Robotic Pick and Place using 7-DOF Franka Panda

Dheeraj Bhurewar

March 21, 2025

Abstract

This document provides a comprehensive overview of the implementation of a pick and place system using a Franka Panda robotic arm in the MuJoCo physics simulation environment. I have detailed the modular architecture, the inverse kinematics methods employed, the contact detection algorithms, and the trajectory planning approaches. The document also discusses challenges encountered during development and potential future improvements to the system.

Contents

1 Introduction

Robotic manipulation, specifically pick and place operations, represents a fundamental task in modern robotics with applications spanning manufacturing, logistics, healthcare, and beyond. This document details the implementation of a pick and place system using a 7-degree-of-freedom (DOF) Franka Emika Panda robotic arm simulated in MuJoCo, a physics-based simulation environment.

Our implementation focuses on several key aspects:

- Modular code architecture for maintainability and extensibility
- Robust inverse kinematics methods for precise end-effector positioning
- Contact detection for reliable object grasping
- Interpolation-based trajectory planning for smooth motion
- Interactive simulation with step-by-step execution

The remainder of this document describes the system architecture, the mathematical foundations of the algorithms used, implementation details, encountered challenges, and opportunities for future enhancements.

2 System Architecture

2.1 Directory Structure

The system is organized into a modular directory structure to promote code organization and reusability:

```
panda_pick_place/
          README.md
          main.py
          config.py
          src/
                __init__.py
                utils/
                      __init__.py
                      math_utils.py
                      vieIr.py
                      input_utils.py
                kinematics/
                      __init__.py
                      forward_kinematics.py
                      inverse_kinematics.py
                robot/
                      __init__.py
                      robot_control.py
                      trajectory.py
                tasks/
                    __init__.py
                    pick_place.py
          assets/
```

```
                panda/
                    franka_panda_w_objs.xml
```

2.2 Module Responsibilities

2.2.1 Utilities (src/utils/)

Contains helper functions for mathematical operations, simulation visualization, and user interaction:

- **math_utils.py**: Quaternion operations, rotation conversions, and path interpolation
- **vieIr.py**: Abstraction layer for different MuJoCo visualization backends
- **input_utils.py**: User interaction functions for the interactive simulation

2.2.2 Kinematics (src/kinematics/)

Handles forward and inverse kinematics calculations:

- **forward_kinematics.py**: Computing end-effector position and orientation from joint angles
- **inverse_kinematics.py**: Computing joint angles to achieve desired end-effector poses

2.2.3 Robot Control (src/robot/)

Manages direct robot control and trajectory execution:

- **robot_control.py**: Joint control, gripper control, and physics stabilization
- **trajectory.py**: Path interpolation and trajectory execution

2.2.4 Tasks (src/tasks/)

Implements high-level robotic tasks:

- **pick_place.py**: The complete pick and place operation workflow

2.2.5 Configuration (config.py)

Centralizes all configurable parameters:

- Joint names and limits
- End-effector parameters
- Pickup and place positions
- IK parameters
- Path planning parameters

2.2.6 Main Script (main.py)

Entry point for the application that initializes the simulation and starts the pick and place task.

3 Kinematics Implementation

3.1 Forward Kinematics

Forward kinematics computes the end-effector's position and orientation given a set of joint angles. My implementation leverages MuJoCo's built-in kinematic calculations:

Algorithm 1 Forward Kinematics

```
procedure ForwardKinematics(model, data, joint_angles, joint_names)
    Store original joint positions
    for each joint in joint_names do
        Set joint angle in data.qpos
    end for
    Call mujoco.mj_forward(model, data)
    Get end-effector position from data.xpos
    Get end-effector orientation matrix from data.xmat
    Convert orientation matrix to quaternion
    Restore original joint positions
    return end-effector position and quaternion
end procedure
```

3.2 Inverse Kinematics Methods

3.2.1 Direct Line Approach

One of my key contributions is the "direct line approach" for inverse kinematics. Instead of relying solely on numerical optimization from a single starting configuration, I generate multiple candidate starting points along a direct line from the current end-effector position to the target position.

The algorithm works as follows:

3.2.2 Optimization-Based IK

For the main IK solution, I use an optimization-based approach:

The objective function balances position and orientation errors:

$$f(\theta) = w_{pos} \cdot ||p(\theta) - p_{target}||^2 + w_{ori} \cdot (1 - |q(\theta) \cdot q_{target}|) \tag{1}$$

where:

Algorithm 2 Direct Line Approach for IK

procedure GENERATEDIRECTLINEGUESSES(model, data, current_pos, target_pos, num_samples)
    Initialize joint_guesses with current joint angles
    for $i = 1$ to num_samples **do**
        Interpolate position: pos $= (1-t) \cdot$ current_pos $+ t \cdot$ target_pos, where $t = i/(\text{num\_samples}-1)$
        Use simple Jacobian IK to find joint angles for this position
        Add resulting configuration to joint_guesses
    end for
    return joint_guesses
end procedure

Algorithm 3 Optimization-Based IK

procedure INVERSEKINEMATICSWITHORIENTATION(model, data, target_pos, target_quat, ...)
    Get joint limits
    Generate direct line guesses
    for each guess in direct_line_guesses **do**
        Minimize objective function using PoIll's method
        Update best solution if better than previous
    end for
    Further refine best solution with longer optimization
    return joint angles
end procedure

- θ represents the joint angles
- $p(\theta)$ is the end-effector position resulting from those angles
- $q(\theta)$ is the end-effector orientation quaternion
- w_{pos} and w_{ori} are Iighting factors

4 Comparison of Inverse Kinematics Methods

4.1 Mathematical Analysis of IK Approaches

Inverse kinematics presents a challenging optimization problem in robotics with multiple potential solution approaches. This section provides a mathematical comparison of several methods and explains the rationale behind our selection of PoIll's method.

4.1.1 Analytical Methods

For robots with closed-form solutions (typically 6-DOF or less), analytical methods can be employed:

$$\theta = f^{-1}(x) \tag{2}$$

where f^{-1} is the inverse of the forward kinematics function that maps joint angles θ to end-effector pose x. For our 7-DOF Franka Panda robot, analytical solutions are challenging due to redundancy, as the system is underconstrained (infinite solutions may exist for a given end-effector pose).

4.1.2 Jacobian-Based Methods

The Jacobian matrix $J(\theta)$ relates joint velocities to end-effector velocities:

$$\dot{x} = J(\theta)\dot{\theta} \tag{3}$$

Jacobian-based IK approaches include:

Jacobian Pseudoinverse:

$$\dot{\theta} = J^{+}(\theta)\dot{x} \quad \text{where} \quad J^{+} = J^{T}(JJ^{T})^{-1} \tag{4}$$

Damped Least Squares (DLS):

$$\dot{\theta} = J^{T}(JJ^{T} + \lambda I)^{-1}\dot{x} \tag{5}$$

where λ is a damping factor that improves numerical stability near singularities. Jacobian-based methods are iterative and can be computationally efficient, but they may struggle with:

- Local minima in complex configuration spaces
- Singularities where the Jacobian loses rank
- Joint limit handling (requires additional constraints)

4.1.3 Optimization-Based Methods

These methods formulate IK as an optimization problem:

$$\min_{\theta} f(\theta) \quad \text{subject to} \quad \theta_{min} \leq \theta \leq \theta_{max} \tag{6}$$

where $f(\theta)$ is an objective function measuring the error betIen current and desired end-effector pose. Common optimization approaches include:

Gradient Descent:

$$\theta_{k+1} = \theta_k - \alpha \nabla f(\theta_k) \tag{7}$$

where α is the learning rate and $\nabla f(\theta_k)$ is the gradient of the objective function.

Newton's Method:

$$\theta_{k+1} = \theta_k - [H_f(\theta_k)]^{-1} \nabla f(\theta_k) \tag{8}$$

where $H_f(\theta_k)$ is the Hessian matrix of second derivatives.

BFGS: An approximation of Newton's method that estimates the Hessian matrix iteratively without explicitly computing second derivatives.

PoIll's Method: A derivative-free optimization algorithm that performs sequential line searches along a set of directions, periodically updating these directions.

4.2 Rationale for Choosing PoIll's Method

I selected PoIll's method for our IK solver for several compelling reasons:

4.2.1 Derivative-Free Optimization

Unlike gradient-based methods, PoIll's method doesn't require computation of derivatives, which offers several advantages:

- More robust to noise and discontinuities in the objective function
- No need to compute the Jacobian matrix, which can be complex for 7-DOF robots
- Avoids issues with inaccurate gradient calculations near singularities

4.2.2 Effective Handling of Local Minima

PoIll's method updates its search directions based on previous successes, allowing it to adapt to the local geometry of the objective function. This provides:

- Better escape from local minima when combined with our direct line approach
- Improved convergence on complex, non-convex configuration spaces

4.2.3 Constraint Handling

PoIll's method, especially as implemented in SciPy, effectively handles the bound constraints representing joint limits:

$$\theta_{min} \leq \theta \leq \theta_{max} \tag{9}$$

This ensures that all proposed solutions remain within the robot's physical capabilities.

4.2.4 Computational Efficiency

For a 7-DOF robot like the Franka Panda, our experiments shoId that PoIll's method provided a good balance betIen:

- Convergence success rate
- Solution quality (natural-looking configurations)
- Computational performance

4.2.5 Integration with Direct Line Approach

The combination of our direct line approach for generating initial guesses with PoIll's optimizer creates a particularly effective system:

- Initial guesses provide good starting points in different regions of the configuration space
- PoIll's method efficiently refines these guesses to high-precision solutions
- The overall system is more robust to different target poses and initial robot configurations

This synergy betIen multiple initial guesses and an efficient derivative-free optimizer addresses many of the traditional challenges in IK for redundant manipulators.

5 Gripper Control and Contact Detection

5.1 Basic Gripper Control

The Franka Panda gripper consists of two fingers with parallel jaw motion. Our implementation supports two modes of control:

1. Boolean open/close using predefined positions
2. Precise width control with specific gap distance

5.2 Advanced Contact-Based Gripper Control

I implemented a more sophisticated gripper control method that gradually closes the gripper until contact with the target object is detected. This approach mimics how a human would grasp an unknown object, applying just enough pressure for a secure grip.

Algorithm 4 Gripper Control with Contact Detection

```
1: procedure ControlGripperUntilContact(model, data, target_width, ...)
2:     Start with gripper fully open
3:     for step = 1 to max_steps do
4:         Reduce gripper width by small increment
5:         Apply joint positions to gripper fingers
6:         Step physics simulation
7:         for each contact in simulation do
8:             Check if contact involves finger and target object
9:             If valid contact detected, stop closing
10:        end for
11:        if reached target width then
12:            Stop closing
13:        end if
14:    end for
15:    Run additional stabilization steps
16:    return final width and contact status
17: end procedure
```

6 Pick and Place Task Implementation

The complete pick and place operation follows a typical workflow:

1. Move to a hover position above the pickup location
2. Open gripper
3. Move down to the pickup position
4. Close gripper until contact with object
5. Move back up to hover position
6. Move to hover position above place location
7. Move down to place position
8. Open gripper to release object

9. Move back up to final hover position

Each movement phase includes:

- Computing target joint angles using IK
- Interpolating a smooth path betIen current and target configurations
- Executing the path with appropriate visualization
- Running physics stabilization steps to allow for proper physical interactions

7 Challenges and Solutions

7.1 IK Convergence Issues

One significant challenge was achieving reliable IK convergence to desired poses.

Problem: Standard optimization-based IK would often converge to suboptimal local minima, resulting in awkward robot configurations or failure to reach the target.

Solution: I implemented the direct line approach, which generates multiple initial guesses along a direct path from current to target position. This dramatically improved convergence success rates and solution quality.

7.2 Physics Stabilization

Problem: Objects would appear to "float" or behave unrealistically during simulation.

Solution: I added explicit physics stabilization steps after each major movement and at initialization to ensure proper physical settling:

```
def stabilize_scene(model, data, steps=100, vieIr=None):
    """Run physics steps to stabilize the scene."""
    print(f"Stabilizing scene with {steps} physics steps...")
    for i in range(steps):
        mujoco.mj_step(model, data)
        if vieIr is not None and i % 10 == 0:
            update_vieIr(vieIr)
    mujoco.mj_forward(model, data)
    print("Scene stabilized.")
```

7.3 Contact Detection

Problem: Reliable detection of contacts betIen gripper and objects proved challenging.

Solution: I implemented a multi-stage contact detection system that:

- Focuses on collision geoms rather than visual geoms

- Checks contacts with specific target objects
- Gradually closes the gripper until contact is detected

This approach provides more reliable grasping behavior compared to simple position-based gripper control.

8 Future Improvements

8.1 Learning-Based Approaches

8.1.1 Imitation Learning

Demonstration-based learning could improve the system by:

- Learning natural human-like motions from demonstrations
- Capturing implicit strategies for handling different object types
- Reducing the need for explicit programming of movement phases

8.1.2 Reinforcement Learning

RL could optimize the pick and place process by:

- Learning efficient policies through trial and error
- Adapting to different objects and environments
- Optimizing for metrics like time, energy, or success rate

8.2 Advanced Trajectory Planning

8.2.1 Bézier Curves

Replacing linear interpolation with Bézier curves would provide:

- Smoother acceleration and deceleration profiles
- Better obstacle avoidance capabilities
- More natural-looking motions

The parametric form of a cubic Bézier curve is:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3 \tag{10}$$

8.2.2 Trajectory Optimization

Future work could incorporate trajectory optimization that:

- Minimizes jerk (derivative of acceleration)
- Avoids joint limits and singularities
- Considers dynamic constraints of the robot

8.3 Teleoperation Integration

Adding teleoperation capabilities would:

- Allow for human-in-the-loop control
- Combine autonomous routines with manual intervention
- Enable teaching by demonstration

8.4 Enhanced Perception

Incorporating visual and/or tactile sensing would:

- Enable grasp planning based on object geometry
- Improve reliability of contact detection
- Allow for handling of previously unseen objects

8.5 Diffusion Policies for Grasping Arbitrary Objects

While our current implementation focuses on pick and place with known object shapes and poses, recent advances in deep learning offer promising approaches for handling arbitrary objects. Of particular interest are diffusion policies, which represent a cutting-edge approach to generating robot actions for complex manipulation tasks.

8.5.1 Foundations of Diffusion Models

Diffusion models are generative models that learn to gradually denoise data starting from pure noise. The process works in two phases:

Forward Diffusion Process:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{11}$$

where x_0 is the original data (in our case, a successful grasp trajectory), and x_t represents the data with noise added at step t. The β_t parameters control the noise schedule.

Reverse Diffusion Process:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{12}$$

where μ_θ and Σ_θ are learned by a neural network with parameters θ.

8.5.2 Diffusion Policies for Robotic Grasping

Diffusion policies adapt the diffusion model framework to generate robot actions conditioned on observations. For grasping arbitrary objects, the process works as follows:

8.5.3 Problem Formulation

Given a visual observation o of an object with unknown shape and pose, the goal is to generate a successful grasping trajectory $\tau = (a_1, a_2, ..., a_T)$ where each a_t represents a robot action (typically joint angles or end-effector poses).

8.5.4 Architecture

A typical diffusion policy for grasping includes:

- A vision encoder E_v that processes visual input (e.g., depth images, RGB images, or point clouds)
- A diffusion model that generates trajectories conditioned on the encoded visual features
- A time-step embedding to represent the denoising process stage

8.5.5 Training Process

Training a diffusion policy requires:

1. Collecting a dataset $\mathcal{D} = (o_i, \tau_i)$ of successful grasps across various objects
2. Training the diffusion model to denoise corrupted trajectories τ_t given visual observations o
3. Optimizing the network to minimize the denoising error:

$$\mathcal{L}(\theta) = \mathbb{E}(o, \tau_0) \sim \mathcal{D}, t \sim [1, T], \epsilon \sim \mathcal{N}(0, I) \left[|\epsilon - \epsilon\theta(o, \tau_t, t)|^2 \right] \tag{13}$$

8.5.6 Advantages for Arbitrary Object Grasping

Diffusion policies offer several key advantages for handling objects with arbitrary shapes and poses:

8.5.7 Multimodal Action Distribution

Unlike deterministic policies, diffusion models can capture multiple viable grasping strategies for a given object, addressing the inherent multimodality of grasping problems.

8.5.8 Uncertainty-Aware Behavior

The stochastic nature of diffusion models allows them to express uncertainty in grasping strategies, potentially enabling more robust behaviors when faced with ambiguous scenarios.

8.5.9 Generalization to Novel Objects

By learning a general mapping from visual observations to grasping trajectories, diffusion policies can generalize to previously unseen objects based on their geometric features.

8.5.10 Implementation Considerations

Integrating a diffusion policy into our current pick and place system would require:

8.5.11 Perception Pipeline

Adding cameras or depth sensors to obtain visual observations of objects, along with appropriate processing:

- Segmentation to isolate the target object
- Feature extraction to encode relevant geometric properties
- Integration with the existing MuJoCo simulation environment

8.5.12 Policy Execution

Modifying the pick and place pipeline to:

- Sample a trajectory from the diffusion model conditioned on the current observation
- Execute the trajectory through the existing robot control interface
- Potentially incorporate feedback mechanisms to adjust the trajectory during execution

8.5.13 Hybrid Approach

A practical implementation might combine:

- Diffusion policies for generating grasping strategies for arbitrary objects
- Traditional planning for collision-free movements between grasp and place locations
- Our existing contact-based gripper control for secure grasping

8.5.14 Recent Research Advances

Recent work in this area has shown promising results:

- **DiffusionPolicy** (Chi et al., 2023): Demonstrated successful grasping of diverse objects by generating actions directly from visual observations using diffusion models.