Virtual Learning from Demonstration Final Report

Jun Dong - dongj2 - 661125791

Logan Gittelson - gittel - 660894626

I . Motivation

It is hard to convey certain knowledge to robots by programming them and designing each step in their plan. A very straightforward example is to find a stable position to grasp objects. Grasping the same part of two plates that are the same shape might return success for one while breaking the other simply because they are made of different material. This form of knowledge is very abstract, it varies from object to object. Even the same object could require different methods of interaction in different environments. Thus it is necessary to enable robots to learn by themselves and for humans to only need to teach them which features are import to learn from.

Learning from Demonstration (LfD) is based on this idea. We have seen some amazing results, for example a robot arm cooking pancakes, robot fingers opening bottle cap, etc. However, the method that has been used for these example is constrained in such a way that only roboticists can train the robots because most other people have neither a real robot to work with nor the necessary knowledge.

This project, therefore, is aimed at developing a Virtual Learning from Demonstration (VLfD) system that enables ordinary people to control simulated robots using their own motion with the help of natural interface devices such as Microsoft's Kinect. Developing this simulation system first helps make VLfD accessible before intelligent robots are commonly available. Our long term perspective for this project is to incorporate it into games, so that anyone who plays this robot game will automatically be helping researchers train robots.

II. Related Work

Learning from Demonstration (LfD) started in the 1980s and tries to make robots learn without actually programming them. There are several survey papers in the field, including Argall et al 2010; Billard et al 2013.

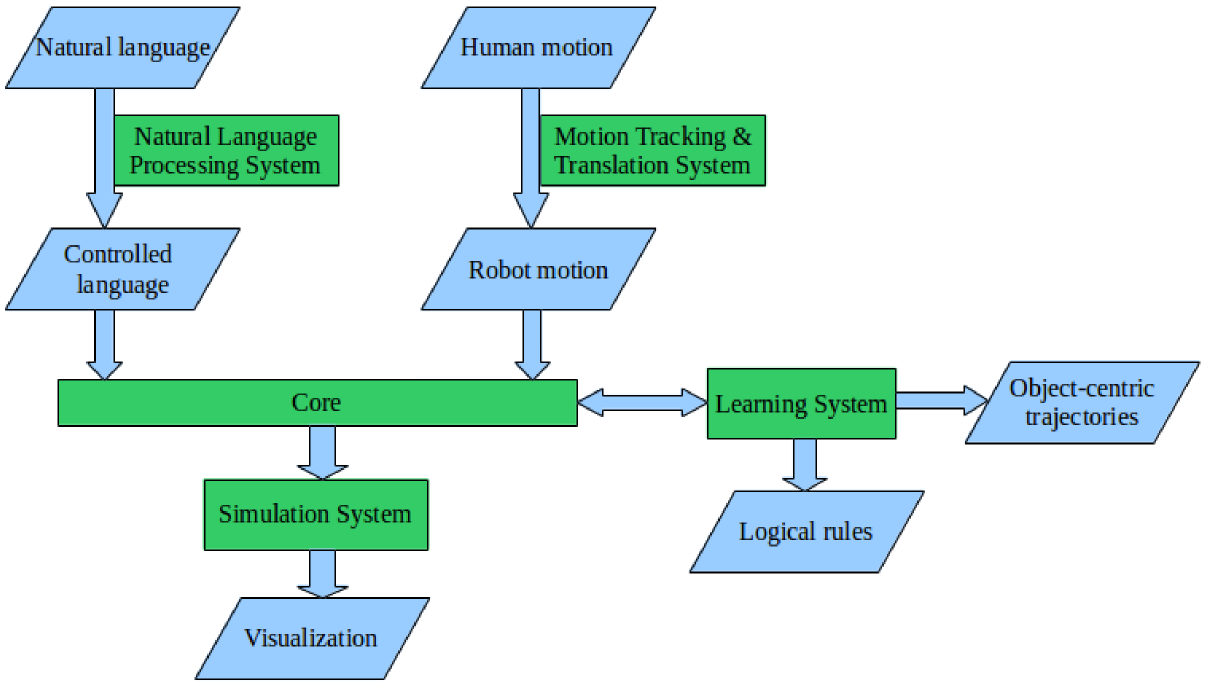
Argall et al 2010 is one of the most cited paper in the LfD community, which not only analyzed recent work, but also segmented the learning problem into two fundamental phases: gathering demonstration examples and deriving a policy from these examples. They mentioned the choice of demonstrator is very important to the approach, while for us human are considered the best demonstrators since most of the tasks we are aiming to handle in the future will be too difficult for machine demonstrators. This paper also talks about being able to iteratively demonstrate and train robots, but that will be out of the scope of this course's project and therefore left for future work.

Billard et al 2013 is a more recent work that also did very comprehensive job in summarizing the issues, methods and techniques for LfD. One very important issue pointed out in this paper is what to imitate and how to measure the correctness. As described in the work, children learn to imitate everything at first, but then improve to be able to eventually only imitate the actions required to achieve the goal. The paper discusses how we may track effects of actions instead of learn the actions themselves. Our project is also aimed at imitating the effect of demonstrations in the future, like which part of the object should be pushed to make it move in a certain direction. This course's project is constructing the first piece of the VLfD system that will be used for learning research. For now, it's simply imitating whole actions without any consideration of the goal at hand.

III. Algorithm / Technique

System Structure

The image below shows the main structure of the system.



The Core System serves as the center of the whole VLfD project. It reads in buffered data from the Motion System and the Natural Language Processing (NLP) System, while the Simulation System receives commands from the Core System to move robot joint angles to certain values. Once the basic framework meets our requirements, the Learning System can be added, which also interacts with the Core

System. That remains to be done in our future work. There will be too many other issues to be resolved, including deciding on what features to learn from. The Natural Language Processing (NLP) System is a combined sub-project for another course and therefore will not be one of the focuses of this report.

Threading

Threading is one of the most important techniques we used in the implementation. The Motion System has a separate thread that runs the GLUT visualization of the tracked human / skeleton and stores positional information into a global variable. The main thread of the Motion System, on the other hand, maintains the motion buffer using the information it collects from the other thread. Similarly, the NLP System uses two threads for human input; one for typing and the other for speech recognition.

Core System

The Core System, which holds an instance of the Motion System class and an instance of the NLP System class, loops between reading Motion / NLP buffered data and sending commands to the Simulation System. With the position and orientation data collected by the Motion System, an Inverse Kinematics (IK) Module is used to translate the task space trajectory into joint space trajectory, which is easier for the robot arm to follow. The calculated joint trajectory is then sent to the Simulation System. One hack we did here was to only consider the most recent several points instead of the whole buffered trajectory. This helps give us a more real time effect for the robot motion, otherwise we could be imitating a motion the user did several minutes ago.

Inverse Kinematics

We used the Kinematics & Dynamics Library (KDL) and modified its Weighted Damped Least Square (WDLS) iterative IK solver. The improvement in our version is that it will automatically check singular values when solving IK problem. If the singular values are too small, we will stick to the WDLS method. Otherwise, the solver will use pseudo-inverse method to achieve faster convergence.

Simulation System

The Simulation System consists of two parts: the system itself that interfaces with the Core System and the Gazebo plugin for the powerball arm. The first part implements a motion controller that provides task space trajectory control and joint space trajectory control. Currently, we only implemented the joint space trajectory, which basically receives joint trajectory commands from the Core System, composes a Gazebo message for it and sends the message to the Gazebo Powerball plugin. The Powerball plugin, which inherits from the Gazebo ModelPlugin, has full access to the robot arm’s physical properties, including joint force, link positions, etc. When the plugin receives the joint trajectory message, it will calculate the difference between the target joint values and the current joint values, which will be used in PID control to populate the target torques to send to each joint motor.

Motion System

The Motion System class is structured in such a way that there should only be one object of it. Once the object is created it initializes a separate thread that runs the function (wasmain(), a modified NiTE sample function) which sets up and executes the main GLUT loop. For every frame the main GLUT loop grabs the depth map and calculated skeleton data (if available) and then puts the tracked points from the skeleton into a global buffer which the Motion System can access.

When the Motion System is called to set robot usable coordinates, via the pass-by-referenced double-ended queue (deque), it translates the right-hand coordinates in to meters relative to the torso position for the robot to use directly as end-effector position. The left-hand coordinates are translated into row, pitch, and yaw values (relative to the torso position and the user's maximum range) which are meant to be used for fine-adjustment positioning of the arm, but is not currently implemented. After the program grabs and translates all these values the global buffer is reset and the function ends.

While the GLUT main loop is running, a window is shown which draws the depth map and overlaid skeleton on to the screen. This window serves to show the robot's input and demonstrates how smoothly the whole interface runs; it is also very useful for debugging.

Some minor changes were implemented during the translation step to reduce the amount of data being passed to the Simulation System. If neither the left nor right hand have changed their positions (relative to the chest) from the last time observed that data will not be added to the dequeue. This change unburdened some of the work from the inverse kinematics calculation, without losing noticeable accuracy, and helped reduce noise.

The packages used for the motion system include OpenNI with the PrimeSense Sensor Module and the PrimeSense NiTE control middleware. The files which grab and display the kinect data are a modified version of NiTE’s “Players” sample code. Originally the example simply displayed this information. For this project code was added which appended certain points from the skeleton into a global buffer. Furthermore, the motion system class was also created to act as an interface between the former example and the rest of the VLfD system by grabbing and translating data points. The skeleton is generated using NiTE, which depends on the OpenNI and Sensor Module libraries. As of April 23, 2014 PrimeSense has been acquired by Apple and their website is no longer online, fortunately OpenNI and Sensor Module are still available via Github and we’ve included the files for NiTE with this project.

IV. Results

The whole system builds and runs correctly. It can track human demonstrator motion with a GLUT visualization of the tracked skeleton, and the simulated robot can move correspondingly to the demonstrated end effector positions, if the IK solutions for those positions are available. The accuracy of position following is good enough for what we need it for in our system. We set the acceptable joint error to be 0.01 radian.As far as we know, the Kinect system also gives very stable and precise position data, which are accurate to the fractions of a centimeter.

Core System Limitations and Bugs

Since considering the orientation requirement only compresses the workspace of the robot and hinders the performance of the IK solvers, it has not yet been implemented. We simplified the problem of tracking the full 6D pose into simply tracking 3D position first.

One common problem with using threads is that we need to synchronize between each thread so that they will not run in an order which causes a deadlock or crash. Thus we used several sleep function in the system. This might potentially cause the system to slow down. This will need further testing in the future.

Motion System Limitations and Bugs

Although we feel that our project has come a long way, there are still a few bugs and limitations which we have not yet ironed out. One of the largest bugs, most noticeable bugs is “Error 139”. Error 139 occurs whenever our system is closed and then tries to reopen, it is always accompanied by the program closing. Fortunately, this error is resolved easily; if the Kinect is unplugged and then plugged back in the system will run normally (until the next time it closes). We believe this error is caused by some sort of lock on the Kinect drivers not being let go when our system closes.

There is a limitation with the motion system where calibration can only be attempted once. Whenever the main GLUT loop starts it will attempt to identify a user and then calibrate a skeleton to them. Once a user is identified and differentiated from the background the program will look for a “Psi” pose (arms extended to the side at shoulder level, forearms bent upwards), if a wireframe skeleton is unable to be mapped once this pose is detected calibration will fail and will not be attempted again (until next run). This limitation was also present in the original sample our code was adapted from.

V. Challenges Faced

Finding, Installing, and Modifying the Necessary Kinect Code

One of the first challenges we faced was finding the tools to complete our project. Having never worked with Kinect before, we had to figure out not only how to generate and grab the data, but also how to display it and share it with the rest of our system. We found multiple libraries which interacted with the Kinect, but soon figured out we needed NiTE to do any sort of full body tracking. Given that the provider of the necessary packages (PrimeSense) had recently gone offline and there was no official source for the information we needed this was a challenge. After struggling through multiple tutorials which proved only partially helpful we eventually got the software installed and were able to start figuring out how to interact with it.

Due to lack of documentation, time, and expertise, we decided that one of the included demos had most of what we needed and a relatively simple modification was all that was necessary. We were able to display the coordinates of the points we needed, but now we had to figure out how to get them to the rest of the system. Given that the program we modified did not end until it was shut down we decided the best thing to do would be to call it in its own thread and pass the values through a global buffer (originally a pass-by-reference buffer was tested, but that proved unfruitful). We now had all the code to start tracking the user, display what the Kinect output, and pass the relevant information to wherever we wanted, all in real time.

Designing control scheme

Designing the system we faced a pretty fundamental question we had no certain answers for “How should the robot be controlled?”. We needed to control not only the end effector (hand) position, but we also wanted to control the orientation of the arm. Mechanically, the robot arm and a human arm are not exactly the same; some positions that might be natural for a human might be impossible for the robot, and vica versa. We decided that instead of trying to mimic a human exactly we should use two hands to control this one arm. Obviously one hand could be used for end effector position, so we decided to use the other for orientation of the robotic arm. Movements towards or away from the camera with the left hand would control row, left or right of the body would control pitch, and up or down would be yaw. We implemented tracking and transformation of these values, but unfortunately they posed too much of a complication for the inverse kinematics solver. We tested the system without the orientation specifications and it worked wonderfully, we decided to just go with this and leave orientation specification for future work.

Software Independency

Robot Operating System (ROS) is currently the most popular robotics research platform. A lot of the robotics packages are shared on the platform, however, we wished to make our system as independent as possible and provide a wrapper for ROS instead of totally depending on it. This objective caused a lot of problems for us since many of easy-to-use packages are exclusively ROS plugins. Instead of using the well-known ROS message passing system, we used gazebo specific message passing system and had a lot of trouble in finding the correct documentation for it. Learning to use its API was another headache altogether. In the future we can improve this part of the project to interface with multiple kinds of simulators, with or without ROS.

VI. System Installation and Running

Our system runs on Ubuntu. We tested it on Ubuntu 12.04 and 13.10. We wrote installation scripts to make the software easy to use and test. Following the instructions for installing will add several dependencies package into your ubuntu system. Please see our “Installation and Running” documentation for more details.

VII. Conclusion

How Long it Took and Who Did What

Our team of two met regularly three times a week over the course of two and a half weeks. Additional meetings were held as necessary and resulted in an average of one and a half extra meetings per week. On average, we met for four hours at each meeting (ranging from one hour to seven hours), this resulted in an estimated total of 45 hours of meeting time. We followed a SCRUM process using both a product backlog and a sprint backlog, with three sprints total. We also both spent time coding for the project individually. A rough estimate of 20 hours each were spent coding outside of meetings, but this time was not tracked.

Jun Dong created the SCRUM documents and the overall design of the system. Jun is responsible for the implementation of all special data structures used along with the creation and debugging of core and simulation systems. He also wrote all the installation scripts and cmake files which allowed for the integration of all the subsystems and their dependent packages into one cohesive system.

Logan Gittelson created the motion system, which grabs data from the Kinect, transforms it, and allows it to be accessed from the rest of the system. He also researched usable Kinect libraries, and served as the user while testing the system and creating videos.

Possible applications

This technology has many possible applications in the future. VLfD alone would prove useful in many areas including manufacturing, the service industry, and personal robotics. VLfD could allow robots to learn how to interact with different material, perform various physical tasks (within its structural limitations), and dynamically adjust to new interactions. More generally, natural interaction has implications for telepresence, assistive technologies, and seamless virtual interactions. These technologies have the potential to revolutionize both the analog and digital worlds that we live in today.

VIII. Bibliography

[1] B. Argall, S. Chernova, M. Veloso, B. Browning, Robotics and Autonomous Systems 57 (2009) 469-483
[2] A. Billard, D. Grollman, Robot Learning by Demonstration, Scholarpedia (2013)
[3] Kinematics and Dynamics Library (KDL), http://www.orocos.org/kdl
[4] CMU PocketSphinx, http://cmusphinx.sourceforge.net/
[5] OpenNI, http://structure.io/openni
[6] E. Lim, “Installing OpenNI, NiTE, SensorKinect on Ubuntu 12.04”, http://xpectomas.blogspot.com/2013/08/installing-openni-nite-sensorkinect-on.html