CS8803 Mobile Manipulation Final Report

Xiyang Wu
xwu391@gatech.edu

April 25, 2020

1 Introduction

1.1 Problem Statement

This project intends to investigate the problem: capturing the flying object with the robotic arm. The proposed robot model in this project is the Pincher robot. Instead of irregular objects, this project will use a uniform flying disc as the target. Though shaking from time to time, the rotating disc could maintain a generally stable pose during flight, which makes it easy to capture. The trajectory of the flying disc will be simplified as a parabola. The initial position of the flying disc and the base of the manipulator will be fixed. The mathematical analysis will assume that the trajectory of the target is reachable and known by the robot. The robot arm is required to generate the appropriate posture to catch the flying disc, while each joint will move simultaneously and precisely when the proposed pose generates, ignoring the transmission delay. A mathematical model of the capturing pose is required.

1.2 Related Work

As a classic problem in manipulation, many papers about autonomous control on fast movements like capturing and hitting that flying object with the robot arm have been proposed. U. Frese *et al.* [1] present 3D visual tracking system for the objects on flight, which is robust to the variance of the light condition, and implement the EKF-based object catching method on the DLR light weight robot arm. T. Senoo *et al.* [2] propose high-speed robot system that realizes ball batting task with the hybrid trajectory generator, which could relieve the effect of the nonlinear dynamics. R. Lampariello *et al.* [3] implements the B-splines to parameterize the trajectory and utilizes the KNN, SVM, GPR to optimize the initial trajectory. In this case, this method could achieve the real-time the flying object capturing task. S. Kim *et al.* [4] presents a practical real-time controlling system that enables the robot arm to catch irregular objects. This system realizes the in-line analysis of the object's mass distribution and determine the correct capturing posture and force based on the mathematical model and human demonstration. The final catching configuration is generated through the probabilistic model, considering the constraints on the joint output.

1.3 Project Contribution

Unlike the trajectory planning task for 2-D serial robots, the planning task in $SE(3)$ space introduces complicated kinematics problem. The robot is analyzed with the Denavit-Hartenberg Convention, which is an effective analyzing tool for the manipulator in $SE(3)$ space. To model the flying disc capturing task, a link model of the robot is designed separately in the GTSAM and MATLAB environment. The majority of the trajectory planning and inverse kinematic tasks are implemented in the MATLAB environment, including

the initial setting of the robot and the flying disc, and the trajectory planning and visualization tasks of the end effector, while MATLAB Robotic Toolbox is an effective tool in model visualization. After that, since the based of the link model in MATLAB is fixed by default, the implement of the exponential map and the advanced trajectory planning tasks with the iterative inverse kinematics method is completed in the Python and GTSAM environment. All the contribution in this project is made by myself.

2 Approach

2.1 Scenario Setting

In very beginning of this project, due to the difficulty in modeling the manipulator in the Python environment, I will implement the disc capturing method on the modified Pincher robot with the MATLAB Robotics Toolbox. Pincher robot is a 5-DOF robot. Its figure is Fig. 1. This robot has a vertical joint that can rotate in the X-Y plane, three serial horizontal joints and 1 joint that controls the gripper. Due to the size of the flying disc and its flying behaviour, while its motion in the vertical direction is considered as the free fall process, the original size of the Pitcher robot will be scaled for three times. To simplify the model, define the base frame S at the same location of Joint 1, so that the unit twist of the joint 1 is $S_1 = (0, 0, 1, 0, 0, 0)$. The second joint is 12.0 cm above the previous joint, so that its unit twist is $S_2 = (1, 0, 0, 0, 12.0, 0)$. The third and fourth joint locates at 31.5 cm and 63 cm higher than the joint 2 respectively. The unit twist for joint 3 is $S_3 = (1, 0, 0, 0, 43.5, 0)$, while the unit twist for joint 4 is $S_4 = (1, 0, 0, 0, 75.0, 0)$. The joint 5, which is 19.5 cm higher than the joint 4, will not be considered during the inverse kinematics, since it controls the gripper, which could not affect the spatial position of the manipulator's terminal point when capturing. The size of the gripper, which is 12.0 cm, will also be considered in the trajectory planning, since it introduces extra spatial constraints on the capturing motion.

Fig. 1. The Pincher robot

As for the capturing target, the shape of the flying disc is simplified as a cylinder. The cylinder's radius is 10 cm and its height is 1 cm. In the first scenario that I mainly analyze in this project, the initial position of the trajectory locates in $(50, -100, 120)$. The initial velocity vector is $(0, 400, 0)$, which means that the flying disc has a constant speed in the y direction, while it follows a free fall in the z direction, so that its trajectory follows a parabola trajectory in general. Considering the constraints in the disc capturing task, the capturing process should follow the process below: since the trajectory of the disc comes across the configuration sphere of the Pincher when the manipulator's base is fixed, as I mentioned in the problem statement. Based on the trajectory of the flying disc mentioned above, the capturing process is divided into four stages.

- The first stage corresponds to the flying disc trajectory with at least 70 cm in height. Since the initial position of the flying disc is higher than the maximum height that the Pincher robot could reach, and the capturing pose requires the manipulator to force the fourth link to stay parallel with the ground. In this case, the proposed spatial pose (x_t, y_t, z_t) the terminal point of the manipulator is the projection of the flying disc on the configuration sphere of the manipulator, though the effect of the last link, the gripper size and the radius of the flying disc should also be considered. The x axis of the terminal point, on the other hand, always points to the center of the flying disc.

- The second stage corresponds to the flying disc trajectory with lower than 70 cm, but still locates out of the configuration sphere. In this stage, the terminal point of the manipulator will try to maintain the same height as the flying disc, so that the z component of the terminal point equals to the height of the flying disc. However, due to the structure limitation of the robot, the x and y component of the terminal point corresponds to the projection point of the flying disc center on the configuration sphere, after appropriate compensation.

- The third stage happens when the height of the flying disc is lower than 70 cm, and its trajectory finally enters the configuration sphere of the manipulator. During this process, The axis of the terminal point of the manipulator should always be parallel with the ground when capturing, while the last link of the robot should maintain the same height as the flying disc. The operation trajectory of the terminal point, on the other hand, must take the physical constraint introduced by the flying disc into account, otherwise the actual flying disc could be hit by the robot and then fall down. The solution to this is to ensure the terminal point of the manipulator to follow a trajectory that locates in an approximately parallel plane to the trajectory of the target object. The distance between these trajectories is greater than the sum of the radius of the flying disc and the gripper size, in order to avoid any potential collision. Fig. 2 is the sketch of the tracking action, while all the relative positions of the manipulator, the gripper and the flying disc are presented in this figure. The ideal trajectory of the terminal point during this process follows the trajectory in the end effector trajectory plane, which is marked by blue dash line. The trajectory of the flying disc, on the other hand, follows the trajectory in the disc trajectory plane, which is marked by red dash line. These trajectory planes are parallel so that potential collision could be eliminated when implementing the tracking action.

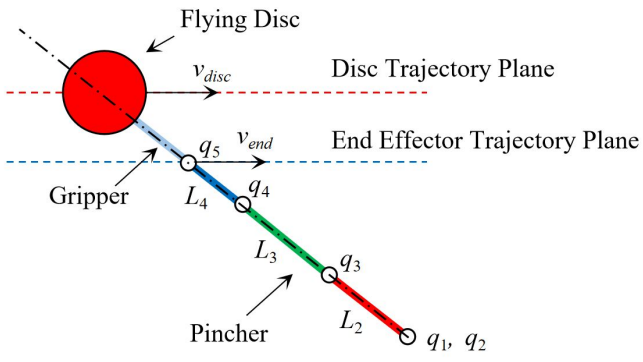


Fig. 2. The sketch of the tracking action

- The final stage of the capturing process begins at the same time that the terminal point enters and the ideal trajectory and could track the object for several iterations, as Fig. 2 mentions. This stage requires the robot move towards the center of the flying disc by continuing to track the trajectory of the flying disc. When the flying disc flies within the capturing scope of the gripper, the gripper begins to move forward by 2 cm for every 0.1 s. In this case, after 5 iterations, the flying disc could be captured

and the simulation is end. Considering the reaction time for the gripper, the 0.5 s for capturing is reasonable. The sketch of the final position between the robot and the flying disc is shown in Fig. 3. According to Fig. 3, the rim of the gripper should be co-planar with the symmetric axis of the flying disc. The fourth link of the manipulator, which is L_4 in the sketch, should be parallel with the ground. The constraints on the trajectory should always be satisfied so that potential collision could be eliminated when implementing the capturing action.

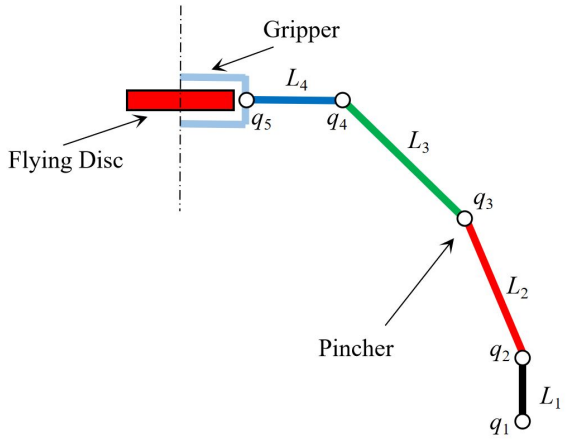


Fig. 3. The sketch of the final state of the capturing action

An advanced scenario is to introduce the velocity component in the x and z direction. The previous scenario simplifies the trajectory of the flying disc as a flat throw, so that the distance between the flying disc and the base of the manipulator is a fixed value. However, the throwing action in the real does not always this ideal situation. The trajectory could be an oblique line in the X-Y plane, and the flying disc could have an initial velocity in the z direction, which makes it become an oblique throw. Based on this, I proposed the second scenario for analysis in this project by introducing the initial velocity in the x and z direction to simulate the oblique throwing trajectory. In this scenario, I choose the initial position of the flying disc in the X-Y plane is $(50, -100, 105)$. The initial velocity vector is $(-200, 400, 100)$. In this case, the initial velocity vector of the flying disc is $(-2, 4, 1)$. A more complicated scenario is helpful to test the robustness of the trajectory planning method.

2.2 Methodology

2.2.1 Denavit-Hartenberg Convention

A classic analyzing method for the manipulator in $SE(3)$ space is the Denavit-Hartenberg Convention. According to the structure of the Pitcher robot, its D-H Convention is shown in Table. 1. From this table, θ_i are the output values for each joint. Since the Pincher robot should be vertical to the ground when all of the joint outputs are zero, I will introduce an offset angle $\pi/2$ at the second joint.

	θ	d	a	α	offset
Link1	θ_1	12.0	0	$\pi/2$	0
Link2	θ_2	0	31.5	0	$\pi/2$
Link3	θ_3	0	31.5	0	0
Link4	θ_4	0	19.5	0	0

Table 1: D-H Convention for the Pincher robot

Based on the transformation process between the proximate joint, which is introduced by the definition of the D-H convention, the general transformation matrix for joint i modeled by the D-H Convention is

$$T_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Incorporate the D-H Convention result in Table. 1 and the transformation matrices in Eq. (2), the corresponding transformation matrix for each joint is

$$T_1^s = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ \sin\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 1 & 0 & 12.0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_2^1 = \begin{bmatrix} -\sin\theta_2 & -\cos\theta_2 & 0 & -31.5\sin\theta_2 \\ \cos\theta_2 & -\sin\theta_2 & 0 & 31.5\cos\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$T_3^2 = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & 31.5\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & 31.5\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad T_t^3 = \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & 19.5\cos\theta_4 \\ \sin\theta_4 & \cos\theta_4 & 0 & 19.5\sin\theta_4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The transformation matrix from the base frame to the terminal frame is Eq. (3). In this equation, $\alpha = \theta_2 + \theta_3 + \pi/2$ and $\beta = \theta_2 + \theta_3 + \theta_4 + \pi/2$.

$$\begin{aligned} T_t^s(q) &= T_1^S(q_1)T_2^1(q_2)T_3^2(q_3)T_t^3(q_4)X_t^n \\ &= \begin{bmatrix} \cos\theta_1\cos\beta & -\cos\theta_1\sin\beta & \sin\theta_1 & \cos\theta_1(31.5\cos(\theta_2+\pi/2)+31.5\cos\alpha+19.5\cos\beta) \\ \sin\theta_1\cos\beta & -\sin\theta_1\sin\beta & -\cos\theta_1 & \sin\theta_1(31.5\sin(\theta_2+\pi/2)+31.5\sin\alpha+19.5\sin\beta) \\ \sin\beta & \cos\beta & 0 & 12.0+31.5\sin\theta_2+31.5\sin\alpha+19.5\sin\beta \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3}$$

2.2.2 The Jacobian Matrix

Calculating the Jacobian matrix for the robot is an important step in the inverse kinematics and trajectory planning, since this matrix establishes the relationship between the input value, like the commanded velocities in the joint space and the velocity of the robot's base, and the spatial position of the end effector. For the Pincher robot, this relationship follows Eq. (4).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = J(q) \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \end{bmatrix} \tag{4}$$

In Eq. (4), $\begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ is the velocity in the pose space, $\begin{bmatrix} \dot{q}_1 & \dot{q}_2 & \dot{q}_3 & \dot{q}_4 \end{bmatrix}^T$ is the commanded velocities in the joint space while the matrix $J(q)$ is the Jacobian matrix for the manipulator. Based on the D-H Convention in Section 2.1.1 and the composed transformation matrix in Eq. (3), the Jacobian

matrix for the Pincher robot is

$$J(q) = \begin{bmatrix} -\sin\theta_1 l_{c1} & -\cos\theta_1 l_{s1} & -\cos\theta_1 l_{s2} & -\cos\theta_1 l_{s3} \\ \cos\theta_1 l_{c1} & -\sin\theta_1 l_{s1} & -\sin\theta_1 l_{s2} & -\sin\theta_1 l_{s3} \\ 0 & l_{c1} & l_{c2} & l_{c3} \\ 0 & \sin\theta_1 & \sin\theta_1 & \sin\theta_1 \\ 0 & -\cos\theta_1 & -\cos\theta_1 & -\cos\theta_1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

In this equation, the substitutions l_{c1}, l_{c2}, l_{c3}, l_{s1}, l_{s2} and l_{s3} equal to

$$\begin{cases} l_{c1} = 31.5\cos(\theta_2 + \pi/2) + 31.5\cos\alpha + 19.5\cos\beta \\ l_{c2} = 31.5\cos\alpha + 19.5\cos\beta \\ l_{c3} = 19.5\cos\beta \\ l_{s1} = 31.5\sin(\theta_2 + \pi/2) + 31.5\sin\alpha + 19.5\sin\beta \\ l_{s2} = 31.5\sin\alpha + 19.5\sin\beta \\ l_{s3} = 19.5\sin\beta \end{cases} \tag{6}$$

2.2.3 The iterative inverse kinematics method

The inverse kinematics is the foundation of the trajectory planning. Without the appropriate inverse kinematics method, we could not derive out the feedback to control the output in the joint space with the trajectory in the pose space. However, the actual manipulators, especially for the spatial manipulators in the $SE(3)$ space, are always complicated and unfeasible to derive out their inverse kinematics result, since the Jacobian matrices for these manipulators are always singular, and one point in the pose space may introduce several solutions in the joint space. Due to difficulty introduced by implementing the iterative inverse kinematics method in the manipulators in the $SE(3)$ space. To address the inverse kinematics problem, the iteration process of the joint values could be considered as the following optimization problem.

$$\min_q \|p_{desire} - p(q)\|^2 \tag{7}$$

In this equation, p_{desire} is the intended pose of the manipulator's end effector, while the $p(q)$ is the current pose of the end effector.A common updating policy of $p(q)$ is to differentiate the updating process. During each iteration, the updated pose of the end effector $p(q + \delta q)$ is

$$p(q + \delta q) = p(q) + J(q)\delta q \tag{8}$$

Since the Jacobian matrix for the manipulator is always singular, δq could not be acquired by simply calculating the inverse of the Jacobian matrix. The basic idea to access δq is to use the pseudo inverse matrix. In this case, δq equals to

$$\delta q = J(q)^{\dagger}(p_{desire} - p(q)) \tag{9}$$

However, this method is unstable for some cases. In my implementation in the Python and GTSAM environment, the trajectory proposed by this method is corrugated and the optimization time is relatively long. A more stable method to inverse the Jacobian matrix is to use the Tikhonov regularization. This method introduces a penalty λ into the original optimization problem, so that the modified optimization problem is

$$\min_q \|p(q + \delta q) - p(q) - J(q)\delta q\|^2 + \|\lambda\delta q\|^2 \tag{10}$$

The updating step δq for the joint value q is

$$\delta q = (J(q)^T J(q) + \lambda^2 I) J(q)^T (p_{desire} - p(q)) \tag{11}$$

Based on the equations mentioned above, the iterative inverse kinematics method implemented in this project follows the steps below: for each simulation iteration, first initialize the joint value q with the values in the previous iteration. If the error between the desired pose and the actual one is smaller than the tolerance, the iteration loop will break. If not, the new joint value is updated by following Eq. (11). The value of the penalty term λ is decided by the Levenberg-Marquardt method. The initial value of λ is λ_0. For each iteration, if the pose error dose not increase, the penalty will constantly decrease by β until the pose error begin to increase. The final output result is constraint with the angle scope $[-\pi, \pi]$.

2.3 Implementation

2.3.1 The MATLAB Implementation

To take advantage of the visualization module and well-designed functions, the early stage of the kinematic analysis and the trajectory planning task for the Pitcher robot is designed with the MATLAB Robotics Toolbox generally. The basic step of the implementation is to model the manipulator with the D-H Convention. While all the joints of the manipulator are rotation joint, the link model of the Pincher model is connected with the $Revolute()$. After the link model of the manipulator has been established, the trajectory of the flying disc and the intended pose for the terminal point of the manipulator, which corresponds to the current position of the flying disc, are introduced to the simulation.

With the proposed transformation matrix for the terminal point in the base frame, the inverse kinematics of the manipulator could be implemented. However, one serious problem that comes from the MATLAB Robotic Toolbox is its inverse kinematics function. Since the Pincher robot is a non-redundant manipulator in $SE(3)$ space, more than one inverse kinematics result could be derived from the same terminal point's pose in the pose space. The inverse kinematics function in the MATLAB Toolbox, on the other hand, could only give one result, which could introduce some unrealistic or discrete points in the proposed trajectory. One way to solve this is to use the iterative inverse kinematics method in Section 2.2.3, but it is hard to calculate the pose error with the MATLAB. The detailed discussion of the inverse kinematics method is covered in the Python implementation. In the MATLAB simulation, to minimize the effect of the instability, I introduce the angular constraint on the corresponding joint and use the $ikcon()$ method to take these constraints into account. In this case, the inverse kinematics function could only send out the solution within a limited space, which could make the result more reasonable. Based on the inverse kinematics result, the forward kinematics and the plotting of the current pose of the manipulator could be done based on the the angle of each joint. The variation of each joint's angle will also be covered in the result.

2.3.2 The Python and GTSAM Implementation

Unlike the implement with the MATLAB Robotic Toolbox, simulate the manipulator with Python and GTSAM needs to establish the model manually. The model establish process basically incorporates the frame transformation between each joint, which introduces the $Pose3()$ transformation model in the GTSAM. The forward kinematics model of the manipulator is derived by the traditional transformation model and the exponential map model separately. The pose of the end effector of the manipulator is decided by the current spatial position of the flying disc, based on the method proposed in Section 2.1. Since the pose matrices in the GTSAM package are all represented in the Lie algebra form, the order the coordinates generated from the model needs to be modified to fit the Cartesian coordinates. The inverse kinematics function in the Python environment, on the other hand, is implemented with the iterative method, which is proposed in the $ik()$ method.

However, implement the iterative inverse kinematics method directly on the spatial manipulator is unfeasible. The optimization process to find the suitable q value is long, and the inverse kinematics result could be discrete between two proximate iterations, even though introducing the equivalent Jacobian matrix that corresponds to the derivation of Yaw, Pitch Roll angle, rather the angular velocity ω_x, ω_y and ω_z. To avoid this problem, I simplify the manipulator model in the Python implementation. According to the trajectory design plan that proposes in Section 2.1, the yaw angle of the end effector only corresponds to the output of θ_1, which is decided by the x and y coordinates of the flying disc. Ignoring the first joint, the rest of the joints could be taken as the serial link manipulator, and the sum of the joint value θ_2, θ_3 and θ_4 equals to $-\pi/2$. In this case, the original $SE(3)$ inverse kinematics problem could be taken as an equivalent problem in $SE(2)$ space. The modified relationship between the velocity in the pose space and the joint space is given in Eq. (12).

$$\begin{bmatrix} \dot{x}_{sub} \\ \dot{y}_{sub} \end{bmatrix} = J_{sub}(q) \begin{bmatrix} \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} \tag{12}$$

In Eq. (12), x_{sub} is the different between the position of the joint 3 in X-Y plane, while y_{sub} is the variation of the height of the joint 3. The corresponding Jacobian for this sub-structure of the manipulator is

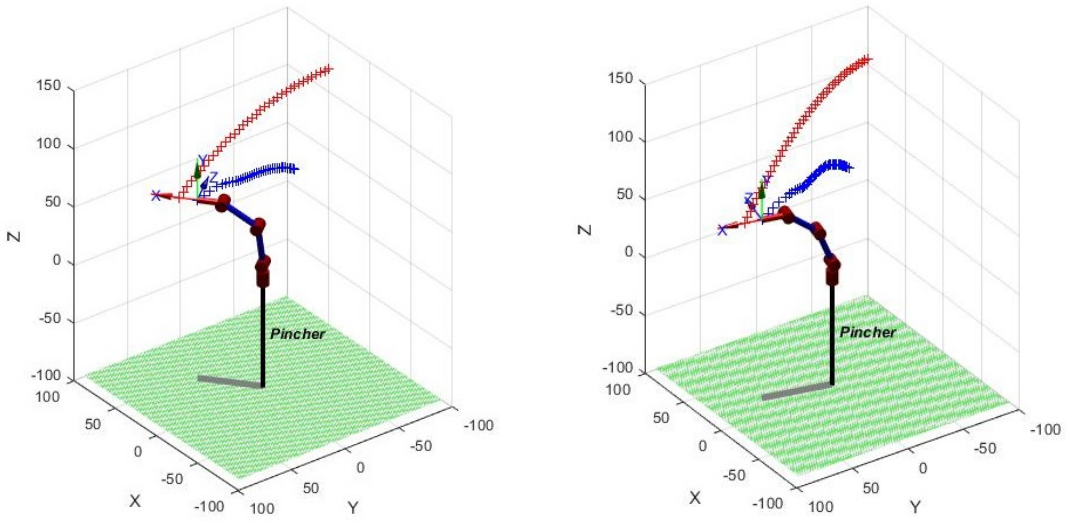
$$J(q) = \begin{bmatrix} -31.5\cos\theta_2 - 31.5\cos(\theta_2+\theta_3) & -31.5\cos(\theta_2+\theta_3) \\ -31.5\sin\theta_2 - 31.5\sin(\theta_2+\theta_3) & -31.5\sin(\theta_2+\theta_3) \end{bmatrix} \tag{13}$$

After each iteration, the values of the rest joints are decided by the spatial position of the flying disc and the current value of joint 2 and 3. By selecting the initial penalty value λ_0 as 100 and the penalty updating step β as 2, the penalty value is also updated in each iteration. The iteration loop will continue until the pose error is smaller is 10^{-3}. The trajectory planning result is presented as the simulation of the manipulator, which is plotted by the $matplotlib$ package, and the variation of the distance between the end effector and the flying disc in X-Y plane, since the relative spatial position of the end effector is absurd due to the perceptive angle.

3 Results

3.1 MATLAB Simulation Result

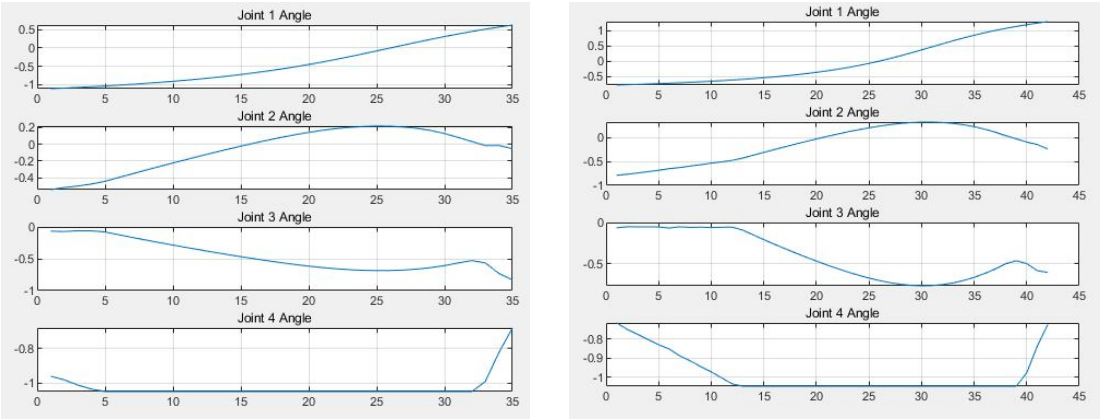
The MATLAB simulation generally incorporates the manipulator model implementation and the trajectory planning. Based on the method and model mentioned in Section 2, the link model of the Pincher robots, which is designed with the D-H Convention, and the trajectory of the flying disc and the end effector are shown in Fig. 4. In both two scenarios, the simulation time step 0.01 s, and the entire simulation time scope is 0.5 s. The red parabola curvature in both two figures is the trajectory of the flying disc. The blue irregular curvature is the proposed trajectory of the end effector. Both two scenarios are included in Fig. (4), while Fig. 4(a) is the flat throwing scenario and Fig. 4(b) is the oblique throwing scenario. According to the result, the output values of the joint 2, 3, 4 are always negative. The phenomenon makes the spatial pose of the manipulator always fits the convex spatial curvature, which makes the result more reasonable. The trajectory of the end effector always locates in the parallel plane from the plane that contains the trajectory of the flying disc's center before the capturing action. Though the trajectories in Scenario 1 and 2 fit the same variation pattern in general, the proposed trajectory for oblique throwing scenario has more fluctuation, which may be introduced by the extra velocity component of the flying disc in the x and z direction. As for the time needed to catch the flying disc, In Scenario 1 (Flat Throw), the manipulator takes 35 iterations to catch the flying disc. In Scenario 2 (Oblique Throw), the manipulator takes 42 iterations to catch the flying disc.



(a) (b)

Fig. 4. The trajectory of the flying disc (Red) and end effector (Blue) in MATLAB
(a) Scenario 1 (b) Scenario 2

To evaluate the feasibility of the trajectory planning method, the angle variation of each joint in each scenario during the simulation is shown in Fig. 5. From angle variation plot during the simulation, we could find that the general variation trend of each joint's angle is smooth. Though the changing pattern is generally the same in both two scenarios, the variation pattern of the angle for joint 3 is steeper in the oblique throwing, which may be introduced by the initial velocity of the flying disc in the z direction. Since the angle of joint 3 varies more heavily in the oblique, the capturing action for the flying disc is more difficult to implement, which could explain the fluctuation in the planned trajectory and the extended operation time in Scenario 2.



(a) (b)

Fig. 5. The angle variation for each joint during the simulation (a) Scenario 1 (b) Scenario 2

The simulated tracking action in the MATLAB environment is shown in Fig. 6, while the red plate in the scenario represents the flying disc. When the rim of the manipulator's gripper reaches the symmetric axis of the flying disc, the simulation is end. Two videos about the trajectory of the end effector and the flying disc, as well as the capturing process for each scenario have been included in the attachment of this report.

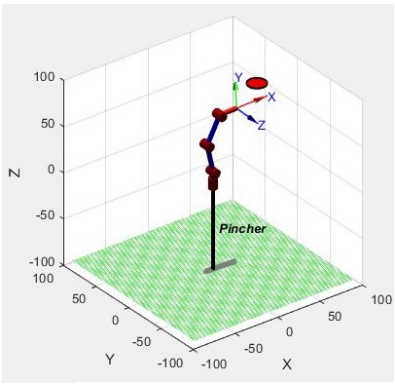
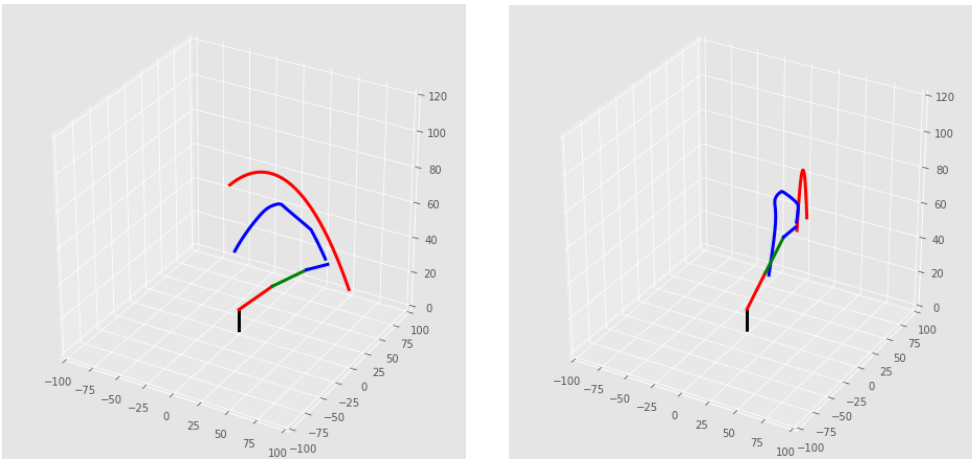


Fig. 6. The simulated tracking action in MATLAB

3.2 Python and GTSAM Simulation Result

The goal of the Python and GTSAM simulation is to verify the manipulator built with the tradition frame transformation and the exponential map method, and implement the iterative inverse kinematics method manually. In the Python simulation section, I also implemented two scenarios, the flat throwing scenario and the oblique throwing scenario. In both two scenarios, the simulation time step 0.01 s, and the entire simulation time scope is 0.5 s. The simulation result for both two scenarios are shown in Fig. 7 (a) and (b). The attachment of this report include the implementation and the full animation of the capturing process of the manipulator.

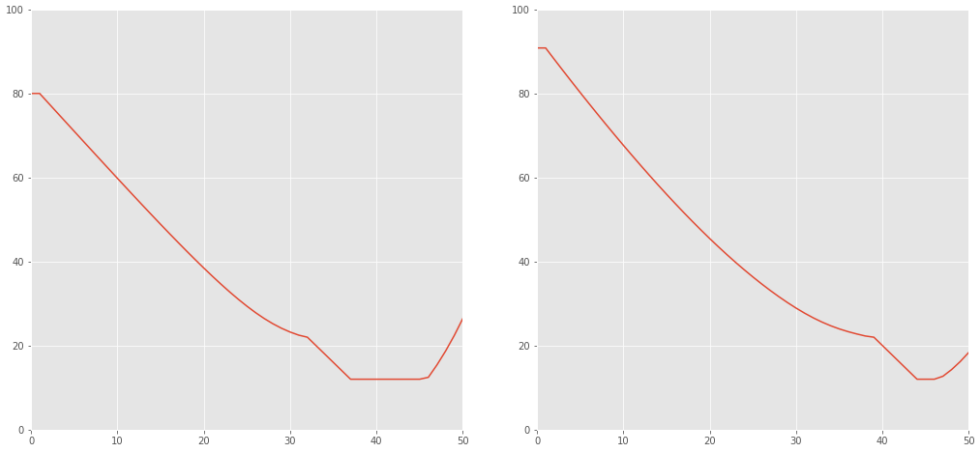


(a) (b)

Fig. 7. The trajectory of the flying disc (Red) and end effector (Blue) in Python
(a) Scenario 1 (b) Scenario 2

Due to the inconvenience introduced by the fixed view perceptive in Python animation, the simulation may not make sense when evaluating the performance of the algorithm, though the pose of the fourth link is always parallel with the ground. A more straightforward way to determine whether the flying disc has been caught by the manipulator is to compute the distance variation plots between the terminal point of the end effector and the center of the flying disc in X-Y plane. The distance variation plots for both two scenarios are proposed in Fig. 8. From the distance variation plot, we can find that the flying disc could be caught in the 35th iteration in Scenario 1, while Scenario 2 needs 42 iteration to caught the flying disc. As for the distance variation during the simulation, the minimum distance between the manipulator and the flying

disc is 10 cm in both two scenarios, which satisfies the successful capturing condition. Both two results match the simulation results in the MATLAB section, so that we could consider that the current manipulator simulation in the Python and GTSAM environment is feasible.



(a) (b)

Fig. 8. The distance between the flying disc and end effector in X-Y plane
(a) Scenario 1 (b) Scenario 2

4 Discussion

This project analyzed robotic kinematics problem for the capturing the flying disc, which follows the flat throwing and oblique throwing motion trajectory, with the 5-DOF Pincher robot. While the spatial curvature trajectory planning in the $SE(3)$ space is relatively complicated, this project proposed the trajectory planning method that generally works for the objects that follows the parabola trajectory. In the MATLAB simulation part of this project, I mainly analyzed the forward kinematics of the manipulator with the D-H Convention. Besides, I implemented the trajectory planning method that is generally based on projection of the current spatial position of the flying disc on the configuration sphere. With this method, the end effector could track the trajectory of the flying disc that locates out of the configuration sphere simultaneously, which makes it easier to catch the disc. The result shows that this model works well on both two scenarios proposed in the MATLAB environment, including the flat and oblique throwing scenario. The trajectory of the end effector proposed and the corresponding output of each joint is continuous and smooth, which accurately matches the ideal pose of the end effector. The flying disc, on the other hand, could be caught with the simulation time scope, according to the simulation result.

As for the implementation in the Python and GTSAM environment, the goal of this part is to investigate the forward and inverse kinematics manually. Instead of establishing the model with the well-made function in MATLAB, I built the link model of the Pinch robot with the GTSAM function in two separate ways, the traditional method with the transformation matrices and the exponential map. The forward kinematics result of both two methods are the same as the result derived from the MATLAB Robotics Toolbox. The inverse kinematics method in the Python environment is based on the regularized iterative method. To implement the inverse kinematics accurately, I converted the manipulator model into the simplified serial link form. Though the joint angle may vary heavily at some points, the simulation result confirms the feasibility of this method. In future investigation, a more general and robust method is needed to solve the inverse kinematics problem, so that more complicated trajectory planning and manipulator simulation could be enabled in the Python and GTSAM environment.

Though the current trajectory planning works well in the given scenarios, more advanced experiments show that the current design only works when the flying disc locates with the sub-configuration that $y_t > 0$ and the distance between the flying disc and the manipulator's base in the X-Y plane the should be greater than 30 cm respectively. When the spatial constraint is not satisfied, the inverse kinematics may introduce unrealistic results. To solve this problem, a modified trajectory planning needs to be proposed so that the manipulator could catch the flying disc that comes across its configuration sphere without extra constraints.

As for the future work that could be done on this project, the first thing is to model the manipulator in the realistic simulation environment, like the ROS and Gazebo. Right now, I have finished the kinematics part for the trajectory. However, implement the trajectory planning method on the simulated manipulator or even the actual robot, is much different from the mathematical analysis in the MATLAB and Python. More constraint on the manipulator's structure could be introduced. At the same time, the dynamic analysis for the manipulator could be feasible, while the mass distribution of the manipulation is offered by the model in the simulator. Based on this, a more precise and feasible trajectory planning method could be realized with the simulator, which helps the implementation of the flying disc capturing task on the actual robot.

Another thing that needs to be considered is the trajectory of the flying disc. In this project, to simplify the model and focus on the kinematics, I suppose that the initial position and velocity of the flying disc is fixed, though two different scenarios have been proposed. However, the trajectory of the flying disc in the real world is uncertain. Many factors, like the wind speed, the air resistance and the initial force vector applied on it could seriously affect the shape of the eventual trajectory. A vision-based object perception method is needed to track the object online. This method should be able to separate the object from the background, track the target all the time and predict the next spatial position of the object with the state estimation method like the Kalman filter. With the perception module, the manipulator could plan the trajectory of the end effector online to get accustomed with the uncertain nature of the flying object's trajectory, which will make the manipulator much more robust to the difficult capturing task in the real world.

5 Meta-learning

The basic idea of this project is to further investigate the kinematics and trajectory planning for the manipulator. Though the class note has provided the kinematics analysis for the serial robot with three links, it is still not feasible in the real world, while the actual robot is designed in $SE(3)$ space in general. Out of the 2-D inverse kinematics example offered in class and assignments, I implemented the trajectory planning task to capture the flying disc, which follows a parabola trajectory in the $SE(3)$ space with a 5-DOF robot in detail. The analysis of the robot in the $SE(3)$ space is different from the analysis for the serial robot in the $SE(2)$ space, which is mentioned in the class notes. It introduces the complicated analysis of the manipulator's structure. The trajectory planning in the $SE(3)$ space, basically, incorporates the forward and inverse kinematics for the given robot. Through this process, I further enhanced my knowledge on the content covered in the second half of this course. Besides, by implementing the model with some popular robot analyzing and simulation tools, like the MATLAB Robotics Toolbox and the GTSAM, I become more familiar with these effective tools. Furthermore, since I intend to work on a project about the flying object capturing task with the robot manipulator, the effort on this project gives me a meaningful perceptive on how the object capturing task could be designed. My later work, like the simulation or even the implementation on the actual robot, could be done based on my result in this project.

References

[1] U. Frese, B. Bauml, S. Haidacher, G. Schreiber, I. Schaefer, M. Hahnle, and G. Hirzinger, "Off-the-shelf vision for a robotic ball catcher," in *Proceedings 2001 IEEE/RSJ International Conference on*

Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180), vol. 3, pp. 1623–1629 vol.3, 2001.

[2] T. Senoo, A. Namiki, and M. Ishikawa, "Ball control in high-speed batting motion using hybrid trajectory generator," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1762–1767, 2006.

[3] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, "Trajectory planning for optimal robot catching in real-time," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3719–3726, 2011.

[4] S. Kim, A. Shukla, and A. Billard, "Catching objects in flight," *IEEE Transactions on Robotics*, vol. 30, no. 5, pp. 1049–1065, 2014.