the thread should be cut and waits.
3. Surgeon moves on to the next task.

In this case, the surgeon never needs to relinquish control of the first two arms. In order to quickly test and develop technologies to accomplish this sort of intelligent assistance, I set out to develop simulations of a variety of different manipulation tasks and to design the algorithms that would allow intelligent human/machine collaboration to assist the completion of these tasks. The first task I targeted was a simple peg transfer task, based on a similar exercise used to train surgical roboticists.

2 Background

This work targets two main robotic systems: a pair of Barrett WAM arms and the Intuitive Surgical Da Vinci. Both of these systems are used in a variety of interesting tasks. In particular, I wanted to be able to build a *generative* model of an arbitrary task, not merely a *discriminative* model.

The Language of Surgery project at JHU has collected a large amount of surgical data used for skill classification and for providing feedback to surgeons in training [9, 1]. This includes video data, as in Fig. 1, and kinematics data. It does not include annotations for objects of interest such as the needle, thread, and insertion points. These would Recent work has looked into automatic segmentation of video and kinematic data from these procedures.

In addition, work done by Sebastian Bodenstedt and Nicolas Padoy under Professor Hager used a Gaussian Mixture Regression model to learn when to rotate a ring during a wire walking task [2]. Other unpublished work by Dr. Padoy partially automated certain surgical tasks with the Da Vinci, but this automation was not very robust or generalizable to different scenarios.

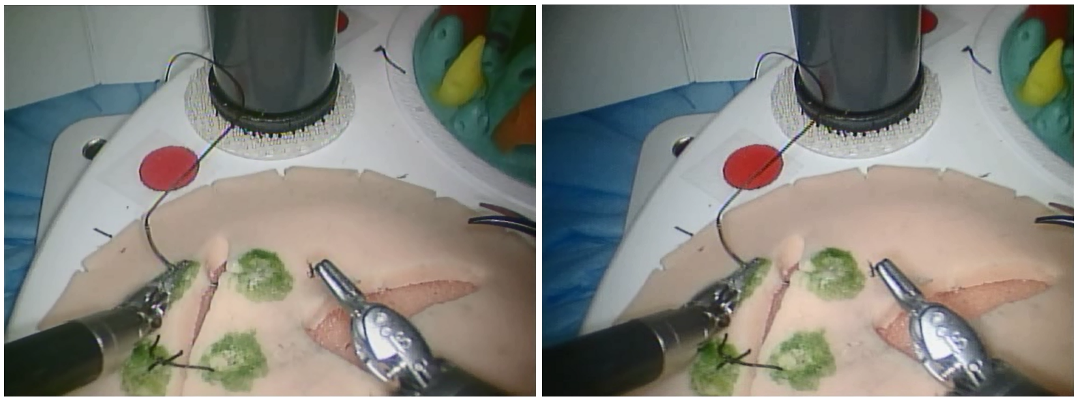


Figure 1: Example of video data from the Language of Surgery data set.

I looked into a few different approaches for modeling the motions performed during a given task. Earlier work from CIRL [2] used Gaussian Process regression models to assist teleoperation of a simple wire-following task. Other work submitted to IROS used a set of demonstrations to learn Another option I have been interested in looking into is the use of Gaussian Process Latent Variable Models (GP-LVM), which have been extensively used in areas such as human pose estimation [3]. Others have proposed a simple approach, in which a deformable registration is computed between a demonstration and test environment and this transformation is applied to the demonstrated trajectory [8]. This approach has the advantage of being straightforward, but does not generalize well to substantially different environments and does not learn which features are important to reproduce a trajectory.

Of particular interest to me is inverse optimal control (IOC), which computes an optimal control reward function determining which actions will be taken. As described in previous work on maximum entropy inverse optimal control [11], the key formula given in Eq. (1) says that the probability of taking an action u_0 at a given state s_0 is proportional to the sum of the expected future rewards, divided by a normalizing term Z. This term Z is the sum of all future expected rewards from all possible actions.

$$P(u_0|s_0) = \frac{1}{Z}\sum_t r(s_t, u_t) \tag{1}$$

As one might imagine, computing Z is very expensive. One of my goals for this project was to decide on an approach for learning from demonstration that could be practically applied to a simulated or real robot with many degrees of freedom.

In order to develop a simulation, I needed to settle on a specific teleoperation task to perform. I decided to focus on simulating a peg transfer task, where the user picks up a ring from a peg with one arm, hands it to the next, and puts it back down on another peg. This is based off an exercise surgical robotics trainees perform in order to improve their dexterity and skill with the Da Vinci; in my case, it works well because neither arm can accomplish the task alone.

3 Algorithm Design

The first goal of my project was to research and decide on an approach for applying learning from demonstration to the task of human/machine collaboration with real (or realistically simulated) robots for a peg transfer task.

Recent work in continuous, locally optimal IOC has proposed a more efficient method for solving the IOC problem [5]. Instead, they locally approximate the reward function with the Laplace approximation, modeling it as a Gaussian log likelihood with gradient g and Hessian H, where n is the dimensionality of the state/action space:

$$\mathscr{L} = \frac{1}{2} g^T H^{-1} g + \frac{1}{2} \log|-H| - \frac{n}{2} \log 2\pi \tag{2}$$

This is combined with a Gaussian kernel function, describing which environmental features are important. The kernel log likelihood is given by Eq. (3), where K is the Gaussian covariance matrix such that $K_{ij} = k(f^i, f^j)$, and λ and β are the parameters of the Gaussian kernel function k given in Equation (4). The set of features at a given state s_i are represented by f_i out of the set of all features F.

$$\log P(y, \lambda, \beta | F) = -\frac{1}{2} y^T K^{-1} y - \frac{1}{2} \log|K| + \log P(\lambda, \beta | F) \tag{3}$$

As Eq. (3) is a gaussian process, y is the hidden true reward function, λ the Gaussian kernel weights, and β is the covariance of the Gaussian distribution. The parameters y, β, and λ can be jointly optimized to solve the problem. Derivatives and other formulas are outlined in the paper [5].

$$k(f^i, f^j, \lambda, \beta) = \beta \exp\left(-\frac{1}{2} \sum_k \lambda_k \left[(f^i_k - f^j_k)^2 + 1_{i \neq j} \sigma^2\right]\right) \tag{4}$$

In this case, the features u at any point s would be the difference in position and orientation between the arms and the objects in the environment: the pegs, the ring, magnetic bars, etc. In addition, one useful feature would be the relationship between the ring and the pegs. After all, the ring must be oriented correctly to fit onto a peg.

The authors of this code have posted their MATLAB code online; I was able to run their tests. After I have integrated my simulation with the ROS MATLAB plugin, I can attempt these experiments myself.

3.1 Sub-Task Segmentation

Another of this projects' initial goals was to determine how to model different components of a task in a collaborative setting. While inverse optimal control is an excellent way to learn how to generate a trajectory in a new environment, it does not solve structured tasks like the ones I am interested in. As such, I divide the task into segments. There are four segments in the peg transfer task I am looking at:

1. Arm 1 picks up the ring, lifting it directly up off of the first peg.
2. Arm 1 moves the ring to the center of the test area.
3. Arm 2 moves to the center of the test area and grabs the ring.
4. Arm 1 releases the ring.
5. Arm 2 moves the ring so that it catches on the second peg, and lets go.

Each of these sub-task segments has slightly different behavior, and a clear beginning and end point. The easiest way to segment what actions should take place relies on the fact that a human operator is intended to be in the loop in every scenario: instead of the user pressing a button to take control of the second arm, they will press a button to say that the second arm should now act. During the peg transfer task, this means that pressing the button would play a trajectory that takes the second arm to a ring held by the first arm, closes the gripper, and then releases the ring around the second peg. Segments would begin or end at any button press: either a grasp or switching arms. This approach would work best if I was replaying a demonstrated trajectory, as proposed by some related work [8, 4]. I plan on implementing something like this first.

Another option is more complex, but also more interesting and generalizable. In this case, I would learn a Gaussian Mixture model to classify which of the different segments was being performed at any given time. This classification would take place based on which arms are moving, how much they are moving, and how they are moving in relation to the environment. Then, I could solve a separate IOC problem for each segment of the task to learn a reward function governing trajectory generation.

4 Simulation Development

My simulation is based off of work by Jon Bohren, who has been developing tools for integrating ROS and Orocos RTT, open-source software for real-time robot control. However, there were a number of problems and this did not work "out of the box" as I had been hoping.

After implementing an inverse kinematics controller, I found that I could not start both robots at once without Gazebo crashing. This problem was because of a race condition in the Orocos Gazebo plugin, the software which executes joint commands on the simulated robot. I also needed to modify the ROS launch files and Barrett URDF description in order to start two arms in different locations, running with their own separate controllers.

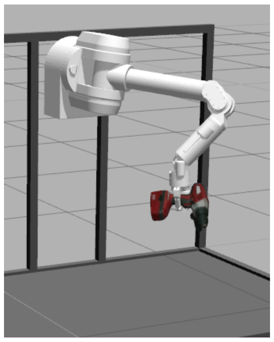


Figure 2: Testing the integral gains to allow the arm to pick up objects.

When I finally had the simulation running, I had to tweak the PID controller settings and friction coefficients to allow the arms to pick up objects. A PID (proportional-integral-derivative) controller calculates position error over time and compensates for it; a poorly configured controller would struggle to pick up even a very light object. I tested the arm until it could reliably pick up a range of different objects in simulation, as per Fig. 2.

4.1 Collaboration Tasks

One early goal was to build a toolkit for learning from demonstration research. There are two parts to this: the software to run the robot and the set of tasks. I build a package called `lcsr_collab` containing a number of different example tasks:

- Transfer a ring from one peg to another
- Take apart and reassemble a square constructed of magnetic blocks
- Perform some motions in free space relative to 2-4 markers representing arbitrary objects

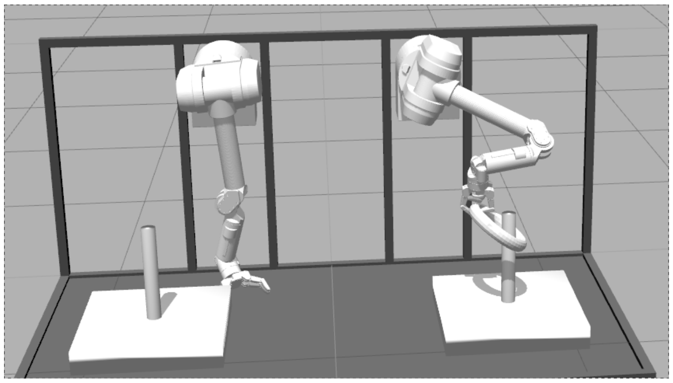


Figure 3: Example of the transfer task: object needs to be moved from right peg to left peg.

I created a set of URDF models to go alongside these tasks. The first task, transfering a ring from one peg to another, can be seen in Figure 3. I also wrote a plugin package `lcsr_construction_plugin` which publishes TF coordinate frames from Gazebo for all links in an URDF model. My trials use this package to publish the necessary information to record features and to show object positions in the ROS `rviz` GUI.

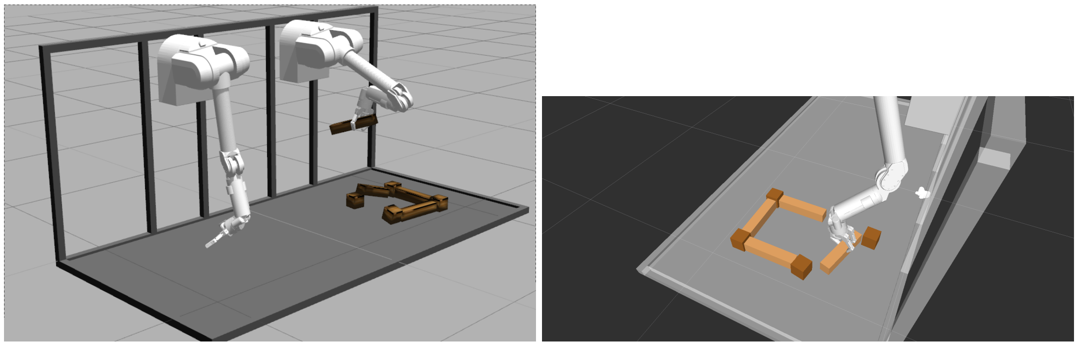


Figure 4: ROS construction task, shown in Gazebo (left) and Rviz (right)

The `lcsr_construction_plugin` package also allows a user to configure magnetic joints between different components, and has an option to ”teleport” components into position if they are close enough together to simulate latching into place. Otherwise, a simulated joint will latch the two objects together if they are within a certain radius of the correct position. Joints are configured such that they do not allow freedom of motion, but if torque on one of these joints exceeds a threshold set in the model file it will detach and the components can move independently again. The package allows you to construct arbitrary shapes in URDF files out of Xacro macros for end joints and connecting bars, but I tested it with a simple square shape, as shown in Figure 4.

The simulated robots are controlled by my `lcsr_spacenav` package. A client node runs in the background, listening to messages from the 3DConnexion Space Navigator mouse. It publishes two coordinate transforms storing the linear and angular coordinates of the desired position of the end of each arm. The client also publishes messages indicating the desired state of the gripper: open or closed. The definition of ”closed” can be fixed from custom ROS launch files and the command line to suit the task. For example, in the peg transfer task the Barrett hand’s fingers can wrap around the ring, so it closes all the way. In the construction task the fingers do not close all the way, or it would press down on the blocks too hard and drop them. When the left button is pressed, the node switches which arm the user is controlling, and when the right button is pressed it toggles the state of the gripper for that arm (open or closed).

4.2 Feature Collection

Once the simulation was up and running, I needed utilities for collecting observations and replaying them.

I built a package called `lcsr_replay` that allow users to specify a list of coordinate frames as reference points and another list as features. It can record and replay into `rosbag` files, which are a standard format associated with a number of different tools in ROS.

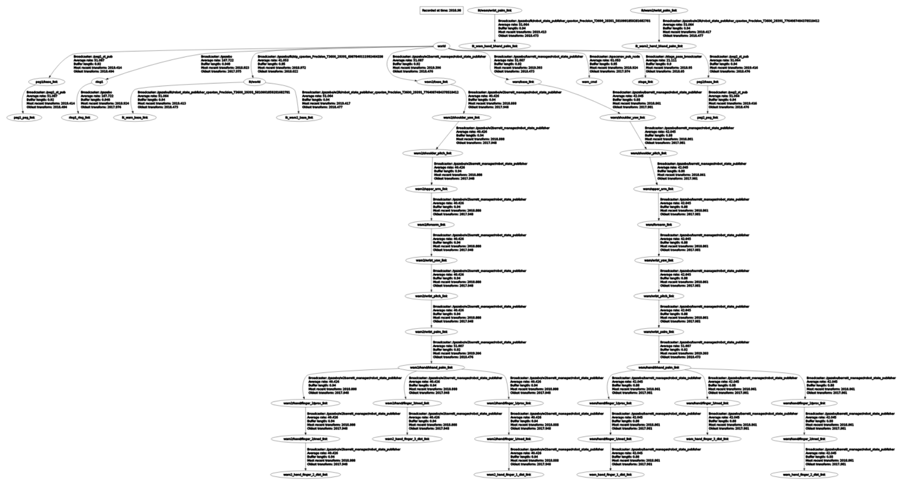
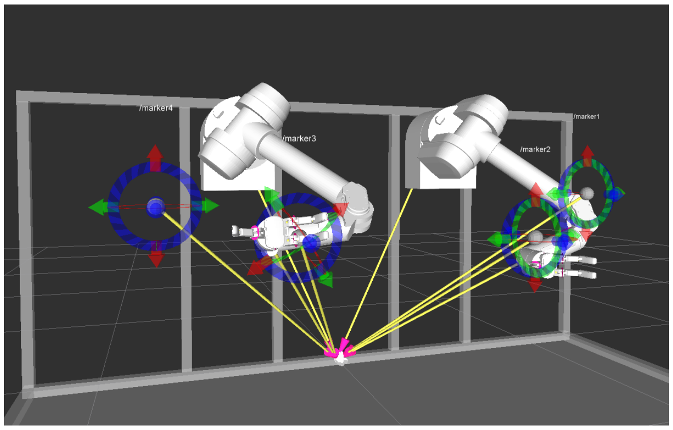


Figure 5: Tree relating the positions of different objects in the peg transfer task.

In my ROS demos, everything is associated with a coordinate frame: the base of the peg stand, the ring, the peg itself, each joint in the robot arm, and the stage the robot is standing on. This tree is large and very complex, but completely describes the state of the world, as per Fig. 5. The relative positions of objects in the environment to the arms and to one another are assumed to be the features determining how an action is performed.



I used the `lcsr_replay` package with the marker test to see if my early simulation could replay a trajectory in a slightly different environment. The test environment is shown in Fig. 4.2; I simply moved each arm back and forth between the two markers in front of it, and then moved the markers and replayed the test. I used OpenCV to compute an affine transformation between the locations of markers in a demonstration trial and in a test environment. You can run the `registered_b2play` utility in this ROS package to do this using any saved demonstration.

5 Conclusions

This project has resulted in a useful toolkit for simulating a variety of collaborative tasks involving two arms. This has put me in a good position to continue my work and bridge the gap between the approaches I examined for robot learning from demonstration and the real world. I can currently save and analyze stored trajectories and replay them with the simulated WAM arms.

While my early experiments with affine transformations were unsuccessful, I am optimistic that I can improve them with more complex algorithms; I was not able to test any inverse optimal control or machine learning based code on my simulated robots, but I have a plan and I have been able to walk through the tasks outlined using straightforward controls. I would consider this project a success in that it has set me up to succeed in future research.

6 Project Management

The overarching goal of this project was to get acquainted with a system on which I could quickly develop and test different approaches for learning from demonstration, as applied to human/robot collaboration. These were ideally a group of tasks where a human sets high-level goals and does some difficult components, with automated systems acting as an assistant where possible. I completed the background research, planning, and simulation development phases of this project, but I still need to integrate the actual learning from demonstration algorithms with my code and move on to testing with real robots.

6.1 Accomplished vs. Not Accomplished

I spent a good amount of time early on in the semester working on my early goals and processing Da Vinci data. I was able to use the large Language of Surgery data set, but registration between

My minimum deliverables involved coming up a plan for detecting different segments of a task and for learning from demonstration; I outlined how I intend to do this above. I was also able to get a simulation together, and have established the tools needed to collect features, record demonstrations, and control the robot automatically, even if the algorithms are not yet finished. I did not finish automation of the peg transfer task, but this should be accomplished soon.

Midway through the semester, it became increasingly apparent that getting a simulation of multiple robots capable of collaborating with one another was increasingly important. As a result, I needed to delay work on the actual Da Vinci. Working with the Da Vinci opens up a whole host of issues: its vision system needs to be worked out, it interacts with an unpredictable deformable environment, and as it is more challenging to simulate I need to program on a physical robot and compete with others for time. While early in the semester I worked on processing and analyzing Da Vinci data from the Language of Surgery project, I moved away from developing learning from demonstration tools based on this data.

In addition, I was unfortunately not able to make enough progress to actually run the IOC algorithms I was interested in on the WAM arms. A majority of my time was spent getting the simulation working, with the various tools I need to develop and test approaches for learned human/robot collaboration. Many of my early deliverable goals focused on coming up with ideas and background research into what might be feasible while I determined how to proceed with developing a testbed system for these ideas.

I was hampered in this project by a vague plan and lack of direction at times. Having a vague plan was somewhat necessary because I saw this project largely as a way to get introduced to the systems and tools I need to continue work on human/machine collaborative systems over the coming months. Additionally, I did not have a clear picture as to what system I should use and what systems our lab would converge on in this semester in the way that I do now.

6.2 What I Learned

Before this project, I do not believe I had a very concrete plan as to how I was going to go about testing and developing software for learning from demonstration. As a result, I had a very vague project plan, which has caused me some issues over time. Were I to take this class again, I would put a lot more time and effort into my initial project proposal.

Additionally, I learned a great deal about robot control and about all of the different tools I needed to use to run the simulation. I am now much more experienced with open-source software tools used for robotics:

- *ROS:* I learned about the URDF format for designing objects in ROS, and how to incorporate models from 3D modeling suites such as Blender. I used the Space Navigator node in ROS for the first time, and used it to build the system for teleoperating my simulation. I learned quite a bit about ROS launch files and parameters, which are used to store things like robot and object models.

- *Gazebo:* I learned about how to write a plugin for Gazebo, and how to simulate different objects and their interactions.
- *OpenCV:* I used OpenCV to find key points in images, and I used it to compute an affine transform between two sets of 3D points for my tests.
- *Orocos RTT:* I learned about using the Orocos tools for different controllers, and the Orocos debugging console. I also learned about
- *Integration between all of the above:* I learned about using an URDF with a Gazebo plugin, and about the limitations of doing so. I used Orocos, OpenCV, and Gazebo together with ROS and in conjunction with ROS nodes, and I learned about publishing TF frames.

I also learned a lot about different technologies for inverse optimal control, and made some good progress towards implementing one system early on that I have not yet adapted to the simulation. I was surprised by how much time it took to familiarize myself with tools like Gazebo, writing plugins, and actually getting a working simulation together and off the ground, even though I was starting from code that was mostly working for one robot (the ROS/Orocos integration package and Barrett manager, written by Jon Bohren).

6.3 Future Direction

This work has left me in an excellent position to continue my research over the summer. There are a number of different improvements I have planned, and I want to investigate and test the applicability of certain approaches, such as that of Levine and Koltun [5], on tasks performed by the simulated WAM arms. Below, I elaborate on specific future tasks I plan on completing in the coming weeks.

Improving Construction Simulation Right now, I have a working simulation of a simple magnetic construction/destruction task. However, the blocks only link together in a specific, preset way, and right now there is no simulated magnetic force acting on them. This is partly because Gazebo does not provide a simple interface to detect collisions. To fix this, I can make each magnetic block a simulated touch sensor, and check to see which other block collided with it. Each block should have up to 6 rotation joints, instead of having a preset number; these joints should at first point to nothing, or to the block itself. When a collision is detected, instead of reactivating a joint that already exists, I can set the end point of the joint to the block that collided with the first one.

MATLAB interface In order to experiment with different algorithms, I need a better way to visualize trajectories and features. I want to use the new MATLAB/ROS interface with a slightly modified version of my file I/O in order to read trajectories as matrices in MATLAB and write them back out to files. As an added bonus, I can readily plug this code into previous research code from Amir Masoud [4] and into code for Continuous Inverse Optimal Control [5].

Improved Task Modeling Recently my co-workers and I have been looking into applying temporal logic to reasoning about events and how they relate to one another [10]. In some previous work, researchers analyzed a basketball game between two players, applying simple predicates like ”has possession of ball” to identify actions [6]. Other work has now applied this to object recognition [7]. This approach may be useful in robot collaboration: predicates describing which arm has possession of the ring or whether the ring is still around the peg, for example, would be useful in determining what the status of the task is and when intervention should occur.

Real-World Tests One of the advantages of the framework I used to develop this software is that this software will now be used by all members of my lab, and the software and performance is almost identical to that of the physical Barrett WAM arms. I am hoping to use this simulation and the tools I have build to develop software that can then be tested on the real Barrett WAM arms; hopefully this will lead to published work in ICRA or another future robotics conference.

References

[1] Narges Ahmidi, Yixin Gao, Benjamín Béjar, S. Swaroop Vedula, Sanjeev Khudanpur, René Vidal, and Gregory D. Hager. String motif-based description of tool motion for detecting skill and gestures in robotic surgery. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2013*, Nogoya, Japan, 2013. Springer, Springer.

[2] Sebastian Bodenstedt, Nicolas Padoy, and Gregory Hager. Learned partial automation for shared control in tele-robotic manipulation. In *2012 AAAI Fall Symposium Series*, 2012.

[3] Carl Henrik Ek, Philip HS Torr, and Neil D Lawrence. Gaussian process latent variable models for human pose estimation. In *Machine learning for multimodal interaction*, pages 132–143. Springer, 2008.

[4] Amir M. Ghalamzan E., Chris Paxton, Gregory Hager, and Luca Bascetta. Robot learning from demonstration: from imitation to emulation. 2014.

[5] S. Levine and V. Koltun. Continuous inverse optimal control with locally optimal examples. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, volume 1, pages 41 – 48, 2012.

[6] Vlad I Morariu and Larry S Davis. Multi-agent event recognition in structured scenarios. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3289–3296. IEEE, 2011.

[7] Daniel Nyga, Ferenc Balint-Benczedi, and Michael Beetz. Pr2 looking at things: Ensemble learning for unstructured information processing with markov logic network. ICRA PREPRINT, 2014.

[8] John Schulman, Ankush Gupta, Sibi Venkatesan, Mallory Tayson-Frederick, and Pieter Abbeel. A case study of trajectory transfer through non-rigid registration for a simplified suturing scenario. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4111–4117. IEEE, 2013.

[9] Lingling Tao, Gregory D. Hager, and René Vidal. Segmentation and recognition of surgical gestures from kinematic and video data. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2013*, Nogoya, Japan, 2013. Springer, Springer.

[10] Son D Tran and Larry S Davis. Event modeling and recognition using markov logic networks. In *Computer Vision–ECCV 2008*, pages 610–623. Springer, 2008.

[11] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, pages 1433 –1438, 2008.