



TALLINN UNIVERSITY OF TECHNOLOGY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL POWER ENGINEERING AND MECHATRONICS

**RESEARCH ON INDUSTRIAL MANIPULATOR
TRAJECTORY OPTIMIZATION BY APPLYING
NONLINEAR MODEL PREDICTIVE CONTROL**

**TÖÖSTUSMANIPULAATORI TRAJEKTOORI
OPTIMEERIMISE UURIMINE, KASUTADES
MITTELINEAARSET MUDEL-ENNETAVAT
JUHTIMIST**

MASTER THESIS

Student: SHUANG LI

Student code: 194356MAHM

Supervisor: SALEH RAGHEB SALEH ALSALEH
ALEKSEI TEPLJAKOV

Tallinn 2021

(On the reverse side of title page)

AUTHOR'S DECLARATION

Hereby I declare, that I have written this thesis independently.

No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"....." 20....

Author:

/signature /

Thesis is in accordance with terms and requirements

"....." 20....

Supervisor:

/signature/

Accepted for defence

"....."20... .

Chairman of theses defence commission:

/name and signature/

Non-exclusive licence for reproduction and publication of a graduation thesis¹

I _____ (author's name)

1. grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis

_____,
(title of the graduation thesis)

supervised by _____,
(supervisor's name)

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

_____ (date)

¹ The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Department of Electrical Power Engineering and Mechatronics
THESIS TASK

Student: Shuang Li 194356MAHM

Study programme, MAHM02/18 Mechatronics

main speciality: Mechatronics

Supervisor(s): Saleh Ragheb Saleh Alsaleh, Engineer, 620 3201;

Aleksi Tepljakov, Research Scientist, 620 2116

Consultants:

Thesis topic:

(in English) Research on Industrial Manipulator Trajectory Optimization by Applying Nonlinear Model Predictive Control

(in Estonian) Tööstusmanipulaatori trajektoori optimeerimise uurimine mittelineaarse mudeli ennustava juhtimise abil

Thesis main objectives:

1. Research on existing methods and involve a general understanding of the topic
2. Implement the optimization method on the selected model and environment
3. Evaluate the performance of the theory

Thesis tasks and time schedule:

| No | Task description | Deadline |
|----|---|------------|
| 1. | Research about existing methods on relevant topics | 05/03/2021 |
| 2. | Compile a sufficient methodology by analyzing the background research | 13/04/2021 |
| 3. | Robot model selection and simulation design | 04/05/2021 |
| 4. | Write the report of the thesis | 14/05/2021 |

Language: English **Deadline for submission of thesis:** "15" May 2021

Student: Shuang Li

"15" May 2021

/signature/

Supervisor: Saleh Ragheb Saleh Alsaleh ".....".....20....a

Aleksi Tepljakov ".....".....20....a

/signature/

Head of study programme: ".....".....20....a

/signature/

Terms of thesis closed defence and/or restricted access conditions to be formulated on the reverse side

CONTENTS

| | |
|---|----|
| CONTENTS | 5 |
| List of Figures | 6 |
| PREFACE | 7 |
| List of abbreviations and symbols | 8 |
| 1. INTRODUCTION..... | 9 |
| 1.1 Motivation..... | 9 |
| Thesis structure | 10 |
| 2. LITERATURE REVIEW..... | 12 |
| 2.1 Existing Pattern | 12 |
| 2.2 Modeling and Motion | 14 |
| 2.2.1 Configuration and DoF..... | 14 |
| 2.2.2 Spatial Representation and Transformation | 14 |
| 2.2.3 Kinematics..... | 15 |
| 2.3 Trajectory generation and Dynamics | 15 |
| 2.4 Optimization | 16 |
| 2.4.1 Time-optimal..... | 16 |
| 2.4.2 Energy consumption..... | 17 |
| 2.4.3 Singularity | 17 |
| 2.4.4 Collision avoidance..... | 17 |
| 3 METHODOLOGY | 19 |
| 3.1 Overview | 19 |
| 3.2 Motion Planning | 20 |
| 3.3 Nonlinear Model Predictive Control | 21 |
| 4. SIMULATION STUDIES..... | 24 |
| 4.1 Digital model selection..... | 24 |
| 4.2 Experiment setup..... | 25 |
| 4.3 Experiment discussion | 38 |
| 5. CONCLUSION | 45 |
| 6. DISCUSSION | 46 |
| LIST OF REFERENCES | 50 |
| APPENDICES | 53 |
| Appendix 1 – Programming code for MATLAB | 53 |

List of Figures

| | |
|---|----|
| Figure 2.1 The standard Denavit and Hartenberg link parameters [13] | 15 |
| Figure 3.3.1 Basic NLMPC Control Loop [26] | 21 |
| Figure 4.1.1 Working range of the Yumi Robot [37] | 25 |
| Figure 4.1.2 ABB FlexPendant [36] | 25 |
| Figure 4.2.1 ABB Yumi with single arm in home configuration | 26 |
| Figure 4.2.2 Trajectories in task space and joint space..... | 27 |
| Figure 4.2.3 Initial position of the robot model..... | 28 |
| Table 4.2 Yumi movement parameters [36]..... | 28 |
| Figure 4.2.4 Flowchart of applied methodology | 29 |
| Figure 4.2.5 Logic control of the pick and place process | 30 |
| Figure 4.2.6 End-effector move to approaching position. | 31 |
| Figure 4.2.7 End-effector move to placing position | 32 |
| Figure 4.2.8 Final position of the robot model | 32 |
| Figure 4.2.9 Display message on the command window in MATLAB | 33 |

PREFACE

The research done in this dissertation was to explore the solution for the industrial manipulator trajectory optimization problem by applying a nonlinear model predictive control model. A digital model of the ABB Yumi robot was used in MATLAB to move along with a working environment trajectory with two fixed obstacles to reach the target. Firstly, interpolation polynomials were used to generate the first segment of the trajectory, point-to-point planning, and motion tracking applied in Simulink under torque control. Then, a pick and place task was executed with a nonlinear model predictive controller under constraints to compare with the previous method about the optimization results for the trajectory.

The author wishes to express her sincere appreciation to supervisors Saleh Ragheb Saleh Alsaleh and Aleksei Tepljakov for their generous help and superb support in carrying out my studies and the thesis.

Keywords: Digital robot, Trajectory optimization, Nonlinear Model Predictive Control, Collision avoidance

List of abbreviations and symbols

| | |
|-------|------------------------------------|
| Cobot | Collaborative Robot |
| DoF | Degree of Freedom |
| FW | Forward Kinematics |
| IK | Inverse Kinematics |
| NLMPC | Nonlinear Model Predictive Control |
| RBT | Rigid Body Tree |

1. INTRODUCTION

The market for robotic manipulators is growing tremendously due to their high-quality performance and productivity in industrial applications [1]. Most operations in industries such as object pick and place, painting, sketching, welding, all mentioned above, are highly dependent on various manipulators. The manipulator should move along through one or a set of intermediate waypoints without stopping, building the path over time, so-called trajectory planning. Trajectory planning has been widely discussed in the academic field from several aspects. The problem of trajectory optimization is a common requirement to adapt the manufacturing process and improve the productivity of the robotic system. To obtain an optimal trajectory, understanding the factors that affect manufacturing efficiency while planning the revolution is crucial. One standard scheme of trajectory planning can be summarized into three steps [1]–[3]:

1. Identification of specific manipulator and determination of the relationship between each joint parameter of the robotic system in space.
2. Waypoints selection by applying forward and inverse kinematics.
3. Design Dynamics formulation to Control the motion of the manipulator.

Another approach is to investigate optimization methods to motion planning, which can involve certain constraints to optimize the trajectory of the manipulator by calculating the motion under general conditions [4]. The advantage of this class of methods is that the optimization process focuses not only on time required for the trajectory but also on other objectives such as collision avoidance, torque minimization, and energy reduction [5]–[8].

This thesis aims to explore a solution for the trajectory optimization problem of the industrial manipulator by applying a nonlinear model predictive controller. The proposed methods are validated on a digital twin of the ABB Yumi robot created in the MATLAB environment. A rigid body tree representing rigid body connectivity with joints is used to build the manipulator model in MATLAB. This model later executes the pick-pack motion as required using our proposed method, where constraints and adjustments are implemented as multiple pairs of functions.

1.1 Motivation

Robotic manipulators or industrial robots are widely used in modern industries. For example, there can be infinite trajectories to move an object from one point to another point. From the

industry perspective, it is crucial to generalize all the expectations for each specific execution, such as collision avoidance, human safety, torques on each joint of the robot are satisfied. However, from a theoretical point of view, more researches have been carried out based on the pre-programming solutions of manipulators. More analytical studies are required to provide a solution from another perspective.

On the other hand, modeling and simulation are widely used to represent a physical or logical system to gather data and make an informed decision based on the data. If we can decompose the constraints into functions and ranges, algorithms may help improve efficiency or even predict potential risks.

All questions mentioned above encourage the author to continue research on this thesis work.

Because of all the research mentioned above, the current problems in the field are shown as below:

- Physical robots are pre-programmed to accomplish industrial tasks, which is less flexible when a change is needed [9].
- Testing and adapting the physical robot requires continuous and accurate data resources, which is difficult to obtain in the actual manufacturing procedure [10].
- The existing research on trajectory optimization mainly focuses on algorithms developing. Thus, only the simplified model of the manipulator model is used and lacks flexibility for converting to operations.
- As a result of preliminary research, the data in trajectory optimization is almost confined, disintegrated, and dormant.
- Keeping control of an integrated robot while planning the trajectory or various constraints needs to be considered problematic.
- Existing trajectory optimization methods are not fully developed on physical robots due to high cost and sophisticated technical limitations.
- Less academic research about the possibility of integrated robot maintenance and more people focused on safety analysis.

Thesis structure

Chapter 1 is the introduction

Chapter 2 of this document is dedicated to Literature Review.

Chapter 3 will talk about the thesis's methodology, the solution of the questions, hardware, and software selection will be implemented in this chapter.

Chapter 4 will be depicting the results from the program.

Chapter 5 will be dedicated to Conclusion and the Future Scope of the work.

2. LITERATURE REVIEW

2.1 Existing Pattern

There is a difference between a path and a trajectory. A path is a spatial concept that describes how we travel from point A to point B in the world. And a trajectory is a path with a timetable. It tells us how fast we can travel along the path and when we should arrive at each stop. An elemental character of a trajectory is smooth motion with time. Furthermore, the process to include other constraints and minimize (or maximizes) some performance measure is trajectory optimization.

Many researchers have explored many optimization schemes from various perspectives from the different objectives of the optimization parameters. Common optimization parameters include time [11] and energy [12][13]. There are a few studies that focus on other aspects, such as collision avoidance [14], limitation on the acceleration difference of the trajectory [15], improvement of stiffness performance [16]. Although the optimization objectives are various, the pattern that needs to be investigated is similar, and the basic design can be summarized in the following table.

We can find that the trajectory optimization scheme can be formally divided into numerical and analytical through the above table. The numerical method can visualize each parameter and is flexible to adjust according to different operation requirements. However, using algorithms to calculate each time step's optimal position can save time compared with kinematics calculation for the entire trajectory.

Table 1 An overview of existing research methods

| Application | Optimization Method | Manipulator Modelling | Modelling | Motion | Dynamic | Ref |
|--|--|---|---|---|---|--------------|
| Time-optimal | PSO | CRO-S80 (6 DoF) | D-H parameter | FK & IK | 4-3-4 Spline interpolation | [17] |
| Trajectory planning | PSO | Specific robot 3DoF | Not separate motion planning, trajectory is calculated by algorithms and cost functions | | | [4] |
| Trajectory planning and tracking | NLMPC | Aerial robot | Vector representation | First-order semi-implicit method discrete time continuous dynamic | | [18] [19] |
| Desired trajectory tracking | NLMPC | Aerial manipulator (3DX-X8 coaxial multirotor with a 3DoF serial arm) | Multirotor-arm system considered has n DoFs, vector representation | Two sinusoid trajectories | Cost and constraints functions are used to track the end effector | [20] |
| Time-optimal | Higher order IK | 6-DoF industrial robot | Vector representation | FK & IK | | [21] |
| Online synchronization and time-optimal | A quadratic program is used to solve path optimization problem | 6-DoF industrial robot (experiment on physical robots) | | Third order splines are used to do geometric path planning | Comau Racer3 robot with R1C controller | [22] |
| Dimensions direction and magnitudes for velocity and force control | Algorithms for solving velocity and force control | 6-DoF industrial robot (ABB IRB 120 robot arm) | Vector representation using body twist | | IRC5 controller | [23] |

Moreover, some theoretical methods and new technology from other researches show potentials for trajectory optimization. One approach is dedicated to planning the trajectory for a 6DoF robotic manipulator to perform image-based visual servoing (IBVS) [17] tasks. Instead of a spatial construct motion, the concept of trajectory planning aims to reach the target feature by applying a combination of the known feature motions. Another perspective is to investigate machine learning, i.e. the widespread use of algorithms [18]. The study shows the evolutionary algorithms can be classified into ES, GA, and estimation of distribution algorithms. Following with three objective functions to minimize the joint errors and energy consumption, minimize the end-effector's tracking error, maximize the workspace, respectively. Those functions are tested on a 3DoF manipulator in two case studies, and the result shows with various constraints, algorithms can improve the performance according to the motion sequentially.

In the following section, the process of trajectory planning, which is preliminary, will be discussed separately.

2.2 Modeling and Motion

2.2.1 Configuration and DoF

The position and orientation that can demonstrate the manipulator in the specific geometric frame are called configurations. For the industrial manipulator, the Cartesian coordinate system configuration also indicates their joint rotation angles [19]. Trajectory planning is performed in the space compiled by every configuration of the manipulator. Degree of Freedom (DoF), refers to the number of joints in an arm manipulator. Each joint has one degree of freedom. When the robot has more joints than it needs, the extra degree is redundant; when the joint axis alignment occurs, the manipulator has lost one degree of freedom.

2.2.2 Spatial Representation and Transformation

A typical reason for a trajectory is to shift the end-effector of a manipulator from A to B over a specific time frame in robotics. How to generate trajectory in the space will be answered in this section. If a manipulator is a snake moving toward the food on the grass, all the joints on the snake moving together and the distance between the snake and food is known, only a single generalized coordinate is needed to describe their location. More generalized coordinates are required to describe the rotations and motions completely for a robotics manipulator with a fixed base, rotational joints, and prismatic links. Many types of research mentioned above have shown that the initial and endpoint is the beginning of trajectory planning. The number of degrees of freedom (DoF) of the manipulator is known as the number of no holonomic constraints on the manipulator. A point in its configuration space can represent any position and orientation of the manipulator. Any point in the configuration space can be generalized to a point in the task space but the inverse is depending on the task space, as its name.

In terms of configuration space, those axes of motion correspond to the manipulator's degrees of freedom [20]. The manipulator configuration can be represented as a vector $q \in R^N$ where N is the number of degrees of freedom. In that case, the concept behind motion from point A to point B is the motion from an initial configuration vector to a final configuration vector.

2.2.3 Kinematics

Forward kinematics is generating the motion from manipulator configuration to its end-effector pose. And the inverse which the target posed of the end-effector is given, determination of the manipulator is known as inverse kinematics. When we describe the posture of a manipulator end-effector, a sequence of relative poses is required. The most common approach to describe a serial-link manipulator is Denavit-Hartenberg notation. (DH parameter) The relationship between two nearby joint axes is defined as the notation of a link, the standard D-H link parameters are shown in Figure 2.1 as below. Six parameters represent translation and rotation. One consequence of the kinematics solution for trajectory plan is the complexity of algebraic solution subjects to the number of joints, and more analytical solution should be considered.

Another consequence is when the joint limits appear, or singularities which means not all poses within the manipulator's workspace can be achieved. No solution can be generated by kinematics in that case. One standard method to overcome is adding constraints on the manipulator.

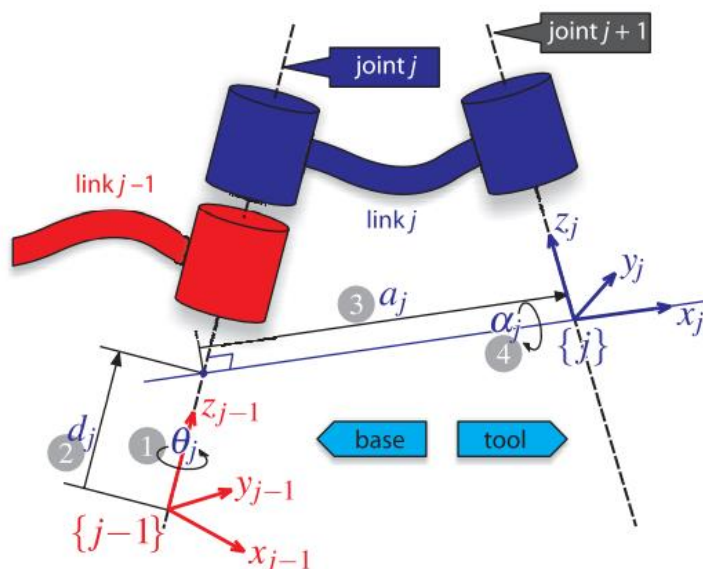


Figure 2.1 The standard Denavit and Hartenberg link parameters [20]

2.3 Trajectory generation and Dynamics

As mentioned before, trajectory planning is the motion of a manipulator end-effector from point A to point B over time. It is essential to analyze the action from a practical perspective, the end-effector is moved by forces and torques applied to the joints. When the manipulator's end-effector moves with a velocity, exerting appropriate dynamic equations that describe the manipulator state by understanding the joint torque. Compute the required joint forces

according to the target trajectory and the rigid-body dynamic forces is the question before choosing the control system for manipulator motion.

A common approach to controlling the manipulator at the academic level is to list the functions for compiling the trajectory from the initial pose to the end pose, for example a series formula of 4-3-4 trajectory polynomial [21], S-curve interpolation algorithms [22], and 5-3-5 spline trajectory [23]. As mentioned above, polynomial splines are used widely in trajectory generation, and higher order polynomials can generate the trajectories smoother.

2.4 Optimization

2.4.1 Time-optimal

Time optimization is the most frequently discussed topic in the trajectory optimization process. It is a critical weight for improving the productivity of robotics systems. According to its name, time optimization can be interpreted as moving the end-effector of the manipulator from the initial point (or pose) to the endpoint (or pose) in the minimum time period. As robotics manipulators in manufacturing industries increase, research on time optimization is also growing in diversity. One of the common approaches is to formulate a trajectory generation problem as a motion function. With system dynamic and other bounds constraints, minimizing time over the motion along the path is time optimization [24][25]. Algorithms have also provided a research direction on this topic. It is well known that cubic polynomial interpolation and b-spline interpolation are simple to compute and can easily transition with boundary conditions satisfied. However, computing the additional constraints functions, which is the traditional optimal method, will increase the difficulty of trajectory generation. Also, higher-order polynomials are needed to generate smoother trajectories [26][23]. One research proposes a method that optimizes the running time on each joint uses genetic algorithms after generating the trajectory by cubic polynomial interpolation. In that case, the period with the shortest time during the trajectory is obtained as objective. The objective function is under all the kinematics constraints, such as velocity limits, acceleration and jerk on each joint's limit [26]. Another algorithm is introduced for solving trajectory problems of industrial manipulators, called trajectory-planning beetle swarm optimization. In the research, the optimization process is presented as a group of beetles in the working space, the position of each beetle stands a solution of the problem. Trajectory planning is introduced as a fourth-order polynomial time function, a time minimum model is indicated with few pairs cost functions, obstacle avoidance and other mechanical limits are dedicated as a set of constraints, the fitness of each beetle is updated at each time step. There is an obvious result on execution time compared to the new algorithm with genetic algorithm and particle swarm optimization algorithm [19].

2.4.2 Energy consumption

Energy consumption can directly reflect the energy efficiency. Reduce the energy consumption can be achieved from researching on different perspectives, for example, decrease weight, regenerative braking, and optimize the energy management system [27]. Shorten the operation time is one strategy to provide an optimization on existing hardware solutions [13]. Solving a trajectory planning problem by calculating the input torques required to move a manipulator along a path in a range of execution time. And the optimization function is subject to a set of dynamic constraints and cost functions. Several researches show the similar pattern in which defined power consumption function and minimize the cost at each operation [28]. In this section, dynamic programming is also applied to solve the energy consumption problem by controlling the dynamic system within a state space.

2.4.3 Singularity

Singularity is a problem when the rotational axis of the end term in the sequence becomes parallel to the rotation axis of the base term. When the manipulator approaches a particular configuration in which trigger singularity in one direction, a certain degree of freedom will be lost so that there is no achievable solution for the manipulator to move into that direction. Therefore, it is possible that the solution is not physically achievable due to the limits of joint angles and potential collision within the manipulator itself. In that case, an end pose may not be achievable for the end-effector due to singularity since the effective degrees of freedom is reduced. Especially the inverse kinematics problem solution of a redundant manipulator can be infinite since it is widely recognized that requiring a six or more than six DoF manipulators to follow the generated trajectory in industries. One approach is dedicated a set of constraints solutions along with the end-effector task [29].

2.4.4 Collision avoidance

There are various perspectives on collision avoidance, such as preventing collision with itself, avoiding collision between a manipulator and the workspace, and avoiding collision between manipulator and human co-worker. The last of which is often applied to collaborative manipulators, where a manipulator and human share workspace and collaborate on industrial operations [30], [31]. To further refine the problem, collision avoidance can be divided into collision detection and motion plan when a possible collision is detected. One approach is to compile repulsion vectors and a function of the error

between end-effector position and the goal position with a mathematical representation of manipulator kinematics, the manipulator is controlled on the joint velocity level. When the minimum distance between end-effector of the manipulator and human or obstacle is reached, the end-effector velocity decrease [32]. In addition, for complex industrial collaborative operation, the collision avoidance solver is frequently generated in a state architecture [33]. Combined the signal flow with all functionalities affecting the response time of the system is another potential for the topic, since for most robotics systems do not provide the movement equation of each component that is involved in the integrated system [34].

Due to the current problems mentioned above, few key points we will focus on during the thesis development:

- **Joint space or Cartesian space:** path planning to move the end-effector smoothly via points in joint space or cartesian space. Computation is simple in joint space but highly depends on operations; Cartesian space generation is complex to compute but singularities with working environment could cause joint rates.
- **Robot type:** the collaborative robot will be used because it is widely used in the real industrial process. Demand for collaborative robot is increasing with the automation industries.
- **Safety considers:** for human-being, it is possible to be attacked by the cobot during the collaboration. We choose Yumi robot because it can provide more safety than other cobots, such as Universal robot arms. On the other hand, safety functions are designed by default in Yumi promise the robot will keep safe in industrial scenarios.
- **Data visualization:** All the sensors are integrated so that the performance data will be applicable in software provided by ABB. Thus, it will be a challenge to analyze the data or implement it at a numeric level.

In this paper, a digital model of Yumi is used in MATLAB. A simple pick and pack trajectory designed by kinematics calculation and nonlinear model predictive controller is used to optimal the trajectory.

3 METHODOLOGY

The thesis aims to explore a solution for trajectory optimization problem of industrial manipulator. A method for this problem is presented in this section. This method was chosen because the optimization problem in trajectory planning is non-linear problem, therefore the data that can be mixed by numerical and non-numerical data, it is complex to understand the concepts, summarize a general solution for similar academic problems, and gather in-depth insights into the trajectory optimization problem. Therefore, the data should be analyzed as a whole system.

The following content will be divided into three parts to implement the method. First section introduces the overview process and control parameters of the manipulator trajectory planning problems. Second section describe the meaning and structure of the cost function about NLMPC. Third section gives the generated nonlinear model predictive controller under constraints about NLMPC.

3.1 Overview

The topic of manipulator trajectory planning is a landmark problem in contemporary industrial manufacturing. Cartesian space and joint space are the two spatial types of trajectory planning coordinate systems, but the smoothness of the trajectory is always a critical test in the planning phase. In this section, the overview of a trajectory preparation process was obtained.

A nonlinear model predictive controller is used in MATLAB to speed up trajectory planning process and generate a collision-free trajectory for industrial manipulator from an initial point to a desired point. First, for a specific problem, some required input variables are needed as control parameters. Then, the function starts its iterations, and the cost function is used at each time step to obtain the collision-free trajectory under defined constraints. In this paper, a pick and pack operation is executed by a collaborative robot. The position of initial and desired end-effector poses are given, two approaches are obtained to find the trajectories. The traditional polynomial function is used to interpolate the motion behavior of the robot model. Two set of waypoints of the path are generated in task-space and joint space respectively. And for the situation with obstacle, we use nonlinear model predictive controller to obtain a collision-free, closed-loop trajectory generation. As shown in Figure 3.1.1, the flow chart shows how the robot execute the operation in this work.

3.2 Motion Planning

As we mentioned in Chapter 2, In terms of configuration space, the axes of motion for a robot correspond to its degree of freedom. The robot's configuration is presented as a vector, and the configuration to represent a pose with orientation of a robot in a 3-dimensional environment is a vector $q = (x, y, z, \theta_r, \theta_p, \theta_y)$ which x, y, z represent the position and $\theta_r, \theta_p, \theta_y$ represent the orientation, respectively. In all these cases, when a trajectory is required from point A to B, which means a smooth motion is required from an initial configuration vector to a desired configuration vector.

For generating the trajectory in task space environment, initial position of the end-effector and final position of the task are provided in the robot's base frame under configuration space. Then the trajectories times is defined based on moving distance of the end-effector and desired tool speed. It is similar with industrial robot programming, several waypoints are needed to allow the robot move smoothly along the path.

For joint space trajectory generation, an inverse kinematics is used to obtain the initial and desired joint configurations. However, the inverse kinematics does not have one unique solution for the end-effector position. Therefore, for revolute joints with infinite range, the solution may be overmuch which means a trajectory with furthest distance can be included. To avoid that situation, those joints are wrapped to the interval $[-\pi, \pi]$. According to the number of constraints, different order polynomial function can be selected. Cubic splines are widely used to interpolate the trajectory for both position and rotation, but less satisfying in boundary conditions; Quartic spline can associate a better performance with fourth order polynomials, but more coefficients are needed to determine. Considered the complex of the execution, a smooth trajectory is generated by a third-order polynomial through multiple waypoints and presented using B-spline through control points.

If we keep using only the numerical method to generate the path via several points, then the trajectory has to be divided into multi segments. The robot is required begins at rest at point A and ends at B_i (*i is the - th segment of the whole trajectory*), but travels through (or close to) the middle configuration without stopping. To attain continuous velocity, the ability of the robot to reach each middle configuration will be sacrificed. And the problem is over constrained. As the consequence, quantic polynomials are needed to generate the trajectory which can meet those boundary conditions on position, velocity and acceleration at initial and end points at each segment.

On the other hand, each axis (or DoF) of the robot has to move different distance on each motion segment, and it is possible that when it is moving with its maximum speed in a minimum time before it can achieve its desired position. In that case, the configuration at

each segment is needed to define before the robot motion, the distance that each axis needs to move for the segments and the maximum reachable velocity on that axis are important to plan the trajectory on each segment. Furthermore, the computing of quantic polynomials at each time step is complex, the complexity of the computation will directly affect the time consumption. Due to the above considerations, the concept of nonlinear model predictive controller will be introduced in the next section to reach the collision avoidance feature for this work.

3.3 Nonlinear Model Predictive Control

In this section, the concept of trajectory optimization is introduced by applying nonlinear model predictive controller. The basic scheme of a NLMPCC control loop is present in FIGURE 3.3.1. As seen below, it is important to evaluate the system states based on thee output measurements. The basic NLMPCC structure shows that an optimal signal is computed under a minimum given cost function, execute the previous part of the optimal input until new estimates are ready, new estimates is executed in the system, and repeat the process. One of the advantages of NLMPCC is high stability. The control problem addressed in this paper is formulated as a continuous-time close-loop optimal problem with finite horizon, constrained by the continuous-time nonlinear model.

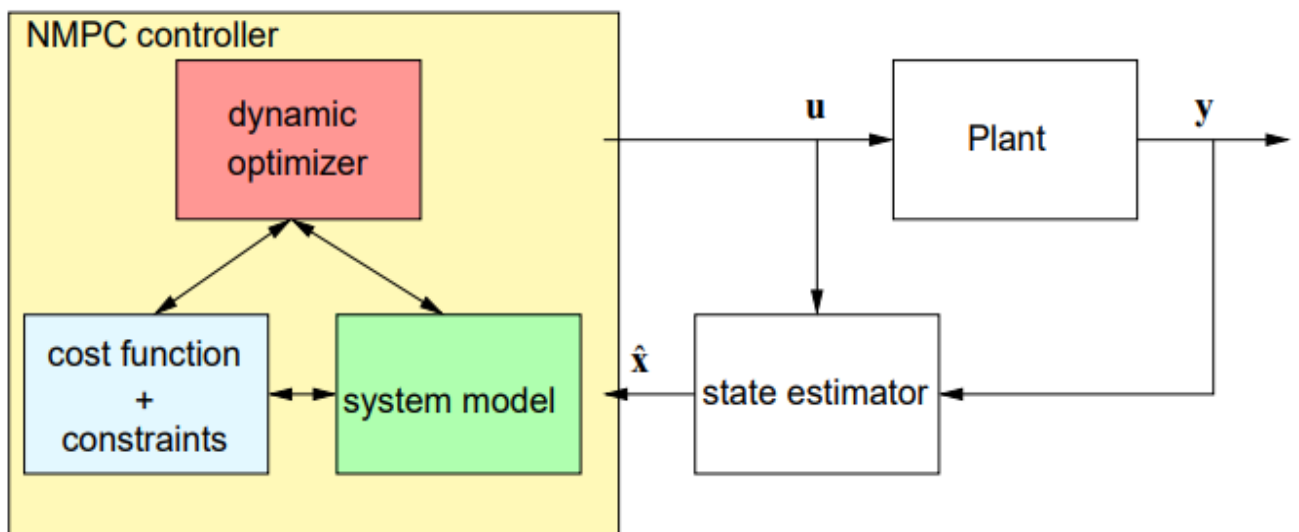


Figure 3.3.1 Basic NLMPCC Control Loop [35]

From the previous section, one of the challenges is to improve the trajectory in joint space when it starts moving from initial position. A nonlinear model predictive controller is used to compute the optimal control by applying a nonlinear prediction model, a nonlinear cost function, and nonlinear constraints.

The trajectory planning problem is a nonlinear, time-varying dynamic problem. To control the position output of the end-effector, a closed-loop can be used to minimize the position error and provide the reliable input for next iteration. The optimization process involves the desired of the control trajectory execute from the initial state to the final state in minimum time T without any collision. First, the input of the model is defined as joint accelerations $u = \ddot{q}$ where the joint positions are denoted by q and their velocities are denoted by \dot{q} , then the states of the model are $x = [q, \dot{q}]$. The dynamics of the model is given by

$$\dot{x} = \begin{bmatrix} 0 & I_N \\ 0 & 0 \end{bmatrix} \cdot x + \begin{bmatrix} 0 \\ I_N \end{bmatrix} \cdot u \quad (3.3.1)$$

Where I denotes the $N \times N$ identity matrix

N is the number of DoF of the robot

The output of the model is defined as

$$y = [I_N \quad 0] \cdot x \quad (3.3.2)$$

Therefore, the nonlinear model predictive controller (NLMPC) has $2N$ states, N outputs, and N inputs.

The cost function for the NLMPC is a custom nonlinear cost function, which is defined similarly to a quadratic tracking cost with a terminal cost to improve the stability of the NLMPC system.

$$J = \int_0^T P_{des} - \dot{p}(q(t)) Q_r (P_{des} - p(q(t))) + \dot{u}(t) Q_u u(t) dt + P_{des} - \dot{p}(q(T)) Q_t (P_{des} - p(q(T))) + \dot{q}(T) Q_v \dot{q}(T) \quad (3.3.31)$$

Where the matrices Q_r - the running weight on desired end-effector pose,

Q_u - the terminal cost weight on desired end-effector pose,

Q_t - input cost weight on joint accelerations,

Q_v - terminal joint velocity cost weight on joint velocities,

P_{des} - the desired end-effector pose

$P(q(t))$ is the difference transform between the joint positions $q(t)$ and end-effector.

Another feature of NLMPC is constraints predefined, isolating those limits is important since they can lead to undesired consequences. In this paper, the constraint is to keep a safe distance between joints of robot and obstacles, so called collision. To avoid collision when executing the trajectory, one inequality function is introduced in the controller with the cost function. The controller is need to satisfy the inequality constraints as bellow

$$d_{diff} \geq d_{safe} \quad (3.3.4)$$

Where, d_{diff} - the distance between the robot body to the obstacle.

In this inequality constraint function, the base and end-effector of the robot bodies are not included. The obstacle should be marked if it is more than one in the task.

The Jacobians functions are provided for the state function, output, custom cost and inequality constraints for the predictive model to improve the efficiency of the simulation. As we mentioned above, constraints in NLMPC is an important factor to the final optimization. Constraints are typically classified as hard or soft depending on the actual control system. Hard constraints, such as actuator saturation constraints are not to be violated, whereas soft constraints, such as additional constraints for energy efficiency, are nor strictly required to be met in this case. As a result, one way to guarantee the feasibility of iterative optimization of NLMPC is to involve soft constraint relaxation factors to improve the feasibility of NLMPC. That topic is not the major focus for this thesis, but it is necessary to mention for the future scope.

4. SIMULATION STUDIES

In this chapter, the proposed method is demonstrated with simulation. The robot is used in this paper is ABB Yumi dual arms robot. The simulation is programmed in MATLAB and the digital model of Yumi is considered.

4.1 Digital model selection

The selection of the robot is a crucial section for this work. Arm-type robot or manipulator is typically used in manufacture on a fixed base, in those cases a local work cell is sufficient to allow robot perform high repetitive executions. And parts are posed in an order that provide the maximum speed and precision for the robot. No human interaction is included in the workspace due to the hazardous and safety consideration that high-speed robots can meet in daily task executions. However, a new trend in industries that involving human interaction with robot to finish the task is significant in most of industries, so that collaborative robots market (or cobot) is growing in the past decades [10]. This work is presented with a digital model of ABB Yumi IRB14000. There are many cobots in the market, and Yumi is the world' first fully collaborative robot and Its high flexibility in automated manufacturing is well known. In this work, a digital model of Yumi is programmed in MATLAB due to the practical consideration. More focused on the procedure of creating the trajectory and optimization, no physical robot was involved in this work. Therefore, a simple explanation about the model is given in the following section.

Yumi collaborative robot has 7 axes on each arm can execute flexible motion and work side-by-side on the same tasks with humans. On the other hand, Yumi has a lightweight base and dual arms covered with a floating plastic casing in soft padding, which promises the safety for human workers. The front and top view of the Yumi robot are shown as below in Figure 4.1.1.

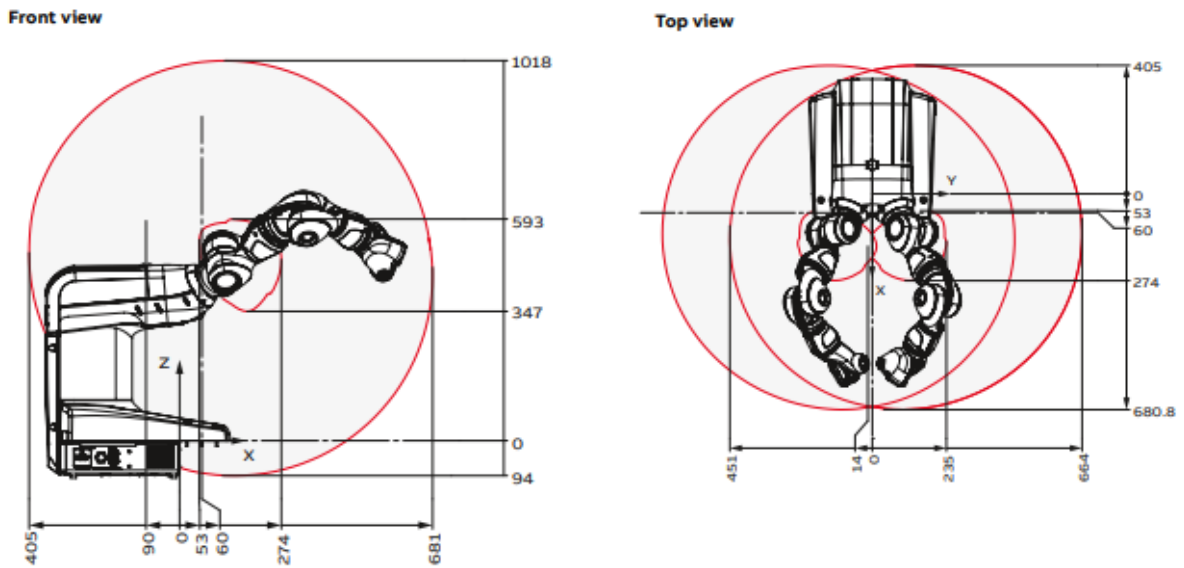


Figure 4.1.1 Working range of the Yumi Robot [37]

As can be seen from Figure 4.1.1, the reachable range of each arm can be observed, approximate motion range on each arm is 681 mm, 1018 mm, 664 mm along X, Y, Z axis respectively. For programming the robot, a handheld operator panel is used to program robot in real time. Figure 4.1.2 shows the ABB FlexPendant which uses to program, modify programs.



Figure 4.1.2 ABB FlexPendant [36]

4.2 Experiment setup

In this section, the simulation environment was introduced. In this paper a digital twin of ABB Yumi collaborative robot with 7 DOF at each arm is used. The configuration of a 7-joint robot would be its joint angles $q = (q_1, q_2, q_3, q_4, q_5, q_6, q_7)$. The digital model of ABB Yumi with a single arm in MATLAB is shown in Figure 4.2.1.

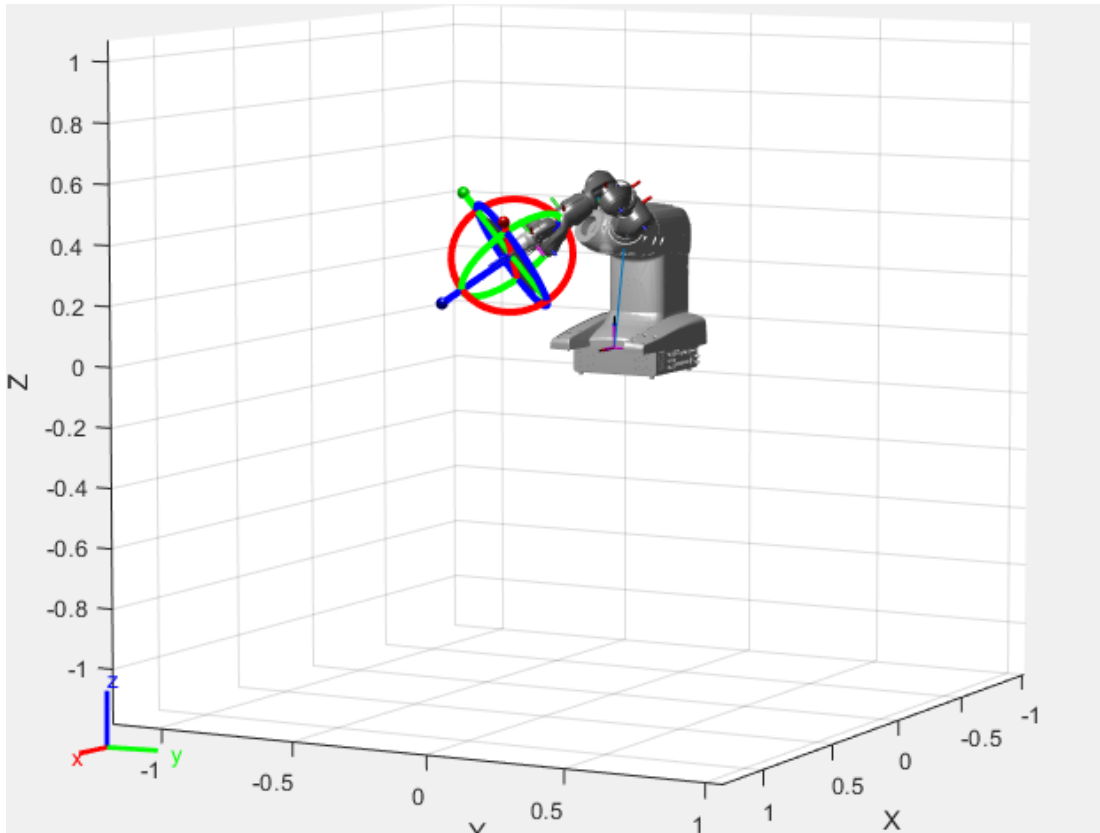


Figure 4.2.1 ABB Yumi with single arm in home configuration

To get the simulation environment closer to the actual production process, more obstacles and workspace are added into the visualization in Figure 4.2.2. As mentioned in previous chapter, the trajectory can be generated in either joint-space or Cartesian space. The trajectories both in joint and task space are shown in Figure 4.2.2. It is obvious to see from the Figure 4.2.2 that the robot shows the smooth performance from the peak to the desired position in joint space, but the motion from initial position to the middle way of the desired position has visible error. The motion in task space is smooth and regular in each time step at the beginning, however, the end-effector shows a linear trajectory when is approaching to the desired position. In this case, only following a trajectory that is generated by traditional polynomial is not enough either in practical production request or academic research.

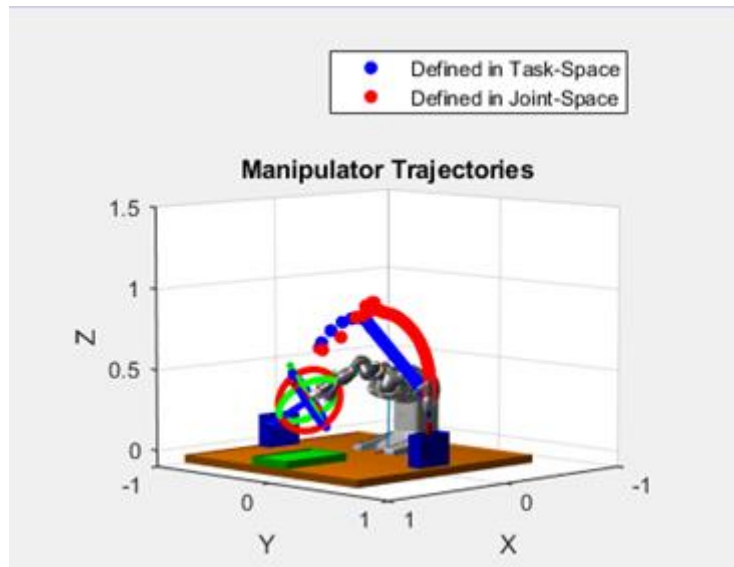


Figure 4.2.2 Trajectories in task space and joint space

From the previous chapter, it can be seen that the polynomial interpolation as a numerical analysis can roughly reach the desired position. To get closer to the operation in real for the simulation, an obstacle is added to the operation environment as shown below in Figure 4.2.3.

So far, the environment of the simulation is constructed separately by including the workstation which is a brown rectangular, two grey rectangular stand placing belts as target places, three parts are needed to be moved placed in front of the robot and two spheres represent two fixed obstacles. These objects are made by combing collision primitives and placed in the appropriate locations. The environment is used for collision checking in the trajectory optimization procedure. When the environment and robot model are ready, the initial and final pose of the end effector were defined according to the home configuration and location of the parts, respectively. All of the above mentioned can be seen in the following Figure.

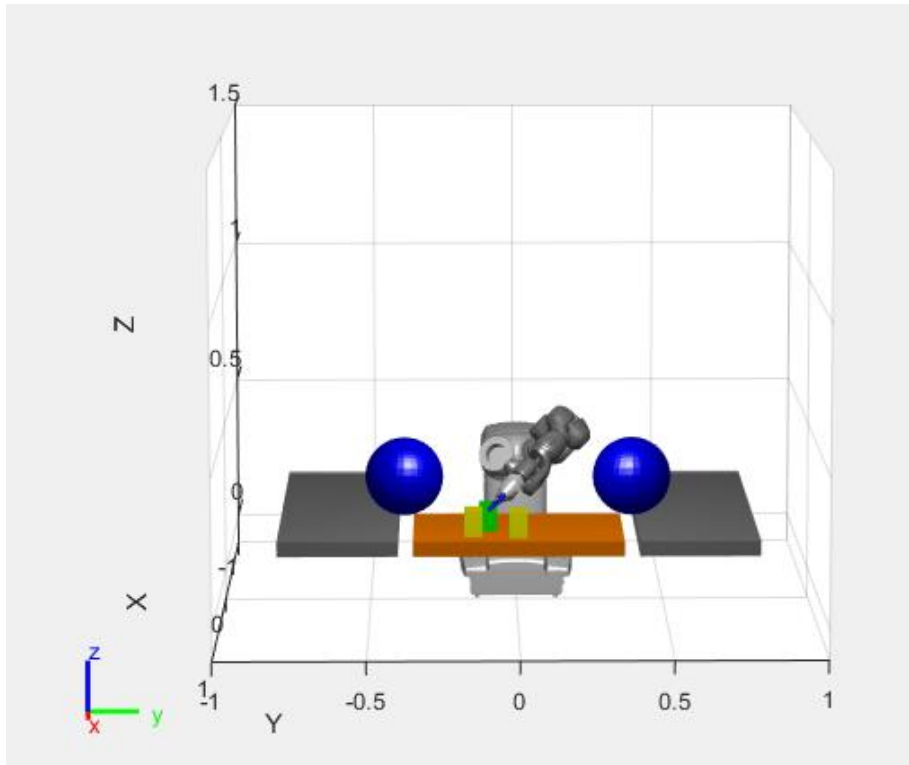


Figure 4.2.3 Initial position of the robot model

The initial configuration of the model shows above can be written as $currentRobotConfig = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$, the configuration consists 9 position instead of 7, that is because the end-effector of the robot has two grippers. The relationship between 7 axes on a single arm of ABB Yumi are shown as below in Table 4.2

Table 4.2 Yumi movement parameters [36]

| Axis | Motion Type | Motion Range |
|--------|-------------|--------------------|
| Axis 1 | Rotation | -168.5° to +168.5° |
| Axis 2 | Bend | -143.5° to +43.5° |
| Axis 7 | Rotation | -168.5° to +168.5° |
| Axis 3 | Bend | -123.5° to +80° |
| Axis 4 | Rotation | -290° to +290° |
| Axis 5 | Bend | -88° to +138° |
| Axis 6 | Rotation | -229° to +229° |

The workflow in this thesis is shown as below in Figure 4.2.4. When the digital twin of the robot model is imported to MATLAB, the configuration of the robot will be initialized to home configuration and start executing the task. When the detected part number is greater than four or equal to four which means all the defined parts are picked and placed, the task ends.

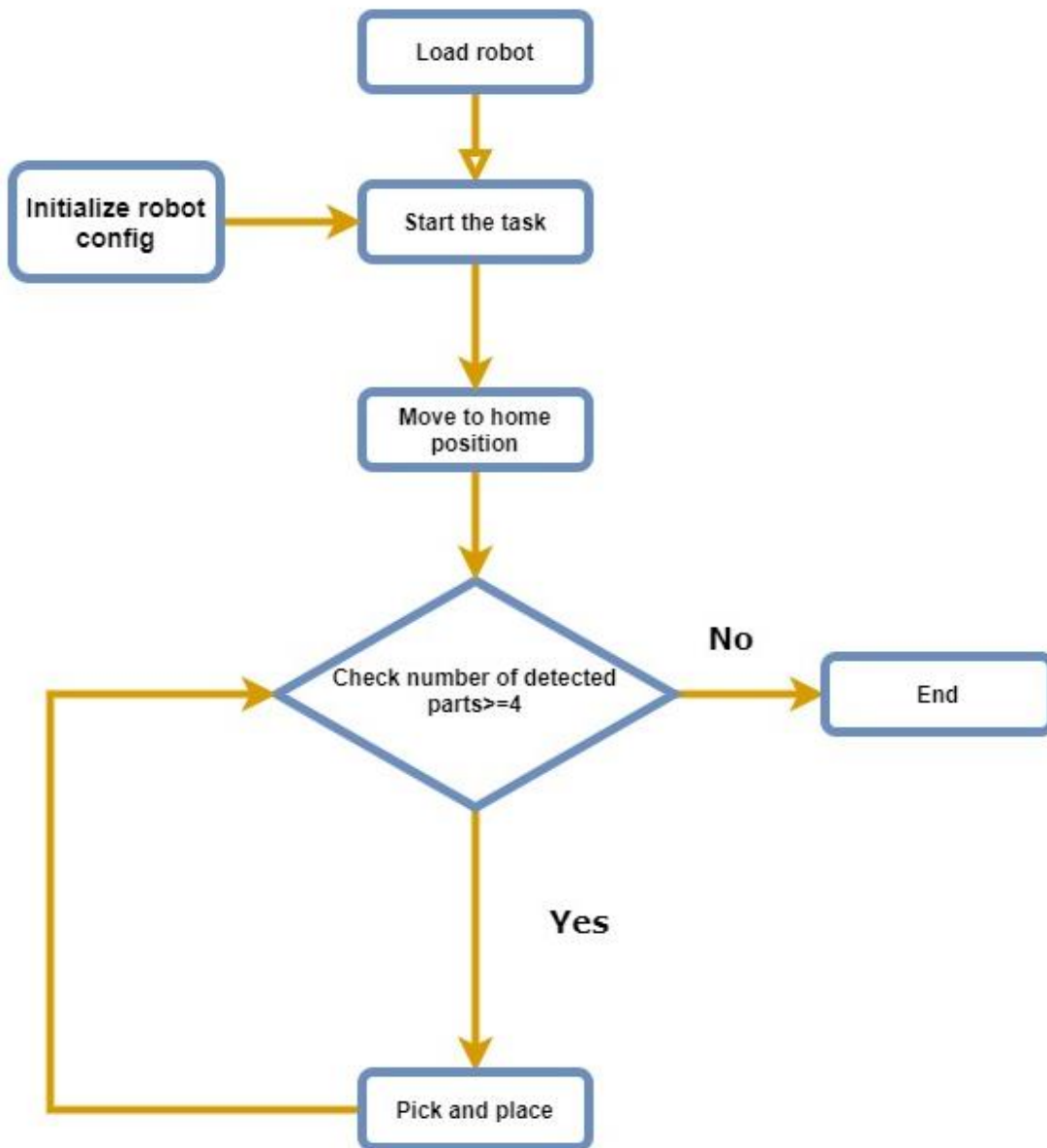


Figure 4.2.4 Flowchart of applied methodology

In this thesis, a logic control toolbox was used to execute pick and place process. The logic diagram is shown as below in Figure 4.2.5. First, a command is used to compute the task-space grasping pose that is needed for the end effector to pick up the part. The pose of this grasping motion is determined by the pose of the part to be picked up, in this thesis the target parts are known. However, the method for reaching the grasp pose can be replaced by more advanced methods, such as machine vision based on object poses. The intermediate step is approaching and moving towards the parts. Secondly, a simulated gripper is used to pick up parts. The picked part is added as a collision to the robot rigid

body tree so that the obstacle avoidance status can execute to avoid the collision of the picked part. When the gripper is deactivated, the part is removed from the collision meshes of the rigid body tree. Then the model travel to a retracted position and departs from the other parts and part picked successfully information display on the command window.

Placing the part is using the similar logic with the picking process. The placing positions are defined as almost same with the belt location. When a part is placed, the placing pose is updated so that the next part can be placed elsewhere. The message of part has been placed then show on the command window.

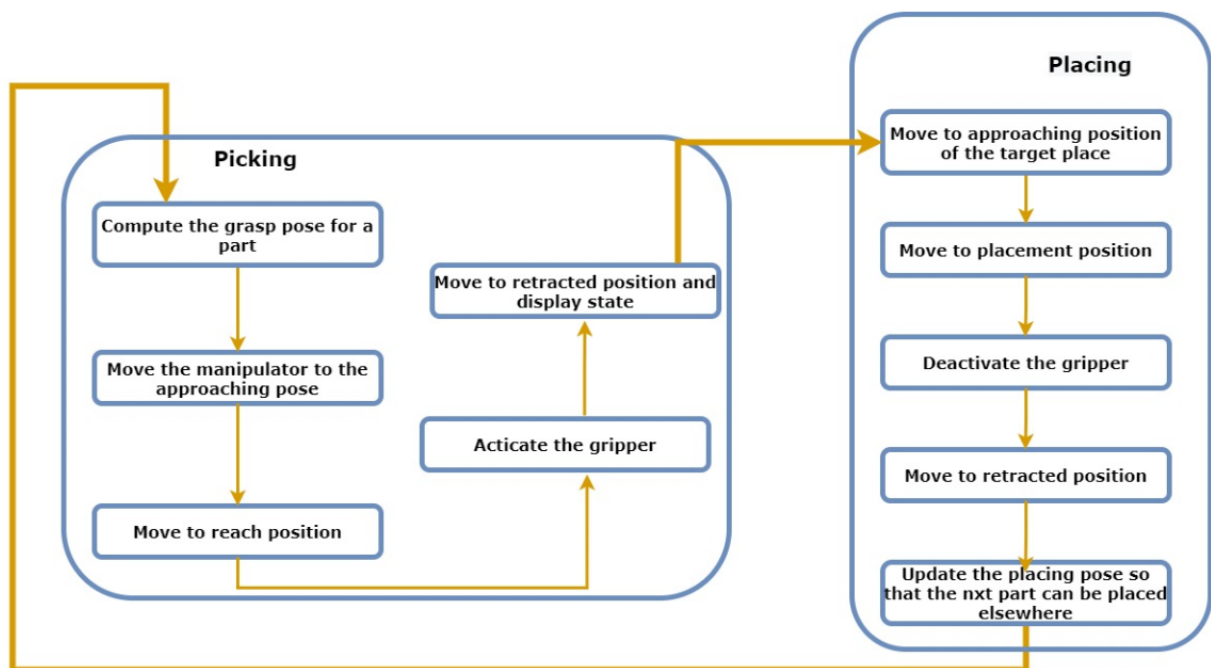


Figure 4.2.5 Logic control of the pick and place process

The difficulty of trajectory planning is computing high order interpolation polynomial functions. It is common to divide the whole trajectory into multi-segments so that lower order polynomial is enough to reach the desired position. However, less flexibility and more preparation are tricky to deal with. In this thesis, NLMPC is used to move the robot to move between various specified poses without collision. In this thesis, all the obstacles are fixed, which means no moving collision is defined to avoid. Each iteration calculates the position, velocity, and acceleration of the robot to avoid collision as they travel towards the desired position. The obstacles are given as spheres to make sure the accurate approximation of the constraint Jacobian in the definition of the nonlinear model predictive control functions. The Jacobian is used to for creating a linear approximation of the known distance and the closet points concept. The joint space motion model is used as a helper function in this work to simulate the motion of the robot under computed torque control while tracking the reference

trajectory with the object. The visualization then is updated to show the current status.

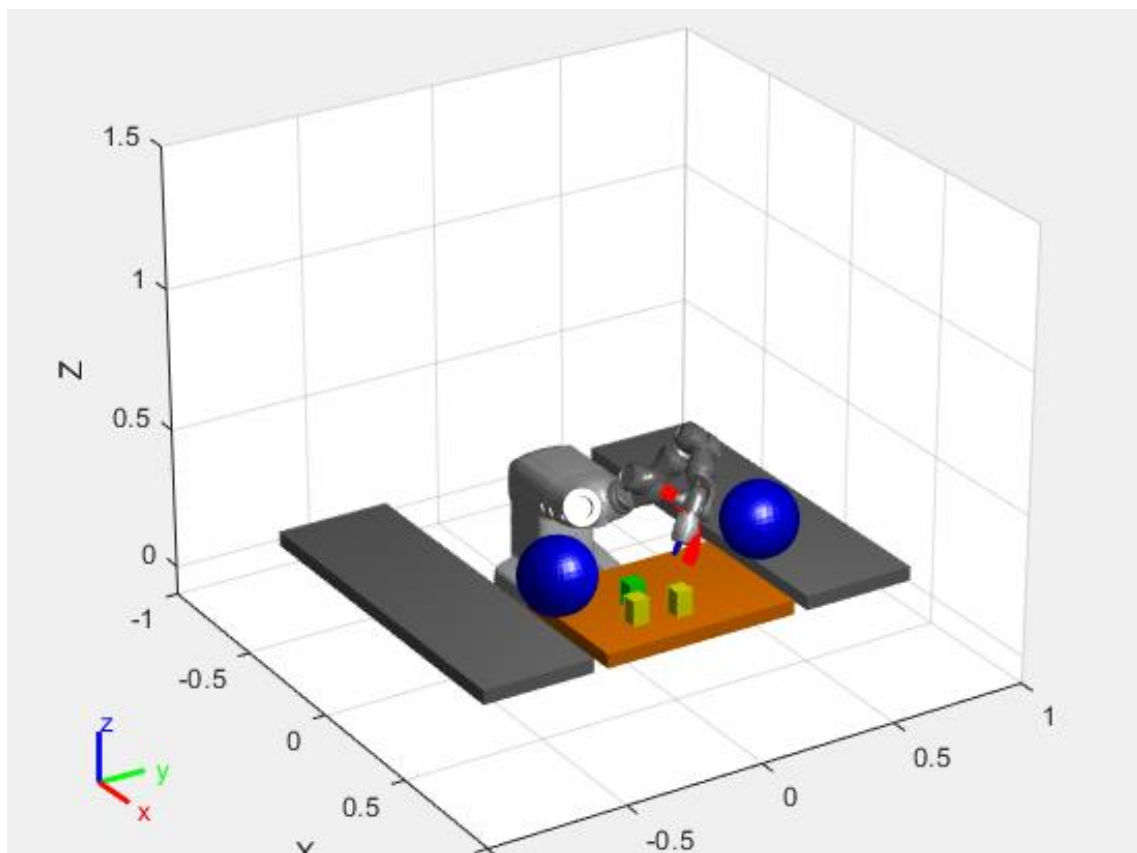


Figure 4.2.6 End-effector move to approaching position.

As shown above in Figure 4.2.6, for each time step, the NLMPC calculate the control process and execute the motion with the visible trajectory mark. In the thesis, three parts are required to pick and place on two belts separately. From the Figure which is shown below in Figure 4.2.7 it is obvious that the gripper was separate from the robot, it is relevant to mention that one thing when testing the method for trajectory optimization with a digital model is to check the self-collision, and the mechanism features about the robot are needed to define as hard constraints.

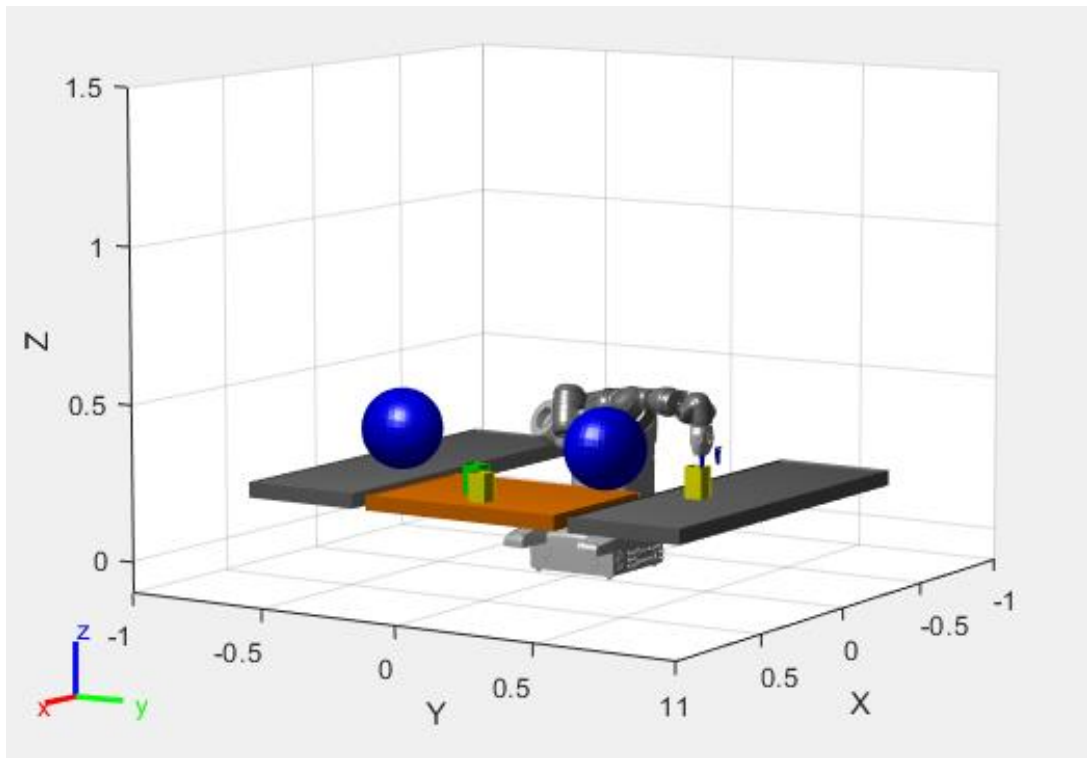


Figure 4.2.7 End-effector move to placing position

When three parts are placed on the desired position on the belts, respectively. The state flow that is used to track the robot trajectory then detects all the parts are placed, the intermediate step is to move the robot arm to home configuration. The task is executed successfully. The final pose of the robot is shown as below in Figure 4.2.8. As can be seen, two parts are placed on the right belt and one part is placed on the left belt. And robot is moving from the away position to home position.

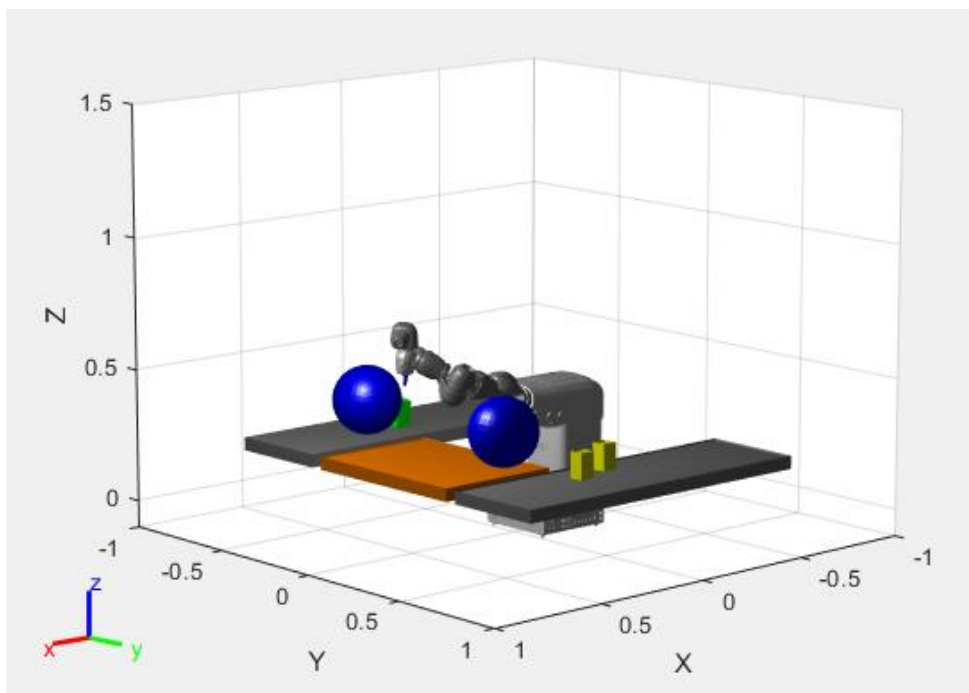
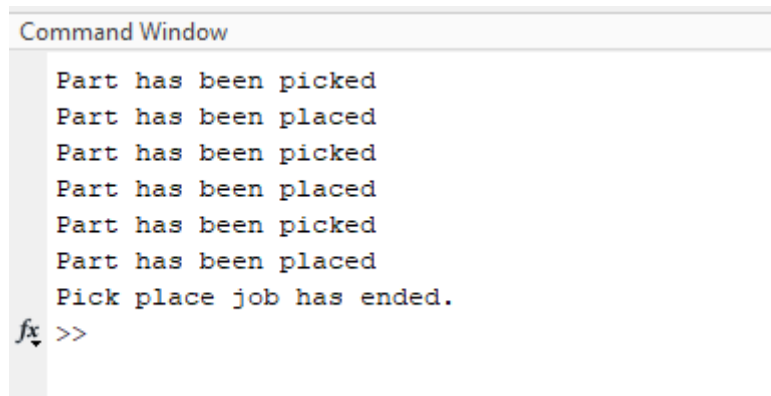


Figure 4.2.8 Final position of the robot model

The state flow is used in MATLAB to support the task execution by applying command functions. When the common finished execution, the state flow send a trigger event to start the flow chart and the next step will be proceed to continue the task execution. When the execution is finished, the notification is showed on the command window, the message is shown as below in Figure 4.2.9.



```
Command Window
Part has been picked
Part has been placed
Part has been picked
Part has been placed
Part has been picked
Part has been placed
Pick place job has ended.
fx >>
```

Figure 4.2.9 Display message on the command window in MATLAB

The common approach among the existing methods to track and analyze the robot motion using MATLAB script and Simulink to simulate the task. For example, the first segment of the trajectory which was introduced in the Chapter 3. The position tracking during the trajectory execution is shown as below in Figure 4.2.10. In this example, a PD controller is used to control the velocity torque. Desired configuration, desired joint velocity, and desired joint acceleration are inputs of the controller, output of the controller is applied torque on each joint. As can be seen, qdOut (18) and qdOut (17) are represented left and right finger of the gripper, respectively. The gripper is controlled by the command logic on the gripper of the robot.

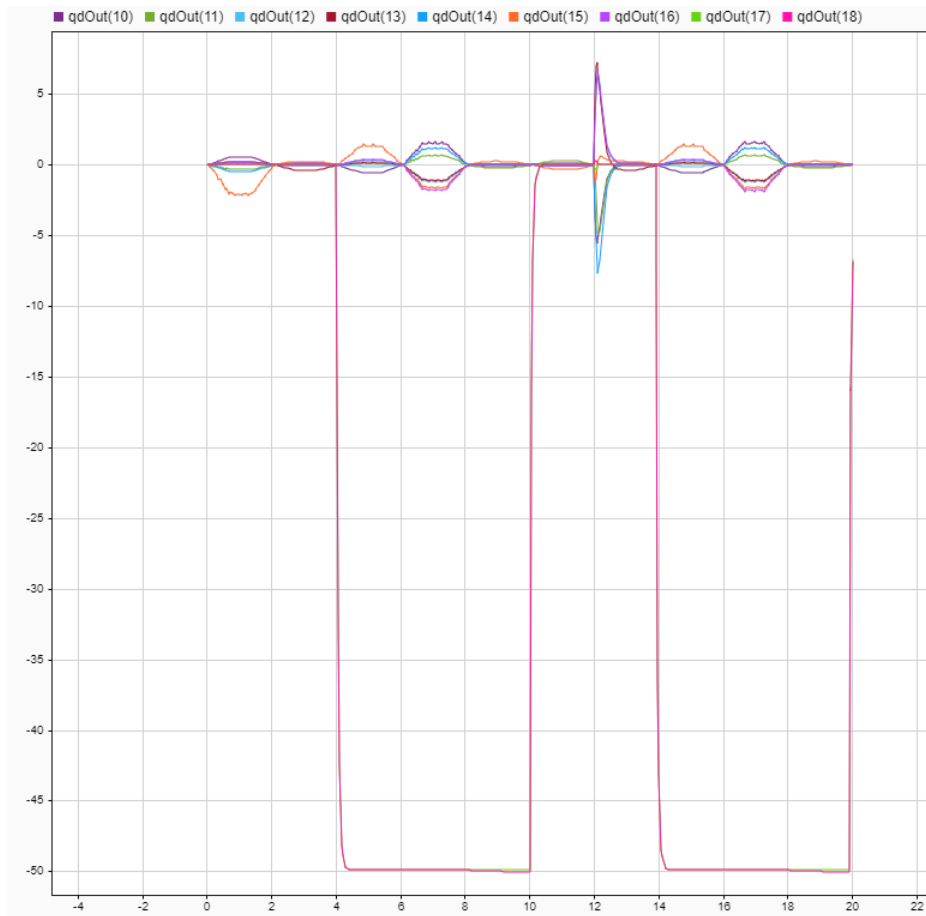


Figure 4.2.10 Position change of robot motion with PID controller

From qdOut (10) to qdOut (16) provide how the joints move during the task execution. Whereas the motion state of the robot is switched based on reading the configuration sequence from the trajectory scheduler. Therefore, the trigger to move the robot and open the gripper is to read the specific value from configuration sequence matrix. In that case, the configuration sequence should be predefined precisely for the complete trajectory. As an example, the trigger to start motion state can be shown as below in Figure 4.2.11. To check if the gripper joints have reached their target, the difference between the current gripper pose and the gripper target pose is calculated. If the difference is smaller than a defined threshold, then the robot has reached the target position.

```

switch motionState
case 1
    % Grasp: Move from approach to grasp & open the gripper
    wpts = configSequence(:,2:3);
    closeGripper = false;
case 2
    % Close the gripper
    wpts = [configSequence(:,3) configSequence(:,3)];
    closeGripper = true;
case 3
    % Depart grasp
    wpts = configSequence(:,3:4);
    closeGripper = true;
case 4
    % Move to place approach
    wpts = configSequence(:,4:5);
    closeGripper = true;
case 5
    % Move to place
    wpts = configSequence(:,5:6);
    closeGripper = true;
case 6
    % Open the gripper
    wpts = [configSequence(:,6) configSequence(:,6)];
    closeGripper = false;
case 7
    % Depart place
    wpts = configSequence(:,6:7);
    closeGripper = false;
case 8
    % Cycle to grasp approach
    wpts = configSequence(:,[7 2]);
    closeGripper = false;
end

```

Figure 4.2.11 Motion State of the robot without NLMPC

Another part of the tracking process is to check if the arm states match the target waypoints. And waypoints are predefined so that a requirement of applying this method is to define the waypoints manually. This is increasing the complexity of designing the robot trajectory, the dimension of the configuration sequence is required to create from position in 3D environment and convert to matrix expression.

On the contrary, using NLMPC to generate the collision-free trajectory between an initial configuration given by home configuration on each joint and a final target orientation in task-space under the constraints. The function used in this thesis simulates the dynamic tracking process of the robot to those reference trajectories as modelled based on the digital model of the robot in closed-loop control, and visualization then is updated with rigid body three.

The NLMPC function is defined, the inputs of the model are the joint accelerations $u = \ddot{q}$. The

prediction model of a NLMPC contains a state function to predict how the system state develop over time and an output function which is used to calculate the output of the system with regard to state and input variables. In this thesis, a continuous-time prediction state function is used, because a nonlinear MPC controller is a discrete-time controller, in this case the state function is a continuous-time, the controller inherently discretizes the model using the trapezoidal rule. This method is capable of handling moderately stiff models, and the prediction accuracy is predicated based on the controller sample time. For example, a long sample time could result in an inaccurate prediction. The NLMPC function in this thesis is used in MATLAB shown as below in Figure 4.2.12. The Jacobian function on output can be seen on the Figure 4.2.12 as well, it is common approach to define an analytical Jacobian for the state function to improve computational efficiency. If the Jacobian is missing or not clarified, the controller will compute one based on numerical perturbation in MATLAB. The output function of the NLMPC is relevant to the states and inputs at the present control interval to the outputs.

```
function dxdt = nlmpeModel(x,u)
    dxdt = zeros(size(x));
    dxdt(1:numJoints) = x(numJoints+1:end);
    dxdt(numJoints+1:end) = u;
end

function [C, D] = nlmpeJacobianOutputModel(x,u)
    C = zeros(numJoints, numJoints * 2);
    C(1:numJoints, 1:numJoints) = eye(numJoints);
    D = zeros(numJoints, numJoints);
end
```

Figure 4.2.12 NLMPC function and Jacobian output model

The NLMPC allow for the use of generic custom cost functions. One typical method to optimize the control actions is to minimize the cost function across the prediction horizon. Because the cost function value must be a scalar, the cost function is required to compute at each prediction horizon step and the results are added together. The cost function and its Jacobian function is shown as below in Figure 4.2.13(a-b). The Jacobian function is used to improve the computational efficiency of the cost function.

```

function cost = nlmpcCostFunction(X,U,e,data, poseFinal, robotL,
endEffector, Qr, Qt, Qu, Qv)
    % Running Cost
    costRunning = 0;
    for i= 2:p+1
        jointTemp = X(i,1:numJoints);
        taskTemp = getTransform(robotL, jointTemp', endEffector);
        anglesTemp = rotm2eul(taskTemp(1:3,1:3), 'XYZ');
        poseTemp = [taskTemp(1:3,4);anglesTemp'];
        diffRunning = [poseFinal(1:3)-poseTemp(1:3);
angdiff(poseTemp(4:6),poseFinal(4:6))];
        costRunningTemp = diffRunning' * Qr * diffRunning;
        costRunning = costRunning + costRunningTemp + U(i,:)*Qu*U(i,:)';
    end

    % Terminal cost
    costTerminal = diffRunning'* Qt * diffRunning +
X(p+1,numJoints+1:end)*Qv*X(p+1,numJoints+1:end)';

    % Total Cost
    cost = costRunning + costTerminal;
end

```

Figure 4.2.13(a) Cost function of NLMPC

Eventually, the visualization shows the robot in the working environment when it moves parts to the desired positions successfully. And the robot avoids two fixed obstacles in the working area, three parts are placed either on the left belt or right belt depends on the type of the parts. The robot carries on working until all of the parts have been placed.

```

function [G,Gmv,Ge] = nlmprJacobianCost(X,U,e,data, poseFinal, robotL,
endEffector, Qr, Qt, Qu, Qv)
    % Initialize Jacobians
    G = zeros(p,numJoints*2);
    Gmv = zeros(p,numJoints);
    Ge = 0;

    % Update G
    for i=1:p
        jointTemp = X(i+1,1:numJoints);
        taskTemp = getTransform(robot, jointTemp', endEffector);
        anglesTemp = rotm2eul(taskTemp(1:3,1:3), 'XYZ');
        poseTemp = [taskTemp(1:3,4);anglesTemp'];
        diffRunning = [poseFinal(1:3)-poseTemp(1:3);
angdiff(poseTemp(4:6),poseFinal(4:6))];

        % From geometric to analytical robot Jacobian
        rx = anglesTemp(1);
        py = anglesTemp(2);
        B = [ 1 0 sin(py); 0 cos(rx) -cos(py)*sin(rx); 0 sin(rx)
cos(py)*cos(rx) ];

        % Robot Jacobian
        robotJacobianTemp =
geometricJacobian(robot,jointTemp',endEffector);
        robotJacobian = robotJacobianTemp;
        robotJacobian(1:3,:) = robotJacobianTemp(4:6,:);
        robotJacobian(4:6,:) = B\robotJacobianTemp(1:3,:);

        % Running cost Jacobian
        G(i,1:numJoints) = (-2 * diffRunning' * Qr * robotJacobian);
        Gmv(i,:) = 2 * U(i+1,:) * Qu;
    end

    % Terminal cost Jacobian
    G(p,1:numJoints) = G(p,1:numJoints) + (-2 * diffRunning' * Qt *
robotJacobian);
    G(p,numJoints+1:end) = 2 * X(p+1,numJoints+1:end) * Qv;

end

```

Figure 4.2.13(b) Jacobian function on cost function of NLMPC

4.3 Experiment discussion

In this experiment, parameters can be divided into two groups. One is the predefined parameter specific for the task, on the other hand, few parameters are used to control the optimization result. In this section, the result of this experiment is discussed.

In the experiment, the safety distance is defined as a safety distance away from the obstacles. This value is applied in the inequality constraint function of the NLMPC. When the safety distance is specified as 0.005, three parts can be picked and placed successfully. However, when the safety distance is increasing, to compare with the previous result, the value of safety distance was changed to 0.02. The result was shown in Figure 4.3.1 as below. Safety distance is used in one constraint function in NLMPC, which means the controller

optimizes its control moves to content all of constraints. When the safety distance is 0.02, a step that the robot model failed to compute a feasible trajectory appeared. In this experiment, when a failed step occurs, the robot model continues to calculate another trajectory to achieve the task. To test if the safety distance can affect the continuous execution of desired task. Safety distance was increased to 0.2, the result is shown in Figure 4.3.2 as below.

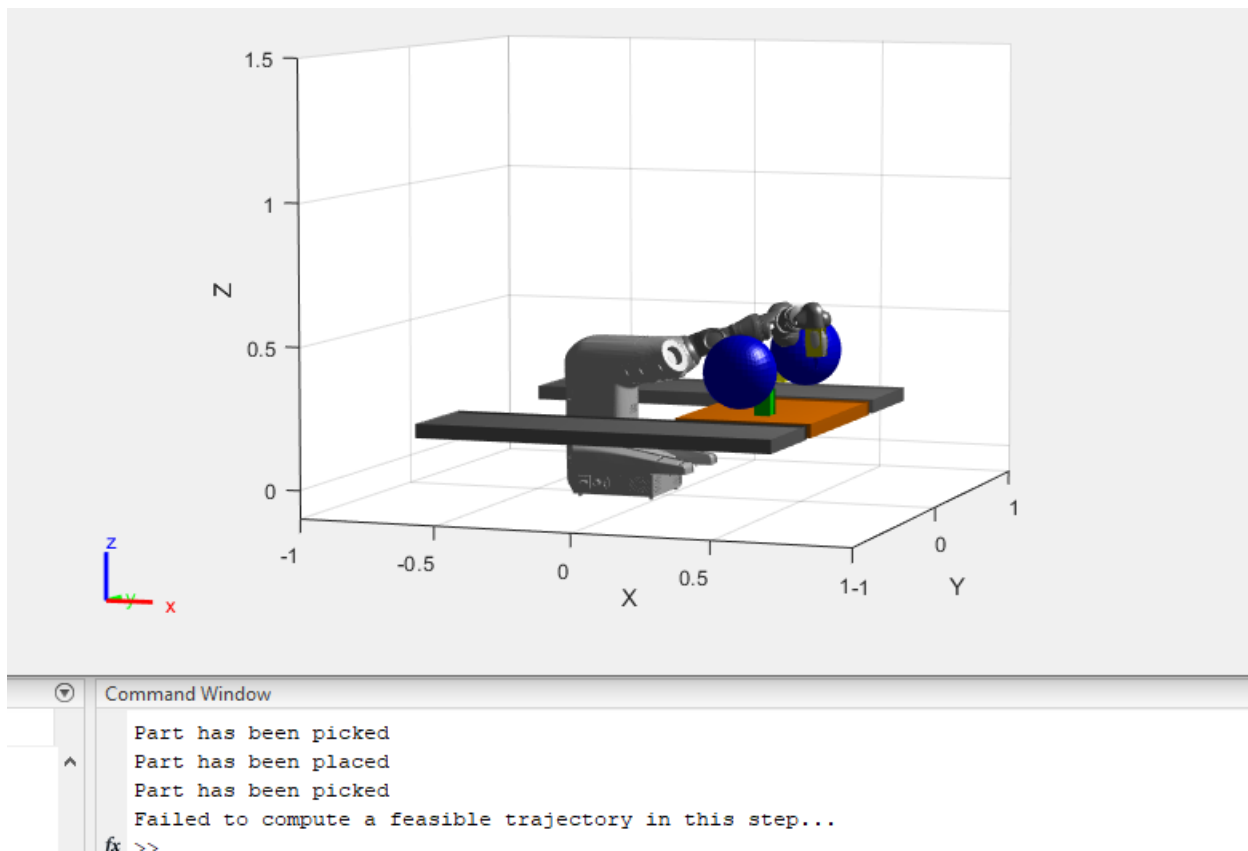


Figure 4.3.1 Trajectory generation with 0.02 safety distance

Since the safety distance was defined as each joint away from obstacles. In this experiment, two spheres are used to represent obstacles. When the safety distance is greater than a desired task configuration can be reached. Therefore, the tsk execution will be interrupted.

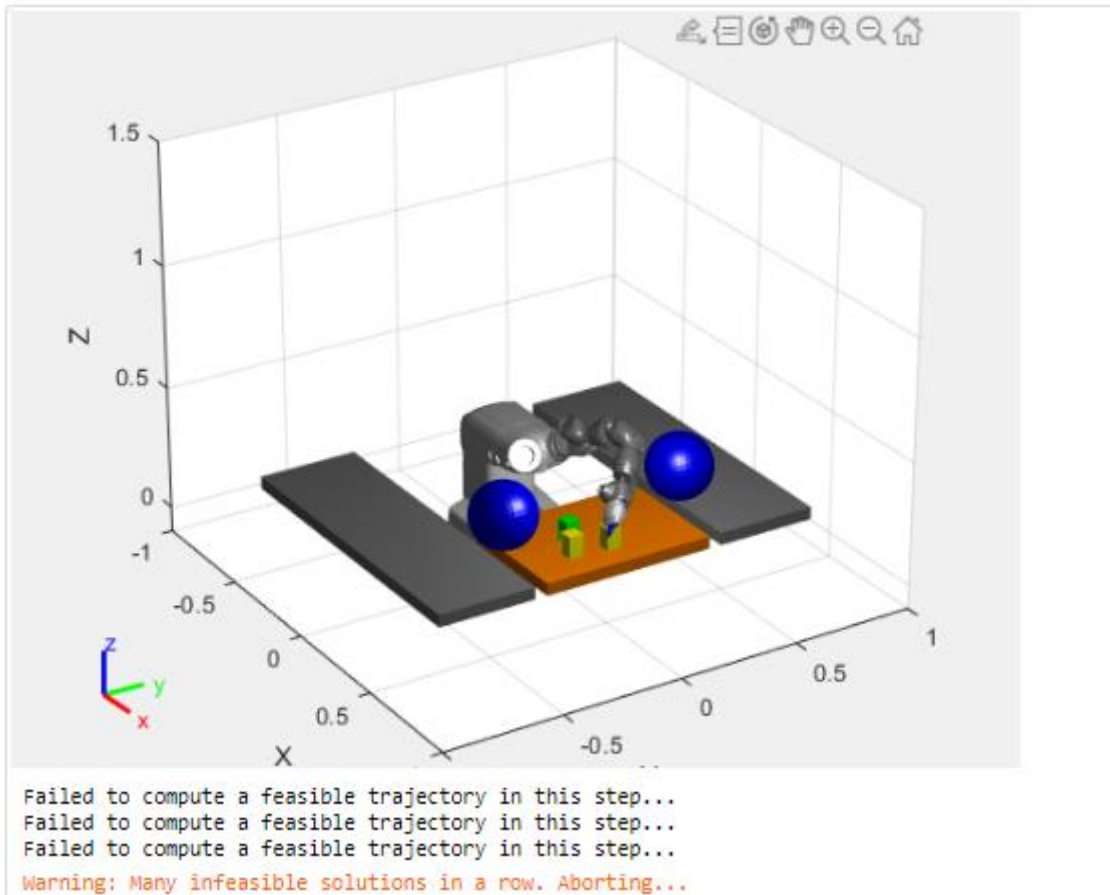


Figure 4.3.2 Trajectory generation with 0.02 safety distance

On the other hand, if the safety distance is too small, the value of safety distance was 0.00005 as shown in Figure 4.3.3 as below. One segment of the whole trajectory was printed on the Figure. As can be seen, the distance before robot model and blue obstacle was very close. There is no sensor applied for the robot model in the experiment simulation. Therefore, the position and relationship between joints and obstacles couldn't reflect the practical situation. The potential collision can cause inaccurate data for analyzing and the simulation data then lose its reference value.

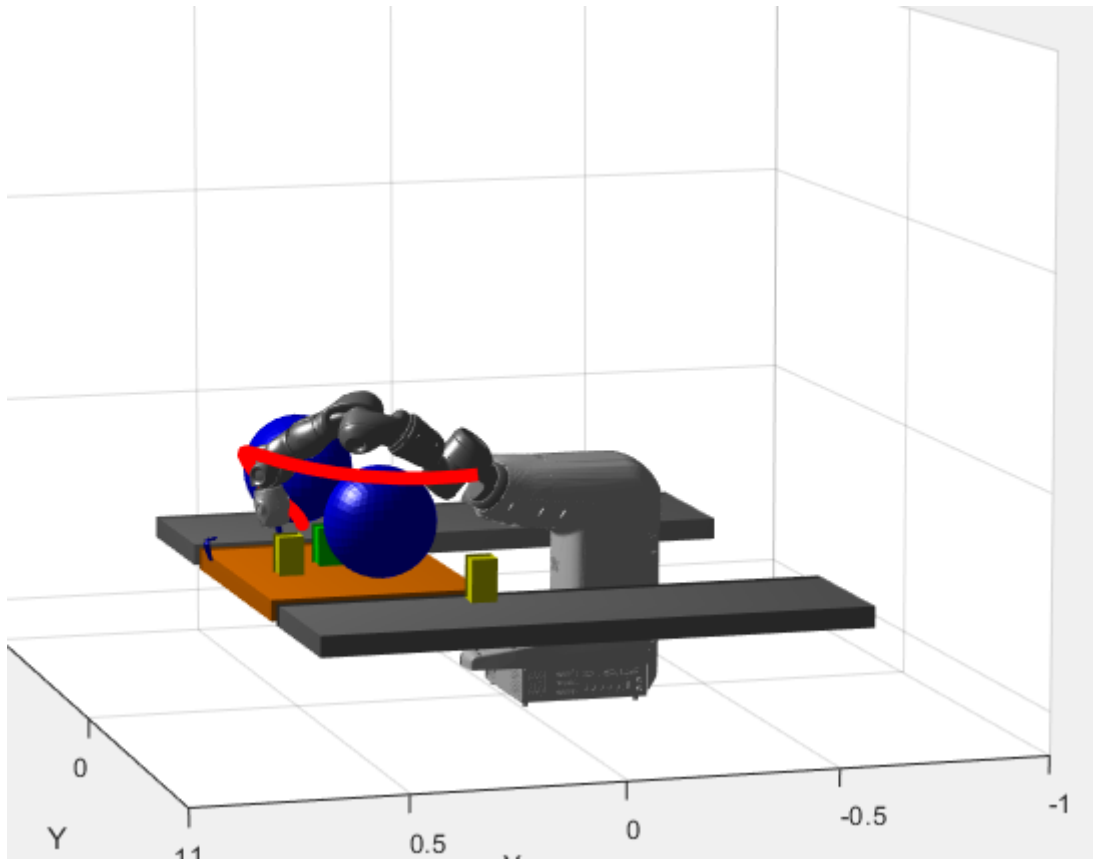


Figure 4.3.3 Trajectory generation with 0.00005 safety distance

Another important factor in this experiment is the weight on cost function. As mentioned in Chapter 3, to assess the value of these output metrics, the cost weights are introduced in NLMPC. For this experiment, the running cost weight and terminal cost weight on desired end-effector pose were set as $[10 \ 10 \ 10 \ 0 \ 0 \ 0]$ and $[10 \ 10 \ 10 \ 1 \ 1 \ 1]$. The robot model can move along with the generated trajectory with these cost weights smoothly. Later, the cost weights were adjusted to $[10 \ 10 \ 10 \ 0 \ 0 \ 0]$ and $[10 \ 10 \ 10 \ 0 \ 0 \ 0]$, the weights on orientation of the end-effector were set to zero. And the result can be seen in Figure 4.3.4 as below. It took more time for robot to pick the target part. Since there was no weight on orientation of the end-effector, so the gripper could not reach the desired position without rotation on end-effector. Furthermore, when the gripper picked the part with long time step, the trajectory from initial position to desired position was closer to the obstacle than previous one.

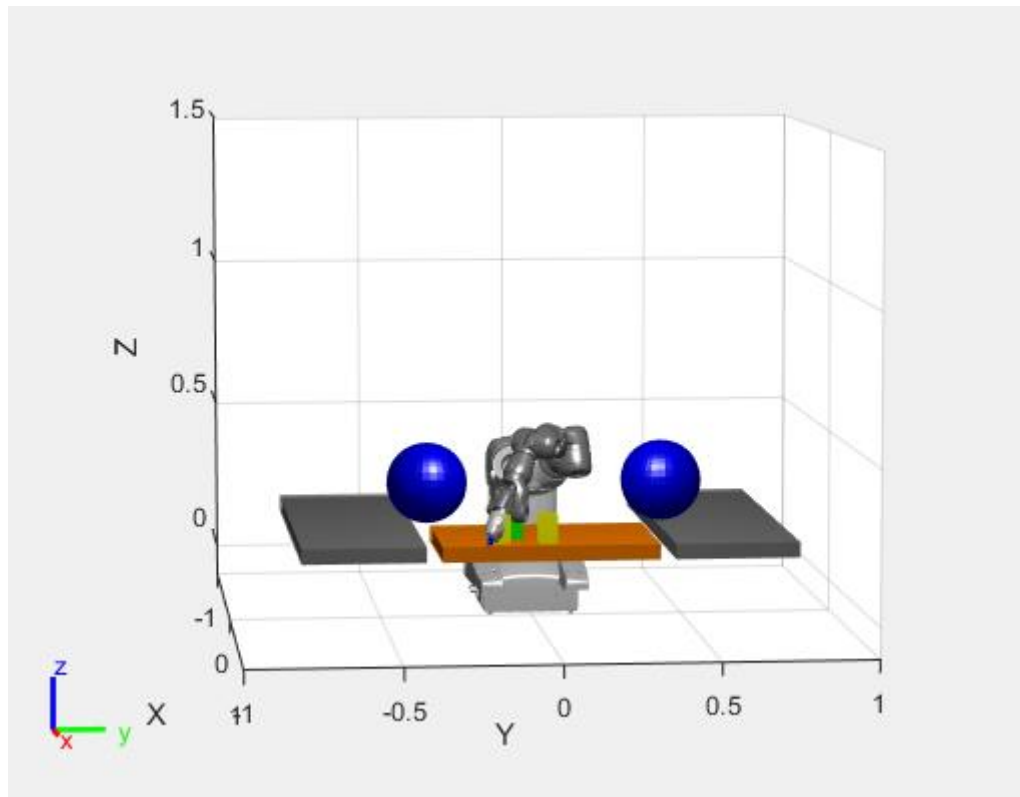


Figure 4.3.4 Trajectory generation with zero weight on orientation

On the other hand, it is a complex process to develop the cost function of NLMPC, which to determine the parameter of NLMPC by minimizing the cost function. Therefore, formulation of the cost function and customization of the cost function have introduced a broader range of NLMPC tuning techniques. Selection of the cost weight for NLMPC is a crucial factor, as can be seen from Figure 4.3.4 and Figure 4.3.5, when the cost weight was zero on orientation of end-effector, the robot spent more time to approach the target position. Accurately adjusting these weights to reduce the overall cost function while maintaining stable controller output is a difficult and continuing research challenge.

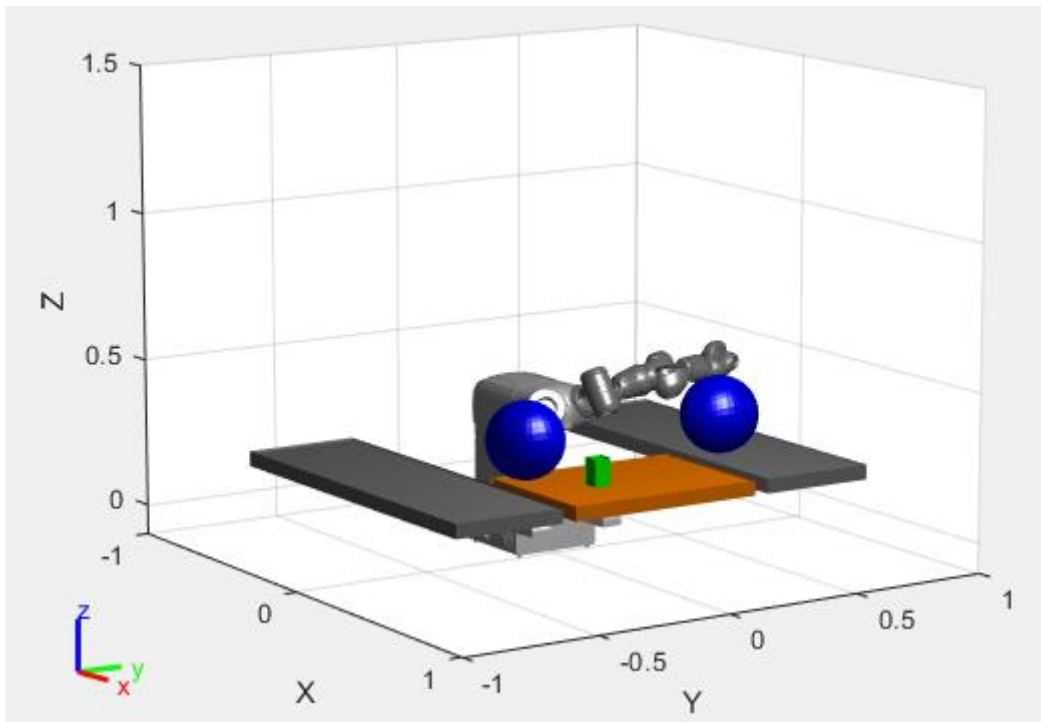


Figure 4.3.5 Trajectory generation with zero weight on orientation (potential collision)

The Figure 4.3.6 shown as below is another situation, the running cost weight on the end-effector was set to $[5 \ 5 \ 5 \ 0 \ 0 \ 0]$, and the terminal cost weight on the end-effector was set to $[5 \ 5 \ 5 \ 1 \ 1 \ 1]$. Compare with the value was used in this experiment, the weights on the orientation on the end-effector remains the same, however, the weights on the position of end-effector was decreased. It is observed clearly from the Figure 4.3.6 that the joints of the robot were almost aligned along to Y axis, in that case it is possible to cause singularity problem during the process. The selection of the cost weight to solve the optimization of cost function is to determine the efficiency of the NLMPC.

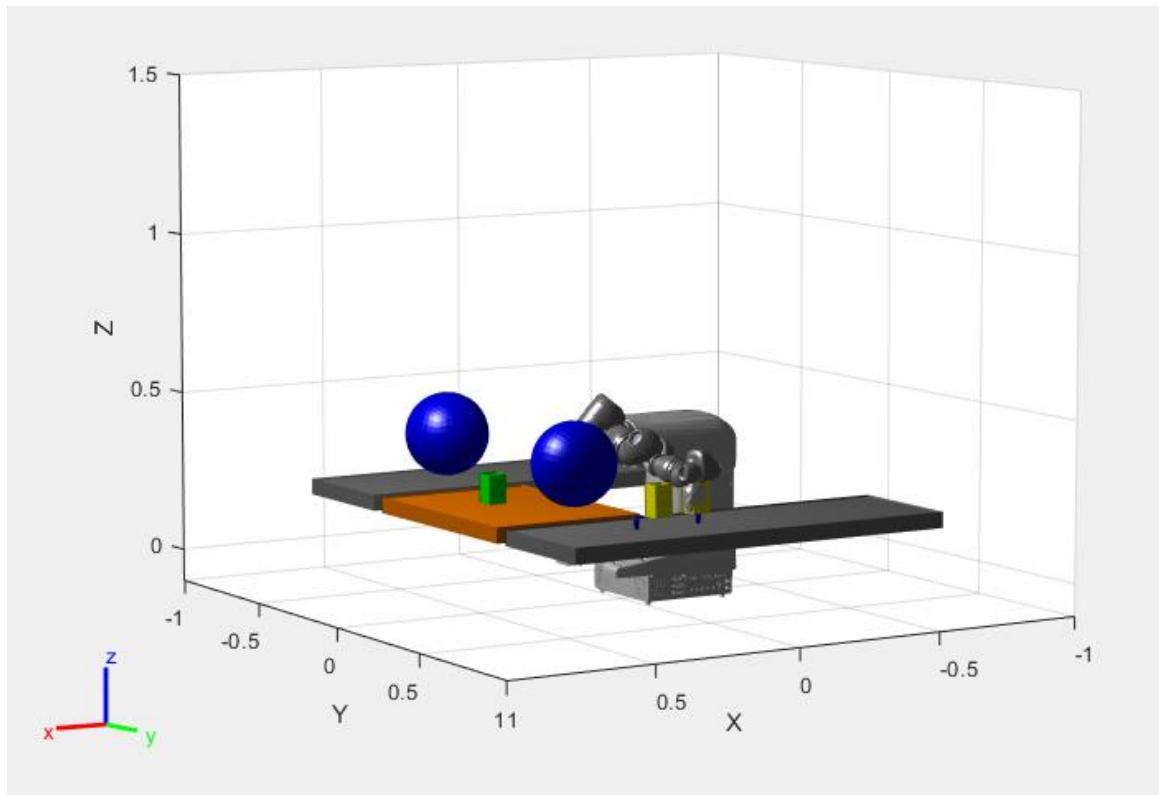


Figure 4.3.6 Trajectory generation with zero weight on orientation (potential collision)

5. CONCLUSION

A trajectory optimization method using nonlinear model predictive control on interpolation polynomial was proposed in this thesis work. A digital model of ABB Yumi robot was applied. In this case, a single arm of the model was required to reach multi-segment trajectories. A nonlinear model predictive controller was designed to achieve the desired position without collision.

The initial idea is to explore a solution due to the high complexity of industrial manipulator trajectory planning. Most of the trajectory planning is programmed in specific software offline or programming the manipulator trajectory in real-time with a customized panel. However, the modern manufacturing process requires various types of the manipulator are sharing the workspace, so high flexibility in which adjustment on trajectory planning of manipulator is needed. The digital models of industrial manipulators were used in the research to test existing methodologies. On the other hand, the current studies are major focused on improving polynomials functions or machine learning to calculate the optimal trajectory based on the desired final position. The author of this thesis tried to explore where a digital manipulator model could be easily programmed, adjusted, and valid by combining numerical and analytical solutions in the process.

The digital model of ABB Yumi robot and MATLAB was used for this work. The structure of the program was divided into two parts, the motion planning process and the control scheme. The program was tested, and it is essential to mention that NLMPC is helpful for any trajectory planning in which collision avoidance is involved. Force control and velocity control were written in NLMPC, even if the improvement is not significant from the trajectory, but the flat changes on velocity and acceleration of the model were shown from the program.

This thesis aims to explore a method that can quickly generate the trajectory from the initial position to the final position directly, or few intermediate waypoints are acceptable. One pick and place task was executed with an ABB Yumi digital model in MATLAB, NLMPC generated the trajectory under few constraints. Eventually, the program shows a reliable trajectory without collision with the obstacles, and less industrial robot programming skill is needed for adjusting the schedule.

6. DISCUSSION

6.1 Limitation

There are few limitations that the author encountered during this work are briefly discussed as below:

- The initial configuration of the robot is predefined manually, the situation that joints of the robot have self-collision before moving to the desired point. However, this problem doesn't exist in a physical robot. Therefore, it is vital to have a calibration process to check if the digital model is under collision.
- The process is only presented with a single arm because it is impossible to program the motion for two arms simultaneously. This issue will not make affection on single-arm cobots.
- Computation for trajectory planning is heavy, NLMPC shows a significant improvement in the execution speed. However, The NLMPC is integrated as a whole function Constraints or boundary conditions are compiled together, which means more adjustment is needed when changing the model.
- The NLMPC can provide a collision-free trajectory for the robot. However, the edge of the robot arm was very close to obstacles without detection from the NLMPC. More relaxation factor can be added on the soft constraints to generate the trajectory with high safety.
- The method is not implemented on the physical robot. More considerations are required to demonstrate the process in the real robot, such as connectivity between MATLAB and other software or robot controller.
- There are difficulties of nonlinear predictive control. The first is the nonlinear modeling. Constructing a high-quality object model for nonlinear prediction is obligated for optimizing predictive control performance because the controller is a designer based on the initial state of the model, and modeling errors affect controller performance. Due to the uncertainty and large-scale existence of objects in industrial production, building nonlinear modeling particularly is required for each task. On the other hand, nonlinear predictive controllers involve a nonlinear program at each time step, which is computationally demanding and an inadequate, ineffective tool. Investigating generally applicable and efficient optimization algorithms is a challenging topic.

6.2 Future improvements

The idea of this thesis is to explore a solution of trajectory optimization for industrial robots. Several further improvements from the author are described in this section.

- During the research process during this thesis work, the mechanical limitation of the physical robot is the primary consideration that the proposed method is not implemented on the physical robot. The digital model is helpful when analyzing the methodology. However, it is still a challenge about how to apply the theory in the practical manufacturing process.
- Adding stability constraints (e.g., terminal constraints or inequality constraints, shrinkage constraints) to the optimization problem theoretically solves the MPC stability problem. Researchers could investigate a more satisfactory approach in terms of engineering applications to improve the performance of NLMPC.
- Introducing new technology such as Virtual Reality (VR) into the simulation will be another potential to provide a reliable validation for the proposed method. The advantage of VR is that the cobot can interact with a human in a virtual environment with minor limitations.
- In academic experiments, less data or complex data is the major challenge to evaluate the outcome. It is valuable to extract data from practical operations and further analyze using big data techniques.
- The state of the system is not always fully measurable in engineering applications. Therefore, when studying the trajectory optimization problem and NLMPC, it could be more efficient to investigate output feedback predictive control, particularly for multivariable nonlinear systems.

SUMMARY

This thesis aims to explore a method for the trajectory optimization problem of the industrial manipulator using a nonlinear model predictive controller. The nonlinear model predictive controller was used to generate a collision-free trajectory in the simulation environment with obstacles.

The first step in putting the theoretical idea into practice was to investigate on current technologies. The author formed a scheme of solving trajectory optimization problems based on the existing studies. It was carried out from the previous researches that interpolation polynomials are essential to calculate the path from the initial position to the final pose. Controllers can be used to track the dynamic motion of the robot. But there were no such complex industrial robot models widely used in the modern industrial manufacturing process. Moreover, the existing methods were developed based on the numerical concept. More analytical technologies are the potential to solve the trajectory optimization problem that is lacking information.

A digital twin of the ABB Yumi robot with a single arm was used to travel from an initial position and pick a target part with the robot. Finally, the robot placed the picked part to the desired position, the trajectory was generated by using a NLMPC to satisfy all the constraints.

As the result, the robot provided a collision-free trajectory without computing the multi-segments of the trajectory. Eventually the proposed method helps to provide a reliable trajectory with minimum industrial programming technical knowledge.

KOKKUVÕTE

Selle lõputöö eesmärk on uurida meetodit tööstusmanipulaatori trajektoori optimeerimise probleemiks, kasutades mittelineaarset mudeli ennustavat kontrolleri. Mittelineaarset mudelit ennustavat kontrolleri kasutati takistustega simulatsioonikeskkonnas kokkupõrgeteta trajektoori genereerimiseks.

Esimene samm teoreetilise idee elluviimisel oli uurida praeguseid tehnoloogiaid. Autor moodustas olemasolevate uuringute põhjal skeemi trajektoori optimeerimise probleemide lahendamiseks. Varasemate uuringute põhjal tehti interpoleerimispolünoomid, et arvutada tee algpositsioonist lõpliku poosi. Kontrolleri saab kasutada roboti dünaamilise liikumise jälgimiseks. Kuid selliseid keerukaid tööstusrobotite mudeleid, mida tänapäevases tööstusprotsessis laialdaselt kasutatakse, ei olnud. Veelgi enam, olemasolevad meetodid töötati välja arvulise kontseptsiooni põhjal. Analüütilisemad tehnoloogiad võimaldavad lahendada trajektoori optimeerimise probleemi, millel puudub teave.

Algasendist liikumiseks ja robotiga sihtosa valimiseks kasutati ühe käega ABB Yumi roboti digitaalset kaksikut. Lõpuks paigutas robot valitud osa soovitud asendisse, trajektoori genereeriti NLMPC abil kõigi piirangute rahuldamiseks.

Selle tulemusel pakkus robot kokkupõrgeteta trajektoori ilma trajektoori mitme segmendi arvutamiseteta. Lõpuks aitab kavandatud meetod pakkuda usaldusväärset trajektoori minimaalsete tööstusliku programmeerimise tehniliste teadmistega.

LIST OF REFERENCES

- [1] P. Saraf and R. N. Ponnalagu, "Modeling and Simulation of a Point to Point Spherical Articulated Manipulator Using Optimal Control," *2021 Int. Conf. Autom. Robot. Appl. ICARA 2021*, pp. 152–156, 2021, doi: 10.1109/ICARA51699.2021.9376496.
- [2] W. Xu, J. Zhang, B. Liang, and B. Li, "Singularity analysis and avoidance for robot manipulators with nonspherical wrists," *IEEE Trans. Ind. Electron.*, vol. 63, no. 1, pp. 277–290, 2016, doi: 10.1109/TIE.2015.2464176.
- [3] W. Rackl, R. Lampariello, and G. Hirzinger, "Robot excitation trajectories for dynamic parameter estimation using optimized B-splines," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 2042–2047, 2012, doi: 10.1109/ICRA.2012.6225279.
- [4] J. J. Kim and J. J. Lee, "Trajectory optimization with particle swarm optimization for manipulator motion planning," *IEEE Trans. Ind. Informatics*, vol. 11, no. 3, pp. 620–631, 2015, doi: 10.1109/TII.2015.2416435.
- [5] C. C. Tsai, C. C. Hung, and C. F. Chang, "Trajectory planning and control of a 7-DOF robotic manipulator," *2014 Int. Conf. Adv. Robot. Intell. Syst. ARIS 2014*, pp. 78–84, 2014, doi: 10.1109/ARIS.2014.6871496.
- [6] C. Schuetz, J. Baur, J. Pfaff, T. Buschmann, and H. Ulbrich, "Evaluation of a direct optimization method for trajectory planning of a 9-DOF redundant fruit-picking manipulator," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2015-June, no. June, pp. 2660–2666, 2015, doi: 10.1109/ICRA.2015.7139558.
- [7] T. Bo Jorgensen, K. Debrabant, and N. Kruger, "Robust optimization of robotic pick and place operations for deformable objects through simulation," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2016-June, pp. 3863–3870, 2016, doi: 10.1109/ICRA.2016.7487574.
- [8] D. Pavlichenko and S. Behnke, "Efficient stochastic multicriteria arm trajectory optimization," *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 2017-Sept, pp. 4018–4025, 2017, doi: 10.1109/IROS.2017.8206256.
- [9] Y. Chen, Y. Ding, J. Jin, and D. Ceglarek, "Integration of process-oriented tolerancing and maintenance planning in design of multistation manufacturing processes," *IEEE Trans. Autom. Sci. Eng.*, vol. 3, no. 4, pp. 440–453, 2006, doi: 10.1109/TASE.2006.872105.
- [10] D. Kragic, J. Gustafson, H. Karaoguz, P. Jensfelt, and R. Krug, "Interactive, collaborative robots: Challenges and opportunities," *IJCAI Int. Jt. Conf. Artif. Intell.*, vol. 2018-July, pp. 18–25, 2018, doi: 10.24963/ijcai.2018/3.
- [11] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," *Robot. Sci. Syst.*, vol. 8, pp. 209–216, 2013, doi: 10.15607/rss.2012.viii.027.
- [12] X. Chen *et al.*, "An energy optimization based planning approach for moving bottle grasping task using a seven-DoF robotic arm," *2017 IEEE Int. Conf. Mechatronics Autom. ICMA 2017*, pp. 833–839, 2017, doi: 10.1109/ICMA.2017.8015924.
- [13] O. Wigstrom, B. Lennartson, A. Vergnano, and C. Breitholtz, "High-level scheduling of energy optimal trajectories," *IEEE Trans. Autom. Sci. Eng.*, vol. 10, no. 1, pp. 57–64, 2013, doi: 10.1109/TASE.2012.2198816.
- [14] G. Kahn *et al.*, "Active exploration using trajectory optimization for robotic grasping in the presence of occlusions," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2015-June, no. June, pp. 4783–4790, 2015, doi: 10.1109/ICRA.2015.7139864.
- [15] J. Jeevamalar and S. Ramabalan, "Optimal trajectory planning for autonomous robots - A review," *IEEE-International Conf. Adv. Eng. Sci. Manag. ICAESM-2012*, pp. 269–275, 2012.
- [16] M. Li, H. Wu, and H. Handroos, "Stiffness-maximum trajectory planning of a hybrid kinematic-redundant robot machine," *IECON Proc. (Industrial Electron. Conf.)*, pp. 283–288, 2011, doi: 10.1109/IECON.2011.6119325.
- [17] M. Keshmiri and W. F. Xie, "Image-based visual servoing using an optimized trajectory planning technique," *IEEE/ASME Trans. Mechatronics*, vol. 22, no. 1, pp. 359–370, 2017, doi: 10.1109/TMECH.2016.2602325.
- [18] S. Ivvan Valdez, S. Botello-Aceves, H. M. Becerra, and E. E. Hernandez, "Comparison between a Concurrent and a Sequential Optimization Methodology for Serial

- Manipulators Using Metaheuristics," *IEEE Trans. Ind. Informatics*, vol. 14, no. 7, pp. 3155–3165, 2018, doi: 10.1109/TII.2018.2795103.
- [19] L. Wang, Q. Wu, F. Lin, S. Li, and D. Chen, "A new trajectory-planning beetle swarm optimization algorithm for trajectory planning of robot manipulators," *IEEE Access*, vol. 7, pp. 154331–154345, 2019, doi: 10.1109/ACCESS.2019.2949271.
- [20] P. Corke, *Robotics, Vision and Control*, vol. 118. Cham: Springer International Publishing, 2017.
- [21] M. Gao, P. Ding, and Y. Yang, "Time-optimal trajectory planning of industrial robots based on particle swarm optimization," *Proc. - 5th Int. Conf. Instrum. Meas. Comput. Commun. Control. IMCCC 2015*, pp. 1934–1939, 2016, doi: 10.1109/IMCCC.2015.410.
- [22] I. Mobasher, Y. Farzaneh, B. Lotfi, and J. Enferadi, "A Trajectory Planning for a 3-RRR manipulator to remove Backlash in Actuating Joints," *5th RSI Int. Conf. Robot. Mechatronics, IcRoM 2017*, no. IcRoM, pp. 540–545, 2018, doi: 10.1109/ICRoM.2017.8466235.
- [23] C. D. Porawagama and S. R. Munasinghe, "Reduced jerk joint space trajectory planning method using 5-3-5 spline for robot manipulators," *2014 7th Int. Conf. Inf. Autom. Sustain. "Sharpening Futur. with Sustain. Technol. ICIAFS 2014*, pp. 1–6, 2014, doi: 10.1109/ICIAFS.2014.7069580.
- [24] R. Verschueren, N. Van Duijkeren, J. Swevers, and M. Diehl, "Time-optimal motion planning for n-DOF robot manipulators using a path-parametric system reformulation," *Proc. Am. Control Conf.*, vol. 2016-July, pp. 2092–2097, 2016, doi: 10.1109/ACC.2016.7525227.
- [25] D. Urwx *et al.*, "LQGXVWULDO URERW XQGHU NLQHPDWLF DQG G \ QDPLF."
- [26] G. Li and Y. Wang, "Industrial Robot Optimal Time Trajectory Planning Based on Genetic Algorithm," *Proc. 2019 IEEE Int. Conf. Mechatronics Autom. ICMA 2019*, pp. 136–140, 2019, doi: 10.1109/ICMA.2019.8816319.
- [27] R. Saidur, "A review on electrical motors energy use and energy savings," *Renewable and Sustainable Energy Reviews*, vol. 14, no. 3. Pergamon, pp. 877–898, Apr. 01, 2010, doi: 10.1016/j.rser.2009.10.018.
- [28] O. Wigström and B. Lennartson, "Energy optimization of trajectories for high level scheduling," *IEEE Int. Conf. Autom. Sci. Eng.*, pp. 654–659, 2011, doi: 10.1109/CASE.2011.6042472.
- [29] M. Alberich-Carramiñana, M. Garolera, F. Thomas, and C. Torras, "Partially flagged parallel manipulators: Singularity charting and avoidance," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 771–784, 2009, doi: 10.1109/TRO.2009.2018970.
- [30] M. Li, J. Mou, L. Chen, Y. Huang, and P. Chen, "Comparison between the collision avoidance decision-making in theoretical research and navigation practices," *Ocean Eng.*, vol. 228, p. 108881, May 2021, doi: 10.1016/j.oceaneng.2021.108881.
- [31] W. Zhang, J. Yuan, Q. Li, and Z. Tang, "An automatic collision avoidance approach for remote control of redundant manipulator," *Proc. 2008 IEEE Int. Conf. Inf. Autom. ICIA 2008*, no. 051111015, pp. 809–814, 2008, doi: 10.1109/ICINFA.2008.4608109.
- [32] M. Safeea, P. Neto, and R. Bearee, "On-line collision avoidance for collaborative robot manipulators by adjusting off-line generated paths: An industrial use case," *Rob. Auton. Syst.*, vol. 119, pp. 278–288, Sep. 2019, doi: 10.1016/j.robot.2019.07.013.
- [33] P. Ennen, D. Ewert, D. Schilberg, and S. Jeschke, "Efficient collision avoidance for industrial manipulators with overlapping workspaces," in *Procedia CIRP*, Jan. 2014, vol. 20, no. C, pp. 62–66, doi: 10.1016/j.procir.2014.05.032.
- [34] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," *arXiv*, pp. 6252–6259, 2017.
- [35] S. Equilibria, *21st Benelux Meeting on Systems and Control Book of Abstracts*, no. March. 2002.
- [36] ABB Robotics, "ABB FlexPendant plastic software," ABB, [Online]. <https://new.abb.com/products/robotics/application-software/plastic>
- [37] ABB Robotics, Product specification IRB14000, Västerås: ABB, 2018.
- [38] Plan and Execute Collision-Free Trajectories Using KINOVA Gen3 Manipulator. Available: <https://www.mathworks.com/help/robotics/ug/plan-and-execute-collision-free-trajectory-kinova-gen3.html>

- [39] Pick-and-Place Workflow Using Stateflow for MATLAB. Available:
<https://www.mathworks.com/help/robotics/ug/pick-and-place-workflow-using-stateflow.html>
- [40] Specify Constraints for Nonlinear MPC. Available:
<https://www.mathworks.com/help/mpc/ug/specify-constraints-for-nonlinear-mpc.html>

APPENDICES

Appendix 1 – Programming code for MATLAB modified from MathWork[38][39][40]

Plan and Execute a pick-place task using digital twin of ABB Yumi robot

Load the ABB Yumi rigid body tree (RBT) model with one arm

```
% Plan and Execute Task- and Joint-Space Trajectories Using ABB Yumi
Manipulator with a single arm
% Load the KINOVA Gen3 rigid body tree (RBT) robotL model.
clc
clear
robot = loadrobot('abbYumi','DataFormat','row','Gravity',[0 0 -9.81]);
robotL=robot.copy
robotL.removeBody('yumi_link_1_r')
robotL.getBody('gripper_1_base')

%Add the picked part to rigidbodytree for the logic chart to check if the part
is picked

body1 = rigidBody('pickedPart');
jnt1 = rigidBodyJoint('pickedPart','revolute');
basename = robotL.BaseName;
addBody(robotL,body1,'gripper_1_finger_r');

%Set the initial robot configuration to homeCofiguration
currentRobotJConfig = homeConfiguration(robotL);

%Build the coordinator, which deals with the robot dynamic Control
coordinator = exampleHelperCoordinatorPickPlace(robotL,currentRobotJConfig,
"gripper_1_base");

%Define the home configuration of the robot in this task, two placing poses are
defined for placing diferent types of objects

edit exampleHelperFlowChartPickPlace.sfx
coordinator.HomeRobotTaskConfig = trvec2tform([0.4, 0, 0.6])*axang2tform([0 1 0
pi]);
coordinator.PlacingPose{1} = trvec2tform([0.23 0.62 0.33])*axang2tform([0 1 0
pi]);
coordinator.PlacingPose{2} = trvec2tform([0.23 -0.62 0.33])*axang2tform([0 1 0
pi]);

coordinator.FlowChart = exampleHelperFlowChartPickPlace('coordinator',
coordinator);

%A dialog is used to start the pick-place task execution. The simulation
started by clicking on Yes

answer = questdlg('Do you want to start the pick-and-place job now?', ...
```

```

        'Start job', 'Yes', 'No', 'No');

switch answer
case 'Yes'
%     Trigger event to start Pick and Place in the Stateflow Chart
    coordinator.FlowChart.startPickPlace;
case 'No'
%     iviz = interactiveRigidBodyTree(robotL);
%     ax = gca;
%     exampleCommandBuildWorld(ax);
%     currentrobotLJConfig = homeConfiguration(robotL); %Original
    End Pick and Place
    coordinator.FlowChart.endPickPlace;
    delete(coordinator.FlowChart);
    delete(coordinator);
end

```

Logic flow to Control and track the pick - place workflow execution

properties

```

    FlowChart
    Robot
    World = {};
    Parts = {};
    Obstacles = {};
    DetectedParts = {};
    RobotEndEffector
    CurrentRobotJConfig
    CurrentRobotTaskConfig
    NextPart = 0;
    PartOnRobot = 0;
    HomeRobotTaskConfig
    PlacingPose
    GraspPose
    Figure
    TimeStep
    MotionModel
    NumJoints
    NumDetectionRuns = 0;
    PathHandle
end

```

methods

```

function obj = exampleHelperCoordinatorPickPlace(robot,
currentRobotJConfig, robotEndEffector)
    obj.Robot = robot;
    obj.CurrentRobotJConfig = currentRobotJConfig;
    obj.RobotEndEffector = robotEndEffector;
    obj.CurrentRobotTaskConfig = getTransform(obj.Robot,
obj.CurrentRobotJConfig, obj.RobotEndEffector);
    obj.TimeStep = 0.1; % Visualization time step
    obj.MotionModel = jointSpaceMotionModel('RigidBodyTree',
obj.Robot);
    obj.NumJoints = numel(obj.CurrentRobotJConfig);

    % Initialize visualization
    obj.Figure =
interactiveRigidBodyTree(obj.Robot, 'ShowMarker', false, 'Frames', 'off');
    obj.Figure.Configuration = obj.CurrentRobotJConfig;

```

```

obj.Figure.ShowMarker = false;
hold on
axis([-1 1 -1 1 -0.1 1.5]);
view(58,25);
end

function visualizeWorld(obj)
try
bench = obj.World{1};
belt1 = obj.World{2};
belt2 = obj.World{3};

% Render world
[~, p1] = show(bench);
[~, p2] = show(belt1);
[~, p3] = show(belt2);

p1.FaceColor = [1 0.5 0];
p1.FaceAlpha = 1.0;
p1.LineStyle = 'none';

p2.FaceColor = [128,128,128]/255;
p2.FaceAlpha = 1.0;
p2.LineStyle = 'none';

p3.FaceColor = [128,128,128]/255;
p3.FaceAlpha = 1.0;
p3.LineStyle = 'none';

% Visualize obstacles
for i=1:numel(obj.Obstacles)
[~, obs] = show(obj.Obstacles{i});
obs.LineStyle = 'none';
obs.FaceColor = 'b';
end

drawnow;
catch
end
end

function visualizeParts(obj)
for i = 1:length(obj.Parts)
tempPose = [0,0,0]; % to set transformation reference
correctPose = obj.Parts{i}.mesh.Pose;
obj.Parts{i}.mesh.Pose = trvec2tform(tempPose);
[~, obj.Parts{i}.plot] = show(obj.Parts{i}.mesh);
obj.Parts{i}.plot.LineStyle = 'none';
obj.Parts{i}.plot.Handle = hgtransform;
obj.Parts{i}.plot.Parent = obj.Parts{i}.plot.Handle;
obj.Parts{i}.mesh.Pose = correctPose;
obj.Parts{i}.plot.Handle.Matrix = obj.Parts{i}.mesh.Pose;
obj.Parts{i}.plot.FaceColor = obj.Parts{i}.color;
end
drawnow;
end

function visualizeRobot(obj, robotStates, trajTimes)

```

```

    % Visualize robot motion
    for k = 1:length(trajTimes)
        configNow = robotStates(k,1:obj.NumJoints);
        obj.Figure.Configuration = configNow;
        obj.Figure.ShowMarker = false;
        % Update current robot configuration
        obj.CurrentRobotJConfig = configNow;
        obj.CurrentRobotTaskConfig = getTransform(obj.Robot,
obj.CurrentRobotJConfig, obj.RobotEndEffector);
        % Visualize parts
        if obj.PartOnRobot~=0
            obj.Parts{obj.PartOnRobot}.mesh.Pose =
obj.CurrentRobotTaskConfig * trvec2tform([0 0 0.04]);
            obj.Parts{obj.PartOnRobot}.plotHandle.Matrix =
obj.Parts{obj.PartOnRobot}.mesh.Pose;
            end
            drawnow;
            pause(0.05);
        end
    end

    function visualizePath(obj, positions)
        poses = zeros(size(positions,2),3);
        for i=1:size(positions,2)
            poseNow = getTransform(obj.Robot, positions(:,i)',
obj.RobotEndEffector);
            poses(i,:) = [poseNow(1,4), poseNow(2,4), poseNow(3,4)];
        end
        obj.PathHandle = plot3(poses(:,1), poses(:,2), poses(:,3),'r-
', 'LineWidth',5);
        drawnow;
    end

    % Display current job state
    function displayState(obj, message)
        disp(message);
    end

    % Delete function
    function delete(obj)
        delete(obj.FlowChart)
    end

end

end

```

Collision – free trajectory generation between an initial configuration and a target position by using a nonlinear model predictive controller

```

% Plan collision-free trajectories using nonlinear model predictive control
curFormat = robot.DataFormat;
robot.DataFormat = 'column';

% Disable display messages
mpcverbosity off;

% Get number of joints
numJoints = numel(robot.homeConfiguration);

% Get number of obstacles

```



```

numObstacles = numel(obstacles);

% Get number of collision bodies
if ~isempty(obstacles) && avoidCollisions
    [~, separationDist, ~] = checkCollision(robot, homeConfiguration(robot),
obstacles, 'IgnoreSelfCollision', 'On', 'Exhaustive', 'on');
    tempDistances = separationDist(1:robot.NumBodies,1:numObstacles);
    bodyIndices =
find((isinf(tempDistances(:,1))|isnan(tempDistances(:,1)))==0);
    numBodies = numel(bodyIndices);
else
    numBodies = robot.NumBodies;
end

% Current robot joint configuration
currentRobotJConfig = wrapToPi(jointInit');

% Final (desired) end-effector pose
anglesFinal = rotm2eul(taskFinal(1:3,1:3), 'XYZ');
poseFinal = [taskFinal(1:3,4);anglesFinal']; % 6x1 vector for final pose: [x,
y, z, phi, theta, psi]

% Initialize safety distance away from the obstacles
safetyDistance = 0.005;

% World of obstacles
world = obstacles;

%% Set up the Nonlinear Model Predictive Controller (NLMPC)
% Cost weights
Qr = diag([1 1 1 0 0 0]); % running cost weight on desired end-effector pose
[x, y, z, phi, theta, psi]
Qt = diag([10 10 10 1 1 1]); % terminal cost weight on desired end-effector
pose [x, y, z, phi, theta, psi]
Qu = diag([1 1 1 1 1 1 1 1 1])/10; % input cost weight on joint accelerations
qDdot
Qv = diag([1 1 1 1 1 1 1 1 1])/10;

% Model joints as double integrators
nx = numJoints * 2; % [q,qDot]
ny = numJoints; % [q]
nu = numJoints; % [qDdot]

% Initialize nlmpc object
nlobj = nlmpc(nx,ny,nu);

% Solver time step
Ts = mpcTimeStep; % seconds
nlobj.Ts = Ts;

% Configure NLMPC solver functions
nlobj.Model.StateFcn = @(x,u) nlmpcModel(x,u);

nlobj.Model.OutputFcn = @(x,u) x(1:numJoints);

nlobj.Optimization.CustomCostFcn = @(X,U,e,data)
nlmpcCostFunction(X,U,e,data, poseFinal, robot, endEffector, Qr, Qt, Qu, Qv);

if ~isempty(world) && avoidCollisions
    nlobj.Optimization.CustomIneqConFcn = @(X,U,e,data)
myIneqConFunction(X,U,e,data, safetyDistance, world, robot);

```

```

end

nlobj.Jacobian.OutputFcn = @(x,u) nlmprJacobianOutputModel(x,u);

nlobj.Jacobian.StateFcn = @(x,u) nlmprJacobianModel(x,u);

nlobj.Jacobian.CustomCostFcn = @(X,U,e,data) nlmprJacobianCost(X,U,e,data,
poseFinal, robot, endEffector, Qr, Qt, Qu, Qv);

if ~isempty(world) && avoidCollisions
    nlobj.Jacobian.CustomIneqConFcn = @(X,U,e,data)
nlmprJacobianConstraint(X,U,e,data, world, robot);
end

nlobj.Optimization.SolverOptions.FunctionTolerance = 0.01;

nlobj.Optimization.SolverOptions.StepTolerance = 0.01;

nlobj.Optimization.SolverOptions.MaxIter = 5;

nlobj.Optimization.UseSuboptimalSolution = true;

nlobj.Optimization.ReplaceStandardCost = true;

% nlobj.Optimization.SolverOptions.Display = 'iter-detailed';

nlobj.Optimization.SolverOptions.ConstraintTolerance = 0.01;

% Set constraint on States and MV.
stateMinValues = {-174.53;-2.2000;-174.53;-2.5656;-174.53;-2.0500;-174.53;-
2.5656;-174.53;...
    -0.8727;-0.8727;-0.8727;-0.8727;-0.8727;-0.8727;-0.8727;-0.8727;-0.8727};
stateMaxValues =
{174.53;2.2000;174.53;2.5656;174.53;2.0500;174.53;2.5656;174.53;...
    0.8727;0.8727;0.8727;0.8727;0.8727;0.8727;0.8727;0.8727;0.8727};

nlobj.States = struct('Min',stateMinValues,...
    'Max',stateMaxValues);
nlobj.MV = struct('Min',{-1;-1;-1;-1;-1;-10;-10;-10;-
10},'Max',{1;1;1;1;1;10;10;10;10});

% Time horizon in seconds
p = 5;
nlobj.PredictionHorizon = p; % prediction horizon
nlobj.ControlHorizon = 1; % control horizon

%% Generate Reference Trajectories using Closed-loop trajectory optimization

% Initial conditions
x0 = [currentRobotJConfig', zeros(1,numJoints)];
u0 = zeros(1,numJoints);
options = nlmprmoveopt;
maxIters = 50;
success = 1;

% Initialize arrays to store the results
positions = zeros(numJoints,maxIters+1);
positions(:,1) = x0(1:numJoints)';
velocities = zeros(numJoints,maxIters+1);
velocities(:,1) = x0(numJoints+1:end)';

```

```

accelerations = zeros(numJoints,maxIters+1);
accelerations(:,1) = u0';
timestamp = zeros(1,maxIters+1);

% Run nlmprc iteratively over the specified time horizon until goal is
% achieved or up to maxIters.
mv = u0;
time = 0;
numInfeas = 0;
% Uncomment below to display all successful outputs
% disp('Calculating collision-free trajectory...')
for timestep=1:maxIters
    % Optimize next trajectory point
    [mv,options,info] = nlmprcmove(nlobj,x0,mv,[],[], options);

    if info.ExitFlag < 0
        numInfeas = numInfeas + 1;
        disp('Failed to compute a feasible trajectory in this step...')
    end

    if numInfeas>2
        warning('Many infeasible solutions in a row. Aborting...')
        success = 0;
        break;
    end

    % Update initial state and time for next iteration
    x0 = info.Xopt(2,:);
    time = time + nlobj.Ts;

    % Store trajectory points
    positions(:,timestep+1) = x0(1:numJoints)';
    velocities(:,timestep+1) = x0(numJoints+1:end)';
    accelerations(:,timestep+1) = info.MVopt(2,:)';
    timestamp(timestep+1) = time;

    % Check if goal is achieved
    jointTempFinal = info.Xopt(2,1:numJoints);
    taskTempFinal = getTransform(robot, jointTempFinal', endEffector);
    anglesTempFinal = rotm2eul(taskTempFinal(1:3,1:3), 'XYZ');
    poseTempFinal = [taskTempFinal(1:3,4);anglesTempFinal'];
    diffTerminal = abs([poseFinal(1:3)-poseTempFinal(1:3);
angdiff(poseTempFinal(4:6),poseFinal(4:6))]);
    if all(diffTerminal<tolerance)
        break; % goal achieved
    end
end

robot.DataFormat = curFormat;

%% Output the reference trajectories
tFinal = timestep+1;
positions = positions(:,1:tFinal);
velocities = velocities(:,1:tFinal);
accelerations = accelerations(:,1:tFinal);
timestamp = timestamp(:,1:tFinal);
return;

%% Helper nlmprc functions

```

```

function dxdt = nlmpcModel(x,u)
    dxdt = zeros(size(x));
    dxdt(1:numJoints) = x(numJoints+1:end);
    dxdt(numJoints+1:end) = u;
end

function [A, B] = nlmpcJacobianModel(x,u)
    A = zeros(numJoints*2, numJoints * 2);
    A(1:numJoints, numJoints+1:end) = eye(numJoints);
    B = zeros(numJoints*2,numJoints);
    B(numJoints+1:end,:) = eye(numJoints);
end

function [C, D] = nlmpcJacobianOutputModel(x,u)
    C = zeros(numJoints, numJoints * 2);
    C(1:numJoints, 1:numJoints) = eye(numJoints);
    D = zeros(numJoints, numJoints);
end

function cost = nlmpcCostFunction(X,U,e,data, poseFinal, robotL,
endEffector, Qr, Qt, Qu, Qv)
    % Running Cost
    costRunning = 0;
    for i= 2:p+1
        jointTemp = X(i,1:numJoints);
        taskTemp = getTransform(robotL, jointTemp', endEffector);
        anglesTemp = rotm2eul(taskTemp(1:3,1:3), 'XYZ');
        poseTemp = [taskTemp(1:3,4);anglesTemp'];
        diffRunning = [poseFinal(1:3)-poseTemp(1:3);
angdiff(poseTemp(4:6),poseFinal(4:6))];
        costRunningTemp = diffRunning' * Qr * diffRunning;
        costRunning = costRunning + costRunningTemp + U(i,:)*Qu*U(i,:)';
    end

    % Terminal cost
    costTerminal = diffRunning'* Qt * diffRunning +
X(p+1,numJoints+1:end)*Qv*X(p+1,numJoints+1:end)';

    % Total Cost
    cost = costRunning + costTerminal;
end

function [G,Gmv,Ge] = nlmpcJacobianCost(X,U,e,data, poseFinal, robot,
endEffector, Qr, Qt, Qu, Qv)
    % Initialize Jacobians
    G = zeros(p,numJoints*2);
    Gmv = zeros(p,numJoints);
    Ge = 0;

    % Update G
    for i=1:p
        jointTemp = X(i+1,1:numJoints);
        taskTemp = getTransform(robot, jointTemp', endEffector);
        anglesTemp = rotm2eul(taskTemp(1:3,1:3), 'XYZ');
        poseTemp = [taskTemp(1:3,4);anglesTemp'];
        diffRunning = [poseFinal(1:3)-poseTemp(1:3);
angdiff(poseTemp(4:6),poseFinal(4:6))];

        % From geometric to analytical robot Jacobian
        rx = anglesTemp(1);
        py = anglesTemp(2);
    end
end

```

```

        B = [ 1 0 sin(py); 0 cos(rx) -cos(py)*sin(rx); 0 sin(rx)
cos(py)*cos(rx) ];

        % Robot Jacobian
        robotJacobianTemp = geometricJacobian(robot, jointTemp', endEffector);
        robotJacobian = robotJacobianTemp;
        robotJacobian(1:3,:) = robotJacobianTemp(4:6,:);
        robotJacobian(4:6,:) = B\robotJacobianTemp(1:3,:);

        % Running cost Jacobian
        G(i,1:numJoints) = (-2 * diffRunning' * Qr * robotJacobian);
        Gmv(i,:) = 2 * U(i+1,:) * Qu;
    end

    % Terminal cost Jacobian
    G(p,1:numJoints) = G(p,1:numJoints) + (-2 * diffRunning' * Qt *
robotJacobian);
    G(p,numJoints+1:end) = 2 * X(p+1,numJoints+1:end) * Qv;

end

function cineq = myIneqConFunction(X,U,e,data, safetyDistance, world, robot)
    % Copyright 2019 The MathWorks, Inc.
    allDistances = zeros(p*numBodies*numObstacles,1);
    for i = 1:p
        collisionConfig = X(i+1,1:numJoints);
        [~, separationDist, ~] = checkCollision(robot, collisionConfig',
world, 'IgnoreSelfCollision', 'On', 'Exhaustive', 'on');
        tempDistances = separationDist(1:robot.NumBodies,1:numObstacles);
        tempDistances(all(isinf(tempDistances) | isnan(tempDistances),2),:) =
[]; % remove inf and nans
        tempDistances(isinf(tempDistances) | isnan(tempDistances)) = 0;
        allDistances((1+(i-
1)*numBodies*numObstacles):numBodies*numObstacles*i,1) =
reshape(tempDistances', [numBodies*numObstacles,1]);
    end
    cineq = -allDistances + safetyDistance;
end

function [G,Gmv,Ge] = nlmpcJacobianConstraint(X,U,e,data, world, robot)

    % Initialize Jacobians
    G = zeros(p, numJoints*2, p*numBodies*numObstacles);
    Gmv = zeros(p, numJoints, p*numBodies*numObstacles);
    Ge = zeros(p*numBodies*numObstacles,1);

    iter = 1;
    for i=1:p
        collisionConfig = X(i+1,1:numJoints);
        [~, ~, allWntPts] = checkCollision(robot, collisionConfig', world,
'IgnoreSelfCollision', 'On', 'Exhaustive', 'on');
        for j=1:numBodies
            for k=1:numObstacles
                bodyNow = bodyIndices(j);
                wtnPts = allWntPts(1+(bodyNow-1)*3:3+(bodyNow-1)*3, 1+(k-
1)*2:2+(k-1)*2);
                if isempty(wtnPts(isinf(wtnPts) | isnan(wtnPts)))
                    if any((wtnPts(:,1)-wtnPts(:,2))~=0)
                        normal = (wtnPts(:,1)-wtnPts(:,2))/norm(wtnPts(:,1)-
wtnPts(:,2));

```

```

        else
            normal = [0;0;0];
        end
        bodyJacobian = geometricJacobian(robot,collisionConfig',
robot.BodyNames{bodyNow});
        G(i, 1:numJoints, iter)= -normal' * bodyJacobian(4:6,:);
    else
        G(i, 1:numJoints, iter) = zeros(1, numJoints);
    end
    iter = iter + 1;
end
end
end
end

end
end

```

Move the robot from current position to a desired pose

```

% Parts will be picked according to order in coordinator.DetectedParts list
coordinator.NextPart = coordinator.NextPart + 1;
if coordinator.NextPart<=length(coordinator.Parts)
    % Objects are placed on either belt1 or belt2 according to
    % their type
    if coordinator.DetectedParts{coordinator.NextPart}.type == 1
        coordinator.DetectedParts{coordinator.NextPart}.placingBelt =
1;
    else
        coordinator.DetectedParts{coordinator.NextPart}.placingBelt =
2;
    end
    % Trigger Stateflow chart Event
    coordinator.FlowChart.partsDetected;
    return;
end

% Trigger Stateflow chart Event
coordinator.FlowChart.noPartsDetected;

```

The logic of pick and place for the robot model

```

% Parts will be picked according to order in coordinator.DetectedParts list
coordinator.NextPart = coordinator.NextPart + 1;
if coordinator.NextPart<=length(coordinator.Parts)
    % Objects are placed on either belt1 or belt2 according to
    % their type
    if coordinator.DetectedParts{coordinator.NextPart}.type == 1
        coordinator.DetectedParts{coordinator.NextPart}.placingBelt =
1;
    else
        coordinator.DetectedParts{coordinator.NextPart}.placingBelt =
2;
    end
    % Trigger Stateflow chart Event
    coordinator.FlowChart.partsDetected;
    return;
end

% Trigger Stateflow chart Event
coordinator.FlowChart.noPartsDetected;

```

Move the robot to a target position

```
%mpcTimeStep = 0.6; Set time step to different value
mpcTimeStep = 0.2;
    [positions, velocities, accelerations, timestamp, success] =
exampleHelperPlanExecuteTrajectoryPickPlace(coordinator.Robot, mpcTimeStep,
coordinator.Obstacles, coordinator.RobotEndEffector,
coordinator.CurrentRobotJConfig, taskConfig, tolerance, avoidCollisions);
    if success==0
        error('Cannot compute motion to reach desired task configuration.
Aborting...')
    end

    %% Execute the trajectory using low-fidelity simulation
targetStates = [positions;velocities;accelerations]';
targetTime = timestamp;
initState = [positions(:,1);velocities(:,1)]';
trajTimes = targetTime(1):coordinator.TimeStep:targetTime(end);

    [~,robotStates] = ode15s(@(t,state)
exampleHelperTimeBasedStateInputsPickPlace(coordinator.MotionModel,
targetTime, targetStates, t, state), trajTimes, initState);

    %% Visualize trajectory
% Uncomment below to display all successful outputs
% disp('Executing collision-free trajectory...')
visualizePath(coordinator,positions);
visualizeRobot(coordinator, robotStates, trajTimes);

% Deleta path on plot
coordinator.PathHandle.Visible = 'off';

% Update current robot configuration
coordinator.CurrentRobotJConfig = positions(:,end)';
coordinator.CurrentRobotTaskConfig = getTransform(coordinator.Robot,
coordinator.CurrentRobotJConfig, coordinator.RobotEndEffector);

% Trigger Stateflow chart Event
coordinator.FlowChart.taskConfigReached;
end
```

Build the simulation environment

```
% Construct the Workstation (only for visualization)
bench = collisionBox(0.5, 0.7, 0.05);
belt1 = collisionBox(1.3, 0.4, 0.05);
belt2 = collisionBox(1.3, 0.4, 0.05);

TBench = trvec2tform([0.4 0 0.2]);
TBelt1 = trvec2tform([0 -0.6 0.2]);
TBelt2 = trvec2tform([0 0.6 0.2]);

bench.Pose = TBench;
belt1.Pose = TBelt1;
belt2.Pose = TBelt2;

coordinator.World = {bench, belt1, belt2};
```

```

obs1 = collisionSphere(0.13);
Tobs1 = trvec2tform([0.4 0.38 0.4]);
obs1.Pose = Tobs1;

obs2 = collisionSphere(0.13);
Tobs2 = trvec2tform([0.4 -0.38 0.4]);
obs2.Pose = Tobs2;

coordinator.Obstacles = {obs1, obs2};

% Add the parts, which are only used for visualization and
% simulation. A separate tool ensures that when a part is
% gripped, it is included in the collision detection stage of
% the trajectory optimization workflow.
box2 = collisionBox(0.06, 0.06, 0.1);
box3 = collisionBox(0.06, 0.06, 0.1);
box1 = collisionBox(0.06, 0.06, 0.1);

% Move the parts into position
TBox2 = trvec2tform([0.5 -0.15 0.26]);
TBox3 = trvec2tform([0.52 0 0.26]);
TBox1 = trvec2tform([0.4 -0.1 0.26]);

box2.Pose = TBox2;
box3.Pose = TBox3;
box1.Pose = TBox1;

% Set the part mesh and color
part1.mesh = box2;
part2.mesh = box3;
part3.mesh = box1;

part1.color = 'y';
part2.color = 'y';
part3.color = 'g';

part1.centerPoint = tform2trvec(part1.mesh.Pose);
part2.centerPoint = tform2trvec(part2.mesh.Pose);
part3.centerPoint = tform2trvec(part3.mesh.Pose);

part1.plot = [];
part2.plot = [];
part3.plot = [];

coordinator.Parts = {part1, part2, part3};

% Visualize world and parts
visualizeWorld(coordinator)
visualizeParts(coordinator)

% Trigger Stateflow chart Event
coordinator.FlowChart.worldBuilt;
end

```