Teach Robot to Dance!

Final Report

Shiji Liu, Shaoxiong Yao, Jinze Liu, Xu Han

Table of contents

Introduction and Value Proposition

Nowadays, most robot motion controls rely on either pre-programmed or teleoperated by humans, both of which have some drawbacks. Pre-programmed robot motion control is not adaptive. Programmers need to spend a lot of time on programming the robot and debugging in order that the robot can finish a certain task. Moreover, in this case, robots' functions are limited by the program, and when the working environment changes, robots have to be re-programmed. Meanwhile, the robot teleoperated by humans through remote controllers, such as joysticks, is slow, imprecise and non-intuitive. Therefore, it is hard to enable robots to finish complicated tasks by teleoperation. What's more, training a qualified robot player can be time-consuming.

So, imagine that you bought a human robot and want to make it dance in front of your class. If you want to enable robots to dance by pre-programming, you need to consider every single movement of the robot, input the position of each joint and joint angle to the program, and debug for a long time in order that the robot does not lose balance. If we want to teleoperate the robot dance, you may need a lot of practice. Some familiar motions, such as walking, raising hands, may be stored in the controller, but it is hard to combine them and make beautiful dance. There should be an easier way for robots to imitate human motion.

In our project, we designed and implemented a user-friendly system that can enable robots to mirror human motion. It uses a kinect to collect human motion, retarget it to robot and let robot to do the same motion. The control method is intuitive, and the user can focus only on the dance itself without being trained to use the controller. Moreover, the usage of our system is not limited to dance. Our robot motion data can be stored and used as a start of the learning process for the robots which people can teach to do various tasks, such as cooking.

Project Technical Objectives

Our group aims to make humanoid robot control more intuitive to human beings. Based on this idea, we implemented the project "Teach Robot to Dance", which included a set of easy-to-use tools that enable the humanoid robot we chose to mirror human dance. To enable robots mimicing human dance, our project achieves the following objectives.

- Real-time human motion capturing by kinect
- Real-time mapping of human motion onto a robot
- Smooth robot motion control

We used a pepper humanoid robot to implement our robot. Due to the budget constraint, we choose to implement our project with simulation software. Since all commands to control the pepper robot are identical for both simulation software and real robots, our project should also work on real-world robots.

Due to the limitation of pepper robot joints and function of kinect, motion on hands, head and legs cannot be mirrored. We give a set of allowed motions, as shown in figure 1. Users can do arbitrary transitions and combinations between those motions.

Figure 1. Set of allowed motions

Also, to ensure user experience when using our project outcome, we will create a GUI that can

- Help users to find the best position to start the dance
- Give a short tutorial on how to use our project
- Give a set of allowed motions to users to show the limitation of our project.

Project Architecture

In our project, we will invite a user to dance in the motion capture region. The pepper robot in simulation software will follow the user's motion in real-time.

Motion capture: Our motion capture system uses a built-in library of Kinect v1 to capture the dancer's motion and output human 3D keypoints in real-time.

Motion retargeting: The motion retargeting system will receive the dancer key points from the motion capture system as input. Then our motion retargeting system will generate joint value commands on the Pepper humanoid robot.

Motion smoothing: Due to sensory noise, the robot real-time dance motion can be jerky. We enable this option to smooth the dance motion and replay it. The motion smooth system will take a sequence of joint value commands from the motion retargeting system as input and generate smoothed robot control commands.

The flow chart of our project architecture is shown below.

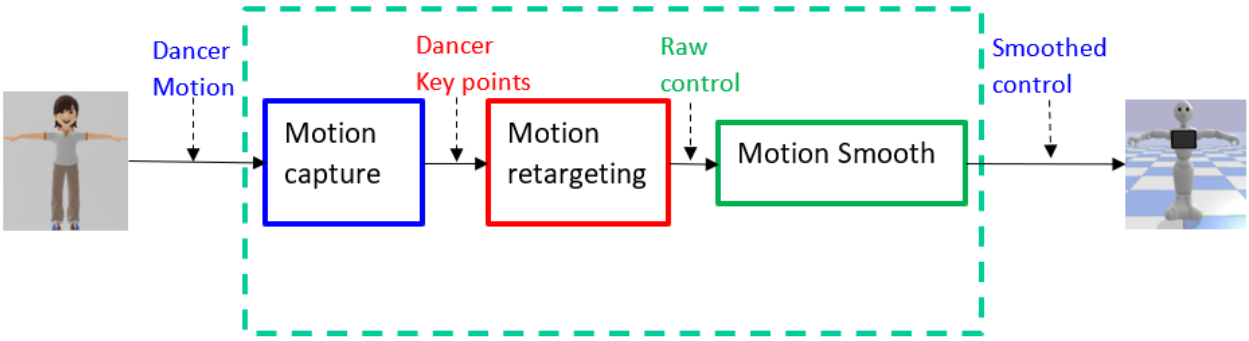


Figure 2: high-level diagram of our project.

Project Prototype Implementation

Robot Choice

We choose to use Pepper humanoid robot[1], an image of the humanoid robot is shown below.

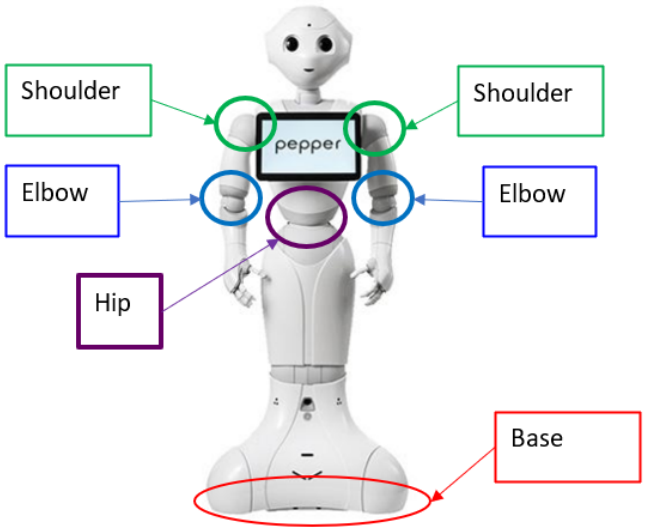


Figure 3: Pepper the humanoid robot. The project we implemented controls 5 joints on the pepper robot: Left/Right Shoulder, Left/Right Elbow, Hip and Knee, and controls the base of the Pepper robot. The joints and base we control are marked in the image above[1].

To enable the pepper robot to imitate human motion, we control 5 revolutional joints that are most relevant to pepper dance motion. We also control the base motion to let the pepper robot rotate follow 2D rotation and translation of the user.

Pepper robot has been used in entertainment dancing in ,

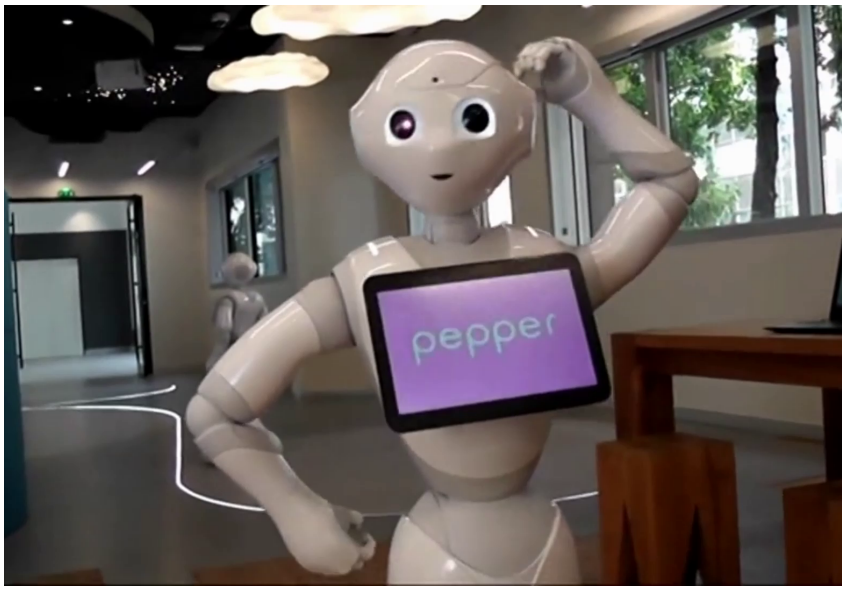


Figure 4: Pepper robot dance demonstration image[3].

This example demonstrates that the pepper robot has the potential to closely imitate human motion. There are several important features let use choose Pepper robot:

- Arm flexibility: joint controller of robot can achieve pretty high rotational speed to follow human motion in real-time.
- Large joint space: the pepper robot has a very large space of feasible joint values. This enables the pepper robot to imitate human motion.

Because of time and space constraints, we use the pepper robot entirely in simulation. We details the setup of simulation environment and control library we used:

- Physics engine: we used PyBullet[4] as the physics engine, the Bullet engine has a state of the art simulation speed and accuracy. It is practical to simulate complex articulated bodies in this physics engine.
- Robot control library: we used qiBullet[5] robot control library developed by SoftBank company, this library provides an interface to directly control robot joint angles, the joint value commands are executed by built in PD controllers in Bullet.

We draw the following diagram showing the simulation environment setup:

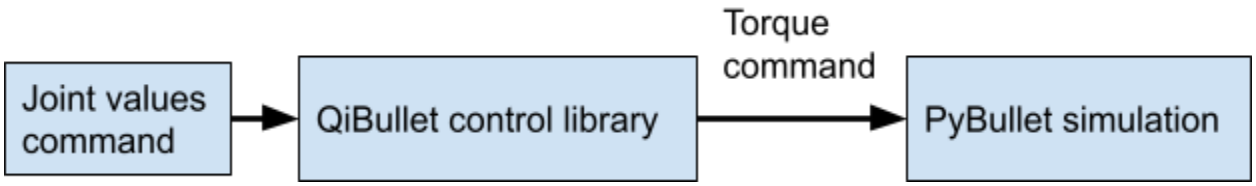


Figure 5: simulation environment setup, user input pepper robot joint values, qiBullet calculate torque command, pybullet simulate pepper robot motion.

The joint values command is the output of our system. This user interface is intuitive and can transfer to real robot motion. The output of our motion retargeting system is

Project Environment Setup

We used several hardware equipments in our project, here is a table showing the these equipments:

Table 1: Hardware equipments used in our project

Hardware component	Detailed type	Function
Screen	Toshiba TV (42.5 inches) [6]	Display robot simulation motion in real-time
Kinect camera	Kinect v1[2]	Capture human motion

Laptop	Legion Y540-151RH [7]	Real-time robot simulation, note this laptop is equipped with Nvidia GeForce RTX 2060
Web camera	Webcam 1080p HD Computer Camera [8]	Display user motion motion

Here is a diagram showing the setup of our testing environment.

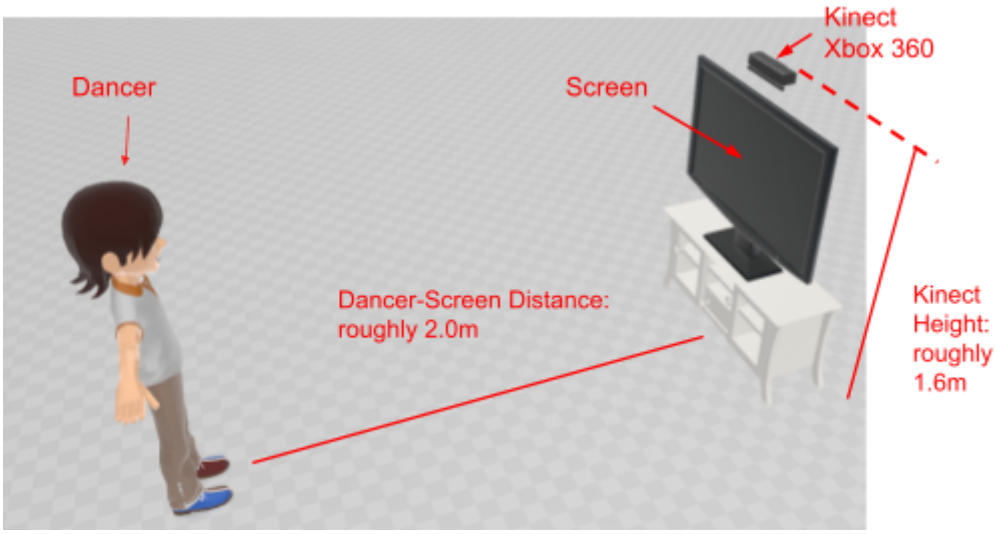


Figure 6: testing environment setup.

As is shown in Figure 6, the testing environment setup is as follows:

- The Kinect camera is mounted upon a screen, with its height with respect to the ground to be roughly 1.6m.
- The screen is 0.96 meter in width and 0.57 meter in height, which ensures the dancer can clearly see what is being shown on the screen.
- The initial position of the dancer is about 2.0 meter in front of the screen. The precise initial position varies from person to person, and is determined in the calibration section in the Graphic Interface Design section below. By adjusting the initial position, we ensure that the Kinect sensor can capture all body keypoints (definition of body keypoints is in Motion capture system section below).

Other requirements:

- Lighting condition: to have the Kinect camera working appropriately, we recommend users set up the equipment with indoor light condition. We have not tested kinect camera performance outdoors but are concerned that motion capture accuracy can decrease.
- Number of users: we currently only support one user in the system, an extra number of people in the motion capture region can distract motion capture results.

Motion capture system

The motion capture system used the built-in library of Kinect v1. Placement of the Kinect sensor has been described in the Project Environment Setup section above. Note that the output of the Kinect motion capture system is **3D human keypoints**. The key points detected by the Kinect camera is shown below:

Figure 7: 3D human key points detected by the Kinect camera[9].

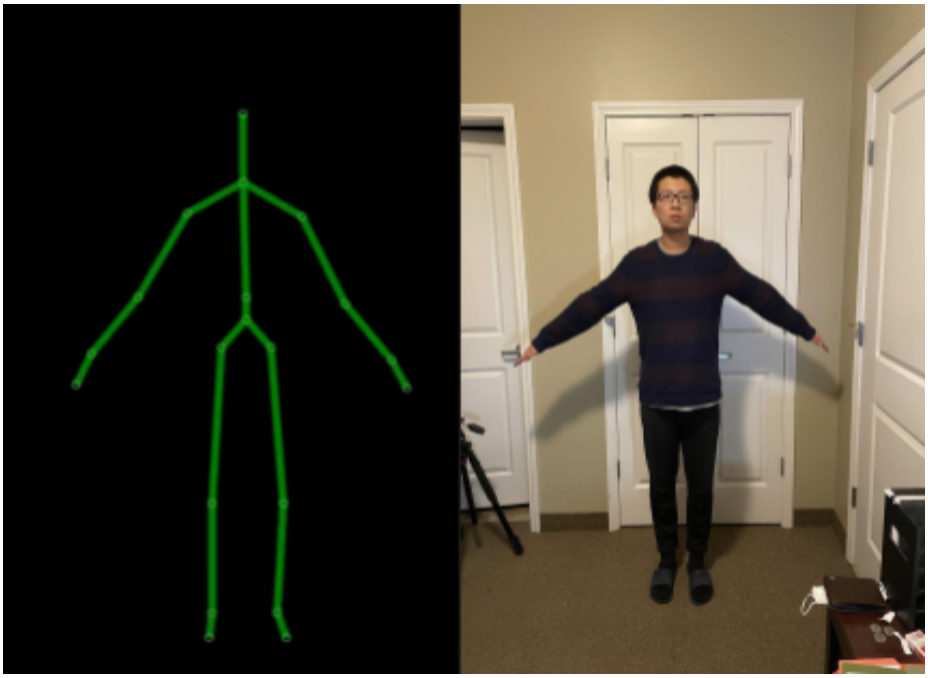


Figure 8: right is an image of dancer pose, left is the 3D skeleton captured by Kinect sensor

Note that the motion capture system is running on a process independent of physics simulation. To achieve inter-process communication, we used UDP communication protocol and windows socket. Because both processes run on the same machine, the missing data can be ignored. We implemented data transmission in an asynchronous way such that the motion retargeting system will use the most recent skeleton at each iteration.

Motion retargeting system

Motion retargeting system receives human 3D skeleton detected in Motion capture system, and aims to generate a set of robot joint values that can be used to control the Pepper robot. This is the most important and complex system in our project. Here is a overview diagram of the motion retargeting system, detail of each components shown in this diagram will be discussed below:

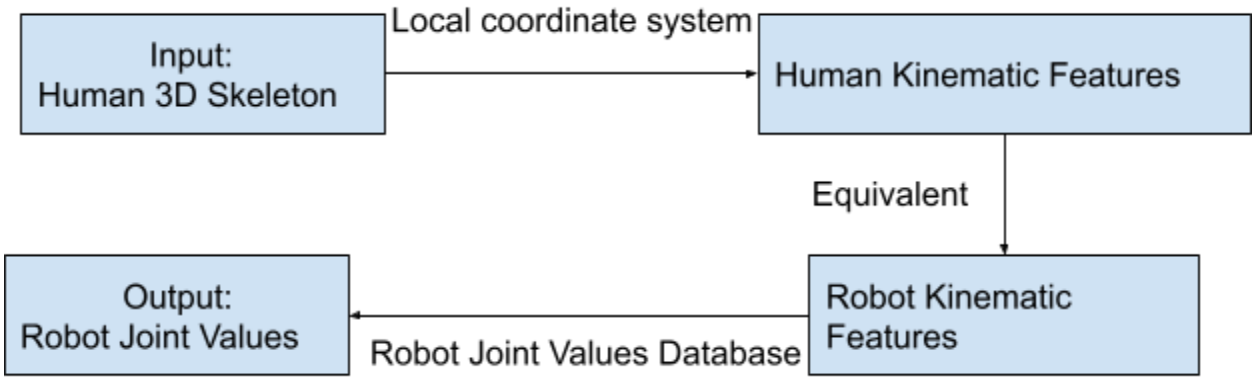


Figure 9: Algorithm overview diagram.

Input:

The input of the motion retargeting system at each time step is a 3D human skeleton detected in the Motion capture system, note that we only use key points relevant to dance. A figure showing the relevant key points is shown below:

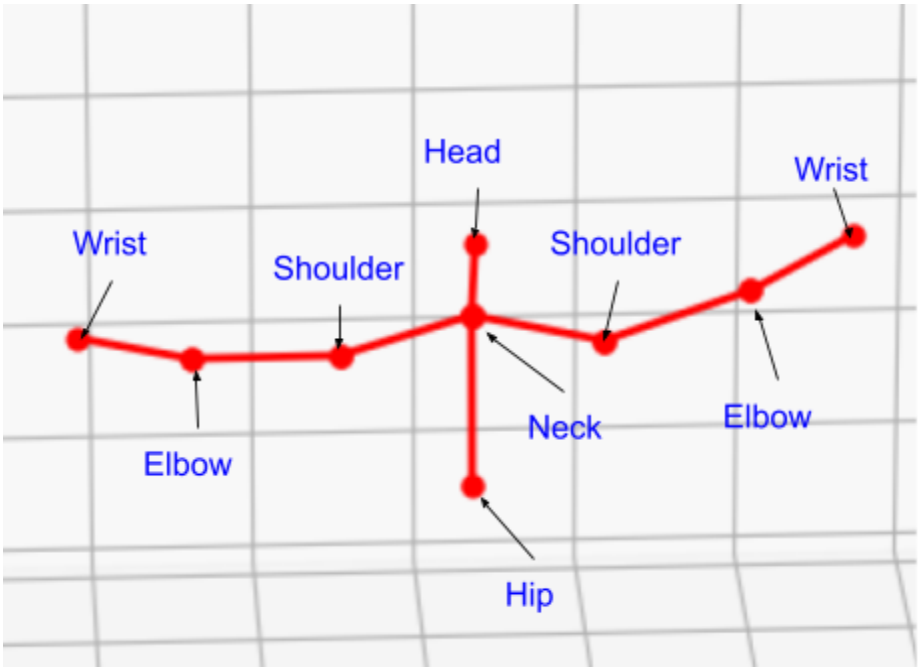


Figure 10: key points used for motion retargeting.

As shown in figure 10, we only used key points in the upper human body for motion retargeting, the knee and hip key points shown in figure 7 are not put in use.

Step1: definition of local coordinate frame

We use the Hip joint position as kinematic root position and we define notations on joint positions:

Joint name	Notation	Joint name	Notation
Hip	$\vec{Hip}$	Head	$\vec{Head}$
Left shoulder	$\vec{LS}$	Right shoulder	$\vec{RS}$
Left elbow	$\vec{LE}$	Right elbow	$\vec{RE}$
Left wrist	$\vec{LW}$	Right wrist	$\vec{RW}$

We define the robot joint local coordinate frame in a way invariant to robot translation and rotation. The following image shows the reason for the invariance in robot motion.

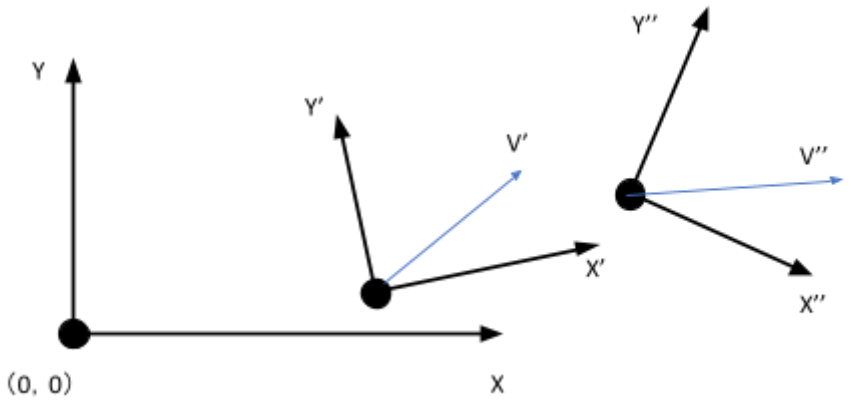


Figure 11: illustration of local coordinate system in 2D. X-Y frame is a global frame, X'-Y' frame and X"-Y" frame are local frames after rigid motion. The representation of **unit vectors** V' and V" are different in the X-Y frame. However, their representations are the same in each local frame.

Left shoulder local coordinate:

$$\hat{y}_{LS} = \frac{\vec{LS} - \vec{Head}}{\|\vec{LS} - \vec{Head}\|},$$
$$\hat{x}_{LS} = \frac{\hat{y} \times (\vec{Head} - \vec{Hip})}{\|\hat{y} \times (\vec{Head} - \vec{Hip})\|},$$
$$\hat{z}_{LS} = \hat{x}_{LS} \times \hat{y}_{LS}.$$

Left elbow local coordinate:

$$\hat{y}_{LE} = \frac{\vec{LE} - \vec{LS}}{\|\vec{LE} - \vec{LS}\|},$$
$$\hat{x}_{LE} = \frac{\hat{y} \times (\vec{LS} - \vec{Head})}{\|\hat{y} \times (\vec{LS} - \vec{Head})\|},$$
$$\hat{z}_{LE} = \hat{x}_{LE} \times \hat{y}_{LE}.$$

The right shoulder and elbow local coordinate frames are defined symmetrically. The hip joint uses the world coordinate frame.

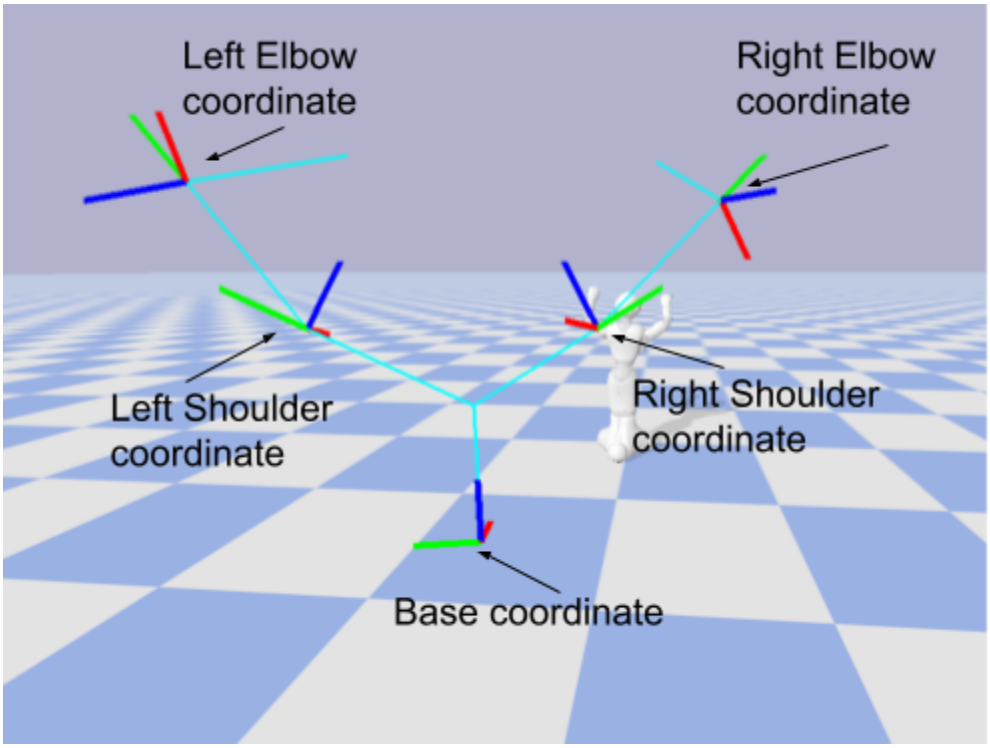


Figure 12: The back view of the 5 coordinate systems on human/robot: base coordinate, Left / Right Shoulder coordinates, Left / Right Elbow coordinates. The red lines in the image represent x-axis for local coordinates, green lines represent y-axis, blue lines represent z-axis.

Step 2: definition of human kinematic features and robot kinematic features

After the coordinate systems have been defined, we define a set of kinematic features to uniquely represent human kinematic poses. This set of kinematic features are invariant to both translational and rotational motion. The same set of kinematic features can be defined and applied to the Pepper robot. We will use these features to inversely compute robot joint values given human 3D skeleton.

We define the kinematic features associated to each joint in the system:

Left shoulder features:

$$LS_{f1} = (\vec{LE} - \vec{LS}) \cdot \hat{x}_{LS},$$
$$LS_{f2} = (\vec{LE} - \vec{LS}) \cdot \hat{y}_{LS},$$
$$LS_{f3} = (\vec{LE} - \vec{LS}) \cdot \hat{z}_{LS}.$$

Left elbow features:

$$LE_{f1} = (\vec{LW} - \vec{LE}) \cdot \hat{x}_{LE},$$
$$LE_{f2} = (\vec{LW} - \vec{LE}) \cdot \hat{y}_{LE},$$
$$LE_{f3} = (\vec{LW} - \vec{LE}) \cdot \hat{z}_{LE}.$$

Hip joint features:

$$Hip_{f1} = (\vec{Head} - \vec{Hip}) \cdot \hat{x}_{world},$$
$$Hip_{f2} = (\vec{Head} - \vec{Hip}) \cdot \hat{y}_{world},$$
$$Hip_{f3} = (\vec{Head} - \vec{Hip}) \cdot \hat{z}_{world}.$$

We hold the assumption that the human kinematic features are **equivalent** to the robot kinematic features. With this assumption, we can derive robot joint features based on input human 3D skeleton. The remaining work for the motion retargeting system is to find the link between robot kinematic features and the robot joint values.

Robot base motion:

To get robot base control commands, we initialize the robot control frame from initial several skeletons. We use two additional joints: left ankle and right ankle. Their positions are denoted as $\vec{LA}$ and $\vec{RA}$. Given several initial skeletons, the base coordinate frame is defined using,

$$\hat{y}_{Base} = \frac{\vec{LA} - \vec{RA}}{\|\vec{LA} - \vec{RA}\|},$$
$$\hat{x}_{Base} = \frac{\hat{y}_{Base} \times (\vec{Head} - \vec{Hip})}{\|\hat{y}_{Base} \times (\vec{Head} - \vec{Hip})\|},$$
$$\hat{z}_{Base} = \hat{x}_{Base} \times \hat{y}_{Base},$$
$$\vec{o}_{Base} = (\vec{LA} + \vec{RA})/2.$$

Here $\vec{LA}$ and $\vec{RA}$are average positions over the first several skeletons. The base control command are calculated with respect to the base coordinate system,

$$x_{Base} = (\vec{Hip} - \vec{o}_{Base}) \cdot \hat{x}_{Base},$$
$$y_{Base} = (\vec{Hip} - \vec{o}_{Base}) \cdot \hat{y}_{Base},$$
$$\theta_{Base} = atan2(\hat{y}_{Base} \cdot (\vec{LS} - \vec{RS}), \hat{x}_{Base} \cdot (\vec{LS} - \vec{RS})) - \frac{\pi}{2}.$$

We use 2D translation and rotation commands to control the robot base.

Step 3: robot kinematic features database collection

Given joint positions of the robot, we can get the robot key points, and can calculate robot kinematic features based on the definition in step 2. We collect the database of the mapping from robot joint values to robot kinematic features using the following procedure:

```
Function collectDatabase(joint_name):
    joint_value_range = robot.value_range(joint_name)
    database = {}
    for value in joint_value_range:
        robot.set_joint(joint_name, value)
        feature = robot.kinematic_features()
        database[value] = feature
    return database
```

Note that each joint has two revolutional dimensions on the pepper robot. A table showing correspondence is in the Appendix. The returned database is a dictionary from two joint values to three kinematic features,

```
database example:
    joint_value1, joint_value2:
    feature1, feature2, feature2;
```

We define the local coordinate frame in a way such that robot motion is independent of the motion of other parts. So we collect the joint value database independently for each joint.

The value range of each joint are uniformly spaced values in the robot joint limit range, we have a parameter to decide the number of points to choose. We currently choose 20 points for each revolutional dimension and there are $20 \times 20 = 400$values used for each joint.

Step 4: mapping from kinematic features to robot joint values

To find joint joint values from the data based, we do nearest neighbor search to find robot kinematic features given human kinematic features.

For each joint, the database takes in the human kinematic feature with three parameters. Using Euclidean distance on kinematic features, we find the robot kinematic feature with the smallest distance. Then we can retrive two joint values for this joint from the database.

To speed up the search procedure, we use k-d tree [14] to load the database. We detailed the procedure in the following code.

```
load joint_value_matrix (n rows, 2 columns, matrix)
load kinematic_feature_matrix (n rows, 3 columns, matrix)
joint_kd_tree = KDTree(kinematic_feature_matrix)

Function search_joint_value(kinematic_feature):
     index = joint_kd_tree.nearest(kinematic_feature)
     return joint_value_matrix[index]
```

We will use `search_joint_value` to search joint values for each joint. We will output joint values commands to the robot control library.

```
LeftShoulder: [pitch angle, yaw angle]
LeftElbow: [pitch angle, yaw angle]
RightShoulder: [pitch angle, yaw angle]
RightElbow: [pitch angle, yaw angle]
```

The interface we used is directly from that implemented using the PD controller from Bullet. The robot will command these commands moving to target joint angles.

Motion Smooth System

Because of sensory noise, the robot motion can be non-smooth in real-time. To improve the dance performance of robots, we implement an algorithm to smooth and replay robot motion. The smoothing algorithm takes a sequence of robot joint values calculated in real-time. It will output a sequence of smoothed joint values. The algorithm used numerical derivative to select keyframe in a sequence of joint values:

```
Function smooth_motion(joint_values_seq):
     smoothed_seq = [joint_values_seq[0]]
```

```
        for joint_values in joint_values_seq:
            if derivative(joint_values)>=eps_1 or
                norm(joint_values-smoothed_seq[-1])>=eps_2:
                smoothed_seq.append(joint_values)
    return smoothed_seq
```

We used five-point formula to compute the numerical derivative on joint values,

$$\dot{q}_k \approx \frac{1}{12}[-25q_k + 48q_{k+1} - 36q_{k+2} + 16q_{k+3} - 3q_{k+4}]$$

Here $\dot{q}_k$ is approximated derivative of joint values at time frame *k*.

The norm on joint values is the Euclidean norm. In practice, we used `eps_1=0.4` and `eps_2=0.5`.

Graphic Interface Design

In order to help users get familiar with our system and play their own dance, we developed a graphical user interface (GUI) for the system. The system was implemented based on PyQt5[16].

Below is the state diagram of our GUI. The user would start from a welcome page. Then, the user should follow the calibration guide to move to a proper position to start his dance. If the user is not familiar with our system, he may choose to watch a tutorial video before the free style section. Also, he may choose to skip the tutorial. In the free style section, users can dance with allowed motion, and our system would save a video for the user. After the freestyle, the user can play his video or start another dance.

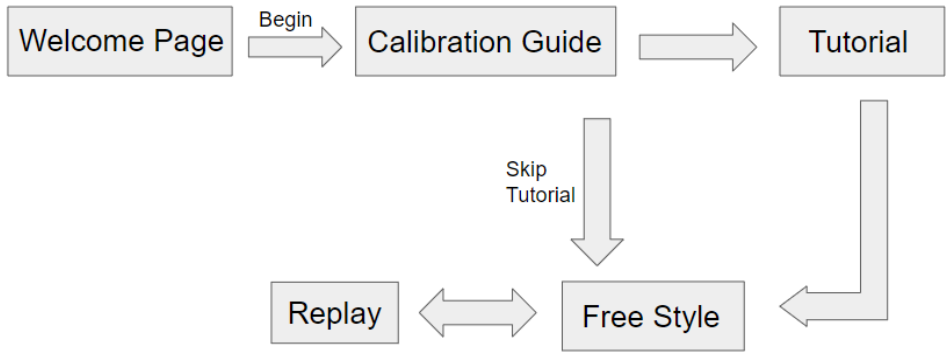


Figure 13: GUI state diagram

The user would start from the welcome page. In this page, we showed our project and team members' names. We also listed all pages on the upper left corner.

Figure 14: Welcome Page of GUI

The calibration page gives the user a suitable position to start his dance. In the calibration page, the user skeleton is shown in red. The user needs to move to the ball's position so that the kinect could catch the whole body of the user.

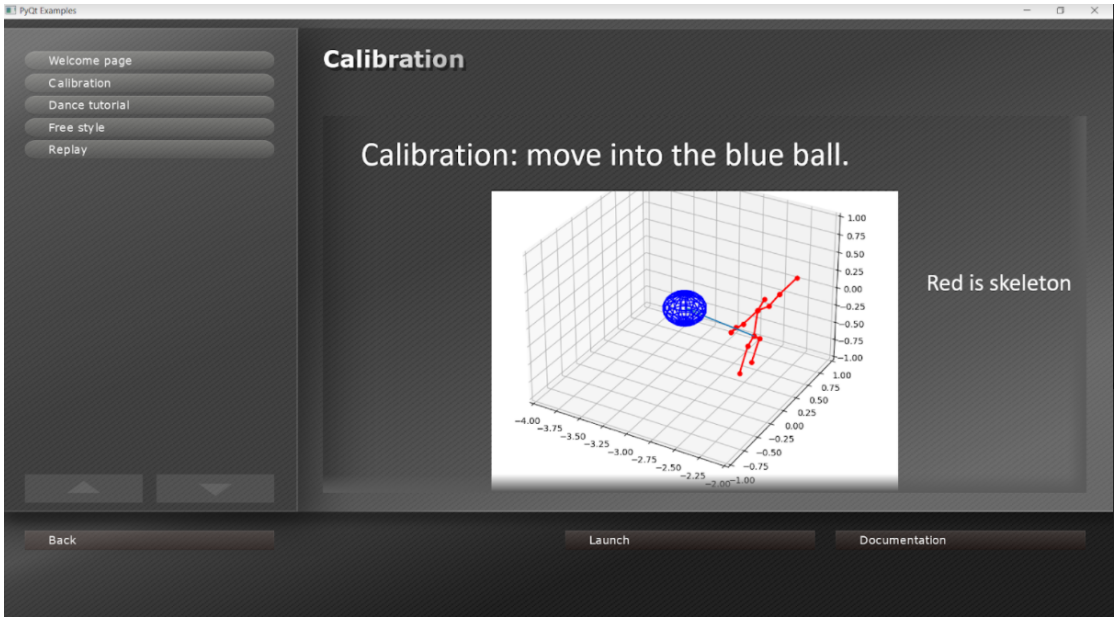


Figure 15: Calibration Page.

By pressing the “Launch” button in the Calibration page, the calibration program will start:

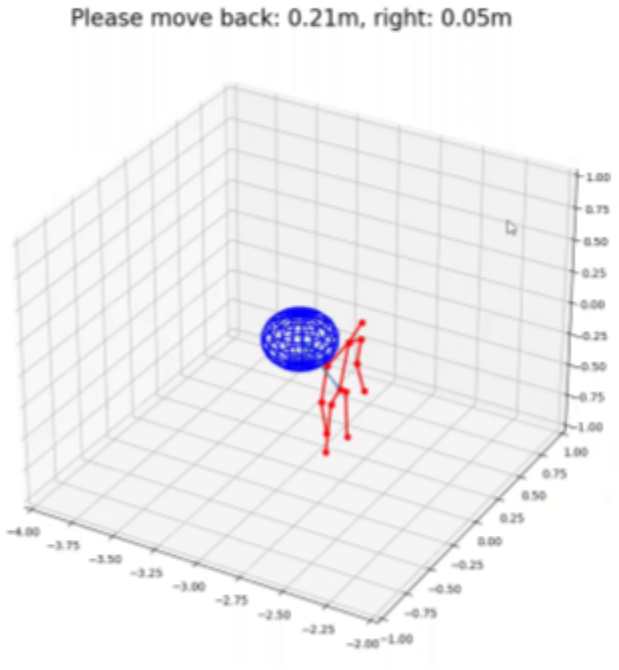


Figure 16: Calibration in progress. The human skeleton is shown in red and the target initial position is the blue ball. The title of the image is an instruction showing the direction the dancer should go. In this case, “move back: 0.21m, right: 0.05m”

After calibration is finished, the dancer can choose whether or not to go through the short tutorial. The tutorial aims to help the dancer understand how our system works.

Figure 17: tutorial page. By pressing the Launch button, the tutorial program will start.

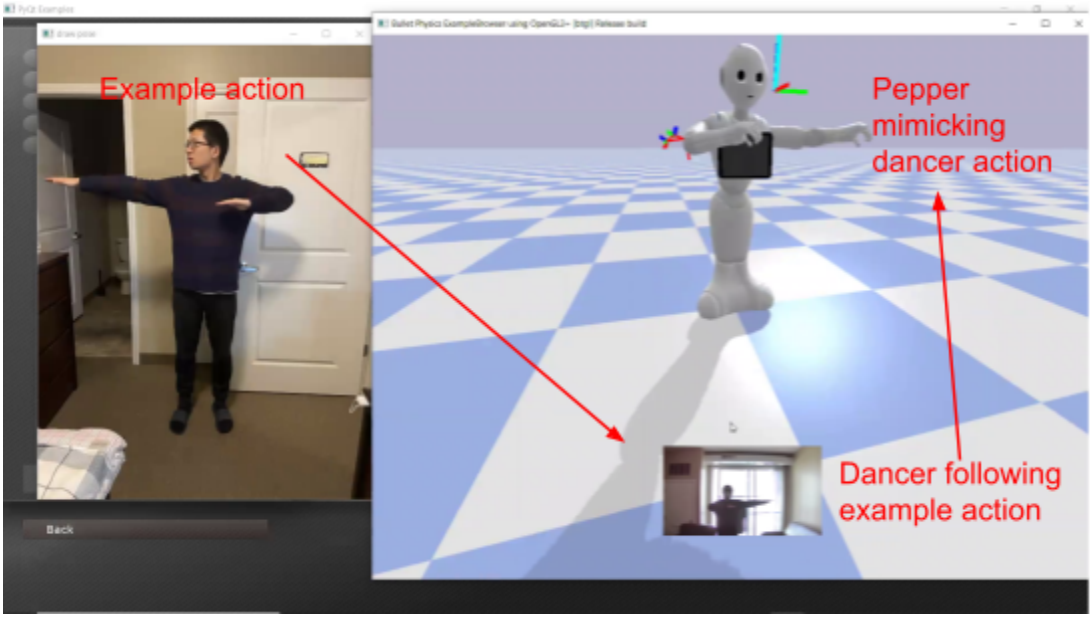


Figure 18: tutorial in progress. Left is the window showing example poses. When the dancer is following the example poses, the Pepper robot shown on the right will mimic the dancer's action.

After the tutorial is completed, the dancer can do a 2-minute freestyle dance. Here is an image of the freestyle page:

Figure 19: the freestyle page. By pressing the Launch button, the freestyle program will start

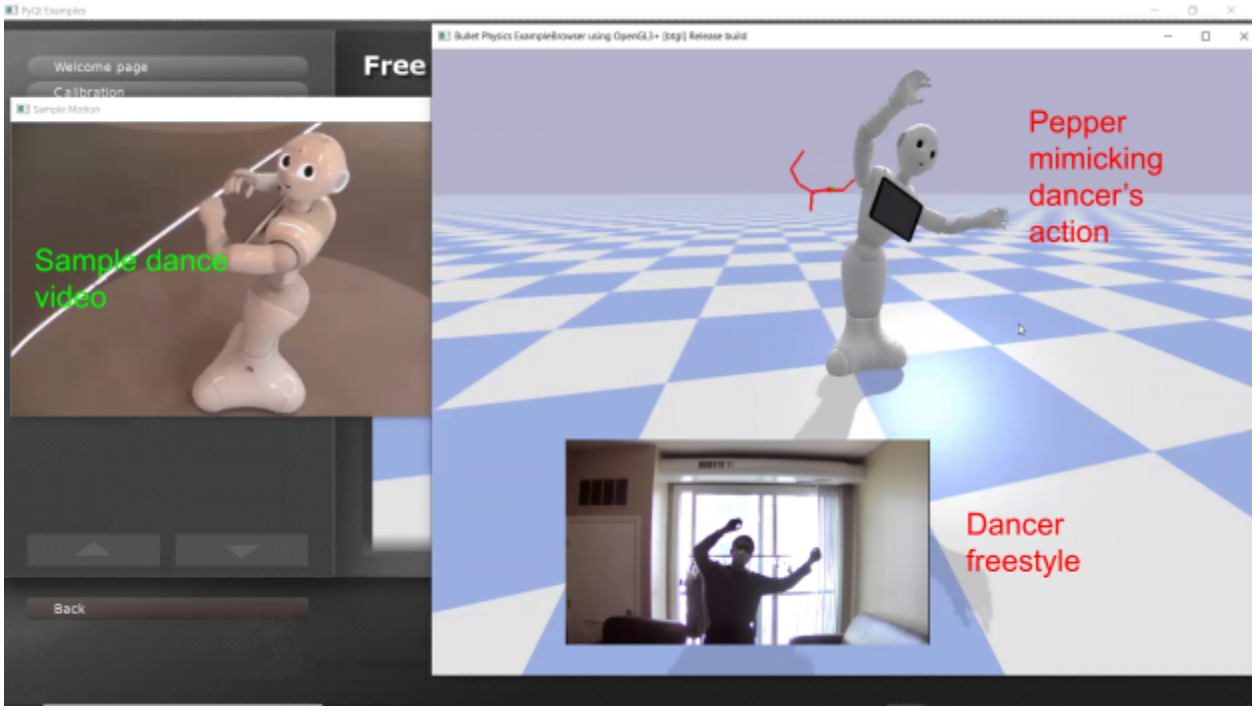


Figure 20: freestyle in progress. On the left is a window showing a sample dance video. This video aims to give dancers some hint in case they cannot come up with freestyle action. As the dancer is freestyling, the Pepper robot will follow the freestyle dance.

During the freestyle dance, we will record the dancer's motion, and replay a smoothed version of the motion after dancing is completed:

Figure 21: the replay page. By pressing the Launch button, the replay program will start.

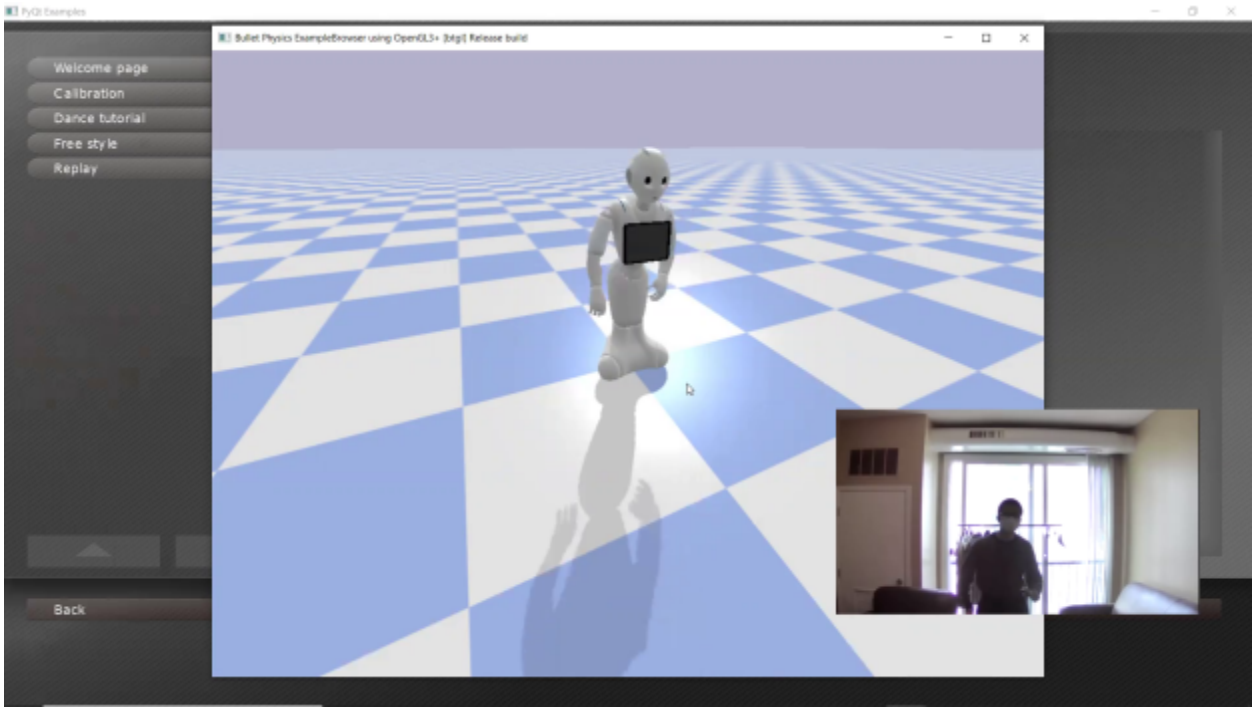


Figure 22: replay in progress. In the middle of the image, the robot is replaying the freestyle dance with the motion smoothed.

Test and Evaluation

Motion capture results

The Kinect v1 motion capture system satisfied the needs of our project. The refresh rate achieved 30 fps and the motion capture quality is reasonable. For each joint, the offset between joint estimated position and ground truth position is typically smaller than 10cm [15]. The dance motion of the user can be captured by the Kinect v1 motion capture system.

Motion retargeting results

The result for the motion retargeting part can be divided into two parts:

- Quality: how good can Pepper robot mimic dancer action
- Speed: how fast can Pepper robot follow dancer action

Quality

We applied in total 18 different **static** dancer actions to test the quality of motion retargeting. These actions include a combination of shoulder, elbow and waist motions. Full tables for the 18 dancer poses and corresponding robot poses are available in appendix table 2 and appendix table 3. Note that the dancer poses and robot poses are mirror images of each other.

Here are 2 examples of robot mimicking dancer actions.

Figure 23: examples of robot mimicking dancer actions. Left is a dancer pose where all key points can be detected. Right is the robot mimicking the dancer pose.

Figure 24: examples of robot mimicking dancer actions. Left is a dancer pose where one hand of the dancer is occluded by body. Right is the robot mimicking the dancer pose.

Our Pepper robot can mimic 17 / 18 dancer poses, and failed in one case where human keypoints cannot be well-detected by the Kinect sensor. Here is the failed test case:

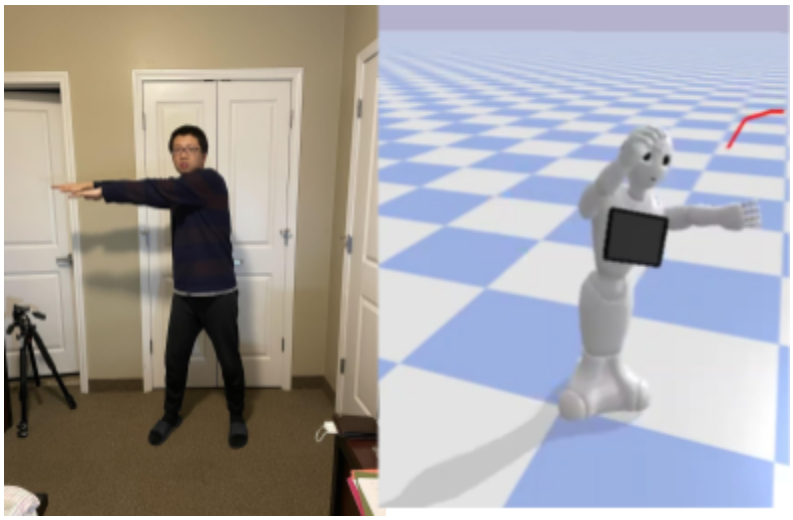


Figure 25: the test case that our system cannot mimic human pose properly. The two arms of the dancer are too close together, and key points cannot be properly detected by the Kinect sensor.

In summary, our motion retargeting system is well-implemented and can properly mimic human dance as long as the human keypoints can be detected.

Speed evaluation

We applied the 18 test cases in sequence, and collected the start and end time for dancer and robot to be in a specific pose.

Table 2: start and end time for specific pose

Index	Dancer Start (s)	Dancer End (s)	Robot Start (s)	Robot End (s)
simple 1	22.031	25.983	22.608	26.359
simple 2	26.984	31.084	26.958	31.490
simple 3	31.910	36.390	32.106	37.041
simple 4	37.391	41.138	37.725	41.608
simple 5	42.072	46.175	42.410	46.655
simple 6	47.584	51.292	47.390	51.694
simple 7	51.893	56.249	52.361	56.634
simple 8	56.651	1:01.331	57.233	1:01.884
simple 9	1:02.199	1:11.521	1:02.484	1:12.218
difficult 1	1:12.321	1:16.419	1:12.684	1:17.306
difficult 2	1:17.486	1:21.429	1:17.973	1:22.021
difficult 3	1:22.867	1:26.622	1:23.089	1:27.070
difficult 4	1:27.282	1:31.704	1:27.738	1:32.285
difficult 5	1:32.705	1:37.569	1:33.899	1:38.093
difficult 6	1:38.304	1:41.906	1:38.701	1:42.623
difficult 7	1:42.831	1:46.807	1:43.356	1:47.679
difficult 8	1:48.060	1:52.379	1:48.278	1:52.652
difficult 9	1:53.247	1:55.299	1:53.720	/

By subtracting the “Robot Start” column against the “Dancer Start” column, we can get the time it takes for the robot to stabilize in a pose that can mimic the human dance. Here is a plot showing the time:

Here is a histogram for the time difference:

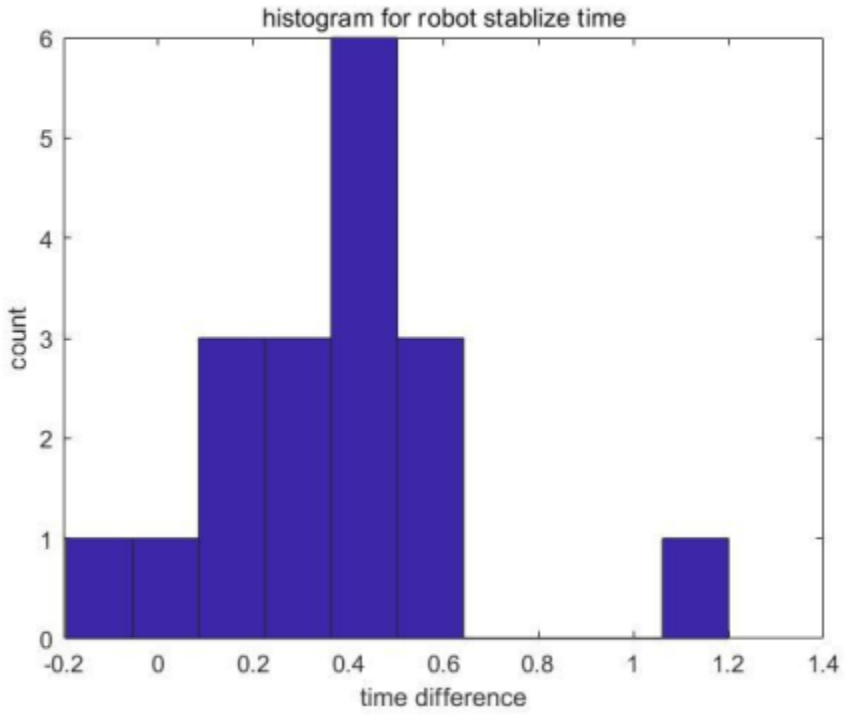


Figure 26: the histogram for the robot stabilization time.

The average time is 0.3819s, with a standard deviation of 0.2898s. There exists a 1.2s outlier, which belongs to the failed test case “difficult 5”. The stabilization time for all other test cases are within 0.6, indicating that our retargeting system has good runtime performance.

Motion smoothing results

To visualize the performance of the motion smooth algorithm, we plot the trajectory of the right wrist joint before and after smoothing. We note that the trajectory after smoothing does not have vertical vibration compared to trajectory before smoothing. The performance of the smoothing algorithm can also be recognized in demo video.

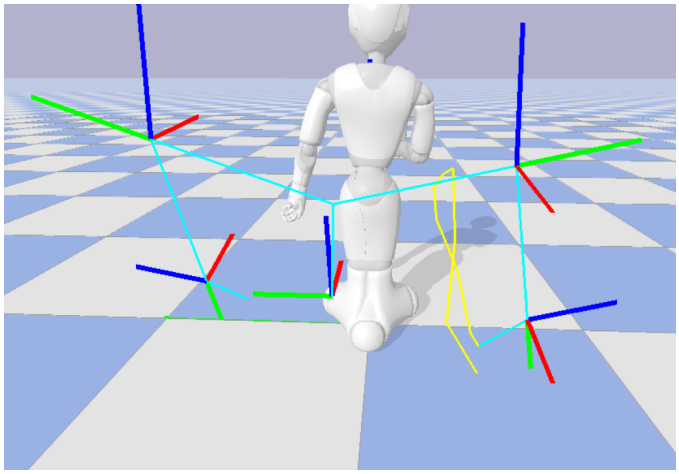


Figure 27: robot wrist trajectory before smoothing.

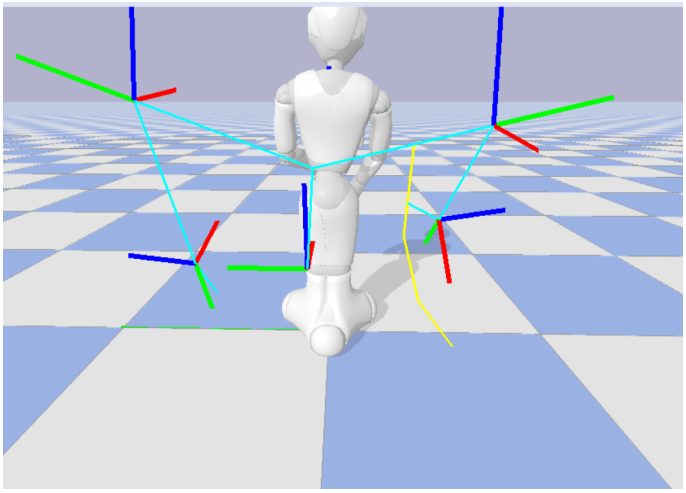


Figure 28: robot trajectory after smoothing.

Findings and Discussion for Next Development Iteration

In this project, we have some lessons learned from using the motion capture and simulation software.

Motion capture system: Now we are using the motion capture system built in Kinect v1 to detect human skeletons. Its detection speed is good. And the invariant kinematic features defined by us are effective that our robot motion resembles the detected human motion.

Simulation software: But the PD controller we used makes the robot motion not smooth. When we give a command to the PD controller, after the robot completes the motion, it will stop until it receives the next command. Therefore, our robot motion is not smooth enough.

Next development iteration: In our next development iteration, we plan to use common camera input or video input to get human motion. This can make our project have fewer equipment restrictions. We plan to use OpenPose[11] to capture human motion for camera input or video input.

Our motion smooth system is not done in runtime. For the next iteration, we plan to do smoothing in runtime.

Future Directions

In the future, we plan to change the robot to have a larger motion range. We only use a small range of motion of the Pepper robot. Because of robot joint limits, our robot can't imitate some human motion. For example, our Pepper robot can't bend down a lot and its hand can't reach its waist. We considered several alternative humanoid robot choices, like NAO robot[12].

We also plan to implement robot head and fingers control. If we can implement head tracking and hand, we can use similar kinematic features to control robot joints. Some vision-based motion capture systems can achieve high accuracy tracking of head orientation and fingers [11]. We will consider using these motion capture systems in the future.

References and Citations

[1] Pepper robot, https://robots.ieee.org/robots/pepper/, retrieved on April 29th, 2021

[2] Link to Kinect camera, https://en.wikipedia.org/wiki/Kinect, retrieved on April 29th, 2021

[3] Pepper robot dance, https://www.youtube.com/watch?v=JvwLLdTOpnw, retrieved on April 29th, 2021

[4] PyBullet, https://pybullet.org/wordpress/, retrieved on April 29th, 2021

[5] Busy, Maxime, and Maxime Caniot. "qiBullet, a Bullet-based simulator for the Pepper and NAO robots." arXiv preprint arXiv:1909.00779 (2019).

[6] Toshiba TV, https://tvna.compal-toshiba.com/us/en/models/43l511u18/, retrieved on April 29th, 2021

[7] Legion-Y540-15, https://www.lenovo.com/us/en/laptops/legion-laptops/legion-y-series/Lenovo-Legion-Y540-15/p/88GMY501214, retrieved on April 29th

[8] WebCamera, https://www.amazon.com/gp/product/B083QM78TV/ref=ppx_yo_dt_b_asin_title_o09_s00?ie=UTF8&psc=1, retrieved on April 29th

[9] Kinect skeleton. S. Motiian, P. Pergami, K. Guffey, C.A. Mancinelli and G. Doretto, “Automated extraction and validation of children’s gait parameters with the Kinect”, Biomedical Engineering Online, 14:112, pp 1- 32, 2015

[10] NAO robot, https://www.softbankrobotics.com/emea/en/nao, retrieved on April 29th, 2021

[11] Cao, Zhe, et al. "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields." IEEE transactions on pattern analysis and machine intelligence 43.1 (2019): 172-186.

[12] Definition of robot joint: https://www.britannica.com/technology/joint-robotics

[13] Image for local coordinate system: https://www.chegg.com/homework-help/questions-and-answers/task--length-distance-3d-space-point-3d-space-represented-using-three-coordinates-cartesia-q28964064, retrieved on April 29th, 2021

[14] K-d tree: https://en.wikipedia.org/wiki/K-d_tree, retrieved on April 29th, 2021

[15] Q. Wang, G. Kurillo, F. Ofli and R. Bajcsy, "Evaluation of Pose Tracking Accuracy in the First and Second Generations of Microsoft Kinect," 2015 International Conference on Healthcare Informatics, 2015, pp. 380-389, doi: 10.1109/ICHI.2015.54.

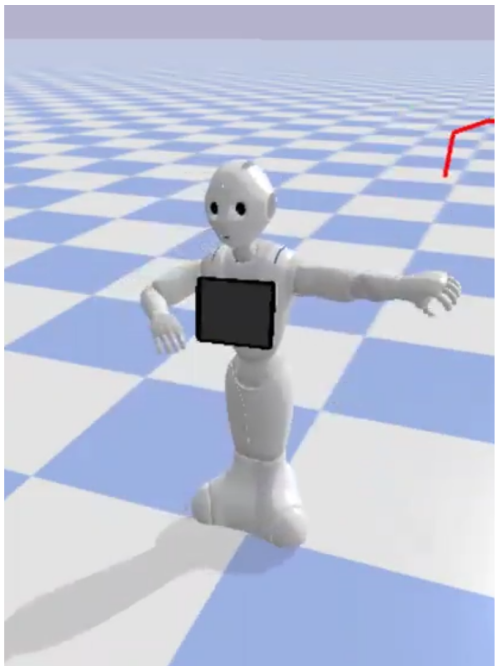
[16] PyQt, https://riverbankcomputing.com/software/pyqt, retrieved on April 29th, 2021

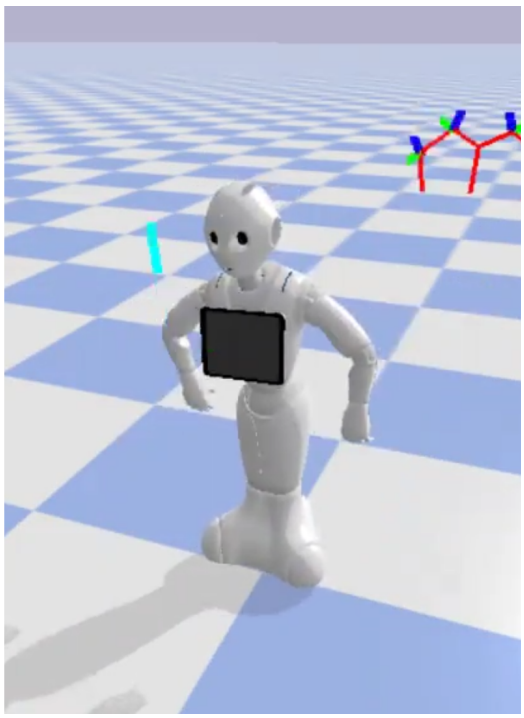
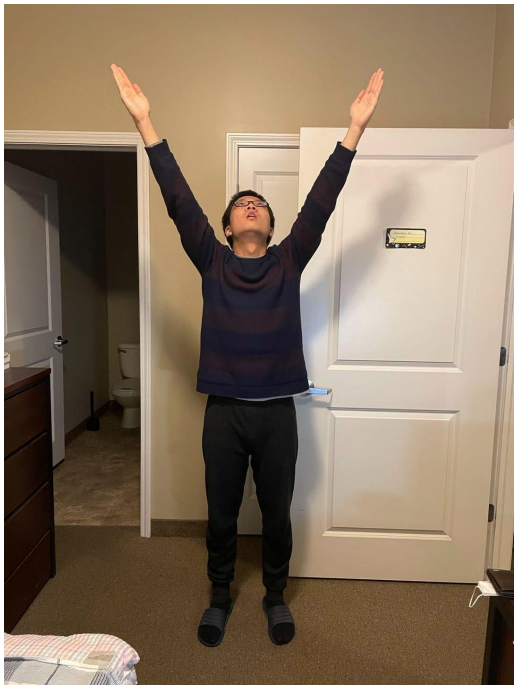
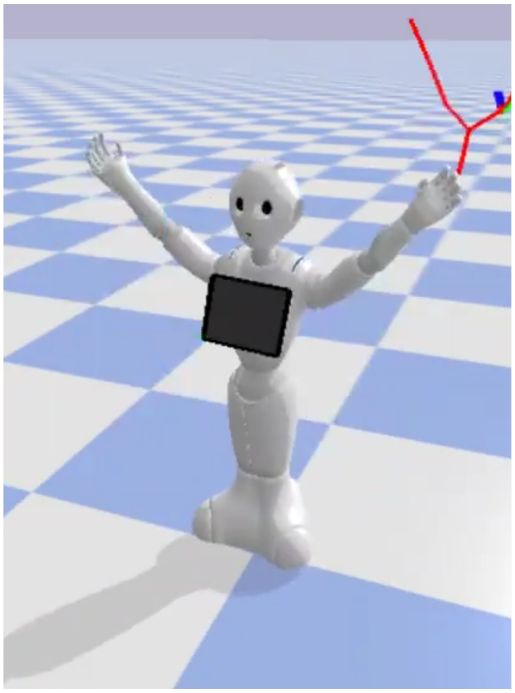
Appendix

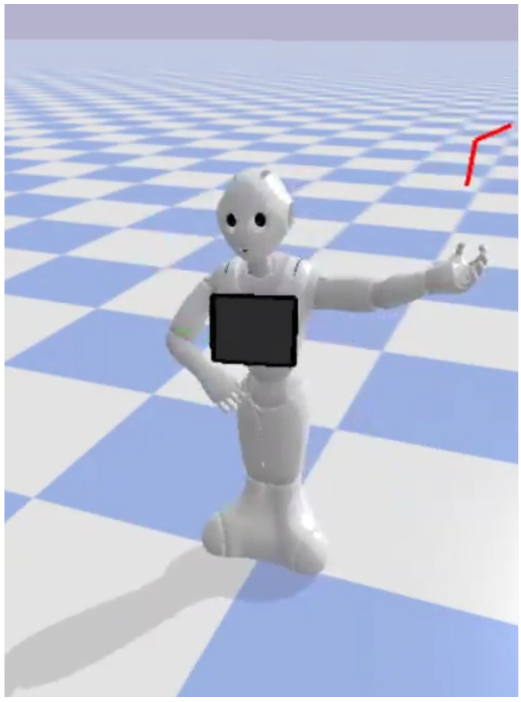
Appendix table 1: human joints and corresponding robot joints

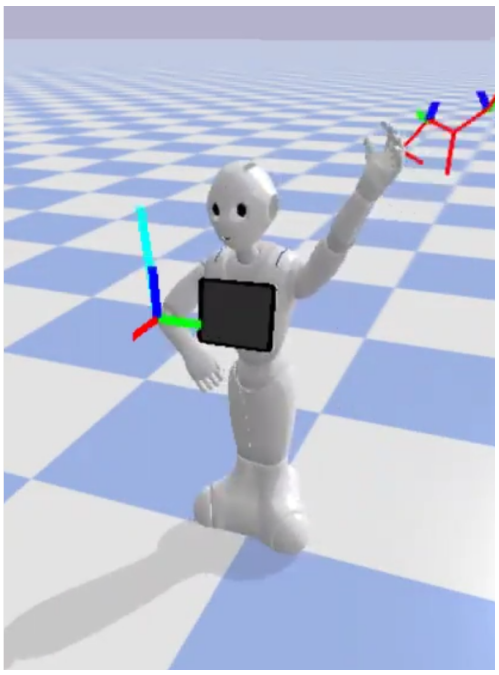
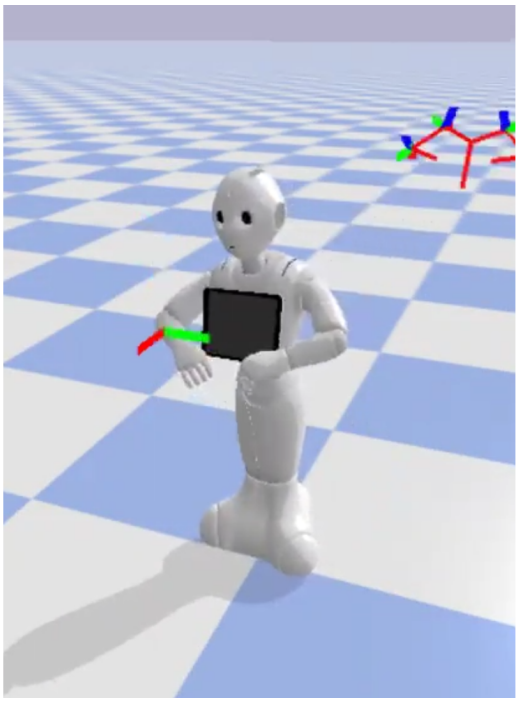
Human joints	Pepper robot joint pair
Left Shoulder	LShoulderPitch
	LShoulderRoll
Right Shoulder	RShoulderPitch
	RShoulderRoll
Left Elbow	LElbowYaw
	LElbowRoll
Right Elbow	RElbowYaw
	RElbowRoll
Hip	HipPitch
	HipRoll

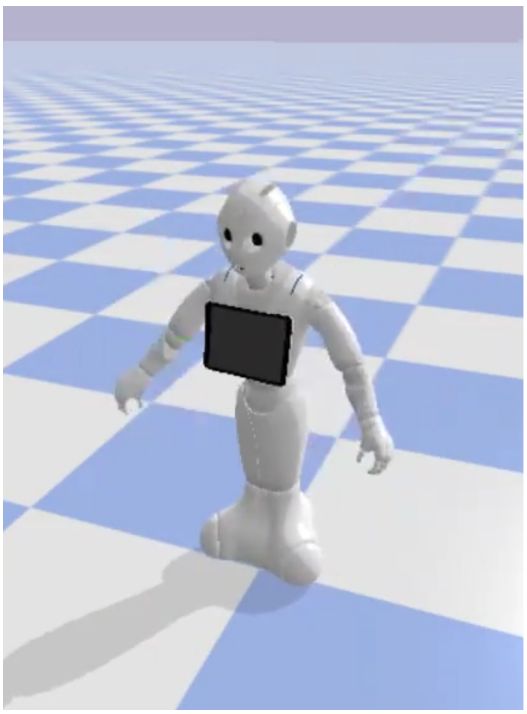
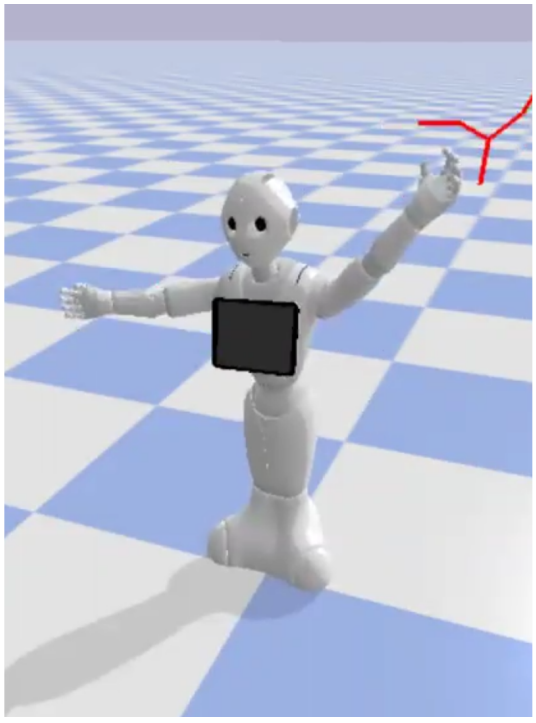
Appendix table 2: dancer pose and corresponding robot pose (simple)

Index	Dancer Pose	Robot Pose
simple 1		

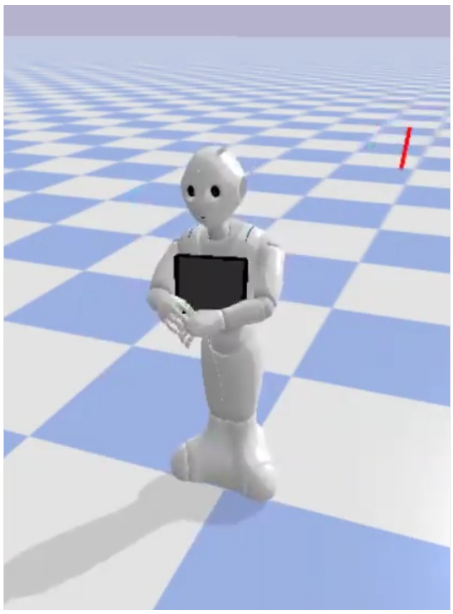
simple 2		
simple 3		

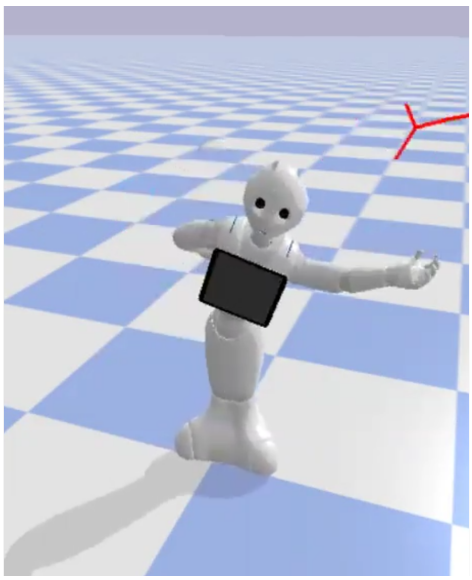
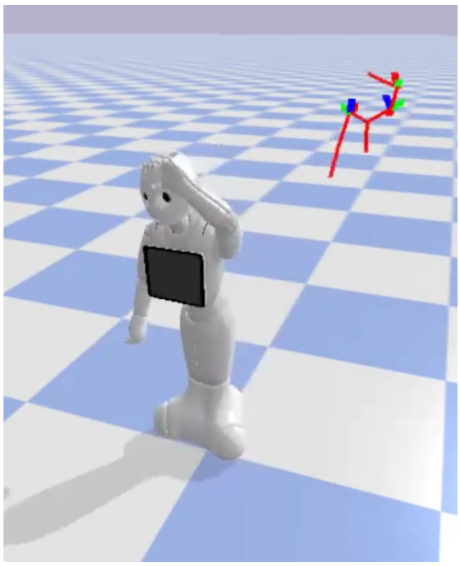
simple 4		
simple 5		

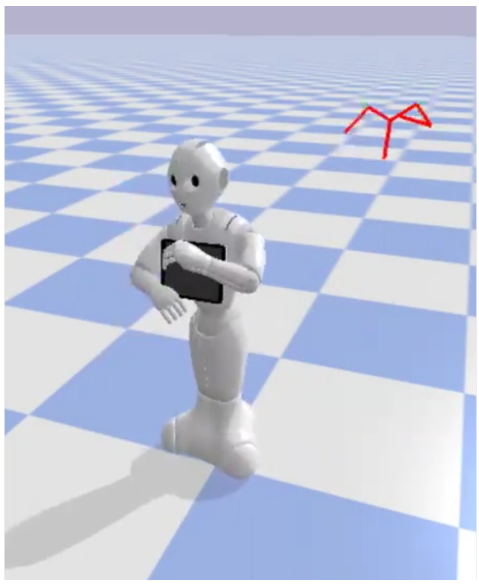
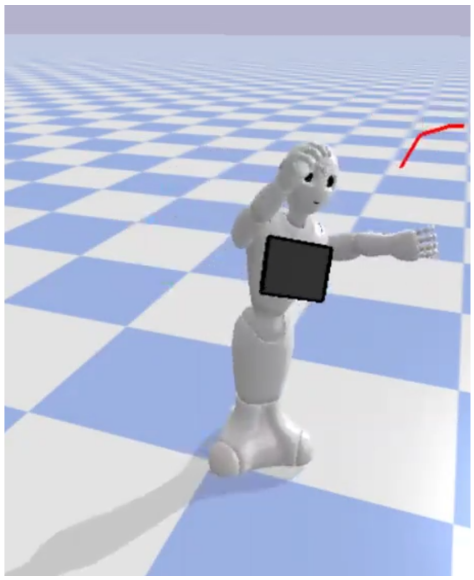
simple 6		
simple 7		

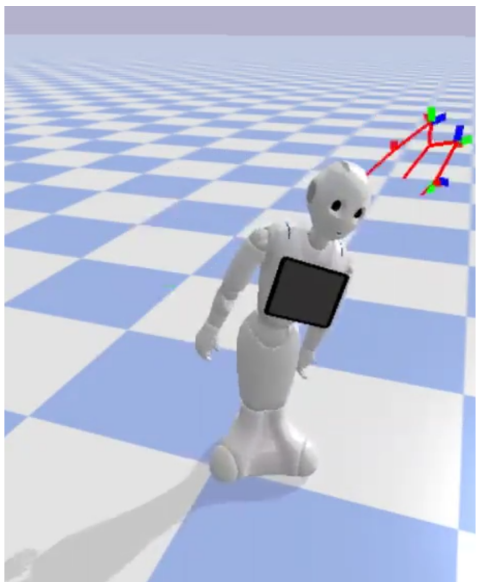
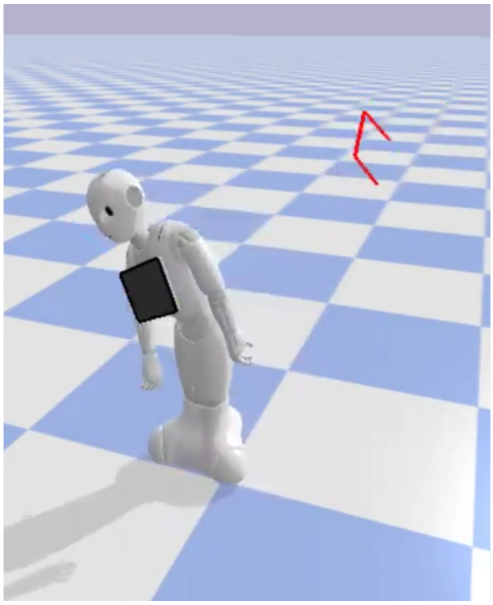
simple 8		
simple 9		

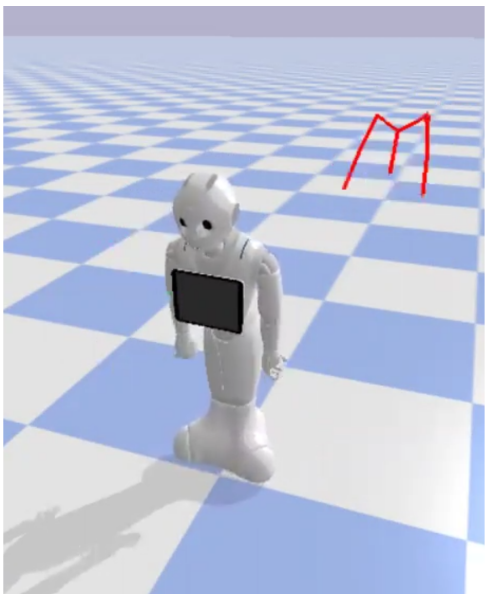
Appendix table 3: dancer pose and corresponding robot pose (difficult)

Index	Dancer Pose	Robot Pose
difficult 1		

difficult 2		
difficult 3		

difficult 4		
difficult 5		

difficult 6		
difficult 7		

difficult 8		
difficult 9