Are You Happy? Infinitely Looping Robotic Art

Jessica Han
Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, MA United States
jess7654@mit.edu

Raymond Wang
Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA United States
rlwang@mit.edu

***Abstract*—The repeatable and precise nature of robotics makes it a prime candidate for a new artistic medium: dynamic art installations. With the recent strides in robotics technology, robots are now capable of reliably performing manipulation tasks. In this paper, we present a framework to enable two robotic arms to perform infinitely looping "run and chase" dystopian-inspired art. In our system, we simulated two 7-degree-of-freedom manipulator arms in *Drake*. One robot draws with chalk on a table while the other robot "chases" the drawing and erases it. We present a full implementation of the most light-weight version of this system, details on implementation of a more robust iteration, and a high-level analysis of a full implementation of this system capable of running on hardware.**

I. Introduction and Motivation

Since the aftermath of World War I, artists have sought to epitomize the despair and melancholy that pervades the human experience. With the prominence of the industrial revolution, this melancholy has become accentuated by the newfound monotony of the everyday. Workers putting parts together at the assembly line, men digging at the pitheads of mines and women sewing at the jenny for hours on end – inspired some of the greatest dystopian artists of the 20th century.

That same tedium evidently still exists today, and many argue it's only exacerbated by the developments and societal changes that exist in the 21st century. We aim to illustrate that tedium through a new lens: robotics. With robotics, we can make the portrayal of monotony that was once restricted to canvas come to life in our 3D world. Since we humans also exist in the 3D world, we hope our portrayal will evoke stronger empathy and emotion than what might be possible with only painting.

II. Artistic Goal

Are you happy?'s endless action loop can be seen in Figure 1. The two robots alternate between using chalk to draw a smiley face on a table and using an eraser to erase the smiley drawn by the other robot. The robots then throw their respective objects and pick up the objects thrown by the other robot in between to switch "roles". Looping dystopian art must strike the balance between repetitiveness and randomness as repeating a truly identical loop would fail to humanize the robots, making them no different than the inanimate tools we see around us (ex. stoplights, fans, etc.). At the same time, a lack of repetitiveness fails to invoke the discomfort that comes from the hopelessness of the robots' endless fates. Thus, we added randomness into what would have been an otherwise identical loop by randomly generating the smiley faces to be drawn by the robots. Our fully realized system would also include randomness in the throwing trajectories of the objects.

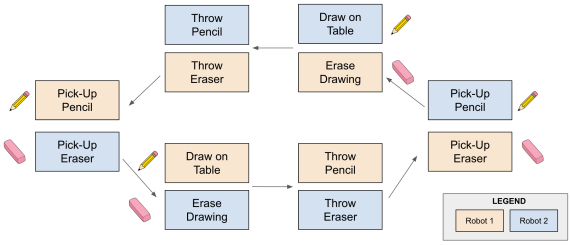


Fig. 1. Artistic Goal

III. Methods

A. Overview

While our system can be implemented in simulation with minimal feedback and primarily feed-forward controls (since we have access to the states of all bodies through the context), in order to realize our art on hardware we would need to develop a full-stack robotic system with perception, grasp selection, trajectory planning, trajectory following, and grasping. Thus, we worked backward from a full-fidelity, real-life system to a minimal viable simulation product to understand what incrementally building up this system would entail. A summary of our development pipeline plan is shown in Figure 2. We were able to successfully implement the Level 1 system (perception not needed) with the pipeline shown in Figure 3 and will discuss our learnings from this process along with a discussion on details for implementing the other levels in the following subsections.

B. Manipulation Station Setup

In this section, we describe the implementation of our multibody plant. The plant consists of two 8-DoF robotic arm and hand along with a chalkboard model. Each of the 8-DoF robotic arms were created by welding the Weiss Robotics Shunk WSG-50 gripper to a Kuka LBR-IIWA 7-Dof manipulator arm.

In most examples throughout this course, we used only one manipulator per plant. Since our setup required two arms, each

System	Subsystem	Implementation Difficulty		
		Level 1	Level 2	Level 3
Perception	**Detecting State of Chalk and Eraser**	States pulled from system plant.	Above head RGB camera - color detection and pose estimation based on basic geometry.	RGBD cameras - point cloud registration for pose estimation chalk and eraser models/known point clouds.
Picking	**Choosing Grasp**	Weld chalk and eraser to gripper frame.	2D grasp selection based on geometry (aim for centroid while keeping gripper parallel to sides of object).	Antipodal grasp selection based on point clouds.
	Grasping	Weld chalk and eraser to gripper frame.	Feedforward grasping with gripper position inside object dimensions (hold as hard as possible).	Closed loop impedance controller with target gripper position inside object dimensions.
Moving **(moving to grasping location, drawing, and throwing)**	**Creating trajectory**	Piecewise interpolation between target poses inside the table.	Optimization based inverse kinematics solver.	Kinematic trajectory optimization solver.
	Following Trajectory	Feedforward differential inverse kinematics controller.	Spatial force controller with a desired z-direction force.	Stiffness controller following a trajectory inside the table.

Fig. 2. Analysis of different simulation implementation difficulties. Realized implementation (minimum viable product) highlighted in green.

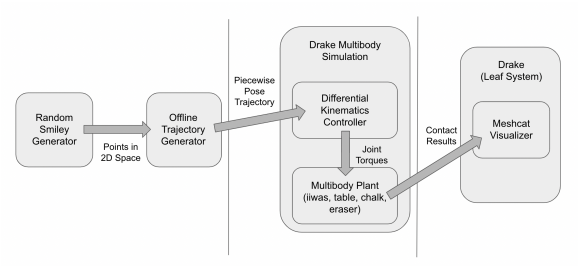


Fig. 3. System pipeline overview

with its own controller and access to input and output ports, we could not build off of many built examples and libraries. Thus, we built the system from the ground up creating one arm at a time. However, we ran into many problems when trying to implement multiple arms and systems within one plant. Throughout the course, most Kuka iiwa arm models were added to the plant via a parser method: *AddModelFromFile*. When we attempted this, we encountered a lack of separation in the input and output ports and naming issues between the multiple manipulators. The two systems had the ports combined into one vector making it difficult to call on the correct port. We also had difficulty specifying which arm to weld objects to.

To have a unified and structured approach to adding the two manipulators to our station, we implemented a custom variation of the *MakeManipulationStation* function. We used a model directive to specify two iiwas with names and initial conditions, along with welding two wsg grippers to the arms. From this model directive method, we learned a simpler and more streamlined method for instantiating a robot into a *Drake simulation*. The choice of our own modified implementation enabled us to integrate the robotic arms with the *MeschatWriting* system and writing tools.

We created the chalkboard table model using shapes in *Drake*. We created the "chalk" using a *Capsule* object (radius = 0.01, length = 0.2). The eraser and table used the *Box* object (eraser: width = 0.05, height = 0.025, depth = 0.125 and table: width = 0.8, height = 1, depth = .2). To eliminate potential problems from the grasping and holding of the chalk and eraser in the first iteration of this system, we welded the chalk to the end-effector of the first robot and the eraser to the end-effector of the second robot. This allowed us to prototype the trajectory planning and test that the *MeshcatWriter* system (described in a later section) and station were connected properly and that the contact geometry of the tools with the table was working properly.

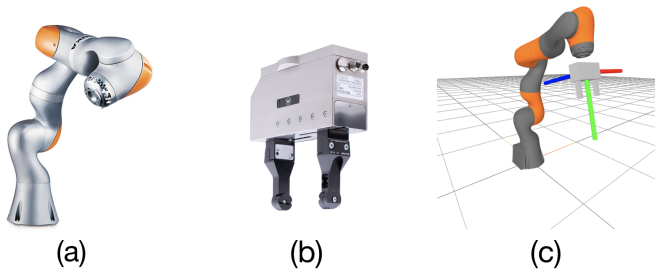


Fig. 4. Robotic arm and gripper. (a) KUKA LBR-IIWA 7-DoF Arm [1] (b) Shunk WSG-50 Gripper [2] (c) Welded WSG-50 model to IIWA arm in Drake [3]

C. Trajectory Planning

To create the trajectories for our robots, we parameterized a smiley face in 2D. Our "smiley generator" used circles and arcs to define key points on smilies with uniform randomly drawn eye size, distance between the eyes, smile width, smile type (open smile comprised of two arcs and closed smile comprised of one arc), and smile size (arc radii). We then mapped this 2D drawing to points in 3D space on the tabletop. For each robot, we computed the desired transformation between the table top and gripper based on the tool being used (orientation and offset of the gripper based on grasp and dimensions of the chalk and eraser). We then transformed these frames from the world frame to the iiwa frames and interpolated between frames to create a time-based trajectory. We sent trajectory format to our simulation because we used a differential inverse kinematics controller to actuate the iiwas and because differential inverse kinematics controllers take the distance and time span between points, calculate velocity for that time period in operational space, use the jacobian to find velocity commands in joint space, and integrate the velocities to give joint position commands, the input trajectory must be in robot frame. In order to prevent the robot from drawing between the smiley features (ex. between one eye or another) we added pre-draw poses (above the next desired point) between different parts feature trajectories to get accurate drawing contact.

In implementing and troubleshooting this system, we learned how to use many Drake tools. By displaying both *meshcat triads* and *multibody triads* we were able to visualize our generated the keypoints (displayed as *meshcat triads*) and check that they aligned with our desired gripper orientations (displayed as *multibody triads* that move with the robot through the trajectory). The understanding of *Drake*'s system's framework we developed in the extensive time spent setting up our simulation was critical in pulling the correct frame from the correct model from the correct plant to draw the triad in the correct *scenegraph* from our overall diagram.

On main drawback of using interpolated key points and a differential inverse kinematics controller is that we are not able to easily incorporate the physical limitations of the system into our simulation. For example, our system does not conform to joint limits constraints and does not solve for solutions that avoid contact with the table. To solve this problem in our current set-up, we limited the smilies generated to ones that do not cause the iiwas to violate these limits.

In the next iteration of this project, we would implement an optimization-based inverse kinematics solver that creates a trajectory that minimizes the distance to our list of desired key points. For the drawing robot, the list of key points would correspond to the position of the end of the chalk. We would add constraints that fix the x, y, and z positions of the tip while allowing for limited rotation about the x and y axes (all points and frames referenced in this section of the paper are described in world frame for convenience but would be implemented in gripper frame) such that the gripper will not hit the table:

$$\min_{q_1,\cdots,q_N} \sum_{n=1}^{N} ||q_n - q_0||_2^2$$

$$\begin{aligned} \text{s.t.} \quad & f_P(q_n) = P \\ & f_R(q_n)_x \leq \delta \\ & f_R(q_n)_y \leq \delta \end{aligned}$$

Where q_0 is a nominal comfortable drawing position that we will determine.

For the erasing robot, the list of key points would correspond to the position of the center point of the bottom of the eraser. We would add constraints that fix the z position of the center point and the rotation about the x and y axes since rotation about the x or y axes would cause the eraser to get stuck on the table (assuming there is no feedback controller in the trajectory following implementation). We would also add constraints that allow for some limited error in the x and y positions because the eraser is larger than the drawn lines.

$$\min_{q_1,\cdots,q_N} \sum_{n=1}^{N} ||q_n - q_0||_2^2$$

$$\begin{aligned} \text{s.t.} \quad & f_P(q_n)_x \leq \delta \\ & f_P(q_n)_y \leq \delta \\ & f_P(q_n)_z = P_z \\ & f_R(q_n)_x = R_x \\ & f_R(q_n)_y = R_y \end{aligned}$$

Where q_0 is a nominal comfortable drawing position that we will determine.

D. Trajectory Following

We first implemented a differential kinematics controller since it does not require feedback from the system and we did not need to access information from the plant context to implement the controller. However, since the controller cannot react based on the state of the system and the iiwa controllers have built-in PD controllers that will command higher torque to the joints when they do not achieve their desired input state, our system is inflexible to contact and will apply extra torque to try to reach the desired position. Thus, we first tested our differential kinematics controller with the friction coefficients of our chalk and eraser bodies set to 0. When reintegrating friction, we had to carefully tune how far into the table we created our desired trajectory to minimize the reactionary frictional forces that could lead to poor trajectory tracking while still achieving the contact required as by our *MeshcatWriter* system.

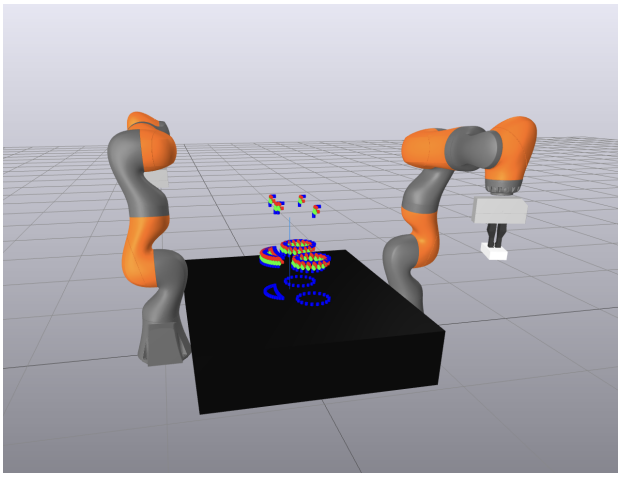


Fig. 5. Trajectories of key points, including pre-drawing and drawing poses, for both robots (red, green, and blue triad clusters above the table) and desired drawing outcome (blue points on the table).

$$F_z = k_p \cdot e + k_d \cdot \frac{\mathrm{d}e}{\mathrm{d}t}$$
$$\text{where } e = F_z - F_{z_{des}}$$

Where F_des would be a nominal value determined by experimentation.

With this controller, we would convert the desired cartesian space forces to joint space forces that we would input into our diagram to actuate our iiwa. The relationship between cartesian space forces and joint space torques is described by:

$$\tau = J^T \cdot F$$

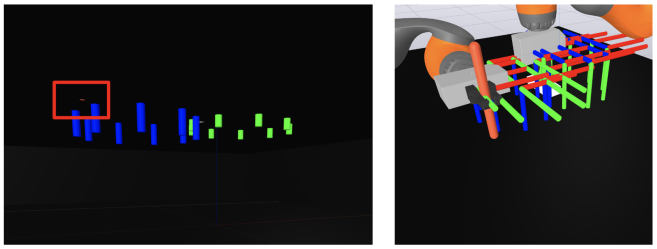
With this implementation, we would still have to tune the PD constants so our system can effectively follow the trajectory, but it would make our system more robust to trajectories that do not maintain perfect z-direction tracking.

E. Writing

In order to visualize the writing, we needed a system that could track the contact geometries between objects and make markings based on those contacts between objects. We utilized the *MeshcatWriter*, a *LeafSystem*, to achieve this. For a specific instance of *MeschcatWriter* system, we can define two objects, "a writing tool", the chalk, and "a drawing board", the table. When the chalk and table are in contact, a cylindrical shape is created on the drawing board at the point of max penetration with a specified color and line width.

For our setup, we used two *MeshcatWriter* systems, one to simulate drawing and one to simulate erasing. For drawing, we draw an orange line based on the contact made between the chalk, and the table and for erasing we take the contact made between the eraser and the table and we draw with a stroke in the same color as the table to simulate erasing. In the next iteration of this project, we would implement a spatial force controller with open-loop x and y-direction feedforward and a closed-loop z-direction force PD controller. Thus, instead of an inverse differential kinematics integrators, we would have torque controllers in our *Drake* system. Our z-directional force controller would be:

1) Contact Issues: During the integration of our robotic arm controllers and writing systems, we tested the writing systems' capability to write. Our visualization showed the chalk to be in contact with and penetrating the table, as shown in Fig 6.



(a) (b)

Fig. 6. Figure contact issues. (a) Chalk in penetration of the board, with contact forces visualized in the red rectangle. (b) Chalk in visual contact with the board.

To investigate this further, we needed a system to visualize or report the contacts, if any, being made. We incorporated another *LeafSystem*, *PrintContactResults* to our diagram to visualize the contacts. To have a baseline comparison, we first implemented this system in the meshcat Writing Demo, to visualize the contact geometries, shown in Fig 7a. We added this contact visualizer system to our station and confirmed that our system was missing the correct contacts, shown in Fig 7b. With Russ's help, we discovered that the contact geometries errors were due to an error in *MeshcatWriter* system. Nevertheless, this exercise was helpful in understanding and displaying contact forces.

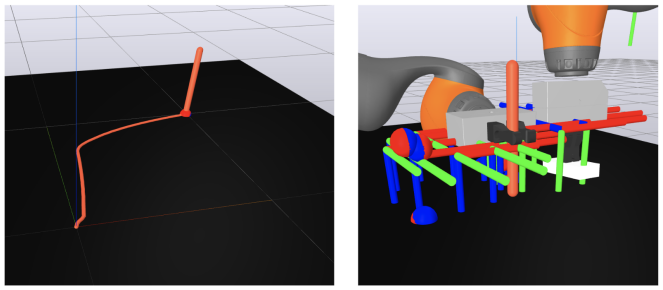
In the next iteration of this project, we would create a more accurate simulation by continuously and incrementally deleting the shapes created by the *MeshcatWriter* during the simulation instead of drawing over them with black shapes. This would allow us to draw and erases continuously on the same table without having to clear or restart the simulation.

IV. Results

The system described in Level 1 of our three-stage implementation in Fig 2. was implemented. We show in our system one robot can draw a randomly generated smiley and the other can erase as seen in Fig 8. This link provides a live demo of what we did: link

V. Discussion

In our simulations, we encountered two main issues. First, for certain smiley faces, the robot does not accurately follow the trajectory. We believe this is due to the fact that the



(a) (b)

Fig. 7. Figure contact issues. (a) Chalk in penetration of the board, shown in the red box with orange tip sticking through. (b) Chalk in visual contact with the board.

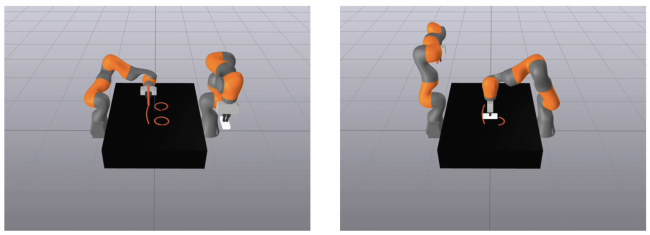


Fig. 8. Robots successfully drawing and erasing randomly generated smiley.

trajectory causes the robot configuration to be near singularities. Second, there is a lag in the simulation visualization of the manipulator movement and erasing being displayed. The orange lines disappear before the actual eraser visually makes contact with the point. Aside from more robust and complete implementations, our system could also benefit from a more streamlined simulation pipeline. We currently run our entire *MeshcatVisualizer* to pull the initial robot poses for our trajectory. Since we have predefined joint positions we may be able to use those. We can also utilize the expected code push that will facilitate this improvement.

Contributions

Jess and Raymond worked collaboratively on the project and often helped each other brainstorm/troubleshoot. For implementation, Jess took the lead on the trajectory generation and trajectory following while Raymond took the lead on the simulation environment development and set-up and meshcat writing.

Code Avaliability

Our code can be found at this link.

Acknowledgment

We would like to thank Professor Russ Tedrake, Boyuan Chen, and Anthony Simeonov for their expertise in the devolepment of this project and for the entire 6.4212 staff for an amazing semester.

References

[1] Kuka. LBR iiwa https://robots.ieee.org/robots/lbriiwa/
[2] Weiss Robotics https://www.tm-robot.com/en/product/weiss-robotics-wsg-50-110/
[3] R. Tedrake, "Robotic Manipulation: Perception, Planning, and Control," Robotic manipulation, 2022. http://manipulation.mit.edu.