High Five: Fabrication and Digital Simulation of Robotic Arms

Kevin Sisk, Intern, Johns Hopkins University Applied Physics Laboratory
Intern Mentor G/T Program, Reservoir High School
Kpsisk05@gmail.com

Introduction:

Robots will change the world. Understanding robotic design and development allows engineers to create machines that impact global communities. Robotic development requires rigorous phases of planning, testing, and design to produce a viable system. Robots can produce solutions to many everyday problems as well as industrial and technological challenges. However, the production of such robots requires iterative testing to achieve reliability and reduce bugs. While testing design variations and programming changes with a functional prototype is a common choice, many robotic systems “are very hard or impractical to actually test kinetically” (T. Dignan, personal communication, December 13, 2023). Although creating a simulation may complicate the robotic project, utilizing such software can save time and money throughout development. Similarly, implementing the robotic design and programming into the final, physical model is much simpler when backed by a functional simulation. The purpose of this paper is twofold: to describe the process of creating a robot at a basic level and to explore the skills necessary to develop an accurate 3D simulation.

Robotic Design Process:

Robotic development involves a multitude of steps to achieve the desired outcome based on initial goals. A problem must first be identified for which the robot serves a purpose. When the need for a robot is present, engineers plan to develop the robot. At the beginning of the process, they focus on the function of the robot that is needed to complete the given task and a design that fits the specifications. Later, they design and build a prototype that works to complete the task or solve the problem. Finally, Engineers iteratively test the robot and adjust components and code to optimize the machine. This process can take weeks, months, or years. The lengthy nature of robotic testing similarly requires lots of money to fund a project. Therefore, the traditional methods of robotic development that rely on physical models to test changes are extremely expensive in both time and money.

Robotic Simulations:

In contrast to the tedious process of physical testing, robotic simulations can be used to cut down on development time and expensive testing. Robotic simulations use physics engines that allow engineers to test and implement features and code without the physical, iterative testing that was once necessary (Andrews, 2023). The physics engines can be customized to apply different amounts of forces such as gravity and friction to match the constraints of the project.

Simulations provide benefits in testing and development. Once the physics of a scene are customized to match the intended environment, the simulation affects the robotic model in the same way that it interacts with its physical environment. Then, a robotic model can be loaded and programmed within the simulated environment. The process of testing and implementing code within the simulation is similar to that of real-world testing but changes are implemented much faster due to the nature of the integrated environment.

Changes to a digital model are easier to test and implement (T. Dignan, personal communication, December 13, 2023). As the robot exists within the environment, it does not need to be reprogrammed each time the code is changed. Consequently, engineers do not have to wait for new code to upload to the robot. Simulations reduce cost as changes to the robot are made digitally, so there is no additional cost for new components.

The proper simulator must be chosen when beginning a new project. CoppeliaSim is a great fit for robotic simulation. CoppeliaSim provides multiple physics engines, and powerful, fluid graphics (Melo et al., 2019). Also, CoppeliaSim contains a multitude of preloaded models. Both mobile and non-mobile robots are present in the software. An assortment of robotic arms are pre-built for rapid testing and integration of common robotic designs. CoppeliaSim is programmed natively with the language Lua but can be programmed in Python as well. Similarly, other languages are available through the included remote application programming interfaces (API).

Python is commonly used for data science and allows for easy manipulation of matrices to control the rotations of joints using the NumPy library (3D rotations, 2020). The Python API in CoppeliaSim allows for faster programming in a familiar language.

Using Arduino:

While the use of robotic simulations can reduce the time and money spent on a project, a physical robot will still ultimately be constructed. The first choice for many hobbyists and some professionals is Arduino: an Italian company that develops open-source microcontroller boards and the Arduino Integrated Development Environment (IDE). Robotics using an Arduino microcontroller allows for simplistic programming and implementation of programming and design. The open-source nature of their products allows other companies to replicate and reproduce their products - which is encouraged because the community of programmers and engineers that use Arduino is expansive. Thousands of example projects have been created to inspire new makers.

Arduino boards come in many shapes and sizes such as the Nano, Mega, and most popular: Arduino Uno. The boards consist of multiple components, but the most important is the microcontroller. The programmable chip works with integrated general-purpose input/output (GPIO) pins on the board to control a variety of sensors and motors. The Arduino IDE uses C++ programming to write scripts and includes thousands of code libraries that offer pre-made functions. Using libraries can expedite the development process. For example, the servo.h library allows for quick and seamless integration of servo motors with an Arduino microcontroller. While similar function and control is possible without the use of a library, developing it from scratch is difficult and time-consuming.

Methods:

Through meetings with my mentor Tom Dignan who specializes in the reverse engineering of autonomous systems, a set of goals was developed to guide my project. A robot was designed to fit the specifications of the project overview. The robot named

Simulator Controlled Robotic Arm Prototype (SCRAP) has three degrees of freedom and a robotic gripper as the tool head. The function of the robot requires precise manipulation of the gripper. SCRAP was presented with an arbitrarily placed box on a grid. The coordinates of the box were known and the robot used the gripper to relocate the boxes to the desired position.

Motion Control:

The location of the gripper is the most important variable. Robotic motion should be planned with emphasis on the orientation and motion of the tool head (English 2019). The gripper functions as the tool head on the EezyBot. Although the position of the tool head is the most important variable to consider, the blocks on the grid must also be taken into account when planning the motion of the robot. If the boxes are neglected, the resultant motion path may collide with them.

SCRAP accepts a target position and the simulated Python script calculates the coordinates needed for each joint to achieve the required position.

Building the Arm:

SCRAP is made of four main components. The electronic components consist of an Arduino Nano microcontroller and four MG90S Servo Motors with metal gearing. The microcontroller is a clone that provides USBC interfacing for power and programming. The metal gears of the servo motor provide more reliability and smoother motion than a motor with plastic gears that are found in most cheap servos.

The frame of the robot is 3D printed using PLA plastic. An open-source model titled “The EezyBot Arm” is available on Thingiverse (a website that hosts 3D models usually under Creative Commons Licensing). The EezyBotArm was designed by user theGHIZmo. The 3D printed parts are assembled using M3 Nuts and Bolts, and 4 mm brass tubing.

Using the model saved time in physical prototyping and allowed for quicker implementation of code as no time was spent on the design of the physical robot. The EezyBotArm circumvented the need for expensive machines or materials in the fabrication stage of SCRAP. The model met all of the criteria for the project while remaining cost-effective.

User Control:

Connecting the digital simulation and the Arduino microcontroller allowed the robotic arm to be controlled with the simulation acting as a graphical user interface (GUI). By replicating the physical robot in the simulated environment, the physical robot is controlled by altering the position of each motor when the corresponding motor in the simulation is changed.

Manual control of SCRAP uses sliders in the simulation to control each motor individually. Each slider can be set from 0° to 180 °. When the value of the slider changes both the simulated joint and the physical joint are rotated to match. The simulation communicates with the Arduino through serial communication which allows the Python program in the simulation to send and receive data with the script that is running on the Arduino. The two programs communicate through the USB cable that tethers the Arduino to the computer.

Integrated Control:

Integrated control allows the simulation of the robot to take two coordinates and move between them free of direct human input. The simulation relies on a database of recorded coordinate positions to move the robotic arm between locations quickly and efficiently. Integrated control circumvents the need for manual input and setting the position of each Servo motor individually.

The integrated control system has three parts. First, the input method uses a text box in CoppeliaSim; the initial and desired coordinates are given. The simulation accepts the coordinates and stores them in separate variables to keep track of where the object begins, and where it must be placed.

Second, the simulation calls on a database to find the necessary motor angles to achieve the desired positions. The database consists of recorded positions through the use of the user control system with the simulated GUI. The physical robot was moved to a multitude of positions around itself on a piece of paper and the angles of each motor were marked. The resulting data created a semicircular polar graph that consists of the points at which the gripper can reach at Z-axis height of zero off of the table. The positions of each motor at each of these points were stored in a spreadsheet and then translated to a nested list in Python. When SCRAP is given a pair of graphed coordinates it finds the points in the nested list and retrieves the motor positions needed to move to that location.

Finally, the integrated control system moves the motors. It is programmed to move in steps that will not knock over the object it is trying to pick up. It begins at the 90° position and rotates the base to the desired angle. SCRAP then lowers the gripper behind the object in the open position. The robot moves the gripper forward and closes it on the object. It returns to the 90° position and rotates to the final angle. Finally, the robot lowers the object to its destination, opens the gripper, moves back, and returns to the 90° position.

Results:

SCRAP can successfully be controlled in two ways. Both are fully functional.

When using the simulation as a GUI the joints of the robot act as expected and move according to the changing angles of the simulation. Using SCRAP as a pick and place machine in this manner consists of three steps. First, the arm must be positioned behind the target object and move forward around the target before closing the gripper. Then, the robot moves to its 90° position and rotates to the angle of the drop location. Finally, the robot must be lowered to place the object in the desired spot before opening the gripper to release it.

The second method uses nested lists to position SCRAP at known locations. With sufficient data recorded in the lists, the robot is able to move to hundreds of individual positions. In this way, the starting position of the object, and the desired final position can be given for SCRAP to move to.

Discussion:

The goal to create a functional pick-and-place robotic arm with the aid of a digital simulation was successfully met. However, much can be improved, and different steps could be taken in the future.

Moving the robot manually using the simulation proved to be insufficient. While the motion was functional, the process was jittery and time-consuming. Positioning the motors accurately took repeated attempts as the needed positions were unknown. Therefore, manual manipulation of the robot is insufficient as an accurate pick-and-place machine.

Alternately, using the lists of coordinates proved to be effective. The Python script that handled the lists worked fast and accurately to locate the desired position. Developing a professional robot with this system of control would be a viable alternative to inverse kinematics.

The hypothesis that the use of rotation matrices, joints, and inverse kinematics will allow an accurate simulation to be created which will rapidly reduce the time needed to produce a functional physical model was unsupported by the final robot. While rotation matrices and inverse kinematics are important parts of developing some robots, they were not needed for SCRAP.

Inverse kinematics were replaced with the lists of coordinates (Recorded Coordinate System). Using nested lists to recall known coordinates allowed the robot to position the motors accordingly for a wide range of known locations that it would need. This function replaced the need for inverse kinematics as the tool for positioning the tool head to a desired location. Inverse kinematics was not used due to the time constraints of implementation. Setting up the robot on a piece of paper and positioning the motors to different coordinates was more time-effective.

Rotation matrices were also not used. They were unneeded due to the Recorded Coordinate System as each position already had set motor values. when the robot retrieved its coordinates it did not need rotation matrices because the line of coordinates held individual servo positions instead.

Hobbyist robotics is a growing field of interest and this project is a small part of a larger movement to make robotics more accessible. While the design and function are not perfect I hope that others interested in robotics will learn from my work and be inspired to improve on what I have done.

Future Work:

Comparing the abilities of inverse kinematics to the Recorded Coordinate System would be beneficial in observing the benefits and shortcomings of both systems as well as which is better suited for the EezyBotArm.

Also, exchanging the servo motors for higher torque stepper motors could increase the accuracy of the robot and reduce the jerkiness of the motion.

The primary shortcomings of SCRAP are the minimal size and range of the model and the strength of 3D printed parts. Improving the size and materials of the robot paired with stronger motors would produce a much more capable product.

Conclusion:

The development of robots is a tedious process that may yield rewarding results. The design process requires many iterations of designing, testing, and altering robots to achieve the desired function. The use of digital robotic simulators provides toolsets that can reduce the time and money spent on robotic projects. In order to develop a working robotic arm controlled by a simulation, communication between the CoppeliaSim scene and an Arduino acts as a controller while the physical robot consists of 3D printed parts. The ability for a functional robot to be created and developed in a hobbyist setting provides the opportunity for new engineers to enter the field of robotics. Increasing the amount of people with knowledge, understanding, and skills to create the machines of the future will lead to an increased rate of innovation.

References

Andrews, G. (2023, June). What is robotics simulation? Nvidia. https://blogs.nvidia.com/blog/2023/06/29/what-is-robotics-simulation/

Dietrich, S. (2022, December 29). Robot motion command types: Understanding linear joint, and arc movement. Control Automation. English, J. (2019, July 1). Robotics motion control: The complex relationship between movement and task. Medical Design Briefs.https://www.medicaldesignbriefs.com/component/content/article/mdb/pub/briefs/34808#:~:text=Robot%20motion%20control%20enables%20articulated,is%20appropriate%20on%20the%20robot.

English, J. (2019, July 1). Robotics motion control: The complex relationship between movement and task. Medical Design Briefs.https://www.medicaldesignbriefs.com/component/ content/article/mdb/pub/briefs/34808#:~:text=Robot%20motion%20control%20enables%20articulated,is%20appropriate%20on%20the%20robot.

Introduction to stepper motors. (2023). Omega Engineering. https://www.omega.co.uk/prodinfo/stepper_motors.html

Mathews, K. (2018, September 24). How robot precision has evolved, enabling more uses. *Robotics Business Review*.https://www.therobotreport.com/how-robot-precision-has- evolved-enabling-more-uses/

Melo, M. S. P., Neto, J. G. D. S., Silva, P. J. L., Teixeira, J. M. X. N., & Teichrieb, V. (2019, October). Analysis and comparison of robotics 3D simulators. *Universidade Federal De Pernambuco*. https://www.researchgate.net/profile/Joao-Marcelo- Teixeira /publication/337789376_Analysis_and_Comparison_of_Robotics_3D_Simulators/links/61ac1a50ca2d401f27c6b83a/Analysis-and-Comparison-of-Robotics-3D-Simulators.pdf

3D rotations and Euler angles in python. (2020, September 7). Meccanismo Complesso. https://www.meccanismocomplesso.org/en/3d-rotations-and-euler-angles-in-python/