Team Bust-A-Move

Computational Robotics Final Report

12/18/2014

***Fiducial Tracking Sub-Team**: Allie Duncan, Kyle McConnaughay*

***Arm Sub-Team**: Rachel Boy, Victoria Coleman, Brooks Willis*

Goals

The combined team's goal was to have a robotic arm learn various dance moves--movement patterns--by observing an example and generalizing the motion so that it could be performed between arbitrary start and end positions, thus busting some sweet moves. We wanted to extract the positions of a dancer's arm joints using fiducial tracking in a video feed through two methods: first using a package, then implementing the algorithm from scratch. We then wanted to use the recorded path of joint positions to learn a control strategy for busting different funky dance moves, using an already implemented package. We then wanted to able to have the arm execute these generalized dance moves.

Problem Solution

Fiducial Tracking

We decided to tackle the problem of tracking joints by attaching fiducials to a person's wrist, elbow, and shoulder. To do this, we used the ar_pose package. The largest problem our team faced was the poor documentation (and thus the inevitable struggle) of using the ar_pose package to track any fiducial. For example, gscam is required by ar_pose but is not even mentioned on the man page. As a result, our team struggled for weeks to get a working system. Rather than moving on after eventually getting a working system, we made an effort to stop and create documentation (and sh files) to recreate the process. In fact, we even uninstalled the entire system and reinstalled it to ensure that our sh documentation was complete. It was particularly rewarding to have our teammates be able to install the system in minutes. This documentation can be found at https://github.com/Brooks-Willis/dance_bot/tree/master/fiducial_tracking.

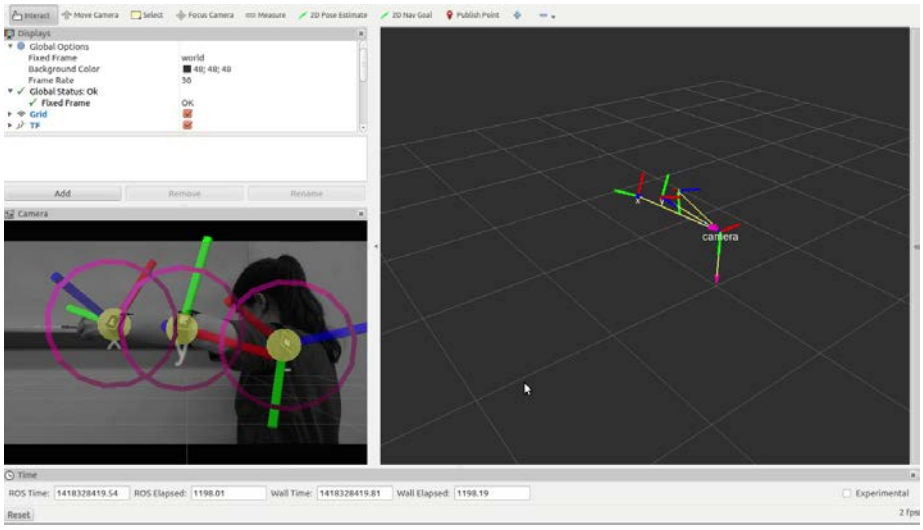


Figure 1:Ar_pose fiducial tracking on a human arm

Arm

Because the arm is a closed system with known dynamics, it would be easy to simply copy the movements of a reference arm in almost real-time to recreate exactly the same move. However, we want the arm to learn, not just copy, and to be able to perform the same dance move in different contexts - for instance, with different start and end points. To do this, we used Dynamic Motion Primitives (DMPs), which learn a control scheme on the basis of an example movement. They represent this control scheme as a set of differential equations with adjustable start

conditions and end attractors, allowing them to generalize the moves they've learned. Because the math for generating and working with motion primitives is complicated, we utilized a ROS package (dmp) that was robust and functional so we could concentrate on getting an entire system--from someone's arm moving in real life to the arm emulating the general behavior--working well.

While the dmp package was monumentally better than writing that functionality ourselves, we found that the robustness of a DMP was highly sensitive to the learning parameters. Different types of move had very different best parameters, so learning an individual move required a fair amount of tuning on our part. Because the primitives have to be learned, we require the entire motion of a move to be processed to create the primitives that can be used by the arm. That means that the arm does not learn in real-time and must have pre-stored learned primitives to draw on to dance. However, once it has those primitives, it can use them to recreate dance moves at different speeds with some variation in start and end points.

Design Decision

Fiducial Tracking

The major design decision of our subsystem was how to track the joints of the arm. We considered all of the following points (and more):

- skeleton tracking using the kinect
- edge detection (keeping the arm on a dark background)
- corner detection (to find joints)
- keypoint matching with a 3D camera

In the end, our decision to use fiducials was based on requirements from the other team and project scope. The other subteam required accurate, reliable data with depth information. We had prior knowledge of ar_pose and knew that it was capable of delivering the desired functionality. Additionally, we only had three weeks, which precluded the possibility of implementing something as algorithmically complex as ar_pose. We decided to work with a technology we were familiar with and confident could be used reliably to achieve these requirements.

Arm

Since we already had an example path from the fiducial team, and the arm has position control implemented, we decided that we wanted to do more than precisely mimic dance moves. We decided we wanted to be able to set arbitrary start and end points and complete movements similar to the training paths without simply doing coordinate transforms from arm to joint positions. Because of the complexity of this problem, we decided to simplify it slightly by just directly controlling the end effector and not the elbow joint as well.

On the architectural level, we chose to save all the learning paths and motion primitives to file so that we can collect a set of dance moves that can be remembered from session to session. This is both practical for use so the robot does not have to continually relearn moves and also practical for the development process so we can have consistent inputs to do more precise comparisons of outputs from various methods of processing. Storing these intermediate files reduced the development time overall by allowing us to debug later parts of the process without having to re-do earlier stages.

Code Structure

Fiducial Tracking

The code for fiducial tracking is all sh scripts because it completely leverages the ar_pose package. These files can be broken down into two categories: installation scripts and execution scripts. Installation scripts help configure the environment to properly run ar_pose. The execution scripts run ar_pose or other dependencies (to check proper functionality). These include:

- gscam
- camera calibration
- ar_pose

To run the entire system (using patternMaker patterns 1, 23, 66) follow the instructions in the README in fiducial_tracking.

Arm

The general overview of the code for the behavior processing and implementation of the movements is as follows: we transform the incoming data from fiducial tracking into the correct coordinate system for the arm, then save a .csv of the demonstrated path. After the .csv has been saved, we load it, pass the points and appropriate tuning constants to the DMP server and save the DMP's. We can then at any time reload the DMP's to plan a move based on the start position and desired end position, and then finally convey the newly planned path to the arm for execution. This architecture is visualized in the figure below. The "DMP Server" is the part of the code provided by the dmp package. We have a python wrapper that communicates with the arm over serial using Roboforth, but all the other components communicate via ROS.

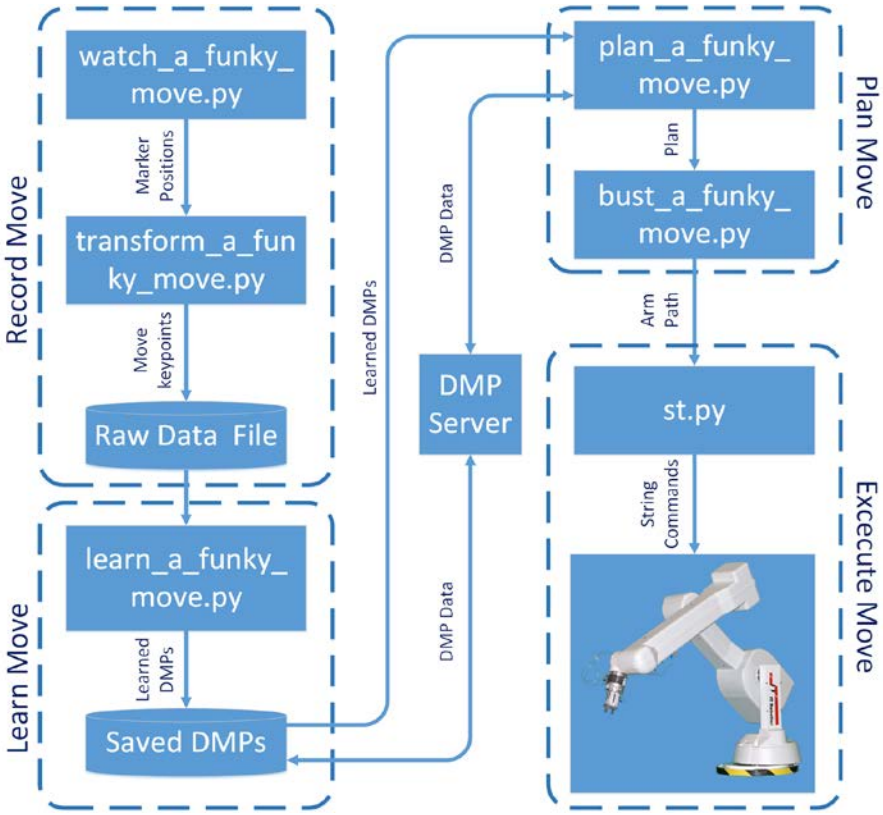


Figure 2:Full system diagram, with script names

The dmp package can be found at https://github.com/sniekum/dmp. All code can be run using launch files found in dance_bot/launch. These launch files are explained in the README found in the launch folder.

Challenges

Fiducial Tracking

As mentioned in problem solution, the greatest source of difficulty was the lack of/poor documentation. Several significant challenges arose from this.

The first major challenge was understanding how and where to change fiducials are placed. The actual pattern files are within a data directory with a "fiducials" file that lists all the patterns to be used. The launch file (created in fiducial_tracking.sh) has the path of the "fiducials" file. However, even when changing this path, it will *always* look for the patterns in the data file. (No matter where the fiducials file is.) This was particularly frustrating because we want our code to be self-contained. We didn't want users to have to edit ar_pose. End the end, we copy our fiducial patterns and add them to the data file in ar_pose automatically when running fiducial_tracking.sh. This is the most elegant solution we could come up with.

Another related challenge was the naming conventions of the patterns. To this day, we don't know the reasoning behind this. Pattern files must be named a .[alphabet_letter]. For example, patt.x, patt.y and patt.z are our fiducials. Naming the fiducials other, more sensical things like patt.elbow resulted in ar_pose crashing.

Finally, our last major challenge was knowing to change the ar_pose threshold. This variable is declared within the launch file (created in fiducial_tracking.sh) and is the only variable exposed by ar_pose. For days, we were mystified why ar_pose would recognize large fiducial images but fail to detect the same small fiducial. In the end, we needed to decrease the threshold, which increases the sensitivity of fiducial detection. Choosing a final value for the threshold was done through trial and error.

Arm

The arm runs on a language called RoboFourth, which is an unintuitive language in the world. The arm has terrible documentation that assumes only control through a GUI, so it mentions of actual command strings to the arm were incomplete, and the controller had little to no helpful feedback, especially when dealing with routes--the only way to get continuous motion. As a result, either a command worked or there was an empty abyss of nothingness with a monolithic arm contemptuously staring you down. Eventually, we got our hands on a LabVIEW wrapper that implemented routes constructed by intercepting the serial messages that the GUI produced due to poor documentation. That was an invaluable resource for being able to implement continuous motion, but we were still plagued with weird bugs such as not executing paths if the points were too close together, and the threshold for "too close" was dependent on the acceleration and speed of the arm.

The DMP package we used also had a few problems. Most notably among the that it was not originally compatible with catkin, as it was originally designed for rosbuild. We actually solved this in two ways, first we installed a rosbuild workspace on top of our catkin workspace so that it could

run. We felt clever for figuring out this reasonably involved solution. During the process we contacted the developer of the package to inform him of the incompatibility; he replied within 24 hours having fixed it, at which point we had to remove the rosbuild workspaces and the package worked like any other. We also realized at this point that we could have originally simply rolled back the package by a few commits and avoided rosbuild in the first place, and then felt less clever.

Dynamic Motion Primitives have several parameters, each of which has a significant effect on the success of the path planning process. An imbalance in these parameters, coupled with a low number of teaching examples (one) lead to some very odd behavior. For example, in figure *** below the arm is meant to follow a V-shaped path. Which it does do for the most part, however once it reaches the goal the first time it bounces back to the tip of the V before finally returning to the goal point. Despite extensive parameter tuning, we were not able to get around this issue for any of our runs of this particular path.

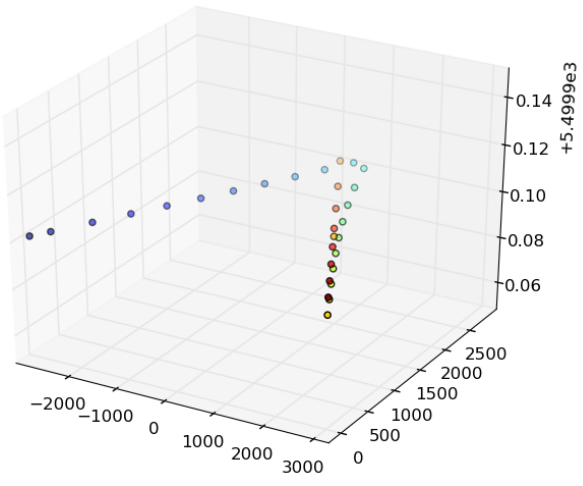


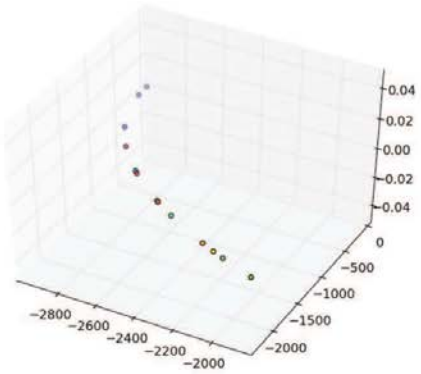
Figure 3:V-shaped path. The arm follows the correct path, but has an extra movement at the end

Path actually followed by the robot when given a V-shaped path to follow. Blue indicates the start of the path, and red is the end. The start and end coordinates have been moved significantly from the original training path, yet the path maintains its shape.

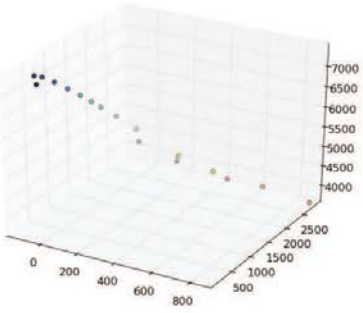
Another problem which arose from the low number of teaching examples we provided the arm was that the paths were not very robust to rotations and distortions. We were readily able to scale and translate the path all over the robot's workspace and maintain a very tight match to the input path. Once the start and goal points were set such that the path would need to rotate or be distorted from its original configuration, however, the accuracy of the output path dropped steeply. This is likely a problem inherent to our low number of training examples. We suspect that the correct solution is to provide the DMP server with more example paths so that it can more robustly transform the path, unfortunately by the time we identified the problem we did not have enough time to implement that specific solution.

Results

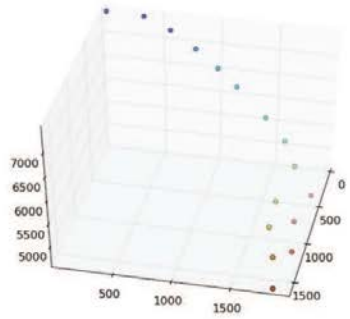
We were able to successful track the fiducials with reasonable accuracy, and after tossing out the incomplete points, we had paths from which to learn DMPs. For simple motions, we were able to generate reasonable paths from the DMPs with appropriately scoped start and end positions. The image below shows the original path from the fiducial tracking and to paths with different end and start points that demonstrate. Notice that they have a very similar form factor (taking the differing rotation of the coordinate frames in account) to the original path but are scaled appropriately to the new start and end positions.



Plot of the input path taken from the fidutial tracking with a linear translation to robot coordinates



Generated path from learned DMP with [0,3000,4000] endpoint



Generated path from learned DMP with [800,2000,50000]

Figure 4:The top graph is the path used for learning the DMP, which the bottom two paths were generated from with a radically different start point and endpoints that differed from eachother, but used the same DMP

Improvements

Fiducial

Given more time, we would have liked to move away from ar_pose (and to instead implement joint tracking without wearing itchy bands). This could be achieved through working with the Kinect. The Kinect requires tracking the whole body - rather than just the arm. So the motions of the entire person could potentially be translated to the robot arm, rather than just one arm.

Arm

Further steps to take with this project include implementing individual joint control instead of just controlling the end effector's position. This would enable the arm to have more dance-like movements, but the problem of mapping becomes significantly more difficult since the robotic arm has significantly fewer degrees of freedom than a human's arm. Furthermore, to have more realistic dance moves, we would need to expand the Roboforth wrapper to allow variable speeds between points. Since the fiducial processing code outputs timestamps with the points of the paths, the speed of the robot could be controlled to mimic the relative timing of the dance move as opposed to the constant speed it currently has.

Another step would be to improve the learning we're doing. One way would be to modify our use of DMPs to allow us to learn a single set of DMPs from multiple training examples. Another option is to investigate alternative ways of specifying different endpoints - for instance rotating the axes, or shifting the origin. This would allow us to cover a wider range of variations in a way that is friendlier to the DMPs. It would also be interesting to set up our system to smoothly chain multiple DMPs to make even funkier dances.

Beyond improving the root functionality, another area needing development is making the entire system into one clean, easy to use program. Currently there are several separate steps that need to be run in succession with a very rudimentary user interface. Bundling the entire system into one script that has an appropriate, even if rudimentary, GUI to do the following thing things: choose when to record, what DMC's to create with custom parameters, naming paths and DMC's, visualizing the path, and running the arm.

Lessons (for the future)

Fiducial Tracking

Our subteam is relatively pleased with the reliability of our final product. With that said, the final product is not what we were expecting in the beginning of the progress. (As a reminder, we were expecting to implement our own joint tracking.) For future students, we recommend doing more thorough research before settling on a project. Had we known the course our project would take (only using ar_pose and mostly dealing with environment configurations), we would have pivoted or potentially changed projects completely.

Arm

One of the best things we did during our development process was to write a node that would replace the controller for the arm and allow us to visualize the path we would send to the arm without having to run it on the arm. Since the process of getting the arm to actually run paths smoothly took most of the project, this was a vital for making progress in a way we could eventually integrate. Writing this sooner would have improved our process even more, and one of our takeaways is to a) not underestimate the amount of time necessary to decipher the obscure inner workings of proprietary hardware, and b) allow for as much parallel development as possible as quickly as possible.

We also learned some about using other people's packages during this project. We received mixed benefits from the dmp package. On one hand, actually writing that code would have taken us the entire project, if not longer, and we would probably not have arrived at even a good proof of

concept. However, using that package locked us into the features it implemented. For instance, we would have loved to try to learn a move on the basis of multiple examples. (But it does not look like the dmp package supports this.) Using an already existing package was definitely the best choice for this scale of project, giving the huge amounts of other integration that needed to happen, and there are certainly many improvements remaining that we could make using the dmp package. Were we to try to implement all of the ideas we had, we would need to either begin modifying the dmp package source code or writing our own implementation.

References

DMP package:

- https://github.com/sniekum/dmp
- http://wiki.ros.org/dmp

DMPs in general:

- http://www-clmc.usc.edu/publications/P/pastor-ICRA2009.pdf
- http://people.cs.umass.edu/~sniekum/pubs/NiekumRSS2013.pdf (we didn't find this until too late, but boy would it have been useful)
- http://www.ros.org/presentations/2009-12-PeterPastor-DMP.pdf

Fiducial tracking:

- http://wiki.ros.org/ar_pose
- http://wiki.ros.org/gscam
- http://wiki.ros.org/camera_calibration
- http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration
- http://wiki.ros.org/uvc_camera

Video of working robot:

- https://www.youtube.com/watch?v=b9oR-3tMqFQ

Our code:

- https://github.com/Brooks-Willis/dance_bot