COLLISION AVOIDANCE SYSTEM FOR HUMAN-ROBOT COLLABORATION

COLLISION AVOIDANCE SYSTEM FOR HUMAN-ROBOT COLLABORATION

By Peige Guo, B.Eng

A Thesis

Submitted to the School of Graduate Studies

In Partial Fulfillment of the Requirements

For the Degree

Master of Applied Science

McMaster University

MASTER OF APPLIED SCIENCE (2020)

McMaster University

(Mechanical Engineering)

Hamilton, Ontario

TITLE: Collision Avoidance System for Human-Robot Collaboration

AUTHOR: Peige Guo, B.Eng

SUPERVISOR: Dr. Gary M. Bone, Professor

NUMBER OF PAGES: xxi, 114

Lay abstract

Robots are expected to become a significant part of our society in the future. They will need to interact and collaborate with people at home and at work. This will require them to rapidly adapt to dynamic situations. This thesis is concerned with solving this problem for a robot arm working close to a person. A depth camera measures colour and depth of the area around the robot. A software algorithm is presented to model the person using this colour and depth information. Two software algorithms are presented to control the robot arm. They try to move the end of the robot arm towards a target location while simultaneously avoiding it colliding with the person. They are tested using a simulated industrial robot arm. The results show that the algorithms working together can prevent collisions between the person and robot, while simultaneously moving the robot towards its target location.

Abstract

To fully exploit the advantages of human-robot collaboration the robot must be allowed to move when the human is close to it or even in contact with it. This thesis presents the development of a collision avoidance system which addresses the safety problem for the case of one person sharing a workspace with a robot manipulator. The system consists of a depth camera that measures both the colour and depth of the scene near to the robot, a laptop computer and several software algorithms. A human modeling algorithm generates a plane model and union of spheres model from the point cloud. Sphere-swept lines are used to geometrically model each link of the robot. Their position and orientation in space are calculated using the robot's joint position measurements and its kinematic model. Two collision avoidance algorithms are presented for controlling the robot's trajectory based on the geometric models for the human and robot, and the robot's desired task. The first collision avoidance algorithm solves the inverse kinematics problem and avoids collisions using an expanded version of the manipulator Jacobian matrix. A second collision avoidance algorithm using nonlinear model predictive control is developed as an alternative approach. The algorithms have been implemented in a simulated environment which includes a human working in the shared workspace with a simple planar robot and with an Elfin 5 industrial robot. A variety of scenarios are simulated and the results are compared. The simulation results showed that the first collision avoidance algorithm may be computed fast enough to be applied in real-time and worked well for static or slowly moving obstacles. The second collision avoidance algorithm had superior performance when the obstacle was moving and when the simulated robot had a realistic time delay. However, its computation time was too long to be used in real-time.

Acknowledgment

First and foremost, I would like to express my sincere gratitude to my research supervisor, Dr. Gary M. Bone, for his valuable guidance and support. He has walked me through all of the stages of this research. Without him, this thesis could never have been finished. My graduate studies have allowed me to learn a lot and gain skills across a wide variety of topics

I also thank my colleagues: Abdelrahman Zaghloul and Behrad Rouzbeh for their advice and encouragement. And I like to express my thanks to all my friends for their support and encouragement.

Finally, thanks to my family who have been nothing but supportive.

Contents

List of Figures

List of Tables

Abbreviations

CLIKFR: Closed-loop inverse kinematics control algorithm exploiting functional redundancy

COVID-19: Coronavirus disease of 2019

D-H: Denavit-Hartenberg

DOF: Degrees-of-freedom

GMMS: Gaussian mixture models

GMRs: Gaussian mixture regressions

HRC: Human-robot collaboration

IMUs: Inertial measurement units

RGB-D: Red, green, blue and depth

SSL: sphere-swept line

SSLs: sphere-swept lines

LED: Light Emitting Diode

MLESAC: Maximum Likelihood Estimator SAmple Consensus

MPC: Model predictive control

NPC: Non-predictive control

NMPC: Nonlinear model predictive control

PCA: Principal component analysis

RANSAC: RANdom SAmple Consensus

RGB: Red, green and blue

SFs: Saliency features

SVD: Singular value decomposition

T-RRT: Transition-based rapidly exploring random tree

YCbCr: Luminance; Chroma: blue; Chroma: red

Nomenclature

a_i : Link length.

$\mathbf{A_1}, \mathbf{A}_2, \mathbf{A}_3, \mathbf{A}_4$: Expanded Jacobian matrix.

b_1, b_2 : Spheres in the SSLs model.

$\mathbf{b_1}, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4$: Expanded target matrix.

$\dot{\mathbf{c}}$: Moving speed of the obstacle.

$\mathbf{c_0}$: Centre of the obstacle.

$\mathbf{C}$: Covariance matrix of each point in the point cloud.

$\mathbf{C_1}, \mathbf{C}_2, \mathbf{C}_3, \mathbf{C}_4$: Joint angle constraints, joint velocity constraints, joint acceleration constraints and collision avoidance constraints.

d : Depth value of a pixel in the depth image.

d_x, d_y : Resolution of the image in the x and y directions.

d_i : Joint offset.

$d(b_1, b_2)$: Separation distance.

$d_p(b_1, b_2)$: Euclidean distance between a point and a line segment.

d_{plane} : Distance from a point to the plane.

$\mathbf{d_q}$: Threshold joint angle distance.

d_s : Desired distance between the robot centerline and obstacle centre.

$d_{\min}$: Closest distance between the obstacle and whole robot arm.

$\mathbf{e_t}$: Extended error vector in the task space.

$\mathbf{I_n}$: Identity matrix of dimension n.

$\mathbf{J_{critical}}$: First 3 rows of the Jacobian matrix of critical point.

$\mathbf{J_{criticalPlane}}$: Jacobian matrix of the critical point on the arm to a plane.

$\mathbf{J_{fPlane}}$: Jacobian matrix of the plane avoidance constraint.

$\mathbf{J(q)}$: Geometric Jacobian matrix of the manipulator.

$\mathbf{J_{qi}}$: Jacobian matrix of joints constraints task.

$\mathbf{J_o}$: Jacobian matrix relating the joint velocities to the end-effector angular velocity.

$\mathbf{J_p}$: Jacobian matrix relating the joint velocities to the end-effector linear velocity.

$\mathbf{J_s}$: Jacobian of sphere collision avoidance task.

k : Number of nearest neighbours.

$\mathbf{K}$: Positive-definite proportional gain matrix.

K_{es} : Gain of sphere obstacle collision avoidance error

L : Length of the voxel cell.

n_s : Total number of obstacles.

N_p : Prediction horizon.

$\mathbf{p_{1,1}}(t), \mathbf{p_{1,2}}(t)$: Two dynamic endpoints of a line segment of SSL.

$\mathbf{p}_{\mathbf{critical}}$: Critical point for whole arm.

$\mathbf{p}_{\mathbf{c1}}, \mathbf{p}_{\mathbf{c2}}, \mathbf{p}_{\mathbf{c3}}, \mathbf{p}_{\mathbf{c4}}$: Closest point to the obstacle on link 1, link 2, link 3 and link 4.

$\mathbf{p}_{\mathbf{ed}}$: Desired end-effector position.

$\mathbf{p}_{\mathbf{joint1}}, \mathbf{p}_{\mathbf{joint2}}, \mathbf{p}_{\mathbf{joint3}}, \mathbf{p}_{\mathbf{joint4}}$: The location of joint 1, joint 2, joint 3 and joint 4.

$\mathbf{p}_{\mathbf{e}}$: Position of the end-effector.

$\dot{\mathbf{p}}_e$: Linear velocity of the end-effector.

$\mathbf{p}_{\mathbf{i-1}}$: Position of the origin of the i^{th} joint frame.

$\mathbf{q}$: Joint position variables.

$q_{i\min}, q_{i\max}$: Minimum and maximum angle of joint i.

r_1, r_2 : Radius of robot arm's SSLs model union of spheres human model.

${}^{3}\mathbf{R}_{6}$: orientation of the end-effector relative to frame 3.

${}^{i+1}\mathbf{T}_i$: Homogeneous transformation matrix.

u_0, v_0 : Origin of the x and y axes in the image coordinate system.

$\mathbf{v}_{1toc1}$: Vector from joint 1 to the closest point to the obstacle on link 1.

w : Weight factor that is associated with the collision avoidance task.

$\bar{\mathbf{w}}, \bar{\mathbf{W}}$: Weight array and weight matrix.

$\mathbf{x}$: Position and orientation of the end-effector.

$\mathbf{x}_{\mathbf{d}}$: Desired end-effector position and orientation.

$\mathbf{z}_{\mathbf{i-1}}$: The z-axis of the i^{th} joint frame.

α_i : Link twist.

θ_a : Actual joint angle.

θ_c : Commanded joint angle.

$\boldsymbol{\theta}_{c,\min}$, $\boldsymbol{\theta}_{c,\max}$: Minimum and maximum joint angle limits.

$\dot{\boldsymbol{\theta}}_{c,\min}$, $\dot{\boldsymbol{\theta}}_{c,\max}$: Minimum and maximum joint velocity limits.

$\ddot{\boldsymbol{\theta}}_{c,\min}$, $\ddot{\boldsymbol{\theta}}_{c,\max}$: Minimum and maximum joint acceleration limit.

$\boldsymbol{\theta}_{\mathbf{c}}$, $\dot{\boldsymbol{\theta}}_{\mathbf{c}}$, $\ddot{\boldsymbol{\theta}}_{\mathbf{c}}$: Joint position, velocity and acceleration commands.

$\boldsymbol{\theta}_{\mathbf{cv,min}}$, $\boldsymbol{\theta}_{\mathbf{cv,max}}$: Minimum and maximum joint velocity commands.

$\boldsymbol{\theta}_{\mathbf{ca,min}}$, $\boldsymbol{\theta}_{\mathbf{ca,max}}$ Minimum and maximum joint acceleration commands.

θ_i : Joint angle.

λ : Scalar damping factor.

$\lambda_1, \lambda_2,$ and λ_3 : Eigenvalues of the covariance matrix $\mathbf{C}$.

τ_d : Number of sampling periods of delay.

$\boldsymbol{\omega}_e$: Angular velocity of the end-effector.

Chapter 1. Introduction

1.1 Preface

Robot manipulators have been widely used in assembly lines for mass production for decades. Recently, the demand for industrial robots has accelerated considerably due to the ongoing trend towards automation (International Federation of Robotics, 2019). The COVID-19 pandemic has also increased the use of robot manipulators in new applications such as cooking (Durban and Chea, 2020) .

To produce more complex products with shorter product life cycles greater human-robot collaboration (HRC) is necessary. This collaboration combines the adaptability and problem-solving skills of humans with the strength, endurance, and precision of robots (Ajoudani *et al.*, 2018). Generally speaking, human-robot collaboration can be classified into two types according to the existence of intentional physical contact between humans and robots. In the type with intentional physical contact, robot manipulators assist human operators in moving heavy objects by pushing/pulling/holding an object and compensating for the object's inertia and weight. On the other hand, for the type without intentional physical contact, robot manipulators act like co-workers and cooperate with human workers while avoiding physical contact and collisions (Villani *et al.*, 2018). For instance, robot manipulators could provide assembly components to human operators during the assembly process (Tan *et al.*, 2010). Another example from industry is that robot manipulators carry large components and finish the non-ergonomic part of the assembly process, and then the human workers continue the part which requires adaptability (Universal Robots, 2018).

To maintain production efficiency, industrial robot manipulators are typically operated at high speed and thus are isolated from humans using fences since high speed collisions could cause serious injuries. Traditionally, whenever there is an overlap of robot manipulator workspaces and human workspaces, the motions of the robot manipulators are ceased, which greatly reduces the production efficiency and limits the applicability of human-robot collaboration. To fully exploit the advantages of HRC the robot must be allowed to move when the human is close to it or even in contact with it, therefore human safety is a critical issue that needs to be considered.

Many researchers have worked on safety in HRC, and there are a lot of existing strategies to avoid personal injuries. Their approaches can be separated into pre- and post-collision strategies (De Luca *et al.*, 2006). Pre-collision strategies are implemented before human-robot collision occurs. Collision avoidance is the primary goal and requires (at least, local) knowledge of the current environment geometry and computationally intensive motion planning techniques. Post-collision methods are focused on collision detection and reaction. They are designed to quickly detect the collision and minimize harm to both humans and robots if unexpected contact occurs.

Pre-collision strategies require that the robot can re-plan the trajectory in which it reaches the target position in a way that avoids the collision with humans in real-time. The perception of the human operator is the first crucial step. With the help sensors that measure both colour and depth (known as "RGB-D sensors" or "RGB-D cameras" where the letters R, G, B and D are abbreviations of red, green, blue and depth, respectively), it is possible to acquire sufficient information about the position of the human operator for re-planning the robot's trajectory (Dal Mutto *et al.*, 2012). This type of sensor is popular in robotic applications, not only for its features, but also for its low cost. Intel RealSense and Microsoft Kinect are commercial RGB-D sensors which

combine a colour video camera with an infrared projector and receiver for measuring depth. They have been broadly used in both the academic community and industry for human position measurement and gesture recognition (Ye *et al.*, 2013).

However, only a few HRC applications have established in industrial assembly or manufacturing areas at present (Matheson *et al.*, 2019). This is due to the stringent demand for safety during autonomous movements of robots in fence-less applications and the high cost of the workspace monitoring systems (Fast-Berglund *et al.*, 2016). Therefore, with the help of RGB-D sensors, a safe and affordable pre-collision system for HRC applications can be developed.

1.2 Objective and Organization

The objective of this research is to develop a collision avoidance system suitable for industrial robot manipulators in general, and for collaborative robots (also known as "cobots") in particular. The system should be able to generate, in real-time, a trajectory for the robot to perform its task while simultaneously avoiding collisions with human workers. This will involve developing algorithms for modelling the robot and human(s), and for collision avoidance. The collision avoidance algorithms will be evaluated in terms of their efficiency (measured by the time required for the robot to complete its assigned task), safety (measured by the closest distance between the human(s) and robot), and their real-time capability (measured by their execution time).

The organization of the thesis is as follows. In chapter 2, the relevant publications on human geometric modelling, robot geometric modelling and robot manipulator collision avoidance algorithms are reviewed. Chapter 3 presents the human and robot modeling algorithms. The non-predictive and predictive collision avoidance algorithms are described in chapter 4. In chapter 5， collision avoidance simulations are presented.

The simulation procedure is described first. After that the simulation results for a 3-DOF planar robot (where "DOF" stands for "degrees-of-freedom") and a 6-DOF collaborative robot are presented and discussed. Chapter 6 draws the conclusions of the research and makes recommendations for future work.

Chapter 2. Literature review

2.1 Introduction

Safety takes priority over all other matters for HRC. Many researchers have worked on safety in HRC, and there are several existing strategies to avoid human injuries. They can be classified into two main categories: pre-collision strategies and post-collision strategies. Pre-collision strategies are implemented before the human-robot collision occurs, either by ensuring collision does not happen in the first place or by bounding robot parameters such as velocity or energy. If unexpected or unpreventable contact occurs, post-collision control methods are designed to quickly detect the collision and minimize harm to both the human and the robot.

Collision avoidance methods belong to the pre-collision category. Geometric models of the robot are necessary for applying these methods. To be more effective at avoiding human(s) in the shared workspace, collision avoidance algorithms need more knowledge than merely knowing the presence or absence of human subjects. Individual body parts must be detected and represented as geometric models. The research on robot geometric models is reviewed in section 2.2. Human geometric modelling techniques are covered in section 2.3. Algorithms for collision avoidance are reviewed in section 2.4. Finally, section 2.5 presents a summary of the review.

2.2 Robot Geometric Modelling for Human-Robot Collaboration

Over the years, several different geometric representations for robot manipulators have been proposed. Balan and Bone (2006) represented each robot link by a union of spheres. e.g., a PUMA 762 was modeled by 41 spheres. This model can work in real-time due to the efficiency of the distance computation. Flacco et al. (2012) presented a

similar spheres-based model. The disadvantage of this approach is many spheres must be used to obtain detailed models.

Another way to geometrically represent robots is to use a sphere-swept line (SSL) to model each link. An SSL is simply the bounding volume created by translating a sphere along a line segment. It is geometrically equivalent to a cylinder with hemispherical endcaps. Although the distance computation with SSLs is more complex than with spheres the use of SSLs is typically more computationally efficient since only a few SSLs are required to model the entire manipulator. This robot modelling approach was used by Bosscher and Hedman (2011), Corrales et al. (2011) and Krämer et al. (2020).

Gerdts et al. (2012) modelled the robot with polyhedrons and utilized Farkas lemma in conjunction with back-face culling to reduce computational complexity. However, the algorithm does not run in real-time due to the high complexity of the additional constraints and optimization parameters introduced by Farkas lemma.

2.3 Human Sensing and Geometric Models for Human-Robot Collaboration

Human geometric modelling is a much more difficult task than robot geometric modelling for two main reasons. First, the shapes of the robot's links are constant and known whereas the shape of the human operator is unknown and variable. Second, the positions of the robot's links in space can be determined using the robot's built in joint angle sensors whereas locating the human's body requires external sensors (such as RGB-D sensors) and software to distinguish it from its surroundings.

Martínez-Salvador et al. (2003) modelled the human by a hierarchy of sphere-based models. They used a complex heuristic algorithm to compute the sphere positions and radii. Balan and Bone (2006) also developed a sphere-based human model

customized to fit the dimensions of an average-sized human. However, neither of these papers used the models in experiments where sensor(s) must be used to obtain the human model in real-time.

Morato et al. (2014) presented a human-robot collaboration approach that uses multiple Microsoft Kinect RGB-D sensor to extract a skeleton model of the human. They then cover head, torso and arms of the skeleton model using nine spheres. They test their approach for a cooperative assembly task using a small 5-DOF robot. Real-time performance is achieved (i.e., 30 Hz sampling rate), but the skeleton identification can fail if the human is handling some equipment. A further disadvantage is the skeleton does not represent the surface of the human's body so it is not ideally suited for ensuring human safety.

Lasota et al. (2014) developed a safety system for close-proximity interaction with standard industrial robots that leverages accurate sensing of a human's location. A PhaseSpace motion capture system was utilized to sense the position of the human worker within the workspace. Consequently, the position of the human in the virtual workspace was approximated by two concentric cylinders: one cylinder for the forearm and one larger-diameter cylinder for the hand. However, the motion capture system requires the user to wear LEDs on their arm and hand that are always visible by the cameras, so it is not a practical solution for industrial environments.

An alternative approach is to calculate distances between humans and robots in depth space, as described in Flacco et al. (2015). They proposed a method that evaluates point-to-object distances working in the depth space of a single Kinect sensor. The depth data is not converted to Cartesian space, and the distances are calculated using rays from the camera's focal point. The distances are used to generate repulsive vectors that are used to control the robot while executing a generic motion task. The real-time

performance of the proposed approach is shown by means of collision avoidance experiments. In their experiments the human is standing far away from background objects. Since this method uses only depth data it may confuse the human with nearby objects which may lead to human-robot collisions.

Ragaglia et al. (2018) proposed a skeleton plus swept volumes approach to model the space occupied by the human worker. Kalman filtering is employed to fuse depth data from multiple Kinect sensors. As with Morato *et al.* (2014) the skeleton identification can fail if the human is handling some equipment so this is not a very reliable approach.

Beckert et al. (2018) enclosed the human upper-body in SSLs to compute their reachable occupancies. The SSLs radii are specified such that they enclose all body parts. They use six motion-capture cameras that track reflecting marker clusters which are affixed to the body. Safeea and Neto (2019) also applied SSLs to represent the human in the workspace. The upper body configuration is captured using five inertial measurement units (IMUs) that are attached to the upper body parts (chest and arms), and a laser scanner that tracks legs. However, these approaches require the user to wear markers or IMUs, require special tracking equipment and do not provide sufficiently comprehensive environment information for collision avoidance. This makes them poorly suited for industrial applications.

2.4 Robot Manipulator Collision Avoidance Algorithms

2.4.1 Motion planning-based algorithms

The purpose of motion planning is to find a continuous, collision-free path from the starting pose of the robot to its target pose. The generated collision-free path is converted to a reference trajectory and sent to the robot's controller once (before the

robot starts moving), unless replanning is performed. Since the state of the robot within a specific time range is determined, its real-time performance cannot be guaranteed.

A motion planner developed by Mainprice et al. (2011) employs a cost-based, random-sampling search to plan safe robot motions within cluttered environments It includes the constraints due to human vision field, human-robot separation distance, and the robot's work space. This planner incorporates a transition-based rapidly exploring random tree (T-RRT) algorithm and local optimization to generate paths.

A variety of planners and approaches that modify both the path as well as the time parameterization of a trajectory to provide the most flexibility have been developed. One approach developed by Yoshida et al. (2010) performs planning and execution asynchronously. Upon changes in the environment, a subroutine replans a trajectory based on the current initialization and passes it to the execution process. The replanning time is 2-3 seconds. Another approach by Kohrt et al. (2013) proposed a grid-based path planning method that restricts the search space to Voronoi regions. The trajectory then emerges from a time parametrization along the path, for example, based on trapezoidal velocity profiles or piecewise polynomial interpolation with imposed velocities and accelerations. However, these papers assume the paths and velocities of the obstacles are known in advance so they are not well suited for avoiding humans.

A different approach is to use a fixed preplanned path and only modify the time parametrization of the corresponding trajectory. Recent research applying this approach was proposed by Beckert et al. (2018). By scaling the time parametrization, the robot responds to a dynamic environment by either slowing down or speeding up its motion, enabling a time-shared collaboration. The robot's velocity, acceleration, and jerk limits are included in their solution. The controller verifies a given nominal trajectory, ensuring safety with formally verified path-consistent fail-safe maneuvers. The

positions of both the manipulator and the operator over time are predicted by computing their reachable occupancies. The simulation shows that the robot will take 1.6 seconds to stop before impact. However, the robot will freeze and delay production if a human is blocking the predefined path.

Motion planning-based methods allow for a more proactive approach to ensuring safety. They can search for the optimal path that meets certain constraints, such as minimum distance and shortest time, etc. However, this approach is not appropriate for maintaining safety with more dynamic environments. Besides, there are practical limits to constantly replanning based solely on the current configuration of a rapidly changing environment.

2.4.2 Prediction-based algorithms

If humans and robots are working close to one another, the ability to anticipate the actions and movements of members of a human-robot team is critically important for providing safety within dynamic HRC. Typically, the state of the robot is known, and the state of humans can be obtained through vision or other motion capture sensors. If the person's movements conform to a particular probability model, the state of the person at the next moment can be predicted. By providing this information to the upper-level planner, or the lower-level control strategy, actions and paths that will result in safe and efficient interaction can be determined.

Mainprice and Berenson (2013) introduced a method that applies labeled demonstrations of reaching motions to generate models for prediction of workspace occupancy. In this approach, separate Gaussian mixture models (GMMs) are trained for each goal position for a particular task, and Gaussian mixture regressions (GMRs) are used to generate representative reaching motions. Then, based on observation of the

initial segment of a new reaching motion and the computed GMMs and GMRs, the approach calculates the likelihood of occupancy of each voxel within a simulated shared workspace. The robot then selects actions and paths that minimize incursion into the regions of the workspace expected to be occupied by humans. However, neither the prediction nor the motion planning processing time achieved sufficient real-time performance to be used online for a real application, and the average correct classification at the early stages of the trajectory was still very low, with 50% correct classification achieved after processing 43% of the motion trajectory.

Perez-D'Arpino and Shah (2015) also focused on predicting reaching locations based on human demonstrations, but with a time-series analysis that utilizes multivariate Gaussian distributions over the tracked degrees of freedom of the human arm defined for each time step of the motion. The system uses the learned models to perform Bayesian classification on the initial stages of motion to predict where a person will reach toward and to select robot actions that minimize interference. The models take the sequence of points along the motion trajectory into consideration, allowing for better discriminability and higher classification confidence very early in the process of the human's motion. This approach assumes a known dynamics model of the world and a known goal representation. However, these assumptions constrain the adaptability and generality of the system for many real-world tasks.

Ragaglia et al. (2018) proposed a novel trajectory generation algorithm which incorporates the safety requirements for an industrial robot involved in HRC. Their trajectory generation algorithm modifies a pre-programmed trajectory to prevent collisions between the robot and the worker modelled as a set of swept volumes. However, this algorithm has difficulties with occlusions, which are one of the significant problems with HRC scenarios. Indeed, when occlusions occur, the

information about the position of some human anatomical parts is no longer available. This can result in unnecessary limitations on the trajectories that the robot is allowed to follow.

With prediction-based algorithms, a robot can produce safe movements proactively by including predictions of the human's actions and motions, instead of relying on frequent replanning. This is especially useful for highly dynamic situations where replanning each time the environment changes is impractical. While prediction is helpful in ensuring safe HRC, it is essential to note that the efficacy of this approach is directly related to the accuracy of the chosen prediction algorithm. In other words, a poor choice of prediction algorithm will reduce the safety of the HRC.

2.4.3 Non-predictive Control-based Algorithms

Another common method for achieving safety during human-robot interaction is through non-predictive control (NPC) of robot motion. This type of safety provision is often the simplest method for enabling safe HRC, as it does not require complex prediction models or planners. Various techniques and methods designed to perform collision avoidance through non-predictive control will be reviewed in this section.

One popular approach for collision prevention via robot control is the calculation of danger criteria and fields, such as the potential field approach developed by Khatib (1985). This method can produce more sophisticated safety behaviors by defining a field of repulsive vectors that guide the robot's motion, modifying its trajectory in response to dynamically changing environmental factors. One work that integrated the potential field approach with other factors was developed by Kulić and Croft (2005). They incorporated a safety control module that considers safety factors such as separation distance and velocity to generate a danger index to be used by a potential field controller.

The estimated emotional state of the user, inferred from skin conductance and heart rate measurement, was also integrated into this danger index. Haddadin et al. (2010) similarly developed a collision avoidance technique based on the potential field method. Their approach accommodates not only the virtual forces caused by proximity to the robot, but also actual physical contact. The algorithm, designed to have sufficiently low complexity to run in real-time, is based on local reactive motion planning along with velocity scaling, a function not only of distance but also the direction of approach. The resulting system can produce smooth paths that avoid sudden accelerations and are thus more physically interpretable by humans. However, these papers assume that the information about the environment needed to avoid obstacles is already available, ignoring the need for sensors. Also, the efficacy of these methods is directly linked to the strategy used to construct the potential field.

Bosscher and Hedman (2011) presented a collision avoidance algorithm based on speed control. The algorithm searches for a joint velocity that most closely matches the desired one, in the sense of the quadratic Euclidean norm, subject to constraints on the relative speed between approaching bodies. Collision avoidance can be achieved by preprocessing joint velocity references inside of the tracking controller. The robot can avoid collisions with an obstacle moving at a maximum speed of approximately 0.5 m/s. However, since this only modifies the execution of motions, but not the planning, the robot reacts to the environment in a way that is not covered by the plan. Furthermore, as in Beckert et al. (2018), the robot will freeze and delay production if a human is blocking the predefined path.

Zanchettin et al. (2016) introduced a velocity scaling approach that takes advantage of redundant degrees of freedom, to maintain safety while retaining productivity. In this work, a safety region is calculated based on the robot's velocity and

braking distance, as well as a clearance parameter that takes uncertainties in measurement and modeling into account. The collision avoidance algorithm is calculated at the joint space level to allow for real-time deployment. It uses redundant degrees of freedom to move the robot's joints away from the human while still maintaining the right end effector position. This enables the robot to continue to perform its task while maintaining both a greater distance of separation from the human and a higher speed.

Liu and Tomizuka (2016) developed a method aimed at achieving real-time safety with formal guarantees employing set invariance theory and reachability analysis. In their work, the robot motion planning and control problem is posed as a constrained optimal control problem. A safety index, which depends on the relative distance among humans and robots, is evaluated using the ellipsoid coordinates attached to the robot links that represent the distance between the robot arm and the worker. The safety index is used as a constraint in the optimization problem so that a collision-free trajectory within a finite time horizon is generated on-line iteratively for the robot to move towards the desired position. To reduce the computational load for real-time implementation, the formulated optimization problem is further approximated by a quadratic program. The sampling time is 5.5 ms when the human and robot are modelled by ten SSLs and three SSLs, respectively. This method has the disadvantage of requiring a human tracking system. Furthermore, only the safety is considered, and not the robot's productivity.

2.4.4 Model Predictive Control-based Algorithms

Model predictive control (MPC) is another control-based method for HRC. MPC establishes a look ahead tracking controller by repeatedly solving an optimal control problem over a finite time horizon, and then applies the first element to the plant, which is the robot in this thesis. One advantage of MPC is that it can explicitly consider the

dynamics of low-level joint motion controllers in its prediction model. Like prediction-based methods, MPC algorithms predict the future motions of the human and robot. In general, MPC algorithms use simpler human prediction models and shorter prediction horizons. This allows them to recompute the optimal control solution based on current sensor information more frequently than with the re-planning approach. This makes them better suited to situations where the obstacle motions are less predictable.

Balan and Bone (2006) presented an MPC-like planning algorithm for HRC. Their algorithm solves an optimization problem to select the end-effector path direction from a finite set that balances between the robot rapidly approaching its target configuration and the robot maximizing its distance to the human over the prediction horizon. The robot's motion was predicted using a transfer function model of its time response at the joint level. The human's motion was predicted at the sphere level using the weighted mean of past velocities. As a test scenario, the authors developed a simulation of a human walking toward a moving Puma robot arm. The planning algorithm used a sampling period of 33.3 ms and a prediction horizon of 0.333 s. Captured human motion data was used to simulate the human's movements. They used Monte Carlo simulations, consisting of 1000 random human walking paths passing through the robot workspace, to validate their approach. Note that no experiments with a real robot were conducted.

Zube (2015) presented a collision avoidance strategy based on nonlinear MPC (NMPC) that is applicable to both fixed-base and mobile manipulators (i.e., a manipulator mounted on a mobile robot). NMPC is applied to solve the inverse kinematics of a redundant manipulator and control the end-effector pose in Cartesian space. It has the advantages that the nonlinear kinematics are not approximated by a linear model and hard constraints (e.g., joint limits) are included. A sampling period of 100 ms and a prediction horizon of 1 s were used with the NMPC. In terms of

disadvantages, this paper only presents simulation results that ignore modelling uncertainty, sensor noise and the robot's deadtime. No simulations of moving obstacles like humans were included either.

Another recent paper that uses NMPC for online trajectory optimization was presented by Krämer et al. (2020). The NMPC problem is solved by direct collocation based on a hypergraph structure which allows it to efficiently adapt to structural changes in the optimization problem caused by moving obstacles. The proposed algorithm has the further advantage that it incorporates constraints on joint velocities into the motion planning. It has the disadvantage that is assumes the obstacles are stationary over the prediction horizon. Simulation results for "imitated robotic pick-and-place experiments" are included. A sampling period of 100 ms and a 2.5 s prediction horizon were used. The results show that the MPC controller allows the robot to successfully approach a moving target configuration without prior knowledge of the reference motion. The simulated obstacles consisted of a single moving sphere and a static human modelled by seven SSLs. Their simulations ignore the difficulties caused by modelling uncertainty and sensor noise.

2.5 Summary

This chapter began with a review of the different geometric representations of the robot and human used in the literature. The complexity of the geometric representations is related to the accuracy of the models and computational costs directly. For real-time applications such as robot collision avoidance, the computational efficiency of the geometric model must be regarded as the first priority.

The state-of-the-art literature on collision avoidance algorithms was also reviewed. Motion planning-based methods provide a proactive approach to ensuring safety.

However, in HRC applications these planners are challenged by having to rapidly re-plan new paths and motions due to the dynamic nature of any environment occupied by people. Prediction-based motion planning methods are especially useful for highly dynamic situations, but their effectiveness is directly related to the accuracy of the obstacle prediction algorithms. The performance of collision avoidance algorithms employing NPC and MPC benefits from their more frequent use of sensor feedback compared to motion planning-based methods. This makes them more effective for applications where the obstacle motions are less predictable, such as HRC.

Chapter 3. Human and Robot Modelling

3.1 Introductions

Geometric models of the human and the robot manipulator are necessary to implement the collision avoidance algorithms. In this thesis, three geometric primitives are used to represent objects. The front of the human's torso is represented by a plane. A plane is also used to represent the table the robot is mounted on. The human's arm(s) (and head if their neck is bending forwards) that protrude in front of the torso plane are modeled as by unions of spheres. The links of robotic manipulator is represented by SSLs. These representations have been chosen because they produce simple models that can be used to quickly calculate the distance between any two objects.

3.2 Human Modelling Algorithm

3.2.1 Algorithm Description

In this work, a point cloud approach for human detection is applied. We employ an Intel RealSense D415 RGB-D sensor to acquire the point cloud. This sensor produces two images: a RGB colour image (captured by a colour camera) and a depth image (from a depth camera). The depth image is a two-dimensional image with integer valued pixels that measure the depth of that point in space (in millimeters measured along the camera's optical axis). The depth camera can be modelled by the pinhole camera model (Darwish et al., 2019) as shown in Fig. 3.1.

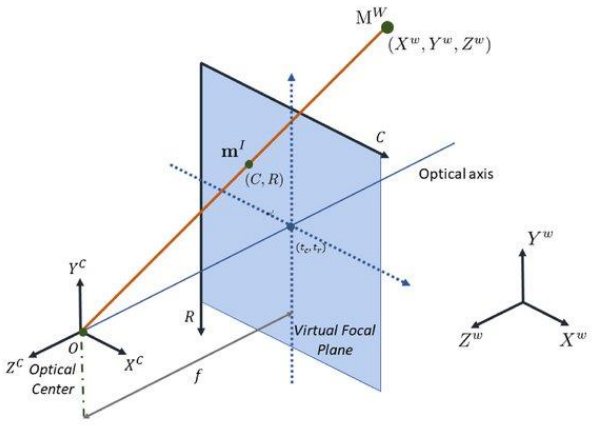


Figure 3.1: Depth camera model (Darwish *et al.*, 2019)

The two-dimensional depth image information must be converted into the three-dimensional point cloud, where each pixel of the depth image has a corresponding point in the point cloud.

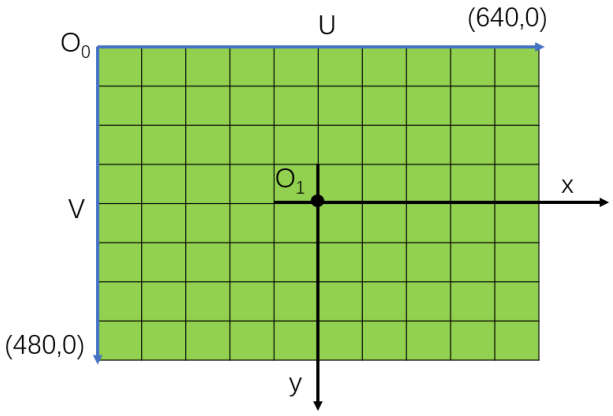


Figure 3.2: Imaging plane coordinate system

As shown in Fig. 3.2, (u, v) is the image coordinate system expressed in pixels, while (x, y) represents the camera's Cartesian coordinate system expressed in

millimeters. Then the relationship between two coordinate systems is as follows (Chu et al., 2019):

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} 1/d_x & 0 & u_0 \\ 0 & 1/d_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.1}$$

where u_0 is the origin of the x-axis in the image coordinate system, v_0 is the origin of the y-axis in the image coordinate system, d_x is the resolution in the x-axis direction, d_y is the resolution in the y-axis direction. The equation for converting each point in the depth image data to point cloud data is then (Chu et al., 2019):

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} d \\ (u_x - u_0)d / d_x \\ (v_y - v_0)d / d_y \end{bmatrix} \tag{3.2}$$

where d is the depth value of the point (ux, vy) from the depth image. In this thesis, the depth image resolution used is 640×480. So $u_0 = 320$ and $v_0 = 240$ in the above equations.

In order to segment out the point cloud that belongs to the human, a method combining human skin colour detection and point cloud clustering is used. Each point has six channels. Three channels store x, y, and z location information while the other three store red, green, and blue colour information. The processes of human skin colour detection and points cloud clustering are explained below.

The objective of human skin detection is to find the skin regions in an image, which requires separating the skin and non-skin points (Shaik et al., 2015). Colour space is a mathematical model that represents colour information as different colour components.

A variety of colour spaces are available for skin detection, including: RGB colour space, Hue-based colour space, Luminance-based colour space, and perceptually uniform colour space. RGB colour space is not preferred for colour-based object detection and colour analysis because of the mixing of the colour (chrominance) and intensity (luminance) information and its nonuniform characteristics (Zarit et al., 1999). Luminance-based approaches discriminate colour and intensity information even under uneven illumination conditions. We apply the Luminance-based colour space named YCbCr to perform skin colour detection and segmentation. The transformations between YCbCr colour space and RGB colour space are given by:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{3.3}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.4 \\ 1 & -0.343 & -0.711 \\ 1 & 1.765 & 0 \end{bmatrix} \begin{bmatrix} Y \\ Cb-128 \\ Cr-128 \end{bmatrix} \tag{3.4}$$

Skin detection based on YCbCr colour space can be done using by thresholding the three colour components. When an RGB colour image is transformed into the YCbCr colour image, the resultant image is comprised of an intensity component (Y) and chrominance components (Cb and Cr). In this thesis, thresholds are only applied to the chrominance components as shown by the following pseudocode:

```
if (77 ≤ Cb ≤ 230) and (140 ≤ Cr ≤ 165) then
  Point is skin colour
else
  Point is not skin colour
end
```
(3.5)

We apply human skin colour detection to monitor the volume of interest, which is the workspace of the robot. In the next step, the first point in the human skin colour area will be used as the first seed point to form the cluster of points belonging to the human body by applying a region growing method.

The basic idea of region growing is to group points with similar properties to form regions. Specifically, first, we find a seed pixel for each region that needs to be segmented as the starting point for growing. Then we merge the points in the neighborhood around the seed point that have the same or similar properties, determined according to a certain predetermined growing or similarity criterion, as the seed point into the area where the seed point is located. Use these new points as new seed points and continue the above process until no more points that meet the conditions can be included. In this way, the growing of the region is complete (Adams and Bischof, 1994).

Principal component analysis (PCA) has been widely used to estimate local saliency features (SFs) that are used in the region growing (Rabbani et al., 2006). In this work, PCA is used to estimate the normal and curvature of points. The region growing process consists of four basic tasks, as shown in Figure 3.3.

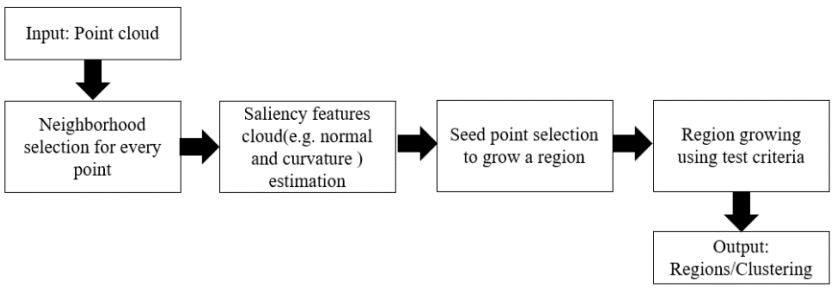


Figure 3.3: Region growing process (Rabbani *et al.*, 2006).

Neighborhood selection is an important task for accurate SFs estimation. There are three common methods for neighborhood selection: 1) k-NN ; 2) fixed-distance neighborhood; and 3) neighborhood within a voxel. For (2) and (3), the numbers of points in a neighborhood are different due to uneven sampling. We apply k-NN to get k points of a local neighborhood for an interesting point based on the k-D search algorithm (Xiao and Wenming, 2009). This approach can deal well with uneven point densities, which is common in points cloud data. By manual tuning, we found that $k = 10$ works well for estimating the surface normal. For the second task – SFs estimation, the normal and curvature are estimated for all the points in the data using principal components analysis (PCA). The 3×3 covariance matrix $\mathbf{C}$ of each point $\mathbf{p}_i$ is calculated by the following equation (Khaloo and Lattanzi, 2017):

$$\mathbf{C} = \frac{1}{k} \sum_{j=1}^{n} (\mathbf{p}_j - \mathbf{p}_i)^T (\mathbf{p}_j - \mathbf{p}_i) \tag{3.6}$$

where k is the number of nearest neighbours and $\mathbf{p}_j$ are the k nearest neighbours of $\mathbf{p}_i$.

Next, by performing Singular Value Decomposition (SVD) on the covariance matrix (Golub and Reinsch, 1970), it is possible to compute the eigenvectors $\mathbf{v_3}, \mathbf{v_2}$ and $\mathbf{v_1}$ and the corresponding eigenvalues $\lambda_1, \lambda_2,$ and λ_3 of the covariance matrix $\mathbf{C}$. The eigenvalues are sorted in the order: $\lambda_3 \geq \lambda_2 \geq \lambda_1$. With this approach $\mathbf{v}_1$ approximates the normal vector at point $\mathbf{p}_i$. The curvature is given by:

$$c = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}, \quad \lambda_3 \geq \lambda_2 \geq \lambda_1 \tag{3.7}$$

The first point that is recognized in the human skin area is selected as the first seed point. A region is grown incrementally based on the spatial connectivity among the points starting from the seed point. We define two points as spatially connected based on their normal vectors and curvatures. The region growing algorithm for finding the "human body point cloud" is then:

Algorithm 3.1 The complete region growing method to cluster human body cloud

Inputs: Point cloud $\{\mathbf{P}\}$, User-defined neighborhood size k, angular threshold $=\theta_{threshold}$ and curvature threshold$=c_{threshold}$.

Output: Human body point cloud $\{\mathbf{H}\}$.

1. Initialize: current region $\{\mathbf{R_c}\}$, and cloud of current seeds $\{\mathbf{S_c}\}$;
2. Find $k-\text{NN}$ for all points in $\{\mathbf{P}\}$ using the Matlab command:

   $\mathbf{N_p} = \text{knnsearch}(\mathbf{P},\mathbf{P},k)$;
3. PCA-based normal and curvature estimation for all points in $\{\mathbf{P}\}$:

   for $i=1$ to size $\{P\}$ **do**

   Denote NN of i^{th} point as $\mathbf{N_{p_i}}$;

Calculate the covariance matrix $\mathbf{C}$ using $\mathbf{N}_{\mathbf{p_i}}$ and (3.6);

Calculate eigenvectors $\mathbf{V}\,(\mathbf{v}_3, \mathbf{v}_2$ and $\mathbf{v}_1)$ and their corresponding eigenvalues λ $(\lambda_3 \geq \lambda_2 \geq \lambda_1)$ of $\mathbf{C}$: $[\mathbf{V}, \lambda] = eig(\mathbf{C})$;

Normal vector: $\mathbf{n}_i = \mathbf{v}_1$;

Curvature c_i is calculated by Equation (3.7);

end

4. **while** $\{\mathbf{P}\}$ is not empty, **do**

5. Select the first point in human skin area $\mathbf{h}_1$ as initial seed $\mathbf{h}_1 \rightarrow \mathbf{p}_i$;

6. $\mathbf{p}_i$ **insert** $\{\mathbf{R}_c\}$ & $\{\mathbf{S}_c\}$

7. $\mathbf{p}_i$ **remove** $\{\mathbf{P}\}$

8. **for** $m = 1$ to k **do**

9. Calculate angle from $\mathbf{n}_m$ to the seed point normal vector $\mathbf{n}_{p_i}$: $\theta_m = \arccos\left|\mathbf{n_m} \bullet \mathbf{n_{p_i}}\right|$;

10. **if** $\theta_m \leq \theta_{threshold}$ **do**

11. $\mathbf{p}_m$ *insert* $\{\mathbf{R}_c\}$

12. $\mathbf{p}_m \underrightarrow{remove} \{\mathbf{P}\}$

13. **if** $abs(c_{p_m} - c_{p_i}) \leq c_{threshold}$ **do**

14. $\mathbf{p}_m \underrightarrow{insert} \{\mathbf{S}_c\}$

15. end

16. end

17. **end**

18. Add current region to human body point cloud:
$\{\mathbf{R}_c\} \underrightarrow{insert} \{\mathbf{H}\}$

19. end

With point clouds, the computation time increases with the number of points. To reduce the number of points the human body point cloud is resampled using a process called voxelization. A voxel is a cube with a predefined edge length. This edge length defines the resolution of the voxelized cloud. Figure 3.4 is an illustration of a voxel cell, where L is the edge length of the voxel cell.

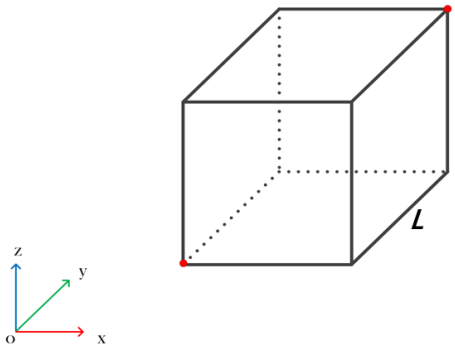


Figure 3.4: Voxel cell.

The voxelization step is shown in Figure 3.5. First, the bounding box is calculated for original point cloud. Second, using the predefined voxel cell size, the bounding box of the point cloud is divided into cubic cells. Third, the cloud points are assigned to the cells they are inside. Note that many of the clouds will be unoccupied. Finally, the new resampled point cloud is given the centroids of the occupied voxel cells. Since several of the original points may lie inside each voxel cell (as shown in Figure 3.5) the resampled cloud typically includes much fewer points than the original cloud.

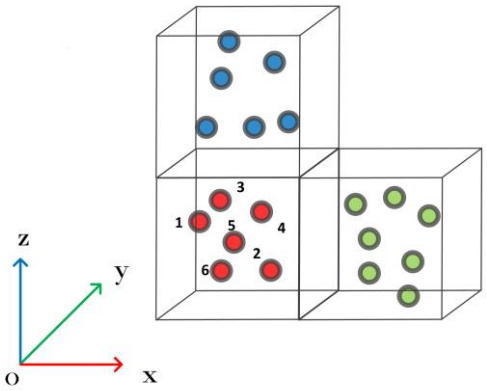


Figure 3.5: Point cloud resampling using voxel cells.

After resampling the human body point cloud, we fit a plane to the front of the torso and fit spheres to human's arm(s) (and head if their neck is bending forwards). We apply the built-in MATLAB function "*pcfitplane*" to fit a plane to the point cloud. To use this function, the maximum allowable distance from an inlier point to the plane must be specified. This function is motivated by the Maximum Likelihood Estimator SAmple Consensus (MLESAC) algorithm, which is a variant of the RANdom SAmple Consensus (RANSAC) algorithm. The MLESAC algorithm is a generalization of RANSAC, which adopts the same sampling strategy but attempts to maximize the

likelihood of the solution, as opposed to the number of inliers (Torr and Zisserman, 2000). MLESAC is useful in situations where most of the data samples belong to the model, and a fast outlier rejection algorithm is needed. We use this function to return the equation for the plane, the linear indices to the inlier and outlier points in the point cloud input. In our purpose, the inlier points are the torso (and head if the neck is not bending forwards), while the outlier points belong to the protruding human arm(s) (and head if the neck is bending forwards). The algorithm for fitting the union of spheres to the outlier points is:

Algorithm 3.2 Union of spheres fitting algorithm

Inputs: Outlier point cloud $\{\mathbf{P}_o\}$ from pcfitplane, user-defined sphere radius R_s, plane safety tolerance d_{pst} and minimum points number $N_{\min}$;

Output: Centroids matrix of fitting spheres $\mathbf{C_s}$;

1. **initialization** Max distance $d_{\max} = abs(d_{pst}) + 10^{-6}$, centroids matrix $\mathbf{c_s}$ and centroid input c_{in};
2. **while** $d_{\max} > abs(d_{pst})$ **do**
3. Calculate distances between $\{P_o\}$ and the torso plane $\mathbf{D_{toplane}}$;
4. Max distance $d_{\max} = \max(\mathbf{D_{toplane}})$;

5. **if** $d_{\max} < abs(d_{pst})$ **do**
6. **break**;
7. **end**
8. Find the point has with max distance $p_{\max}$;
9. $c_{in} = p_{\max}$;
10. Previous number of points in sphere $n_{previous} = 0$;
11. Current number of points in sphere $n_{current} = 1$;
12. **while**

size $\{P_c\} > N_{\min}$ & $norm(p_{\max} - c_{in}) < R_s$ & $n_{previous} - n_{current} < 0$ **do**

13. $n_{previous} = n_{current}$;
14. Find neighbors of c_{in}:

$\{P_{nr}\} = \text{findNeighborsInRadius}(\{P_o\}, c_{in}, R_s)$;

15. $n_{current} = size\{P_{nr}\}$;
16. Find current cloud in sphere: $\{P_{is}\} = \{P_{nr}\}$;
17. $c_{in} = mean\{P_{is}\}$;
18. **end**;
19. **if** $n_{current} > N_{\min}$ **do**
20. c_{in} $\underline{insert}$ $\mathbf{C_s}$;
21. **end**
22. **if** $n_{current} \le N_{\min}$ **do**
23. **break;**
24. **end**;
25. **end**

3.3 Robot Modeling

3.3.1 Kinematic Model

A serial robotic manipulator can be represented as a chain of links (rigid bodies) connected to each other by moveable joints. A kinematic model is used to define the relationship between the positions and orientations of the links and the values of the joint angles. In this research, a collaborative robot, Elfin 5 (Figure 3.6) made by Han's Robot, will be used for simulating the collision avoidance algorithms. Elfin is a 6-DOF robotic manipulator with six rotational joints (also known as revolute joints). The robot has a maximum payload of 5 kg.

Figure 3.6: The Elfin 5 robot made by Han's Robot.

Kinematics is the study of the movements without considering the causes that give rise to them, so it only involves distances, angles, velocities and accelerations (Todd, 1986). The position and orientation of each link in space is defined by a Cartesian coordinate frame attached to it. Although there are several ways to define one coordinate frame relative to another, the Denavit-Hartenberg (D-H) representation has become the standard in robotics (Hartenberg and Denavit, 1964). Its parameters are the joint angle

θ_i, joint offset d_i, link length a_i and link twist α_i. This representation simplifies the kinematic model by imposing limits on the definitions of the coordinate frames, as described in on page 31 of (Spong *et al.*, 2006). The corresponding D-H parameters of the Elfin 5 robot can be found in Table 3.1. Note that a 100 mm offset for a gripper is taken into consideration.

Table 3.1: D-H parameters of the Elfin 5 robot

Link i	θ_i [rad]	d_i [mm]	a_i [mm]	α_i [rad]
1	θ_1	220	0	$\pi/2$
2	θ_2	100	380	π
3	θ_3	-100	0	$\pi/2$
4	θ_4	420	0	$-\pi/2$
5	θ_5	100	0	$\pi/2$
6	θ_6	280	0	0

There are two branches of manipulator kinematics, namely: forward kinematics, and inverse kinematics. The first one is used for calculating the Cartesian positions and orientations of the end-effector or intermediate links/joints and from a given values of the joint angles.

The forward kinematics of the Elfin 5 robot will be described first. A 4×4

homogeneous transformation matrix may be used to define the position and orientation changes between a pair of coordinate frames. It is composed of a rotation matrix R and a position vector p as follows:

$$F = \begin{bmatrix} & R & & p \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

The D-H parameters can be used to a derive a set of six transformation matrices, one for each of the robot's joints. The transformation matrices defining the change from the coordinate frame for link i to the coordinate frame for link $i+1$, is written: ${}^{i+1}T_i$. It is obtained using (3.9), where $C\theta_{i+1}$ and $S\theta_{i+1}$ stand for $\cos\theta_{i+1}$ and $\sin\theta_{i+1}$.

$${}^{i+1}T_i = \begin{bmatrix} C\theta_{i+1} & -S\theta_{i+1}C\alpha_{i+1} & S\theta_{i+1}S\alpha_{i+1} & a_{i+1}C\theta_{i+1} \\ S\theta_{i+1} & C\theta_{i+1}C\alpha_{i+1} & -C\theta_{i+1}S\alpha_{i+1} & a_{i+1}S\theta_{i+1} \\ 0 & S\alpha_{i+1} & C\alpha_{i+1} & d_{i+1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

The concatenation of the multiple links can be done by multiplying the individual transformation matrices, resulting in one compound transformation matrix. For example, computing the location and orientation of the end-effector with respect to the robot base frame can be done by multiplying every link transformation of the robotic arm as in (3.10). This results in a homogeneous transformation matrix where the positions vector describes the position of the tool and the rotation matrix the orientation of the tool with respect to the base coordinate system. For a 6-DOF robot like Elfin, the total transformation matrix representing the forward kinematics equals:

$${}^{0}T_6 = {}^{0}T_6 \bullet {}^{1}T_2 \bullet {}^{3}T_3 \bullet {}^{3}T_4 \bullet {}^{4}T_5 \bullet {}^{5}T_6 \tag{3.10}$$

The goal of inverse kinematics is to find the joint angles from the given desired position and orientation of the end-effector. There are two main strategies for solving the inverse kinematics of a manipulator: closed-form solutions and numerical solutions. Closed-form solutions use an analytical approach and can be based on geometrical properties. Numerical solutions use iterative techniques to find the desired joint angles but require more computational time (Craig, 2004). Because it is desired to keep the calculation time as low as possible in real-time interactions, it was chosen to formulate a closed-form solution for the Elfin 5. This solution is based on section 4.4 of (Craig, 2004).

To begin the solution, the desired position and orientation of the end-effector is specified using the transformation matrix for coordinate frame 6 as defined by:

$$ {}^{0}T_{6} = \begin{bmatrix} n_{6x} & o_{6x} & a_{6x} & p_{6x} \\ n_{6y} & o_{6y} & a_{6y} & p_{6y} \\ n_{6z} & o_{6z} & a_{6z} & p_{6z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11} $$

First, we can determine p_5 by moving a distance d_6 along the z-axis of coordinate system 6. The direction of this z-axis is indicated by the third column of the rotation matrix portion of ${}^{0}T_{6}$. This gives:

$$ p_5 = \begin{bmatrix} p_{5x} \\ p_{5y} \\ p_{5z} \end{bmatrix} = p_6 - d_6 \bullet \begin{bmatrix} a_{6x} \\ a_{6y} \\ a_{6z} \end{bmatrix} \tag{3.12} $$

After p_5 has been calculated, it is possible to calculate θ_1, θ_2 and θ_3 with the following equations:

$$\theta_1 = \text{atan2}(p_{5y}, p_{5x}) - \pi \tag{3.13}$$

$$\cos\theta_3 = \frac{{p_{5x}}^2 + {p_{5y}}^2 + (p_{5z} - d_1)^2 - {a_2}^2 - {d_4}^2}{2a_2 d_4} = U \tag{3.14}$$

and

$$\theta_3 = \text{atan2}(\pm\sqrt{1-U^2}, U) \tag{3.15}$$

where the positive square root gives the elbow down solution and the negative square root gives the elbow up solution. The corresponding solutions for angle θ_2 are given by:

$$\theta_2 = \text{atan2}\left(\frac{p_{5z} - d_1}{\sqrt{{p_{5x}}^2 + {p_{5y}}^2 + (p_{5z} - d_1)^2}}, \frac{\sqrt{{p_{5x}}^2 + {p_{5y}}^2}}{\sqrt{{p_{5x}}^2 + {p_{5y}}^2 + (p_{5z} - d_1)^2}}\right) - \text{atan2}\left(\frac{d_4 \sin\theta_3}{\sqrt{{p_{5x}}^2 + {p_{5y}}^2 + (p_{5z} - d_1)^2}}, \frac{a_2 + d_4\cos\theta_3}{\sqrt{{p_{5x}}^2 + {p_{5y}}^2 + (p_{5z} - d_1)^2}}\right) \tag{3.16}$$

The equations for the inverse orientation kinematics may be used to calculate the values of the wrist's joint angles (i.e., the minor axes). The method begins by first calculating the orientation of the end-effector relative to frame 3 as follows:

$${}^3R_6 = {}^0R_3^{-1} \bullet R = {}^0R_3^T \bullet \mathrm{R} = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \tag{3.17}$$

where ${}^0\mathrm{R}_3$ can be calculated using forward kinematics and the joint variables we found using the inverse position kinematics equations. The Elfin 5 is equipped with a Euler wrist, as is common with industrial robots. For a Euler wrist, the rotation matrix

is:

$$ {}^{3}R_{6} = \begin{bmatrix} C\theta_4 C\theta_5 C\theta_6 - S\theta_4 S\theta_6 & -C\theta_4 C\theta_5 S\theta_6 - S\theta_4 C\theta_6 & C\theta_4 S\theta_5 \\ S\theta_4 C\theta_5 C\theta_6 + C\theta_4 S\theta_6 & -S\theta_4 C\theta_5 S\theta_6 + C\theta_4 C\theta_6 & S\theta_4 S\theta_5 \\ -S\theta_5 C\theta_6 & S\theta_5 C\theta_6 & C\theta_5 \end{bmatrix} \quad (3.18) $$

Equating (3.17) and (3.18) gives the matrix equation

$$ \begin{bmatrix} C\theta_4 C\theta_5 C\theta_6 - S\theta_4 S\theta_6 & -C\theta_4 C\theta_5 S\theta_6 - S\theta_4 C\theta_6 & C\theta_4 S\theta_5 \\ S\theta_4 C\theta_5 C\theta_6 + C\theta_4 S\theta_6 & -S\theta_4 C\theta_5 S\theta_6 + C\theta_4 C\theta_6 & S\theta_4 S\theta_5 \\ -S\theta_5 C\theta_6 & S\theta_5 C\theta_6 & C\theta_5 \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} $$

(3.19)

The remainder of the method involves solving for angles θ_4, θ_5 and θ_6 using the known u values.

3.3.2 Geometric Model

Based on Elfin 5 robot's kinematic configuration, only links 2, 4, and 6 are modeled as SSLs for use in the collision avoidance algorithm. The reasons become clear by examining Figure 3.7 and Table 3.1. Link 1 is not modelled since it is stationary, and links 3 and 5 have zero length. As a result, the complete geometric model of the robot, shown in Figure 3.8, may be obtained from the three SSL line segment endpoints and their radii. In this thesis, a radius of 40 mm is used for SSLs of link 6 while a radius of 50 mm is for link 4 and link 2.

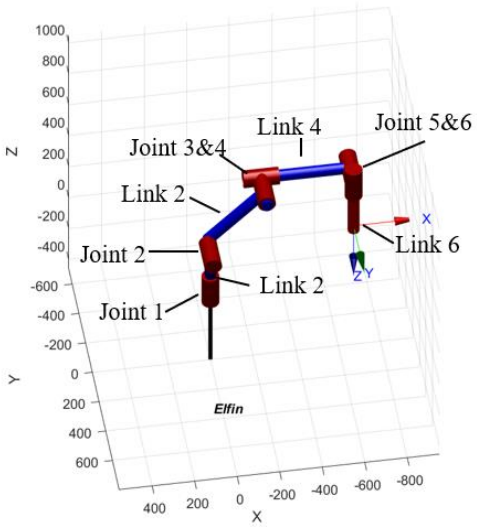


Figure 3.7: The kinematic skeleton of the Elfin 5 robot

Figure 3.8: SSLs model of the moving links of the Elfin 5 robot

3.3.3 Time Domain Model

In practice, all robots have some delay in their response due to the dynamics of their electrical and mechanical components. The joint position command sent at discrete time t will not be executed by the robot at discrete time t since the robot cannot reach commanded locations instantly. The delay required for the robot to execute a command results in the following equation:

$$\theta_a(t+1) = \theta_c(t+1-\tau_d) \tag{3.20}$$

where θ_c is the commanded joint angle, θ_a is the actual joint angle, and τ_d is

the number of sampling periods of delay. Equation (3.20) applies to a single joint only, but can be extended to model the entire robot by considering θ_a and θ_c as column vectors, of length equal to the number of the robot joints.

The delay τ_d of the Elfin 5 robot was obtained experimentally. Each joint was tested separately. The set of commands sent to the robot are moving one joint 45 degrees and then back to the initial position. The commands and corresponding actual joint positions are recorded. Figure 3.9 is a zoomed-in plot of the joint 1 test results.

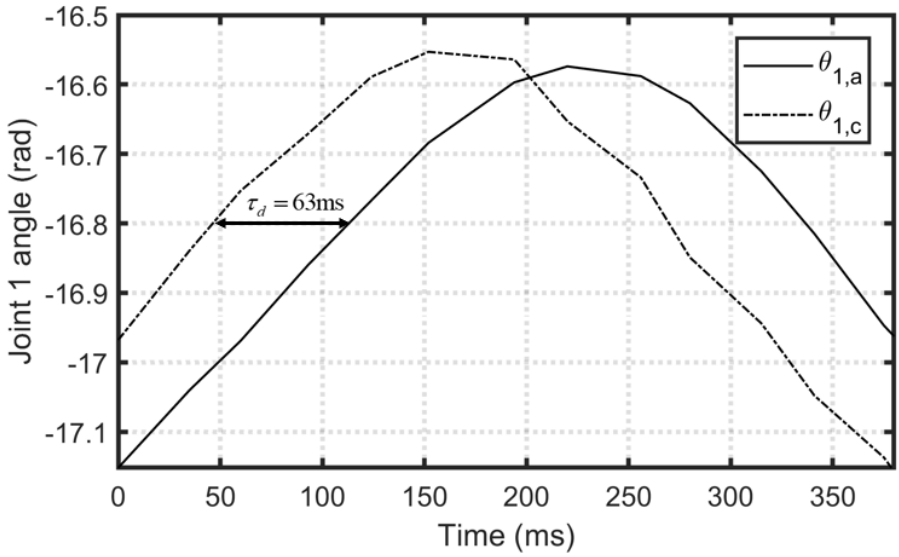


Figure 3.9: Joint 1 delay measurement experimental results for the Elfin 5 robot (zoomed-in to show the delay τ_d).

From these experiments, the inherent delay from the time the robot is ordered to take actions until it begins to execute the order is 63 ms, 55 ms, 54 ms, 63 ms, 54 ms and 63 ms for joint 1, joint 2, joint 3, joint 4, joint 5 and joint 6 respectively. In this

thesis a sampling period of 33.3 ms is used to match the 30 Hz sampling frequency of the Intel RealSense RGB-D sensor. To be conservative the maximum experimentally measured delay of 63 ms is approximated as a delay of two sampling periods (*i.e.*, 66.6 ms) in our simulations.

In addition to the delay, the position, velocity and acceleration limits also must be taken into consideration. The following inequalities should be satisfied:

$$\theta_{c,\min} \le \theta_c \le \theta_{c,\max} \tag{3.21}$$

$$\dot{\theta}_{c,\min} \le \dot{\theta}_c \le \dot{\theta}_{c,\max} \tag{3.22}$$

$$\ddot{\theta}_{c,\min} \le \ddot{\theta}_c \le \ddot{\theta}_{c,\max} \tag{3.23}$$

where $\theta_{c,\min}$ is the minimum joint angle limit, $\theta_{c,\max}$ is the maximum joint angle limit, $\dot{\theta}_{c,\min}$ is the minimum joint velocity limit, $\dot{\theta}_{c,\max}$ is the maximum joint velocity limit, $\ddot{\theta}_{c,\min}$ is the minimum joint acceleration limit and $\ddot{\theta}_{c,\max}$ is the maximum joint acceleration limit.

3.4 Summary

This chapter presented the human modeling and robot modeling approaches used in this research. The algorithms that generate the human plane and union of spheres model from RGB-D data were described. Bounding SSL models for the robot links can be calculated from the presented kinematic equations, the sensed joint angles, and the SSL radii. The SSL link models, and the plane and spheres human model, will be used to conservatively estimate the human-robot closest distance by calculating the closest

distance between those geometric objects in chapters 4 and 5. To properly simulate the robot's time response in chapter 5, a time domain model that includes the delay time was found using experimental data.

Chapter 4 Collision Avoidance Algorithms

4.1 Introduction

In this chapter, the collision avoidance algorithms are presented in detail. The first algorithm solves the inverse kinematics problem and avoid collisions using an expanded version of the manipulator Jacobian matrix. A damped least-squares inverse of the Jacobian is applied to compute the commanded joint velocities from given end-effector velocities. The Cartesian task space error is corrected by adopting a closed-loop inverse kinematics algorithm. Collision avoidance tasks produce constraints that are systematically incorporated into this algorithm. A second collision avoidance algorithm using NMPC is designed as an alternative approach.

4.2 Inverse kinematics calculation

For a 6 DOF robot, the direct kinematics equation of a robot manipulator with an open kinematic chain can be written in the form:

$$\mathbf{x} = f(\mathbf{q}) \tag{4.1}$$

where $\mathbf{q}$ is the (6×1) vector of joint position variables, $\mathbf{x}$ is the (6×1) vector of end-effector variables (including position variables and orientation variables), and $f(\cdot)$ denotes the nonlinear direct kinematics function.

The Jacobian matrix gives the relationship between the joint velocities and the corresponding end-effector linear and angular velocities. The calculations for the Jacobian matrix presented in this section are mainly based on pages 105-113 of

(Siciliano, 2009). It is desired to express the end-effector linear velocity $\dot{\mathbf{p}}_e$ and angular velocity $\boldsymbol{\omega}_e$ as a function of the joint velocities $\dot{\mathbf{q}}$, i.e.,

$$\dot{\mathbf{p}}_\mathbf{e} = \mathbf{J}_\mathbf{p}(\mathbf{q})\dot{\mathbf{q}} \tag{4.2}$$

$$\boldsymbol{\omega}_\mathbf{e} = \mathbf{J}_\mathbf{o}(\mathbf{q})\dot{\mathbf{q}} \tag{4.3}$$

In (4.2), $\mathbf{J}_\mathbf{p}$ is the (3×6) matrix relating the contribution of the joint velocities $\dot{\mathbf{q}}$ to the end-effector linear velocity $\dot{\mathbf{p}}_\mathbf{e}$, while in (4.3) $\mathbf{J}_\mathbf{o}$ is the (3×6) matrix relating the contribution of the joint velocities $\dot{\mathbf{q}}$ to the end-effector angular velocity $\dot{\boldsymbol{\omega}}_\mathbf{e}$. $\mathbf{J}_\mathbf{p}$ and $\mathbf{J}_\mathbf{o}$ can be partitioned into the (3×1) column vectors $\mathbf{J}_{\mathbf{p}i}$ and $\mathbf{J}_{\mathbf{o}i}$ as follows:

$$\mathbf{J}_\mathbf{p} = \begin{bmatrix} \mathbf{J}_{\mathbf{p}1} \cdots \mathbf{J}_{\mathbf{p}i} \cdots \mathbf{J}_{\mathbf{p}6} \end{bmatrix} \tag{4.4}$$

$$\mathbf{J}_\mathbf{o} = \begin{bmatrix} \mathbf{J}_{\mathbf{o}1} \cdots \mathbf{J}_{\mathbf{o}i} \cdots \mathbf{J}_{\mathbf{o}6} \end{bmatrix} \tag{4.5}$$

In compact form, (4.2) and (4.3) can be rewritten as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}}_\mathbf{e} \\ \boldsymbol{\omega}_\mathbf{e} \end{bmatrix} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{4.6}$$

where $\mathbf{J(q)}$ is the (6×6) manipulator geometric Jacobian matrix. In expanded form it is:

$$\mathbf{J(q)} = \begin{bmatrix} \mathbf{J}_\mathbf{p}(\mathbf{q}) \\ \mathbf{J}_\mathbf{o}(\mathbf{q}) \end{bmatrix} = \begin{bmatrix} \mathbf{J}_{\mathbf{p}1} \cdots \mathbf{J}_{\mathbf{p}i} \cdots \mathbf{J}_{\mathbf{p}6} \\ \mathbf{J}_{\mathbf{o}1} \cdots \mathbf{J}_{\mathbf{o}i} \cdots \mathbf{J}_{\mathbf{o}6} \end{bmatrix} \tag{4.7}$$

A simple and systematic way to calculate $\mathbf{J(q)}$ using direct kinematics relations

is shown in the following equation:

$$\begin{bmatrix} \mathbf{J}_{pi} \\ \mathbf{J}_{oi} \end{bmatrix} = \begin{cases} \begin{bmatrix} \mathbf{z}_{i-1} \\ \mathbf{0} \end{bmatrix} & \text{for a prismatic joint} \\ \begin{bmatrix} \mathbf{z}_{i-1} \times (\mathbf{p}_e - \mathbf{p}_{i-1}) \\ \mathbf{z}_{i-1} \end{bmatrix} & \text{for a revolute joint} \end{cases} \tag{4.8}$$

where $\mathbf{z}_{i-1}$ is the z-axis of the ith joint frame, $\mathbf{p}_{i-1}$ is the position of the origin of the ith joint frame, and $\mathbf{p}_e$ is the position of the end-effector.

In (4.8), vectors $\mathbf{z}_{i-1}$, $\mathbf{p}_e$, and $\mathbf{p}_{i-1}$ are all functions of the joint variables. In particular:

$\mathbf{z}_{i-1}$ is given by the third column of the rotation matrix ${}^0\mathbf{R}_{i-1}$, i.e.,

$$\mathbf{z}_{i-1} = {}^0\mathbf{R}_1(q_1)\ldots{}^{i-2}\mathbf{R}_{i-1}(q_{i-1})\mathbf{z}_0 \tag{4.9}$$

where $\mathbf{z}_0 = [0 \quad 0 \quad 1]^T$ allows the selection of the third column.

$\mathbf{p}_e$ is given by the first three elements of the fourth column of the transformation matrix ${}^0\mathbf{T}_e$, i.e., by expressing $\tilde{\mathbf{p}}_e$ in the (4 × 1) homogeneous form:

$$\tilde{\mathbf{p}}_e = {}^0\mathbf{T}_1\ldots{}^5\mathbf{T}_6\tilde{\mathbf{p}}_0 \tag{4.10}$$

where $\tilde{\mathbf{p}}_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$ allows the selection of the fourth column.

$\mathbf{p}_{i-1}$ is given by the first three elements of the fourth column of the transformation matrix ${}^0\mathbf{T}_{i-1}$, i.e., it can be extracted from:

$$\tilde{\mathbf{p}}_{i-1} = {}^{0}\mathbf{T}_{1}\ldots{}^{i-2}\mathbf{T}_{i-1}\tilde{\mathbf{p}}_{0} \tag{4.11}$$

The above relation in (4.6) can be inverted to provide the so-called Jacobian control method for manipulators:

$$\dot{\mathbf{q}}_{c} = \mathbf{J}^{-1}(\mathbf{q})\dot{\mathbf{x}} \tag{4.12}$$

where the "c" subscript refers to the commanded value rather than the actual value. The pseudoinverse method (see (Whitney, 1969)) for the least-squares minimum-norm gives the following solution:

$$\dot{\mathbf{q}}_{c} = \mathbf{J}^{*}\dot{\mathbf{x}} = \mathbf{J}^{T}(\mathbf{J}\mathbf{J}^{T})^{-1}\dot{\mathbf{x}} \tag{4.13}$$

Equation (4.13) is preferable to be used as the control law if the Jacobian matrix has full row rank. However, there always are some configurations at which the Jacobian is either rank deficient or ill-conditioned. Singularity will occur at these configurations. This problem will be overcome using the following damped least-squares method (see (Wampler and Leifer, 1988)):

$$\dot{\mathbf{q}}_{c} = (\mathbf{J}^{T}\mathbf{J} + \lambda\mathbf{I}_{n})^{-1}\mathbf{J}^{T}\dot{\mathbf{x}} \tag{4.14}$$

where $\mathbf{I}_{n}$ is the n dimension identity matrix (with n equal to the number of columns of $\mathbf{J}$), and $\lambda > 0$ is a scalar damping factor.

4.3 Inclusion of Constraints

4.3.1 Sphere obstacle collision avoidance constraint

In real applications, many constraints (e.g. the finite torque of joint) can stop the realization of the basic inverse kinematics law (4.14). In this thesis, we assume the

constraints on the manipulator are unilateral and can be represented as strict inequalities, namely

$$f_i > 0, \quad i = 1, ..., n_s \tag{4.15}$$

where f_i denotes the function of i^{th} constraint, and n_s is the total number of obstacles.

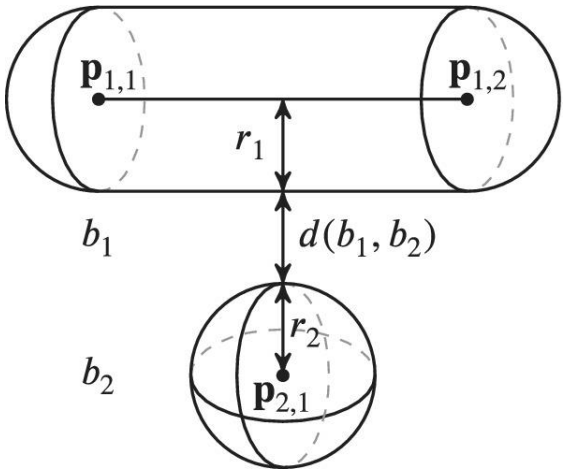


Figure 4.1: Distance between a SSL b_1 and a sphere b_2 (Krämer *et al.*, 2020)

The robot is modeled as SSLs. The distance between a sphere and SSL is shown in Figure 4.1. A SSL is composed of a line segment defined by the two dynamic endpoints $\mathbf{p}_{1,1}(t)$ and $\mathbf{p}_{1,2}(t)$. We define r_1 as the radius of robot link's SSL models and r_2 as the radius of the spheres in the union of spheres human model.

Let d_s be the desired centre-to-centre distance between the robot centreline and

obstacle centre. For a collision to be avoided: $d_s \geq r_1 + r_2$.

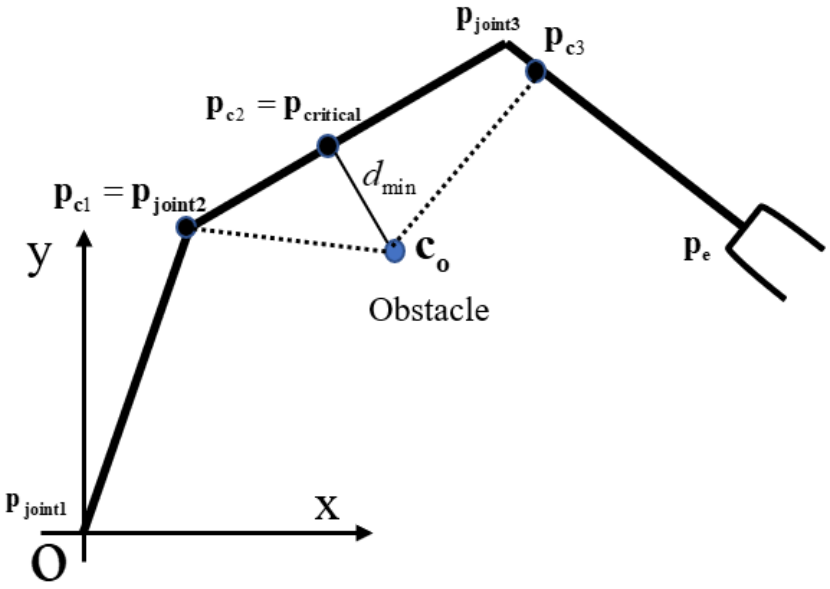


Figure 4.2: Geometry of a planar manipulator showing the point nearest to the obstacle

See Figure 4.2 for a planar example. Where $\mathbf{c}_o$ is the centre of the obstacle, $d_{\min}$ the closest distance between the obstacle and whole robot arm, $\mathbf{p}_{critical}$ is the critical point for whole arm, $\mathbf{p}_{joint1}$ is the location of joint 1, $\mathbf{p}_{joint2}$ is the location of joint 2, $\mathbf{p}_{joint3}$ is the location of joint 3, $\mathbf{p}_e$ is the location of end-effector , $\mathbf{p}_{c1}$ is the closest point to the obstacle on link 1, $\mathbf{p}_{c2}$ is the closest point to the obstacle on link 2, and $\mathbf{p}_{c3}$ is the closest point to the obstacle on link 3. Taking link 1 as an example, the way to calculate the closest point to the obstacle on the link is as follows:

$$\mathbf{v}_{1toc1} = \frac{\left((\mathbf{c}_o - \mathbf{p}_{joint1})^T(\mathbf{p}_{joint2} - \mathbf{p}_{joint1})\right)(\mathbf{p}_{joint2} - \mathbf{p}_{joint1})}{\left(\left\|\mathbf{p}_{joint2} - \mathbf{p}_{joint1}\right\|\right)^2} \tag{4.16}$$

$$l_{\text{par1}} = \frac{(\mathbf{c_o} - \mathbf{p_{joint1}})^{\mathrm{T}}(\mathbf{p_{joint2}} - \mathbf{p_{joint1}})}{\left(\left\|\mathbf{p_{joint2}} - \mathbf{p_{joint1}}\right\|\right)^2} \tag{4.17}$$

$$\begin{cases} \mathbf{p_{c1}} = \mathbf{p_{joint1}} & \text{if } l_{par1} \le 0 \\ \mathbf{p_{c1}} = \mathbf{p_{joint2}} & \text{if } l_{par1} \ge 1 \\ \mathbf{p_{c1}} = \mathbf{p_{joint1}} + \mathbf{v}_{1toc1} & \text{otherwise} \end{cases} \tag{4.18}$$

where $\mathbf{v}_{1toc1}$ is the vector from joint 1 to $\mathbf{p_{c1}}$ and l_{par1} is a parameter that determines whether $\mathbf{p_{c1}}$ located at $\mathbf{p_{joint1}}$, $\mathbf{p_{joint2}}$ or a position between $\mathbf{p_{joint1}}$ and $\mathbf{p_{joint2}}$. Then $d_{\min}$ can be determined using:

$$d_{\min} = \min\left(\left\|\mathbf{p_{c1}} - \mathbf{o_c}\right\|, \left\|\mathbf{p_{c2}} - \mathbf{o_c}\right\|, \left\|\mathbf{p_{c3}} - \mathbf{o_c}\right\|, \left\|\mathbf{p}_e - \mathbf{o_c}\right\|\right) \tag{4.19}$$

The critical point is then given by:

$$\begin{cases} \mathbf{p_{critical}} = \mathbf{p_{c1}} & \text{if } d_{\min} = \left\|\mathbf{p_{c1}} - \mathbf{o_c}\right\| \\ \mathbf{p_{critical}} = \mathbf{p_{c2}} & \text{if } d_{\min} = \left\|\mathbf{p_{c2}} - \mathbf{o_c}\right\| \\ \mathbf{p_{critical}} = \mathbf{p_{c3}} & \text{if } d_{\min} = \left\|\mathbf{p_{c3}} - \mathbf{o_c}\right\| \\ \mathbf{p_{critical}} = \mathbf{p}_e & \text{if } d_{\min} = \left\|\mathbf{p}_e - \mathbf{o_c}\right\| \end{cases} \tag{4.20}$$

We define:

$$f_1 = 0.5\left(\|\mathbf{d}\|^2 - d_s^2\right) = 0.5\left(\mathbf{d}^{\mathrm{T}}\mathbf{d} - d_s^2\right) > 0 \tag{4.21}$$

where $\mathbf{d} = \mathbf{p_{critical}} - \mathbf{c}_o$. This expression of f_1 has same function as $f_1 = \|\mathbf{d}\| - d_s$, but simplifies the equation for $\dot{f}_1$ (see (Sciavicco and Siciliano, 1988)).

Differentiating (4.21) with respect to time gives

$$\dot{f}_1 = \mathbf{d}^{\mathrm{T}}\dot{\mathbf{c}} + \mathbf{J_f}\dot{\mathbf{q}} - d_s\dot{d}_s \tag{4.22}$$

with $\mathbf{J_f} = \mathbf{d}^{\mathbf{T}}\mathbf{J}_{\mathbf{critical}}$, where $\mathbf{J}_{\mathbf{critical}}$ is a (3 × 6) matrix, and $\dot{\mathbf{c}}$ is the moving speed of the obstacle. $\mathbf{J}_{\mathbf{critical}}$ is equal to the first 3 rows of the Jacobian matrix of the critical point, which can be calculated based on (4.8).

In this way, if $\dot{f} \le 0$, the joints determining the position of the critical point are prevented from moving it closer to the obstacle. As a matter of fact, a link which is a candidate for a collision is forced to move tangentially around the imaginary sphere centred at the obstacle and of radius d_s. It should be noted that in (4.22) the case of moving obstacle ($\dot{\mathbf{c}} \neq \mathbf{0}$) has been considered

The kinematic control of a manipulator with a collision avoidance constraint for an obstacle may be written as follows

$$\begin{cases} \mathbf{J}\dot{\mathbf{q}}_{\mathbf{c}} = \dot{\mathbf{x}}_{\mathbf{d}} \\ \mathbf{J_f}\dot{\mathbf{q}}_{\mathbf{c}} = b_f \end{cases} \tag{4.23}$$

where $b_f = -\mathbf{d}^{\mathbf{T}}\dot{\mathbf{c}}$

Let $\mathbf{A_1} = [\mathbf{J_A}; \mathbf{J_f}]$ be the expanded Jacobian matrix and $\mathbf{b_1} = [\dot{\mathbf{x}}_{\mathbf{d}}, b_f]$ be the expanded target matrix to give the compact form

$$\mathbf{A_1}\dot{\mathbf{q}}_{\mathbf{c}} = \mathbf{b_1} \tag{4.24}$$

From (4.14), the solution to (4.24) is

$$\dot{\mathbf{q}}_{\mathbf{c}} = (\mathbf{A_1}^{\mathbf{T}}\mathbf{A_1} + \lambda\mathbf{I_n})^{-1}\mathbf{A_1}^{\mathbf{T}}\mathbf{b_1} \tag{4.25}$$

where $\mathbf{I_n}$ is the n dimension identity matrix, with n equal to the number of

columns of $\mathbf{A}_1$.

4.3.2 Varied weight method

We adopt the varied weight method proposed by (Xiang et al., 2012). Let w be the weight factor that is associated with the collision avoidance task. Define the weight array $\bar{\mathbf{w}} = [1;\ 1;\ 1;\ w]$ and weight matrix $\bar{\mathbf{W}} = diag(\bar{\mathbf{w}})$. Incorporating $\bar{\mathbf{W}}$ into (4.25) gives:

$$\dot{\mathbf{q}}_\mathbf{c} = (\mathbf{A}_1^{\mathrm{T}}\bar{\mathbf{W}}\mathbf{A}_1 + \lambda\mathbf{I})^{-1}\mathbf{A}_1\bar{\mathbf{W}}\mathbf{b}_1 \tag{4.26}$$

The weight factor w is related to the distance between the obstacle and the critical point on the robot. $w = 0$ refers to the collision avoidance constraint is not activated, and $w > 0$ means it is activated. To smooth the activation process, the following equation is used to calculate the weight factor:

$$w = \frac{1}{1 + e^{(\|\mathbf{d}\|/d_s - 1)\alpha}} \tag{4.27}$$

where $\|\mathbf{d}\|$ is the distance between the obstacle centre and the critical point; and α determines the smoothness of the curve. In this research we found setting $\alpha = \frac{d_s}{2.5}$ to be an effective choice. For example, when d_s =100 mm, the relationship between w and $\|\mathbf{d}\|$ is shown in Figure 4.3.

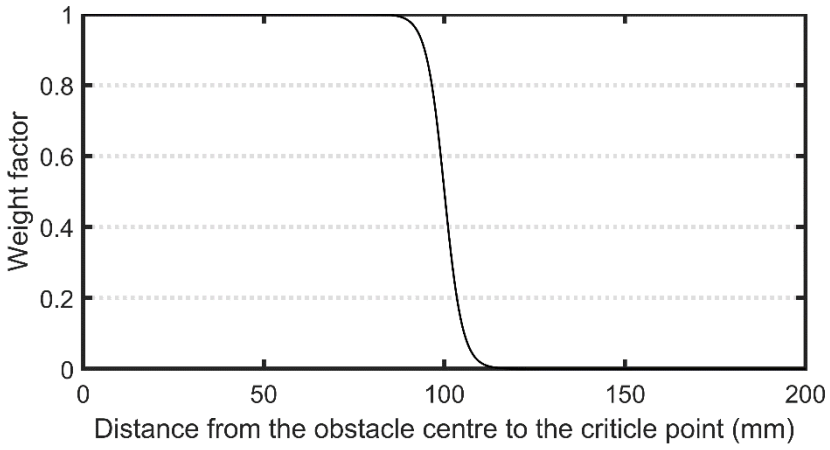


Figure 4.3: Relationship between the distance to critical point $\|\mathbf{d}\|$ and the weight factor w when d_s =100 mm.

4.3.3 Multiple sphere obstacles collision avoidance constraint

To deal with multiple obstacles, the expanded matrices should be: $\mathbf{A_2} = \mathbf{[J_A; J_{f1}; J_{f2}; ...]}$ and $\mathbf{b_2} = \mathbf{[\dot{x}_d, b_{f1}, b_{f2}, ...]}$. The weight array will be: $\bar{\mathbf{w}}_2 = [1; 1; 1; w_1; w_2; ...]$, $\bar{\mathbf{W}}_2 = diag(\bar{\mathbf{w}}_2)$ .Then (4.21) will become:

$$\dot{\mathbf{q}}_\mathbf{c} = (\mathbf{A_2}^\mathbf{T}\bar{\mathbf{W}}_2\mathbf{A_2} + \lambda\mathbf{I})^{-1}\mathbf{A_2}\bar{\mathbf{W}}_2\mathbf{b_2} \tag{4.28}$$

4.3.4 Joints angle limits constraint

Assume that a joint angle is constrained between values $q_{i\min}$ and $q_{i\max}$,The joint limits can be represented by:

$$q_{i\min} < q_i < q_{i\max} \tag{4.29}$$

A threshold distance $\mathbf{d_q}$ can be defined with the intent that if the distance of the current $\mathbf{q_i}$, from either of the two limits becomes less than $\mathbf{d_q}$, joints constraints task should be activated. Define the error:

$$\mathbf{e_{q_i}} = \mathbf{d_q} - \mathbf{d_{qi}} \tag{4.30}$$

where $\mathbf{d_{qi}} = \mathbf{q_{imax}} - \mathbf{q_i}$ or $\mathbf{d_{qi}} = \mathbf{q_i} - \mathbf{q_{imin}}$.

Define the joints constraints task Jacobian $\mathbf{J_{qi}}$ (1 × 6):

$$\mathbf{J_{qi}} = \begin{cases} [0\,0\,...1...\,0], & \dot{\mathbf{e}}_{\mathbf{qi}} = \dot{\mathbf{q}}_{\mathbf{i}} \\ [0\,0\,...-1...\,0], & \dot{\mathbf{e}}_{\mathbf{qi}} = -\dot{\mathbf{q}}_{\mathbf{i}} \end{cases} \tag{4.31}$$

In $\mathbf{J_{qi}}$ only the i^{th} factor is nonzero, and the left are 0.

The joints limits constraint weight factor is as follows:

$$\begin{cases} w_{j(i)} = 1 & \text{if } \mathbf{q_{imax}} - \mathbf{q_i} < \mathbf{d_q} \text{ or } \mathbf{q_i} - \mathbf{q_{imin}} < \mathbf{d_q} \\ w_{j(i)} = 0 & \text{otherwise} \end{cases} \tag{4.32}$$

The overall sphere obstacle collision avoidance and joints constraints expanded Jacobian is:

$$\dot{\mathbf{q}}_{\mathbf{c}} = (\mathbf{A_3}^{\mathbf{T}}\bar{\mathbf{W}}_{\mathbf{3}}\mathbf{A_3} + \lambda\mathbf{I})^{-1}\mathbf{A_3}\bar{\mathbf{W}}_{\mathbf{3}}\mathbf{b_3} \tag{4.33}$$

where $\mathbf{A_3} = [\mathbf{J_A}; \mathbf{J_{f1}}; \mathbf{J_{f2}}; ...; \mathbf{J_{q1}}; ...\mathbf{J_{qi}}]$, $\bar{\mathbf{w}}_{\mathbf{3}} = [1;1;1; w_1; w_2; ...w_{j1}; ...w_{ji}]$, $\bar{\mathbf{W}}_{\mathbf{3}} = diag(\bar{\mathbf{w}}_{\mathbf{3}})$, $\mathbf{b_3} = [\dot{\mathbf{x}}_{\mathbf{d}}, b_{f1}, b_{f2}, ..., b_{fJoints}]$.

4.3.5 Plane collision avoidance constraint

We assume that the plane surface being avoided is larger than the robot's reach so it is equivalent to the problem of avoiding an infinite plane. Based on this assumption, the critical point on the robot arm will always one of joints or the end-effector.

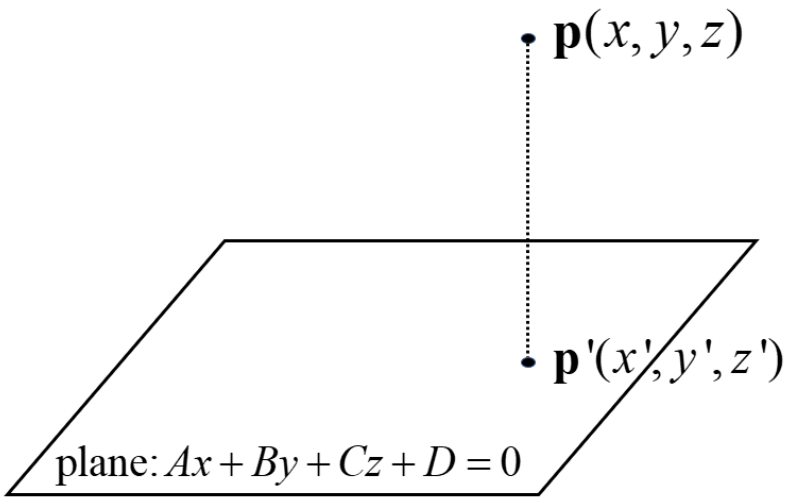


Figure 4.4: The point $\mathbf{p}(x, y, z)$ and its projection onto the plane $\mathbf{p}'(x', y', z')$.

A plane can be represented by:

$$Ax + By + Cz + D = 0 \tag{4.34}$$

To calculate the closest distance between the point and the plane, d_{plane}, the point $\mathbf{p}(x, y, z)$ is projected onto the plane as shown in Fig. 4.4. Then the distance

$d_{plane} = \| \mathbf{p} - \mathbf{p}' \|$ is given by:

$$\begin{aligned} d_{plane} &= \frac{|Ax + By + Cz + D|}{\sqrt{A^2 + B^2 + C^2}} \\ x' &= x - \frac{Ax + By + Cz + D}{A^2 + B^2 + C^2} \\ y' &= y - \frac{Ax + By + Cz + D}{A^2 + B^2 + C^2} \\ z' &= z - \frac{Ax + By + Cz + D}{A^2 + B^2 + C^2} \end{aligned} \tag{4.35}$$

For a collision to be avoided: $d_{plane} \geq d_p$ and $d_p \geq r_1$.

Next, similar to the sphere obstacle collision avoidance, $\mathbf{J_{fPlane}} = \mathbf{d_{plane}}^{\mathbf{T}} \mathbf{J_{criticalPlane}}$ and b_{fPlane} is 0 or an arbitrary negative scalar. $\mathbf{J_{fPlane}}$ is (1 × 6) Jacobian matrix of the plane avoidance constraint. $\mathbf{J_{criticalPlane}}$ is (3 × 6) Jacobian matrix of the critical point on the arm to a plane. For a typical 6 DOF robot arm, $\mathbf{J_{criticalPlane}}$ will be first 3 rows of the Jacobian matrix at joint 3, joint 5, and the end-effector.

The weight factor is:

$$w_{plane} = \frac{1}{1 + e^{(d_{plane}/d_P - 1)\alpha}} \tag{4.36}$$

A robot manipulator is typically mounted on a horizontal flat table with the robot's Z axis normal to the table. Therefore, if the robot's origin and table's origin both lie in the XY plane, the distance from a critical point on the robot to the table plane is simply: $d_{plane} = z_{critical} - r_1$ where $z_{critical}$ is the Z coordinate of $p_{critical}$.

The overall sphere obstacle collision avoidance, joints constraints and plane avoidance expanded Jacobian is:

$$\dot{\mathbf{q}}_c = (\mathbf{A}_4^T \bar{\mathbf{W}}_4 \mathbf{A}_4 + \lambda \mathbf{I})^{-1} \mathbf{A}_4 \bar{\mathbf{W}}_4 \mathbf{b}_4 \tag{4.37}$$

where $\mathbf{A}_4 = [\mathbf{J}_A; \mathbf{J}_{f1}; \mathbf{J}_{f2}; ...; \mathbf{J}_{fJoint}; \mathbf{J}_{fPlane}]$, $\bar{\mathbf{w}}_4 = [1;1;1; w_1; w_2; ...w_{j1}; ...w_{j6}; w_{plane}]$, $\bar{\mathbf{W}}_4 = \mathbf{diag}(\bar{\mathbf{w}}_4)$ and $\mathbf{b}_4 = [\dot{\mathbf{x}}_d, b_{f1}, b_{f2}, ..., b_{fJoints}, b_{fPlane}]$.

4.4 Closed-loop inverse kinematics implementation

4.4.1 Main task only

In practice, $\dot{\mathbf{q}}_c$ must be numerically integrated to get the vector of desired or commanded joint angles to send to the robot controller. This numerical integration will drift over time causing errors in the end-effector pose. Obtaining the joint velocities using (4.14) and integration is analogous to open-loop control since the end-effector pose is not used in the calculation. In this section a closed-loop inverse kinematics method intended to reduce the end-effector pose errors will be presented (see (Sciavicco and Siciliano, 1988)).

Let $\mathbf{x}_d$ (6 × 1) be the desired end-effector pose. The pose error vector $\mathbf{e}$ (6 × 1) can be defined as:

$$\mathbf{e} = \mathbf{x}_d - \mathbf{x} \tag{4.38}$$

Differentiating (4.38) with respect to time gives:

$$\dot{\mathbf{e}} = \dot{\mathbf{x}}_d - \dot{\mathbf{x}} = \dot{\mathbf{x}}_d - \mathbf{J}\dot{\mathbf{q}} \tag{4.39}$$

As in (4.13), the result using the open-loop pseudoinverse method is:

$$\dot{\mathbf{q}}_c = \mathbf{J}^* \dot{\mathbf{x}}_d = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}\dot{\mathbf{x}}_d \tag{4.40}$$

Adding a feedback term, $\mathbf{Ke}$, produces the closed-loop pseudoinverse method:

$$\dot{\mathbf{q}}_c = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1}(\mathbf{K}\mathbf{e} + \dot{\mathbf{x}}_d) \tag{4.41}$$

where $\mathbf{K}$ is a positive-definite proportional gain matrix. Its elements affect the error convergence rate.

Finally, employing the damped least-squares method, as was done in (4.14), gives:

$$\dot{\mathbf{q}}_c = (\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I}_6)^{-1}\mathbf{J}^T(\mathbf{K}\mathbf{e} + \dot{\mathbf{x}}_d) \tag{4.42}$$

4.4.2 Additional constraint tasks

1) Sphere collision avoidance task

With reference to Figure 4.1, define the error:

$$e_s = d_s - d_{sphere} \tag{4.43}$$

where $d_{sphere} = \|\mathbf{d}\|$.

Define the sphere collision avoidance task Jacobian $\mathbf{J}_s(3 \times 6)$:

$$\mathbf{J}_s = \mathbf{d}^T\mathbf{J}_{\mathbf{critical}} \tag{4.44}$$

2) Joints angle constraints task

Joints angle constraints task error can be calculated from (4.30) and the joints

constraints task Jacobian is the same as (4.31).

3) Plane collision avoidance task

In analogy to the sphere obstacle avoidance task, define

$$e_p = d_p - d_{plane} \tag{4.45}$$

Define the plane avoidance task Jacobian $\mathbf{J_p}$ (3×6)

$$\mathbf{J_p} = (\mathbf{p} - \mathbf{p}')^{\mathrm{T}} \mathbf{J}_{\mathbf{criticalPlane}} \tag{4.46}$$

4.4.3 Closed-loop inverse kinematic control law

The extended error vector $\mathbf{e_t}$ in the task space includes the end-effector error vector $\mathbf{e}$ from (4.38), along with errors defined in (4.30), (4.43) and (4.45) as follows:

$$\mathbf{e_t} = \begin{bmatrix} \mathbf{Ke} + \dot{\mathbf{x}}_\mathbf{d} \\ K_{es} e_s + b_f \\ e_{qi} + b_{fJoints} \\ e_p + b_{fPlane} \end{bmatrix} \tag{4.47}$$

The closed-loop control law is:

$$\dot{\mathbf{q}}_\mathbf{c} = (\mathbf{J_c}^{\mathrm{T}} \mathbf{J_c} + \lambda \mathbf{I_6})^{-1} \mathbf{J_c}^{\mathrm{T}} \mathbf{e_t} \tag{4.48}$$

where

$$\mathbf{J}_c = \begin{bmatrix} \mathbf{J} \\ \mathbf{J}_s \\ \mathbf{J}_{qi} \\ \mathbf{J}_p \end{bmatrix} \tag{4.49}$$

Finally, the varied weight version of (4.48) is:

$$\dot{\mathbf{q}}_c = (\mathbf{J}_c^T \bar{\mathbf{W}}_4 \mathbf{J}_c + \lambda \mathbf{I}_6)^{-1} \mathbf{J}_c^T \bar{\mathbf{W}}_4 \mathbf{e}_t \tag{4.50}$$

4.4.4 Closed-loop Kinematic Control Law Exploiting Functional Redundancy

Kinematic redundancy is related to the number n of DOFs of the structure, the number m of operational space variables, and the number r of operational space variables necessary to specify a given task. Equation (4.52) may be interpreted as the differential kinematics mapping relating the n components of the joint velocity vector to the $r \leq m$ components of the velocity vector $\dot{x}$ of concern for the specific task. In the case of a 6 DOF robot, that is not intrinsically redundant when considering both the 3 position variables and 3 orientation variables we have: $n = m = r = 6$.

The Jacobian matrix $\mathbf{J}$ can be written in terms of two submatrices as in (4.7). If we decide that the end-effector drops orientation tracking and performs position tracking only, then we have chosen to ignore the $\boldsymbol{\omega}_e$ portion of $\dot{\mathbf{x}}$. This makes the arm functionally redundant with $r = 3$ and allows us to replace $\mathbf{J}$ with $\mathbf{J}_3$ in the kinematic control law (4.50). The new control law is

$$\dot{\mathbf{q}}_c = (\mathbf{J}_{cr}^T \bar{\mathbf{W}}_4 \mathbf{J}_{cr} + \lambda \mathbf{I}_6)^{-1} \mathbf{J}_{cr}^T \bar{\mathbf{W}}_4 \mathbf{e}_r \tag{4.51}$$

where

$$\mathbf{J_{cr}} = \begin{bmatrix} \mathbf{J_3} \\ \mathbf{J_s} \\ \mathbf{J_{qi}} \\ \mathbf{J_p} \end{bmatrix} \tag{4.52}$$

and

$$\mathbf{e_r} = \begin{bmatrix} \mathbf{K e_{pe}} + \dot{\mathbf{x}}_{\mathbf{dpe}} \\ e_s + b_f \\ e_{qi} + b_{fJoints} \\ e_p + b_{fPlane} \end{bmatrix} \tag{4.53}$$

with $\mathbf{e_{pe}} = \mathbf{p_e} - \mathbf{p_{ed}}$, where $\mathbf{p_{ed}}$ (3×1) desired end-effector position.

Under the control of (4.52), the end-effector can track a desired trajectory, and the three redundant DOF's are used to perform subtasks, which include obstacle avoidance, joints angle constraints and plane avoidance. For example, let's say we have three subtasks with r values of r_1, r_2 and r_3, respectively. As long as $r_1 + r_2 + r_3 \leq 3$ both the main task and subtasks will not fail. If $r_1 + r_2 + r_3 > 3$, failure may occur in one or more of those tasks. If we want to guarantee the success of all tasks, a proper strategy to decide which subtask will be activated is crucial. The weight matrix $\bar{\mathbf{W}}_4$ can be used for this purpose. The weight factor of the main task is always 1. Each subtask's weight factor is changing from 0, meaning the subtask is not activated, to 1, meaning the subtask is fully activated.

For a specific robot application where the end-effector's position and orientation are crucial only at the start and end points (e.g. bin picking), it is reasonable to drop the orientation tracking requirement to guarantee the success of safety related subtasks when one of them is activated.

Based on the above, we propose the following closed-loop inverse kinematics control algorithm exploiting functional redundancy (CLIKFR) for collision avoidance:

Algorithm 4.1

1. Set the values of $\mathbf{x}_{\mathbf{start}}, \mathbf{x}_{\mathbf{goal}}$, d_s, d_q, and d_p. Also, define the sampling period, T_s, and final error tolerance, $\mathbf{e}_{\mathbf{t}}$. Set the iteration steps, *n*, equal to 1.

2. Generate the $\mathbf{x}_{\mathbf{d}}$ and $\dot{\mathbf{x}}_{\mathbf{d}}$ trajectories using the linear segments with parabolic blends (LSPB) algorithm (e.g., page 184 of (Siciliano, 2009)).

3. Set $\mathbf{x(1) = x_d(1)}$ and $\dot{\mathbf{q}}(1) = \mathbf{0}$.

4. Compute $w_1 \cdots w_6 \, w_{j1} \cdots w_{j6}$ and w_{plane} for the n^{th} iteration

5. If $w_1 + \cdots w_{f1} + \cdots w_{plane} < w_t$, deploy control law (4.50) to generate $\dot{\mathbf{q}}(n+1)$; else, deploy control law (4.51) to generate $\dot{\mathbf{q}}(n+1)$.

6. Calculate $\mathbf{q}(n+1)$ by Euler integration of $\dot{\mathbf{q}}(n+1)$.

7. Calculate $\mathbf{x}(n+1)$ from $\dot{\mathbf{q}}(n+1)$ using (4.1).

8. Calculate the comprehensive error $\mathbf{e(}n\mathbf{)} = \left\| \mathbf{x}_{\mathbf{goal}} - \mathbf{x(}n+1\mathbf{)} \right\|$

9. If $\mathbf{e}(n) > \mathbf{e}_{\mathbf{t}}$, set $n = n+1$ and go to step 4.

Stop.

4.5 Nonlinear Model Predictive Control

An NMPC algorithm for manipulator collision avoidance is designed in this section. Robot tasks are considered that are defined in the Cartesian space by a desired reference position and orientation of the end-effector. The goal is to minimize the deviation of the end-effector position and orientation from their reference values, which is described by the pose error vector $\mathbf{e}$ in Equation (4.39). Based on this purpose, the following cost function is proposed:

$$\text{cost} = \sum_{i=1}^{N_p} \left\| \mathbf{Q}\left(\mathbf{x_d}(k+i) - \mathbf{x}^*(k+i)\right) \right\| \tag{4.54}$$

where N_p is the prediction horizon; $\mathbf{Q}$ is a weighting matrix for the position and orientation errors; k is the current sampling instant; $\mathbf{x_d}(k+i)$ is the future desired end-effector pose; and $\mathbf{x}^*(k+i)$ is the predicted end-effector pose. This cost function equals the sum of the weighted pose error prediction norms. The following optimization problem is then solved every sampling instant k:

$$\mathbf{q_c}(k+i) = \arg\min_{k} \text{cost} \qquad \forall \text{i} \in \{1, 2, \ldots, N_p\} \tag{4.55}$$

Subject to:

$$\mathbf{C}(\mathbf{q_c}(k+i)) \leq 0 \tag{4.56}$$

where $\mathbf{q_c}(k+i)$ is the predicted vector of joint position commands, $\mathbf{C} = [\mathbf{C_1}\, \mathbf{C_2}\, \mathbf{C_3}\, \mathbf{C_4}\, \mathbf{C}_5]^{\mathrm{T}}$ contains joint angle constraints $\mathbf{C_1}$, joint velocity constraints $\mathbf{C}_2$, joint acceleration constraints $\mathbf{C}_3$, sphere collision avoidance constraints $\mathbf{C}_4$, and

plane collision avoidance constraints $\mathbf{C}_5$.

The actual joint position, velocity and acceleration limits should be satisfied in (3.21), (3.22) and (3.23). In the controller, the joint position, velocity and acceleration commands also should be limited by following equations:

$$\begin{cases} \mathbf{q_c} = \mathbf{max}(\mathbf{q_c}, \mathbf{q_{c,min}}) \\ \mathbf{q_c} = \mathbf{min}(\mathbf{q_c}, \mathbf{q_{c,max}}) \end{cases} \tag{4.57}$$

$$\begin{cases} \dot{\mathbf{q}}_\mathbf{c} = \mathbf{max}(\dot{\mathbf{q}}_\mathbf{c}, \dot{\mathbf{q}}_{\mathbf{cv,min}}) \\ \dot{\mathbf{q}}_\mathbf{c} = \mathbf{min}(\dot{\mathbf{q}}_\mathbf{c}, \dot{\mathbf{q}}_{\mathbf{cv,max}}) \end{cases} \tag{4.58}$$

$$\begin{cases} \ddot{\mathbf{q}}_\mathbf{c} = \mathbf{max}(\ddot{\mathbf{q}}_\mathbf{c}, \ddot{\mathbf{q}}_{\mathbf{ca,min}}) \\ \ddot{\mathbf{q}}_\mathbf{c} = \mathbf{min}(\ddot{\mathbf{q}}_\mathbf{c}, \ddot{\mathbf{q}}_{\mathbf{ca,max}}) \end{cases} \tag{4.59}$$

where $\mathbf{q_c}$ is the joint position command; $\dot{\mathbf{q}}_\mathbf{c}$ is the joint velocity command; $\ddot{\mathbf{q}}_\mathbf{c}$ is the joint acceleration command; $\dot{\mathbf{q}}_{\mathbf{c,min}}$ and $\dot{\mathbf{q}}_{\mathbf{c,max}}$ are minimum and maximum joint position command; $\dot{\mathbf{q}}_{\mathbf{cv,min}}$ and $\dot{\mathbf{q}}_{\mathbf{cv,max}}$ are minimum and maximum joint velocity command; and $\ddot{\mathbf{q}}_{\mathbf{ca,min}}$ and $\ddot{\mathbf{q}}_{\mathbf{ca,max}}$ are minimum and maximum joint acceleration command. Based on (4.57)-(4.59), $\mathbf{C_1}$, $\mathbf{C}_2$ and $\mathbf{C}_3$ are defined as follows:

$$\begin{cases} \mathbf{C_1}(\mathbf{q_c}(k+i)) = \mathbf{0}, & \text{if } \mathbf{q_{c,min}} \le \mathbf{q_c}(k+i) \le \mathbf{q_{c,max}} \\ \mathbf{C_1}(\mathbf{q_c}(k+i)) = \mathbf{1}, & \text{otherwise} \end{cases} \tag{4.60}$$

$$\begin{cases} \mathbf{C}_2(\mathbf{q_c}(k+i)) = \mathbf{0}, & \text{if } \dot{\mathbf{q}}_{\mathbf{cv,min}} \le \dot{\mathbf{q}}_\mathbf{c}(k+i) \le \dot{\mathbf{q}}_{\mathbf{cv,max}} \\ \mathbf{C}_2(\mathbf{q_c}(k+i)) = \mathbf{1}, & \text{otherwise} \end{cases} \tag{4.61}$$

$$\begin{cases} \mathbf{C}_3(\mathbf{q_c}(k)) = \mathbf{0}, & \text{if } \ddot{\mathbf{q}}_{\mathbf{ca,min}} \le \ddot{\mathbf{q}}_\mathbf{c}(\mathbf{k}) \le \ddot{\mathbf{q}}_{\mathbf{ca,max}} \\ \mathbf{C}_3(\mathbf{q_c}(k)) = \mathbf{1}, & \text{else} \end{cases} \tag{4.62}$$

The sphere collision avoidance constraints can be defined as follows:

$$\mathbf{C_4}(\mathbf{q_c}(k)) = d_s - d_{sphere}(k) \tag{4.63}$$

Finally, the plane collision avoidance constraints are given by:

$$\mathbf{C}_5(\mathbf{q_c}(k)) = d_p - d_{plane}(k) \tag{4.64}$$

Chapter 5 Simulations

5.1 Introduction

This chapter presents simulation results for the CLIKFR and NMPC collision avoidance algorithms for a variety of scenarios. The performance of the algorithms in each scenario will be discussed. All the simulations have been done in MATLAB R2017b on a Windows 10 laptop with an Intel Core (TM) i5-8250U 1.80 GHz (x64-based processor) with 8 GB RAM. Experiment results for the human modelling algorithm are presented in this chapter as well. The chapter is organized in four sections. In Section 5.2, the procedure used for simulations is described. In Section 5.3, the human modelling algorithm parameter settings and experimental results are presented. In Section 5.4, simulations with a three-link planar robot are carried out to evaluate the collision avoidance algorithms on a relatively simple problem. Simulations of the Elfin 5 robot with real point cloud data of a human in the shared workspace are presented Section 5.5. A summary is presented in Section 5.6 to conclude the chapter.

5.2 Simulation Procedure

The algorithms will be initially tested on a three-link planar robot to allow the correctness of the solution to be determined through graphs and animations. Then, simulations of an Elfin 5 robot with 6 DOF and a human sharing the workspace of the robot will be presented. The robot will be assigned a task of straight-line motions between two predefined end-effector locations. Such motions occur in practice when a robot is performing picking/placing operations, or other similar tasks. The robot should follow a straight line path with constant orientation between the start and goal location

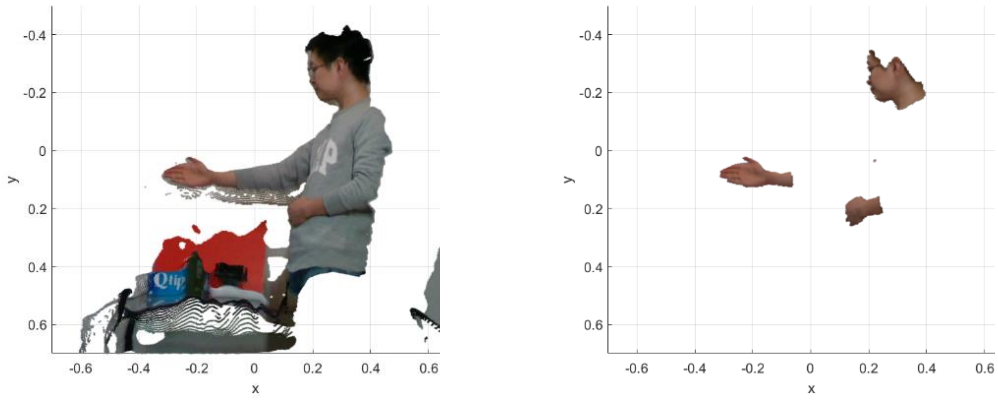
if the human's position in the workspace does not interfere with the robot's position.

To study the behaviour of the algorithm during each test, important simulation parameters are monitored, and their values stored. The recorded parameters are later used to plot graphs and animations. The monitored parameters during the individual simulations are:

- The closest surface distance $d_{\min}$ between human and robot that is used to identify collisions.
- The actual robot joint angles, velocities and accelerations.
- The normalized time $t_{normalize}$ to reach goal location which is resulting from actual collision avoidance operation time dividing by the reference time.
- The execution time $t_{execution}$ in MATLAB which indicates the computational efficiency of the algorithm.

5.3 Testing of the human modelling algorithm

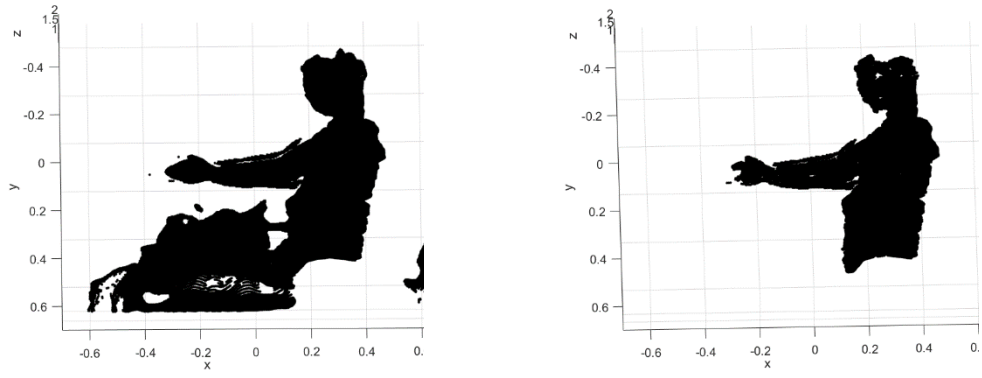
Figure 5.1 shows a result for skin detection using the method presented in section 3.2.1. The region growing process starts from the seed point which is the first point that is recognized as human skin. This result shows that the skin detection algorithm performs well with the data from the Intel RealSense D415 sensor.



(a) (b)

Figure 5.1: Skin color detection result. (a) Test image. (b) Output image

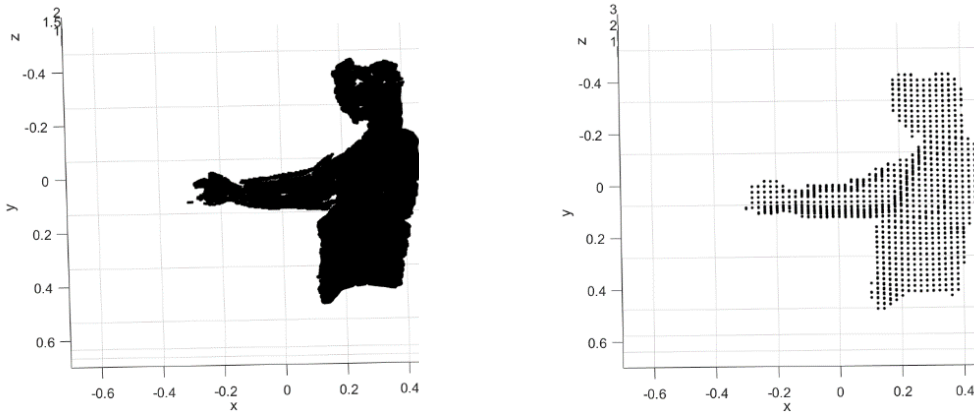
Figure 5.2 shows the experimental result of the region growing with $c_{threshold} = 0.05\,\mathrm{m}^{-1}$ and $\theta_{threshold} = 10\,\mathrm{degrees}$. The algorithm is effective at removing points belonging to the background. The remaining points belong to the human's body. Only a few points such as those near the fingertips are lost.



(a) (b)

Figure 5.2: Region growing test. (a) Original point cloud. (b) Human point cloud.

Experimental results have shown that voxelization can significantly reduce the number of points. This is desirable since it will speed up the processing of the remaining steps. For example, the number of points in the human point cloud will reduce from 61,510 to 2,125 after voxelization using a resolution of 20 mm. Figure 5.3 shows the human point cloud before and after voxelization. No important details have been lost. The execution time was reduced to 0.6 s from 0.83 s for after voxelization.

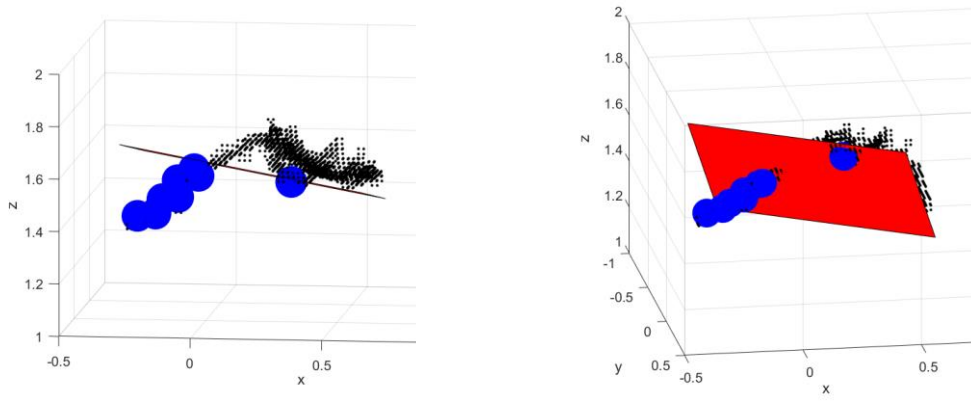


(a) (b)

Figure 5.3: Voxelization result. (a) Original point cloud (b) Point cloud after voxelization

Figure 5.4 shows a result obtained by fitting the plane model in front of the human torso and head and the union of spheres model attached to the protruding human arm. In the union of spheres model, $R_s = 60\,\text{mm}$ and $d_{pst} = 80\,\text{mm}$ are used. In this experiment, six spheres were generated to enclose the human arm, one sphere was generated to enclose a part of the human head and a plane is

generated to protect human torso. The result shows that the human's body is well covered by this simple model.



(a) (b)

Figure 5.4: Human modeling (a) Top view (b) Side view

5.4 Three-link planar robot simulation

5.4.1 Three-link planar robot simulation settings

The D-H parameters used in the simulations of the three-link planar robot are listed in Table 5.1. The length of link 1, link 2 and link 3 are 300 mm, 300 mm and 100 mm respectively. The simulated joints limits are shown in Table 5.2. The assigned task is a straight-line motion from $\mathbf{P}_{\mathbf{3R\_start}} = [-200;\ 400;\ 0]$ mm to $\mathbf{P}_{\mathbf{3R\_goal}} = [200;\ 400;\ 0]$ mm, and keep the orientation of the end-effector with *Roll-Pitch-Yaw angles* equals to $\phi_{3R} = \begin{bmatrix} 0 & 0 & pi/2 \end{bmatrix}$. The duration of the simulated task is

set to 3.3 seconds with a sampling frequency of 30 HZ. These values gives a total number of $t_{dis} = 100$ discrete times in the simulation, and a sampling period $T_s = 33ms$. In simulations using CLIKFR algorithm, the sphere collision avoidance error gain K_{es} is set to 1.

Table 5.1: D-H parameters of the three-link planar robot

Link i	θ_i [rad]	d_i [mm]	a_i [mm]	α_i [rad]
1	θ_1	0	300	0
2	θ_2	0	300	0
3	θ_3	0	100	0

Table 5.2: Joints limits of the three-link planar robot

Joint		Angle (°)	Velocity (°/s)	Acceleration (°/s^2)
1	Max	270	100	500
	Min	-270	-100	-500
2	Max	180	100	500
	Min	-180	-100	-500
3	Max	180	100	500
	Min	-180	-100	-500

A reference trajectory for the assigned task is generated by *mtraj* function from the MATLAB Robotic toolbox. The simulation results of different scenarios will be presented in the following section.

5.4.2 Three-link planar robot simulation results

5.4.2.1 Static Sphere obstacle

A static sphere obstacle is located at $\mathbf{P}_{3R\_obs} = [-200;\ 400;\ 0]$ mm with a radius of $r_2 = 50$ mm. A radius of $r_1 = 40$ mm is used for the robot's SSLs model. Figure 5.5 shows this simulation scenario. In this case, $d_s = r_1 + r_2 = 90$ mm in equation (4.22). The control law switch threshold w_t in algorithm (4.1) is set to 0.1.

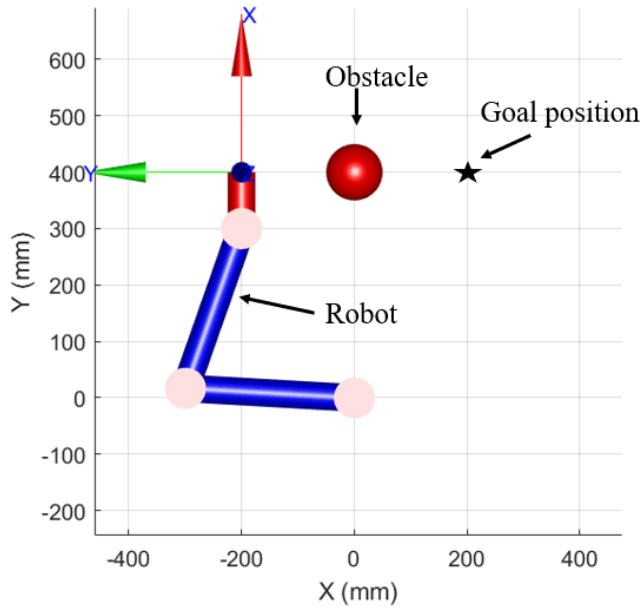
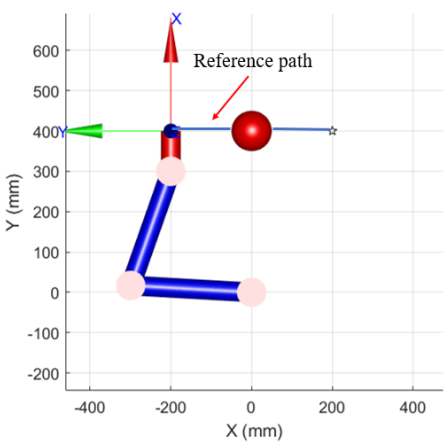
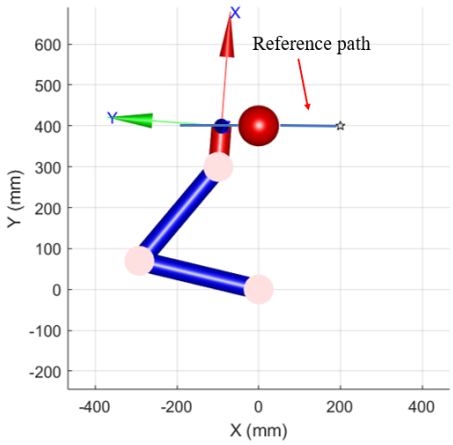


Figure 5.5: Three-link planar robot and a static sphere obstacle simulation scenario with the robot at its start position.

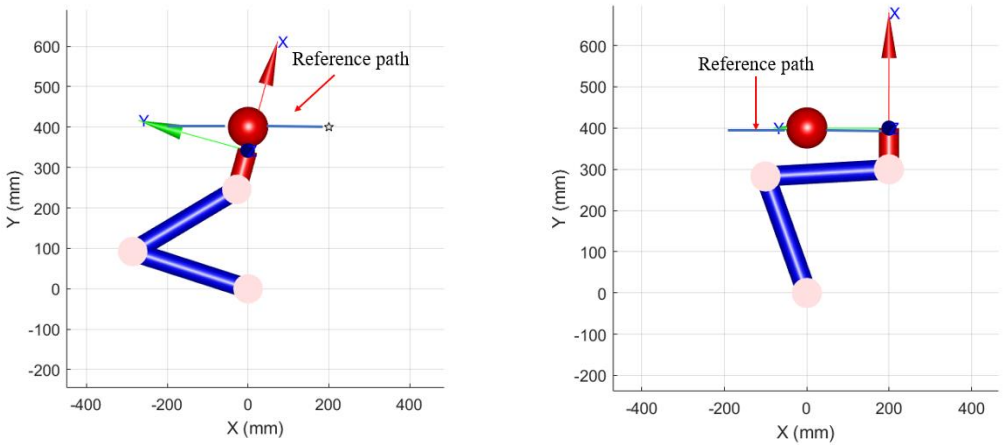
The simulation output is presented in Figure 5.6. These plots suggest the collision avoidance path is efficient. The reference operation time is 3.3 s, while the collision avoidance operation time is 6 s. Then the normalized time is 1.8. The execution time in MATLAB simulation is 1.1s, which means this method is fast enough for real-time collision avoidance applications (since 1.1 < 6). The closest distance between the robot surface and the surface of the static sphere obstacle is plotted vs. time in Figure 5.7. Since the smallest of this value is 0.43 mm, no obstacle robot collision occurred. The position error at the goal location is 0.95 mm, while the orientation error at the goal location is 0.02 degrees. The joint angle, velocity and acceleration are plotted vs. time in Figure 5.8. The joint space trajectories are smooth, which is desirable for implementing this approach in practice.



(a) Time: 0 s



(b) Time: 1.65 s



(c) Time: 3.14 s (d) Time: 6 s

Figure 5.6: Snapshots of the simulation for the planar robot controlled by the CLIKFR algorithm avoiding a static sphere obstacle

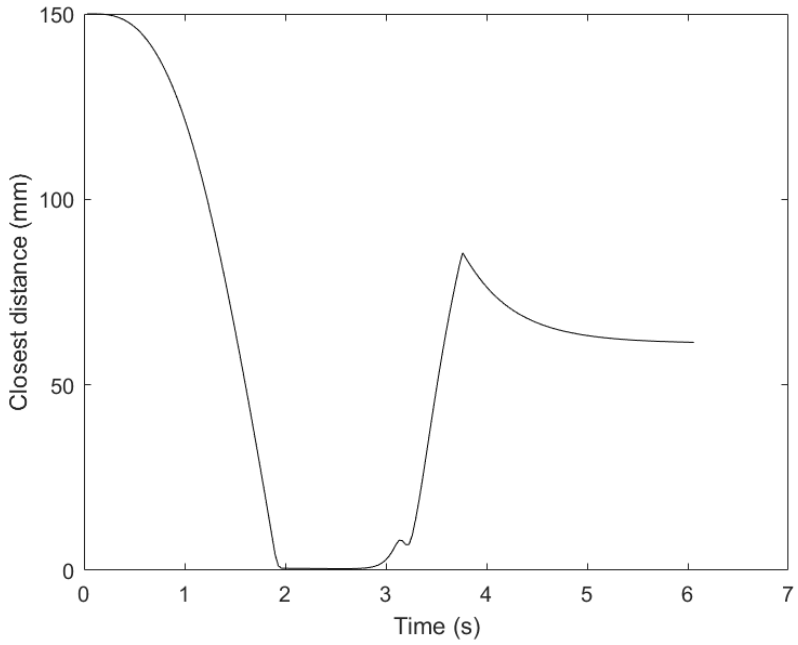


Figure 5.7: Closest distance between the robot surface and the surface of the static obstacle for the planar robot controlled by the CLIKFR algorithm

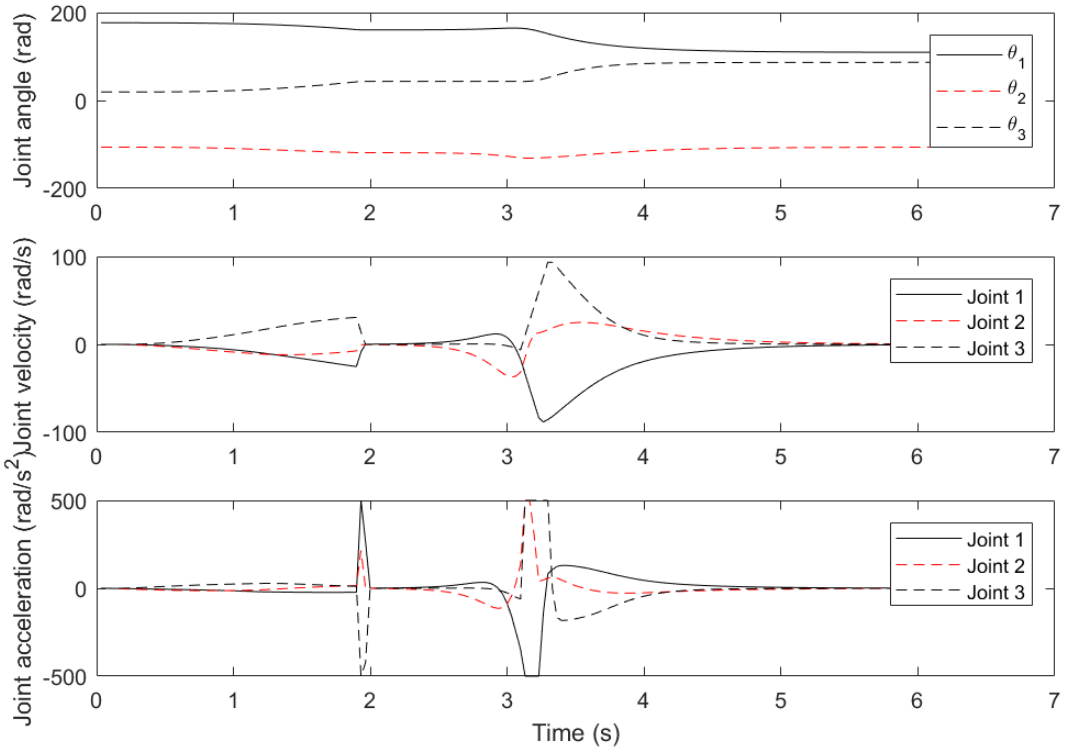


Figure 5.8: Actual joint angles, velocities and accelerations versus time for the planar robot avoiding the static obstacle using the CLIKFR algorithm

5.4.2.2 Plane avoidance

In this case, a plane located at $x = 200\,\text{mm}$ and parallel to the YZ plane acts as the obstacle. A radius of $r_1 = 40\,\text{mm}$ is still used for the robot's SSLs model. We set $d_p = r_1 = 40\,\text{mm}$ and the control law switch threshold $w_t = 0.1$ in algorithm (4.1). Figure 5.5 shows this simulation scenario with the robot at its start position. The goal position and orientation of the end-effector is set to $\mathbf{x}_{\mathbf{goal}} = \begin{bmatrix} -200 & 200 & 0 & 0 & 0 & pi/2 \end{bmatrix}^T$.

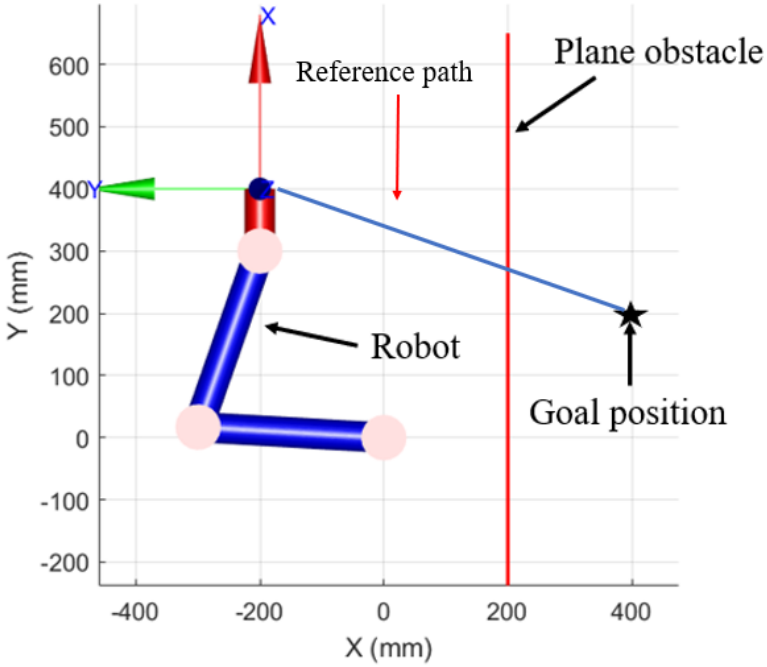
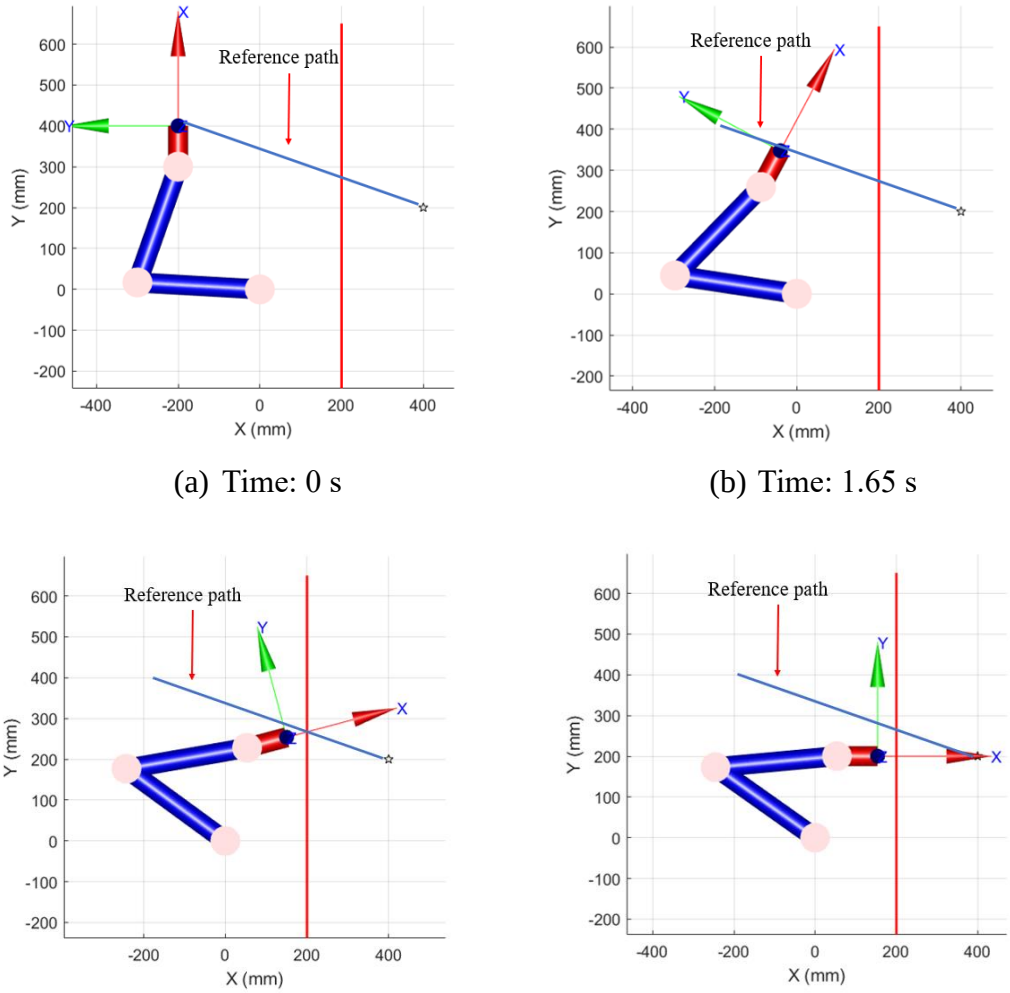


Figure 5.9: Three-link planar robot and a static plane obstacle simulation scenario with robot at its start position

The simulation output is presented in Figure 5.10. These plots suggests that the CLIKFR algorithm can control the robot to perform collision with a plane obstacle and move the end-effector close to the goal position at the same time.



(a) Time: 0 s (b) Time: 1.65 s

(c) Time: 2.64 s (d) Time: 6.6 s

Figure 5.10: Snapshots of the simulation for the planar robot controlled by the CLIKFR algorithm avoiding a plane obstacle

The closest distance between the robot surface and the plane is plotted vs. time in Figure 5.11. Since the smallest of this value is 6.4 mm, no robot collision occurred. For this case, the execution time in MATLAB simulation is 0.9 s, which means this method is fast enough for real-time collision avoidance applications. The position error at the final location is 246.5 mm, while the orientation error at the final location is 90 degrees

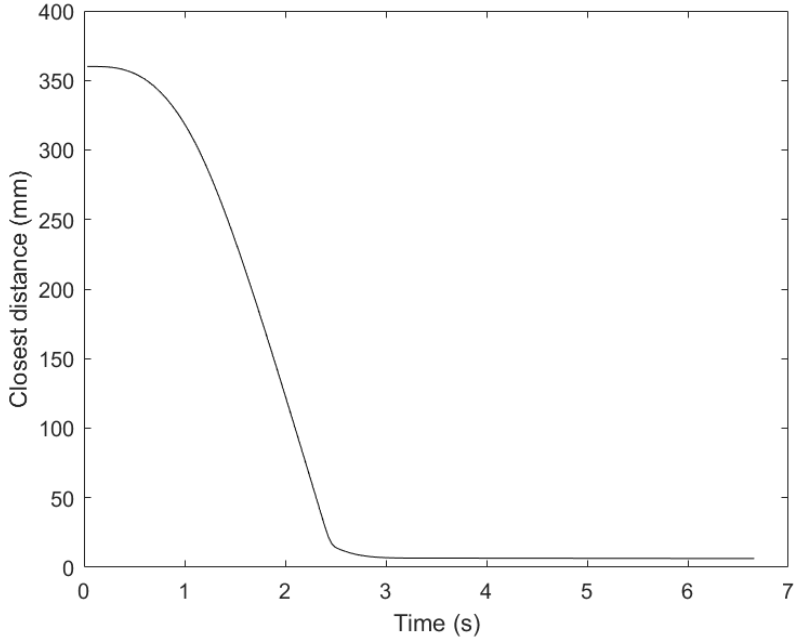


Figure 5.11: Closest distance between the robot surface and the plane for the planar robot controlled by the CLIKFR algorithm

5.4.2.3 Wrist singularity avoidance

Because it includes a damping factor, the CLIKFR controller should be able to overcome manipulator singularity problems. This simulation is designed to test this ability. The simulation scenario in presented in Figure 5.12. The initial position and orientation of the end-effector is set to $\mathbf{x}_{\mathbf{initial}} = \begin{bmatrix} 0 & 400 & 0 & 0 & 0 & pi/2 \end{bmatrix}^T$, while the goal position and orientation of the end-effector is set to $\mathbf{x}_{\mathbf{goal}} = \begin{bmatrix} 0 & 700 & 0 & 0 & 0 & pi/2 \end{bmatrix}^T$. The goal configuration is a singular configuration for this three-link planar robot.

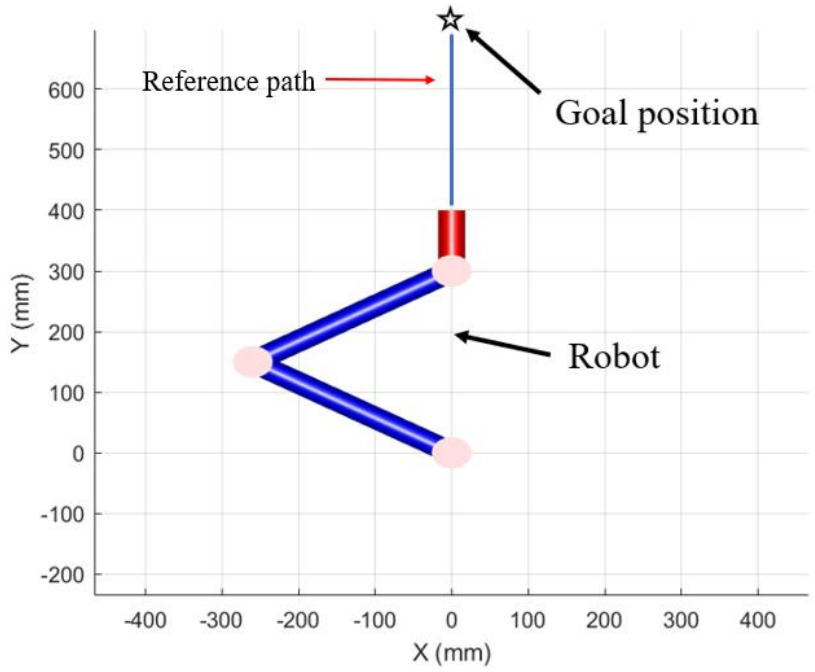


Figure 5.12: Three-link planar robot wrist singularity avoidance simulation setting

Control law (4.50) is deployed to control the robot moving from the initial configuration to the goal configuration. Figure 5.13 shows the joint angle and velocity are plotted vs. time when the damping factor $\lambda = 0$. In this scenario, the position error at the goal location is 0.61 mm, while the orientation error at the goal location is 0.75 degrees. After setting the damping factor to $\lambda = 100$, the resulting joint angle and velocity are plotted vs. time are presented in Figure 5.14. Comparing the plots in these two figures shows that the joint space trajectories become smooth after adding the damping factor. This is desirable for implementing this approach to deal with singular configurations. In this scenario, the position error at the goal location is 0.8 mm, while the orientation error at the goal location is 0.54 degrees.

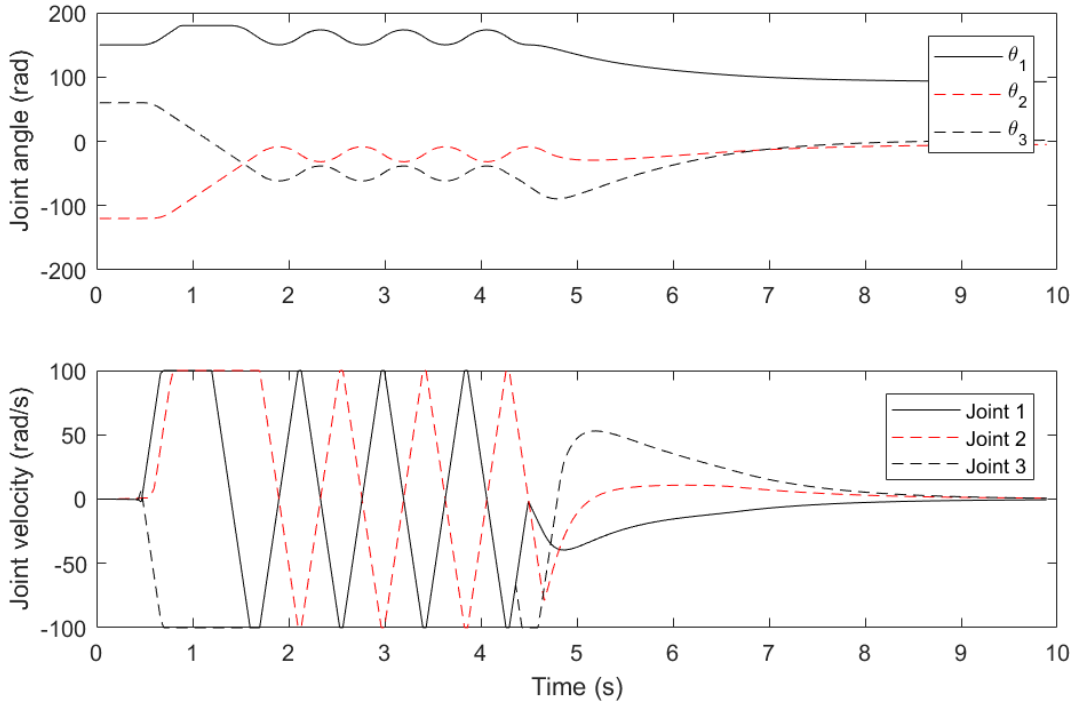


Figure 5.13: Joint angle and velocity versus time for the planar robot when $\lambda = 0$ in the CLIKFR algorithm

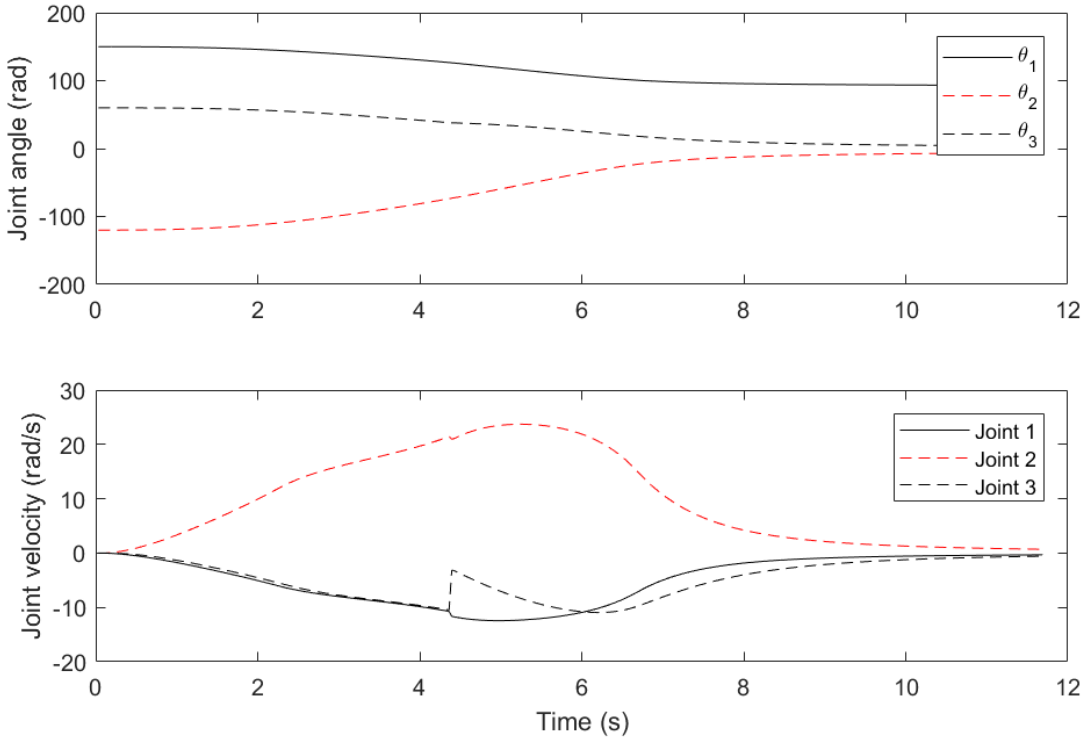


Figure 5.14: Joint angle and velocity versus time for the planar robot when $\lambda = 100$ in the CLIKFR algorithm

5.4.2.4 Dynamic obstacle collision avoidance

To test the performance of CLIKFR algorithm dealing with a dynamic obstacle, a sphere obstacle is placed at $\mathbf{P}_{\text{3R\_obs\_initial}} = [0 \quad 450 \quad 0]^T$ mm with radius of $r_2 = 50\,\text{mm}$ and moving velocity is $\mathbf{v}_{\text{3R\_obs}} = [-40 \quad 0 \quad 0]^T$ mm/s . A radius of $r_1 = 40\,\text{mm}$ is used for the robot's SSLs model. Figure 5.15 shows this simulation scenario with the robot and obstacle at their initial locations. In this case, $d_s = r_1 + r_2 = 90$ mm in equation (4.22).

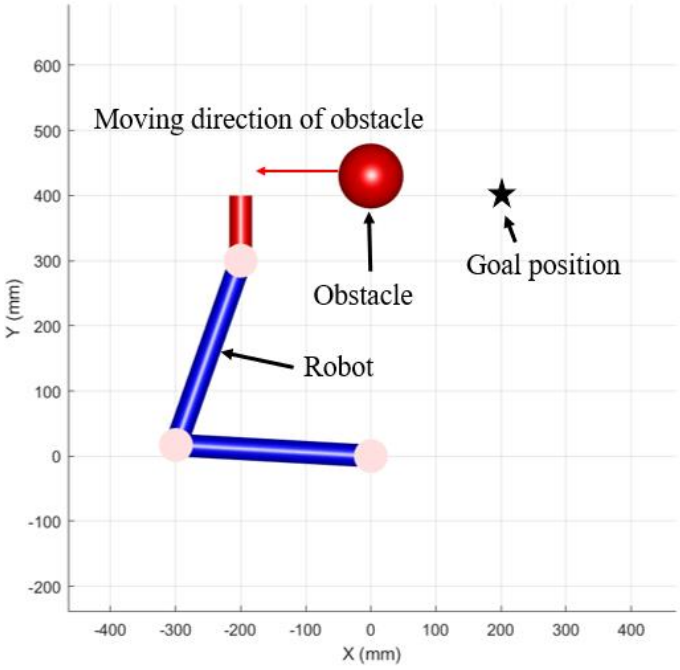
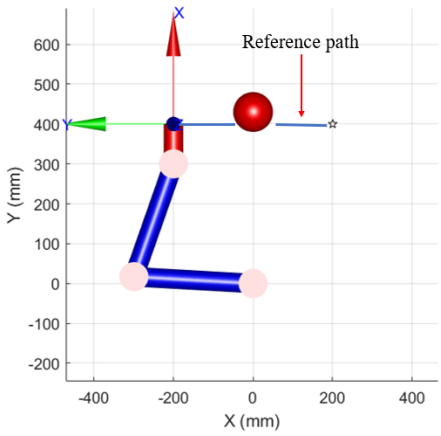
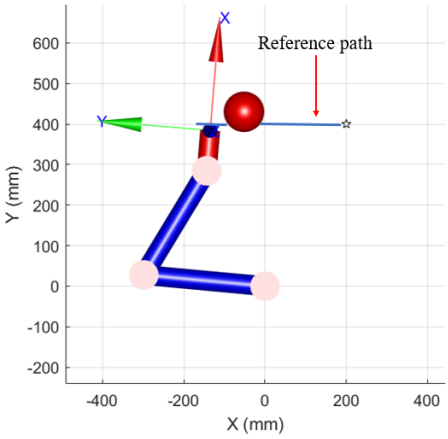


Figure 5.15: Three-link planar robot and a dynamic sphere obstacle simulation scenario with the robot and obstacle at their start positions.

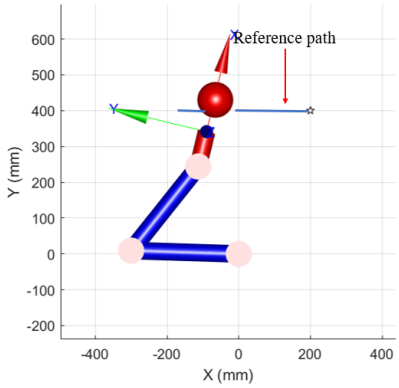
Snapshots of the simulation are presented in Figure 5.16. These plots suggest that the collision avoidance path generated by the CLIKFR is efficient for this dynamic obstacle since the robot does not make a large detour. The reference operation time is 3.3 s, while the collision avoidance operation time is 3.6 s. Then the normalized time is 1.09. The execution time of the MATLAB simulation is only 0.55s.



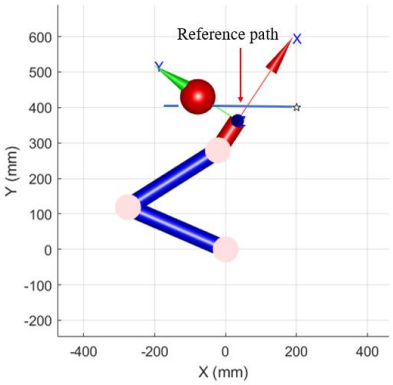
(a) Time: 0 s



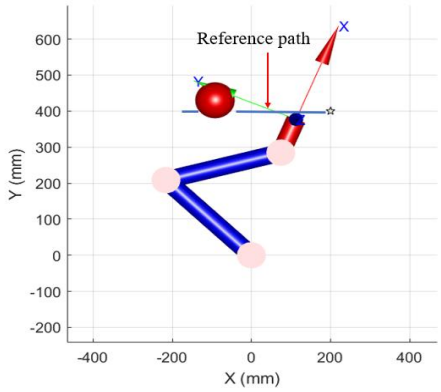
(b) Time: 1.32 s



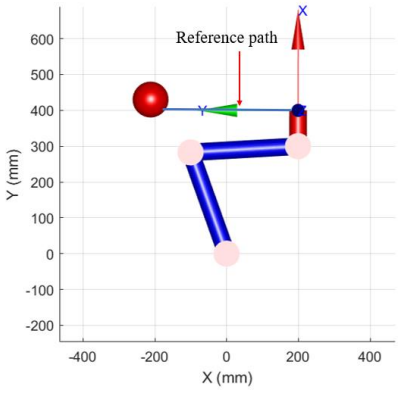
(c) Time: 1.65 s



(d) Time: 1.98 s



(e) Time: 2.31s



(f) Time: 3.6 s

Figure 5.16: Snapshots of the simulation for the planar robot controlled by the LIKFR algorithm avoiding a dynamic sphere obstacle

The closest distance between the robot surface and the surface of the dynamic sphere obstacle is plotted vs. time in Figure 5.17. Since the smallest of this value is 0.32 mm, no obstacle robot collision occurred. The joint angle, velocity and acceleration are plotted vs. time in Figure 5.18. The position error at the goal location is 0.86 mm, while the orientation error at the goal location is 0.24 degrees.

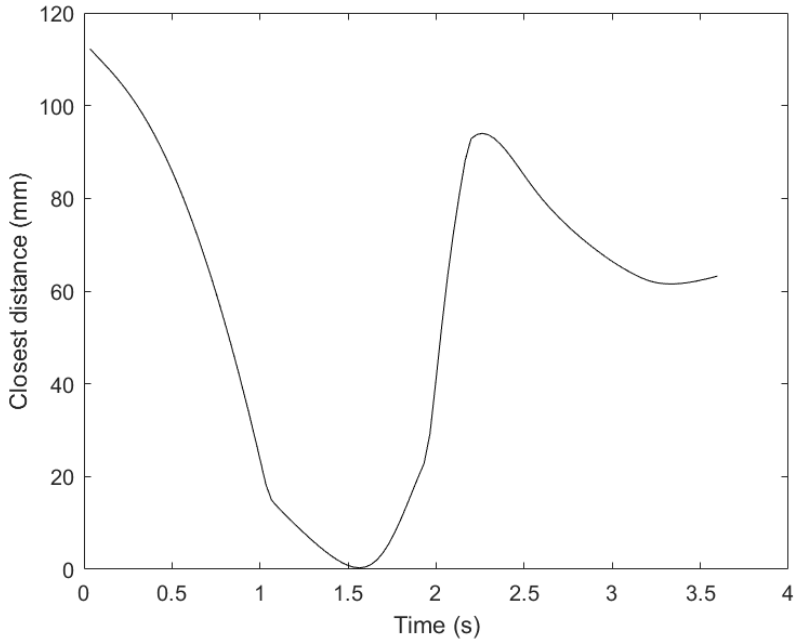


Figure 5.17: Closest distance between the robot surface and the surface of the dynamic obstacle for the planar robot controlled by the CLIKFR algorithm

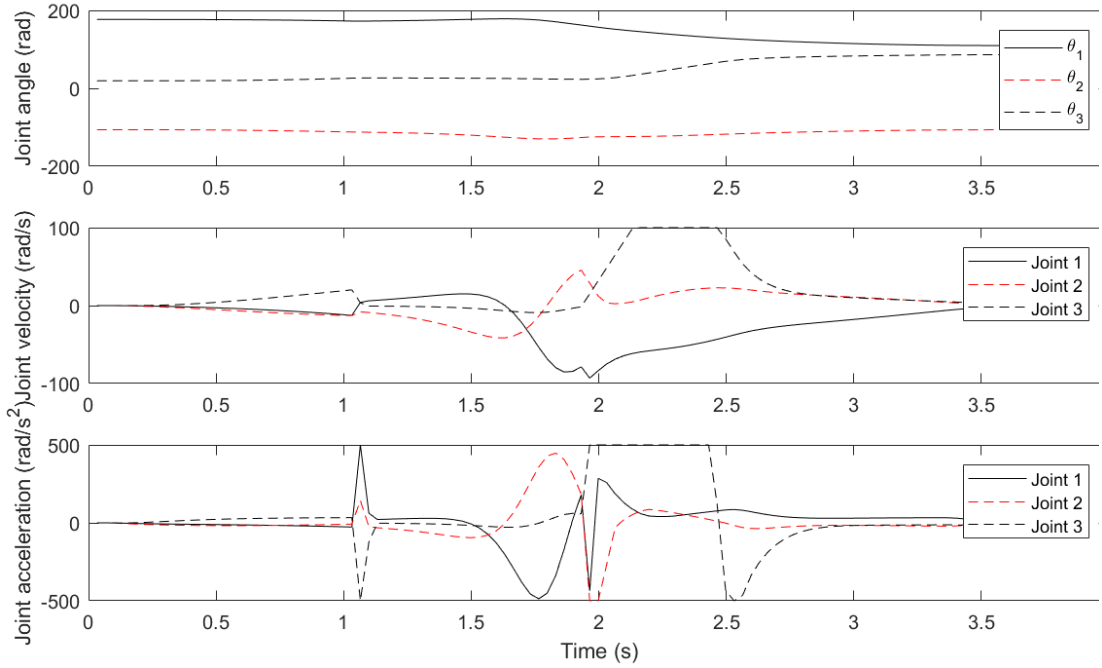
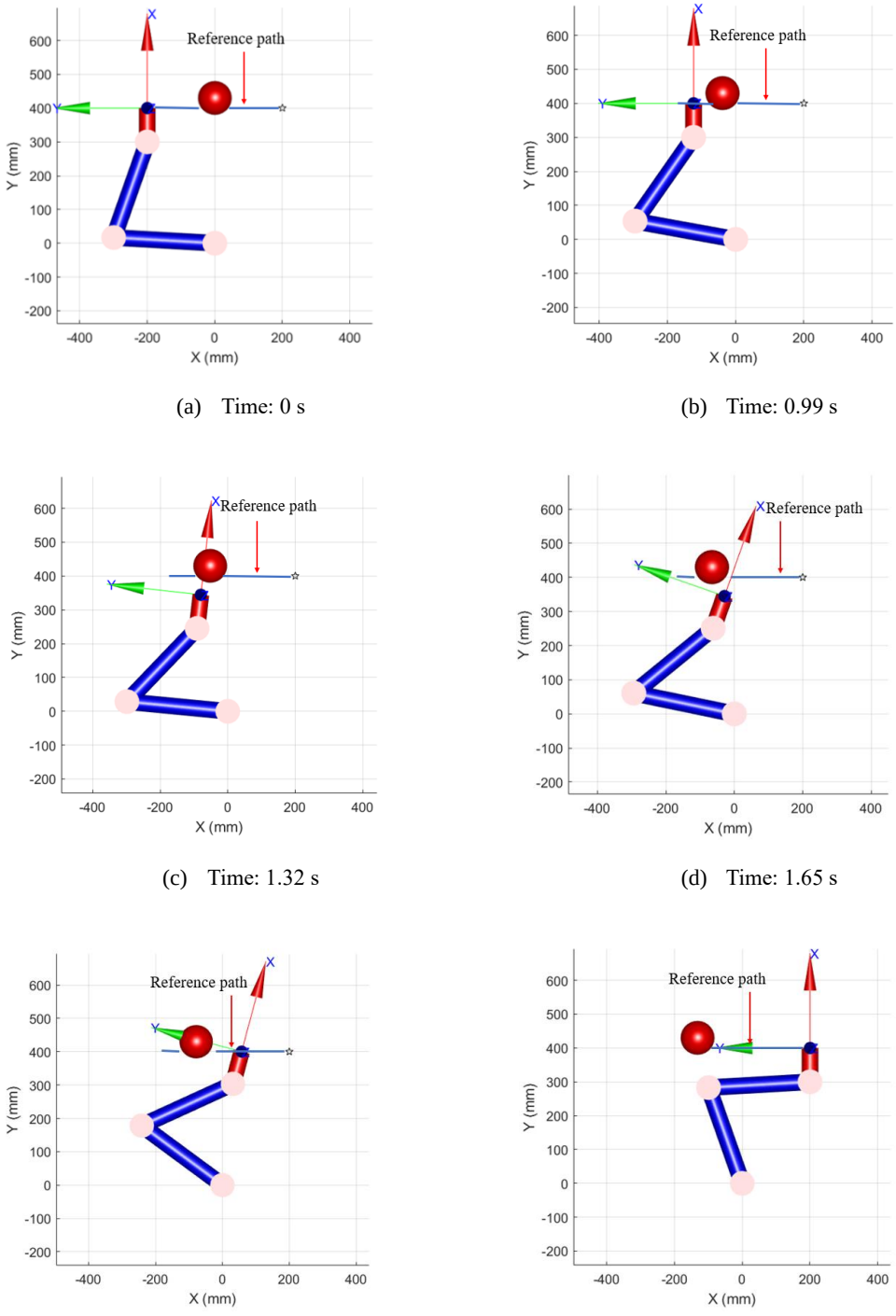


Figure 5.18: Joint angle, velocity and acceleration versus time for the planar robot avoiding the dynamic obstacle using the CLIK algorithm

5.4.2.5 NMPC collision avoidance method

We use the same simulation settings as in Section 5.4.2.4 to test the NMPC controller in this section. The prediction horizon $N_p = 4$ is used for the cost function (4.54). The weighting matrix that is for the position and orientation errors is set to $Q = diag\begin{pmatrix}1 & 1 & 100\end{pmatrix}$. Snapshots of the simulation are presented in Figure 5.19. The reference operation time is 3.3 s. The NMPC approach can control the operation time to be the same with reference time. This is an advantage over the CLIKFR algorithm. Then the normalized time is 1. However the execution time in MATLAB simulation is 70 s so it cannot be implemented in real-time.



(a) Time: 0 s (b) Time: 0.99 s

(c) Time: 1.32 s (d) Time: 1.65 s

(e) Time: 1.98 s (f) Time: 3.3 s

Figure 5.19: Snapshots of the simulation for the planar robot controlled by the NMPC algorithm avoiding a dynamic sphere obstacle

Figure 5.20 The closest distance between the robot surface and the surface of the dynamic sphere obstacle vs. time. Since the smallest value is -5.5×10^{-4} mm there will be a slight contact between the robot and obstacle. However, this can easily be avoided by increasing d_s by a small amount. The position error at the goal location is 0.01 mm, while the orientation error at the goal location is 0.04 degrees. The joint angles, velocities and accelerations are plotted vs. time in Figure 5.21. The velocities and accelerations are clearly less smooth than with the CLIKFR algorithm.

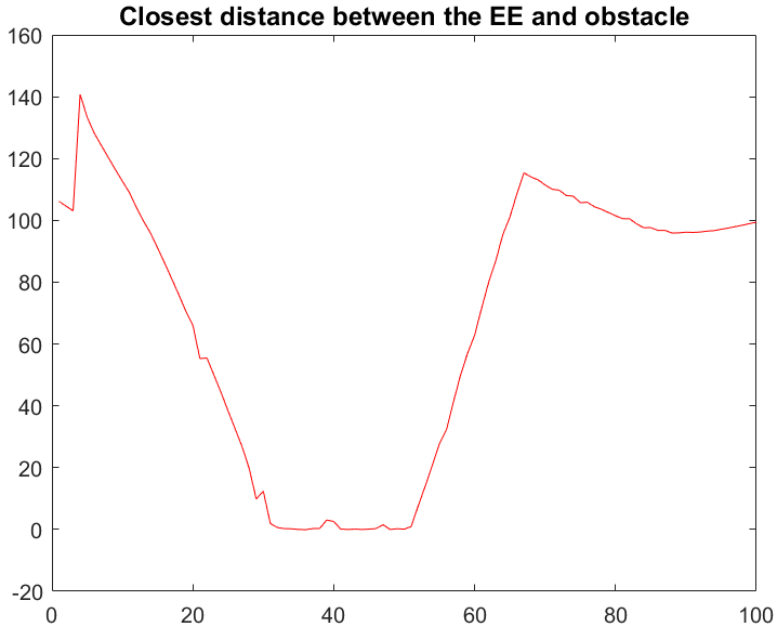


Figure 5.20: Closest distance between the robot surface and the surface of the dynamic obstacle for the planar robot controlled by the NMPC algorithm

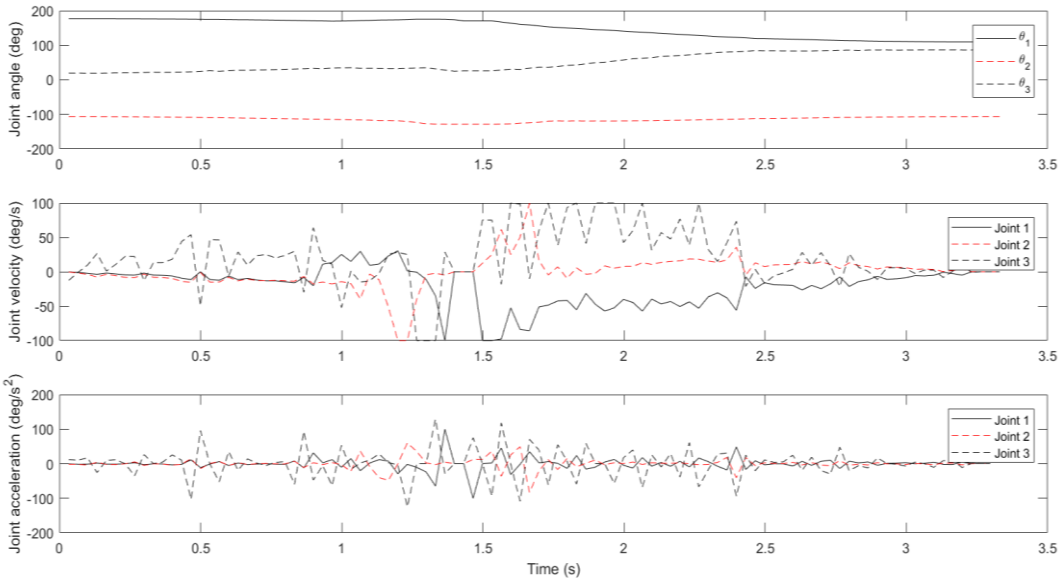


Figure 5.21: Joint angle, velocity and acceleration versus time for the planar robot avoiding the dynamic obstacle using the NMPC algorithm

5.5 Elfin simulation

5.5.1 Simulation settings

The simulation scenarios will consist of a Elfin 5 robot and a human sharing the workspace of the robot. The task assigned to the robot will be a straight-line motion between two predefined end-effector locations. While the robot completes its task, the human works beside it. When parts of human body become obstructive, the robot must focus on avoiding collision with the human while still pursuing its current goal location. The human position in space is continuously monitored by an Intel RealSense D415 sensor and modelled by the union of spheres and plane model, as has been discussed in

Section 5.4. To make the simulations more realistic, a point cloud of a human and a worktable covered with various color objects is used.

The simulation parameters and the initial conditions used in the test will be defined first. The sampling period of the simulation were chosen to agree with D415 sensor's updating frequency which is 30 Hz. This value gives a sampling period $T_s = 33.3$ ms. The duration of the reference path is set to 3.33 s. These values give a total number of 100 reference discrete times in the simulation. The effects of the robot delay also will be investigated by comparing simulation results of $\tau_d = 0$ to the results of $\tau_d = 2$. $\tau_d = 2$ refers to a delay of two samplings periods, or 66.6 ms.

The task assigned to the robot is composed of one trip between two goal locations, located at approximately 0.99 m apart in 3D space. The kinematics parameters of the Elfin 5 robot are given in Table 3.1. Frame zero of the D-H kinematics model that is attached to the fixed base of the robot was located at coordinates $X = 0$ mm, $Y = 0$ mm and $Z = 0$ mm in the work cell. The coordinates of point cloud captured by D415 sensor have been transformed from the sensor's frame to the robot's base frame.

Joint limits of Elfin 5 are shown in Table 5.3.

Table 5.3: Joint limits of the Elfin 5 robot

Joint		Angle (°)	Velocity (°/s)	Acceleration (°/s^2)
1	Max	180	85	250
	Min	-180	-85	-250
2	Max	135	85	250
	Min	-135	-85	-250
3	Max	156	85	250
	Min	-156	-85	-250
4	Max	180	85	250
	Min	-180	-85	-250
5	Max	180	85	250
	Min	-180	-85	-250
6	Max	180	85	250
	Min	-180	-85	-250

5.5.2 Static human limb collision avoidance

The scenario simulated in this section is carried out to investigate the behavior of the CLIKFR algorithm for avoiding part of a static human body, as shown in Figure 5.22. The human's arm remains stationary and will block the way of the robot if the

robot follows the reference trajectory. The initial settings for the simulations are summarized in Table 5.4.

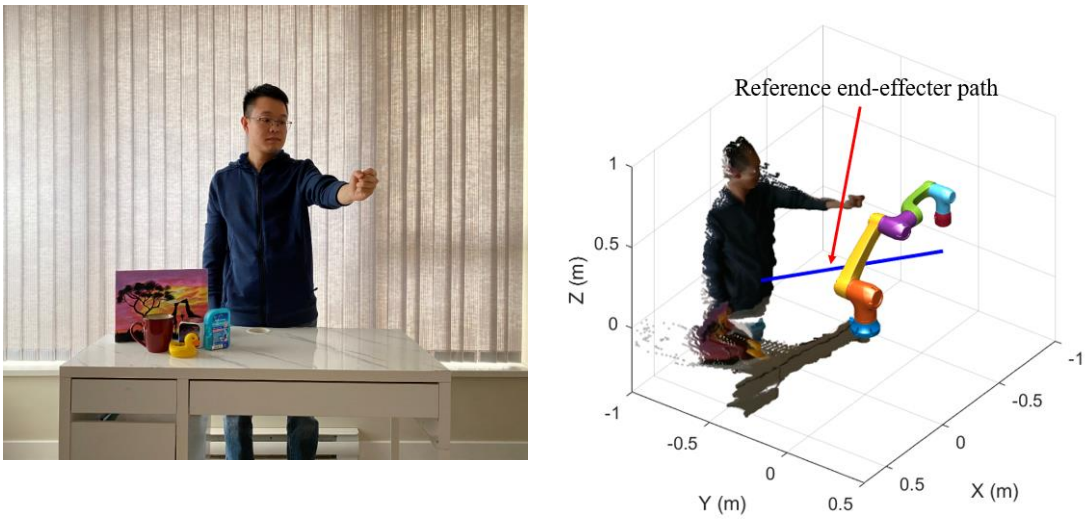


Figure 5.22: Elfin 5 robot and a static human limb simulation scenario of the CLIKFR algorithm

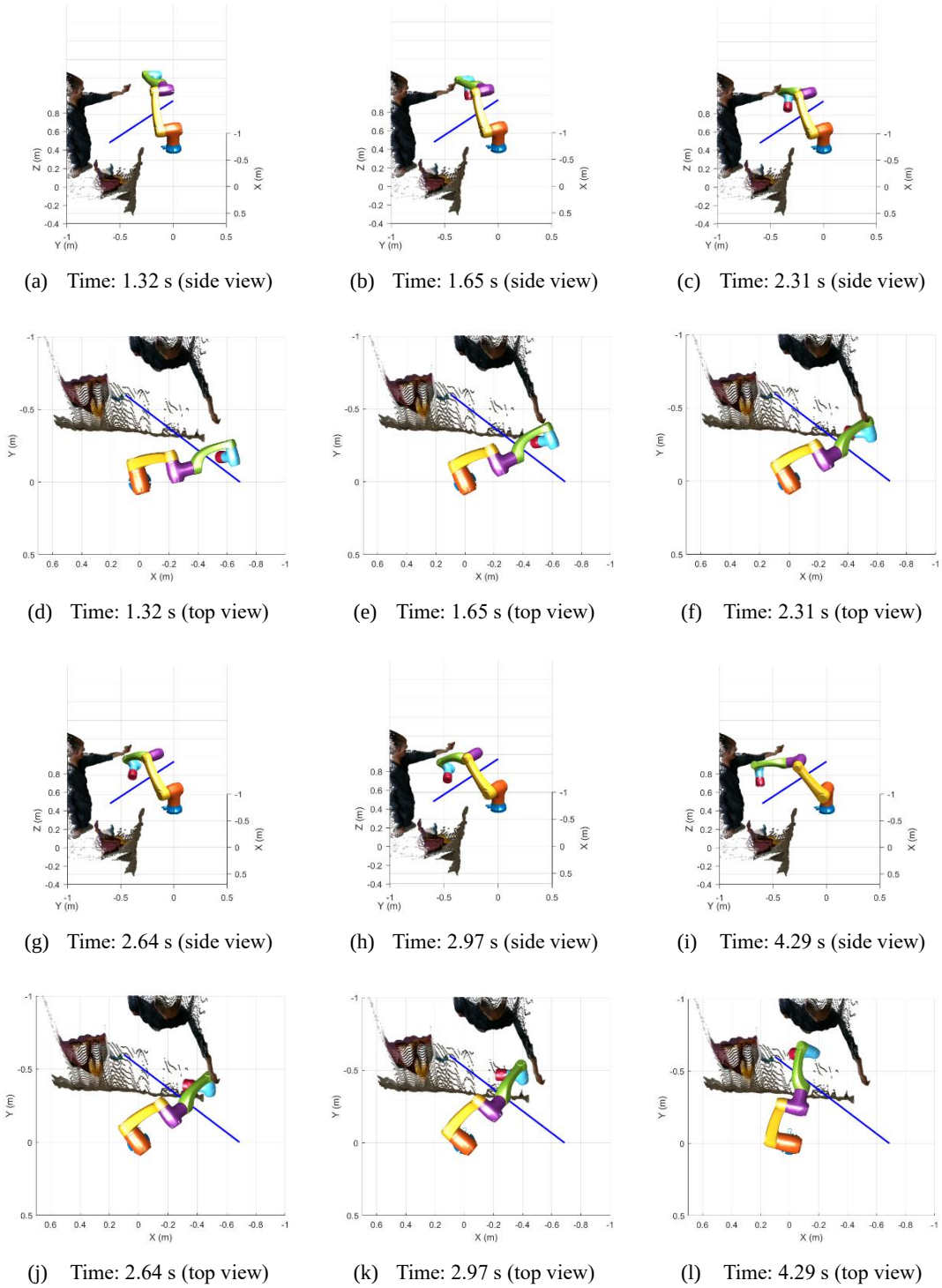
Table 5.4 Input parameter values used in static human limb collision avoidance simulations of the CLIKFR algorithm

Parameter Value	Parameter Description
$\mathbf{x}_{\mathbf{start}} = [-688.7 \quad 0 \quad 133.7 \quad -\pi \quad \pi \quad \pi]^T$	Start position and orientation of the end-effector
$\mathbf{x}_{\mathbf{goal}} = [100 \quad -600 \quad 133.7 \quad -\pi \quad \pi \quad \pi]^T$	Goal position and orientation of the end-effector
$T_s = 0.0333$ s	Sampling time
$N_{reference} = 100$	Reference discrete steps, equal to the reference duration of 3.33 s

$\tau_d = 0$	The number of sampling periods of delay
$\lambda = 100$	Damping factor in CLIKFR controller
$k_{\text{position}} = 1 \times 10^3$	Weight factor for end-effector's position errors
$k_{\text{orientation}} = 1 \times 10^4$	Weight factor for end-effector's orientation errors
$r_1 = 40, 50 \text{ and } 50 \text{ mm}$	Radius of robot's SSL model for links 6, 4 and 2, respectively
$r_2 = 50 \text{ mm}$	Radius of the spheres in the union of spheres human model
$d_s = 100 \text{ mm}$	Desired centre-to-centre distance between the robot centre line and obstacle centre
$d_p = 60 \text{ mm}$	Desired distance between the robot centre line and the plane obstacle
$e_{EE-threshold} = 0.1$	Error threshold to decide whether the end-effector reach the goal pose
$K_{es} = 1$	Gain of sphere collision avoidance error

The outputs of this simulation are presented in Figure 5.23. These plots suggest that collision avoidance path produced by the CLIKFR algorithm is efficient since the detour distance is small. The reference operation time is 3.33 s, while the collision avoidance operation time is 13.7 s. Then the normalized time is 4.16. The execution time in MATLAB simulation is 2.63 s, so this method is fast enough for real-time

collision avoidance applications.



(a) Time: 1.32 s (side view) (b) Time: 1.65 s (side view) (c) Time: 2.31 s (side view)

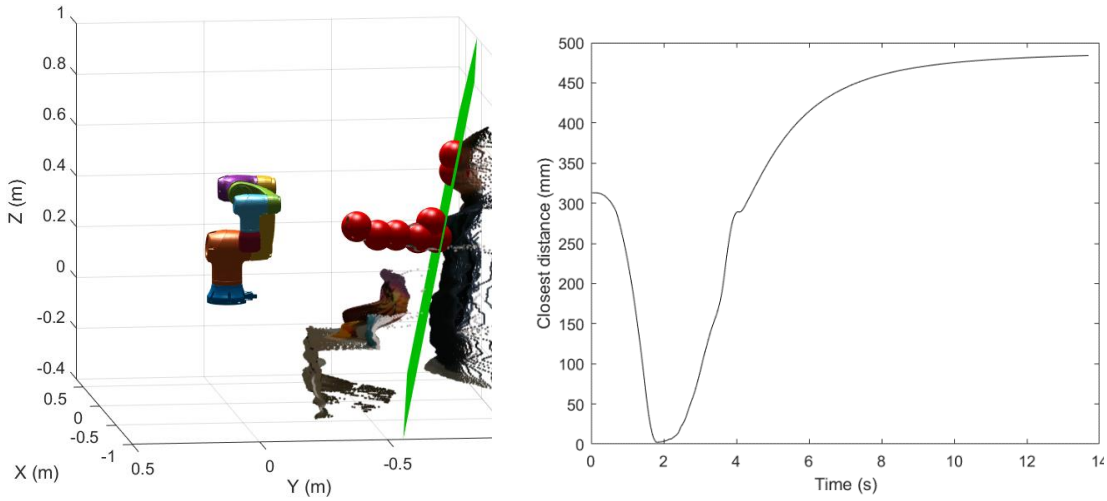
(d) Time: 1.32 s (top view) (e) Time: 1.65 s (top view) (f) Time: 2.31 s (top view)

(g) Time: 2.64 s (side view) (h) Time: 2.97 s (side view) (i) Time: 4.29 s (side view)

(j) Time: 2.64 s (top view) (k) Time: 2.97 s (top view) (l) Time: 4.29 s (top view)

Figure 5.23: Snapshots of the simulation for Elfin 5 controlled by the CLIKFR algorithm avoiding a static human limb

The plane and spheres human of this scenario is presented in Figure 5.24 (a). The closest distance between the robot surface and the surface of the static obstacle is plotted vs. time in Figure 5.24 (b). Since the smallest of this value is 2.3 mm, no obstacle robot collision occurred in the process. The position error at the goal location is 0.002 mm, while the orientation error at the goal location is 0.1 degrees. The joint angles, velocities and accelerations are plotted vs. time in Figure 5.25. The accelerations reach their limits at some times during the simulation, e.g. when $t \approx 3.5\,\text{s}$.



(a) (b)

Figure 5.24: (a) The plane and spheres human model. (b) Closest distance between the robot surface and the static human's limb for Elfin 5 robot controlled by the CLIKFR algorithm

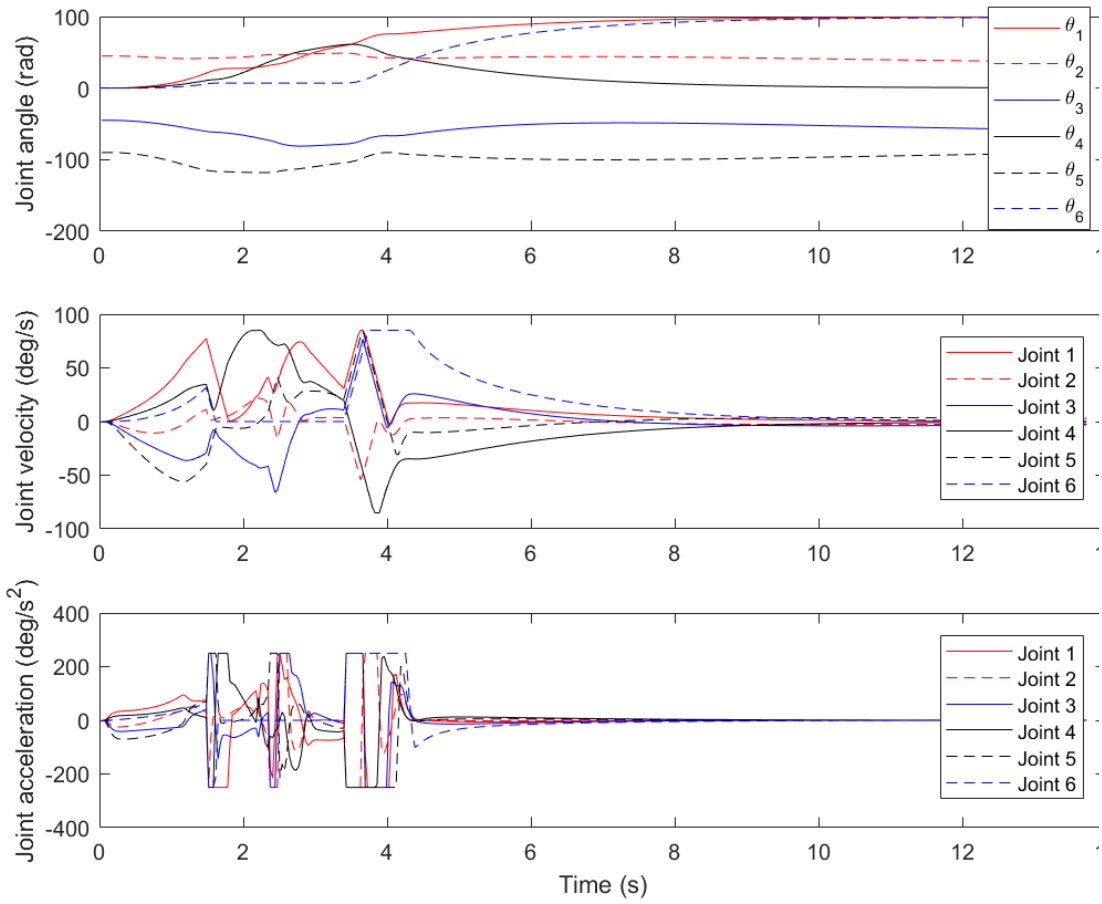


Figure 5.25: Joint angle, velocity and acceleration versus time for Elfin 5 avoiding the static human's limb using the CLIKFR algorithm

5.5.3 Human torso collision avoidance

The scenario simulated in this section is presented in Figure 5.26. A human operator in the work cell is picking something up while the robot is performing the straight-line motion task. The torso of human is blocking the reference end-effector path. The behavior of CLIKFR algorithm will be simulated. The initial settings for this simulation are the same as in section 5.5.2 (given in Table 5.4).

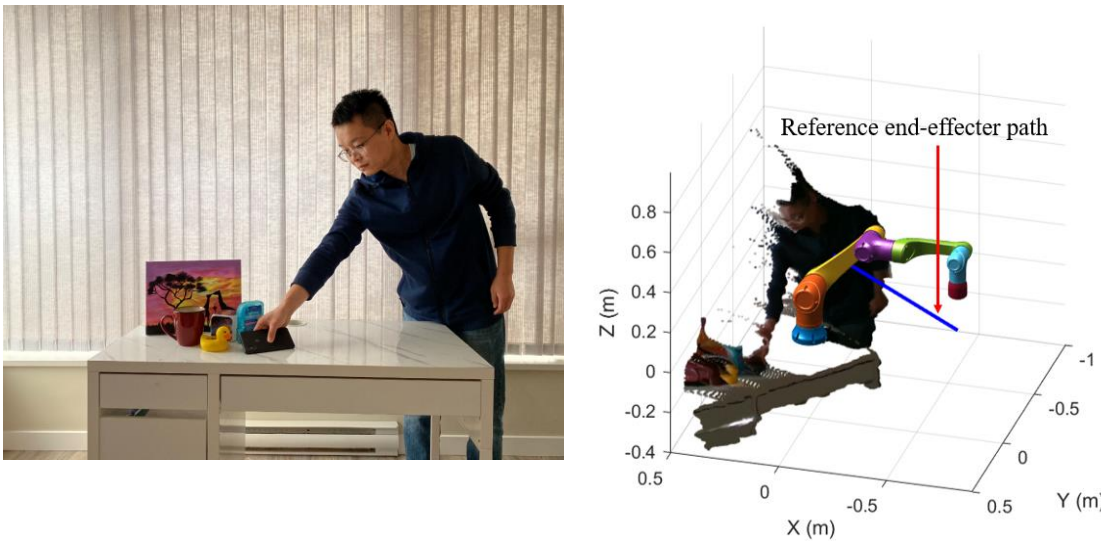
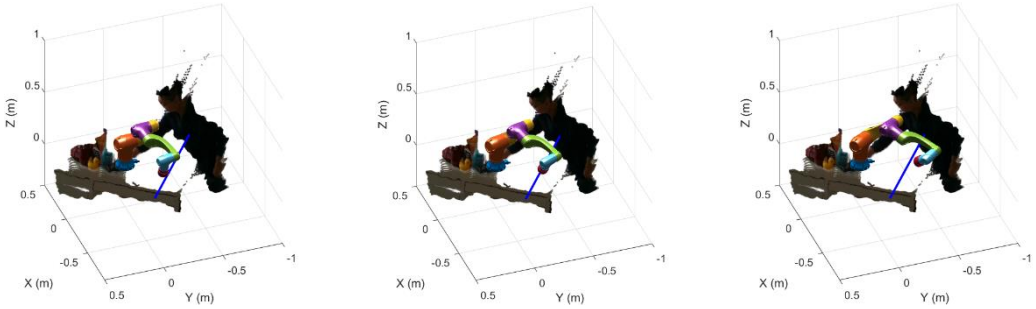
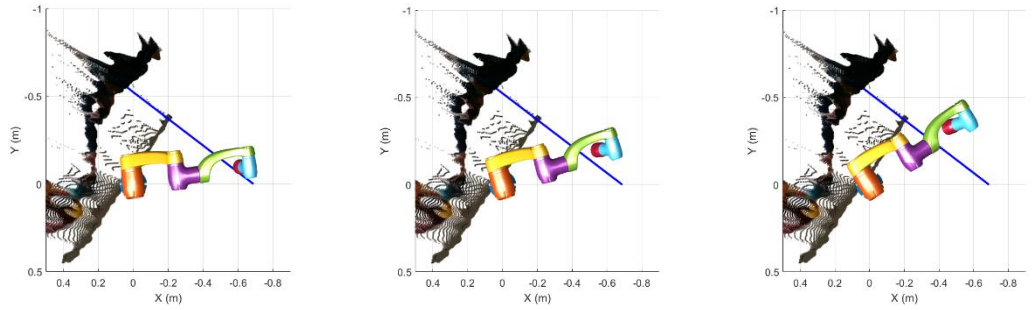


Figure 5.26: Elfin 5 robot and a human torso obstacle simulation scenario used with the CLIKFR algorithm

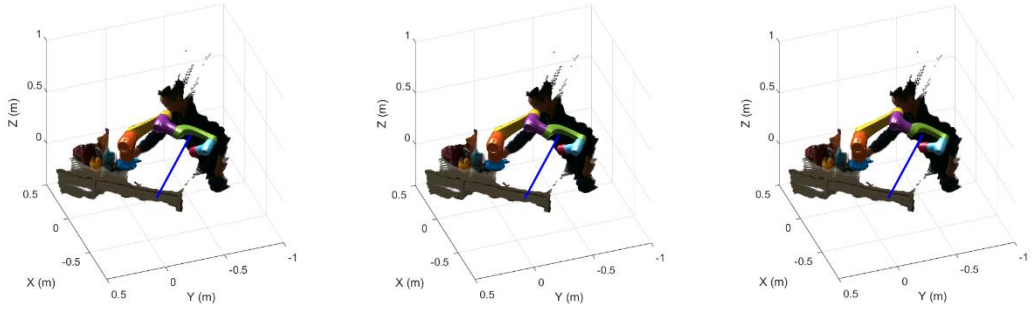
Figure 5.27 presents snapshots of the simulation. These plots suggest that the robot stop moving closer to the human torso after 1.98 s. Since the robot cannot figure out a path to reach the goal position while avoid collision with the human torso, the simulation was stopped manually after 13.2 s. The position error at the final location is 828 mm, while the orientation error at the final location is 57.29 degrees.



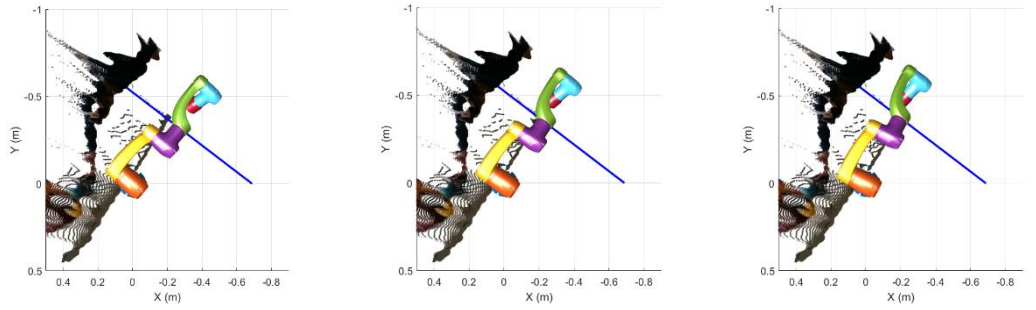
(a) Time: 0.99 s (side view) (b) Time: 1.32 s (side view) (c) Time: 1.65 s(side view)



(d) Time: 0.99 s (top view) (e) Time: 1.32 s (top view) (f) Time: 1.65 s (top view)

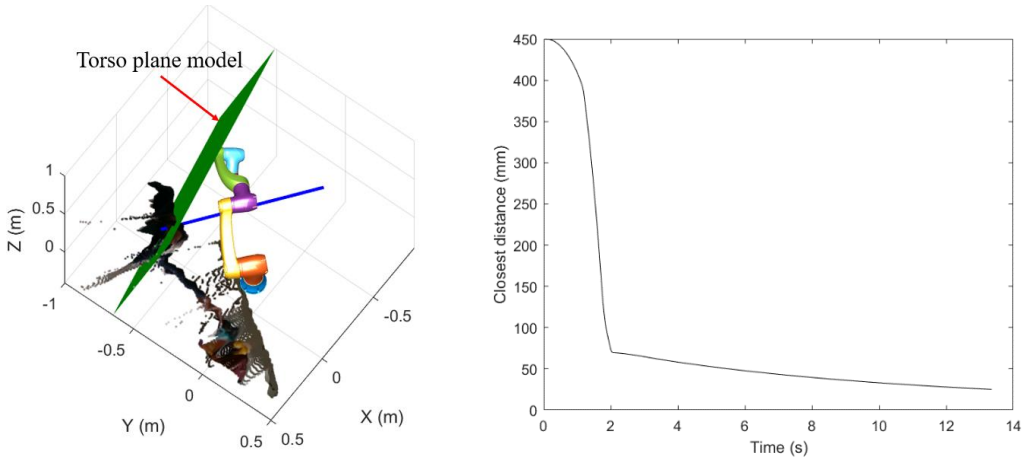


(g) Time: 1.98 s (side view) (h) Time: 6.6 s (side view) (i) Time: 13.2 s (side view)



(j) Time: 1.98 s (top view) (k) Time: 6.6 s (top view) (l) Time: 13.2 s (top view)

Figure 5.27: Snapshots of the simulation for Elfin 5 controlled by the CLIKFR algorithm avoiding a human torso



(a) (b)

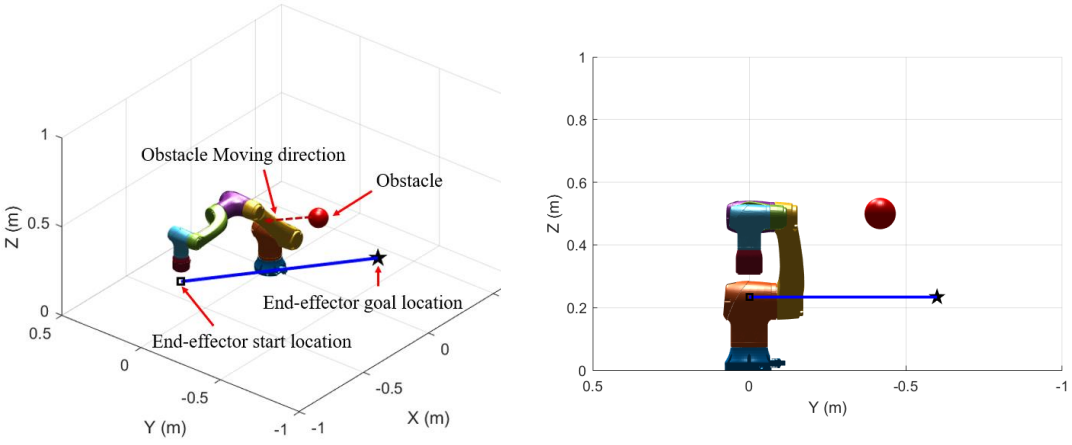
Figure 5.28: (a) Final pose of the robot and the human torso plane model (b) Closest distance between the robot surface and the human torso plane for Elfin 5 robot controlled by the CLIKFR algorithm

The final pose of the robot and the plane model of human torso are shown in Figure 5.28 (a), while the closest distance between the robot surface and the torso plane is plotted vs. time in Figure 5.28 (b). The minimum distance between the robot surface the human torso plane is approximately 25 mm. This result suggests that the CLIKFR algorithm can avoid collision with human torso effectively.

5.5.4 Dynamic human limb collision avoidance

The simulation in this section is to investigate the performance of the CLIKFR algorithm when the obstacle is moving. Since it is very difficult to control the motion

of human body properly in the real robot's absence, one artificial sphere obstacle with a radius of $r_2 = 50$ mm is deployed in this simulation instead of a real point cloud. The task assigned to the robot is the same as in the previous two sections. The initial position of the obstacle is located at $\mathbf{p}_{\mathbf{obsInitial}} = [-138.76 \quad -418.36 \quad 500]^T$ mm. The moving direction of the obstacle is along vector $\mathbf{v}_{\mathbf{obsDirection}} = [-788.7 \quad 600 \quad 0]^T$. This moving direction is parallel to the path from the end-effector's goal location to its start position. Figure 5.29 shows this simulation scenario. The behavior of the CLIKFR algorithm with various obstacle moving velocities, as well as the impact of robot delay τ_d, will be investigated.



(a) Perspective view

(b) Front view

Figure 5.29: Elfin 5 robot and a dynamic obstacle simulation scenario for the CLIKFR algorithm

The first sets of simulations are carried out with different obstacle velocities and $\tau_d = 0$. The obstacle velocities used are: $v_{obstacle}$ = 10, 20, 40, 60, 80, 100, 150 and

200 mm/s. The rest of parameter settings are the same with those in Table 5.4. The closest distance between the robot surface and the surface of the dynamic sphere obstacle with different velocities is plotted vs. time in Figure 5.30. The closest distance for each velocity is shown in Table 5.5.

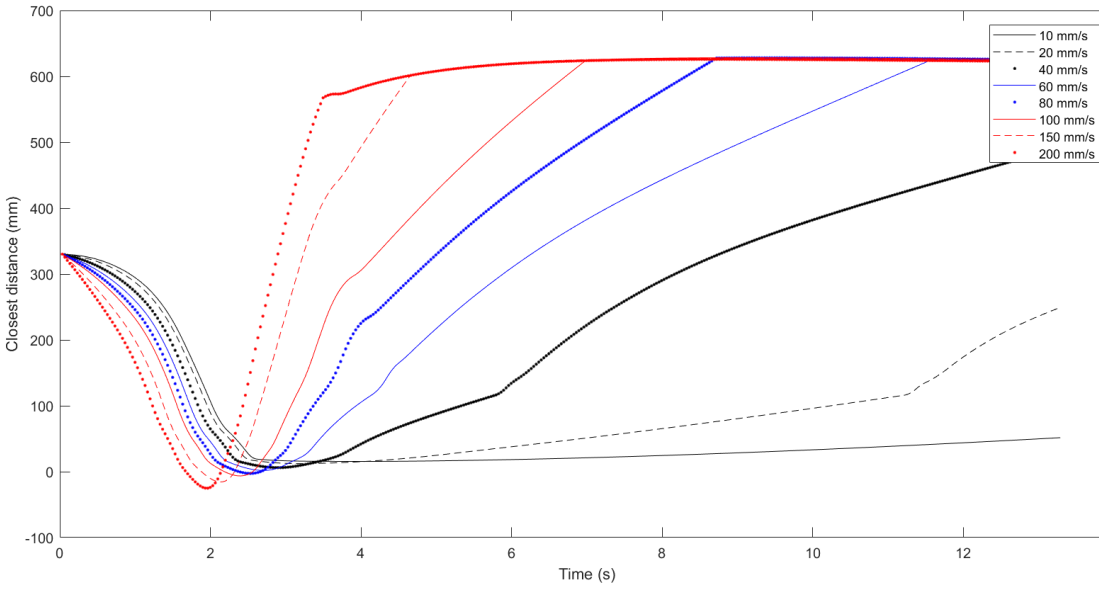


Figure 5.30: Closest distance between the robot surface and the surface of the dynamic sphere obstacle with different velocities controlled by of the CLIKFR algorithm

Table 5.5: The smallest value of distance as a function of obstacle velocity with the CLIKFR algorithm

Obstacle velocity (mm/s)	Separation Distance (mm)
10	15.6
20	12.2
40	6.1
60	1.8
80	-2.6
100	-6.5
150	-15.6
200	-25

Table 5.5 suggests that the separation distance becomes negative if the obstacle velocity is larger than 60 mm/s, which means the CLIKFR algorithm fails in those cases.

The second sets of simulations are carried out with the velocity of the obstacle equal to 20 mm/s. Robot delays of $\tau_d = 0$ and $\tau_d = 2$ are studied. The position error at the goal location is 0.002 mm, while the orientation error at the goal location is 5.4 degrees if $\tau_d = 0$. The position error at the goal location is 0.01 mm, while the orientation error at the goal location is 5.7 degrees if $\tau_d = 2$. Figure 5.40 presents the joint accelerations vs. time when $\tau_d = 0$,while Figure 5.41 shows the results when $\tau_d = 2$.

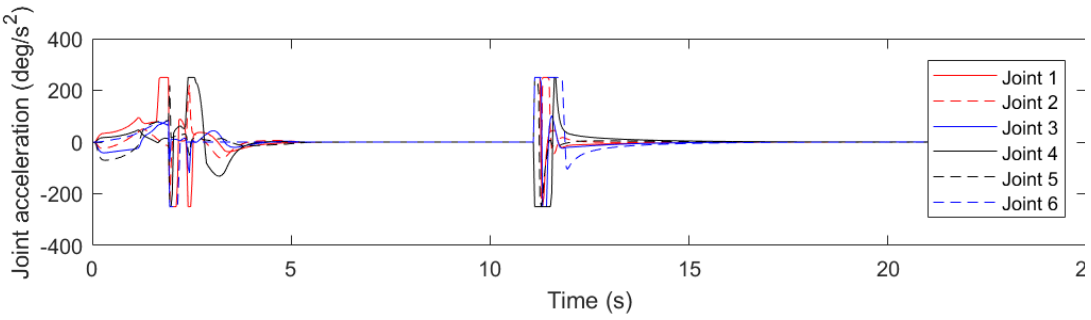


Figure 5.31: Joint accelerations versus time for Elfin 5 avoiding a dynamic obstacle using the CLIKFR algorithm with $\tau_d = 0$

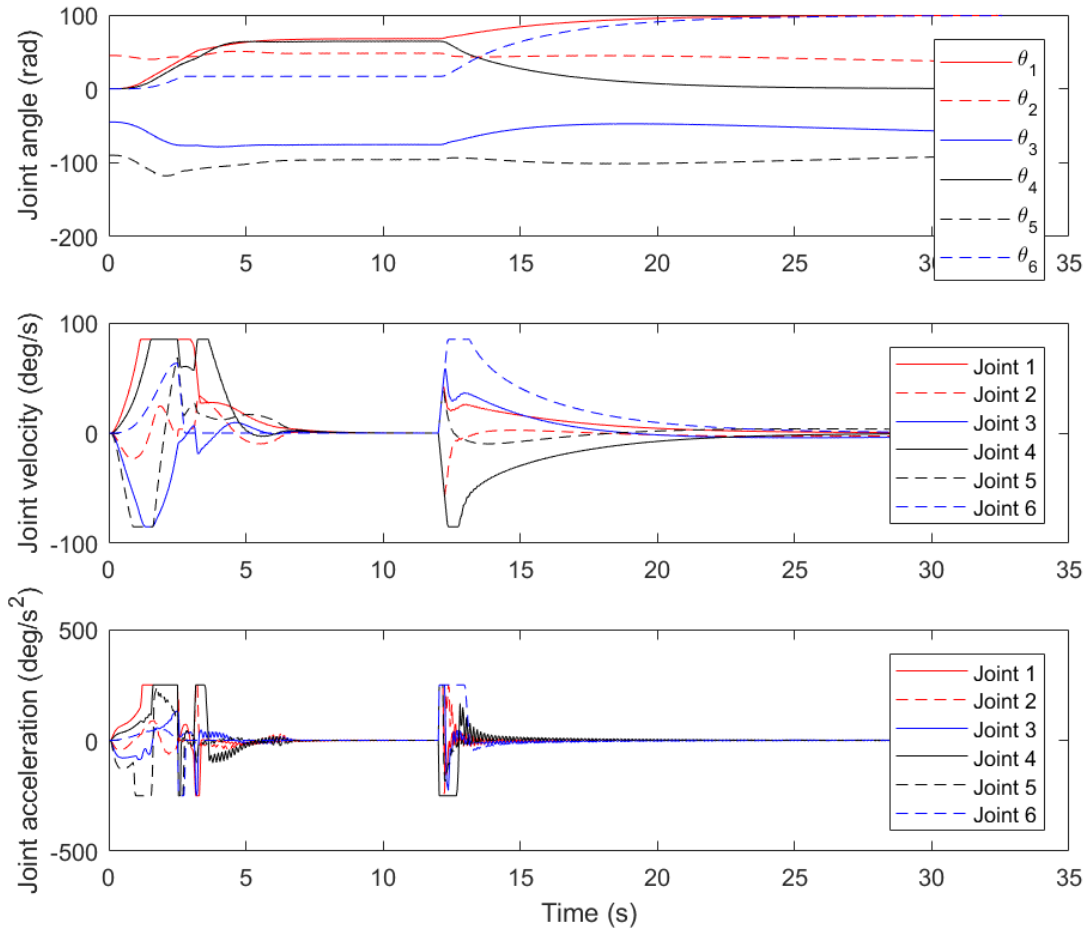


Figure 5.32: Joint accelerations versus time for Elfin 5 avoiding a dynamic obstacle using the CLIKFR algorithm with $\tau_d = 2$

These two plots show that the robot delay will result in chattering of joint acceleration. This is particularly noticeable during the periods 3.2-6.6 s and 12.5-17.3 s. The collision avoidance operation time increases from 22 s to 32.5 s.

5.5.5 NMPC collision avoidance for Elfin

The same simulation scenario as section 5.5.4 is used to test the NMPC controller in this section. The prediction horizon $N_p = 9$ is used with the cost function (4.54). The weighting matrix for the position and orientation errors is set to $Q = diag\begin{pmatrix}1 & 1 & 1 & 50 & 50 & 50\end{pmatrix}$. The velocity of the obstacle is $v_{obstacle} = 200$ mm/s with a robot delay $\tau_d = 2$. Figure 5.42 shows the closest distance between the robot surface and the surface of the dynamic spherical obstacle vs. time. The smallest value is 5 mm which means that the NMPC controller can guarantee the success of collision avoidance with a much higher obstacle velocity than the CLIKFR algorithm.

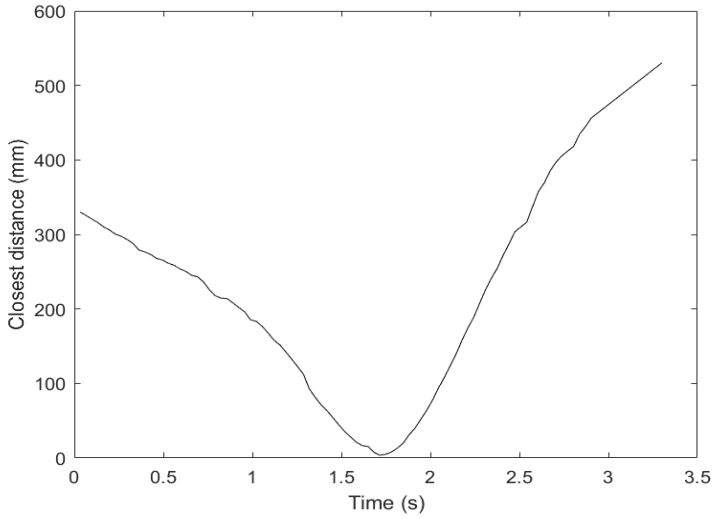


Figure 5.33: Closest distance between the robot surface and the dynamic obstacle for Elfin 5 robot controlled by the NMPC algorithm

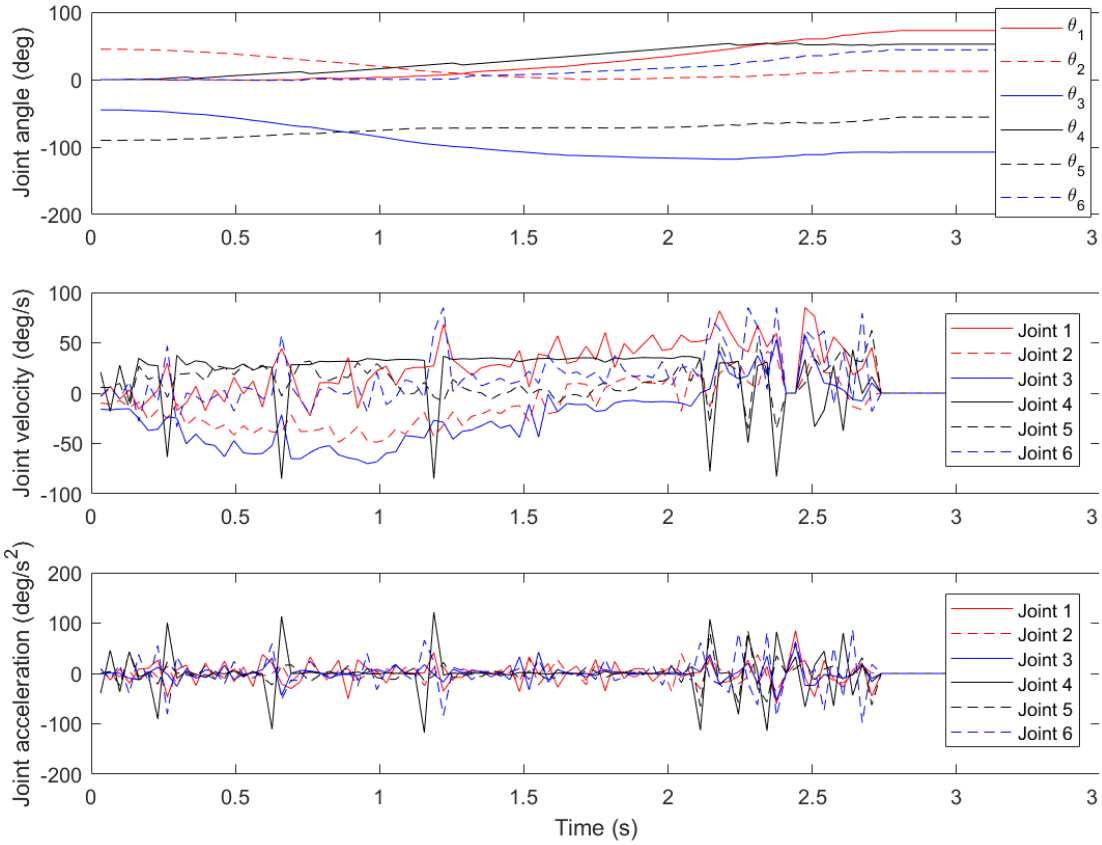


Figure 5.34: Joint angle, velocity and acceleration versus time for Elfin 5 avoiding the dynamic obstacle using NMPC algorithm

The joint angle, velocity and acceleration are plotted vs. time in Figure 5.43. The acceleration is smoother than with CLIKFR with $\tau_d = 2$ (recall Figure 5.41). The reference operation time in this simulation is 3.3 s. The normalized time is 1. The computation time in MATLAB simulation is 190 s, so it is again too slow for real-time implementation.

5.6 Summary of Results

In this chapter, the human modelling algorithm introduced in Chapter 3 has been implemented and tested using experimental point cloud data captured from a scene with a human and multiple color objects. The human skin detection algorithm was able to segment points that belong to the human's skin and generate a correct seed point for the region growing algorithm. The human point cloud was clustered by region growing and then voxelized to reduce the number of cloud points. Finally, the plane model for the human torso and union of spheres model for body parts that protrude in front of the plane have been generated successfully from the human point cloud. The execution time was 0.06 s which is fast enough for real-time implementation.

Five distinct scenarios have been simulated for a 3-DOF planar robot to evaluate the CLIKFR and NMPC algorithms presented in Chapter 4 for a relatively simple problem. As shown in Section 5.4.2.1, the CLIKFR algorithm can produce an efficient and smooth collision-free path when there is one static spherical obstacle. Its normalized time was 1.8. The actual run time in MATLAB was around 1 s for the 3.3 s simulated reference duration. The simulation outputs of Section 5.4.2.2 show that the CLIKFR algorithm can control the robot to perform collision with a plane obstacle and move the end-effector close to the goal position at the same time. The results from the wrist singularity scenario in Section 5.4.2.3 showed that the damping factor in the CLIKFR algorithm can overcome the singularity. For the dynamic obstacle scenario in Sections 5.4.2.4 and 5.4.2.5, the NMPC controller produced a smaller normalized time at the cost of much longer computation time compared to the CLIKFR algorithm.

Finally, four HRC scenarios were simulated for a 6-DOF Elfin 5 robot to study the behaviour of the collision avoidance system combining the human modelling algorithm and collision avoidance algorithm (i.e. CLIKFR or NMPC). Simulations in Section

5.5.2 and Section 5.5.3 demonstrate that the union of spheres model and plane model can protect the human arm and torso from being hit by the robot. These two simulations also suggest that the CLIKFR algorithm is computationally efficient enough for real-time implementation. Results from the dynamic obstacle simulations show the effect of the obstacle's velocity on the success or failure of the collision avoidance algorithm. A velocity of less than 60 mm/s is critical to guarantee a successful collision avoidance for the CLIKFR algorithm with the Elfin 5 robot. The robot's delay will also result in the chattering of joint acceleration with the CLIKFR. The NMPC algorithm simulation results show that it can handle higher human moving velocities and its normalized time is always smaller than that of CLIKFR, but its computational burden is much higher. All simulation results are shown in Table 5.5.

Table 5.5: Collection of CLIKFR and NMPC simulation results

Robot	Scenario	Normalized time	Execution time (s)	Minimum distance (mm)	Position error* (mm)	Orientation error* (deg)
Three-link planar	Static sphere	1.8	1.1	0.43	0.95	0.02
	Static plane	2	0.99	6.4	246.5	90
	Singularity	1.7	1.9	-	0.8	0.54
	Dynamic sphere	1.1	0.55	0.32	0.86	0.24
	NMPC	1	70	-5.5×10^{-4}	0.01	0.04
Elfin 5	Static human arm	4.16	2.63	2.3	0.002	0.1
	Static	4	2.58	25	328	56.15

	human torso					
	Dynamic sphere (60 mm/s)	4.4	0.94	1.8	0.003	0.57
	NMPC	1	190	5	27.5	0.62

*Position error and orientation error in this table refer to errors at the goal location.

Chapter 6 Conclusions and Recommendations

6.1 Summary and Conclusions

In this thesis, a collision avoidance system with an online trajectory generation algorithm for robot manipulators in dynamic environments was presented. It combines a human modelling algorithm with a collision avoidance algorithm. The system's objective is to reach the goal position while simultaneously avoiding collisions. In this system, the proposed human modelling algorithm is used to obtain a union of spheres and plane model of the human worker from the sensed RGB-D point cloud. The point cloud is captured at a frame rate of 30 Hz by an Intel RealSense D415 camera. The human model is computed from the point cloud in 0.06 s using MATLAB code running on a laptop. The links of the robot manipulator are represented by SSLs. These geometric models allow the separation distance between the human and the robot to be calculated quickly.

The CLIKFR algorithm solves the inverse kinematics problem and avoids collisions using an expanded version of the manipulator Jacobian matrix. The Cartesian task space error is corrected by adopting a closed-loop inverse kinematics algorithm. Collision avoidance tasks produce constraints that are systematically incorporated into this algorithm. This collision avoidance algorithm is applied to compute the commanded joint velocities from given end-effector velocities. A second collision avoidance algorithm using NMPC is designed as an alternative approach.

The collision avoidance system and the associated online trajectory generation algorithm were evaluated using simulations. Five distinct scenarios have been simulated for a 3-DOF planar robot to evaluate the CLIKFR and NMPC algorithms for a relatively

simple problem. Four HRC scenarios were simulated for a 6-DOF Elfin 5 robot to study the behaviour of the collision avoidance system combining the human modelling algorithm and collision avoidance algorithm.

The simulation results showed that the CLIKFR algorithm is fast enough to be applied in real-time. For a 6-DOF robot, a computational time of 2.63 seconds for a reference duration of 3.33 seconds has been achieved using MATLAB running on a laptop computer. The separation distance between the human and robot is always larger than zero for a static human operator, which means that no collisions occurred. Also, the smooth profiles of the joint angles and velocities vs. time can be achieved without difficulty by the robot. The NMPC algorithm has better performance than the CLIKFR algorithm when the dynamic obstacle is moving at higher velocities and when the simulated robot has a realistic time delay.

These simulations also revealed the limitations of the CLIKFR and NMPC algorithms. The first issue is related to the moving velocity of the obstacle. A velocity over 60 mm/s will result in the failure of the collision avoidance for the CLIKFR algorithm. The existence of robot delay will cause vibration for a moving obstacle with the CLIKFR algorithm. The major limitation of the NMPC algorithm is its computation time is too long to meet the real-time requirement. Moreover, the trajectory generated by the NMPC controller is not as smooth in general.

6.2 Recommendations for future research

To improve the overall performance of the system, some feature works are suggested. The following are recommendations for future research:

1) For this project only one depth sensor was used to oversee the HRC workspace. It is recommended that multiple sensors are used improve the performance of

the human sensing and eliminate dead spots. Experimenting with other 3D sensors to evaluate the performance of different camera types for HRC is also recommended.

2) The influence of different weight factors for the constraints tasks and the gain of sphere collision avoidance error in the CLIKFR algorithm should be investigated.
3) Due to the COVID-19 shutdown of the campus it was not possible to finish the implementation. Experimental validation should be done to verify the performance of the collision avoidance system using the CLIKFR algorithm.
4) Methods for speeding up the computation of the NMPC algorithm should also be investigated so it can also be experimentally validated.

References

Adams, R. and Bischof, L. (1994) 'Seeded Region Growing', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6), pp. 641–647. doi: 10.1109/34.295913.

Ajoudani, A. *et al.* (2018) 'Progress and prospects of the human–robot collaboration', *Autonomous Robots*. Springer US, 42(5), pp. 957–975. doi: 10.1007/s10514-017-9677-2.

Balan, L. and Bone, G. M. (2006) 'Real-time 3D collision avoidance method for safe human and robot coexistence', *IEEE International Conference on Intelligent Robots and Systems*, (October), pp. 276–282. doi: 10.1109/IROS.2006.282068.

Beckert, D., Pereira, A. and Althoff, M. (2018) 'Online verification of multiple safety criteria for a robot trajectory', *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, 2018-Janua(Cdc), pp. 6454–6461. doi: 10.1109/CDC.2017.8264632.

Bosscher, P. and Hedman, D. (2011) 'Real-time collision avoidance algorithm for robotic manipulators', *Industrial Robot*, 38(2), pp. 186–197. doi: 10.1108/01439911111106390.

Chu, P. M., Sung, Y. and Cho, K. (2019) 'Generative Adversarial Network-Based Method for Transforming Single RGB Image into 3D Point Cloud', *IEEE Access*. IEEE, 7, pp. 1021–1029. doi: 10.1109/ACCESS.2018.2886213.

Corrales, J. A., Candelas, F. A. and Torres, F. (2011) 'Safe human-robot interaction based on dynamic sphere-swept line bounding volumes', *Robotics and*

Computer-Integrated Manufacturing. Elsevier, 27(1), pp. 177–185. doi: 10.1016/j.rcim.2010.07.005.

Cortelazzo, C. D. M. P. Z. G. M. (2012) *Time-of-flight cameras and Microsoft Kinect*. 1st ed. New York: SpringerBriefs in Electrical and Computer Engineering.

Craig, J. J. (2004) *Introduction to Robotics: Mechanics and Control 3rd*. Prentice Hall.

Darwish, W. *et al.* (2019) 'A Robust Calibration Method for Consumer Grade RGB-D Sensors for Precise Indoor Reconstruction', *IEEE Access*, 7, pp. 8824–8833. doi: 10.1109/ACCESS.2018.2890713.

Demand for robot cooks rises as kitchens combat COVID-19 (2020) *https://abcnews.go.com/Technology/wireStory/demand-robot-cooks-rises-kitchens-combat-covid-19-71768876*.

Executive Summary World Robotics 2019 Industrial Robots (2019) *https://ifr.org/downloads/press2018/Executive%20Summary%20WR%202019%20Industrial%20Robots.pdf*.

Fast-Berglund, Å. *et al.* (2016) 'Evaluating Cobots for Final Assembly', *Procedia CIRP*. Elsevier B.V., 44, pp. 175–180. doi: 10.1016/j.procir.2016.02.114.

Flacco, F. *et al.* (2012) 'A depth space approach to human-robot collision avoidance', *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 338–345. doi: 10.1109/ICRA.2012.6225245.

Flacco, F. *et al.* (2015) 'A Depth Space Approach for Evaluating Distance to Objects: with Application to Human-Robot Collision Avoidance', *Journal of Intelligent and Robotic Systems: Theory and Applications*, 80, pp. 7–22. doi:

10.1007/s10846-014-0146-2.

Gerdts, M. *et al.* (2012) 'Path planning and collision avoidance for robots', *Numerical Algebra, Control and Optimization*, 2(3), pp. 437–463. doi: 10.3934/naco.2012.2.437.

Golub, G. H. and Reinsch, C. (1970) 'Singular Value Decomposition and Least Squares Solutions', *Numer. Math. 14, 403–420*.

Haddadin, S. *et al.* (2010) 'Real-time reactive motion generation based on variable attractor dynamics and shaped velocities', *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 3109–3116. doi: 10.1109/IROS.2010.5650246.

Hartenberg, J. and Denavit, R. (1964) *Kinematic Synthesis of Linkages*. New York: McGraw-Hill,.

Khaloo, A. and Lattanzi, D. (2017) 'Robust normal estimation and region growing segmentation of infrastructure 3D point cloud models', *Advanced Engineering Informatics*. Elsevier Ltd, 34, pp. 1–16. doi: 10.1016/j.aei.2017.07.002.

Khatib, O. (1985) 'Real-time obstacle avoidance for manipulators and mobile robots', *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 500–505. doi: 10.1109/ROBOT.1985.1087247.

Kohrt, C. *et al.* (2013) 'An online robot trajectory planning and programming support system for industrial use', *Robotics and Computer-Integrated Manufacturing*. Elsevier, 29(1), pp. 71–79. doi: 10.1016/j.rcim.2012.07.010.

Krämer, M. *et al.* (2020) 'Model predictive control of a collaborative manipulator considering dynamic obstacles', *Optimal Control Applications and Methods*,

(December 2018), pp. 1211–1232. doi: 10.1002/oca.2599.

Kulić, D. and Croft, E. A. (2005) 'Safe planning for human-robot interaction', *Journal of Robotic Systems*, 22(7), pp. 383–396. doi: 10.1002/rob.20073.

Lasota, P. A., Rossano, G. F. and Shah, J. A. (2014) 'Toward safe close-proximity human-robot interaction with standard industrial robots', *IEEE International Conference on Automation Science and Engineering*. IEEE, 2014-Janua, pp. 339–344. doi: 10.1109/CoASE.2014.6899348.

Liu, C. and Tomizuka, M. (2016) 'Algorithmic safety measures for intelligent industrial co-robots', *Proceedings - IEEE International Conference on Robotics and Automation*. IEEE, 2016-June(i), pp. 3095–3102. doi: 10.1109/ICRA.2016.7487476.

De Luca, A. *et al.* (2006) 'Collision detection and safe reaction with the DLR-III lightweight manipulator arm', *IEEE International Conference on Intelligent Robots and Systems*, pp. 1623–1630. doi: 10.1109/IROS.2006.282053.

Mainprice, J. *et al.* (2011) 'Planning human-aware motions using a sampling-based costmap planner', *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 5012–5017. doi: 10.1109/ICRA.2011.5980048.

Mainprice, J. and Berenson, D. (2013) 'Human-robot collaborative manipulation planning using early prediction of human motion', *IEEE International Conference on Intelligent Robots and Systems*. IEEE, pp. 299–306. doi: 10.1109/IROS.2013.6696368.

Martínez-Salvador, B., Pérez-Francisco, M. and Del Pobil, A. P. (2003) 'Collision Detection between Robot Arms and People', *Journal of Intelligent and Robotic Systems: Theory and Applications*, 38(1), pp. 105–119. doi:

10.1023/A:1026252228930.

Matheson, E. *et al.* (2019) 'Human-robot collaboration in manufacturing applications: A review', *Robotics*, 8(4), pp. 1–25. doi: 10.3390/robotics8040100.

Morato, C. *et al.* (2014) 'Toward safe human robot collaboration by using multiple kinects based real-time human tracking', *Journal of Computing and Information Science in Engineering*, 14(1), pp. 1–9. doi: 10.1115/1.4025810.

Perez-D'Arpino, C. and Shah, J. A. (2015) 'Fast target prediction of human reaching motion for cooperative human-robot manipulation tasks using time series classification', *Proceedings - IEEE International Conference on Robotics and Automation*, 2015-June(June), pp. 6175–6182. doi: 10.1109/ICRA.2015.7140066.

Rabbani, T., van den Wildenberg, F. and Vosselman, G. (2006) 'Segmentation of point clouds using smoothness constraint', *International archives of photogrammetry, remote sensing and spatial information sciences*, 36(5), pp. 248–253.

Ragaglia, M., Zanchettin, A. M. and Rocco, P. (2018) 'Trajectory generation algorithm for safe human-robot collaboration based on multiple depth sensor measurements', *Mechatronics*. Elsevier, 55(May 2017), pp. 267–281. doi: 10.1016/j.mechatronics.2017.12.009.

Safeea, M. and Neto, P. (2019) 'Minimum distance calculation using laser scanner and IMUs for safe human-robot interaction', *Robotics and Computer-Integrated Manufacturing*. Elsevier Ltd, 58(February), pp. 33–42. doi: 10.1016/j.rcim.2019.01.008.

Sciavicco, L. and Siciliano, B. (1988) 'A Solution Algorithm to the Inverse Kinematic Problem for Redundant Manipulators', *IEEE Journal on Robotics and*

Automation, 4(4), pp. 403–410. doi: 10.1109/56.804.

Shaik, K. B. *et al.* (2015) 'Comparative Study of Skin Color Detection and Segmentation in HSV and YCbCr Color Space', *Procedia Computer Science*. Elsevier Masson SAS, 57, pp. 41–48. doi: 10.1016/j.procs.2015.07.362.

Siciliano, B. (2009) *Robotics: Modeling, Planning, and Control, IEEE Robotics and Automation Magazine*. doi: 10.1109/MRA.2009.934833.

Spong, M. W., Hutchinson, S. and Vidyasagar, M. (2006) 'Robot modeling and control', pp 65-103.

Tan, J. T. C. *et al.* (2010) 'Safety strategy for human-robot collaboration: Design and development in cellular manufacturing', *Advanced Robotics*, 24(5–6), pp. 839–860. doi: 10.1163/016918610X493633.

Todd, D. J. (1986) *Fundamentals of robot technology: An introduction to industrial robots, teleoperators and robot vehicles*. 1st edn.

Torr, P. H. S. and Zisserman, A. (2000) 'MLESAC: A new robust estimator with application to estimating image geometry', *Computer Vision and Image Understanding*, 78(1), pp. 138–156. doi: 10.1006/cviu.1999.0832.

Universal Robots (2018) *Cobot-application on the engine assembly line*. Available at: https://www.universal-robots.com/case-stories/opel/[Online; accessed 8-October-2018].

Villani, V. *et al.* (2018) 'Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications', *Mechatronics*. Elsevier, 55(February), pp. 248–266. doi: 10.1016/j.mechatronics.2018.02.009.

Wampler, C. W. and Leifer, L. J. (1988) 'Applications of damped least-squares methods to resolved-rate and resolved-acceleration control of manipulators', *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME*, 110(1), pp. 31–38. doi: 10.1115/1.3152644.

Whitney, D. E. (1969) 'Resolved Motion Rate Control of Manipulators and Human Prostheses', *IEEE Transactions on Man-Machine Systems*, 10(2), pp. 47–53. doi: 10.1109/TMMS.1969.299896.

Xiang, J., Zhong, C. and Wei, W. (2012) 'A varied weights method for the kinematic control of redundant manipulators with multiple constraints', *IEEE Transactions on Robotics*. IEEE, 28(2), pp. 330–340. doi: 10.1109/TRO.2011.2173834.

Xiao, Z. and Wenming, H. (2009) 'Kd-tree Based Nonuniform Simplification of 3D Point Cloud', in. 2009 Third International Conference on Genetic and Evolutionary Computing.

Ye, M. *et al.* (2013) 'A Survey on Human Motion Analysis', *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications*, 8200, pp. 149–187. doi: 10.1007/978-3-642-44964-2_8.

Yoshida, E., Yokoi, K. and Gergondet, P. (2010) 'Online replanning for reactive robot motion: Practical aspects', *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*. IEEE, pp. 5927–5933. doi: 10.1109/IROS.2010.5649645.

Zanchettin, A. M. *et al.* (2016) 'Safety in Human-Robot Collaborative Manufacturing Environments: Metrics and Control', *IEEE Transactions on*

Automation Science and Engineering, 13(2), pp. 882–893. doi: 10.1109/TASE.2015.2412256.

Zarit, B. D., Super, B. J. and Quek, F. K. H. (1999) 'Comparison of five color models in skin pixel classification', *Proceedings - International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, RATFG-RTS 1999*, pp. 58–63. doi: 10.1109/RATFG.1999.799224.

Zube, A. (2015) 'Cartesian nonlinear model predictive control of redundant manipulators considering obstacles', *Proceedings of the IEEE International Conference on Industrial Technology*. IEEE, 2015-June(June), pp. 137–142. doi: 10.1109/ICIT.2015.7125089.