Real-Time Gesture-Based Posture Control of a Manipulator

Guillaume Plouffe

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science
in
Electrical and Computer Engineering

University of Ottawa

January 2020

Abstract

Reaching a target quickly and accurately with a robotic arm containing multiple joints while avoiding moving and fixed obstacles can be a daunting (and sometimes impossible) task for any user behind the remote control. Current existing solutions are often hard to use and to scale for all user body types and robotic arm configurations. In this work, we propose a vision-based gesture recognition approach to naturally control the overall posture of a robotic arm using human hand gestures and an inverse kinematic exploration approach using the FABRIK algorithm. Three different methods are investigated to intuitively control a robotic arm's posture in real-time using depth data collected by a Kinect sensor. Each of the posture control methods are users scalable and compatible with most existing robotic arm configurations. In the first method, the user's right index fingertip position is mapped to compute the inverse kinematics on the robot. The inverse kinematics solutions are displayed in a graphical interface. Using this interface and the left hand, the user can intuitively browse and select a desired robotic arm posture. In the second method, the user's right index fingertip position and finger direction are respectively used to determine the end-effector position and an attraction point position. The latter enables the control of the robotic arm posture. In the third method, the user's right index finger is mapped to compute the inverse kinematics on the robot. Using static gesture with the same hand, the user's right index finger can be transformed into a virtual pen that can trace the form of the desired robotic arm posture. The trace can be visualized in real-time on a graphical interface. A search is then performed using an inverse kinematic exploration and the Dynamic Time Warping algorithm to select the closest matching possible posture. In the last two proposed methods, different search strategies to optimize the speed and the inverse kinematic exploration coverage are proposed. Using a combination of Greedy Best First search and an efficient selection of input postures based on the FABRIK's algorithm characteristics, these optimizations allow for smoother and more accurate posture control of the robotic arm. The performance of these real-time natural human control approaches is evaluated for precision and speed against static (i.e. fixed) and dynamic (i.e. moving) obstacles in a simulated experiment. An adaptation of the vision-based gesture recognition system to

operate the AL5D robotic arm was also implemented to conduct further evaluation in a real-world environment. The results showed that the first and third methods were better suited for obstacle avoidance in static environments not requiring continuous posture changes. The second method gave excellent results in the dynamic environment experience and was able to complete a challenging pick and place task in a difficult real-world environment with static constraints.

Acknowledgements

I would like to thank Dr. Pierre Payeur and Dr. Ana-Maria Cretu for their guidance, motivation and support over the course of this research project and throughout the development of the related publication.

I would like to thank my loving wife for her support and our two wonderful children that entered our world during the course of this project.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background

The action of remotely controlling a manipulator is of particular interest when the environment involves a separation, either physical or spatial, between the manipulator and the controller. Such applications are usually referred to as teleoperations [1]. An example of physical separation could be as simple as a protective glass window behind which hazardous materials need to be confined during manipulation. A spatial separation could be the astronomical distances involved in space exploration. Teleoperation involves a local site where the user — with the help of input/output devices — operates a device located at a remote site. In the context of telerobotics, typical input/output devices are joysticks, monitors, keyboards, haptic devices, exoskeletons, digital gloves, wearables, markers or cameras, while the object under control at the remote site is a robotic device. Various elements of the robot can be remotely operated, including its position, velocity and acceleration, while information about forces exerted on the servomotors articulating the robotic arm joints in the remote environment can be returned to the user. When information goes both ways between the local and remote sites, the teleoperation is said to be bilateral, otherwise the control is unilateral.

As explained in [1], a telerobotic system can be defined along a spectrum of architectures going from "direct control" to "supervisory control". Between these boundaries lies a blend of both. This is where most system architectures, referred as "shared control", are found. In a pure direct control approach, the user commands the remote robot at the lowest level, directly controlling the motions of the remote robot (i.e. positions and kinematics) like a puppet master. In such a system, there is no intermediary layer whatsoever that could influence the end decision based on available information, such as sensor analytics and artificial intelligence support. On the contrary, in a pure supervisory control system, the user commands the remote robot at the highest level. For example, the user asks the robot to pick up a rock that lies nearby without guiding the process. It is up to the remote robot to gather the necessary information

from its multiple sensors about the remote environment in order to be able to precisely locate and generate an optimized trajectory required to reach and pick up the rock, while avoiding the many obstacles that could interfere along the way. This sounds like a faster solution, less susceptible to human error. Unfortunately, as much as technology and artificial intelligence have evolved over the years, the recognition speed at which a human brain can process and recognize complex obstacles geometry, and hazardous and unpredictable events is still unmatched. When confronted with unplanned situations that the robot's artificial intelligence is not programmed to handle on site, it is often impossible to add a new iteration in the development of the robot program due to time or resource constraint reasons. This is why most solutions favor shared control architectures.

As most solutions involve a user manipulating some sort of robotic device (joystick, haptics, exoskeleton, etc.) at the local site, they are often referred to as "master-slave" systems where the master corresponds to the local robotic device and the slave corresponds to the robot under control at the remote site. However, in the special case of solutions involving gesture recognition through touchless sensors (IR or/and RGB cameras), most publications we came across instead use the term "gesture-based" systems.

1.2 Problem statement

The ability of a human user to naturally control complex systems in real-time with minimal cognitive strain, physical constraints and physical effort has been the focus of research for many decades in the field of robotics. While controlling a multiple—joint robotic arm is possible without constraints in time and space, it becomes extremely difficult when trying to quickly and accurately reach a target while avoiding moving or fixed obstacles. A human user can control the movement of their arm and finger joints with almost no effort within a constrained space. Hence, the human mind capability to rapidly make a spatial model of his intended action and send the resulting movements command to his own arms and fingers in the real world provides an interesting avenue for exploring new approaches for the movement control of a robotic arm.

Many current solutions, detailed in Chapter 2, provide control over all robotic joints. However, they often have one or many of the following problems or limitations:

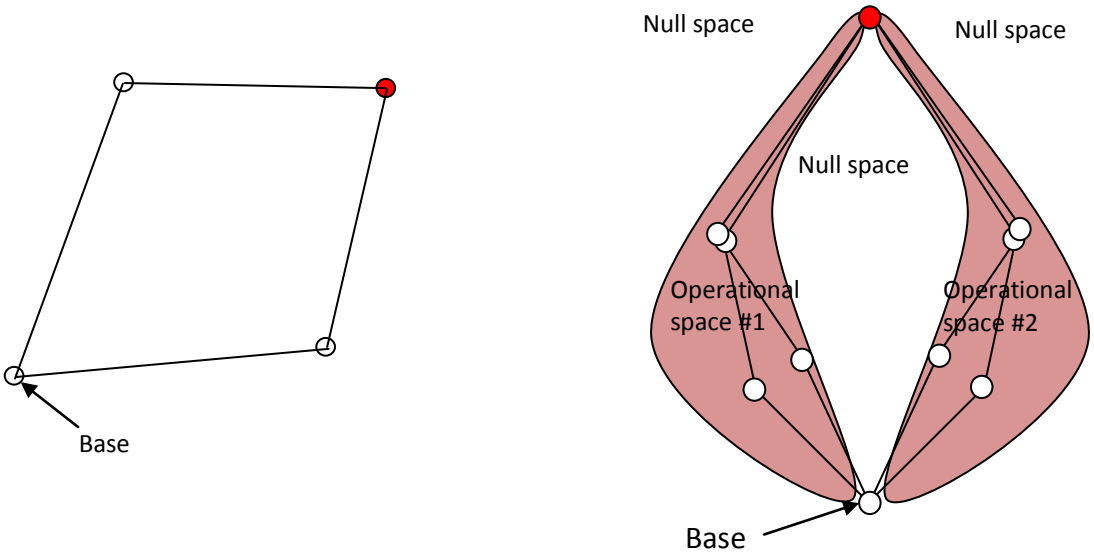
1. Complex in terms of installation or usability
2. Costly in terms of resources or scalability (the device or system requires adaptation to each individual user)
3. Prone to error due to complexity (large number of sensors, usability, etc.) or design (e.g. displacement of exoskeleton over the user's arm during use)
4. Restricted compatibility in terms of type of robotic arm configuration
5. Restricted applications due to limited control
6. Tedious, unnatural or non-intuitive control
7. Requirement of regular calibration
8. Invasive to the user

While current solutions found in the literature are able to overcome most of the problems and limitations mentioned above, so far none has been able to provide a satisfactory response to all of them. More noticeably, none have attempted to control the overall posture of a robotic arm using the movement of the user's arm without mapping directly the user's arm joints (or strategic points) to the robotic arm joints. A robotic arm posture is represented by the position of every joint and the links connecting them in the Cartesian space. The posture can also be defined by the set of angles ϑi (joining link i-1 to i) in the joint space. Most solutions found in the literature limit their control to the end-effector (EE), leaving the choice of the other joints value to an almost uncontrolled and unpredictable process among the possible arm joints solutions. In the context of obstacle avoidance through teleoperation, this is unacceptable.

Considering the set of joints of a robotic arm as a whole, this thesis explores and proposes different approaches for controlling simultaneously every joint of the robot by influencing (controlling) the shape of the robotic arm through different means using the movement of the user's arm. To this end, we use a gesture-based system with visual recognition with the aim of addressing to some degree all of the aforementioned challenges.

1.3 Posture exploration challenges

To achieve our goal of controlling the posture of a remote robotic arm using the static and/or dynamic hand gestures system many difficulties are expected to be encountered during the course of the thesis. In the context of this work, a gesture is defined as a particular sequence of poses (i.e. positions) of the following subset of the human body: arms, hands and fingers. A sequence of the same poses through time is defined as a static gesture while a sequence of poses incrementally changing through time is defined as a dynamic gesture. For example, a user executing a particular sign language number (1, 2, 3, etc.) qualifies as a static gesture while a user drawing an imaginary circle with one of his hand index fingers qualifies as a dynamic gesture. The first challenge is the possibility of multiple distinct operational spaces (or closed sets of possible posture) for a particular target position. That situation often occurs when the robotic arm configuration includes limited joint revolution angles or when obstacles (forbidden operational space) are present inside the reach radius of the robotic arm. All postures within a set can reach another posture that is part of the same set by transitioning through postures that are also part of the same set. Equivalently, any initial posture located outside the set (thus part of another operational space) can move in any direction without ever reaching inside the other set. A simple example of a two-link planar arm reaching a particular end-effector (EE) target with two sets of postures is illustrated in Figure 1(a). In Figure 1(b), boundary postures from two closed sets of multiple postures (operational spaces #1 and 2) of a constrained three-link planar arm are presented. The only way to reach a posture from one set to the other is by either moving the end-effector or the base joint. It is difficult to discern whether more than one closed set of possible postures (i.e. distinct operation spaces) exists for an arm configuration (link length between joints, joint angles limit) and a particular target position. Since the thesis involves controlling the postures of a remote arm, being able to find possible postures from all closed sets is essential if we wish the algorithm to have a better chance of finding the user's desired posture. This process is what we refer to as "posture exploration".



(a) (b)

Figure 1 (a) 2 DOF planar arm exhibiting the only 2 postures able to reach the target (shown in red) with the end-effector (EE); (b) 3 DOF planar arm with joint angle constraints exhibiting two posture samples from 2 distinct closed sets of postures able to reach the target (shown in red) with the EE.

A second challenge occurs when searching for the possible postures matching a desired posture for a particular target position. Evaluating every single possible posture at the output of an Inverse Kinematic (IK) function is too expensive in computer cost. Using an IK algorithm like the Forward and Backward Reaching Inverse Kinematics (FABRIK) [39] combined with a simple greedy algorithm offers the possibility of limiting the exploration to the direction of the closest matching posture iteratively. While being faster, this approach needs to be carefully employed as to avoid a local optimum solution masking the overall optimum solution. As depicted in Figure 2, a local optimum solution can be found (black dash arm) in the neighborhood of the initial posture, while the further away optimum solution stays undiscovered (solid line arm).

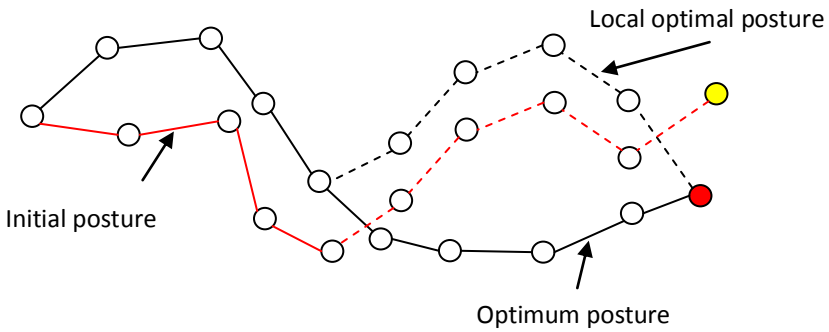


Figure 2 9 DOF planar arm showing a posture solution locally close to an initial posture and the optimal posture solution.

A third challenge in exploring robotic arm postures using inverse kinematics is that a certain coverage in the operational posture space (joint space) from a group of input initial postures does not necessarily generate the same equivalent coverage with the resulting group of possible postures. To clarify, let represent the set of all postures inside the operational posture space as the domain X, the IK of a particular target position t as the function IK(t) and the set of all outputed posture Y as the Codomain of IK(t), as illustrated in Figure 3. To be able to find the best matching possible posture corresponding to the user's desired posture, we need to find the set of initial postures (x) that outputs an image that covers most of Y, thus increasing the probability of finding a posture close to the desired output posture solution. With the aim of improving the IK output coverage (i.e. the image of IK(t)), different strategies in the selection of the set of initial posture input (x) need to be investigated along with a comparison between different IK algorithms to efficiently cover Y. Section 5.2 is dedicated to this investigation.

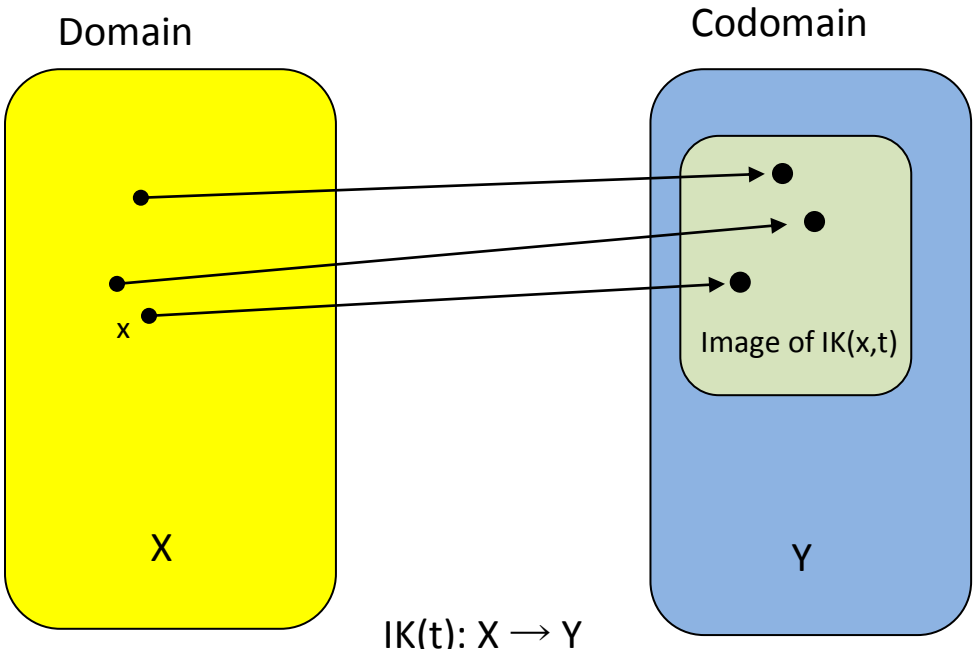


Figure 3 Representation of the operation space of all postures (domain: X), the IK for a particular target position t (IK(t)), the set of all possible output postures (Codomain: Y), the set of input posture x and the image of IK(x, t).

1.4 Objectives

The goal of this thesis is to investigate, design and implement different approaches to the control of a robotic arm that are both natural and complete enough to teleoperate a remote robotic arm in an environment with static and dynamic constraints. Consequently, different posture control solutions are explored by interpreting the user's hand to control the end-effector and overall posture of the robot's joints that overcomes the common limitations stated in Section 1.2.

In the context of this thesis, the term "posture control" refers to the control of the robotic arm as a whole. This means that every joint is directly under the influence of the user who can choose a particular set of joint positions as a solution over another. This represents the highest

level of control over a robotic arm while solution limited to end-effector manipulation would represent one of the lowest.

This level of control needs to be obtained without compromising on the user scalability and the capacity of the solution to adapt to any robotic arm configuration (2D and 3D robotic arm, joint types, number of joints). From our literature research, these two problems represent the downside of all proposed systems using this level of control. Take note that 1D robotic arm are not included in the scope of our work as their posture control is considered to be of no interest.

To interpret the user's hand movement, a gesture-based visual recognition system for static and/or dynamic hand gestures is used to give a natural control over the robotic arm posture that is intuitive without suffering from invasive equipment (sensors, wearable, etc), complex installation and costly system.

To move the robotic arm while avoiding moving objects in the real world, the user should ideally be able to control the robotic arm in real time at a rate the human eye can satisfactorily detect motion. Based on the television and movie industry standards, this rate could be considered to be 24 Hz, which is the minimal standard image-change rate currently supported. To this end, an inverse kinematic (IK) posture exploration approach technique is designed. Different methods of inverse kinematic calculation are investigated during this process to find the most adequate method for controlling the robotic arm posture.

A series of three different approaches will be proposed for this purpose. They will be implemented, tested and evaluated against each other in both simulated and real-world experiments under environments with both static and dynamic constraints.

From the above description, we can formulate the key contribution of this thesis as the following:

- The design and implementation of three different approaches to unilaterally teleoperate in real-time a robotic arm's entire posture using gesture-based vision recognition combined with inverse kinematic exploration approach. The proposed approaches allow a complete and natural control over the robotic arm that is users scalable, compatible with most

existing robotic arm configurations and suitable for obstacle avoidance in environments with static and dynamic constraints.

- Each solution involves the design and implementation of different robotic arm posture control systems adapted to a common hand gesture recognition system. We also propose different search strategies for each solution for mapping these two systems that optimize both the speed (for real-time application) and accuracy in reaching a desired robotic arm posture.
- Each solution is validated in simulation with static and dynamic environment constraints and compared in terms of speed efficiency and accuracy. A real-world pick and place demonstration of each solution is performed in a highly constrained static environment.

1.5 Applications

The range of applications targeted by the proposed posture control solutions is only limited to the range of telerobotic applications involving communication latency low enough to allow low-level control over the remote manipulator (direct control architecture). For applications with high latency, a system with a more supervised architecture, as explained in Section 1.1, is more suitable. Examples of the latter system can be found in remotely controlled robotic applications involving extraterrestrial communication, where waiting periods can reach several seconds (Moon), minutes (Mars) or hours (32 hours to reach the edge of our solar system) [2]. However, for most earth-confined teleoperation applications, a latency of no more than 500 ms is usually required [2]. Hence, based on the objectives listed in Section 1.4, our proposed approaches can be used in any of following applications:

1. Manipulation of hazardous materials (radioactive, chemical or biological);
2. Operation in hazardous environments (nuclear facility, volcanic activity analysis, etc.)
3. Precise medical operations;

4. Disarmament of explosive devices (mines, bombs, etc) for either military or police operations;
5. Assistance in repair or maintenance operations for International Space Station astronauts.

Applied to the world of entertainment, our solutions could also be involved in video games supporting the Kinect or similar devices that require a solution for character posture control. However, latency of less than 200 ms would be required to avoid annoying players [2]. Since our proposed solution aims to have less than 40 ms of latency, this application can be easily targeted as well.

1.6 Thesis organization

This chapter presented the problems and limitations of current solutions found in the literature involving posture control of a slave robotic arm, the objectives of this thesis and the possible applications of our proposed solutions.

In Chapter 2, the previous work covering different master-slave and gesture-based telerobotic approaches is presented. This is followed by an investigation of the different inverse kinematic calculation methods applied in telerobotics and obstacle avoidance systems.

In Chapter 3, the hand detection and gesture recognition system developed in our previous work [3] is presented. This is the base layer over which the proposed approaches presented in this thesis are built.

In Chapter 4, three different approaches are proposed to control the posture of a robotic arm, given any configuration, in a 2D and 3D environment. For simplicity and given the available robotic arm configuration for validating the proposition experimentally, the case of the anthropomorphic arm with 3 DOF and 4 DOF are studied in more detail. Each solution is part of a gesture-based robotic arm control system application.

In Chapter 5, a description of the test environments (robotic arm and simulation software) is presented along with multiple experiments aiming at improving the IK output coverage for both

wide and local/directional posture search. Next, a description of the simulated static and dynamic obstacle environment is disclosed followed by a description of the real-world static obstacle test.

In Chapter 6, the results coming from the simulation and real-world experiments performed in Chapter 5 are presented, analyzed and compared between the different proposed posture control methods.

In Chapter 7, we present the conclusions to the proposed approaches and suggest different related challenges that would be worth investigating further in future work.

Chapter 2 Related work

In this section, we present the scope of our literature research with a focus on the control of robotic arms and the level of control the user has over the entire posture of the slave for both master-slave and gesture-based approaches. In addition to these two types of telerobotic control approaches, this chapter also covers related work on different relevant topics of interest: inverse kinematics, obstacle avoidance and hand gesture recognition.

2.1 Master-slave telerobotic systems

As specified earlier, there are many types of master control approaches that can be used to manipulate a slave robot. In the work of Heunis, Scheffer and Schreve [4], a sophisticated joystick design was developed to directly control a 7 DOF slave system for a robotic medical surgery application. While it was revealed to be intuitive and accurate based on a pick and drop test, no dynamic environment tests were conducted nor associated results presented. Moreover, this solution is based on a complex master system where the multiple encoders are prone to error and in need of recurrent calibration. Much of the same limitations also affect an arcade-game-like joystick presented in [5], where an additional visual feedback using markers to stabilize the slave robot motion was added to enable real-time use.

What could be considered an enhanced form of joystick in a master system is a joystick-like device with haptic capabilities, where a sense of touch is returned to the user under the form of a force, vibration, temperature, etc. In this approach, the work of [6] uses a replica of the slave robot to control the end-effector. This type of system, referred to as homogeneous or isomorphic, consists of a master and slave robot with kinematically similar mechanisms. This is opposed to master-slave systems with kinematically dissimilar mechanisms, which are also referred to as heterogeneous or isomeric systems. In [7][8][9][10], the authors use an heterogeneous master-slave system where the master manipulator is designed specifically to be generic and manipulate the end-effector position from inverse kinematics calculation of various types of slave configurations while receiving feedback forces. In [11], a passive haptic

data glove is used to control the end-effector position. The term "passive" applies to haptic devices that do not use motors to simulate obstacles' feedback from the slave robot. In the data glove of [11], the energy feed to the accelerometer sensor monitoring the user hand and finger movement is dissipated based on the constraints information coming back from the slave system.

In [12], a digital glove, with no haptic capability, is used to position the end-effector. Instead of providing direction commands exclusively based on finger flexures and hand orientations, they also monitor the position of the user's hand relative to their shoulder using an additional sensor. As any other wearable, this approach is invasive and requires regular recalibration.

In [13], a wearable motion capture system covering the upper arm, forearm and hand is able to control the posture of a heterogeneous slave robotic arm. To map the joints from the master to the slave joint space, joints on the slave robot are grouped into "shoulder," "elbow" and "wrist." This system, however, is complex, invasive, and prone to error due to the use of multiple sensors and on the possible displacement of the wearable during use. This approach is also limited to certain types of robotic arms.

Similarly, another approach consists of wearing an exoskeleton structure [14][15][16][17][18][19] around the user's arm to control the robotic arm joints. Such a system is quite complex and usually involves even more sensors than wearable solutions. From our research, wearables often rely solely on the use of accelerometers for positioning, while exoskeletons also include potentiometers and encoders at the mechanical joints. Hence, such a solution requires calibration with a higher risk of error due to defective sensors or shift of the exoskeleton on the user's arm.

In [20], a marker system is mounted on the master (human) to directly map their joint movement to the joints of the robotic arm. Mimicking the exact movement of the human arm as proposed in [20] might be optimal in terms of intuitive control when the robot has a similar configuration (DOF, type of joints, etc.) and constraints (joints angle limits) compared to the human arm, but can become cumbersome or plainly impossible to apply otherwise.

2.2 Gesture-based telerobotic systems

The gesture based systems presented in this section mostly use vision recognition. However, a special case is presented later where a wearable is also involved.

The gesture-based solutions found during our research usually include a vision recognition system as in [21]-[35] to gather information about the user's gestures. For this type of systems, different methods for mapping the user's Cartesian space gestures to the joints space of the robotic arm postures are proposed:

1. Direct mapping
2. Cartesian impedance
3. Forward Kinematics
4. Inverse Kinematics
5. Reserved static or dynamic gestures
6. Gesture interpretation based on their location in the camera field of view
7. A combination of the above methods

In the work of [25], the positions of the user's hands are measured relative to the shoulder joint. The X/Y components of each hand position are then directly mapped in terms of rotation angle to the robotic arm joints. This method not only is counter intuitive but also requires regular calibration. In [26], the different joint angles of the user arm are directly mapped to the joints of the robotic arm. Such a method has the downside of becoming counter-intuitive or impossible to apply with other kinds of robotic arm configuration. In [31], the rotation matrix is used to transform each joint provided by the Kinect skeleton algorithm into the joints space of the NAO robot. While suffering from some obfuscation issues, this solution is also restricted to robot arms of the same configuration as the human.

In [32], a Cartesian impedance control with the user detected arm joint as command point is used to guide the robotic arm motion. This method has the inconvenience of often overshooting when changes in direction occur, which makes it undesirable for environments with obstacles.

In [23][30], the position and orientation of the end-effector is controlled by mapping each robotic joint to the user arm bone angle as recuperated by the Kinect SDK skeleton algorithm using forward kinematics calculations. Obviously, as with [31], a master-slave system relatively close to being homogeneous needs to exist between the user arm and the robotic arm; hence these approaches are limited in terms of robotic arm configuration compatibility. In [29][33], the position of the palm center of the user hand returned by an analysis of the data acquired by the Kinect is used to position the end-effector. The inverse kinematics is then solved to find the robotic arm's joint angles. The author of [33] also added a fingertip detection algorithm (using convex hull) that controls the gripper state using the angle between two fingers and the palm center. In [35], hand segmentation is used to detect the user's index fingertip position, which, combined with the thumb tip and the valley between the thumb and index, can control the end-effector position and orientation. In [28], different intervals of the user elbow angle found using the Kinect are used to send relatively high-level commands, such as move EE to user's hand position, move EE along right user's arm direction, close and open gripper. The proposed solution is unfortunately constrained to a particular pick and place activity. The major drawback coming from each of the four approaches [28][29][33][35] mentioned above is the lack of control over the posture when resolving the robotic arm kinematics with the inverse kinematics calculation from the end-effector information. In [27], the inverse kinematics algorithm is used to control the motion (i.e. position, velocity and acceleration) of all three joints of the robotic arm from the user's arm joint information found using the Kinect. While controlling the robotic arm postures, this approach is limited in arm configuration compatibility, as [23][30][31].

A set of reserved static postures, proposed in [21][22] and a dynamic gesture, proposed in [24] are obtained through the Kinect skeleton algorithm and used to control the velocity vectors of the end-effector in the Cartesian space. In [21], selected user skeleton postures were also added to control the end-effector orientation velocity. Again, no posture control is offered using these approaches when resolving the inverse kinematics from the EE information. However, a second method was proposed in [21] to control each joint’s velocity using reserved static postures from the user skeleton tracked by the Kinect. While offering a posture control for the slave robotic arm, this technique was revealed to be quite unpractical since each robotic

joint needs to be controlled one at a time with a different pose of the human body. Hence, the operation is quite slow, unnatural and requires exhausting physical effort from the user.

In the work found in [34], separated areas of the Kinect field of view are dedicated to controlling each robotic arm joint motion direction. The approach requires a Human Machine Interface (HMI) display showing the different window areas for each joint direction and gripper open/close state, overlaying the user's video image. The user can only control one servo at a time which makes the overall experience very similar to the joystick experience, i.e. quite tedious and unnatural to the user.

In a different gesture-based approach, the author of [36] proposed a system where a wearable camera sensor is mounted around the user's wrist. The recognized finger combinations ({finger 1, finger 2}, {finger 3}, etc.) are mapped to the different velocity vectors of the end-effector, as well as a close/open gripper control scheme. This solution however requires calibration and does not offer control over the posture of the robotic arm.

2.3 Inverse kinematic methods applied to telerobotics

The inverse kinematics is a calculation made from a set of initial joint angles and desired robotic arm end-effector positions (also called target positions) to obtain a posture solution represented by a set of specific joint angles enabling the end-effector to reach the target position. The end solution is often dependant of the initial posture since several solutions are usually possible for a unique target position. Various Inverse Kinematics (IK) methods exist in general robotics from which several were found in the context of telerobotics research in recent years: Jacobian Transpose, the pseudo-inverse Jacobian, Forward and Backward Reaching Inverse Kinematics (FABRIK), Neural Networks, etc.

In the work of [37], the Jacobian Transpose method is used to calculate the inverse kinematics from the robotic arm end-effector information. This method has the advantage of being accurate without converging toward singularities, but suffers from high computational cost. In an attempt to solve this issue, the author proposed a speculation strategy parallelized on a

multithreaded architecture, but more hardware resources (GPUs) are required to significantly decrease the processing time.

In [38], the Joint Space Decomposition (JSD) and a generalization of the pseudo-inverse-based method, with optimally scaled basis vector of the Jacobian null space matrix (NM), are investigated and compared in a robotic system with minimum time optimization for a particular end-effector trajectory (basically, the authors make the EE follow a specific trajectory as fast as they can while dodging fixed obstacles). Both JSD and NM method tests revealed numerous cases where the solution did not converge. However, JSD had the additional problem of encountering singularities. The convergence in terms of number of iterations was the same between both methods, but the computational cost was higher for NM due to a more complex algorithm.

In [39][40], Artistido proposed a Forward And Backward Reaching Inverse Kinematics (FABRIK) method for calculating the inverse kinematics. The new method was compared against the following traditional methods: Cyclic Coordinate Descent (CCD), Jacobian Transpose, Jacobian Damped Least Squares (DSL), Jacobian Damped Least Squares with Singular Value Decomposition (SVD-DLS), Follow-the-Leader (FTL) and the Triangulation algorithm. Experimental results showed that FABRIK required fewer iterations and lower computational cost per iteration to reach a target for both constrained and unconstrained kinematic chains. As with the Jacobian Transpose and the DLS method, FABRIK does not suffer from singularity problems since no matrix inverse needs to be calculated. Additionally, unlike most of the other methods under study, FABRIK is able to systematically generate more realistic and smooth poses. This is major drawback with the CCD and Triangulation approaches. The FABRIK algorithm starts by geometrically positioning the end-effector (joint **i**) at the target location, then moving joint **i**-1 at a distance equal to the length of the link between joint **i** and **i**-1 from the target on a vector crossing the new position of joint **i** and the old position of joint **i**-1. It then continues down to the base joint of the robotic arm. When reaching the base joint, this joint will be positioned at a new location following the algorithm logic. However, since the base joint is fixed and cannot move, it will be geometrically repositioned at its old location. The

algorithm then corrects each joint position accordingly following the same logic it used going down the arm. This process iterates up and down the arm until the positions of both the end-effector and the base reach their targeted position.

In [41], the author compares the FABRIK method against a Neural Network approach, the Jacobian pseudo inverse and Jacobian Transpose. The Neural Network algorithm is a learning method composed of interconnected nodes working in parallel with weighted signal processed by activation function. The method learns from training examples. The study revealed that the Neural Network method greatly outperformed all the other methods in terms of speed, but performed the worst for accuracy. FABRIK and pseudo-inverse Jacobian were close to each other for computational cost when no errors were tolerated, but pseudo-inverse Jacobian suffers from the singularity problem.

An inverse kinematics software library enabling the implementation of the FABRIK algorithm is proposed in [42]. It provides 2D and 3D applications of the IK for various joint types and a visual display of the solutions. The library is written in Java and was created to receive hand and finger positions from a Leap Motion sensor.

In [43], the FABRIK method is used to simulate the control of an end-effector position of a 3 DOF robotic arm manipulator. A PID controller was added to the system to decrease the time to reach the target and the position error. However, since no posture control was proposed, this teleoperation would not be adequate in a constrained environment with obstacles. In [20], described in Section 2.4, a posture control solution was proposed using the FABRIK algorithm to solve the inverse kinematics.

2.4 Obstacle avoidance systems

One of the main problems in remotely controlling a robotic arm is the avoidance of obstacles, either static (fixed) or dynamic (moving). Obstacle avoidance algorithms are usually included in system architectures closer to supervisory control. They are used by the slave system while performing a primary task to automatically prevent collisions based on sensors without any involvement from the user. The solutions proposed in [44][45] target pre-known static

obstacles in the proximity of a fixed robotic arm, that impose a space of possible movements for the robotic arm. In the work of Lin and Nguyen [20], a rapidly exploring tree method is used to calculate a trajectory path free of obstacles to reach a designated target. This approach encompasses time-consuming calculations and thus can be difficult to apply in changing environments. In [46], a multi-tasking approach involves human motion imitation for obstacle avoidance moving the end-effector along a desired trajectory. Another approach is proposed in [47] where human motion imitation is replaced by human gesture imitation with better results. Obstacle avoidance is particularly useful in applications where command delays are too long for the user to make the slave robotic arm react in real-time to the environmental constraints at the remote site. It is however not effective compared to the human capabilities to react to any environmental constraint. For that reason, teleoperation architectures closer to the direct control type are preferable to avoid obstacles for cases where time delays are not an issue.

2.5 Hand gesture recognition

In a gesture-based telerobotics system, a vision recognition component is needed to capture the user's different gestures in real-time. To help in reaching the thesis objective of interpreting the user's hand to control the end-effector and overall posture of the robot's revolute joints, this section focuses on investigating past research related to hand gesture recognition.

The principal components of a hand gesture recognition system are data acquisition, hand localization (e.g. segmentation and tracking), hand feature identification, and gesture recognition based on identified features. The classic solution for data acquisition is color cameras that have already been successfully employed for gesture recognition tasks [48]-[50]. These solutions are, however, sensitive to clutter, lighting conditions, and skin color. Video capture has the extra challenge related to the speed of movement. In terms of 3-D motion capture at the level of the fingers, possible solutions include optical marker systems, accelerometers, magnetic trackers, and data gloves. These require extensive calibration, limit the natural movement of the fingers, and are generally very expensive. Vision-based markerless motion capture became possible largely due to the introduction of the Kinect, a low-cost off-

the-shelf sensor that gathers positional information about an individual's motion. Data captured by this sensor, in RGB-D (color and depth information) form, is often used as a source for hand gesture detection and recognition. While a good overview of gesture recognition is available in [51] and a focus on depth images in [52], the following literature review briefly summarizes the solutions that make use of RGB-D data for the purpose of hand gesture recognition.

Different approaches can be employed for hand localization in acquired data, according to their nature. For depth data, the classical solution is depth thresholding, either empirical or automated. Empirical solutions choose the limits of the most probable search spaces by trial and error and concentrate the computation effort on the hand localization within it. Among automated solutions are the detection of the closest points to the camera [53]-[57] based on the assumption that the hand is the closest object in the scene, and the use of other reference elements in the scene, such as head position [58]-[60] or facial color information [50], to identify the most possible location of the hand. An exhaustive solution is to scan the entire visibility space of the Kinect using overlapping depth intervals until the hands are found [61]. This can be, however, a very lengthy process. In terms of color images, possible solutions for hand segmentation include skin color detection [48], [50], [59], [62]-[64] and Haar-like features [49]. Finally, certain approaches in the literature benefit from both depth and color information [54], [62]-[69]. Some systems impose the intervention of the user to calibrate the system prior to its use; for example, in [67], the location of the hand is recuperated from a wave gesture executed by the user. Others impose constraints, such as the fact that the user wears a colored glove [69] or a black strap on the wrist of the gesturing hand [54], [65], [70] in order to simplify the hand localization procedure and its segmentation. These solutions impede on the invisibility of the interface to the user. Finally, Ryan [61] favors the middle of the screen as an area of higher likelihood for hand detection, therefore avoiding the search along the image sides in order to improve the response time. Such a system forces the user to adapt their behavior to the interface.

After hand localization, the next step is the selection of appropriate features that allow for the recognition of gestures. Among the solutions encountered in the literature for depth data are hand contour, hand palm center [53], [61], finger properties (e.g. count and direction) [71], hand trajectory [58], [72], and gradient kernel descriptors [68], while for color data, popular choices are Haarlet features [67], Gabor features [73], [74], Scale Invariant Feature Transform (SIFT) [68], Speeded Up Robust Features (SURF) [62], Hu moment invariants [64], Histogram of Oriented Gradients (HOG) [75], [76] descriptors, and shape distance metrics, such as the finger–earth mover's distance [54], [65], [66]. Otiniano-Rodríguez and Cámara-Chávez [68], Abid *et al.* [75], and Dardas and Georganas [77] combine features using a bag-of-visual-words approach prior to recognition.

For the recognition of gestures from the selected features, possible techniques proposed in the literature are neural networks [58], [67], Support Vector Machines (SVMs) [62], [75], [77], nearest neighbor classifiers [55], [57], random forests [73], fuzzy approaches [48], and hidden Markov models [62], [74], [76]. Alternatively, a simple method that takes into consideration the position of middle and end points of a gesture trajectory with respect to the initial points is proposed in [64]. Most of the existing solutions for gesture recognition are designed for use either on static [55], [57], [67], [68], [71], [73], [78], [79] or on dynamic gestures [58], [75], [80], [81]. Overall, there are very few solutions that simultaneously work on both static and dynamic gestures [59], [72], [74]. Moreover, most of the current solutions are able to handle a single hand [48], [54], [62], [65], [68], [73], [76], [78], [79] or two hands [61], [64], [71]. To the best of our knowledge, there are no solutions that can track and recognize gestures from multiple hands, as the one proposed in our previous paper [3]. The results, in most cases in form of average recognition rates, are presented on data sets of selected static and/or dynamic gestures varying from 6 to 42 gestures: 6 in [79], 10 in [55], 12 in [57], 14 in [64] and [71], 24 (or 26) [gestures of the American sign language (ASL) alphabet; two gestures in this alphabet are dynamic and some authors work on static gestures only] in [56] and [68], and 42 gestures in [74]. Most results are reported for a single hand [54], [56], [74], [81] and a few for two hands [64], [71].

The objective of our previous work on the topic in [3] was to develop an improved, low-complexity, and real-time solution for the recognition of both static and dynamic gestures executed by one or multiple hands from depth data returned by a Kinect sensor. The gain in performance and speed in this work comes from multiple improvements in the various stages of the solution, including a faster algorithm to search the first pixel of the hand contour; the recognition of gestures independent of their position in the field of view of the Kinect; the use and adaptation of dynamic time warping (DTW) not only as a validation tool [61] but also as a selector for candidate gestures; the normalization of disparity values to improve the recognition rate in the case of multiple hand detection; the use of constraints on the contour to ensure it is closed; and the use of contour information for recognition only in cases when no fingertips are identified, in order to improve the computation time. Moreover, the proposed approach offers an interface that allows the user to train the system with their desired gestures and rapidly test its performance. According to a recent review on gesture recognition [52], this is one missing aspect of current solutions: the capacity to learn a gesture set from the user. The interface is invisible to the user in the sense that it does not require any constraints or calibration procedure. Due to the fact that our system is based only on depth information, cluttered backgrounds, lighting conditions, clothing, or skin color have little impact on the reported performance**.**

This system will be used to capture the user's gesture, hands and fingertips to communicate the desired position of the end-effector and overall posture of the robotic hand. Although no pre-recorded dynamic gesture recognition is needed in the present telerobotic application, the tracing method detailed in Section 4.4 is based on the dynamic gesture recognition sub-system that is part of our work in [3].

2.6 Related work summary

In this chapter, we investigated relevant solutions for the posture control of a robotic arm using master-slave systems. This kind of approach included joysticks, exoskeletons, haptic devices, data gloves and wearables. These systems revealed to be mostly complex, costly, difficult to

scale for different users and robotic arm types, invasive and in need of regular calibration. The use of multiple sensors made those approaches additionally more prone to error. For these reasons, we decided to discard such a system from our solution design aimed at solving the list of problems presented in Section 1.2.

Relying mostly on RGB-D camera sensors, the gesture-based systems found in the literature were simpler and less complex in terms of hardware components, compared to a full robotic master-slave system. After analysis of the solutions, we noticed the following recurring problems: (1) slow control, (2) exhausting control effort, and (3) difficulty to scale. To avoid having slow controls, our solution must be capable of controlling the different motors on the robot as a group. To avoid rapidly exhausting the user while using the controls, we need to use small and simple gestures, sustainable for long periods of time. To scale our solution to most robotic arm configurations, we propose a novel approach that performs an inverse kinematic exploration to find the optimum posture at each time frame, which is explained in Chapter 4.

During our investigation of the different IKs applied to telerobotics, we found two promising calculation methods for real-time use: FABRIK and Pseudo-inverse Jacobian. They are both similarly fast and accurate when converging towards a solution. The FABRIK method has the added benefit however of not suffering from the singularity problem as opposed to the Pseudo-inverse method. Since only the best evaluated posture gets selected during exploration, that issue might not affect the accuracy of the resulting posture solution but rather only the speed of the exploration process. Hence, a comparative experiment is performed in Section 5.2 to identify the optimal IK method for the exploration algorithm based on two criteria: speed and accuracy.

We briefly studied the different approaches used in obstacle avoidance systems for static and dynamic obstacle environments. Such systems encompass arrays of sensors to pre-analyze the environment, enabling the robot to make the proper adjustments for avoiding obstacles while executing tasks commanded by a user. The complexity and computing cost of such a system prohibit its inclusion in the proposed solution.

We also analyzed the current state of the related work in the field of hand gesture recognition that led us to our proposed solution for a static and dynamic natural gesture recognition system in [3]. Presented as an improved solution for both static and dynamic gesture recognition capability, the system gives unmatched capabilities compared to most vision recognition systems used for gesture-based telerobotics found in the literature and presented in Section 2.2. This gives us the opportunity to leverage its capabilities in the present thesis. Indeed, the system plays an essential role by providing a unidirectional interface between human user and machine through human gesture interpretation. More details on the system are presented in Chapter 3. The present work is built on top of this system as an application that leverages both its precision and real-time capability to propose a novel approach in controlling the end-effector and posture of a robotic arm.

Chapter 3 Hand detection and gesture recognition

As stated in Chapter 1, the proposed posture control approaches for robotic posture control takes command from the user's hand movement. This feature is supported by a vision-based recognition system taken from contributions done in our previous work [3]. In this work, the gesture recognition system is adapted to control robotic arms and to improve its performance. We are describing the details of the system in the current chapter to better clarify the overall solutions and the interactions between subsystems. With the vision-based recognition system, the user can intuitively and naturally control the posture of a remote robotic arm by performing simple hand human gestures in front of a Kinect sensor. The posture control system is an application that is built on top of the vision-based recognition system where real-time depth data collected by a Kinect sensor is used to gather information about the hands, and fingertips and detect any kind of pre-recorded static and dynamic human hand gestures.

The framework for hand detection and gesture recognition is illustrated in Figure 4. Raw data coming from the Kinect is used to recuperate depth information on all the pixels of an image. An algorithm is then used to identify each point of a closed contour identified within a given depth interval.

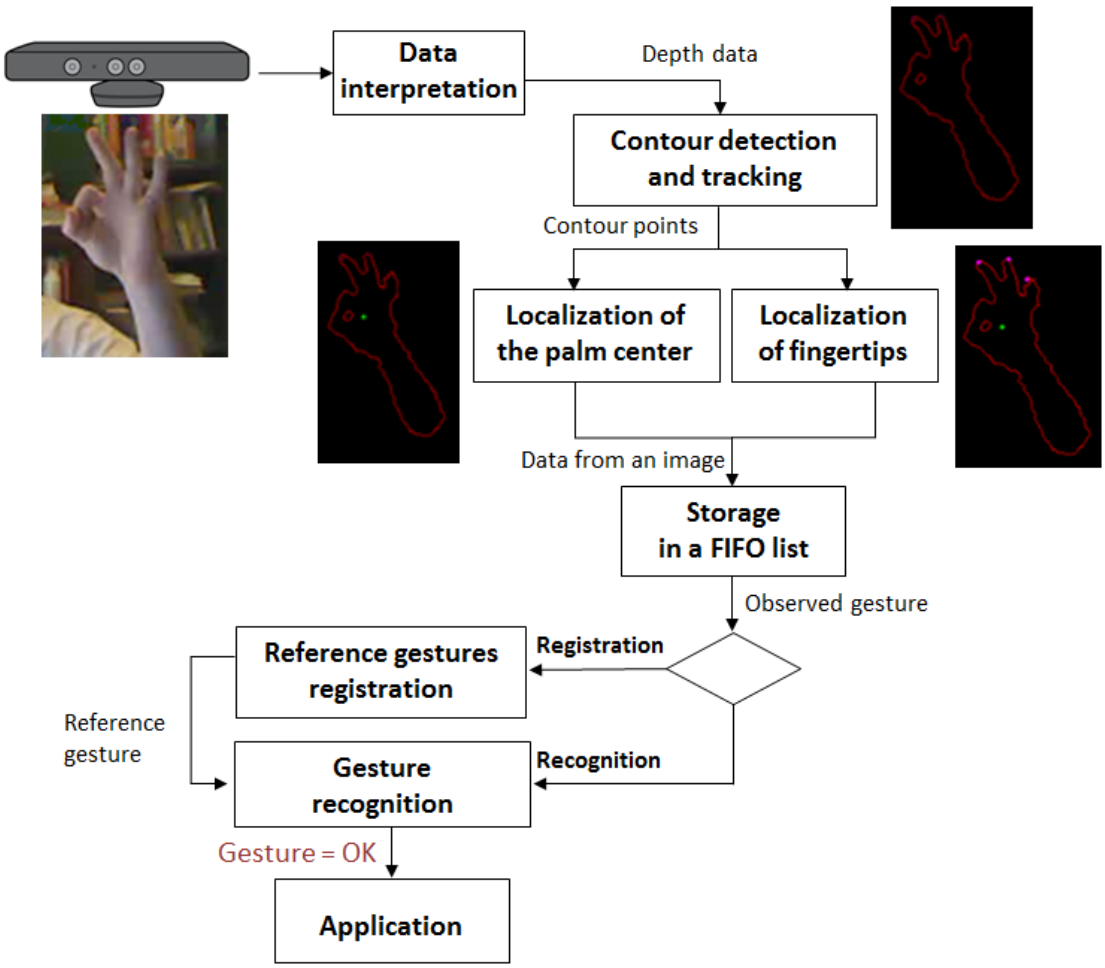


Figure 4 Framework for hand detection and gesture recognition.

Starting from the recuperated contour points, the center of the palm is identified as the center of the largest circle circumscribed in the contour. The fingertips are localized by employing the k-curvature algorithm, which is based on the change in the slope angle of the tangent line at selected points over the contour. During the execution, the hand information (contour, palm center and fingertips) is stored in a fixed-size FIFO list. This list contains data on reference gestures that can be recorded by a user, according to the specific needs of the application. The same list is also used by the dynamic time warping algorithm in order to recognize previously recorded reference gestures.

3.1 Contour search and detection

From the 16-bit raw data collected by the Windows SDK 1.8 for Kinect, the last 13 bits represent the depth information. A 3-bit shift operation is performed to only retain this information. Each time a new frame becomes available from the Kinect, the depth information is recuperated and used to detect one or multiple hand contours. By detecting hands contour at each frame, still or moving hands can be followed.

Detection of the Interest Space – In order to guide the search for the hand contour, the proposed solution starts by identifying the minimal distance (closest) pixel to the Kinect, as illustrated in Figure 5, based on the assumption that one of the hands is the closest object to the camera in the scene. The depth of this pixel is considered to represent the minimal depth for inspection and a constant (its value corresponding to the interval of inspection, i.e. an empirically determined value of 20 cm in this work) is added in order to calculate the maximum depth within which a search is performed for the hands. The search for the hand(s) occurs only within this limited space.

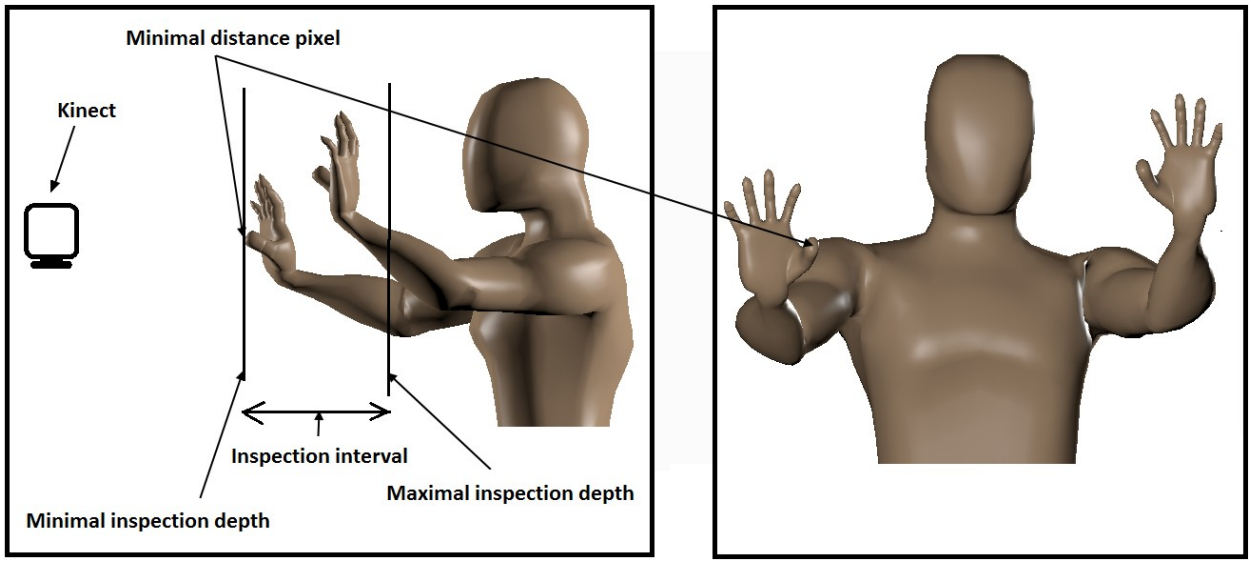


Figure 5 Inspection interval with respect to minimum distance pixel.

This approach is simple and does not impose significant limitations for the user, as humans naturally tend to keep their hands in front of their bodies when gesticulating. In order to not

have to be constrained by the closest pixel, a possible solution is to use other elements in the Kinect image as a reference, such as using the Kinect's skeleton tracking feature to identify hand position. However, due to the fact that this node (hand) is among the least stable nodes to track using the Kinect SDK [84], this solution was not retained in this work.

Search of the First Pixel of a Contour – We assume that all objects not belonging to a hand (e.g. body, other objects in the environment) are located behind the hands at a larger distance from the sensor than the maximal inspection depth. Hence at each frame capture by the Kinect, a search is performed within the inspection interval to identify the first pixel that has at least one neighboring pixel that is not included in the inspection interval. To optimize the search, [61] proposes to inspect every other 5 pixels (e.g. blocks of 5×5) and does not take into consideration the pixels around the edges of the image (20% of pixel lines). In our previous work [3], the search algorithm was improved as follows: A pixel is considered valid for inspection if it is present at a depth inside the inspection interval illustrated in Figure 5. Any other pixel is ignored in the analysis of the frame. The search takes place in blocks of 20×20 pixels. If a pixel is valid and does not possess any neighboring pixels that are invalid, the search continues pixel by pixel towards an invalid neighboring pixel until a contour pixel is found, as illustrated in Figure 6.

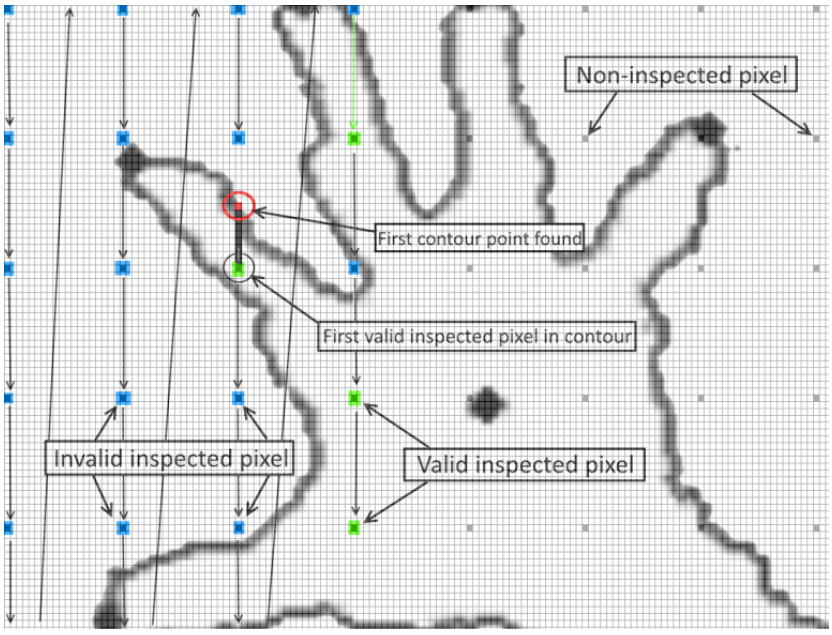


Figure 6 Proposed algorithm for the detection of the first point of the contour inspecting each block of 20×20 pixels.

We verify if this pixel is part of an already found contour and, if not, a new potential hand is found. If a pixel is valid and none of the neighboring pixels are invalid, the search continues with the next block. This algorithm results in a faster search. Because the Kinect is used with the resolution of 640x480 pixels, each image contains 307,200 pixels. In the worst case, the proposed algorithm verifies one pixel for every block of 20x20 pixel of the image, which amounts to a maximum of 768 verified pixels (307,200/400) to confirm an image that does not contain a hand. An improvement of 99.75% (or 100-(768/307,200)×100) is therefore achieved for an image of 640x480 pixel with respect to a full search. The proposed solution improves the search time by 15.25% with respect to the solution proposed by [61] and, unlike it, does not restrict the search to the central part of the image.

Contour Points Detection – Once the initial contour pixel is found, a directional search is performed to identify the full contour of the hand, as in [61]. In the context of this work, the considered directions are up-left, up-right, down-left and down-right. A search direction is favored if it has been used for the previous pixel as well. The algorithm (Algorithm 1 below) includes also a solution to backtrack if an unknown valid configuration is encountered. To

further improve this approach, we added a constraint, in our previous work [3], to only validate closed contours, as illustrated in the pseudo-code of the proposed algorithm shown below:

Algorithm 1 : Contour Point Detection

Input:

The first point, P_1, of a list of contour points belonging to a potential hand contour, **C**

Output:

List of contour points P_i belonging to an identified contour **C**

```
For each potential hand contour C
   From the first point P1 found, trace in order each point of contour and stop
   when the C.length > 700 or when the next point is already present in the
   contour list.
   //if the last point is near the first point
   if ((C.length > 100 and (|LastPoint.x – P1.x| <= 20 pixels and |LastPoint.y -
       P1.y| <= 20 pixels and |LastPoint.z- P1.z| <= 20 pixels))  then
       the Contour C is accepted
     Display the contour points in red
  else
     the Contour C is rejected
end
```

The smallest recognizable hand threshold (in number of contour points) and the maximum gap threshold between the first and last point of a hand contour (Algorithm 1) were adjusted during experimentation to get better results for this thesis compared to the threshold values used in the original work [3].

The detected contour using this procedure for the OK sign in Figure 7a is shown in Figure 7b. In order to identify more than one hand, once the closed contour points of the first hand are discovered, the algorithm continues the search for the first pixel of the next contour and repeats the process until it reaches the end of the image. The pseudo-code above is therefore repeated in a **for** loop for the detection of multiple hands. Furthermore, as described in the pseudo-code, small contours like the inner circle inside the OK sign are discarded from the search if they contain less than 100 points.

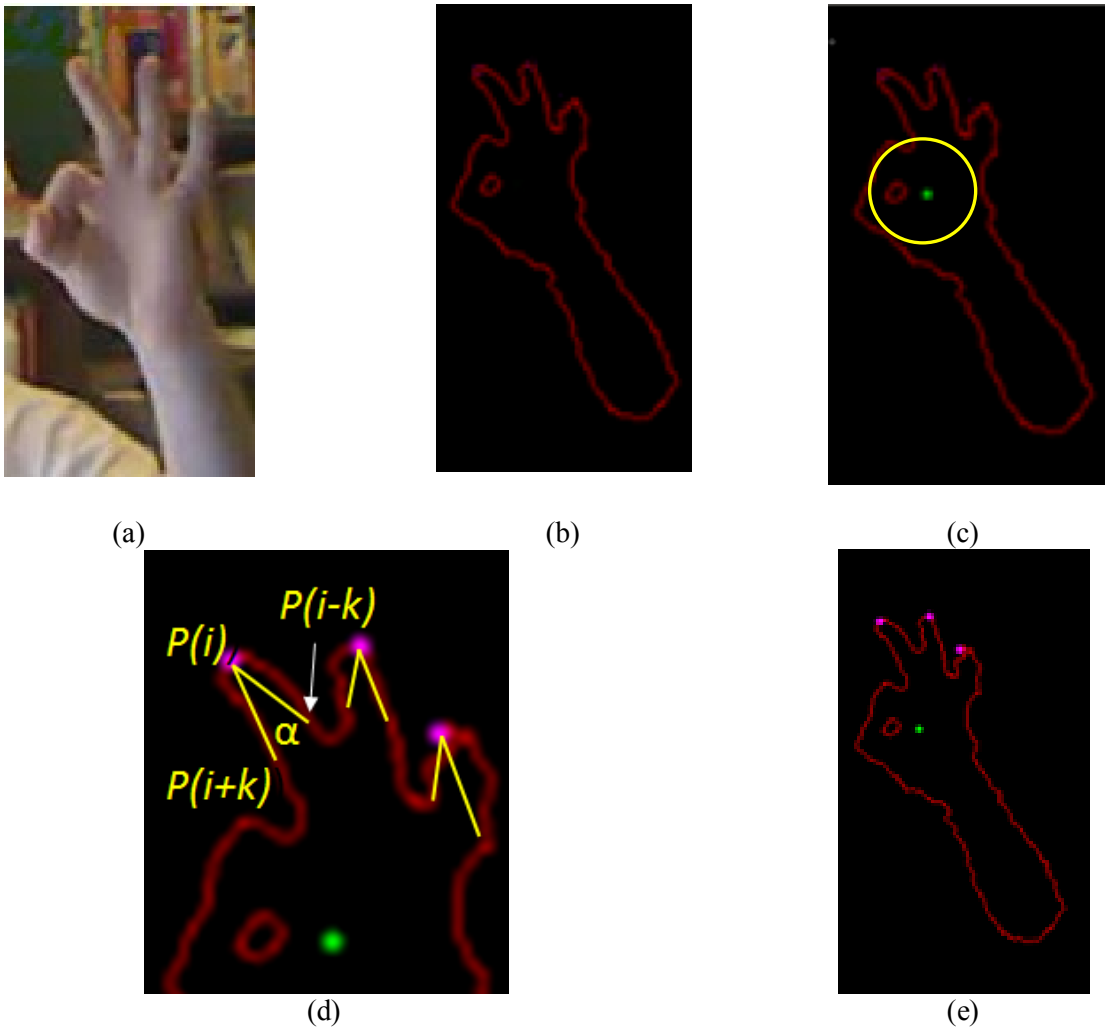


Figure 7 (a) Initial OK gesture, (b) detected contour, (c) largest circle contained in the palm, in yellow and identified palm center, in green, (d) angle calculation for fingertip direction computation, and (e) identified fingertips marked by magenta dots.

3.2 Hand palm center localization

Similar to the approach of [53], the hand palm center is calculated using the contour pixels as well as the interior pixels of the hand. To identify the bounding box of the hand, the minimum and maximum points on the *X* and *Y* axes are computed. The minimal distance (d_{min}) of interior pixels is then calculated for each block of 5×5 pixels with respect to each contour pixel. One contour pixel in 5 was proved experimentally to be sufficient for the final estimation of the hand center. The interior pixel that has the maximal value for d_{min} is considered to be the center of the palm.

In simple terms, the center of the palm is the center of the largest circle that can be circumscribed in the palm, as illustrated in Figure 7c for the case of the OK sign. The proposed approach is summarized in pseudo-code as follows:

Algorithm 2 : Palm center localization

Input:

C = List of contour points from the hand (obtained using Algorithm 1)

I = List of points inside the contour

Output:

Index of palm center coordinates in list **I**

1. For each element *i* of **I**:

   For each element *c* of **C**:

   Calculate the Euclidean distance d_i between *i* and *c*

   If $d_i > d_{i\,min}$ then

   $d_{i\,min} = d_i$

   If $d_{i\,min} < d_{i\,max}$, then

   Return to step 1

   end

   if($d_{i\,min} > d_{i\,max}$) then // Is the minimum distance obtained for i greater than

   // $d_{i\,max}$, the current maximum $d_{i\,min}$?

   $d_{i\,max} = d_{i\,min}$

   end

2. Return the center of palm = element *i* of **I** with the maximum Euclidean distance value obtained ($d_{i\,max}$).

3. Display a green circle at the coordinates of the palm center.

3.3 Localization of fingertips

The k-curvature algorithm is employed to identify the fingertips over the hand contour. The choice of this algorithm is justified by the fact that recent research showed that when compared with other solutions for static hand recognition, it provided the best results in terms of overall success rate, supported the highest range of hand rotations within which it is capable

to perform reliably and had the lowest complexity in terms of specific steps [78]. In simple terms, and as illustrated in Figure 7d, the algorithm uses two equal length measuring tapes, V_A and V_B, of 20 pixels each (e.g. step size, k=20 pixels) and measures at each point the angle α between them (e.g. between $P(i-k)->P(i)$ and $P(i)->P(i+k)$) over the contour of the hand. If the angle has a value between 25° and 55°, a fingertip is considered identified at that point. These angle limiting values are identified experimentally by trial and error. The pseudo-code of the algorithm is summarized in Algorithm 3 below and the detected fingertips for the OK sign are shown in magenta in Figure 7d-e.

Algorithm 3: Fingertip localization

Input:

C = List of hand contour points

$\mathbf{P_1}$ = first point of the contour **C**

c_A = Start point of vector V_A

c_B = End point of vector V_B

Output:

Series of points P_f representing the detected fingertips

$k = 20$; $c_A = \mathbf{P_1} - k$; $c_B = \mathbf{P_1} + k$
For each element c of **C**:

if c < k then

c_A = **C**.length - k + c

else

c_A = c – k

end

if c > **C**.length then

c_B = k - **C**.length + c

else

$c_B = c + k$

end

create vector $V_A(\mathbf{C}[c_A], \mathbf{C}[c])$

create vector $V_B(\mathbf{C}[c], \mathbf{C}[c_B])$

calculate $\theta(c) = \arccos\left|\frac{V_A \cdot V_B}{V_A.\text{length} * V_B.\text{length}}\right|$ (in radians)

transform θ (c) in degrees

If θ (c) <= 55 and θ (c) >= 25

// fingertip detected

display a magenta dot of coordinates c representing the fingertip

end

While this algorithm can sometimes lead to false fingertip detection, particularly when the user's fingers are thin and/or very close to each other, the situation was not encountered frequently enough to be addressed. A possible solution to this problem would be to use an extra constraint on the distance between the center of the palm and the fingertips as well as between the fingertips. Additionally, a solution could be added for the prediction of a missing finger by computing the mean of movement in previous consecutive images, as in [61]. All these would nevertheless have a negative impact on the hand tracking time, while not necessarily bringing significant detection improvements. Fingertip direction vector can be obtained by measuring the vector $\mathbf{V_C = V_A - V_B}$ and calculating the direction vector going from the middle of vector $\mathbf{V_C}$ to the fingertip position.

3.4 Gesture recognition using dynamic time warping

While static recognition can be accomplished by standard pattern recognition techniques (e.g. template matching), dynamic gesture recognition requires the incorporation of a time component in the process. Among the most often employed techniques are time-compressing templates, dynamic time warping, hidden Markov models and time-delayed neural-networks

[51]. In the context of this work, the dynamic time warping algorithm is used to compare the similarity between a reference and a captured gesture, whether the gesture is static or dynamic. The proposed approach allows the user to record a sequence of reference gestures that are saved in a FIFO list. In order to ensure real-time behavior of the system, this list is limited to 40 depth images. When the recording is finished, these images are saved in an xml file. Once a sequence of new images representing a gesture being executed is made available by the Kinect, the recognition module is activated to recognize it based on the features identified in sections 3.2 and 3.3.

The gesture recognition module executes two steps: (1) the search of candidate gestures based on the similarity between the observed gesture and each reference gesture, and (2) the validation of the similarity between each observed version and each identified candidate reference gesture. For the first step, [61] compares the last image of an observed gesture with the last image of each reference gesture in the list of reference gestures by calculating the Euclidean distance. If the reference gesture associated with the lowest distance is within a threshold, the gesture is considered to be a possible candidate. While simple and fast, this solution fails to distinguish between two dynamic gestures whose last image is identical, because it only compares the latter in order to recognize a gesture. To cope with this problem, we propose in [3] the use of DTW algorithm [85][86] not only for validation (in step (2)), but also as a means to identify possible candidates (in step (1)).

The DTW algorithm calculates the disparity (inverse of similarity) between two data series acquired at different times. A matrix containing the Euclidean distances at aligned points over the two sequences is used to calculate the minimal cost (shortest path) between the two series. The choice of direction for the shortest path is associated with certain rules. In particular, the movement is restricted to horizontal, vertical and diagonal directions. A weight is associated with each of these directions, and the shortest path has to be inferior to a threshold in order for the two sequences to be considered similar. In the context of this work, the two series represent the sequences of hand positions in an observed and a reference gesture, respectively. To allow for the recognition of gestures independent of their position on the

image, in step (1), we use the position of fingertips with respect to the center of the palm(s) and the relative position of the palm center(s) with respect to the palm center of the first discovered contour, as illustrated in Figure 8a. The hand disparity value is calculated as a sum of Euclidean distances between each fingertip and the palm center. For example, in order to select a candidate gesture, we compute the disparity between the 40th frame of an observed gesture (Figure 8a) and of a stored gesture (Figure 8b-c), using the following calculation:

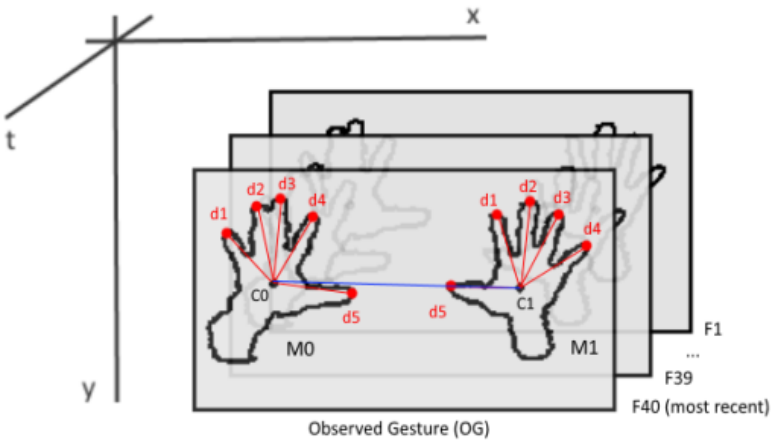
$$\Delta SO_{40} = \frac{\sum_{j=0}^{M-1} \frac{\sum_{i=1}^{N} \Delta d_i}{N} \times w_f + \Delta C_j \times w_p}{M} \quad \textbf{(1)}$$

where ΔSO_{40} is the stored and observed gesture disparity at frame 40, *M* is the number of hands detected in the image, *N* the number of fingers detected in the image, Δd_i is the difference in the distance from the palm center to detected fingertip *i* in the observed and the stored gesture frames, ΔC_j is the difference between the palm centers *j* in the stored and observed gesture frames, and w_f and w_p are the weights for the finger and palm center disparities, respectively (values are listed in Table I). For frames with only one hand, the palm center disparity is always 0 since the position is relative to itself. For the situation illustrated in Figure 8b, eq. (1) becomes

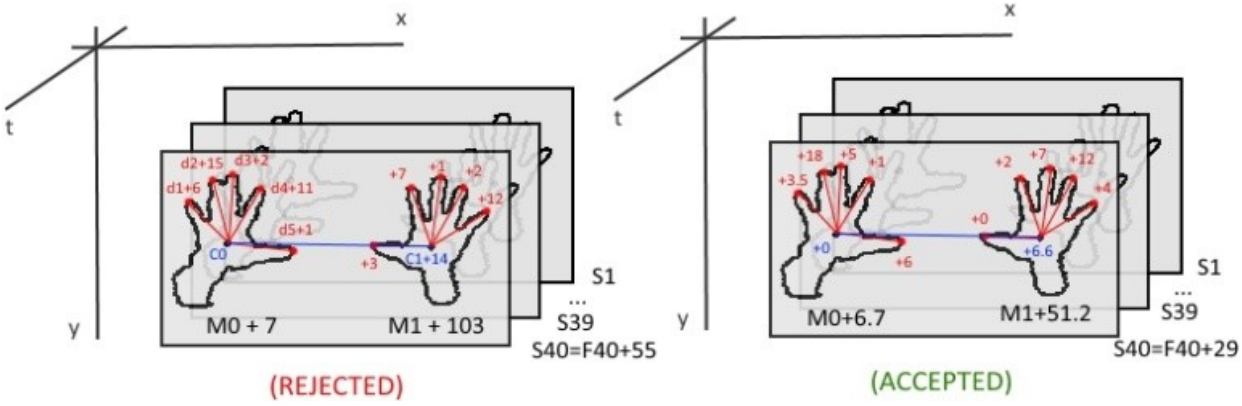
$$\Delta SO_{40} = \frac{\left(\left(\frac{(6+15+2+11+1)}{5} \times 1\right) + (0 \times 7)\right) + \left(\frac{(7+1+2+12+3)}{5} \times 1\right) + 14 \times 7}{2} = 55 \quad \textbf{(2)}$$

For a limited number of gestures in which the fingertips are not visible, the contour points (every 10th point) are used as a basis to calculate the distance. Since this distance gets higher for gestures involving multiple hands and fingers, a normalization operation is applied by the number fingers, *N*, detected in an image in order not to bias the recognition. For the same reason, the disparity value for an entire image is obtained by dividing the total hand disparity over all detected hands by the number of hands, *M*. If this value is lower than a threshold (e.g. the maximum Euclidean distance accepted for a candidate gesture in TABLE I), the gestures are

considered similar (Figure 8c: 29 < 30) or else they are rejected as possible candidates (Figure 8b: 55 > 30).



(a)



(b) (c)

Figure 8 Gesture disparity calculation between the (a) the observed and (b, c) stored gestures for the selection of gesture candidates where Sx and Fx define respectively the stored gesture and the observed gesture of frame x.

In step (2), we use the relative position of the palm center(s) with respect to the corresponding palm center of the last recorded image, as illustrated in Figure 9, to validate the similarity between each observed version and each identified candidate reference gesture. From this we can assign a disparity value for each cell of the DWT matrix in Figure 9.

The gesture disparity is calculated as the sum of each image disparity gathered along the path of least disparity divided by the number of images. To build this path, we start by choosing at

each image (or cell) of the path, starting from the last frame of the candidate and stored gesture, the neighboring cell presenting the least amount of disparity. The set of cell discovered along the entire path (P) can then be use to validate the similarity between a candidate and the observed gesture. To quantify the similarity between candidate and stored gesture, we use eq. (3) based on the set of cells (P), illustrated in Figure 9b, representing the minimum cost path of the DTW matrix:

$$Min_{cost} = \frac{\sum_{e=1}^{\#P} \Delta SO_e}{\#P} \quad \textbf{(3)}$$

where $e = \{(x, y) | x \in S, y \in F,\ (x, y) \in P\}$, #P represents the cardinality of the set P described above, S is the set of stored gesture frames, and F the set of observed gesture frames, each set containing 40 frames. To calculate the stored and observed gesture disparity (ΔSO_e) in a cell of the DTW matrix we use the same formula for the selection of candidates (eq. (1)), but with $\Delta d_{ije} = ||d_{ijx} - C_{jx}| - |d_{ijy} - C_{jy}||$ and $\Delta C_{je} = |||C_{jx} - C_{j=1,x}| - |C_{j,x=40} - C_{j=1,x=40}|| - ||C_{jy} - C_{j=1,y}| - |C_{j,y=40} - C_{j=1,y=40}|||$ where i and j refers respectively to the finger and hand identification number, $x \in S$ and $y \in F$.

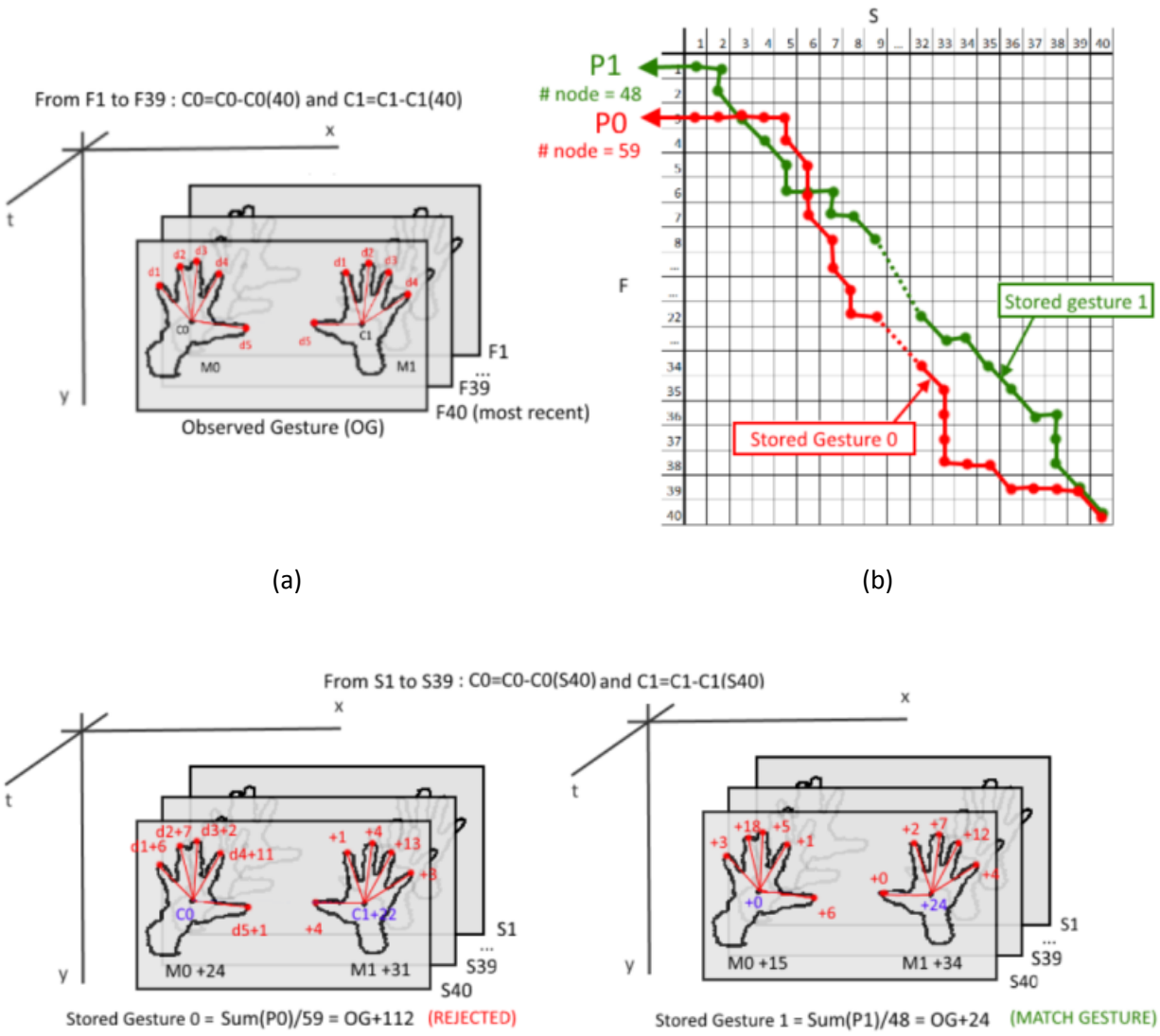


Figure 9 Gesture disparity calculation for the validation of the recognized gesture between (a) observed gesture and (c, d) stored gestures using the (b) DTW matrix.

TABLE I summarizes the empirical choice of parameters used during experimentation for the DTW. The value for the maximum Euclidien distance, in pixels, accepted for a candidate gesture is selected as 30. It represents a good compromise between the number of candidate gestures to be processed and the processing time, as in this case the 40 last images in the list are compared to make a decision. The maximum cost accepted for a path in the DTW matrix per image is set empirically to 30. When we perform a search in the DTW matrix for the shortest path, in the virtual space of frames (observed vs. reference gestures), as illustrated in Figure 10,

the neighboring values of static gestures are in general equal (Figure 10a). Thus any direction in the DTW matrix could be chosen and the associated gesture could be falsely rejected based on the path taken. To avoid this problem, the search is biased (diagonal weight wd is lower than vertical, wv, and horizontal, wh, weights) to favor the next image of both reference and last gesture performed, corresponding to a diagonal movement in the DTW matrix. The weights associated with the directions of movement are experimentally set to 0.005 for vertical and horizontal direction and 0.003 to the diagonal direction, in order to favor the latter. To a lesser extent, these weights play a similar function between frames (observed vs. reference gestures) of dynamic gestures, but generally the neighboring values are different due to the gesture's movement (Figure 10b). In this figure, values in circles inscribed in each cell represent the hand disparity for each pair of observed and stored gestures before applying the DTW direction weights (wh, wv, wd). The product of these weights with the circled values gives the value used for minimal cost computation, indicated in color on the bottom of each cell.

TABLE I Parameters for the Dynamic Time Warping Algorithm.

Parameters	Value
The maximum Euclidean distance, in pixels, accepted for a candidate gesture	30
The number of vertical or horizontal movements allowed in the shortest path (HorizontalMovementThreshold, VerticalMovementThreshold)	10
Maximum cost accepted for a path in the DTW matrix per image (PathCostThreshold)	30
Weight (wp) used for the difference between the relative palm position in the last image of the reference gesture and the one observed during candidate search	7
Weight used for the difference of palm position relative to the last image between each image of the candidate reference gesture and observed gesture at the construction of the DTW matrix	7
Weight (wf) used for the difference between the finger position with respect to the palm center in the last image of the reference gesture and the one of the observed during candidate search	1
Weight used for the difference of finger position with respect to the palm center between each image of the reference gesture and observed gesture at the construction of the DTW matrix	1
Weight used for direction choices (horizontal (wh), vertical (wv), diagonal (wd)) during the search of the shortest path	(0.005, 0.005, 0.003)

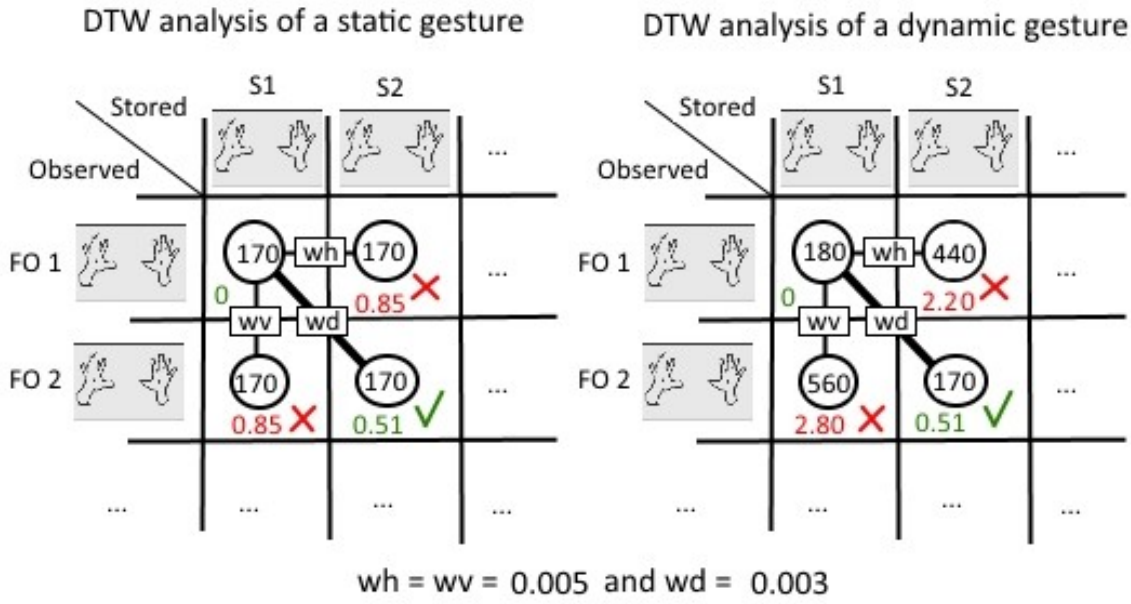


Figure 10 DTW matrix with disparity values for (a) static and (b) dynamic gestures.

To further improve the performance, a maximum number of 10 horizontal and vertical movements are allowed. This results in a faster elimination of divergent candidate gestures. In order to select the winning gesture among the gesture candidates, the one with the smallest disparity value (and that respects the threshold) is chosen.

The pseudo-code summarizing the approach is shown in Algorithm 4.

Algorithm 4 : Gesture recognition using DTW

Input:

$\mathbf{g}_{obs}$ = Last observed gestures (40 images)

$\mathbf{g}_{cand}$ = List of candidate gestures (40 images)

Output:

$\mathbf{g}_{recogn}$ = Name of recognized gesture (or none: gesture not recognized)

For each gesture in $\mathbf{g}_{cand}$:

```
// Get the 2D DTW matrix containing the disparity between each
// image of the candidate (stored) and observed gesture.
dtwMatrix = FindDtwMatrix (g_obs, gesture)
// Use a Greedy algorithm to find the minimum path cost in disparities
// in the DTW matrix; return -1 if the cost between two images is ∞
// or the horizontalMovementCounter > HorizontalMovementThreshold
// or the verticalMovementCounter > VerticalMovementThreshold
pathCost = FindLowestCostPath(dtwMatrix)
// Calculate the average Pathcost per image
minPathcostPerFrame = pathCost / gesture.Frames.Count
// Maximum cost accepted per frame = 30
If (minPathcostPerFrame < 30 and pathCost != -1):
    return g_recogn  //name to be displayed on screen
else
   return none;
end
end
```

3.5 Robotic arm posture control application

In a first attempt to control the end-effector of the robotic arm, the user's hand palm center position was mapped to the Cartesian space of the robotic arm (which in turn was mapped to the joint space through IK calculation). Needleless to say that even with the use of filters, and a very still user's hand, the calculated center position was too unstable due to the amount of recognized contour points around the hand adding up their margin of errors. To achieve a more stable end-effector position, we decided to use the position of the fingertip as the source, combined with an average filter and outlier rejection algorithm. This feature is applied to all three posture control methods proposed in Chapter 4. In the Attraction point method explained later in Section 4.3, the direction of the fingertip is used to control the position of an attraction point. To start or browse within the different modes inside one robotic arm posture control method, pre-recorded static gestures are used. The dynamic gesture algorithm was not used in the current thesis (See Chapter 4), but rather used as source of inspiration to create an adapted version for the recognition of a robotic arm posture in the Tracing posture control method (Section 4.4). The goal is to select the best matching possible posture from a trace of the desired posture executed by the user between the EE and the robotic arm base joint. The method is further detailed in Section 4.4.

Chapter 4 Natural control of a robotic arm posture using gesture recognition with an inverse kinematic exploration approach

Unlike several methods found in the literature about master-slave systems, a contactless approach where the user does not have to wear or touch any kind of device (sensor or controller) is less physically cumbersome for the user. A simple and affordable contactless approach can be found in vision-based gesture recognition. Some gesture recognition systems use markers or wearables that are not considered contactless and come with similar downside consequences as master-slave systems. Indeed, a vision-based gesture recognition system using markers strategically placed on the user for motion tracking not only infringes on the user's movements, but also requires a more complex and expensive system in constant need of calibration [3][35]. Contactless gesture recognition offers simplicity, low material cost and natural gesture opportunities. For these reasons, we decided to use a contactless strategy which consists of recognizing static and dynamic human hand gestures based on depth data collected by a Kinect sensor, as detailed in Chapter 3. Additionally, mimicking the exact movements of the human arm, as proposed by various authors, might be optimal in terms of intuitive control when the robot has a similar configuration (DOF, type of joints, etc.) and constraints (joint angle limits) compared to the human arm, but can become cumbersome or plainly impossible to apply otherwise. The work presented in this thesis leverages the static and dynamic human hand gesture recognition system presented in Chapter 3 to naturally control heterogeneous robotic arms.

4.1 Introduction to the proposed posture selection methods

In this chapter, we investigate alternative methods for resolving the obstacle avoidance problem in conjunction with a given task performed by a user manipulating a remote robotic arm. The proposed approaches perform a posture control through IK exploration and natural gesture recognition to intuitively and naturally allow the user to control both the end-effector

and the overall posture of a given robotic arm in real-time. As stated in the objectives of Section 1.4, the posture control approach is developed for most robotic arm configurations using different types and number of joints and joint constraints. To validate our propositions, we use a 4 DOF anthropomorphic arm configuration as illustrated in Figure 11 to experiment in both simulated and real-world environment. To simplify the description of the three proposed approaches that follow in this chapter, a simple 3 DOF planar arm configuration is use in most of the examples. Take note, however, that they can easily be applied to all kinds of 2D and 3D configurations.

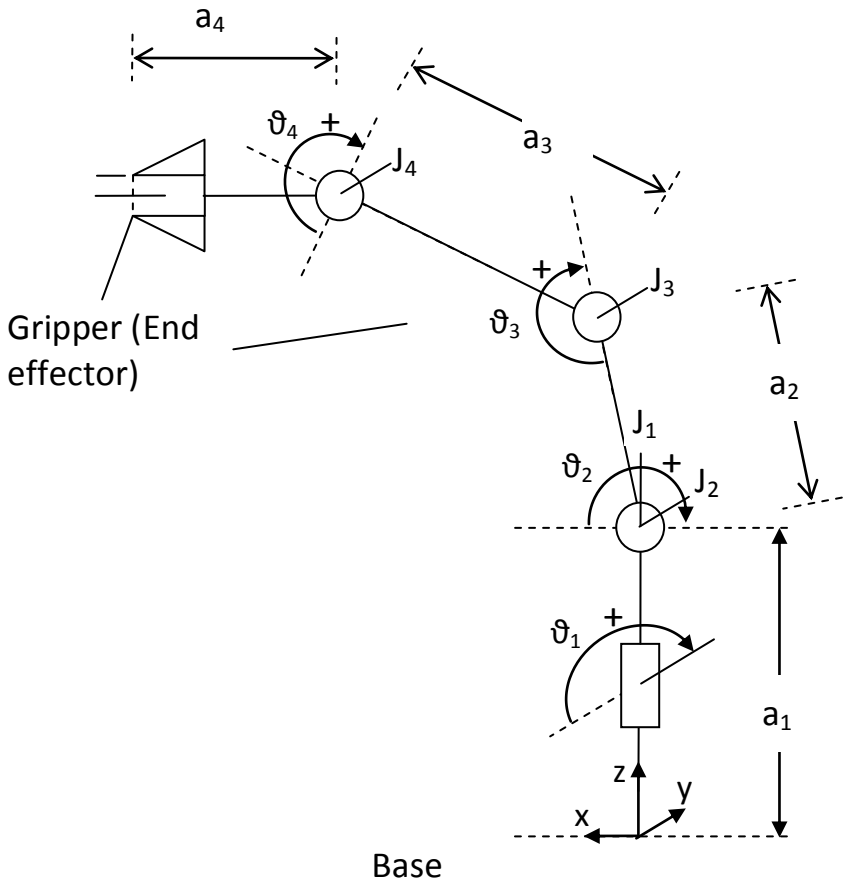


Figure 11 4 DOF anthropomorphic arm.

Recognized static gestures are used to start (without previous calibration) each proposed application, select or change mode inside applications. For all proposed approaches, the "one"

gesture with index finger, as shown in Figure 12, is used as the initial gesture to start each application. This allows the user to start with the hand position in the center of the Kinect field of view and half way between the minimum and maximum range of the Kinect sensor depth of field.

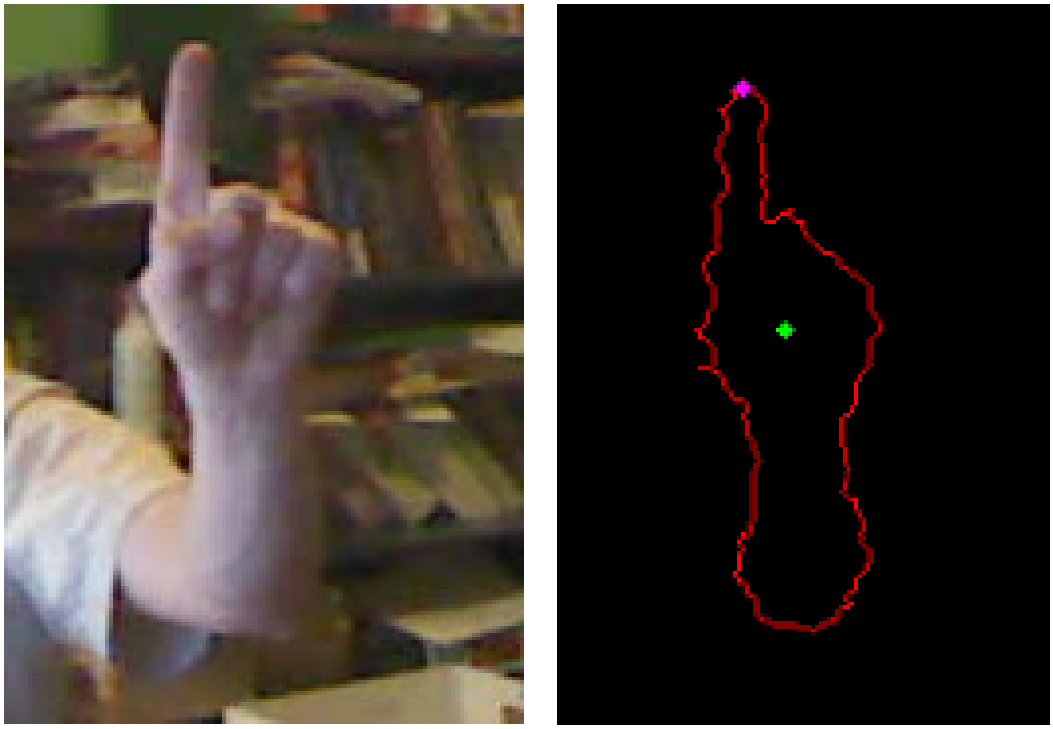
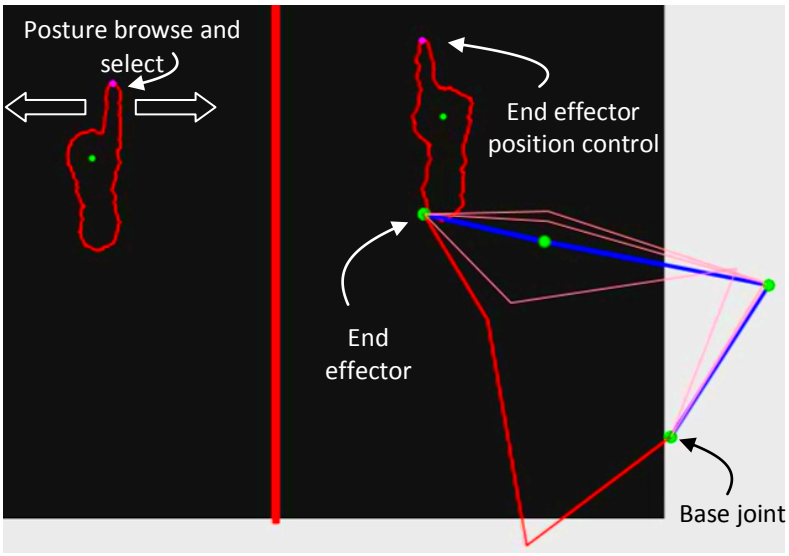


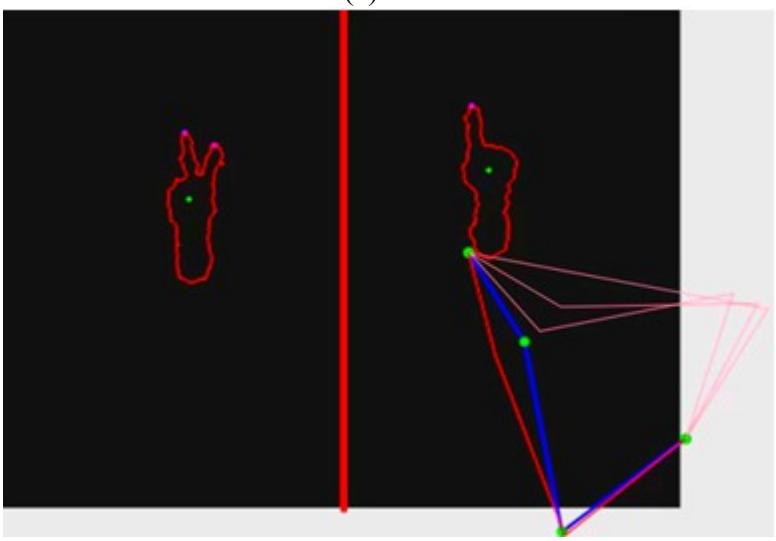
Figure 12 Initial gesture used to start each posture control application : "One" in sign language.

4.1.1 Method 1: Browse and select

A first proposed solution, named “Browse and Select”, involves a graphical interface containing a visual display of the robotic arm with a set of possible postures $\bar{\vartheta}_s = \{\bar{\vartheta}_1, \bar{\vartheta}_2, \bar{\vartheta}_3, \dots, \bar{\vartheta}_i\}$, where $\bar{\vartheta}_i = [\vartheta_1, \vartheta_2, \vartheta_3, \dots, \vartheta_j]_i$ with ϑ_j being the angle between link j-1 and link j connected to joint j, capable of reaching a new specific target end-effector position close to the current robotic arm posture. The user can browse in between the proposed postures and select the desired new posture in real-time using left and right displacement with a dedicated hand, while the other hand positions the end-effector. The aim is to offer the user an equally distributed spread of posture choices. To recognize the intention of the user, a simple tracking of the fingertip position of interest along with the number of detected hands and fingers are used instead of reserved static or dynamic gesture recognition. This strategy gives faster results when selecting a desired posture and allows the user to control the EE position and browse at the same time. In Figure 13a, the browse process is presented with *N*=4 possible postures on the base joint angle plane where pink lines represent (non-selected) possible postures of the arm, the red line is the currently selected possible posture, and the blue line the current posture. The selection process is presented in Figure 13b where the user confirms the currently selected possible posture with a "V" sign, before it becomes the new current robotic arm posture (in blue).



(a)



(b)

Figure 13 (a) The next posture selected (in red) with respect to the current robot arm posture (blue) can be changed among other possible postures (in pink) by moving the left hand finger towards the left or right direction; (b) the currently selected configuration becomes the new posture (changes from red to blue) and a new set of possible postures is generated when the "V" sign is detected on the left hand.

Although this display gives enough information in the case of a 3 DOF planar arm, as shown in Figure 13, some 3D robotic arm configurations might need to display a more complete posture representation using at least two planes of view. Another possibility would also be to present an isometric projection that can be rotated by the user to give a better idea of the selection of postures in a 3D environment. This would require additional human gestures to be defined and assigned to the controls needed for such a feature.

4.1.2 Method 2: Attraction point

In the second proposed solution, named “Attraction point”, only one hand is used for tracking a fingertip direction and position to control respectively the robot posture and EE position. Like the Browse and Select approach, finger tracking was used to capture the intention of the user as there was no need for static or dynamic gesture recognition. In this method, the direction is mapped to a point of attraction that will influence the way the robotic arm joints bend. The control over the posture is done by recalculating the IK with different initial joint orientations. From the set of possible posture found, the posture with the closest set of joints relative to the attraction point is chosen as the current posture. The exploration part of this method was further optimized by favoring the exploration of specific joints angle variation based on their influence on the overall posture distance from the attraction point.

In this approach, we leverage the ability of the gesture recognition system to detect the angle between the axis of the finger passing through the fingertip and the reference axis in the Kinect view plane. This angle is used to calculate the position of an attraction point by directly mapping it to θ_a, the angle between the vector going from the middle point of the EE-Base joint vector (green vector in Figure 14a) to the attraction point (pink line in Figure 14) and the vector going from the base joint to the end-effector as illustrated in Figure 14a. The attraction point is a point located on a circular trajectory of radius equal to the maximum reach distance of the robotic arm and centered on the middle point of the EE-Base joint vector. The point is used to attract joints toward its position and consequently favor a certain posture based on its location.

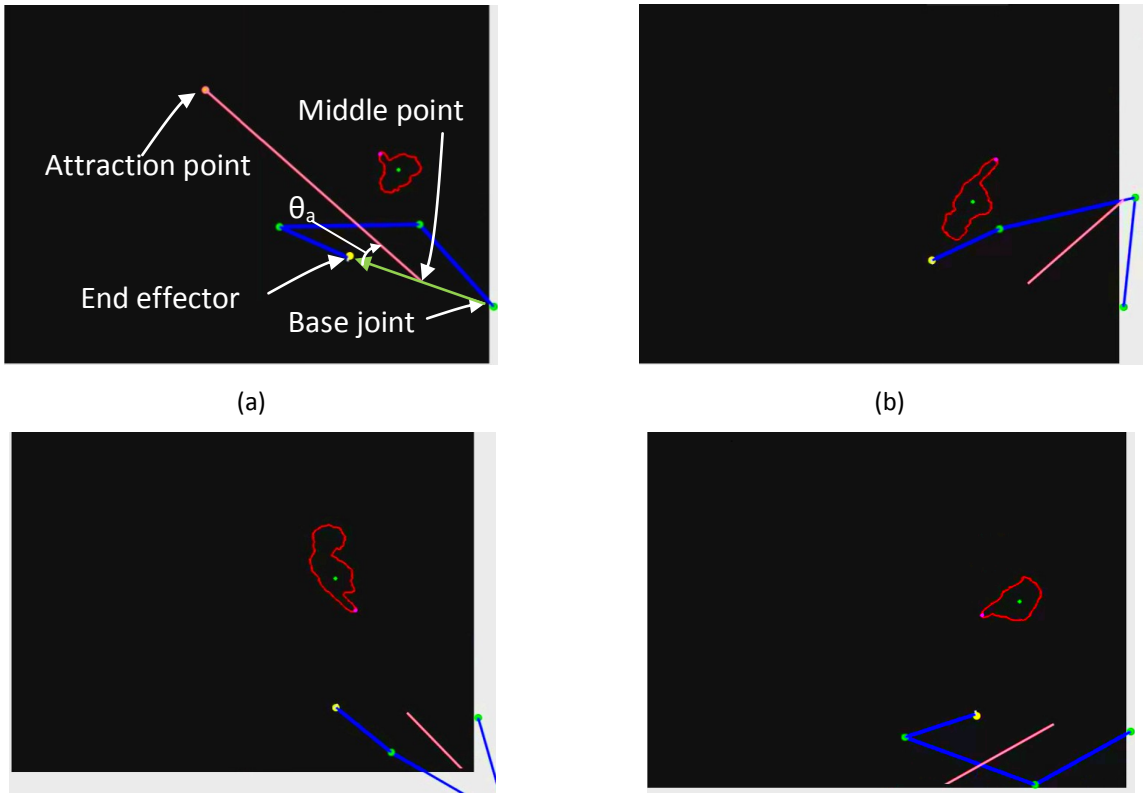
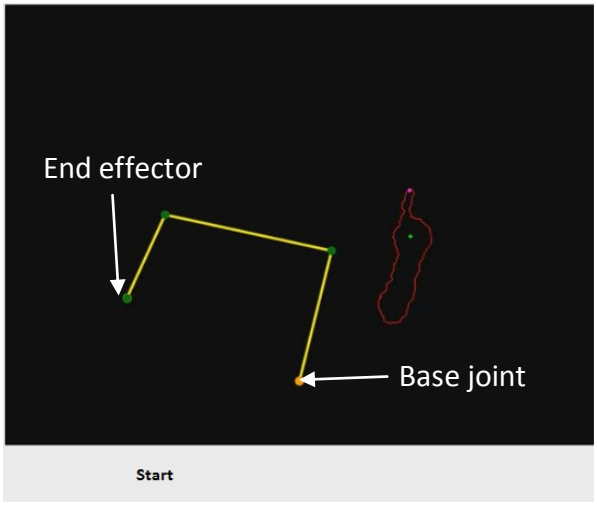


Figure 14 (a)-(d) Illustration in the base joint angle plane of the gesture control using an attraction point controlled by a user's index finger direction and an EE control by the position of the fingertip of the same index finger. The 3 DOF robotic arm is indicated by the blue line and the pink line refers to the directional vector between the attraction point and the middle point between the end-effector and the base joint. The figures represents postures for (a) θ_a=0.846 rad and EE(x, y, z) = (256, 136, 190) mm (b) θ_a=2.356 rad and EE(x, y, z) = (236, 131, 249) mm (c) θ_a=-2.491 rad and EE(x, y, z) = (243, 100, 200) mm (d) θ_a=-0.397 rad and EE(x, y, z) = (207, 91, 206) mm.

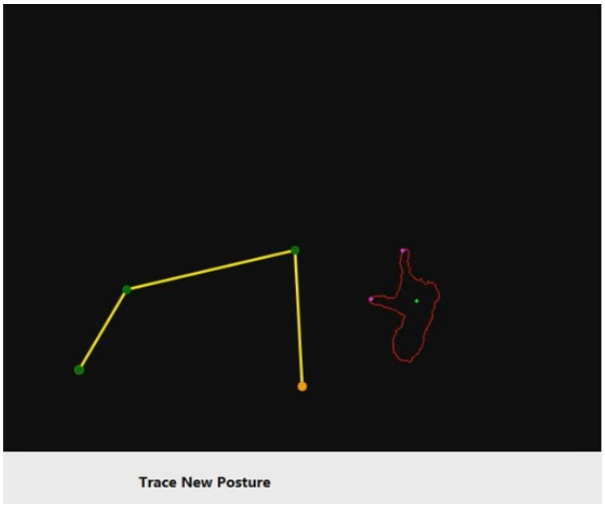
4.1.3 Method 3: Tracing

In the third proposed solution, referred as the "Tracing" application, only one hand is used. The index finger controls the EE position or traces the desired posture when a "Tracing new posture" gesture is recognized. As opposed to the two previous proposed approaches, this method relies on both finger tracking and static gesture recognition to capture the intention of the user. The dynamic gesture capability of the system described in Section 3.4 was not leveraged for the proposed applications presented in this thesis, but was used as inspiration in designing the robotic arm posture recognition system part of the "Tracing" application.

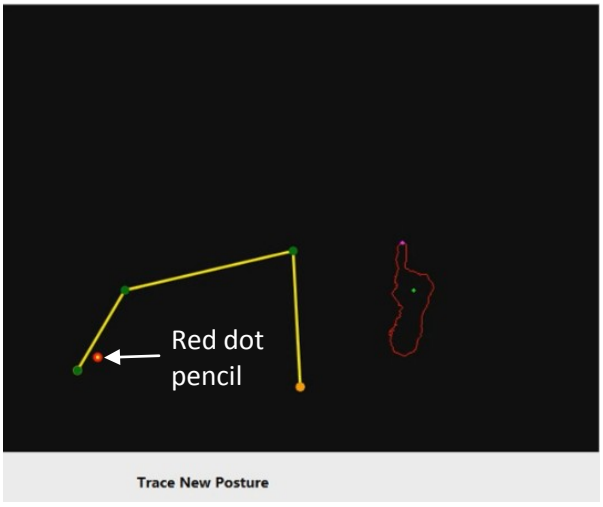
To perform a "Tracing" posture control, the user needs to select "Tracing" on the user software interface and press the “Gesture Recognition” button. The system waits for the initial gesture to start (Figure 15a). If the user performs the initial gesture, the system then waits for either an "End-effector Displacement" gesture (position change of the index fingertip using the "one" gesture shape, see Figure 12) or a "Trace new posture" command gesture (L-shape) detailed in Figure 16. If the user performs an "End-effector Displacement", the robotic arm becomes simply controlled by the IK of its end-effector position, which is mapped to the user's index fingertip position. If the user performs a "Trace new posture" gesture (Figure 15b), the system displays the current posture of the robotic arm and a red dot pencil (Figure 15c). The red dot position is directly mapped to the user’s index fingertip position. The user's index direction is not used in this method. As soon as the dot reaches a distance of 50 mm from the base joint, a first link is added to the desired posture trace and displayed on the simulator interface. The following links get added at 50 mm intervals until we get close enough to the target position (Figure 15d-k). Consequently, each link of the desired posture is of length equal to 50 mm, except the one link to the target which might be less than 50 mm. Once the last link reaching the target is added, the system explores the IK possible postures output and uses the Dynamic Time Warping natural control recognition system to compare and find the most similar posture to the traced desired posture. The most similar posture obtained is then displayed (thin white robotic arm displayed in Figure 15l. To confirm and send the suggested posture to the real robotic arm, the user can perform the "Trace done" command gesture (Closed fist, as shown in Figure 17) as illustrated in Figure 15m-n.


End effector
Base joint
Start

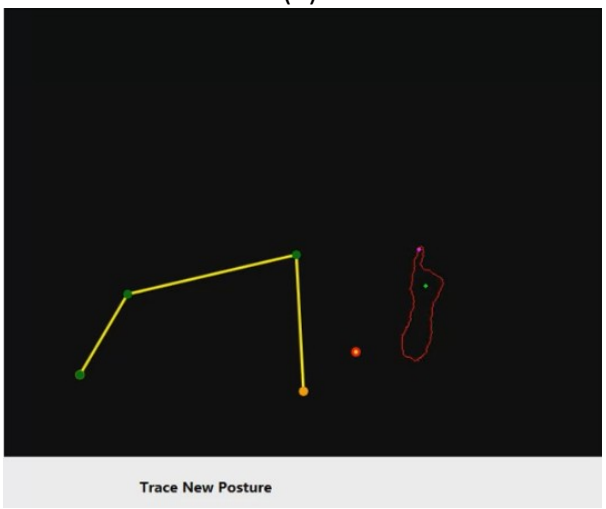
(a)


Trace New Posture

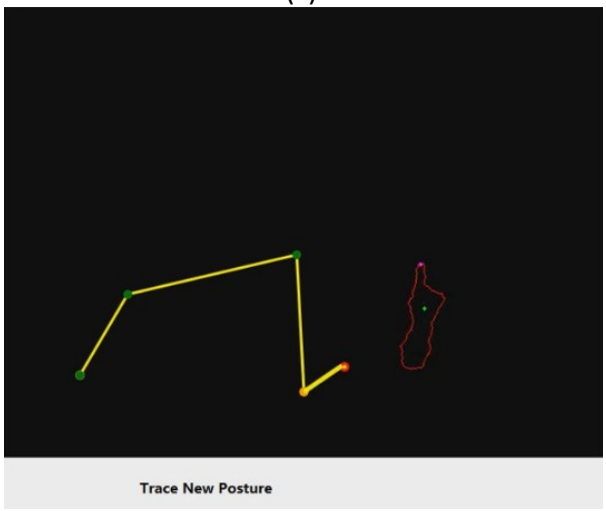
(b)


Red dot
pencil
Trace New Posture

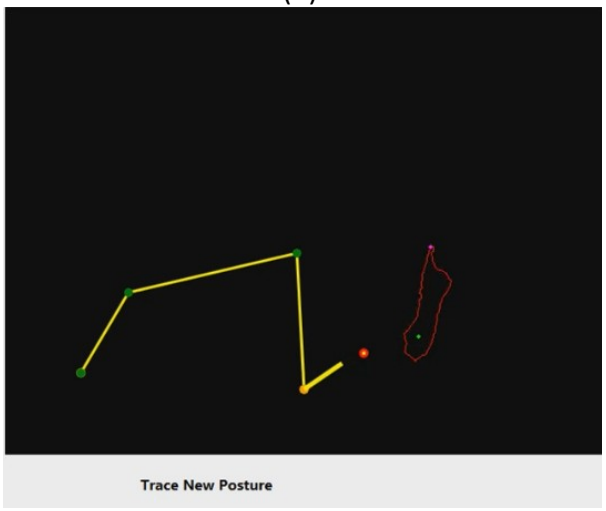
(c)


Trace New Posture

(d)


Trace New Posture

(e)


Trace New Posture

(f)


Trace New Posture

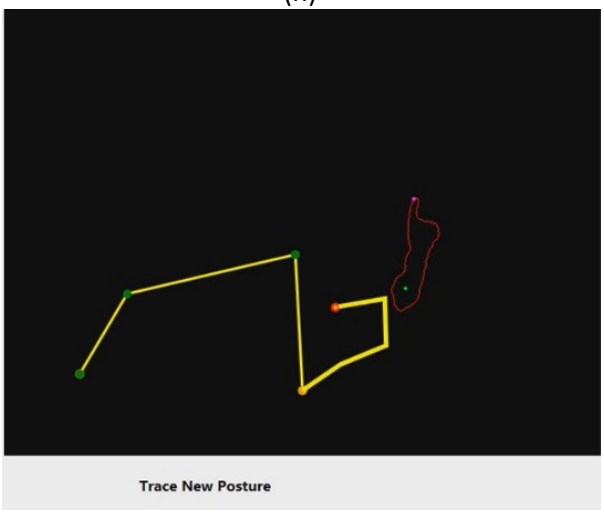
(g)


Trace New Posture

(h)


Trace New Posture

(i)


Trace New Posture

(j)

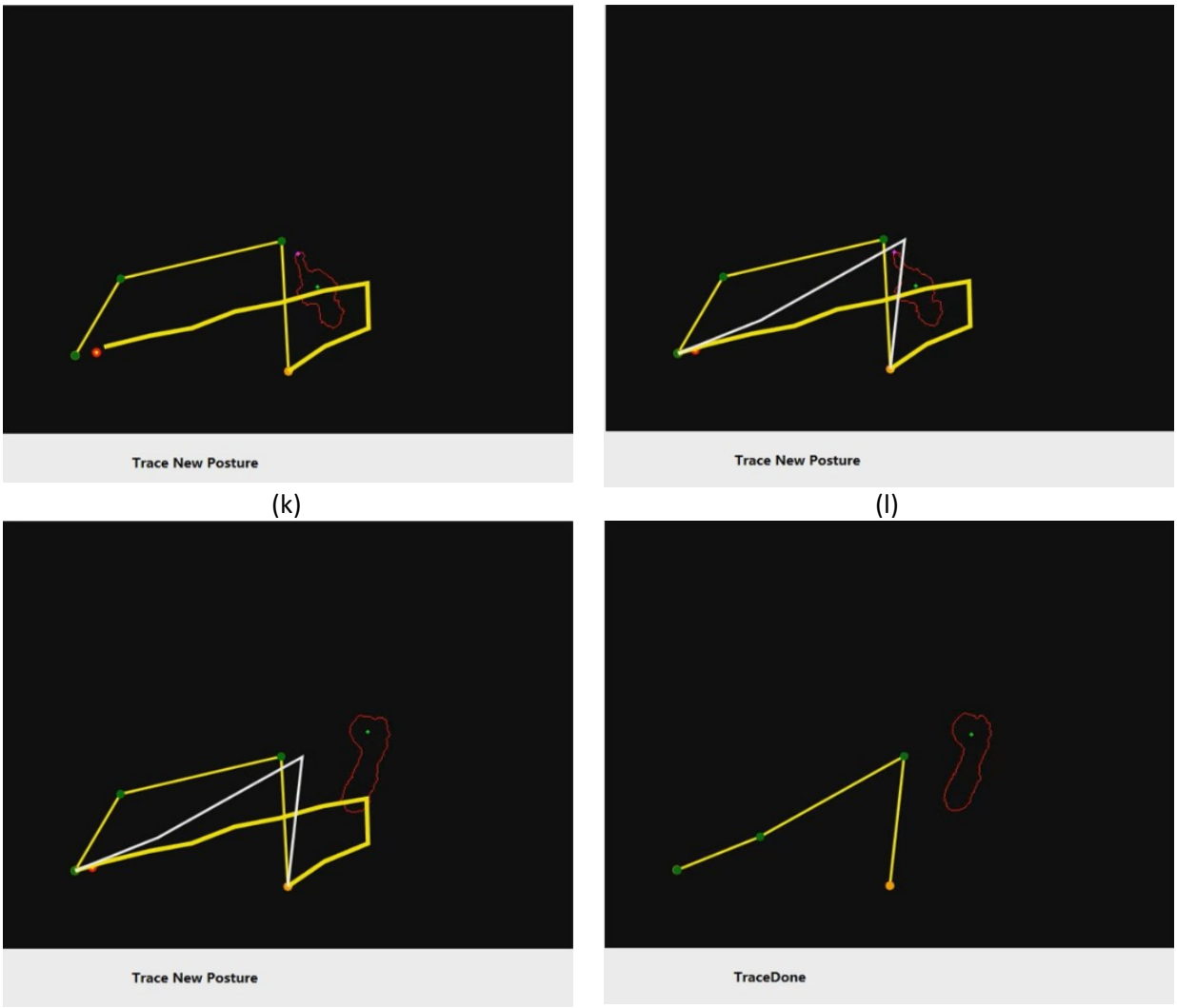


Figure 15 The tracing method interface illustrated on the base joint angle plane: (a) System receives the initialization gesture and waits for an event (EE displacement or gesture recognized). A change in the EE position is triggered by moving the index fingertip ("one" gesture shape); (b) system receives the "Trace new posture" gesture ("L" shape gesture from right hand); and (c) the red dot for drawing the trace appears; (d) to start recording the trace, the first point of the trace must be added (e-k) where a new point of the trace gets added when a fixed distance from the last point is reached; (l) trace is completed when the red dot reaches close to EE, then the most similar possible posture is found and displayed on the interface using thin white links; (m-n) the user accepts the suggested most similar posture to become the current arm posture by performing the "Trace done" gesture (closed fist).

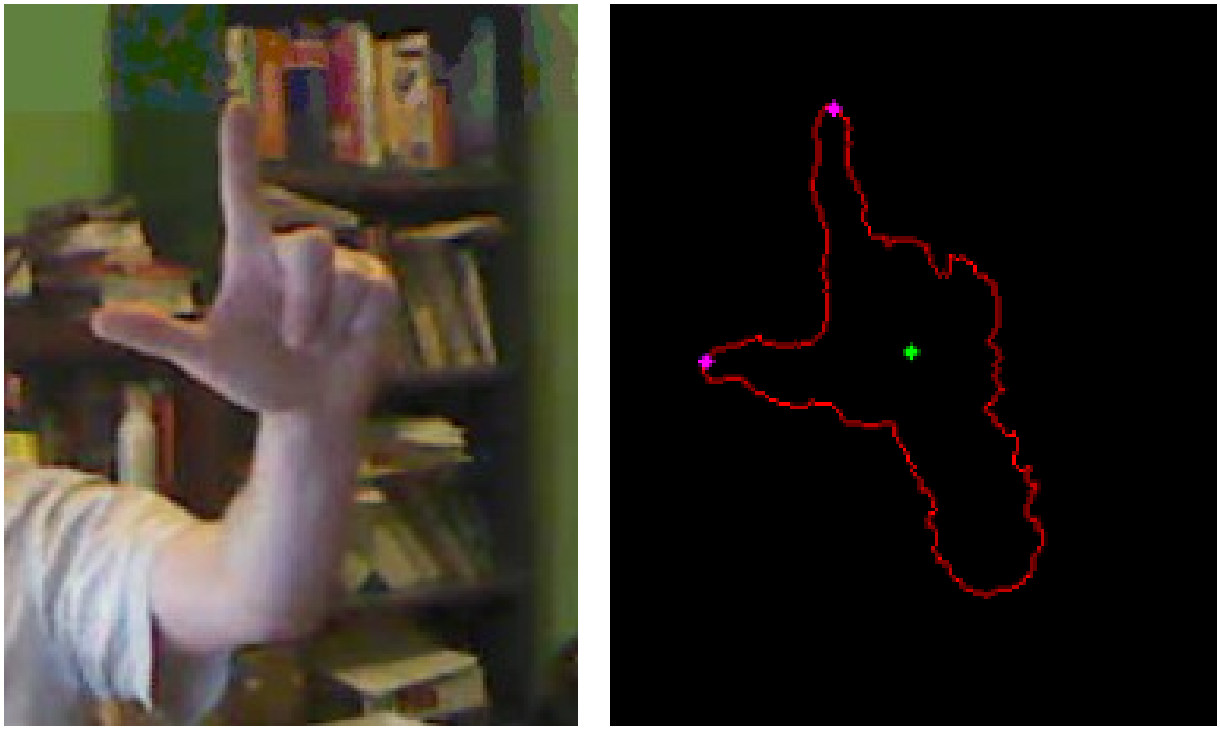


Figure 16 The "Trace New Posture" command is given by user through the execution of the "L shape" gesture.

Figure 17 The "Trace done" command is given by user through the execution of the "Closed fist" gesture.

The posture control system contains the three proposed applications that take their inputs from the hand detection and gesture recognition framework (Figure 4), at the application level, as illustrated in Figure 18. Our use of the hand detection and gesture recognition to control a remote robotic arm device is twofold. First, it leverages the information about the user hands and interprets the index fingertip position according to the application and mode. For example, in the “browse and select” application, if two hands are detected, the index finger of the right hand controls the position of the EE, while the index finger of the left hand browses the proposed postures. Although not implemented in our system for the Attraction point and Tracing methods, the hands role can be switched easily for left-handed and right-handed users. A natural gesture control menu interface could be used to preset the user's preference to this effect.

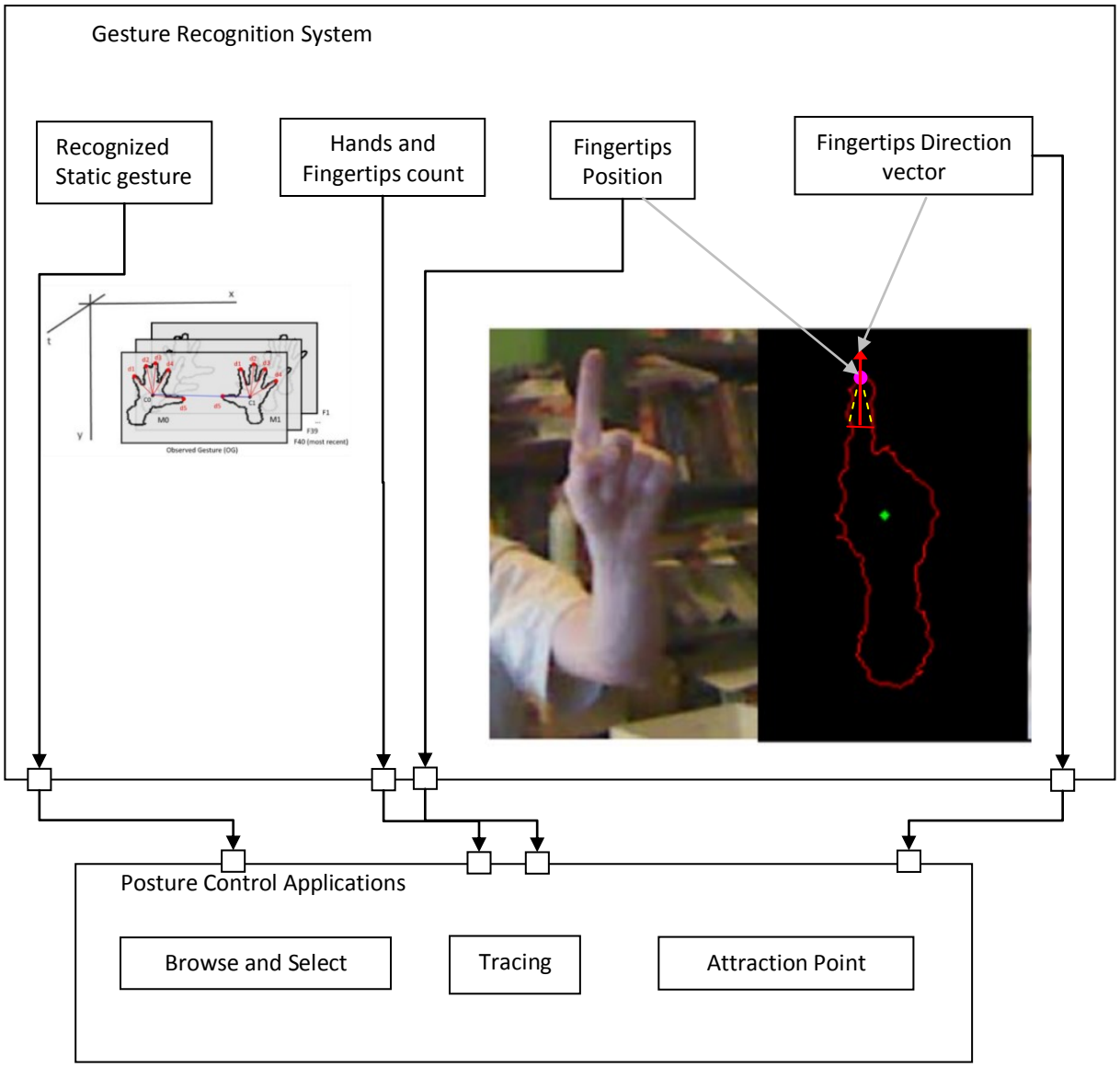


Figure 18 Overall framework of the proposed natural posture control of a robotic arm.

4.2 Teleoperation with posture control using two hands – Browse and select

In the "Browse and select" approach, the user relies on a visual display, as illustrated in Figure 13, indicating in real-time the available (possible) postures in the vicinity of the current robotic arm configuration. The user can browse the available postures using the left hand index and select one of them by making a "V" sign with two fingers to the Kinect sensor with the same hand. Figure 19 illustrates the framework of this approach. Three modes compose this application: Browse, Selection and End-effector displacement. The End-effector displacement is

triggered each time the user's right hand index fingertip moves a certain meaningful threshold distance from its last position in the three dimensional space. When a new position is detected, the position is added to a "First In First Out" (FIFO) queue of the last 10 non-rejected (i.e. non-outlier) EE positions. To detect and reject outliers, we input the resulting FIFO queue to Algorithm 5. We then apply an average smoothing using Algorithm 6 to generate the new EE position and calculate a new set of N possible postures from IK in Algorithm 7. In our application, we use a Smooth factor of 10 and a Rejection tolerance of 7mm. Assuming a frame rate of 15Hz, the EE is expected to move at a velocity below 105 mm/s with the chosen Rejected Tolerance value. Otherwise, the data is rejected as an outlier because the distance between EE from one frame to the next is higher than the Rejection tolerance.

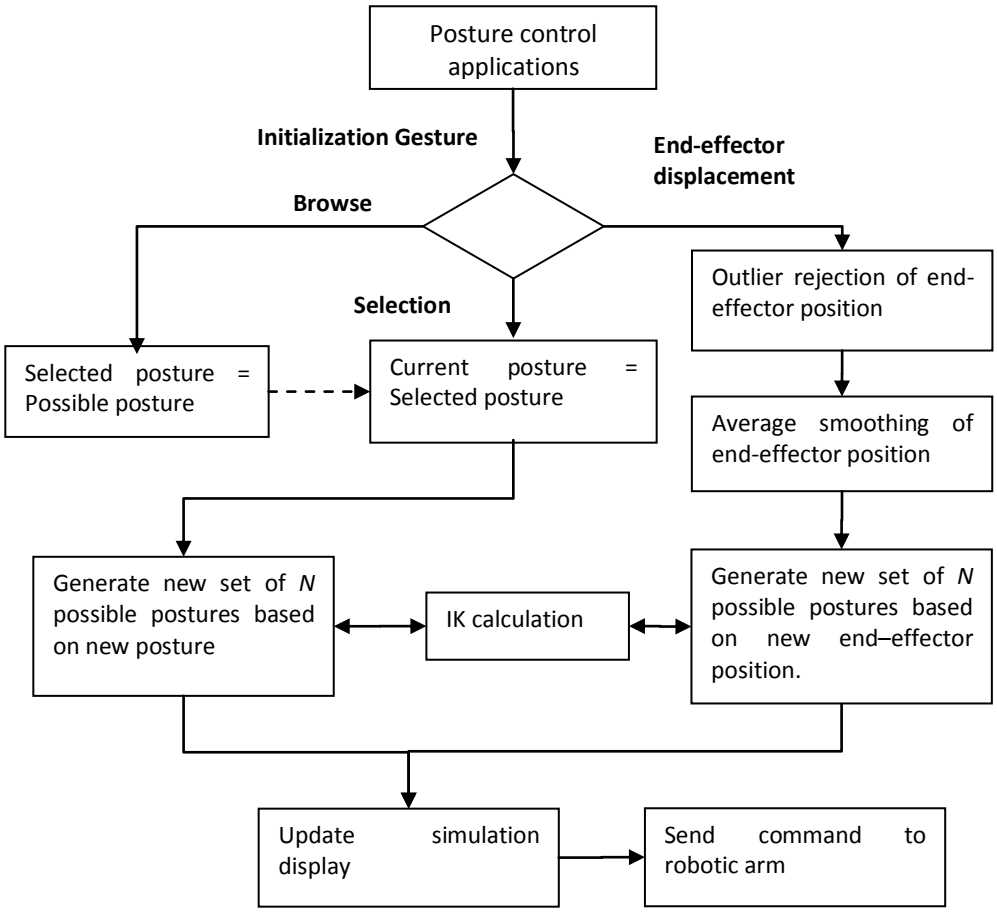


Figure 19 "Browse and select" approach framework.

Algorithm 5 : Outlier rejection

Input:

$\mathbf{Q_{EE}}$ = Unfiltered queue of 3D positions, where $\mathbf{Q_{EE}}$.Count = SF (Smooth factor)

$\mathbf{T_{rej}}$ = Rejection tolerance (> 0) in mm

Output:

$\mathbf{FQ_{EE}}$= Filtered queue **(**without outlier points) of 3D positions

```
For (index = 0; index <  Q_EE .Count; index++) :
   If (index > 0 and index < Q_EE .Count-1)
       // Calculate the Euclidien distance between the current index position
       // and its two immediate neighbours in the queue Q_EE
       tempPreviousDelta = Q_EE[index-1] - Q_EE[index]
      tempNextDelta = Q_EE[index] - Q_EE[index+1]
      If  (tempPreviousDelta.Length < T_rej and tempNextDelta.Length < T_rej) :
         // The point Q_EE[index] is not rejected because it is located
        // inside the intersection space of Q_EE[index-1] and
       // Q_EE[index+1] defined by T_rej where the position is valid.
         FQ_EE.Add(Q_EE[index])
      end If
   Else:
      // We are at either the first or last index in the queue Q_EE.
      //Those points cannot be rejected because they only have one
      // neighbour in the queue (lack of information).
      FQ_EE.Add(Q_EE[index])
   End If
End For
return FQ_EE
```

Algorithm 6 : Average smoothing

Input:

$\mathbf{FQ_{EE}}$= Filtered queue **(**without outlier points) of 3D positions

Output:

$\mathbf{P_S}$= Current position with smoothing applied

SF = $\mathbf{FQ_{EE}}$.Count // Smooth factor

For (index = 1; index < **SF**; index++) :

  // Sum all points

  $\mathbf{P_S.X}$= $\mathbf{P_S.X}$ + $\mathbf{FQ_{EE}}$[index].X

  $\mathbf{P_S.Y}$= $\mathbf{P_S.Y}$ + $\mathbf{FQ_{EE}}$[index].Y

  $\mathbf{P_S.Z}$= $\mathbf{P_S.Z}$ + $\mathbf{FQ_{EE}}$[index].Z

End For

// Divide the sum of all point by SF to get the average

$\mathbf{P_S}$ = $\mathbf{P_S}$ / **SF**

return $\mathbf{P_S}$

The number of candidates postures, M, proposed by the system, can be configured by the user in the software interface before starting the application with the initialization static gesture (Figure 12). These postures are displayed in a graphical interface allowing the user to browse and select one of them to become the current posture. When a left hand is detected with one index fingertip, the "Browse" mode is activated. The user can browse through the different postures available by moving their left index in the left or right direction. When two fingers are detected in the left hand, the last possible posture selected in browsing mode ("Selected posture") becomes the new current posture of the robotic arm. Just like when a new EE position is detected, a new set of possible postures is generated based on the new current posture.

This approach forces the user to constantly pause the EE movement to evaluate and take their next posture decision. While this leaves the user able to control only the EE position on the robotic arm between posture adjustment decisions, it allows a greater stability on the EE movement. It is therefore more appropriate for tasks where posture and EE position accuracy is very important, over speed of execution and continuous posture change capability.

Encapsulated in Algorithm 7, the set of available postures displayed to a user reconsidering the posture of the robotic arm, is gathered by calculating the IK output postures of M = N*(# of varying joints)*2) initial input postures. The set of initial input postures uses N different angle values on both sides of each of the current robotic arm joint value. Each of the N chosen angles on one side of a joint is separated by a step of (joint angle limit – current joint angle)/N) rad starting from the current joint angle value. The joint angle limit refers to the maximum or minimum joint angle value depending on the side of the current joint angle. If one of the initial input postures does not converge to a solution in the IK calculation, then M-1 candidate posture gets displayed to the user. This method can be applied to a robotic arm with revolute joints type oriented along different plane in a 3D environment.

Algorithm 7 : Gathering the set of possible postures

Input: **A** = Current arm posture

T = Target end-effector position

N = Max. number of possible postures to display on each side of A

Output: **Lp** = List of possible arm postures reaching **T** (not exceeding 2**N**)

1. For each joint *i* of **A**:

   S_i = (**A**.Joint[*i*].MaxAngle – **A**.Joint[*i*].CurrentAngle)/**N**

   // Calculate the angle step S_i between each trial to have **N** trials before

   // reaching the joint physical maximum angle

   A.Joint[*i*].CurrentAngle += S_i

While **A**.Joint[*i*].CurrentAngle < **A**.Joint[*i*].MaxAngle and $\mathbf{L_p}$.Count<=**N**/2:

A' = Calculate new arm posture with IK from arm posture **A** and target **T**

// Verify that the resulting posture is not the initial posture or an already

// found possible posture

If A'!= A and A' not in $\mathbf{L_p}$ then

Add **A'** to $\mathbf{L_p}$

End if

A.Joint[*i*].CurrentAngle += S_i

End While

Reset **A** to original joint angle

End For

2. Repeat to add possible postures for joints starting between **A**.Joint[*i*].Angle and **A**.Joint[*i*].MinAngle until $\mathbf{L_p}$.Count=**N**

3. Return $\mathbf{L_p}$

4.3 Teleoperation with posture control using a single hand – Attraction point

In this method, the user does not have to rely on a visual display. The user can influence the robotic arm's overall posture by moving an attraction point. This can be done through the index finger direction vector passing by the index fingertip used to control the position of the end-effector. Hence, both the posture and the end-effector position can be controlled at the same time using only one hand. Figure 20 illustrates the framework of this approach. Three modes compose this application: attraction point displacement, gripper control and end-effector displacement. The "End-effector displacement" is triggered each time the user's hand index fingertip moves a certain meaningful threshold distance relative to its last triggered position in the 3D space. When a new EE displacement is detected, the new position is added to a FIFO queue of the 10 last non rejected EE positions. This queue is used to detect and reject outliers

(Algorithm 5) and apply an average smoothing (Algorithm 6) to generate the new EE position, similar to the first method described in Section 4.2. When a new end-effector position is defined, the application generates a new list L_p of possible postures from IK calculation. The number of posture in L_p can be configured by the user in the software interface before starting the application with the initialization static gesture. These possible postures are evaluated and weighted against the current position of the attraction point according to the formula used in Algorithm 8. The possible posture with the minimum Euclidean distance relative to the attraction point position is selected to become the current posture. Similarly to the "End-effector displacement" mode, the "Attraction point displacement" is triggered each time the user's right hand index finger direction vector rotates a certain meaningful threshold angle in the image plane of the Kinect sensor or the plane defined by the index finger direction vector and its projection on the image plane of the Kinect. When a new angle is detected, the angle is added to a FIFO queue of the 10 last non-rejected attraction point angles. This queue is used to detect and reject outliers and apply an average smoothing to generate the new attraction point position. A third mode, the "Gripper opening", is recognized when two fingers are detected. In this mode, the distance between the two fingertips is mapped to the gripper opening. This last mode does not influence posture selection and was therefore not included in the first method.

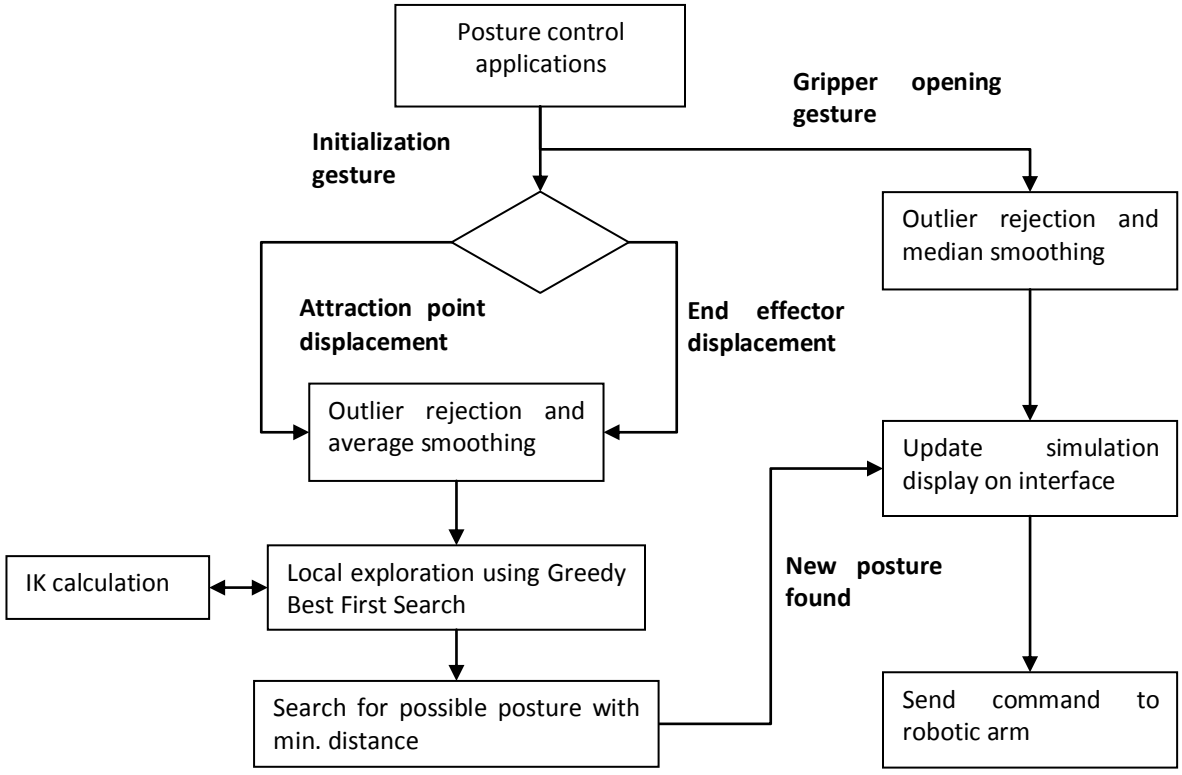


Figure 20 Attraction point framework.

An obvious benefit of this approach is that only one hand is needed by the user to control both the end-effector position and the arm posture, which leaves the second hand free to potentially control another robotic arm with a gripper or a portable camera as the end-effector. Although controlling one robotic arm seems complicated enough, the second arm does not have to performed manipulations as complicated and precise as the first arm, such as recording the action of the first arm with a camera, and could be activated by the user only while the first hand is in pause.

The approach can be seen as complementary to the "Browse and select" solution, described in Section 4.2, as it is less precise but more intuitive and faster to use and capable of continuous posture changes. Indeed, this approach can only control a limited number of possible postures since only one attraction point is controlled by the user. In this method, the same finger used to position the end-effector also controls the selection of the robotic arm posture by measuring the finger's relative angle in the images plane of the Kinect sensor and over its field of view. The resulting angle is then used to position the attraction point around the robotic arm. To change

the posture, the attraction point draws each moveable joint (i.e. all joints except the base joint and the end-effector) closer in its direction. Algorithm 8 presented below was designed to find the posture able to reach the target with a minimum sum of the Euclidean distances between each of its moveable joints and the attraction point. Notice in Algorithm 8 that we also raised each calculated distance (Distance (G, j)) to the power of 3 to avoid favoring postures with very little variation over each of its joint distances relative to the attraction point. Indeed, the attraction point influence might affect all joints but we want to favor cases where a posture increases the distance of one of its joints to allow another to become closer. This helps the attraction point to have even more influence over the robotic arm posture.

The attraction point is positioned on the perimeter of a circle of radius equal to the maximum reach of the robotic arm (L_{MAX}) and centered half way between the end-effector and the base joint (L/2) as represented by the point O_A in Figure 21. This allows the attraction point trajectory to be outside the operational space of the robotic arm. Figure 22a illustrates the direction vector of the index finger that controls the attraction point position. This vector is given along with its projection on the image plane of the Kinect. θ is defined by the angle between the index direction vector projection and the horizontal axis on the image plane of the Kinect and ϕ is defined by the angle between the index finger direction vector and its projection on the image plane of the Kinect. These two parameters control the position of the attraction point. As illustrated in Figure 21 and Figure 22b, the finger direction angle θ controls the angle between the EE-Base vector and the attraction point vector projection (in red) on the plane defined by the EE-Base vector and the z axis, while the ϕ finger direction angle controls the angle between the attraction point vector and its projection on the plane defined by the EE-Base vector and the z axis. The reachable θ angles by the index finger cover the entire range 0 to 2π rad. As for the ϕ angle, it can technically range from $-\pi/2 \leq \phi \leq \pi/2$ rad, but only a reduced subset of this range is possible. Indeed, the articulation constraint on the user's index finger makes it difficult to reach angles $-\pi/2 \leq \phi \leq 0$ rad and the finger recognition system cannot accurately recognize a fingertip position when directly in front of the rest of the hand (i.e. directly between the Kinect and the hand, which is estimated to occur around $\pi/2 \leq \phi < 3\pi/8$ rad). With a range reduced to $0 \leq \phi \leq 3\pi/8$ rad, a transformation is needed to map the index

finger direction parameter ϕ to an equivalent parameter, $\beta(\phi)$, that can position the attraction point without any restrictions on the range ($-\pi/2<\beta(\phi)<\pi/2$ rad): $\beta(\phi) = -\pi/2+(8\pi/3)\phi$.

In Figure 22b, an example of the attraction point method applied to a more complex robotic arm configuration (6 DOF) in a 3D environment is presented. Note that the finger direction information received from the hand and gesture recognition system is a tri-dimensional vector. Hence this solution is applicable in both 2D (Figure 21) and 3D (Figure 22) environment. Moreover, it can support practically any arm configuration (i.e. number of joints, length of links), because it controls the global form of the posture instead of focusing on each joint.

Algorithm 8 : Search for posture with minimal Euclidean distance relative to the attraction point position

---

Input: $\mathbf{L_p}$ = List of possible arm postures

G = Attraction and point position (calculated from the user's index finger direction on the image plane of the Kinect sensor and on the plane defined by the index finger direction vector and its projection on the image plane of the Kinect)

Output: **A** = Possible posture with a minimum sum of the Euclidean distances for each of its moveable joints relative to the attraction point

---

1. For each posture *i* in $\mathbf{L_p}$:

   For each moveable joint *j* in *i*:

   DistanceTotal += (EuclideanDistance(**G**, *j*))^3

   if(DistanceTotal < minDistanceTotal)

   minDistanceTotal = DistanceTotal

   A = *i;*

2. return **A**

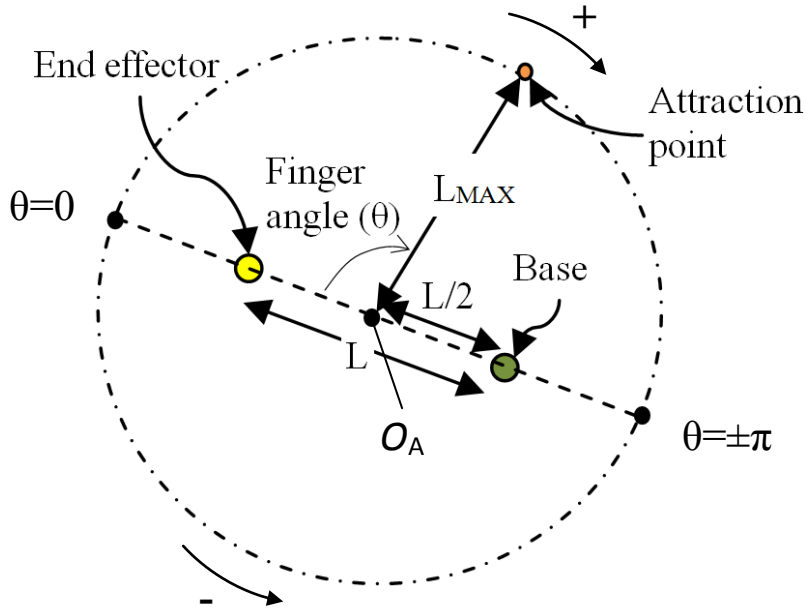
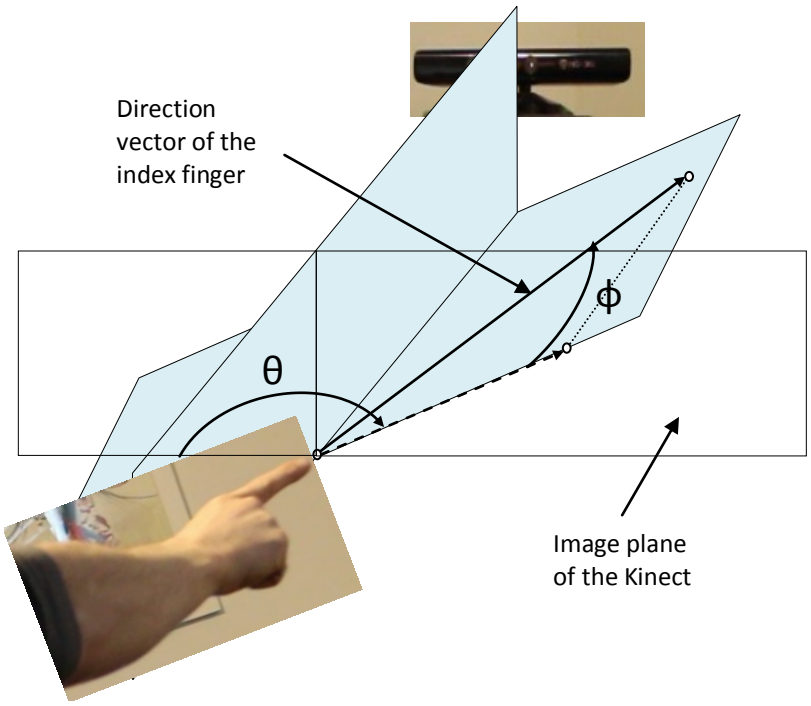
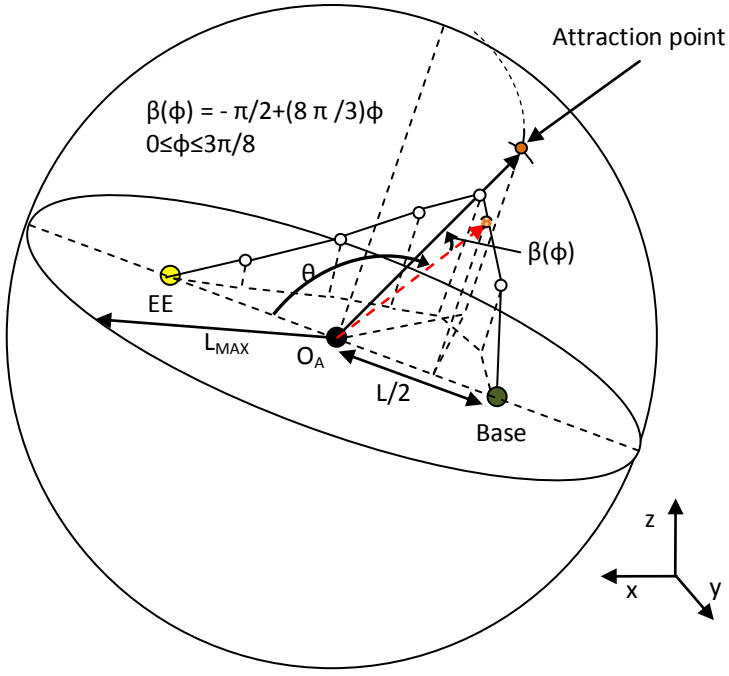


Figure 21 Position of the attraction point in a 2D environment relative to the arm in the x-z plane and the finger angle (θ) on the Kinect image plane.



(a)



(b)

Figure 22 (a)The finger direction vector angle (θ) on the Kinect image plane and the finger direction vector angle (ϕ) between the finger direction vector and its projection on the Kinect image plane. (b) The position of the attraction point in a 3D environment relative to a 6 DOF 3D robotic arm with joints.

Another approach could consist of favoring a certain posture exploration path based on the attraction point position using a greedy algorithm. Since any change in the end-effector or attraction point position is equal to a small increment just above a certain threshold, it is reasonable to assume that the new optimum posture might also be close to the current robotic arm posture. Hence, starting our search using initial postures near the current posture should help find a posture closer to the optimal posture and would be faster compared to calculating the IK of each initial posture from a list of equidistant initial postures across all possible joint angles.

First, the current posture distance is calculated for the new attraction point position. Then all converging postures found from the initial condition position in the neighborhood of the current posture have their distance calculated using Algorithm 8. The possible posture with the lowest distance value takes the place of the current posture and we repeat the process. In Figure 23, we illustrate an example of this method with a 3 DOF planar arm. Assuming a fixed $\vartheta 3$ with the IK method, we start at the joint angle $\vartheta 1 = \vartheta 1(\text{initial}) + \Delta\vartheta 1$ and $\vartheta 2 = \vartheta 2(\text{initial}) + \Delta\vartheta 2$ where $\Delta\vartheta 1=\Delta\vartheta 2=0$. The cubic distance to the attraction point value for this posture is ~500 $x10^6$. We calculate the distance of the IK output posture from every initial posture neighbor to $\vartheta 1=\vartheta 2=0$ and find the one with the lowest distance value. Here, the resulting posture ($\Delta\vartheta 1=1$, $\Delta\vartheta 2=-1$) coming from the initial posture gives us the minimum distance (495 $x10^6$). We then repeat the process above with the initial posture's neighbor to ($\Delta\vartheta 1=1$, $\Delta\vartheta 2=-1$), find the minimum distance (295 $x10^6$) at ($\Delta\vartheta 1=2$, $\Delta\vartheta 2=-1$) and so on, as illustrated by the red line in Figure 23. When evaluating the resulting posture resulting from an initial posture, we verify if this initial posture was not already evaluated. As illustrated by the **While** condition in Algorithm 9, the process is terminated when no resulting posture coming from the initial posture's neighborhood are found with a distance that is lower compared to all previous evaluations.

Δϑ1\Δϑ2	-4	-3	-2	-1	0	1	2	3	4
-4									
-3									
-2									
-1				600	null	null			
0			null	600	500	900			
1			300	495	500	800			
2		275	300	295	null				
3		480	255	500					
4		365	370	290					

Figure 23 Search for the possible posture with lowest Euclidean distance using the greedy algorithm. Different angles (in degrees) of Δϑ1 and Δϑ2 relative to the current initial posture are explored (value units are 1x10^6).

Algorithm 9 : Local posture exploration using greedy algorithm

---

Input: $\mathbf{A_p}$ = Initial arm posture

A_{tt} = Attraction point position relative to O_A

T = Target position

R = Search angle resolution

Output: **S** = Status of success of the search

Return: A_{opt}: Posture with lowest Euclidean distance relative to attraction point

---

minDistance = Infinity

isNewCandidatePostureFound = true

// List of (initial) postures inputted to the IK that have already been explored

ListOfInitialPostureJointPreviouslyExplored = {} // Initially empty

---

```
// List of (resulting) postures outputted by the IK that have already been explored
ListOfResultingPostureJointPreviouslyExplored = {} // Initially empty
1. While isNewCandidatePostureFound:
     isNewCandidatePostureFound = false
    // Find the step of each joint based on constraints
     For each joint j in Ap:
     StepAngle[j] = (j.maxAngle-j.minAngle)/Resolution
    // Get the list of postures in the neighbourhood
    jointStart = 1
    ListOfPostureJointToExplore = getNeighboringPosture(Ap.getAngles(),jointStart, StepAngle);

    //Filter the initial posture condition for duplicate of the one previously explored
    For each posture A in ListOfPostureJointToExplore:
       If ListOfInitialPostureJointPreviouslyExplored.Contains(A):
          ListOfPostureJointToExplore.Remove(A)
       Else:
          ListOfInitialPostureJointPreviouslyExplored.Add(A)
    End For
    // Find the posture Aopt in ListOfPostureJointToExplore with minimal distance to Att:
    For each posture A in ListOfPostureJointToExplore:
        A' = A.CalculateNewPostureFromInverseKinematic(T)
```

```
        If not A' == null:  // IK successfully converge on a new posture solution
          If not ListOfResultingPostureJointPreviouslyExplored.Contains(A'):
            aDistance = = DistanceAttractionPointToArmJoints(A', A_tt);
            If aDistance < minDistance:
                minDistance = aDistance;
                A_opt = A';
                isNewCandidatePostureFound = true;
                aStatus = true; //At least one solution found
          ListOfResultingPostureJointPreviouslyExplored.Add(A')
      End For
    End While
  2. return A_opt
```

Searching for the minimal distance posture by exploring all possible postures with a specific resolution (i.e. number of distributed angles parsed per joint) is of complexity $O(n^a)$ in the best case for a 3 DOF planar robotic arm with the *n* value corresponding to the resolution of the search and *a* the number of joints.

On the other hand, using an exploration based on the locally optimal choice for each iteration (greedy algorithm), we can find the possible posture with minimal distance more rapidly ($O(n*a)$ in the best-case scenario), where *n* is the resolution of the search and *a* the number of joints. Furthermore, the inability of the Greedy algorithm in searching beyond the local optima is often considered a limitation in that it does not guarantee the result to be optimal over the entire space. For our purposes, however, this limitation is beneficial since it confines the search to the operational space region (see Figure 1b) where the current posture resides and where a

continuous transition from one posture to the other is possible without moving the end-effector. Such posture change constraint is required when considering applications where the end-effector must follow a predefined trajectory. In such cases, our proposed solution can make posture changes without deviating the end-effector from its predefined trajectory.

4.4 Teleoperation with posture control with one hand – Tracing

In this section, a third approach is proposed to control the robotic arm posture. The method enables the user to input the desired posture by tracing its form using the user's index fingertip as a virtual pencil. This solution also includes the capacity to move the EE toward a target position and open/close the gripper. Like the "Browse and Select," this approach requires the user to pause the EE movement while they input a particular desired posture (this time using the tracing method). Additionally, only the EE position is controlled between these posture changes. Multiple modes are available through a natural gesture control menu interface that enables the recognition of specific gestures based on the position of the user in the menu. As illustrated in Figure 24, the different reserved gestures needed to navigate the different modes are "Trace new posture," "Trace Done," "Redo Trace," "End-effector displacement," and "Gripper opening."

Once the user completes the tracing of the desired posture with his index fingertip, a distributed gathering of the initial postures is generated for the IK algorithm with the aim of reaching all distinct operational spaces (if multiples exist). At this stage, a low-resolution distributed exploration is sufficient to cover most of the workspace while minimizing the computation cost. For each converging initial posture inputted to the IK algorithm during the low resolution search, a higher resolution Greedy Best First search is performed in its surrounding posture area as described in Section 4.3 (Figure 23). To evaluate the next best initial posture based on their IK output posture, we need to calculate the similarity of each possible output posture relative to the user's trace described in Section 4.1 (i.e. the trace drawn by the user with the red dot pencil and illustrated as a bold yellow line in Figure 15l).

The posture similarity is evaluated using DWT in terms of distance using Algorithm 10. Each Greedy Best First search performed returns the posture found to have the best similarity to the user's trace in their local area (i.e. with a minimum distance value). From those resulting postures, the posture with the minimum distance becomes the suggested posture solution for the user's trace. If no optimum solution is found, then no posture best match suggestion is returned to the user.

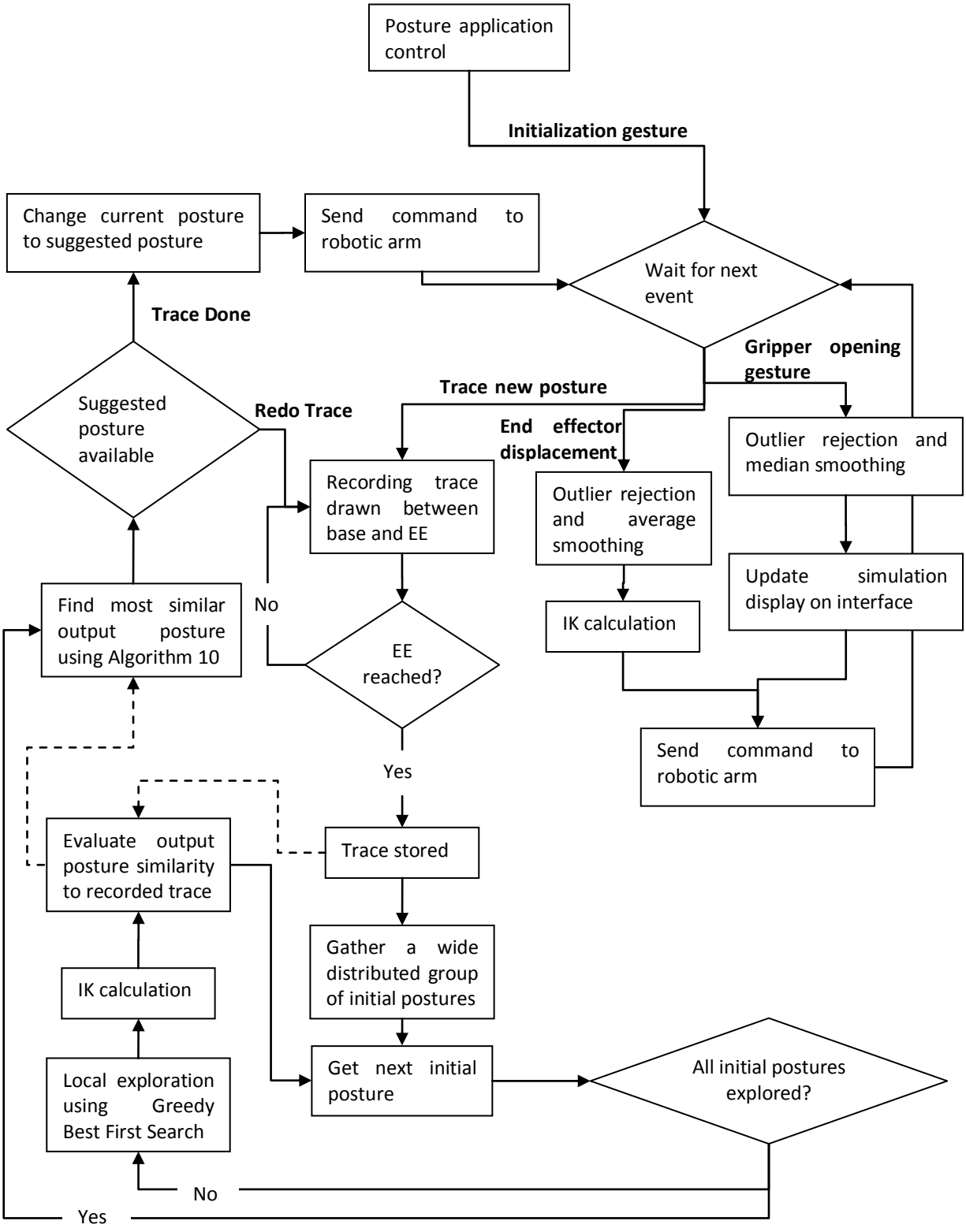


Figure 24 The Tracing posture framework approach.

Algorithm 10 evaluates the similarity of the user's trace relative to IK output posture using the Greedy Best First search on each posture obtained from the wide low-resolution search. The user's trace is most likely to contain a different amount of points compared to a possible gesture. Hence to compare them against each other, the DTW algorithm approach was selected because of its capability to compare two sets of values even if those two sets are of different size, as explained in more detail in Section 3.4. To evaluate in real-time the similarity between a candidate and a desired posture, we develop Algorithm 10 by taking and modifying the disparity evaluation in Algorithm 4, originally used to evaluate the similarity between a candidate and pre-recorded gesture in the hand detection and gesture recognition system (see Section 3.4). The detailed description of Algorithm 10 can be found in Section 3.4. In Algorithm 10, we are basically comparing postures defined by sets of joint positions, while in Algorithm 4, we are comparing gestures defined by sets of fingertips and hand palm center positions. The only other differences between the algorithms are the DTW parameters and their values. In Algorithm 10, the DTW parameters are influenced by the robotic arm configuration, while in Algorithm 4, they are influenced by the recorded gesture configuration.

Algorithm 10 : Similarity evaluation between two postures using DTW

Input: $\mathbf{D_p}$ = Desired posture

$\mathbf{A_c}$ = Posture candidate

Output: **P** = Similarity distance between D_p and $\mathbf{A_c}$

Return: aStatus = Return **true** if posture candidate distance value

can be considered a match with D_p or **false** if rejected

```
aPathCost = Infinity

  // Get the 2D DTW matrix containing the disparity between each possible posture joint and
 // desired posture trace point
dtwMatrix = FindDtwMatrix (Dp, Ac)

 // Use a Greedy algorithm to find the minimum path cost in disparities
 // in the DTW matrix; return -1 if the cost between a joint and a point is 0
 // or the horizontalMovementCounter > HorizontalMovementThreshold
 // or the verticalMovementCounter > VerticalMovementThreshold
pathCost = FindLowestCostPath(dtwMatrix)
 // Calculate the average path cost per desired posture point
averagePathcostPerPoint= pathCost / Dp.Count
   If (averagePathcostPerPoint < 60 and pathCost != -1):
       return true
   else
      return false;
   end
```

The gripper opening is controlled by the measure of the distance between the user's index and middle fingertip. This control over the gripper feature does not impact the control over the posture portion of the solution and details of the Median Smoothing can be found in Appendix A (Algorithm 11). This gripper feature was added to give an idea of the interesting possibilities that can be integrated to the posture control system. For the EE displacement, Algorithm 5 and Algorithm 6 are used, like the previous proposed method, to give a smoother control without outlier data coming from the fingertip position detection system detailed in Algorithm 3.

The parameters for the Dynamic Time Warping algorithm applied to posture similarity evaluation are presented in TABLE II. Compared to the DTW parameters used for gesture similarity evaluation in TABLE I, only the following parameters are used in the context of posture similarity evaluation: Number of horizontal and vertical movement threshold, Maximum cost path accepted and directional weight movement. Furthermore, the values of theses parameters depend on both the number of points in the user's trace (*m)* and the number of joints in the robotic arm configuration (*n*). With the desired posture (i.e. the user's trace) on the horizontal side and the possible candidate posture on the vertical side of the DTW matrix illustrated in Figure 25, the number of allowed vertical and horizontal movements depends on the values of *n* and *m*. If $m \geq n$, we can expect to have at most *m* and at least *m-n* number of consecutive horizontal movements. Using a threshold value smaller than *m-n* would always result in a no match solution. The same can be said with $n \leq m$ and the vertical threshold value. In the case of $m \geq n$, which happens most of the time, we use a horizontal threshold value between *m* and *m-n*, i.e. the $(m-n)+((m-(m-n))/2) = m-n/2$. For vertical movement, we use a threshold of $n/2$. The threshold values cannot be fractioned; hence the upper bound integer is applied. With these horizontal and vertical threshold values, we expect to reject early the highest half of the possible candidate posture with a possible minimum cost path. This gives a good balance between the frequency of "no match found" results (which can be annoying since we need to redo the trace) and match accuracy. The maximum Euclidean distance (path cost) should be dependent on the value of MAX(*n*,*m*), but it was harder to gauge; therefore we simply tried different values before settling with MAX(*n*,*m*)*7 which does not give too many

"no match found" results. The directional weights on the movement used for DTW on gestures were also applied for DTW on postures to similarly favor diagonal movement.

TABLE II Parameters for the Dynamic Time Warping algorithm applied to posture similarity evaluation.

Parameter	Value
The number of vertical or horizontal movements allowed in the shortest path (HorizontalMovementThreshold, VerticalMovementThreshold)	upper bound (Max(*m,n)*-Min(m,n)/2)
Maximum cost accepted for a path in the DTW matrix per tracing point (PathCostThreshold)	MAX(*n*,*m*)*7
Weight used for direction choices (horizontal, vertical, diagonal) during the search of the shortest path	(0.005, 0.005, 0.003)

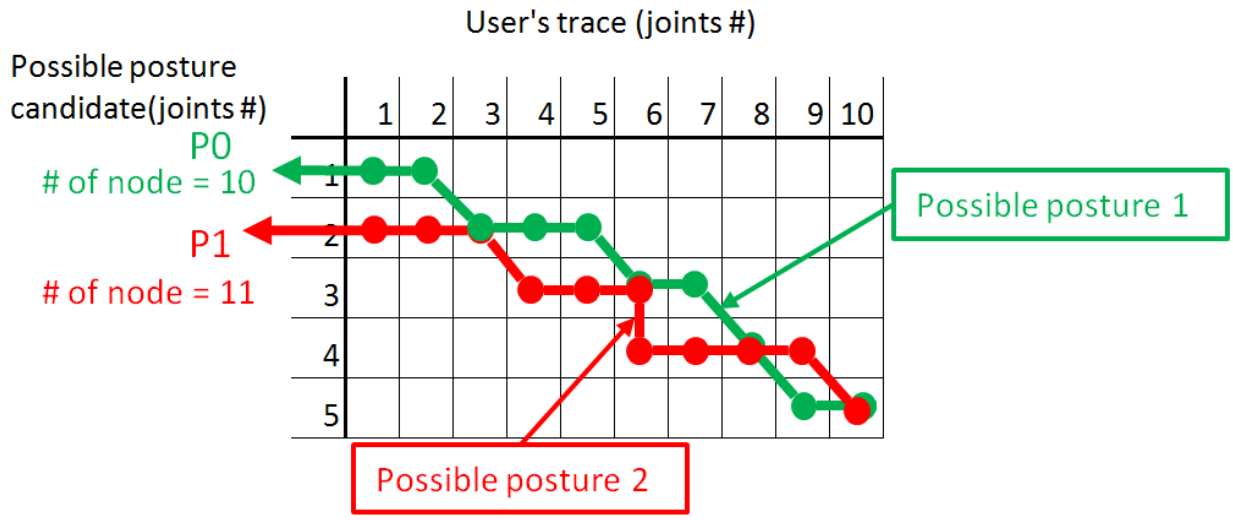


Figure 25 Example of a minimum cost path evaluation in the DWT matrix between the user's trace (m=10) and a possible gesture candidate(n=5).

Chapter 5 Experiments

In this chapter, we start by giving a detailed description of the test environment: the mechanical arm configuration, servo arm controller hardware and the software interface. This is followed by a description of the IK algorithms involved and an investigation on the best set of initial IK input postures for an efficient and rapid way of covering the space of output postures through IK exploration. Similarly, another investigation is performed to find the best set of initial IK input postures to efficiently and rapidly find a matching solution to a desired posture using Greedy Best First search. In both investigations, a comparison between the IK algorithm FABRIK and Pseudo Inverse Jacobian is conducted. Then, in sections 5.4 and 5.5, using the results of the investigations to optimally configure our posture exploration algorithms, we compare our three methods for posture control (Browse and Select, Attraction Point and Tracing) with a no-posture-control approach in their ability to perform a simple task inside two types of constrained simulated environments: static obstacles and dynamic obstacles.

Take note that a reduced version of the Lynxmotion AL5D robotic arm configuration, illustrated in Figure 27, is used for both the investigation and the three posture control comparison experiments. The base joint (J1) being the only joint to move on the ground plane of the AL5D, it is thus completely defined by the end-effector position, without the need to be resolved through a 3D IK exploration. It would have been redundant and unnecessarily complex otherwise. Since the point of the experiments is not to test the functionality of the 3D IK algorithms themselves but rather to improve the IK exploration technique and conduct an experimental evaluation of the different posture control approaches, the base joint is not needed. As a result, the 3-link planar robotic arm, represented by the set of joints J2, J3 and J4 in the AL5D configuration was used in the experiments mentioned above. Technically, this simplification should not affect the scope of our investigation since the best IK algorithm (FABRIK) also supports 3D configurations (and more) using the same logic. Moreover, this simplification allows us to avoid very complex calculation with the Pseudo Inverse Jacobian which is not expected to scale well to more complex configurations for our proposed

approaches based on expected high computer cost. As for comparing the different posture control approaches through experiments, using, a 2D configuration should be enough to prove the viability of these approaches. Based on the description of each posture control approaches, extending them to more complex 3D configurations is simply a matter of integrating a 3D IK algorithm (3D FABRIK [42]) to our solutions and use the third dimension information that is already sent by the hand and gesture recognition system. This information is in fact already processed by our posture control system: 3D position of the joints and EE and 3D direction vector of the index finger in the Attraction point method. As a proof, we performed a 3D posture control operation in the following section.

In Section 5.6, we present a 3D real-world posture control demonstration of a task performed using each posture control methods in a static obstacle environment. In this demonstration, the complete real AL5D robotic arm is used. As stated above, the base joint rotates along a different plane relative to the other joints (shoulder, elbow and wrist) in the AL5D configuration. Hence, for the 3D real-world experiment, the IK exploration can define the angles of the shoulder, elbow and wrist joints and rotate the base joint in the direction of the target position to allow the EE to reach the target in a three dimensional space.

The choice of testing in a simulation environment for most of our experiments (coverage strategy investigations, static and dynamic obstacles comparative experiments) is based on the need for extensive testing using similar conditions. For example, the state of the servos (temperature, wear) or a structural change on the robot during the course of the tests could have influenced our results. Moreover, an undesired behavior caused by an implementation error or a bad design while controlling the robotic arm is far more forgivable in simulation than in a real world experiment. Therefore, the real world experiment was performed once the simulation experiments were conclusive on the reliability of the system.

5.1 Test environment

This section provides a description of the hardware and software used for the experiments. In Figure 26, the main hardware components schematic is presented. Each component is further detailed in this section.

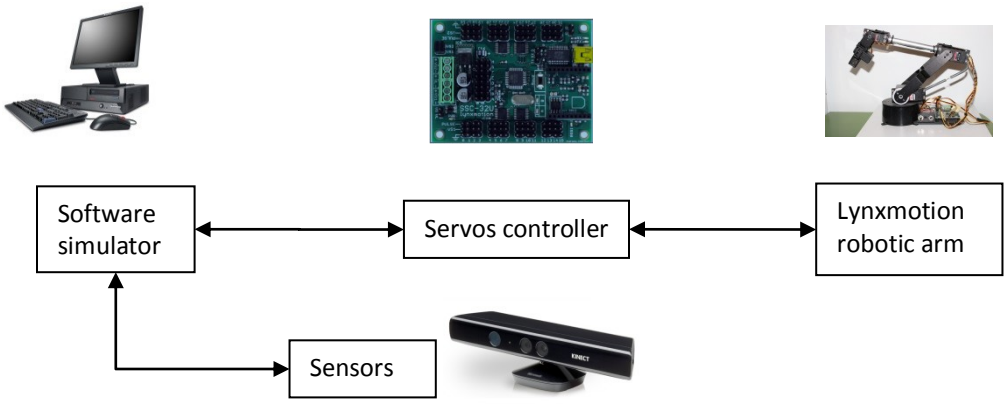


Figure 26 Hardware environment schematic overview.

5.1.1 Lynxmotion robotic arm and servos controller

For all experiments, we use the Lynxmotion robotic arm model AL5D [87]. It is an affordable and robust enough device for developing prototypes for industrial applications. As illustrated in Figure 27 and Figure 29, it includes 4 revolute joints (4 DOF): a ground plane base rotation joint, and single plane rotation shoulder, elbow and wrist joints. It also includes a gripper as the end-effector. The servos are of R/C analog type. They can be controlled using pulse width modulation. Starting with 0V, the servos expect to receive a 5V signal at each 20 ms interval. When received, the servos count the time until the signal goes back from 5V to 0V. This elapsed time gives the specific revolute position. For example, to center a joint servo at 0^{o}, a 1500µs pulse needs to be sent. Instead of developing our own command protocol to programmatically generate these pulses using an Arduino (Botboarduino), we used a Lynxmotion SSC-32U R/C servo controller (see Figure 28). The latter already comes with an embedded program with bidirectional communication based on query commands that allows synchronized group motion with a resolution of 1µs. Command queries are generated based on the user gesture recognized by our application and located on the PC controller. These queries are send to the

SSC-32U, which in turn converts them to pulses to the servos. Various data can be retrieved back from the SSC-32U board such as the servos movement status (currently moving or idle) or the current angular position (pulse width).

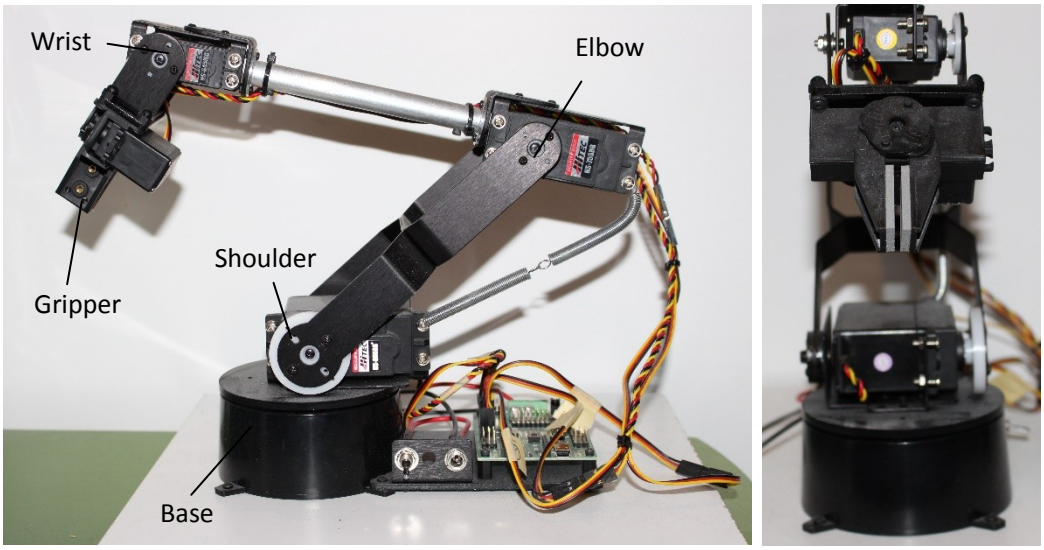


Figure 27 Lynxmotion AL5D robotic arm.

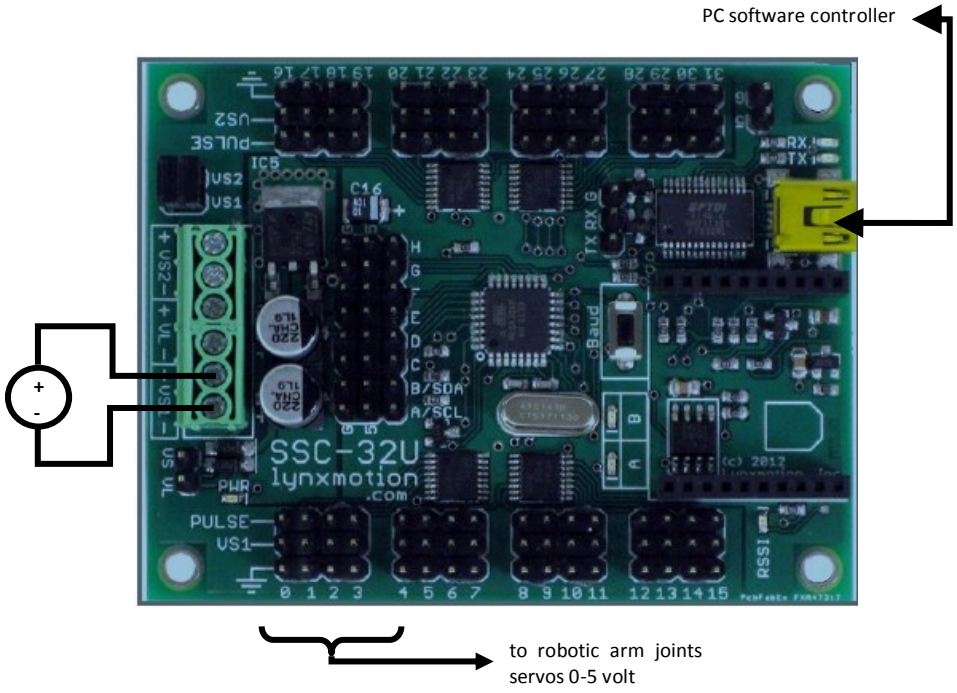


Figure 28 Lynxmotion SSC-32U USB servos Controller [87].

The following commands format [87] interpreted by the SSC-32U were used by our application to control the robotic arm:

- Single joint move command:

<ch> P <pw> S <spd> T <time> <cr>

- <ch>: pin/channel to which the servo is connected (0 to 31) in decimals
- <pw>: desired pulse width (normally 500 to 2500) in microseconds
- <spd>: servo movement speed in microseconds per second
- <time>: time in microseconds to travel from the current position to the desired position.
- <cr>: carriage return (ASCII 13)

- Multiple joint move command:

<ch> P <pw> S <spd> T <time> # <ch> P <pw> S <spd> T <time> <cr> ...

Single joint movement status:

Q <cr>

To generate the intended joints revolute command for our application, we first must be able to calculate the forward or inverse kinematics. To be able to perform those calculations, the kinematic configuration of the robotic arm needs to be analyzed. We use the well established Denavit-Hartenberg (DH) convention which enables a systematic configuration of all reference frames (RF). From the robotic arm schematic information illustrated in Figure 29, we apply the systematic DH approach of assigning RFs and identifying the transformation parameters (or DH parameters). The resulting parameters are presented in TABLE III where q1, q2, q3 and q4 represent the real servos angle that need to be sent to the robotic arm used in our experiments based on its configuration (this entirely depends on how each servos was installed on the robotic arm joints).

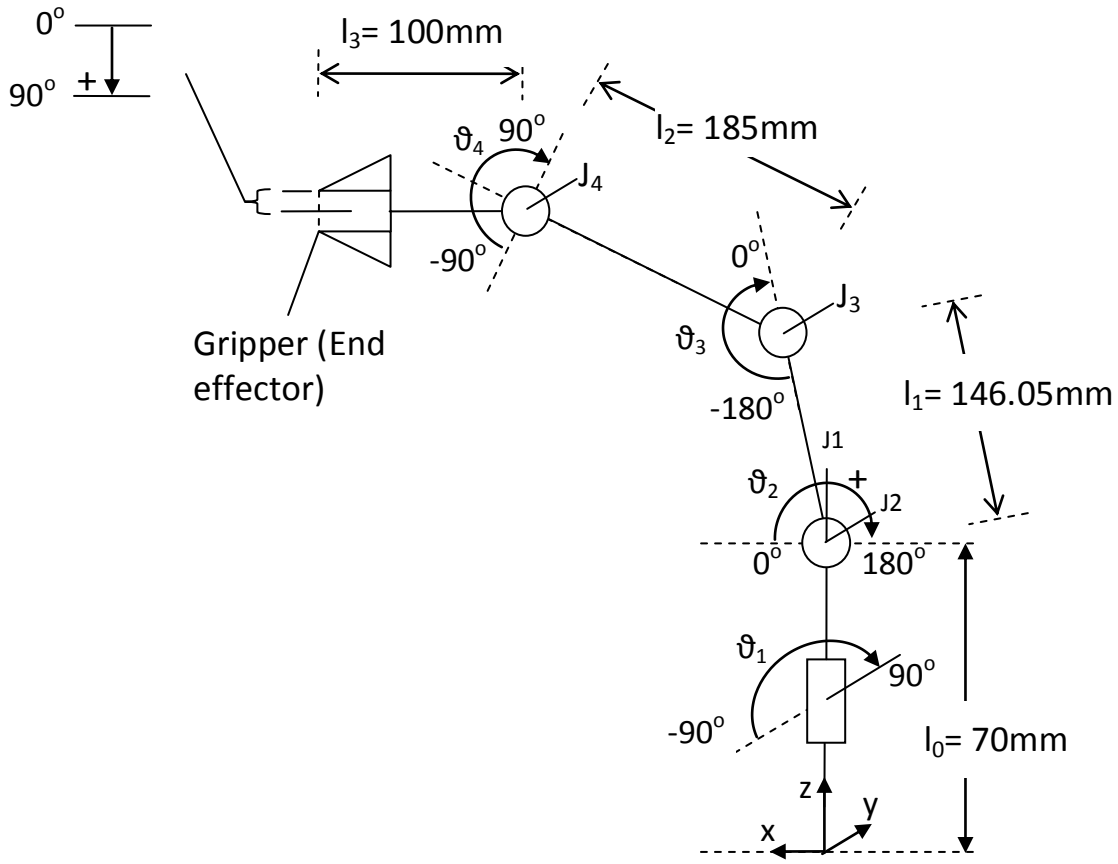


Figure 29 AL5D robotic arm description of the position and angle restriction of each joints.

TABLE III DH parameters for the AL5D, where 0≤qi≤180 for i=1,2,3,4.

i	ϑi (deg)	di (mm)	ai (mm)	αi (deg)
1	q1-90°	70	0	90°
2	q2	0	146.05	0
3	q3	0	185	0
4	q4-90°	0	100	0

5.1.2 Kinect sensor and simulation software

The sensor camera used in our experiments is the Microsoft Kinect v1.0 with the Kinect SDK 1.8. The workstation on which resides the simulation software used to extract and process the data coming from the Kinect sensor is a Hp Z400 with Quad-Core Xeon 3.06Ghz CPU and 16 GB RAM.

The software which includes the modules for the graphic interface simulation, gesture recognition [3] and robotic posture control was implemented in C# using the WPF graphical subsystem with Microsoft Visual Studio 2015. The application was developed and used exclusively on Windows 7.

As part of this work, we developed a simulation software that extends the software we developed initially for the gesture recognition system in [3]. This extension is capable of displaying a simulation of the robotic arm subset of joints subject to posture control, i.e. the 3 DOF planar arm represented by J2, J3 and J4 in Figure 29, for the three proposed posture control applications. The simulation software interface includes a button for initializing the connection to a real mechanical robotic arm. If not triggered, the system defaults to an offline simulation session. During a session, the user can conveniently switch between posture control, choose different test environments and pre-record gestures to be added as reserved gestures in the gesture recognition system to navigate between modes during a session. It also gives an indication of a recognized user's gesture when moving through the menus and modes of the application. The software simulator enables the user to see the different possible postures available in the "Browse and Select" method. It is also a necessary tool for the "Tracing" posture control method that displays the position of the pencil dot (in red) to allow the user to draw the desired posture. Additionally, the simulation software was particularly useful during the development and testing phases of this thesis.

To start sending one of the commands described in Section 5.1.1 (Single joint move, multiple joint move, etc.) to the robotic arm in the real world, the user needs to connect the SSC-32U to the PC controller and press the “Connect arm" button located on the left side of the software interface below the graphic display, as shown in Figure 30.

To record the static gestures used to navigate between mode and posture control in our application, the user can enter the name of the gesture in the field above the “Record Gesture” button in Figure 30. Pressing the Record Gesture button starts recording. The system records the following 40 gesture frames and then automatically stops the recording. The recording length was limited to 40 frames to allow enough time for dynamic gesture execution, differentiate static from dynamic gestures and minimize latency (Section 3.4). The “Stop Recording” button located below can interupt this process before completion.

Figure 30 presents the interface for the "Browse and Select" posture control. To enter this interface, the user first needs to select this method from the Control Method group and enter a value in the field located under the label "Maximum amount of alternative postures". The entered value indicates the amount of possible postures to display along with the current robotic arm posture in the graphic display. At this step, the system waits for an "initialization" gesture defined in Figure 12 from the user before starting to control the robotic arm with the selected method.

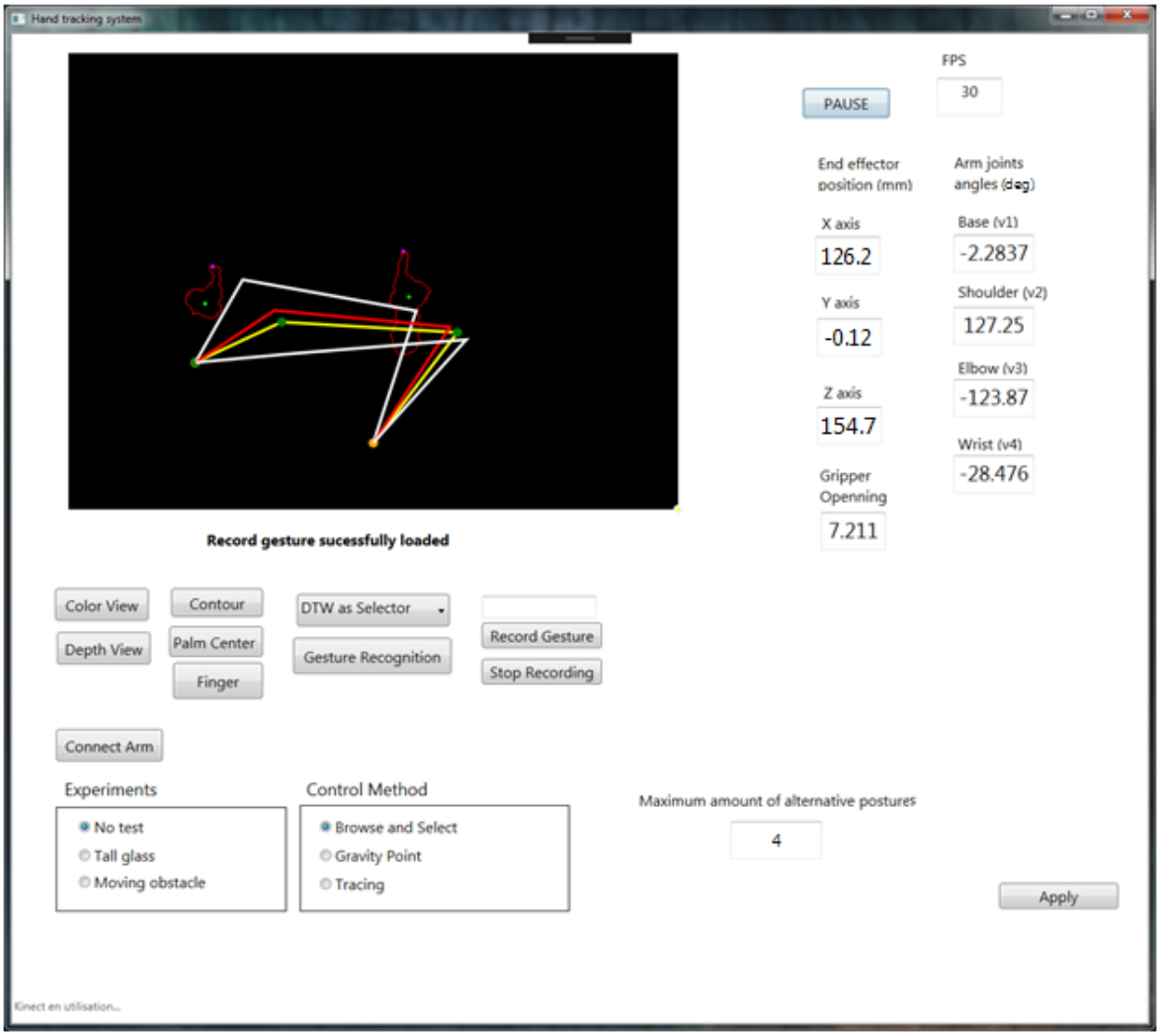


Figure 30 Software interface for the Browse and Select posture control application.

Figure 31 presents the interface for the "Attraction point" posture control which can be initiated by checking the radio button labeled "Gravity point" in "Posture control" box. To enter this interface, the user first needs to select this method from the Control Method group and enter a value in the field located under the label "Posture Exploration Resolution" and next to "High". The entered value indicates the resolution of the Local exploration using Greedy Best First Search detailed in Section 4.3. The "Attraction Point direction" parameter is defined by the angle between vector O_A-EE and vector O_A-Attraction point, i.e. the angle θ in Figure 31. The best posture is selected based on the formula presented in Algorithm 8. At this step, the system waits for an "initialization" gesture from the user before starting to control the robotic arm with the selected method.

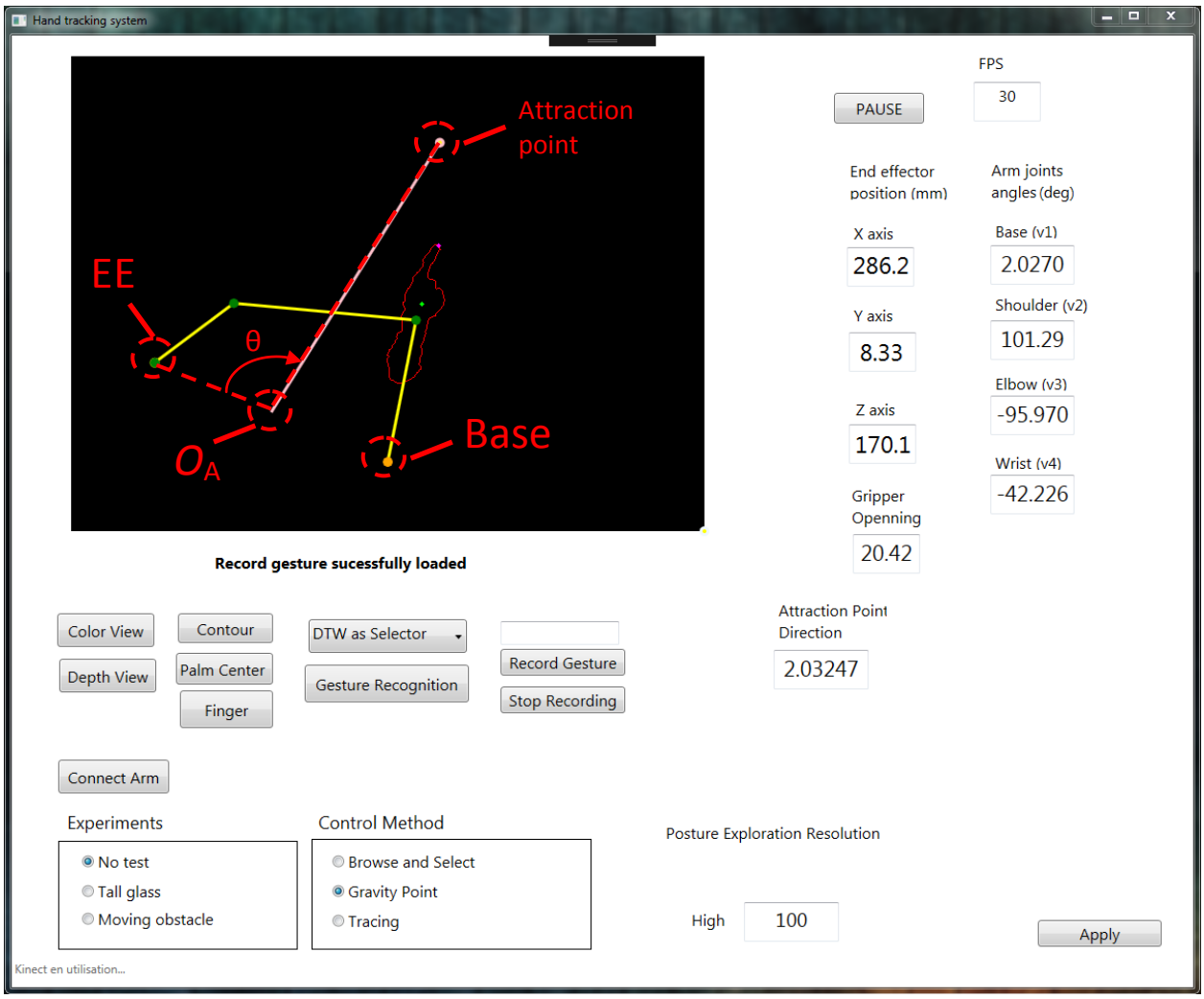


Figure 31 Software interface for the Attraction Point posture control application.

Figure 32 presents the interface for the "Tracing" posture control. To enter this interface, the user first needs to select this method from the Control Method group and enter values in the fields located under the label "Posture Exploration Resolution" and respectively next to "Low" and "High". The values indicate respectively the resolution of search when gathering a wide distributed group of initial postures and the resolution used for the local exploration (greedy best-first search) surrounding each initial posture. From these extensive explorations, we can find, through Algorithm 10, the candidate posture with the minimum similarity distance. At this step, the system waits for an "initialization" gesture from the user before starting to control the robotic arm with the selected method.

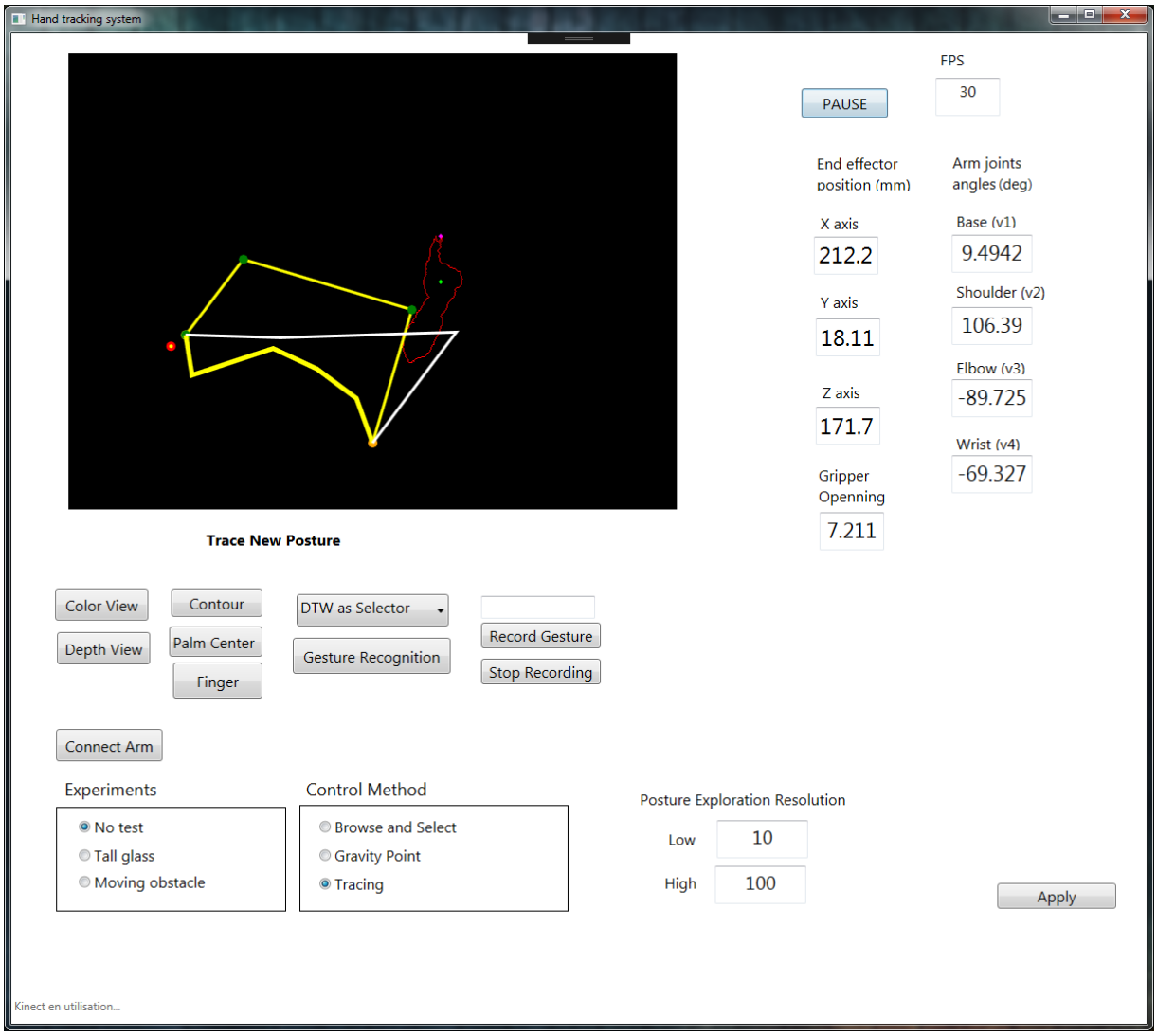


Figure 32 Software interface for the Tracing posture control application.

The software interface gives the user the option of performing two distinct experiments: "Tall glass" and "Moving obstacle". Both options can be selected at any time during the session to start. Take note that the following additional fields are used specifically for the "Moving obstacle" experiment, as indicated in Figure 33:

1. Hit obstacle count: Number of attempts where the robotic arm encounters the moving obstacle before reaching recipient B with the end-effector. In Figure 33, an example of an encounter between the robotic arm and the moving obstacle is given resulting in a

obstacle count value incrementing from 0 to 1 during the fifth attempt to move the EE from recipient A to recipient B.

2. Test is running: Display "True" when the end-effector touches recipient A, indicating that an attempt to move the EE from recipient A to B is in progress. The state changes to "False" when the end-effector touches recipient B or when any part of the robotic arm encounters the obstacle as illustrated in Figure 33.
3. Attempt count: Number of cumulated attempts to move the EE from recipient A to B.

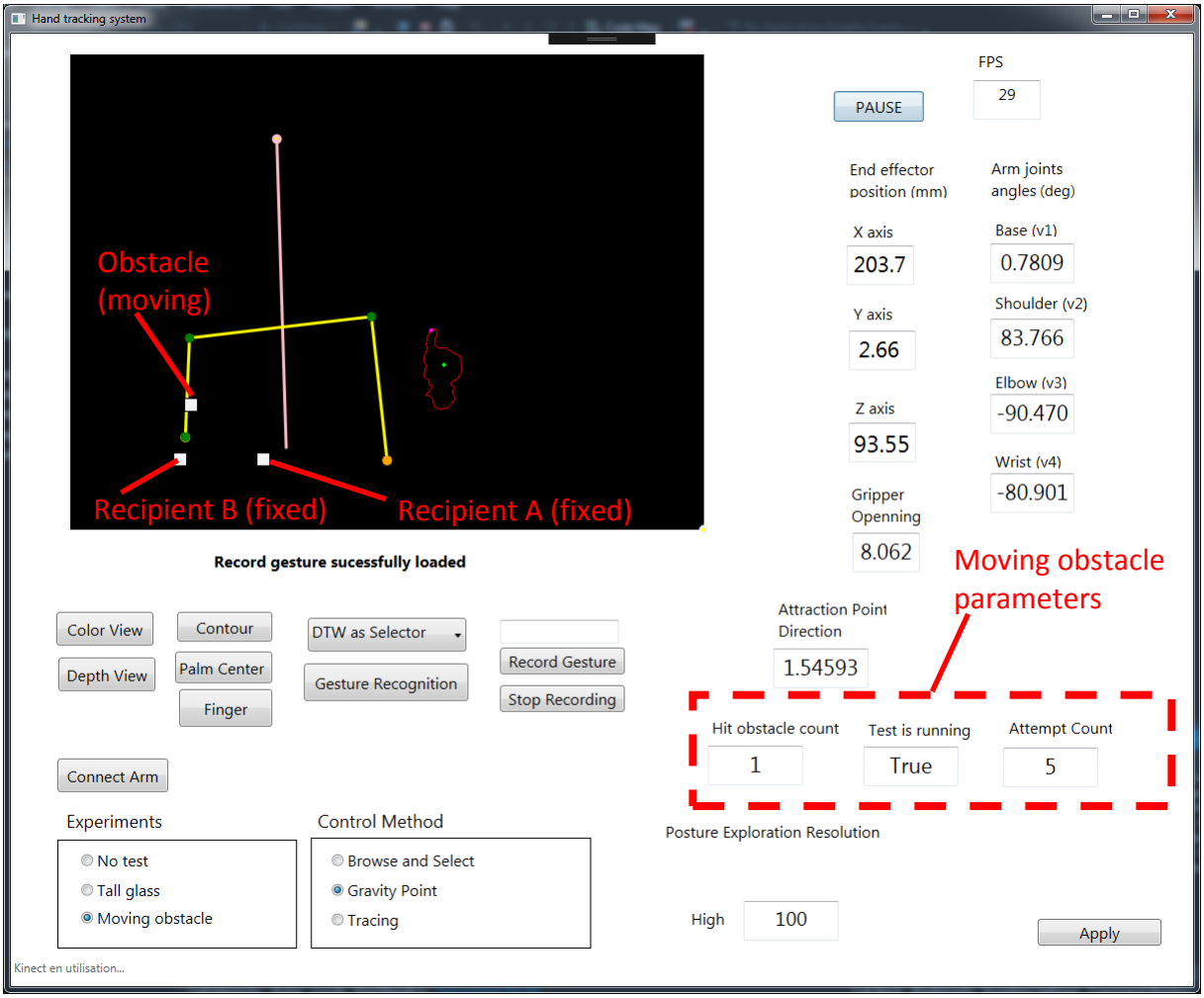


Figure 33 Case of a encounter between the obstacle and the robotic arm during the "Moving obstacle" experiment.

5.2 Exploration coverage

In this section we investigate a way to optimize the IK output coverage to rapidly gather the maximum number of possible postures able to reach a new end-effector position, thus leading to a better chance of finding the best matching posture for the attraction point position or user trace, as explained earlier in Chapter 4. Also, as explained earlier in section 1.3, a good coverage of the inputted initial posture does not guarantee a good coverage of the outputted posture. In this section we examine different initial input coverage and IK algorithms and attempt to determine the right combination to maximize both the speed of execution and IK output coverage. By maximizing the output coverage of possible postures able to reach a certain new end-effector position we consequently increase our chances of closely reaching the optimal desired posture from a new attraction point position or user trace. There is no easy or perfect technique for determining the coverage of a dataset. Some datasets can spread irregularly over a greater area while others can have higher and more regular density over a smaller area. For our research, we decided to simply use the number of distinct postures resulting from the IK calculations explored to quantify the level of coverage. Two neighbouring postures are considered distinct if their distance is greater than a given threshold value.

5.2.1 Multiple closed operational space environment

The term "closed" here refers to the set theory where every point in the set can only be reached by starting from another point contained in the set. In the current context, a point would correspond to a robotic arm posture. To create an environment with multiple closed operational spaces for the purpose of our experiments, the robotic arm configuration needs to be modified. This allows our investigation to include a certain complexity and difficulty in terms of environments.

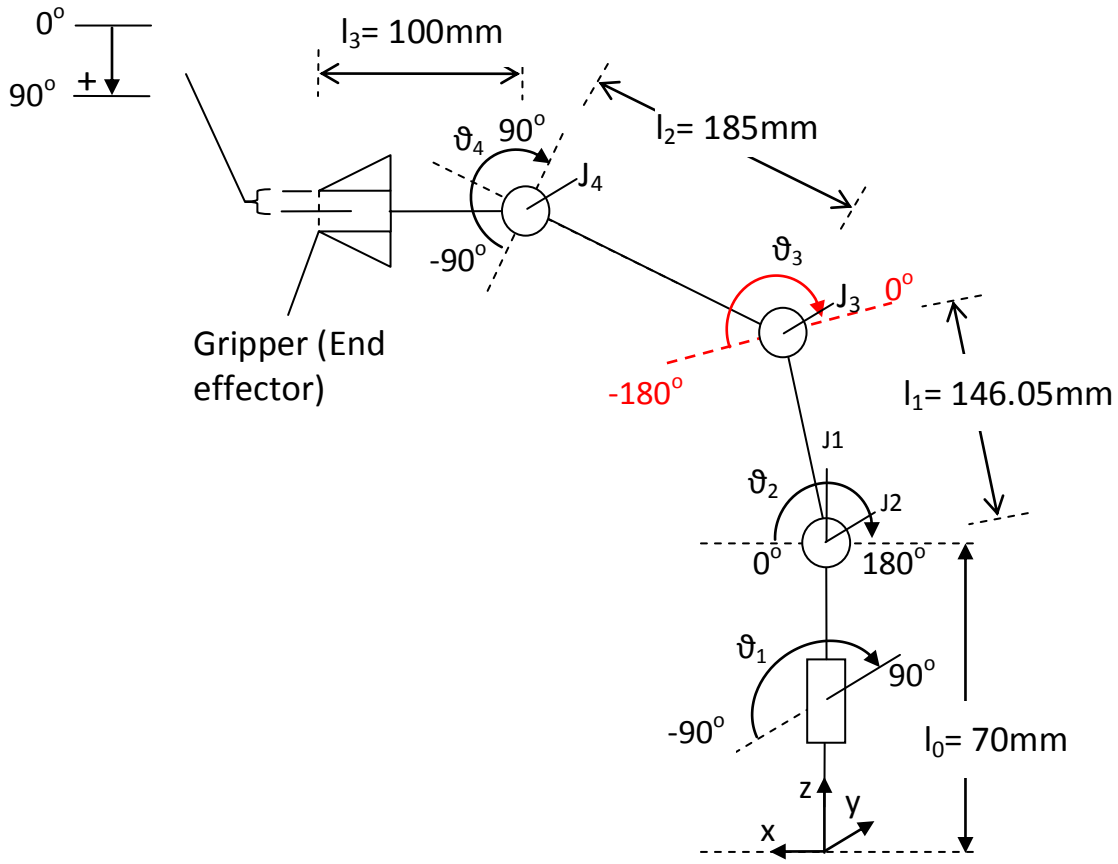


Figure 34 Reconfiguration of the robotic arm joints restriction for the purpose of the posture exploration coverage investigation.

By slightly modifying the robotic arm configuration initially presented in Figure 29, we were able to produce a workspace where two distinct operational spaces can exist for the same end-effector position. The modification consists of inserting an offset of 90 deg on the rotation range of joint ϑ_3 (elbow) as illustrated in Figure 34 and TABLE IV. Now, by strategically placing the target end-effector on the positive z axis, two close sets of possible postures can now exist on both sides of the x axis as depicted in Figure 1b and Figure 35. In this experiment, take note that when the tracing method is performed, the user traces the desired posture with a target end-effector (red dot) at position (x=0, z=350).

TABLE IV DH parameters for the reconfigured simulated AL5D.

i	ϑi (deg)	di (mm)	ai (mm)	αi (deg)
1	q1-90o	70	0	90o
2	q2	0	146.05	0
3	q3+90o	0	185	0
4	q4-90o	0	100	0

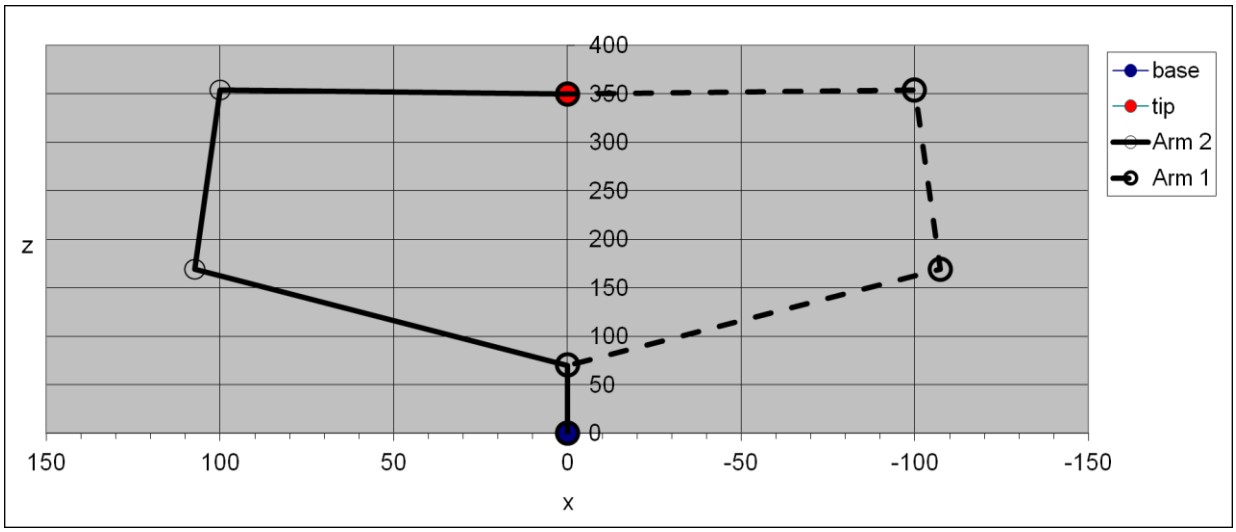


Figure 35 End-effector (tip) position where the modified robotic arm configuration can converge toward two posture solutions belonging to two distinct operational spaces each respectively represented by Arm 1 and Arm 2.

5.2.2 IK algorithms

In this section we describe the characteristics of two different IK algorithms that are used in the IK exploration investigation presented in the following sections: FABRIK and Pseudo Inverse Jacobian. We choose those two particular algorithms because they are one of the fastest to resolve the IK [41]. Out of the two, the best IK algorithm for our solution can be determined by comparing their results in the experiments sections. The results of this experimental comparison are presented in section 5.2.6.

The case of the 3 link-planar arm is detailed in this section since an IK exploration on this configuration is performed in the following experiments which is later used in the 3D posture control demonstration of the AL5D robotic arm.

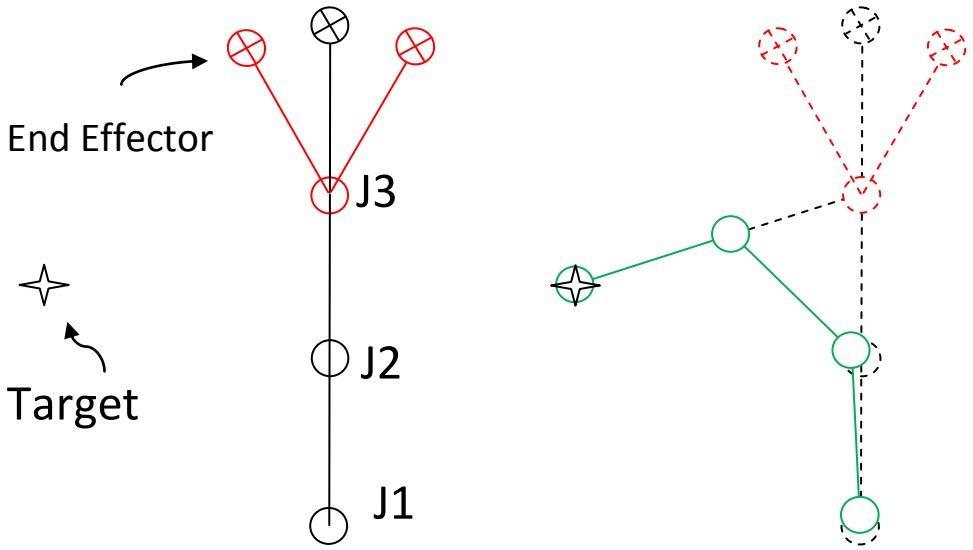
5.2.2.1 Characteristics of the FABRIK algorithm

The detail of the FABRIK IK algorithm was presented back in Section 2.3 based on [39] and [40]. Like most other IK algorithms found in the literature, the solutions (or posture outputs) are influenced by the initial arm revolute joint angle values given as input. However, for this particular IK algorithm, not every joint have the same influence over the output coverage.

Let's consider, as a simple example, a rope on a piece of paper as an n-link planar arm with its base joint fixed (or glued to the paper), the inputted posture as the current rope posture and the target EE position as an "X" mark drawn somewhere inside the reachable space of the rope's EE. Applied to the FABRIK algorithm, the same behavior is observed as someone taking the rope's EE and, following a straight trajectory over the paper plane, moves it to the "X" mark. The smaller the EE displacement and the closer the target EE position is relative to the base, the less likely a particular joint i needs to adjust its angle as i is close to the base joint (i=0). Hence, more often than not, the joints close to the base are not very or only slightly affected. This means that two initial rope postures given any EE displacement will result in greater number of distinct postures the closer to the base their joint angles differ. Consequently, for two postures A and B among the set of input initial postures, they are less likely to converge to similar postures as they possess a common joint i with different angle values close to the base joint i =0. For an n-link three dimensional robotic arm, the theory remains the same. Indeed, the FABRIK algorithm supports 3D configuration with orientation constraint and most common joint type, including the six most common anthropometric joints [39]: Pivot, ball-and-socket, hinge, condyloid, saddle and plane joint.

Taking the simple example of a 3 link planar arm, this means that the last joint before the end-effector would have little to no influence on the resulting posture solutions coming out of the FABRIK. Indeed, as illustrated in Figure 36a-b, a set of initial postures differing only by the angle value of the last joint (J3) always converge toward the same solution. Hence, to find a set of unique solutions more rapidly, it is best to fix the angle value of J3 and parse the angle values of the next joint (J2) when gathering the initial postures set. However, some initial postures may output the same posture solution. To minimize this problem, as stated earlier, we need to find

the joint generating the highest amount of distinct solution per angle step, which is the base joint. Figure 37a-b presents such an example for the 3-link planar arm with three initial postures differing only by their J1 joint angle. For each initial posture, the FABRIK algorithm converges toward a successful distinct posture solution.



(a) (b)

Figure 36 (a) Exploring a possible solution to reach a target by starting with different angles of the joint J3, (b) after one forward iteration, only one solution (in green) was found for all J3 potential angles.

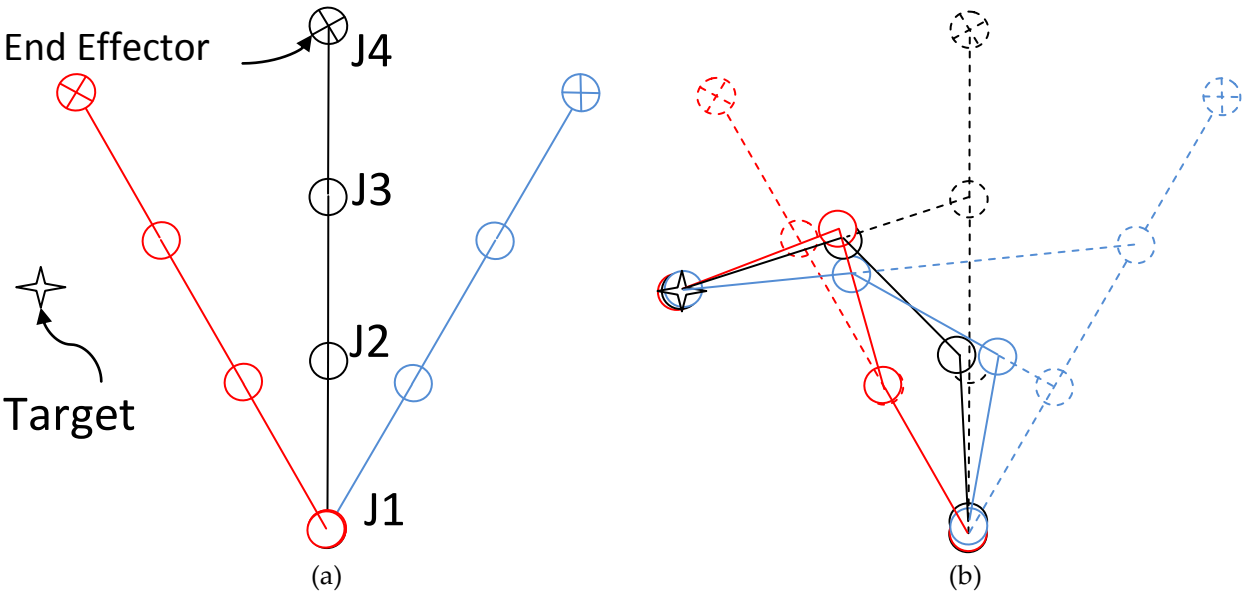


Figure 37 (a) Exploring a possible solution to reach a target by starting with different angles of the base joint J1, (b) after one forward iteration, a different solution (in solid line) was found for each J1 potential angle (one per color).

For the experiment that follows we implemented a simple 2D version of the FABRIK algorithm since it was enough in the IK exploration needed to control the 3D posture of our 4 DOF robotic arm and easier to integrate to our system than a third party library. Take note however that complete 3D FABRIK algorithms, such as CALIKO [42], are available online.

5.2.2.2 Characteristics of the Pseudo Inverse Jacobian algorithm

When using the Pseudo Inverse Jacobian, calculations become more complex compared to FABRIK and need to be tailored, especially for the Jacobian matrix, specifically for a robotic arm configuration. As explains earlier, we are studying the case of the 3-link planar arm. Based on [89] for the 3–link planar arm, using the Denavit-Hartenberg convention, the steps leading to the Jacobian matrix are given as follows:

From eq. (4) representing the homogeneous equation defining the coordinate transformation for each joint of the 3–link planar arm portion of the robotic arm in Figure 34(i.e. J2, J3 and J4) and where c_i corresponds to cos ϑ_i, s_i corresponds to sin ϑ_i, a_i corresponds to the length of the link between joint i and i+1, and ϑ_i correspond to joint i angle between link i-1 and link i,

$$A_i^{i-1}(\vartheta_i) = \begin{bmatrix} c_i & -s_i & 0 & a_i c_i \\ s_i & c_i & 0 & a_i s_i \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \text{ i =2,3,4} \quad \textbf{(4)}$$

we can obtain the direct Kinematic transformation function, Q, illustrated in eq. (5),

$$Q_4^1(\bar{\vartheta}) = A_2^1 A_3^2 A_4^3$$

$$Q_4^1(\bar{\vartheta}) = \begin{bmatrix} c_{234} & -s_{234} & 0 & a_2c_2 + a_3c_{23} + a_4c_{234} \\ s_{234} & c_{234} & 0 & a_2s_2 + a_3s_{23} + a_4s_{234} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(5)}$$

where we can find the rotation matrix, R, and translation vector, O, from eq. (7),

$$Q_4^1(\bar{\vartheta}) = \begin{bmatrix} {}^1R_4 & {}^1O_4 \\ 0 & 1 \end{bmatrix} \quad \textbf{(6)}$$

and where

$$s_{ij} \equiv \sin(\vartheta_i + \vartheta_j),\ s_{ijk} \equiv \sin(\vartheta_i + \vartheta_j + \vartheta_k),$$
$$c_{ij} \equiv \cos(\vartheta_i + \vartheta_j),\ c_{ijk} \equiv \cos(\vartheta_i + \vartheta_j + \vartheta_k).$$

From eq. (5) and eq. (6), the resulting translation vector O, eq. (7), can be represented by some real known function $f_1(\vartheta_2, \vartheta_3, \vartheta_4)$, $f_2(\vartheta_2, \vartheta_3, \vartheta_4)$ and $f_3(\vartheta_2, \vartheta_3, \vartheta_4)$.

$${}^1\bar{O}_4\ (\bar{\vartheta}) = \begin{bmatrix} a_2c_2 + a_3c_{23} + a_4c_{234} \\ a_2s_2 + a_3s_{23} + a_4s_{234} \\ 0 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \quad \textbf{(7)}$$

From these functions (f_1, f_2, f_3), the Jacobian matrix can be found in eq. (8), (9) and (10).

$$J = \frac{\partial\ {}^{1}\bar{O}_4}{\partial \bar{\vartheta}} \tag{8}$$

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial \vartheta_2} & \frac{\partial f_1}{\partial \vartheta_3} & \frac{\partial f_1}{\partial \vartheta_4} \\ \frac{\partial f_2}{\partial \vartheta_2} & \frac{\partial f_2}{\partial \vartheta_3} & \frac{\partial f_2}{\partial \vartheta_4} \\ \frac{\partial f_3}{\partial \vartheta_2} & \frac{\partial f_3}{\partial \vartheta_3} & \frac{\partial f_3}{\partial \vartheta_4} \end{bmatrix} \tag{9}$$

$$J = \begin{bmatrix} -a_2 s_2 - a_3 s_{23} - a_4 s_{234} & -a_3 s_{23} - a_4 s_{234} & -a_4 s_{234} \\ a_2 c_2 + a_3 c_{23} + a_4 c_{234} & a_3 c_{23} + a_4 c_{234} & a_4 c_{234} \\ 0 & 0 & 0 \end{bmatrix}. \tag{10}$$

The inverse position Kinematics can be solved numerically using the following algorithm:

1. Starting with a set of some random initial joint angles, $\bar{\vartheta}$, defining the posture of the robotic arm on the plane defined by the joints J2, J3 and J4:

$$\bar{\vartheta} = [\vartheta_2\ \vartheta_3\ \vartheta_4] \tag{11}$$

2. Iteratively update $\bar{v}$ with the following equation:

$$\bar{\vartheta}_{k+1} = \bar{\vartheta}_k + J_k^{+} \cdot (\ {}^{1}\bar{O}_4^{d} -\ {}^{1}\bar{O}_{4\ k}) \tag{12}$$

where $J^{+} \equiv (J^T J)^{-1} J^T$ (Pseudo inverse of J), $k \equiv Iteration\ \#$ and ${}^{1}\bar{O}_4^{d} \equiv Desired\ position.$ The iteration stops when the error tolerance $\bar{\vartheta}_{k+1} - \bar{\vartheta}_k$ is reached and the ${}^{1}\bar{O}_{4\ k}$ is defined from the end-effector position of $\bar{\vartheta}_k$. For our implementation, J^{+} is calculated using the *Math.Numerics* library [88].

As confirmed in the experiment that follows, the FABRIK algorithm gives faster and more accurate results overall compared to the Pseudo-Inverse Jacobian. Besides requiring more complex IK calculation compared to FABRIK, the output solutions of the Pseudo Jacobian are subject to singularities and cannot converge with a error tolerance of 0 like the FABRIK

algorithm. The Pseudo-inverse was only added in this experiment to give an element of comparison to the FABRIK algorithm in the IK exploration of the posture control calculation.

5.2.3 Improving the IK output coverage: Coverage by varying a single joint

To get a better idea of the influence of the initial inputted posture coverage computed by IK on the (resulting) outputted postures, we first set all joint angle values to a trivial number. We choose the middle angle of every joint, i.e. (ϑ2, ϑ3, ϑ4) = (1.57, 0, 0) rad. From this posture we created a set of 21 initial postures by varying only the ϑ2 joint angle. For this, we selected 21 equidistant ϑ2 joint angles (i.e. equally separated) going from its minimum to maximum value (black dots labelled 1 to 21 in Figure 38). The resulting possible postures outputted from the FABRIK (blue dots labelled F_i) and Pseudo Inverse Jacobian (Yellow dots labelled P_i) IK algorithms are showed in Figure 38.

From initial postures inputted to the FABRIK IK algorithm, we found 16 distinct converging solutions well spread over the two operational space, around ϑ_2 = 0.471 rad and ϑ_2 = 2.66 rad respectively. From the Pseudo Inverse Jacobian, we found that all the initial conditions eventually converged, but on only two possible postures: (0.471, 1.291, 1.052) rad and (2.66, -1.29, -1.052) rad. Hence, this shows in this case a better coverage using the FABRIK algorithm.

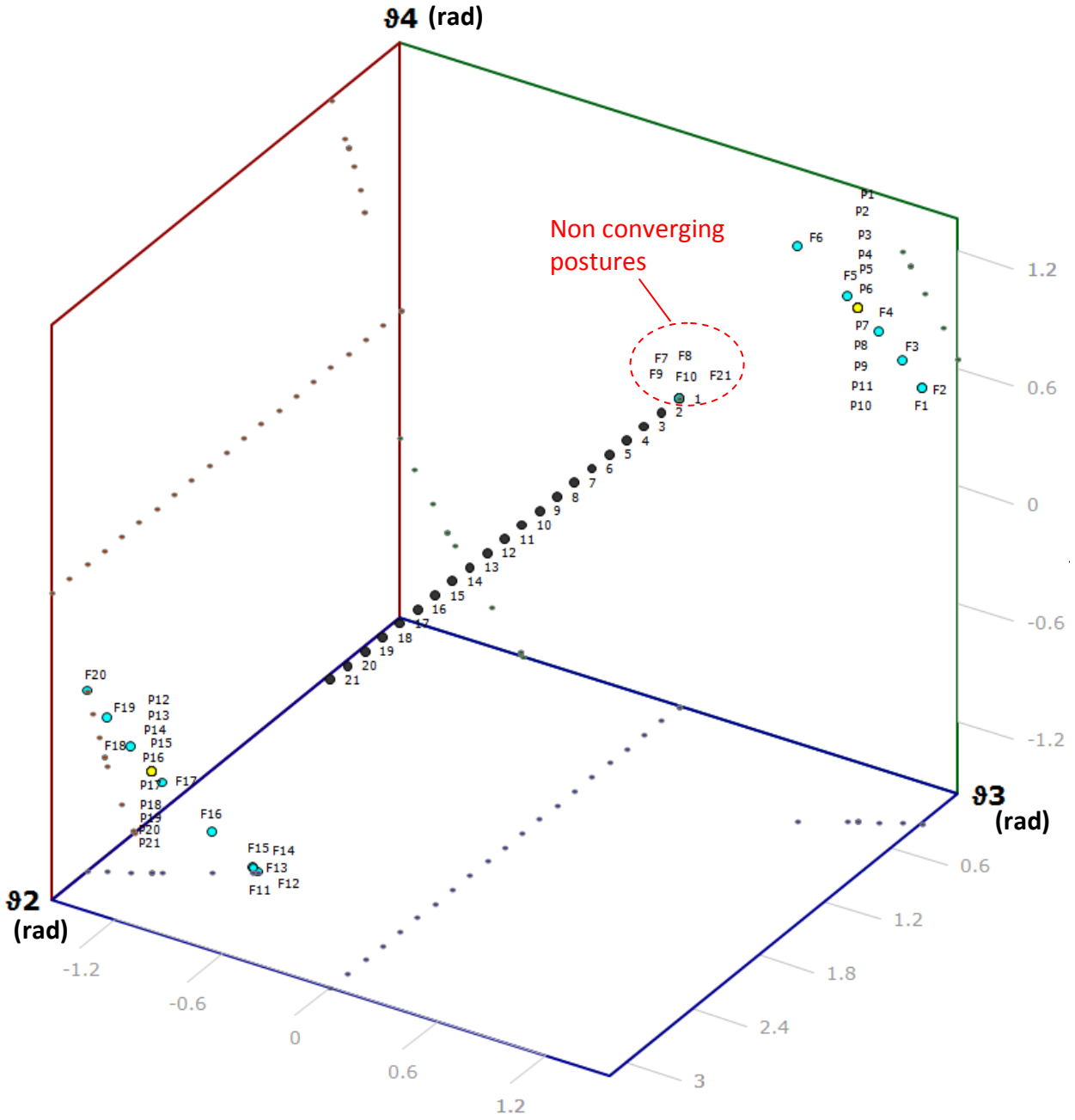


Figure 38 Coverage comparison between FABRIK and Pseudo Inverse Jacobian IK output posture from a common set of initial posture inputs with varying ϑ2 angle only.

In another experiment, we created a set of 21 initial postures from posture (ϑ2, ϑ3, ϑ4) = (1.57, 0, 0) rad, by varying only the ϑ3 joint angle. For this, we selected 21 equidistant ϑ3 joint angles

going from their minimum to maximum values (black dots labelled 1 to 21 in Figure 39). By inputting these postures to the FABRIK and Pseudo Inverse Jacobian IK algorithms, we obtained the posture outputs illustrated respectively by blue dots (labelled F1 to F21) and yellow dots (labelled P1 to P21) in Figure 39.

The IK posture results using FABRIK, F1 to F21, give 11 distinct but very concentrated postures around postures: ($\vartheta 2$, $\vartheta 3$, $\vartheta 4$) = (2.4, -0.88, -1.55) rad. The remaining FABRIK initial postures (F1 to F10) lead to non converging solutions as illustrated at (0, 0, 0) in Figure 39. Using the same 21 initial postures inputted to the Pseudo Inverse Jacobian, we obtained the possible postures P1 to P21. Illustrated in Figure 39, we can clearly see that P1 to P21 give a better coverage of the IK output posture space compared to converging postures outputted from FABRIK (F12 to F21). Indeed, the Pseudo Inverse Jacobian outputted eight far apart distinct results covering the two operational spaces and no non-converging posture.

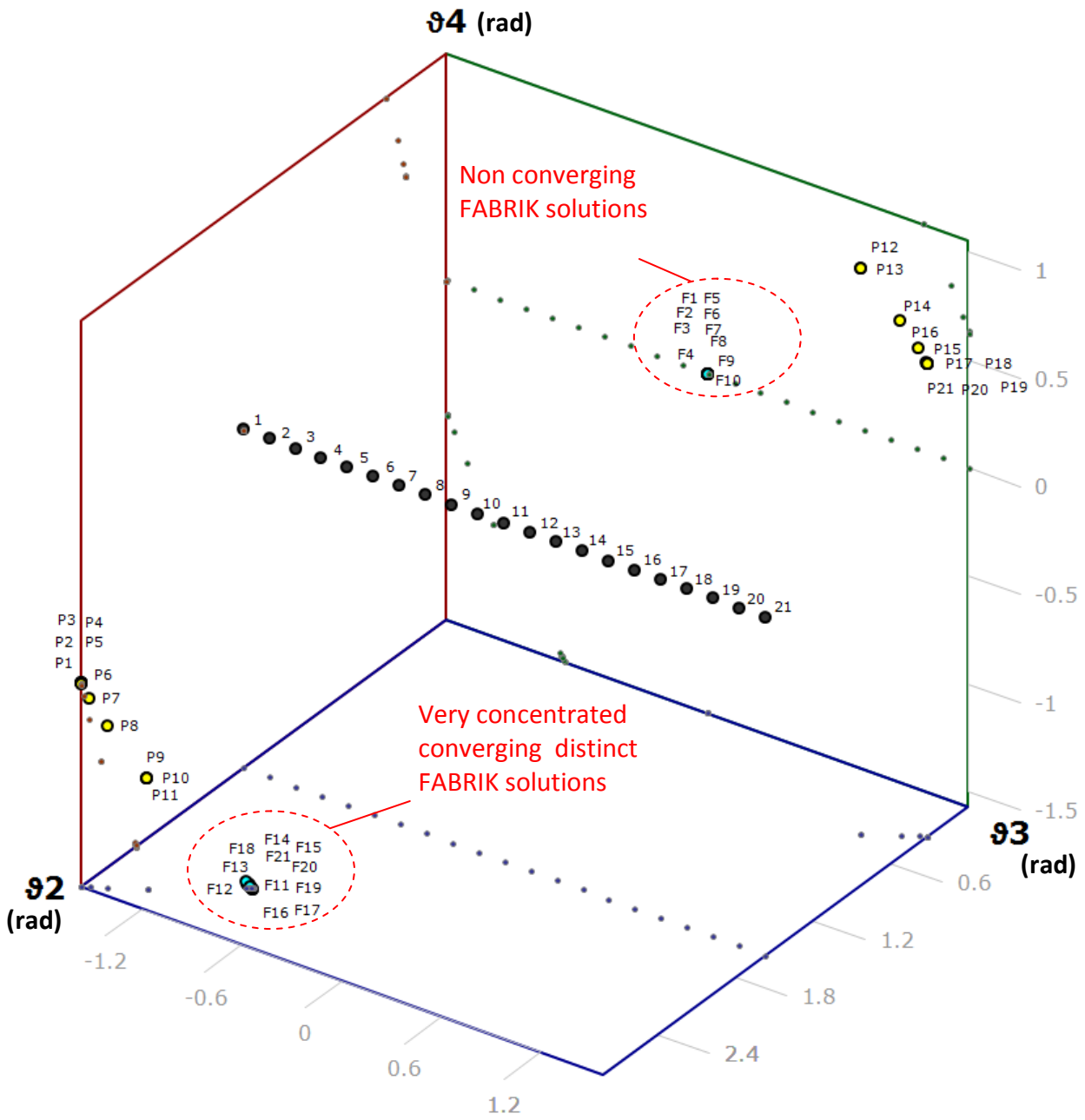


Figure 39 Coverage comparison between FABRIK and Pseudo Inverse Jacobian IK output posture from a common set initial posture inputs with varying ϑ3 angle only.

Although the two last experiments gave clear advantages for one IK algorithm or the other, results could be a lot less obvious when output posture results vary in densities and distribution across the IK output posture space. As mentioned at the beginning of this section, a simple way of quantifying the coverage level of the IK output candidate postures given an initial set of

postures is to evaluate the amount of distinct results given a minimum distance between the IK output postures. Based on our empirical observation of posture spatial concentration in the last two experiments (Figure 38 and Figure 39), a minimum threshold value of 0.004 radian between joints was chosen for the experiments that will follow in this chapter. This threshold might seem small; it is indeed not high enough to resolve every concentrated converging distinct FABRIK solutions found in Figure 39 into separate distinct solutions. Hence, this could falsely give a high amount of spatially constricted solutions the illusion of a high coverage results. It is a risk that such situations may occur, but a small threshold is still better than no threshold as it can still decrease the frequency of false high coverage occurrence by merging to a certain extent concentrated outputted postures. Moreover, testing multiple target EE positions for every initial posture set, should even further mitigate the effect of such occurrence. Using a higher threshold would certainly be a better solution to this problem, but it would be at the cost of decreasing the smoothness potential of having dense IK output postures. Another reason for using a small threshold value is to give enough solutions for cases where the range of possible IK outputted postures is also small. As a relative example, in Figure 39, we can see that the range is relatively large since it goes approximately from P1 to F16 in the operational space located at the bottom left side of the graph.

More tests were realized with various resolutions of $\vartheta 2$ and $\vartheta 3$ joint angles using the Pseudo Inverse Jacobian and FABRIK algorithms. The resolution corresponds to the amount of equidistant angles used for a particular joint. From the results, we observed an increase on the portion of redundant or non-converging results with an increase in the resolution. Indeed, by increasing joint $\vartheta 3$ resolutions from 20 to 100 equidistant angles using the Pseudo Inverse Jacobian algorithm, the numbers of distinct results rose from 9 to 34. In comparison, the FABRIK results increased to 18 distinct converging postures, but they were concentrated around the same area of joint space. It is by similarly increasing the $\vartheta 2$ resolution of the postures inputted in the FABRIK algorithm that the number of distinct outputted results increased with an evenly distributed coverage going from 16 to 64 converging postures.

TABLE V, TABLE VI and TABLE VII show the results of further investigation for various target end-effector positions while varying a different joint from posture (ϑ2, ϑ3, ϑ4) = (1.57, 0, 0) rad, respectively ϑ2, ϑ3 and ϑ4, using a resolution of 100 equidistant angles. Results show very poor results when varying ϑ2 joint angles for the Pseudo Inverse Jacobian IK and when varying the ϑ4 joint angles for the FABRIK IK. The latter results using FABRIK confirmed what we already observed in Section 5.2.2.1. Meaning, fewer distinct possible postures outputted by the IK algorithm are found when exploring different angles of joints closer to the end-effector. Overall, the FABRIK algorithm with ϑ2 or ϑ3 varying joint angles gives better coverage using similar time compared to the other method.

TABLE V IK results by varying only ϑ2 joint angle with resolution = 100 equidistant angles (ϑ2 is separated by 0.031416 radian between each posture in the set of initial postures).

IK Method	Target EE position (mm)	# solution found	Time (ms)
FABRIK	(285, 0.0, 216.05)	73	90
	(100.0, 0.0, 250.0)	9	205
	(250.0, 0.0, 100.0)	16	217
	(350.0, 0.0, 0.0)	84	127
	(600.0, 0.0, 600.0)	1	72
	(0.0 ,0.0, 350.0)	64	148
Pseudo	(285, 0.0, 216.05)	1	162
	(100.0, 0.0, 250.0)	1	196
	(250.0, 0.0, 100.0)	1	167
	(350.0, 0.0, 0.0)	26	173
	(600.0, 0.0, 600.0)	1	89
	(0.0 ,0.0, 350.0)	2	130

The relation between the target position chosen for each experiment and the number of solutions found depends on the distance of the EE relative to the boundary of the operational space of the robotic arm combining all reachable EE positions. An EE located on the boundary, such as (600.0, 0.0, 600.0) will usually have only one possible solution, while an EE located in the middle of the operational space ((285.0, 0.0, 216.05), or (350.0, 0.0, 0.0)) will have the largest number of possible solutions.

TABLE VI IK results by varying only ϑ3 joint angle with resolution = 100 equidistant angles (ϑ3 is separated by 0.031416 radian between each posture in the set of initial postures).

IK Method	Target EE position (mm)	# solution found	Time (ms)
FABRIK	(285, 0.0, 216.05)	86	75
	(100.0, 0.0, 250.0)	9	186
	(250.0, 0.0, 100.0)	22	223
	(350.0, 0.0, 0.0)	83	87
	(600.0, 0.0, 600.0)	1	82
	(0.0 ,0.0, 350.0)	18	190
Pseudo	(285, 0.0, 216.05)	37	186
	(100.0, 0.0, 250.0)	1	217
	(250.0, 0.0, 100.0)	10	218
	(350.0, 0.0, 0.0)	24	200
	(600.0, 0.0, 600.0)	1	90
	(0.0 ,0.0, 350.0)	34	197

TABLE VII IK results by varying only ϑ4 joint angle with resolution = 100 equidistant angles (ϑ4 is separated by 0.031416 radian between each posture in the set of initial postures).

IK Method	Target EE position (mm)	# solution found	Time (ms)
FABRIK	(285, 0.0, 216.05)	1	80
	(100.0, 0.0, 250.0)	1	121
	(250.0, 0.0, 100.0)	0	274
	(350.0, 0.0, 0.0)	1	101
	(600.0, 0.0, 600.0)	1	97
	(0.0 ,0.0, 350.0)	1	102
Pseudo	(285, 0.0, 216.05)	35	185
	(100.0, 0.0, 250.0)	10	199
	(250.0, 0.0, 100.0)	22	214
	(350.0, 0.0, 0.0)	23	196
	(600.0, 0.0, 600.0)	1	91
	(0.0 ,0.0, 350.0)	30	156

5.2.4 Improving the IK output coverage: Coverage by varying all joint angles

Varying all joints in the set of initial postures to get the optimum IK output coverage is probably the most obvious approach. We used this strategy during the early development of the first prototype for the three posture control solutions. As stated previously, we are seeking the optimum IK output coverage that corresponds to the maximum amount of possible postures

able to reach a certain new end-effector position. Hence, offering the best chance of closely reaching the optimal desired posture from a new attraction point position or user trace. The idea is to gather N equidistant initial posture sets across all joint angles range ($\vartheta 2$, $\vartheta 3$, $\vartheta 4$). As illustrated in black dots in Figure 40, we first use a set of N=100 initial postures. To get to this amount of initial postures, 5 equidistant joint angle values for $\vartheta 2$ and $\vartheta 3$, and 4 for $\vartheta 4$ (5 x 5 x 4 = 100), are used to cover each joint min-max angle range as equally as possible. These initial postures are inputted to both the Pseudo-Inverse Jacobian and FABRIK IK algorithms. The resulting postures are shown as yellow dots in Figure 41 for the Pseudo-Inverse Jacobian and Figure 42 for the FABRIK algorithm. For the Pseudo-inverse Jacobian, we observed 40 distinct possible output postures. For the FABRIK algorithm, however, only 15 posture solutions were outputted, resulting in poorer sample coverage compared to the Pseudo-Inverse Jacobian method.

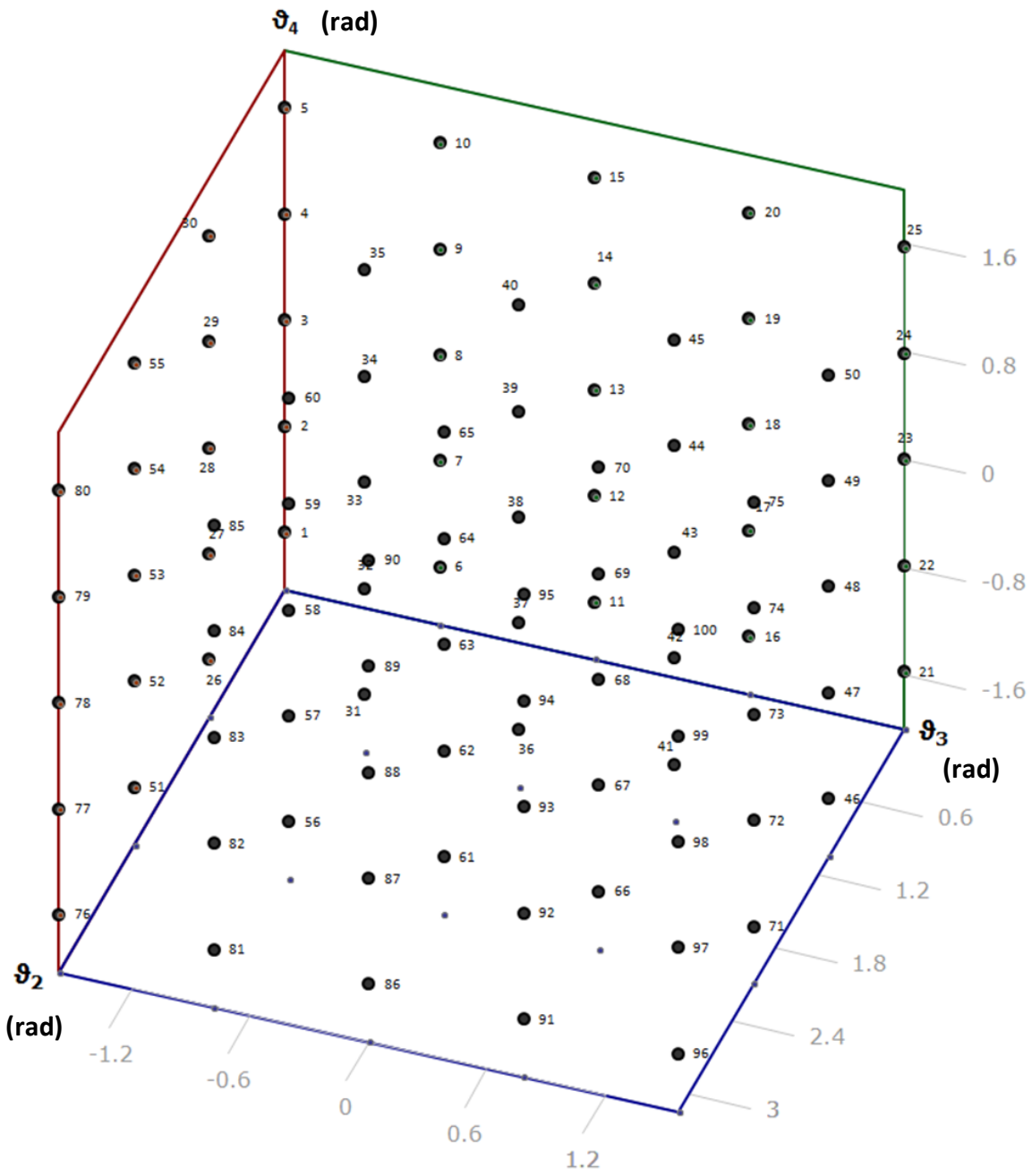


Figure 40 Distributed equidistant initial postures to be inputted in the IK algorithm.

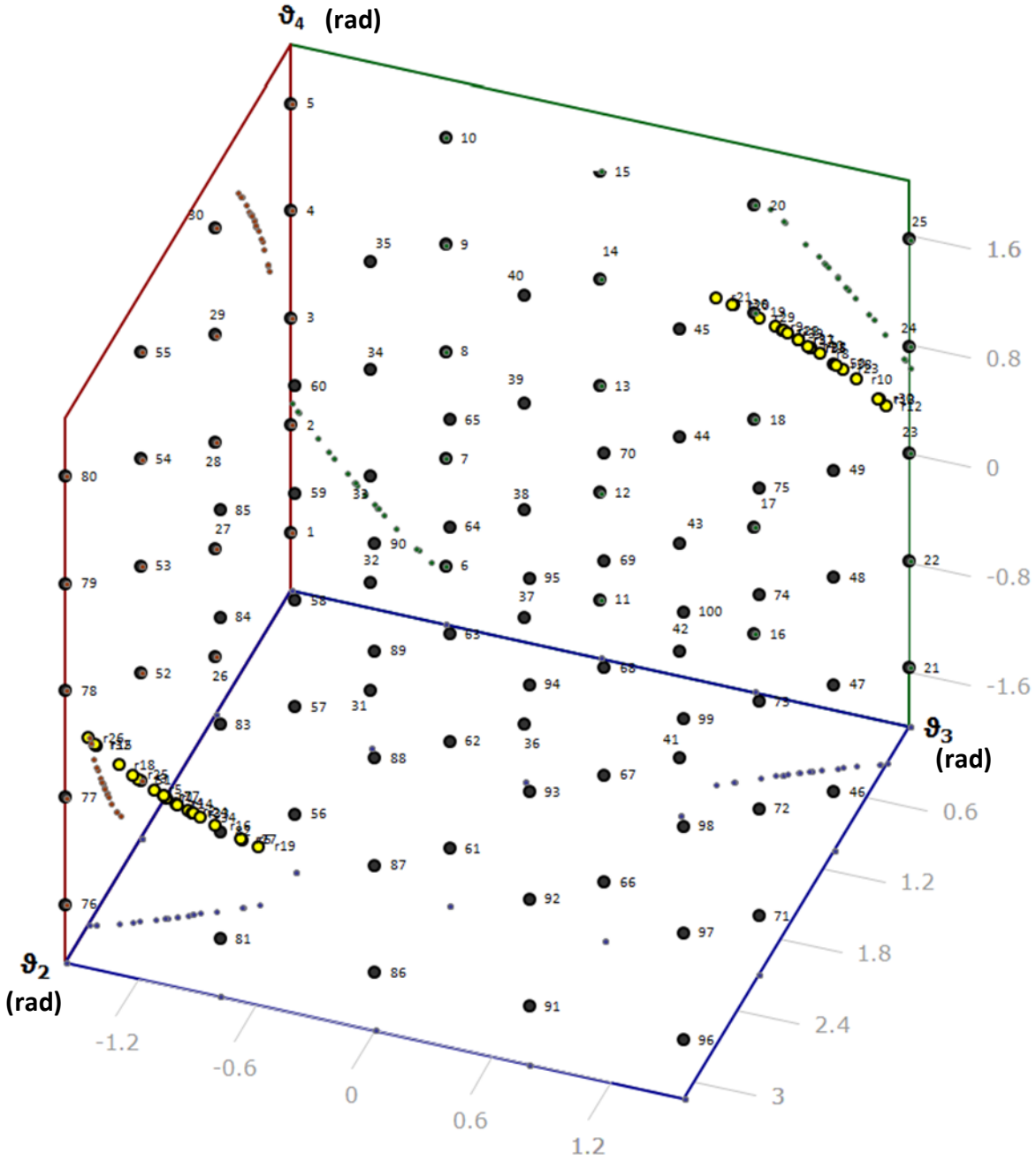


Figure 41 Possible postures (yellow dots) generated from distributed initial postures (black dots) with Pseudo Inverse Jacobian IK and target EE position = (0, 0, 350). Blue, red and green dots are the possible posture projections on respectively the ϑ_2- ϑ_3, ϑ_2- ϑ_4 and ϑ_3- ϑ_4 planes.

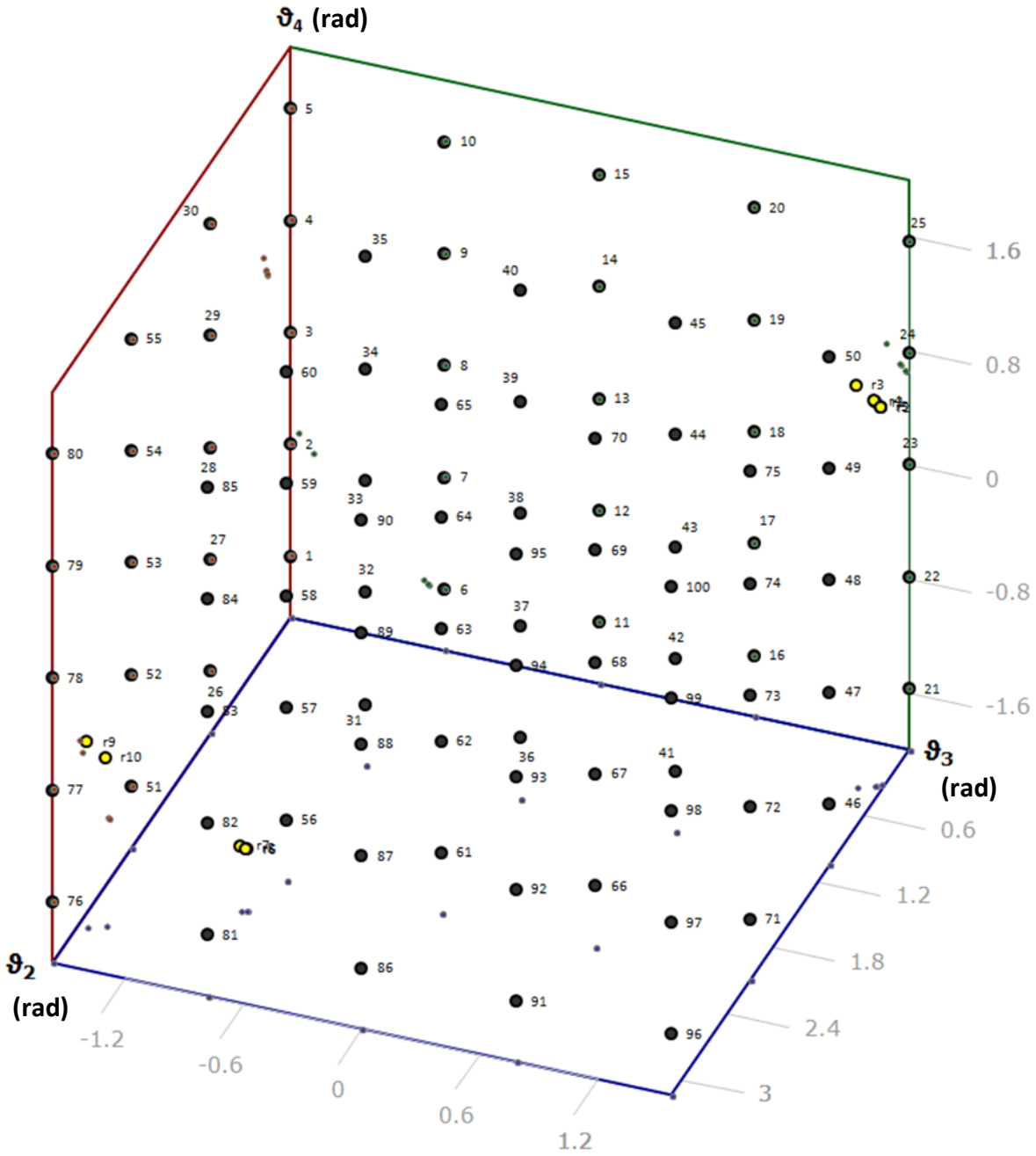


Figure 42 Possible postures (yellow dots) generated from distributed initial postures (black dots) with FABRIK IK and target EE position = (0, 0, 350). Blue, red and green dots are the possible posture projections on respectively the ϑ_2- ϑ_3, ϑ_2- ϑ_4 and ϑ_3- ϑ_4 planes.

In TABLE VIII, we present the same experiments applied to various target end-effector positions along with the resulting amount of possible postures found and time of execution. Compared to

ϑ2 and ϑ3 single varying joints obtained in Section 5.2.3 for the FABRIK algorithm, the resulting outputted posture coverage obtained by varying all joints (ϑ2, ϑ3, ϑ4) to get a total of 100 distributed posture inputs is either equal or inferior across all target EE positions tested with FABRIK algorithm. The results for varying all joints in the initial posture set for the Pseudo Inverse Jacobian are somewhat equivalent to those obtained for ϑ3 and ϑ4 single varying joints obtained in Section 5.2.3, since some end-effector results were better while others were worse or equal.

TABLE VIII Comparison of the possible posture coverage generated from FABRIK and Pseudo Inverse Jacobian for a set of initial postures with resolution of ϑ2:ϑ3:ϑ4 = 5:5:4 (N =100).

IK Method	Target EE position (mm)	# solution found	Time (ms)
FABRIK	(285, 0.0, 216.05)	15	117
	(100.0, 0.0, 250.0)	2	217
	(250.0, 0.0, 100.0)	3	231
	(350.0, 0.0, 0.0)	20	124
	(600.0, 0.0, 600.0)	1	90
	(0.0 ,0.0, 350.0)	15	162
Pseudo	(285, 0.0, 216.05)	31	182
	(100.0, 0.0, 250.0)	1	248
	(250.0, 0.0, 100.0)	9	194
	(350.0, 0.0, 0.0)	27	193
	(600.0, 0.0, 600.0)	1	82
	(0.0 ,0.0, 350.0)	40	173

5.2.5 Improving the IK output coverage: Optimization by varying a subset of joints

From the results obtained in Section 5.2.3, we observed that some joints contributed poorly to the number of possible postures generated by the IK. Indeed, varying only ϑ4 in the set of initial postures inputted to the FABRIK algorithm and varying only ϑ2 in the set inputted to the Pseudo Inverse Kinematics outputted a really low number of possible postures compared to the other initial posture sets in the experiment described in that section. To improve the number of possible posture outputs, we could simply fix the angle values for those low contributing joints and vary all the others in the set of initial postures. Presented in TABLE IX, the results obtained

for both FABRIK and Pseudo Inverse Jacobian outputted better and faster coverage compared to the results obtained varying all joints in Section 5.2.4. Compared to single varying joints investigated in Section 5.2.3, a greater number of output postures from the Pseudo Inverse Jacobian IK algorithm, obtained by varying both $\vartheta 3$ and $\vartheta 4$ equally (TABLE IXa), was obtained using a small increase in computer cost. For the output posture coming from the FABRIK IK calculation, using a set of initial postures inputted with varying $\vartheta 2$ or $\vartheta 3$ angles only, the resulting coverage was better and faster for most target EE positions compared to varying both $\vartheta 2$ and $\vartheta 3$ equally as illustrated in TABLE IXb and Figure 43. This was somewhat of a surprise as it appears that using an initial posture set varying only their joint space coordinates in the $\vartheta 2$ or $\vartheta 3$ single space dimension resulted in a better output posture coverage than varying their joint space coordinates in more joint dimensions (i.e. $\vartheta 2$:$\vartheta 3$ or $\vartheta 3$:$\vartheta 4$ or $\vartheta 2$:$\vartheta 3$:$\vartheta 4$). Hence, using a higher resolution over a set of one varying joint angles for the initial postures gives better results than a lower resolution on a set of two or three varying joint angle postures to catch distinct solutions with the same amount of initial postures.

TABLE IX Comparison of the distributed search for possible postures between FABRIK and Pseudo Inverse Jacobian for resolution of (a) $\vartheta 2$:$\vartheta 3$:$\vartheta 4$ = 10:10:1 and (b) $\vartheta 2$:$\vartheta 3$:$\vartheta 4$ = 1:10:10 angle steps per joint explored.

IK Method	Target EE position (mm)	# solution found	Time (ms)
FABRIK	(285, 0.0, 216.05)	62	120
	(100.0, 0.0, 250.0)	5	215
	(250.0, 0.0, 100.0)	12	238
	(350.0, 0.0, 0.0)	85	112
	(600.0, 0.0, 600.0)	1	84
	(0.0 ,0.0, 350.0)	54	142
Pseudo	(285, 0.0, 216.05)	30	184
	(100.0, 0.0, 250.0)	1	212
	(250.0, 0.0, 100.0)	8	196
	(350.0, 0.0, 0.0)	22	181
	(600.0, 0.0, 600.0)	1	81
	(0.0 ,0.0, 350.0)	32	167

(a)

IK Method	Target EE position (mm)	# solution found	Time (ms)
FABRIK	(285, 0.0, 216.05)	8	107
	(100.0, 0.0, 250.0)	2	203
	(250.0, 0.0, 100.0)	2	230
	(350.0, 0.0, 0.0)	10	91
	(600.0, 0.0, 600.0)	1	94
	(0.0 ,0.0, 350.0)	5	183
Pseudo	(285, 0.0, 216.05)	39	213
	(100.0, 0.0, 250.0)	11	207
	(250.0, 0.0, 100.0)	20	214
	(350.0, 0.0, 0.0)	34	191
	(600.0, 0.0, 600.0)	1	84
	(0.0 ,0.0, 350.0)	42	184

(b)

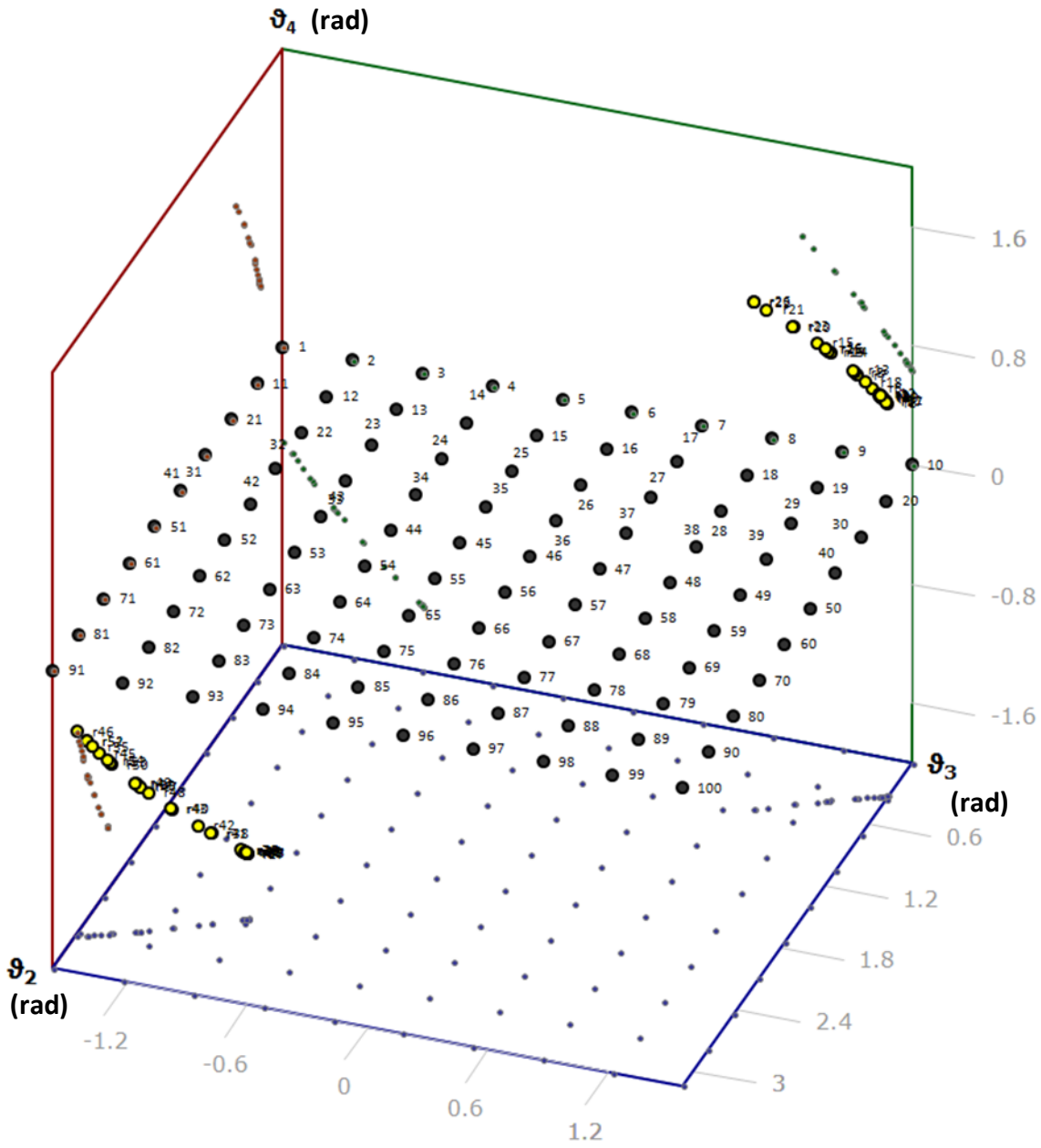


Figure 43 Possible postures generated from distributed initial postures restricted to the plane ϑ2-ϑ3 with FABRIK IK (resolution ϑ2:ϑ3:ϑ4 = 10:10:1). Blue, red and green dots are the possible posture projections on respectively the ϑ_2- ϑ_3, ϑ_2- ϑ_4 and ϑ_3- ϑ_4 planes.

5.2.6 Improving the IK output coverage: Choosing the right IK algorithm

Based on the results obtained in the last section, the initial posture set using single varying $\vartheta 2$ or $\vartheta 3$ joint angle and varying $\vartheta 3$ and $\vartheta 4$ joint angles generated the best output posture coverage using the FABRIK and Pseudo Inverse Jacobian IK algorithm respectively. When grouping the target EE positions together, the output posture results coming from the FABRIK algorithm (with single varying $\vartheta 2$) outnumbered the Pseudo Inverse Jacobian output postures by 61% using 21% less time in computation. Some target EE positions outputted better and/or faster posture coverage in favor of the Pseudo Inverse Jacobian, but with marginal gain. The single varying $\vartheta 3$ angle initial posture set with the FABRIK algorithm was a close second behind the single varying $\vartheta 2$ with 12% less output posture.

In addition, we also experimented with different initial posture set resolution across all joints for two target EE positions using the FABRIK algorithm. The results, presented in TABLE X and TABLE XI, show that using initial postures with varying $\vartheta 3$ only followed by varying $\vartheta 2$ only joint angles gives better overall results for the output posture coverage compared to all the other resolution ratios between $\vartheta 2$, $\vartheta 3$ and $\vartheta 4$. The resolution represents the number of steps per maximum angle range tried in the set of initial postures inputted to the IK algorithm. The maximum angle range of each joint (except the gripper) used in our robotic arm configuration is half a revolution (i.e. π radians). We also observed no trend in the output coverage results that seem to indicate a convergence toward other ($\vartheta 2$, $\vartheta 3$, $\vartheta 4$) ratios. In the two tables presented below, a resolution of 1 is represented by a single value (chosen to be the middle value inside the range).

TABLE X Number of possible postures found with FABRIK in relation with different proportions in resolution between ϑ2, ϑ3 and ϑ4 in the initial posture set used when attempting to reach target = (285, 0.0, 216.05).

ϑ2 resolution (# of steps tried / (max. ϑ2 range = π rad))	ϑ3 resolution (# of steps tried / (max. ϑ3 range = π rad))	ϑ4 resolution (# of steps tried / (max. ϑ4 range = π rad))	Number of possible postures found
100	1 (ϑ2 = -π/2 rad)	1 (ϑ3 = 0 rad)	73
50	2	1 (ϑ3 = 0 rad)	54
33	3	1 (ϑ3 = 0 rad)	60
25	4	1 (ϑ3 = 0 rad)	55
20	5	1 (ϑ3 = 0 rad)	61
16	6	1 (ϑ3 = 0 rad)	59
14	7	1 (ϑ3 = 0 rad)	62
10	10	1 (ϑ3 = 0 rad)	62
5	20	1 (ϑ3 = 0 rad)	57
3	33	1 (ϑ3 = 0 rad)	53
2	50	1 (ϑ3 = 0 rad)	32
1 (ϑ1 = π/2 rad)	100	1 (ϑ3 = 0 rad)	85
1 (ϑ1 = π/2 rad)	1 (ϑ2 = -π/2 rad)	100	1

TABLE XI Number of possible postures found with FABRIK in relation to different resolutions between ϑ2, ϑ3 and ϑ4 trying to reach target = (0.0, 0.0, 350.0).

ϑ2 resolution (# of step tried / (max. ϑ2 range = π rad))	ϑ3 resolution (# of step tried / (max. ϑ3 range = π rad))	ϑ4 resolution (# of step tried / (max. ϑ4 range = π rad))	Number of possible postures found
100	1 (ϑ2 = -π/2 rad)	1 (ϑ3 = 0 rad)	63
50	2	1 (ϑ3 = 0 rad)	30
33	3	1 (ϑ3 = 0 rad)	47
25	4	1 (ϑ3 = 0 rad)	46
20	5	1 (ϑ3 = 0 rad)	47
16	6	1 (ϑ3 = 0 rad)	49
14	7	1 (ϑ3 = 0 rad)	50
10	10	1 (ϑ3 = 0 rad)	55
5	20	1 (ϑ3 = 0 rad)	58
3	33	1 (ϑ3 = 0 rad)	48
2	50	1 (ϑ3 = 0 rad)	52
1 (ϑ1 = π/2 rad)	100	1 (ϑ3 = 0 rad)	17
1 (ϑ1 = π/2 rad)	1 (ϑ2 = -π/2 rad)	100	1

Based on the cumulated results obtained in this section, a set of ϑ2 varying joint angles only as initial postures inputted to the FABRIK algorithm is the best choice for optimum output posture coverage. These results are based on the assumption that the small sample of target EE positions is representative of the entire set of possible target EE positions of the robotic arm.

While those results were obtained using a 3 DOF planar revolute robotic arm, they can be transposed to any number of robotic arm configurations (number of joints, types of joints, etc.) for our three proposed approaches as long as the amount of initial postures obviously does not exceed the amount of possible $\vartheta 2$ angle values.

5.3 Directional exploration: Greedy Best-first search

As explained in Section 4.3, there are situations where a new optimal possible posture is expected to be close to the previous robotic arm posture. The attraction point approach is an example of such a situation. Indeed, the application being real-time, the increment between each attraction point position is very small. Hence, instead of using a wide coverage of initial postures and finding the best posture from their IK output postures, a more local and directional search would be more appropriate. The Greedy best first search seems to be a promising approach in theory, and this section will be devoted to testing it.

The Greedy best first search is a search algorithm that consists in exploring each children of the parent node by evaluating their weight according to a certain rule. If the weight of the parent is greater compared to each of its children, the search stops and returns the parent node as the most promising node. If the child node with the highest weight is greater than the weight of its parent, then the search continues into the children of that node. There is no backtracking allowed using that method which gives better time performance but at the cost of less accurate results. For the experiment described in this section, the rules for evaluating weight are those specific to the Attraction point method calculated in Algorithm 8 where weights are represented by Euclidean distance (for similarity comparison) between postures.

There are two ways of exploring locally using the greedy algorithm. Each time a new best neighbor posture is found, we can either restart a new iteration of the search from (1) the initial posture input that leads to that new best posture (see Figure 23 and Algorithm 9) or (2) the actual new best IK output posture (see Figure 44). A possible upside of the latter approach would be the ability to jump closer to a local optimal posture and end the search faster. The downside could be that better solutions could be missed in areas overlooked by the jumps and

the resulting greater amount of new neighbor postures to explore at each new iteration of the search might reduce the time gained by those jumps. In Figure 44, an example is given where the values presented in the grid correspond to the distances calculated on the IK output posture using initial postures with different angle values of two arbitrary joint $\Delta\vartheta_{iA}$ (vertical header) and $\Delta\vartheta_B$ (horizontal header), where A≠B, relative to the first possible posture found (ϑ_A, ϑ_B) in a local area (indicated in the horizontal and vertical header). Only two joints are used here to simplify the representation. The black dash arrow line represents a jump from an initial posture to its corresponding output posture (ex.: ($\Delta\vartheta_A$=1, $\Delta\vartheta_B$=-1) to ($\Delta\vartheta_A$=2, $\Delta\vartheta_B$=-5)) values. The red arrow line represents a transfer between the last best IK output posture and the new best IK output posture that will start the next iteration of the search.

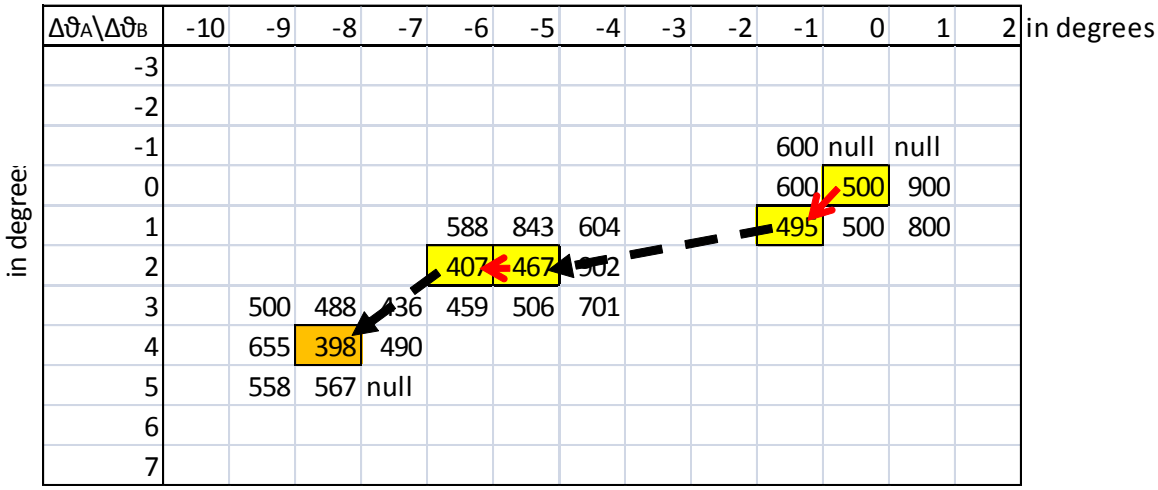


Figure 44 Search for the posture with lowest minimal distance possible with the greedy algorithm using the new best IK output posture to restart each new search iteration. Different angles (in degrees) of ΔϑA and ΔϑB relative to the current initial posture are explored. (Cubic distance values are in units of 1x106 in accordance with Algorithm 8).

Although, we already determined in the last section that the FABRIK algorithm leads to better IK output posture coverage using a set of initial postures with varying ϑ2 only, the current section will still investigate the best coverage obtained from the Pseudo Inverse Jacobian (with an input of initial postures with varying ϑ3- ϑ4 joint angles) since we are using a different search strategy.

The greedy best-first search can still work with IK algorithms subject to singularities like the Pseudo Inverse Jacobian. By exploring all directions at each iteration of the search, the best direction will automatically tend to go around singularities by always choosing the direction with the best matching posture while avoiding non converging directions.

Another matter comes to mind when using a varying single or subset of joints for the set of initial joints. In this strategy, at each iteration of the search, we are exploring the immediate posture neighborhood of the last initial or resulting best posture found in the last iteration. If we restrict the search by varying one or two joints, we limit considerably our exploration before reaching a local optima, hence greatly reducing our chances of reaching a local optima close to the optimum posture. By varying 3 joints, 2 joints or 1 joint angle we can explore respectively 26, 8 and 2 neighborhood postures. To compensate this unequal limitation, we will simply search more than just the immediate neighborhood area to get an amount of explored postures equal to an exploration varying all joints angles.

The initial condition of the test that follows is presented in TABLE XII.

TABLE XII Initial conditions prior to the greedy best-first search experiment comparing the FABRIK and Pseudo Inverse Jacobian IK algorithms.

Initial conditions parameters	Initial conditions Values
Current robotic arm posture	$\{\vartheta 1, \vartheta 2, \vartheta 3, \vartheta 4\}$ = { 0, 2.401, -1.568, -1.42 } rad
Current end-effector (target) position	(x, y, z) = (100.0, 0, 250.0) mm
Current attraction point vector direction	$(\vec{x}, \vec{y}, \vec{z})$ = (-1, 0, 1)

For comparison purposes, this section includes the results for both FABRIK and Pseudo Inverse Jacobian using an exploration varying all joint angles. To trigger the local exploration activity, we change the direction of the current attraction point vector (from its initial vector value defined in TABLE XII) going from O_A to the attraction point (see Figure 21) by -0.1 $\vec{x}$ to $(\vec{x}, \vec{y}, \vec{z})$ = (-1.1, 0, 1) using the XYZ reference frame defined in Figure 29. The results are presented in TABLE XIII and TABLE XIV for the two iteration techniques described earlier in this section and different search resolutions (i.e. the number of angles distributed within the limits of the angle range for each joint). The solution distance is calculated using Algorithm 8.

TABLE XIII Exploration using the initial posture to restart a new iteration.

IK Method	$\vartheta 2$:$\vartheta 3$:$\vartheta 4$ joint resolutions (# of steps tried / (joint max. range = π rad for every joint))	# IK iterations before converging	Solution distance	Time (ms)
FABRIK	15 = 15:15:15	2	44444412.8	94
	25	2	44300860.97	82
	50	3	44300860.97	94
	100	4	44270003.95	81
Pseudo	15	2	44263784.24	107
	25	3	44287384.65	105
	50	3	44283347.05	137
	100	5	44225810.77	115

TABLE XIV Exploration using the IK output posture to restart a new iteration.

IK Method	$\vartheta 2$:$\vartheta 3$:$\vartheta 4$ joint resolutions (# of steps tried / (joint max. range = π rad for every joint))	# IK iterations before converging	Solution distance	Time (ms)
FABRIK	15 = 15:15:15	3	44292466.91	102
	25	2	44300860.97	85
	50	4	44275426.18	85
	100	4	44270003.95	84
Pseudo	15	2	44263784.24	109
	25	3	44233603.41	111
	50	3	44273013.76	116
	100	4	44259198.67	134

The difference in time and accuracy between FABRIK and Pseudo-Inverse Jacobian are marginal. The former is faster and the latter more accurate based on a comparison of the distance results obtained across all resolutions. The results could be explained simply by the initial conditions used for this particular test, which could give slightly better results with the Pseudo-Inverse compared to FABRIK. The resolution did not show much influence on the time of execution or the accuracy for both the FABRIK and Pseudo Inverse methods. When comparing the initial posture against the resulting posture technique to restart a new iteration, both were similar overall in execution time and accuracy.

For the next round of tests, we arbitrarily choose the initial posture iteration technique with a resolution of ϑ2:ϑ3:ϑ4 = 100 steps angle tried/(joint maximum range in radians) since the previous tests showed no significant influence from those parameters. This time, however, different initial conditions for the initial arm posture (ϑ2, ϑ3, ϑ4) and attraction point vector direction, presented in TABLE XV, are studied. The attraction point vector direction is the vector from O_A to the Attraction point illustrated in Figure 21 using the XYZ reference frame defined in Figure 29.

TABLE XV Description of the different initial conditions in terms of posture joint angles, target position and attraction point direction for evaluating the best approach for the local directional search with Greedy Best First Search.

Initial condition #	robotic arm posture {ϑ2, ϑ3, ϑ4} in rad	end-effector (target) position (x, y, z) in mm	attraction point vector direction $(\vec{x}, \vec{y}, \vec{z})$ in mm
1	1.026, -0.43, -1.57	285, 0.0, 216.05	1, 0, 1
2	2.408, -1.576, -1.408	100.0, 0.0, 250.0	-1, 0, 1
3	1.318, -1.402, -1.191	250.0, 0.0, 100.0	0, 0, 1
4	0.318, -0.426, -1.184	350.0, 0.0, 0.0	1, 0, 1
6	2.789, -1.501, -0.743	0.0, 0.0, 350.0	-1, 0, 0

Similar to the last experiment, to trigger the local exploration activity, we simulate a user gesture triggering a change in the attraction point vector by -0.1 $\vec{x}$ from its initial condition. This triggers the system to reposition the arm to a posture with the minimum distance, calculated from Algorithm 8, relative to the new attraction point vector direction. For example, from the initial condition #1 in TABLE XV, the attraction point vector is changed to $(\vec{x}, \vec{y}, \vec{z})$ = (-1, 0, 1) + (-0.1, 0, 0) = (-1.1 0, 1) which becomes the new inputs to Algorithm 9. The output of Algorithm 9 for each initial condition and IK algorithm are presented in TABLE XVI using an initial posture set varying all joint angles (ϑ2, ϑ3, ϑ4) with each a resolution of 100 angular steps tried/(joint maximum range in radian). In the last section, the initial posture set and IK algorithm generating the best IK output coverage was found to be respectively with single varying ϑ2 and the FABRIK algorithm. Expecting to get even lower minimum posture distance results, we use these configurations with a resolution of 100 angular steps tried/(joint maximum range in radian) in Algorithm 9. Results are presented in TABLE XVII. The results using the initial posture

set (varying $\vartheta 3$:$\vartheta 4$) found in the last section to get the best coverage with the Pseudo Inverse Jacobian are also presented in TABLE XVII.

TABLE XVI Comparison between FABRIK and Pseudo Inverse Jacobian IK algorithm with all joint varying angles and common resolution of 100 angular steps tried/(joint maximum range in radian) for the initial condition described in TABLE XV.

IK Method	Initial Condition #	# IK iterations before converging	Posture minimum distance obtained	Time (ms)
FABRIK	1	2	88531933.13	67
	2	2	41633395.32	68
	3	7	34833505.94	67
	4	4	68065134.05	71
	5	2	80351231.82	65
Pseudo	1	14	94688521.56	90
	2	2	41655769.81	90
	3	10	34791065.67	83
	4	16	71495607.68	92
	5	3	80531385.34	86

TABLE XVII Comparison between FABRIK with initial posture set of varying $\vartheta 2$ joint only (resolution of (100,0,0) angular steps tried/(joint maximum range in radian)) using 26 neighbor search and the Pseudo Inverse Jacobian with initial posture set of varying $\vartheta 3$ and $\vartheta 4$ joint (resolution of (0,100,100) angular steps tried/(joint maximum range in radian) using 26 neighbor searches. The initial conditions used for this experiment are described in TABLE XV.

IK Method	Initial Condition #	# IK iterations before converging	Posture minimum distance obtained	Time (ms)
FABRIK	1	2	107483783.6	92
	2	3	41653814.36	104
	3	2	37322820.28	88
	4	4	83667813.33	100
	5	4	80402143.56	88
Pseudo	1	2	96647079.03	109
	2	2	41655769.81	109
	3	3	35612935.38	125
	4	2	79216478.48	110
	5	2	80538452.08	112

As mentioned in the last section, we discovered that inputting a set of initial postures with varying $\vartheta 2$ joints only to the FABRIK algorithm produced the greatest amount of possible postures. Consequently, we expected to find the best output (postures with lowest minimum distance) using the same recipe applied to the local directional search with Greedy Best First search (Algorithm 9). Instead, the results (presented in TABLE XVII) obtained using this recipe revealed the worst output results (by comparing the posture minimum distances) of all the experiments performed so far in this section. The explanation of this result can be found by reviewing the results of the experiments on wide search with low resolution performed in the last section. Using a set of initial postures varying only by their $\vartheta 2$ joint angle with the FABRIK algorithm, we observed large intervals of exploration converging toward no solutions or duplicate solution. From the $\vartheta 2$ varying tests with multiple EE target positions performed in the last section using FABRIK with initial posture resolution ($\vartheta 2$, $\vartheta 3$, $\vartheta 4$ = (100, 0, 0) angular steps tried/(joint maximum range in radian)) (see TABLE V), a detailed analysis of the results for the EE target position of ($\vartheta 2$, $\vartheta 3$, $\vartheta 4$) = (285, 0, 216.05) is given in TABLE XXV of Appendix B. From the results analysis presented, new distinct converging solutions are either found or not for each initial posture input. The data show that there are large interval spans between $\vartheta 2$=0.0 to 0.44 rad and 2.8 to π rad, for a total $\vartheta 2$ range of 0- π rad, where no new converging posture solution occurs. Similar FABRIK results are observed using the other EE target positions in TABLE V.

Looking back at the raw data from the experiment presented in Section 5.2.5, we similarly observed the existence of initial postures spanning intervals with no new converging solution results using a set of initial postures with varying $\vartheta 2$:$\vartheta 3$ angles. However, those intervals are smaller compared to the initial posture sets of $\vartheta 2$ varying joints, as presented in TABLE XXVI of Appendix B.

This makes sense, while parsing between $\vartheta 2$ =0 and $\vartheta 2$ = 0.44 rad for a fixed $\vartheta 3$ and $\vartheta 4$, no convergence is reached, but if $\vartheta 3$ is also varying then there is a better chance for converging conditions to occur that reduce those intervals. When searching locally around a specific posture, varying multiple joint angles increases the distribution uniformity of the different IK

converging results: (1) Converging to a distinct solution, (2) converging to an already found solution and (3) non-converging to a solution. This gives a greater chance to have a least some converging solutions at each iteration of the Greedy Best First search process, instead of none at the first iteration which consequently returns no solutions for the search.

We already know from Sections 4.2 and 5.2.3 that varying $\vartheta 4$ leads to duplicate solutions using the FABRIK algorithm; hence the best set of initial postures for an optimum FABRIK posture output coverage should be with varying $\vartheta 2$:$\vartheta 3$ joint angles. Surprisingly, the accuracy results (in distance units) and speed of execution obtained with varying $\vartheta 2$:$\vartheta 3$ are similar (see TABLE XVIII) compared to varying $\vartheta 2$:$\vartheta 3$:$\vartheta 4$ initial postures (TABLE XVI). Furthermore, we present in TABLE XVIII the results for initial postures with $\vartheta 2$:$\vartheta 3$ angles but with only an immediate neighbor posture exploration at each search iteration (i.e. 8 explored neighbors posture technique). We obtained results accuracy, by comparing each minimum distance posture obtained from Algorithm 8, similar to the results obtained using 26 neighbors posture exploration at the start of each search iteration. The identical distances obtained when comparing the 8 with the 26 neighbors exploration indicate that the same optimal (i.e minimum distance) posture was found using both techniques. Additionally, the execution time was the same between both methods even if ~35% less IK calculation was observed using the 8 explored neighbors posture technique. When comparing FABRIK technique with the Pseudo Inverse Jacobian IK for all varying joint angles, the FABRIK algorithm was able to find more accurate solutions at a lower computing cost. Hence the FABRIK with initial posture with varying $\vartheta 2$:$\vartheta 3$ or $\vartheta 2$:$\vartheta 3$:$\vartheta 4$ was the best choices for the local greedy exploration algorithm. Hence, all those best choices being equal, we decided to use the varying $\vartheta 2$:$\vartheta 3$ initial posture with 8 immediate neighbors exploration per iteration technique for the experiments that follow in the next sections.

TABLE XVIII Comparison between FABRIK with initial posture set of varying ϑ2:ϑ3 joints only (resolution of 100:100:0 angular steps tried/(joint maximum range in radian)) using 8 and 26 neighbors search at each iteration. The initial conditions used for this experiment are described in TABLE XV.

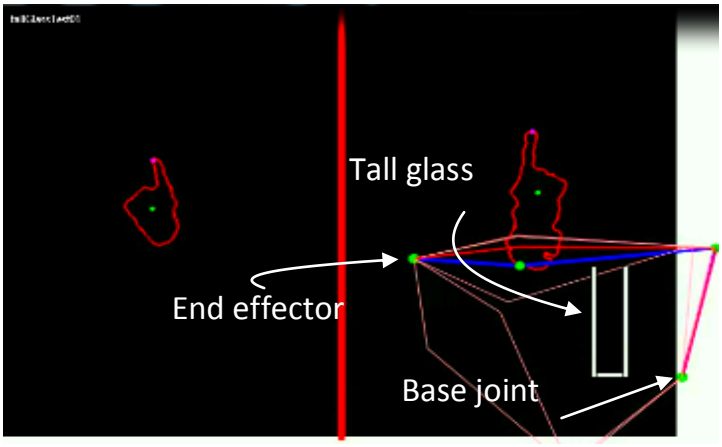
Search resolution (ϑ2:ϑ3:ϑ4) in angular steps tried/(joint maximum range in radian)	# of explored neighbor postures at each iteration	Initial Condition #	# IK iterations before converging	Posture minimum distance obtained	Time (ms)
100: 100: 0	8	1	2	88531933.13	56
		2	2	41633395.32	63
		3	7	34833505.94	63
		4	4	68065134.05	57
		5	2	80351231.82	63
	26	1	2	88531933.13	56
		2	2	41633395.32	65
		3	4	34865643.25	64
		4	4	68065134.05	66
		5	2	80351231.82	64

5.4 Simulated static obstacle avoidance – The tall glass test

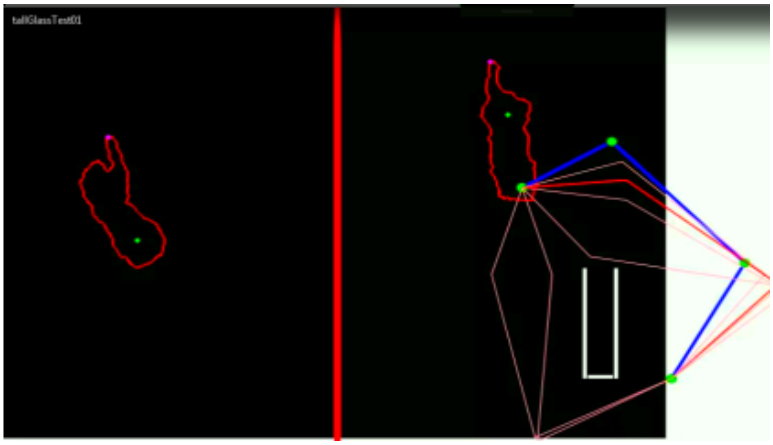
In this section, a task is given to the user that requires the use of a simulation of the joint J2, J3 and J4 of the AL5D configuration but without angle constraints. That means that every joint can potentially have angles between 0 and 360 degrees.

In this first experiment, we test the ability of each posture control method described in Chapter 4 to reach the deepest reachable point allowed by the robotic arm configuration of a tall glass without touching the sides. This is an almost impossible task when relying exclusively on the control of the end-effector without controlling the posture. For our experiment, the virtual glass is 26 pixels wide by 124 pixels tall (equivalent to 2.6 x 12.4 cm) and it is shown in white in Figure 45. Each test begins with a random initial posture where the end-effector is positioned over the glass. When this position is reached, the timer starts (Figure 45a). The initial posture of the robotic arm is rarely an adequate posture from which the user can lead the EE toward the bottom of the glass. Hence, the objective is to change the robotic arm posture to allow the user to move the end-effector toward the bottom of the glass. Using no posture control restricts the

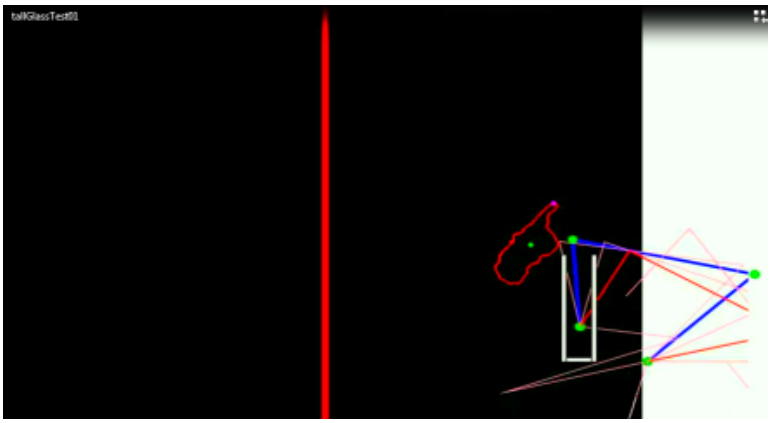
user to only moving the EE toward the bottom of glass while hoping that the posture was initially correct. We can try to change the posture by moving the EE around, but the process is rather tedious (almost futile) while adding more risk of touching the sides of the glass. Using the browse and select method, described in Section 4.2, the user can control the posture by browsing and selecting incrementally better postures (Figure 45b). When the optimum posture is found, we can move the EE to the bottom of the glass by removing our left hand from the Kinect recognition field of view (Figure 45c). When using the Attraction point method, detailed in Section 4.3, we first position the EE over the glass using a random robotic arm posture (Figure 46a). Then we move the EE toward the bottom of the glass with the index fingertip position while simultaneously adjusting the direction of the attraction point in real-time using the pointing direction of the same index finger (Figure 46b-c). For the Tracing method, described in Section 4.4, we first position the EE over the glass using a random robotic arm posture (a). We switch to "Trace New Posture" mode using the user gesture recognition system and trace the desired posture for the task (b). If the suggested posture solution satisfies the user, they can confirm the posture to become the new simulated robotic arm posture (c). Next, we can simply move the EE toward the bottom of the glass (d).



(a)

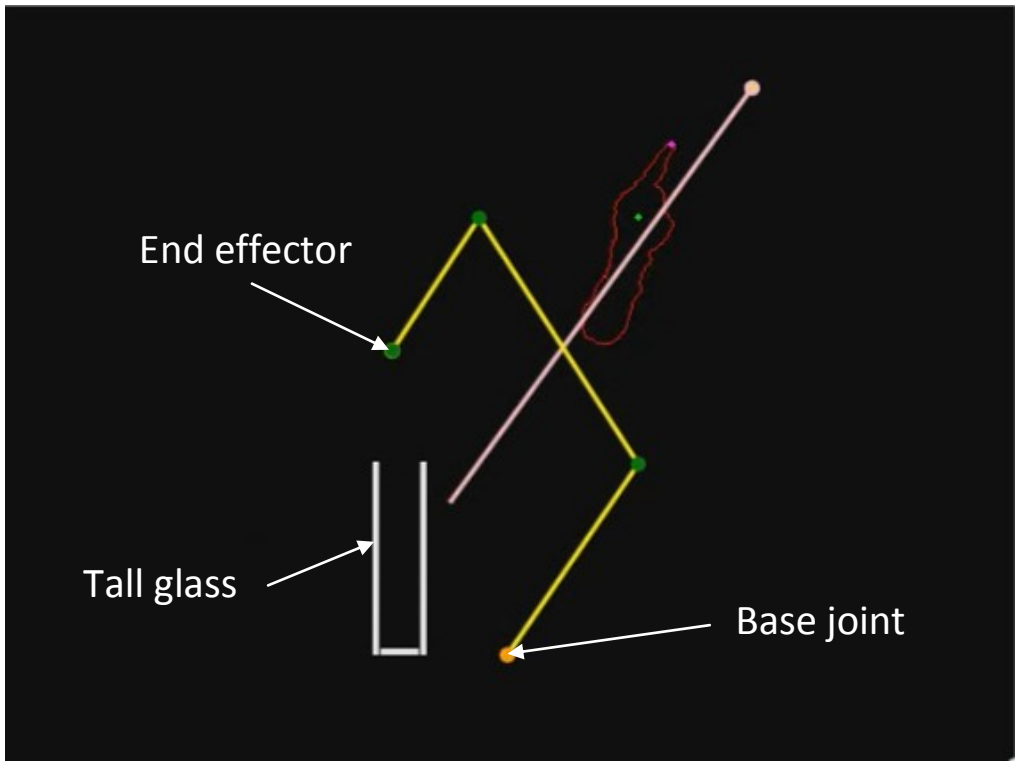


(b)

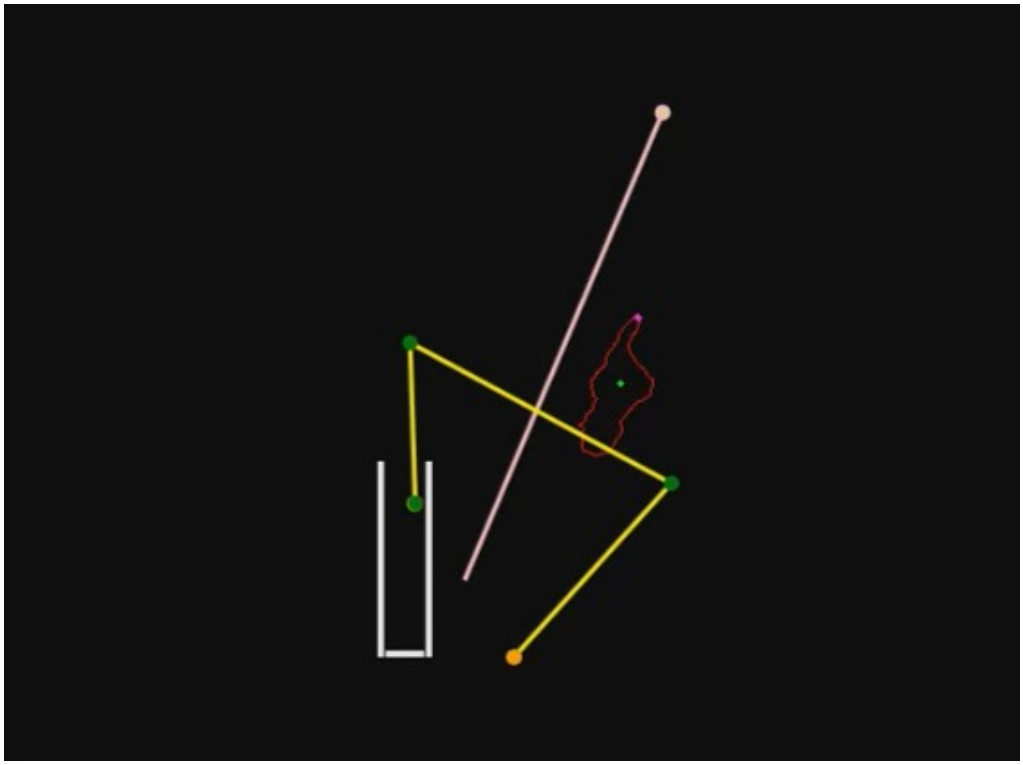


(c)

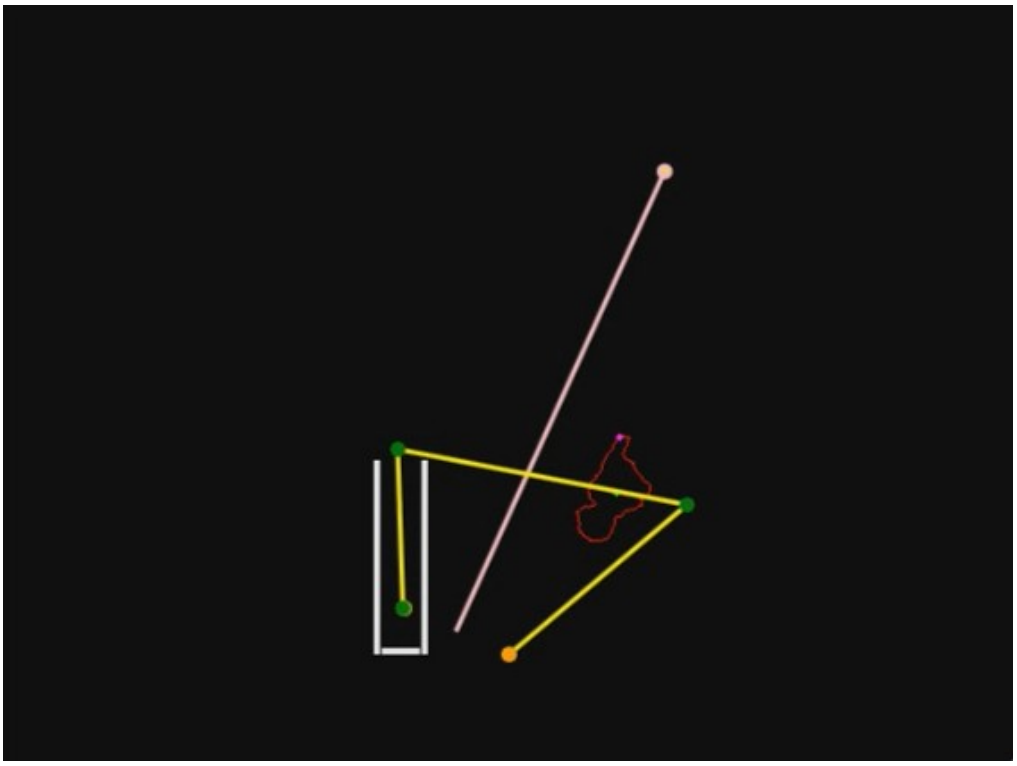
Figure 45 Tall glass test with the Browse and Select method: (a) First posture selected over the glass; timer starts; (b) Browse and select to get to the posture capable of reaching the deepest point possible (with the current configuration of the robotic arm) in the virtual glass, and (c) End-effector reaches the deepest possible point of the virtual glass. The timer stops.



(a)

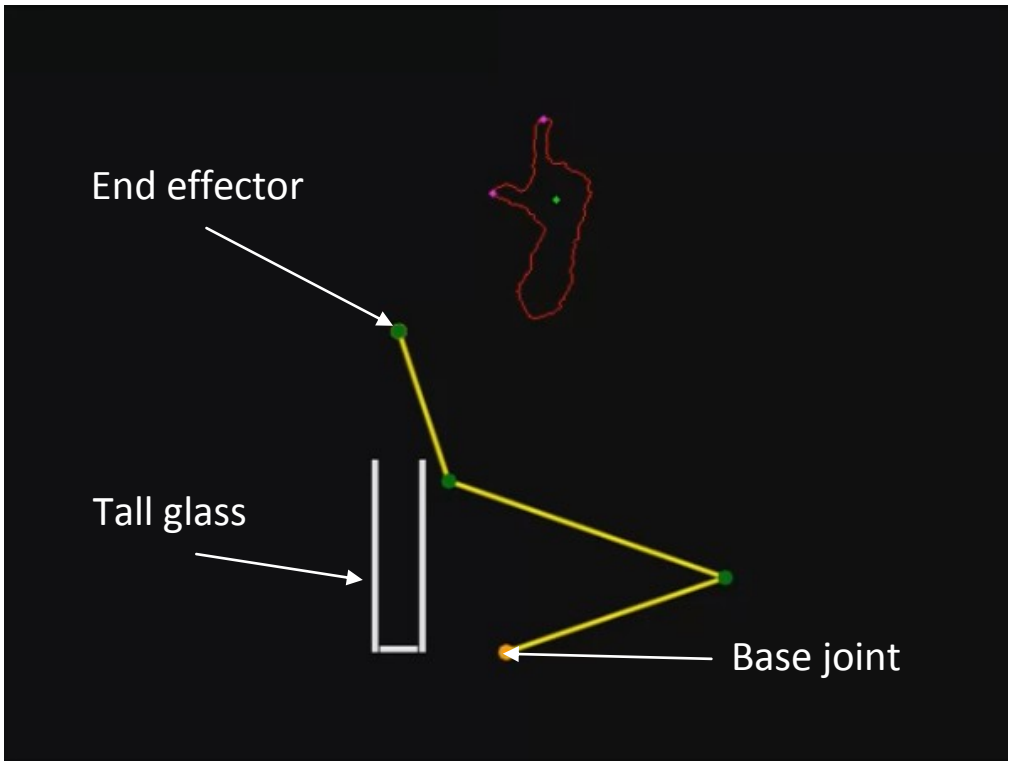


(b)

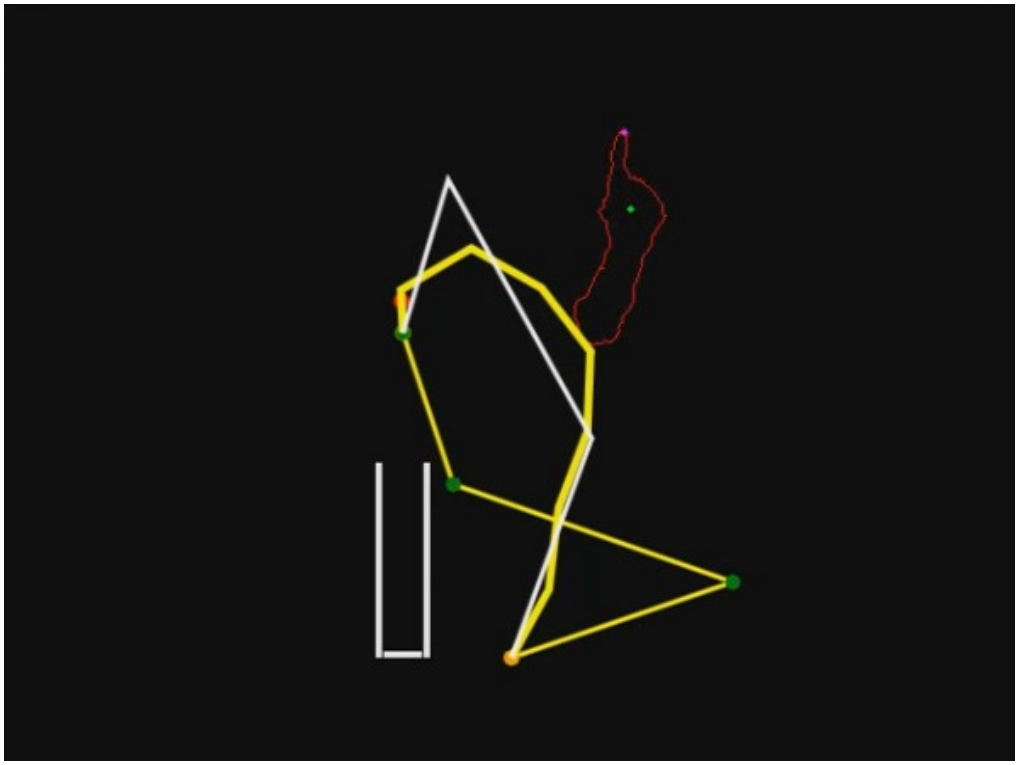


(c)

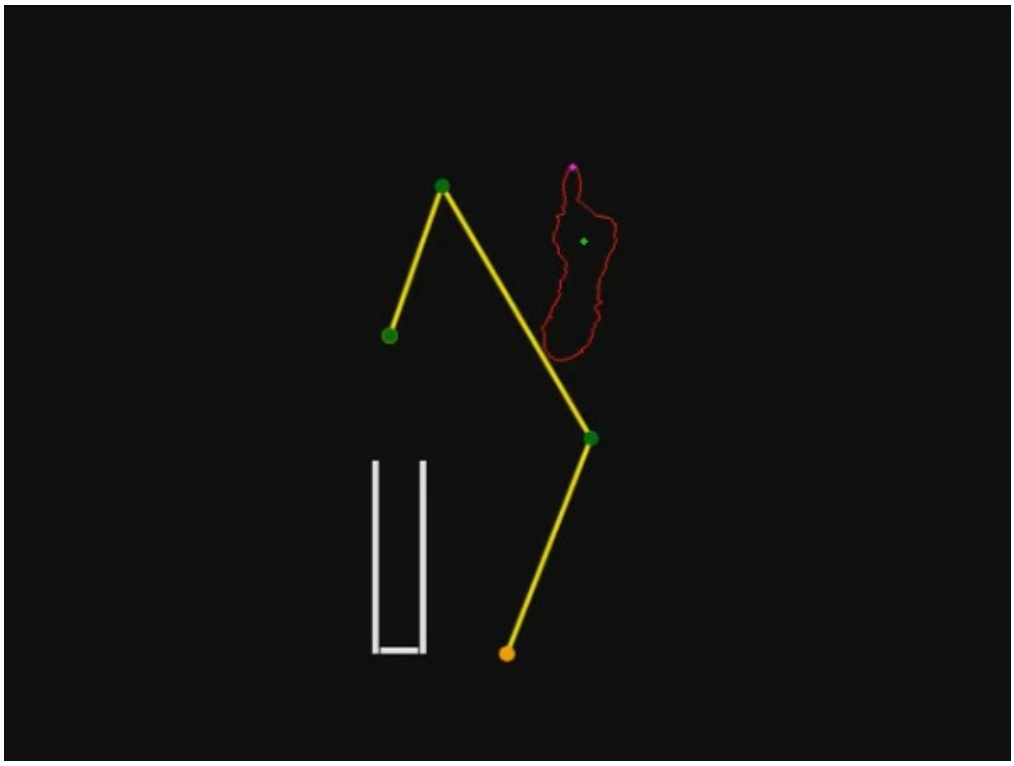
Figure 46 Tall glass test with the Attraction Point method: (a) Position the EE over the glass; timer starts; (b) Adjust the attraction point direction while moving the EE toward the deepest reachable point inside the glass, and (c) End-effector reaches the deepest reachable point the virtual glass. The timer stops.



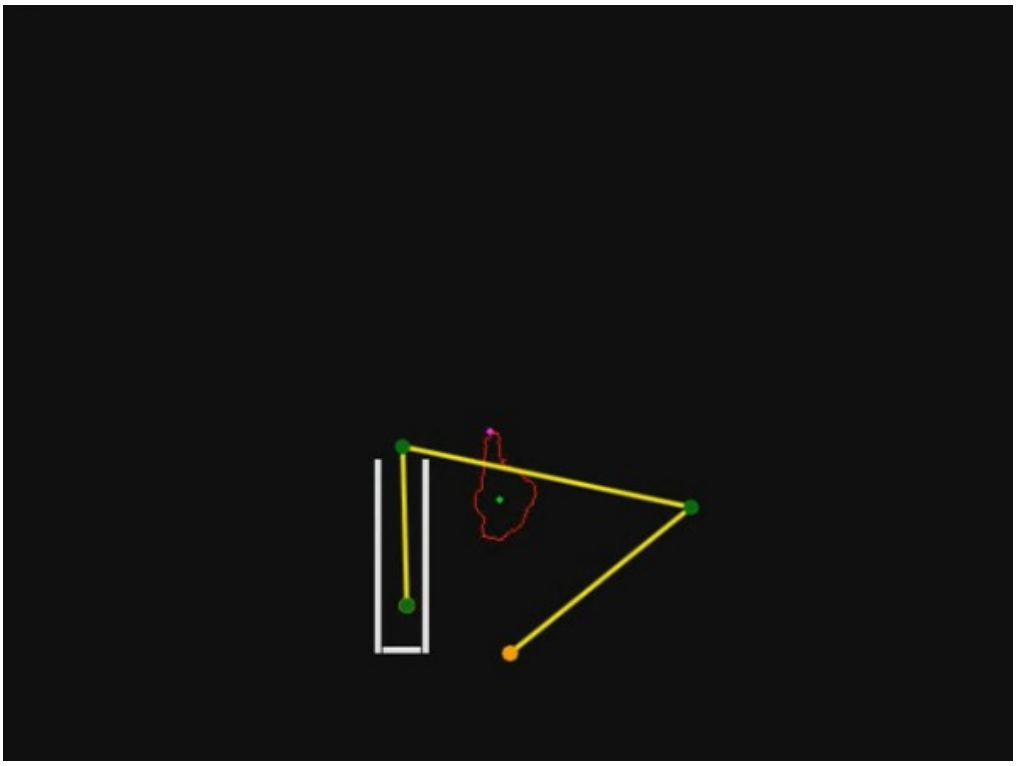
(a)



(b)



(c)



(d)

Figure 47 Tall glass test with the Tracing method: (a) Position the EE over the glass; timer starts; (b) Trace the desired posture for the simulated robotic arm; (c) Confirm the suggested posture solution and move the EE toward the deepest reachable point inside the glass, and (d) End-effector reaches the deepest possible point inside the virtual glass. The timer stops.

For each posture control method, 15 attempts to reach the maximum possible depth inside the glass (based on the arm configuration) were conducted. The performance results evaluating the success rate and average time of execution are presented in TABLE XIX, TABLE XX and TABLE XXI for respectively the Browse and Select, Attraction point and Tracing methods. The no posture control (EE control only) was also tested, but all 15 attempts failed.

TABLE XIX Tall glass test results with the Browse and Select method.

Test no	Attempt Successful?	Elapsed time (sec)
1	Yes	67
2	Yes	16
3	Yes	32
4	Yes	58
5	Yes	17
6	Yes	18
7	Yes	16
8	Yes	29
9	Yes	61
10	Yes	22
11	Yes	24
12	Yes	31
13	Yes	18
14	Yes	30
15	Yes	10
	Success rate =100%	Average time = 29.93

TABLE XX Tall glass test results with the Attraction point method.

Test no	Attempt Successful?	Elapsed time (sec)
1	Yes	21
2	No	32
3	No	19
4	No	16
5	Yes	16
6	Yes	19
7	Yes	74
8	No	17
9	No	9
10	No	21
11	Yes	11
12	Yes	10
13	Yes	11
14	No	12
15	No	18
	Success rate = 46.66%	Average time (Successful attempts only) = 20.4

TABLE XXI Tall glass test results with the Tracing method.

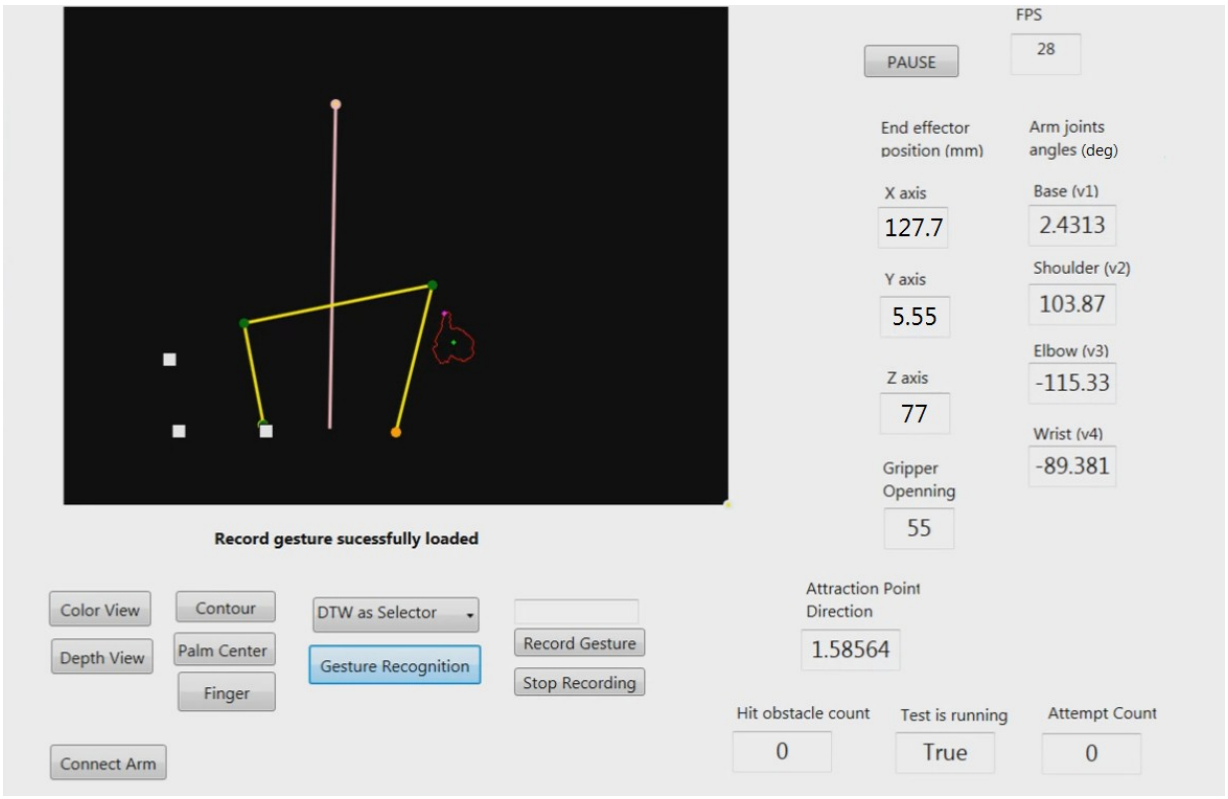
Test no	Attempt Successful?	Elapsed time (sec)
1	Yes	38
2	Yes	38
3	Yes	39
4	Yes	33
5	Yes	32
6	Yes	49
7	Yes	39
8	Yes	43
9	Yes	37
10	Yes	26
11	Yes	78
12	Yes	37
13	Yes	37
14	Yes	37
15	Yes	28
	Success rate = 100%	Average time = 39.4

5.5 Simulated dynamic obstacle avoidance – Moving block

Similar to the experiments conducted in the last section, a simulated AL5D robotic arm with no joint angle constraints was used to complete the tasks. In this experiment, a virtual block is repeatedly moving vertically and horizontally in a configuration inspired by the experiment carried out in [7]. In [7], the authors tested the performance of their solution with moving obstacles while attempting to execute a simple task like drawing a line or an eight-shape figure. For this experiment, the speed of displacement of the obstacle was set to 1 pixel/100ms, which is equivalent to 1cm/s when mapped to real world units. In this scenario, the goal is to pick an object with the end-effector in recipient A, to drop it in recipient B and to return to recipient A without any part of the robotic arm touching the moving obstacle (block) as illustrated in Figure 48. The obstacle (moving block) starts at the top of the vertical path (represented by an orange line), moves down the entire length of the vertical path, goes up a little then goes left and right on the horizontal path before returning up to its initial position. This movement is then repeated over and over.


Attraction Point
End effector
Path of the moving obstacle
Base joint
Recipient A (fixed)
Moving obstacle
Recipient B (fixed)
Record gesture sucessfully loaded
Color View
Contour
DTW as Selector
Depth View
Palm Center
Gesture Recognition
Record Gesture
Stop Recording
Finger
Connect Arm
PAUSE
FPS
29
End effector position (mm)
Arm joints angles (deg)
X axis
244.5
Y axis
50.3
Z axis
153.4
Gripper Openning
55
Base (v1)
8.7382
Shoulder (v2)
82.505
Elbow (v3)
-79.354
Wrist (v4)
-48.013
Attraction Point Direction
1.75401
Hit obstacle count
0
Test is running
False
Attempt Count
0

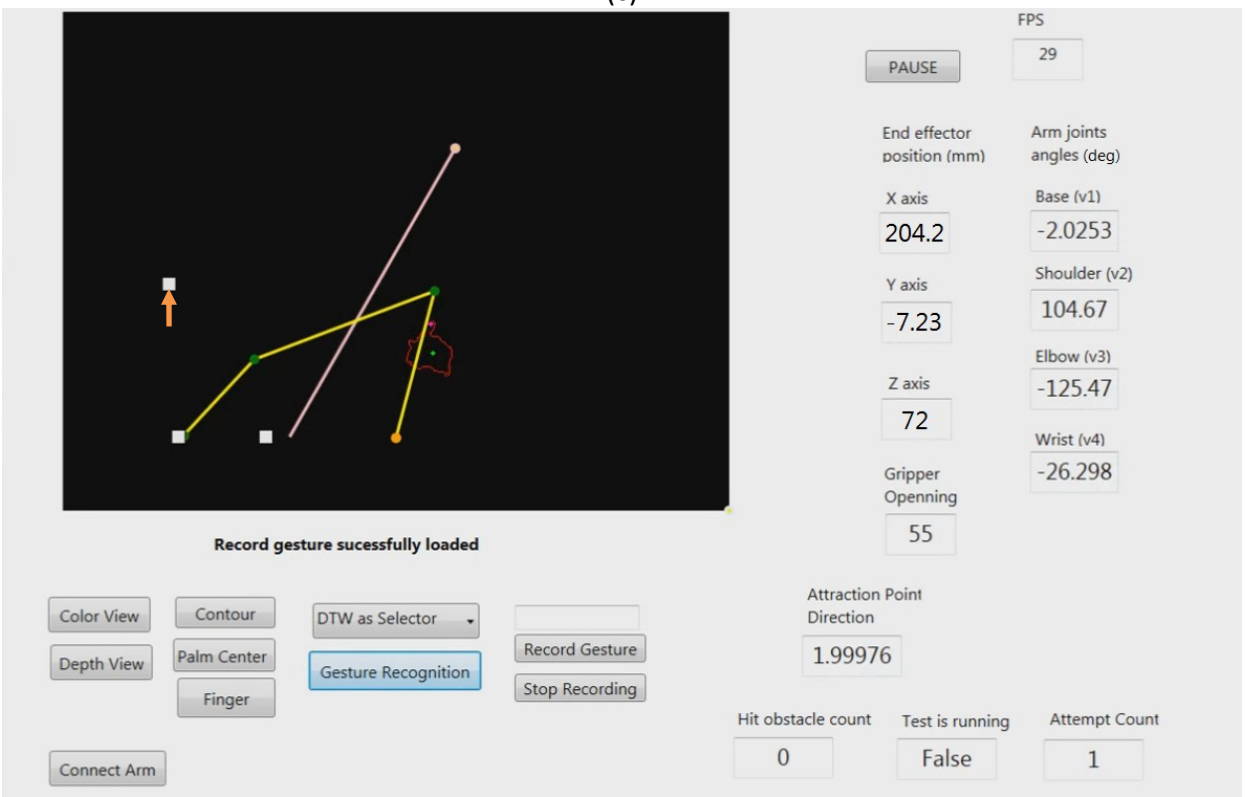
(a)


Record gesture sucessfully loaded
Color View
Contour
DTW as Selector
Depth View
Palm Center
Gesture Recognition
Record Gesture
Stop Recording
Finger
Connect Arm
PAUSE
FPS
28
End effector position (mm)
Arm joints angles (deg)
X axis
127.7
Y axis
5.55
Z axis
77
Gripper Openning
55
Base (v1)
2.4313
Shoulder (v2)
103.87
Elbow (v3)
-115.33
Wrist (v4)
-89.381
Attraction Point Direction
1.58564
Hit obstacle count
0
Test is running
True
Attempt Count
0

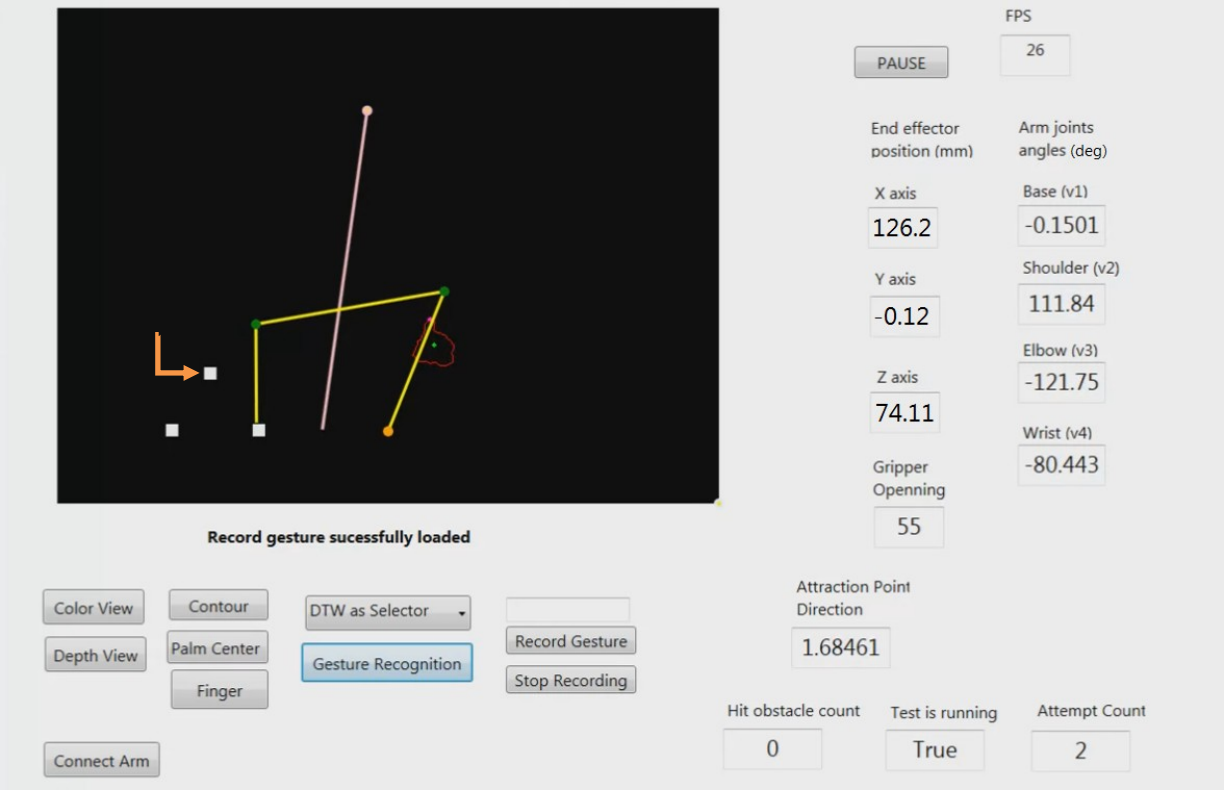
(b)


FPS
27
PAUSE
End effector position (mm)
Arm joints angles (deg)
X axis
171.7
Base (v1)
5.1478
Y axis
16
Shoulder (v2)
124.50
Z axis
108.2
Elbow (v3)
-128.22
Wrist (v4)
-40.909
Gripper Openning
55
Record gesture sucessfully loaded
Attraction Point Direction
2.12558
Color View
Contour
DTW as Selector
Record Gesture
Depth View
Palm Center
Gesture Recognition
Stop Recording
Finger
Connect Arm
Hit obstacle count
Test is running
Attempt Count
0
True
0

(c)


FPS
29
PAUSE
End effector position (mm)
Arm joints angles (deg)
X axis
204.2
Base (v1)
-2.0253
Y axis
-7.23
Shoulder (v2)
104.67
Z axis
72
Elbow (v3)
-125.47
Wrist (v4)
-26.298
Gripper Openning
55
Record gesture sucessfully loaded
Attraction Point Direction
1.99976
Color View
Contour
DTW as Selector
Record Gesture
Depth View
Palm Center
Gesture Recognition
Stop Recording
Finger
Connect Arm
Hit obstacle count
Test is running
Attempt Count
0
False
1

(d)


FPS
26
PAUSE
End effector position (mm)
Arm joints angles (deg)
X axis
126.2
Base (v1)
-0.1501
Y axis
-0.12
Shoulder (v2)
111.84
Z axis
74.11
Elbow (v3)
-121.75
Wrist (v4)
-80.443
Gripper Openning
55
Record gesture sucessfully loaded
Attraction Point Direction
1.68461
Color View
Contour
DTW as Selector
Record Gesture
Depth View
Palm Center
Gesture Recognition
Stop Recording
Finger
Connect Arm
Hit obstacle count
Test is running
Attempt Count
0
True
2

(e)


FPS
26
PAUSE
End effector position (mm)
Arm joints angles (deg)
X axis
114.3
Base (v1)
2.3295
Y axis
5.88
Shoulder (v2)
130.29
Z axis
164.8
Elbow (v3)
-106.05
Wrist (v4)
-88.579
Gripper Openning
55
Record gesture sucessfully loaded
Attraction Point Direction
1.65319
Color View
Contour
DTW as Selector
Record Gesture
Depth View
Palm Center
Gesture Recognition
Stop Recording
Finger
Connect Arm
Hit obstacle count
Test is running
Attempt Count
0
True
2

(f)


FPS
25
PAUSE
End effector position (mm)
Arm joints angles (deg)
X axis
221.1
Base (v1)
-6.8065
Y axis
-26.12
Shoulder (v2)
78.208
Z axis
132.5
Elbow (v3)
-72.822
Wrist (v4)
-90
Gripper Openning
55
Record gesture sucessfully loaded
Attraction Point Direction
1.37381
Color View
Contour
DTW as Selector
Record Gesture
Depth View
Palm Center
Gesture Recognition
Stop Recording
Finger
Connect Arm
Hit obstacle count
0
Test is running
True
Attempt Count
2

(g)

(h)

Figure 48 (a-d) Sequential illustration of the first successful attempt of transporting an object from recipient A to B and (e-h) the first encounter (hit) between the obstacle and the robotic arm during a typical test.

For each posture control method, a series of attempts to complete a task while avoiding a moving obstacle were conducted. The tests performed with the Browse and Select and Tracing methods were quickly abandoned after 15 attempts, as all attempts consistently failed at avoiding the obstacle. On the contrary, the Attraction point method resulted in 47 successful attempts. In TABLE XXII, the performance results evaluating the success rate and average time of execution are presented for the Attraction point method only.

TABLE XXII Moving block tests results with the Attraction point method.

Test no	Attempt Successful?	Elapsed time (sec)
1	Yes	10
2	Yes	7
3	Yes	17
4	Yes	8
5	Yes	7
6	Yes	12
7	Yes	12
8	Yes	11
9	Yes	20
10	Yes	6
11	Yes	12
12	Yes	14
13	Yes	9
14	Yes	12
15	Yes	10
16	Yes	7
17	Yes	10
18	Yes	7
19	Yes	10
20	Yes	6
21	Yes	23
22	Yes	8
23	Yes	6
24	Yes	6
25	Yes	6
26	Yes	5
27	Yes	8
28	Yes	17
29	Yes	8
30	Yes	10
31	Yes	11
32	Yes	9
33	Yes	17
34	Yes	8
35	Yes	6
36	Yes	5
37	Yes	22
38	Yes	8
39	Yes	13
40	Yes	10
41	Yes	8
42	Yes	12
43	Yes	6
44	Yes	6
45	Yes	7
46	Yes	11
47	Yes	7
	Success rate = 100%	Average time = 10

5.6 Real-world static obstacle avoidance – Retrieving an object through an opening

In this experiment, we tried each approach (Browse and Select, Attraction point and Tracing) to demonstrate the capability of real-time touchless natural posture control of a robotic arm. Using 2 standard 12 Kg rubber weights and 2 hockey pucks, we built a simple opening of dimension 9 x 12.5 x 2.7cm (Height x Width x Depth) between a white board marker and the AL5D Lynxmotion robotic arm. Based on the physical dimension and joint angle constraints of our AL5D, moving the robotic arm through the opening is the only way to pick up the marker. The task is to pick up the marker using the gripper through the opening and put it down vertically on the other side as illustrated in Figure 49 and Figure 50 with the Attraction point approach. At the start of the experiment, we placed the arm slightly over the obstacle. We gradually changed the posture to align the arm with the opening before moving the EE forward between the obstacles. Once the EE (gripper) reached the marker, we opened the gripper's jaw by gesturing the "V" sign with a maximum distance between the fingertips. When the gripper's jaw was wide enough to take the marker, we moved the EE forward to enclose the marker. We closed the gripper's jaw around the marker with the "V" sign using a minimum distance between the fingertips. Once the marker was grabbed by the gripper, we retrieved it to the other side of the obstacle as illustrated in Figure 49. On the other side of the obstacle, we decided to put down the marker on the left side of the robotic arm using the posture control to re-orient the gripper to have the marker vertically standing on the table (see Figure 50). Note: The posture control might be restricted to the 3 DOF planar joints $\theta 2$-$\theta 3$-$\theta 4$, but we can still use $\theta 1$ by simply mapping its value to the y axis coordinate of the end-effector (see Figure 29), which corresponds to the horizontal axis of the index fingertip position relative to the Kinect.

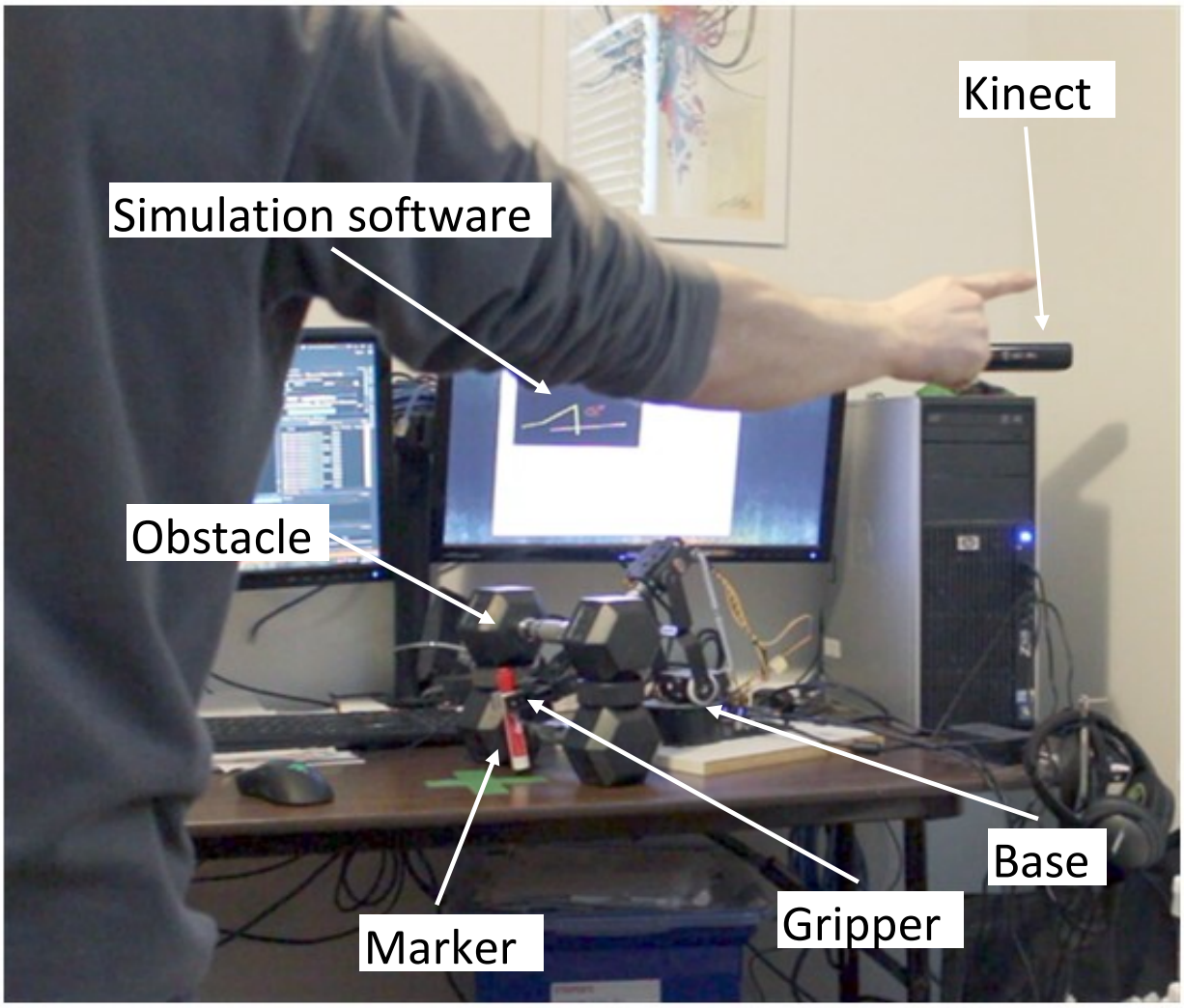


Figure 49 Real-world demonstration of real-time posture control using the attraction point method for the following use case: Retrieving an object through an opening.

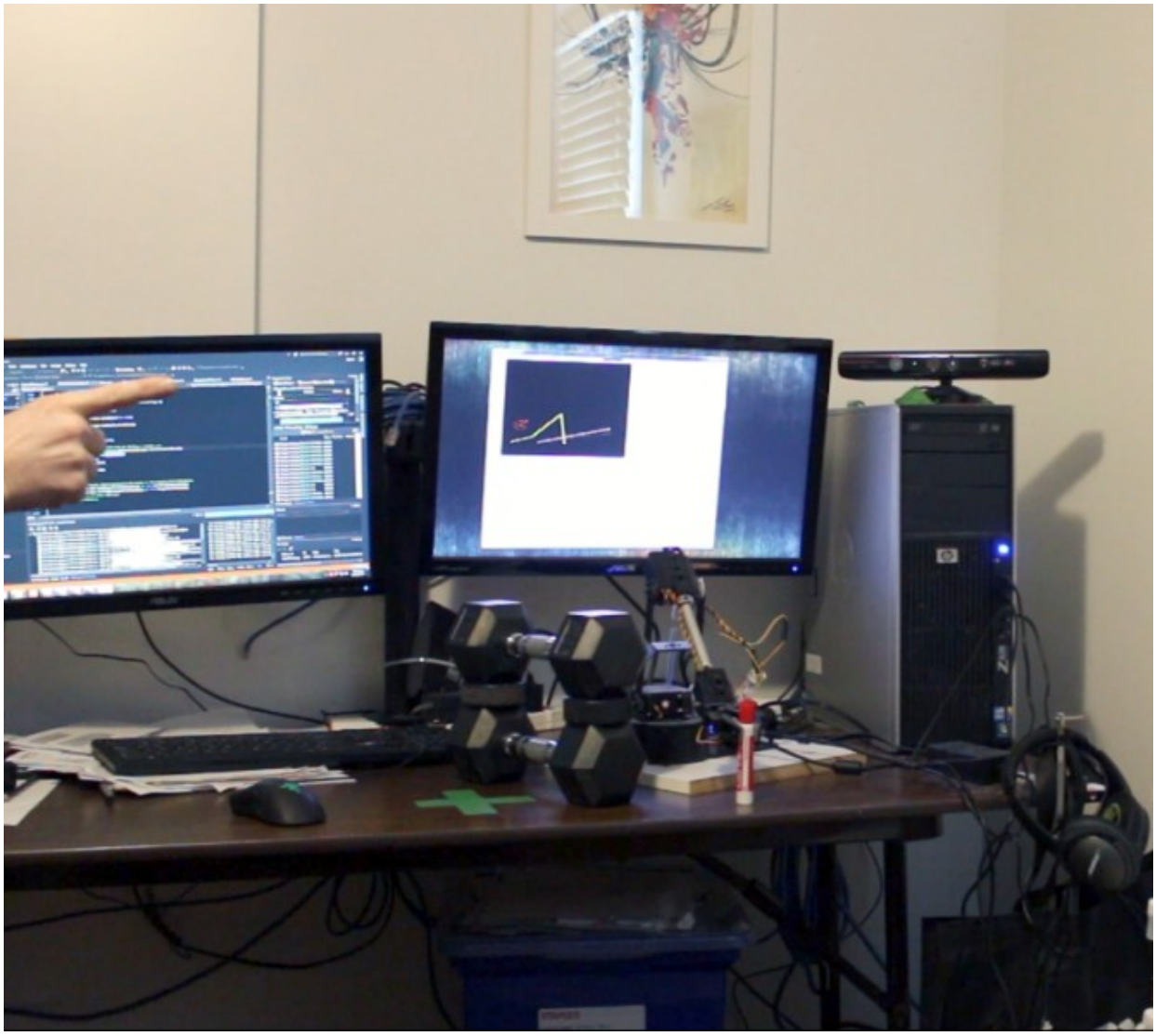


Figure 50 After going back on the other side of the obstacle through the opening with the marker inside the gripper's jaws, the user adjusts the posture of the robotic using the attraction point method arm to place the marker vertically on the table.

For each of the three posture control methods, a series of attempts to complete a pick and place task (including a series of sub tasks) were conducted. The tests performed with the No posture control, Browse and Select and Tracing method were quickly abandoned after 3 attempts. These methods were very difficult to apply while avoiding an obstacle inside a narrow space. To avoid damaging the robotic arm, the tests using these methods were stopped. We performed 15 attempts with the Attraction point method which offered a much better control when faced with the same situation while passing through the obstacle. The detailed results of the experimental validation are presented in TABLE XXIII with the success rate of each sub task.

TABLE XXIII Real-world test with the Attraction point method.

Test no	Able to reach the marker inside the gripper jaws?	Able to pick up the marker?	Able to put down the marker vertically on the right side of the obstacle?	Able to release the marker standing up?	Elapsed time (sec)
1	Yes	Yes	Yes	Yes	206
2	Yes	No	—	—	48
3	Yes	Yes	Yes	No	164
4	No	—	—	—	21
5	Yes	No	—	—	49
6	Yes	Yes	No	—	66
7	Yes	No		—	48
8	Yes	Yes	Yes	Yes	169
9	Yes	Yes	Yes	Yes	156
10	Yes	Yes	Yes	Yes	131
11	No	—	—	—	37
12	Yes	Yes	Yes	Yes	152
13	Yes	Yes	Yes	Yes	221
14	Yes	Yes	Yes	No	303
15	Yes	Yes	Yes	Yes	160
Success rate (%)	87	67	60	47	
Avg completion (sec)					170

Chapter 6 Results analysis

6.1 Coverage strategies

In Section 5.2, we investigated different strategies to obtain the best combination between time spent (computer cost) and IK output posture coverage, in terms of number of IK outputs converging toward distinct posture solutions with the use of an inputted initial postures set from the 3 DOF planar robotic arm constraints represented by angle $\vartheta 2$, $\vartheta 3$, and $\vartheta 4$ of the AL5D illustrated in Figure 29. To this end, we tried different combinations of posture coverage (varying one joint, varying all joints and varying some joints) in the input set of postures given to the IK algorithm, along with different promising IK algorithms (FABRIK and Pseudo-Inverse Jacobian). From our tests, the best combination was revealed by varying only the $\vartheta 2$ angle in the set of input postures with the FABRIK IK algorithm.

In Section 5.3, we investigated different strategies to obtain the best combination between time spent (computer cost) and matching posture accuracy given certain criteria from a set of IK output postures using the greedy best-first search approach. This search approach is intended to be used for the Attraction point posture control because each posture change increment is expected to be continuous just like the attraction point position increment. This means that for each change in the Attraction point position, the next best matching posture (in terms of distance relative to the Attraction point) is expected to be a local optima near the current posture. This strategy was not appropriate for the other two posture control approaches (Browse and Select, Tracing) since the next best matching posture is either directly selected by the user from a list of choices or based on a trace made by the user. In the former case, there is no search occurring, and in the latter, the search needs to cover the entire range of the robotic arm's workspace since the trace can correspond to any possible posture (not restricted to a local optima).

Just like the investigation in Section 5.2, different combinations of posture coverage in the input set of postures given to the IK algorithm were tested for both the FABRIK and Pseudo-

Inverse Jacobian IK algorithms. The best combination was obtained with an initial posture set varying $\vartheta 2$:$\vartheta 3$ or $\vartheta 2$:$\vartheta 3$:$\vartheta 4$ angles with the FABRIK algorithm.

6.2 Coverage strategies with different robotic arm configuration compatibility

Although the coverage strategy investigation was limited to a 3 DOF planar arm, we think that we can extend those findings to an n-link planar or three dimensional robotic arm assuming that the FABRIK algorithm remains the best choice of IK algorithm. For an n-link planar arm with joint i=0 to n, the initial posture set should always start by covering the base joint range (i=0) until the number of initial postures exceeds the number of possible step angles for that joint. If it exceeds, then the exceeding postures should start varying on joint i =1 and so on. The reasoning behind this strategy comes from Section 5.2.2.1, where the different characteristics of the FABRIK algorithm were described. The results obtained with a 3 DOF planar arm satisfy this theory.

Similarly, the results obtained from the investigation on the Greedy best-first search can also be extended to an n-link planar or/and three-dimensional robotic arm. Again, assuming that the FABRIK algorithm remains the best choice of IK algorithm, a more diversified coverage of the initial input posture by varying all joint angles would be the best approach for avoiding cases of output posture neighbourhoods with no converging solutions as encountered in Section 5.3.

6.3 Static and dynamic obstacle experiments

Although not statistically ideal, every experiment was performed by the same single user (the author) with a pre-training of 5 minutes on average by experiment and posture control method used. A more tightly constrained configuration of the robotic arm used in the experiments, in terms of speed (servos) and operational space, still needs to be implemented before being introduced to new users. For example, the robotic arm could be harmed by hitting the table surface or its control board at high speed.

In Section 5.4, we performed the tall glass test, which consists of reaching the bottom of a tall and narrow (virtual) glass with the end-effector. Overall, four series of 15 tests were carried out, each using a different method of posture control: No posture control (EE control only), Browse and Select, Attraction Point and Tracing. As expected, the no posture control gave poor results. In fact, none of our tests using this method were successful in completing the task, given a reasonable amount of time (3 min). Among the different gesture-based telerobotic systems studied in Section 2.2, the vast majority use no posture control when reaching a target with the end-effector. In the next series, we tested the first proposed Browse and Select method. As opposed to no posture control, each test was successful in reaching the bottom of the virtual glass without touching the sides of the glass. The shortest test took 10 seconds while the longest took 67 seconds. The average time was 29.93 seconds to complete each test. For the second proposed posture control approach, the Attraction point method, 7 attempts (47%) were successful in reaching the bottom without touching the sides of the virtual glass. On average, a successful attempt took 20.4 seconds to complete with a minimum of 9 seconds and maximum of 74 seconds. After compiling the results for the third proposed posture control approach, the Tracing method, we obtained a 100% success rate with a more consistent speed of execution over all tests with an average 39.40 seconds compared to the Browse and Select approach. Considering only tests where the first trace done by the user successfully led to a posture able to perform the task (93.3% of all test), the shortest time of execution was 26 seconds and longest 49 seconds. There is only one test where a second tracing needed to be performed because the first one was not adequate. For this case, the task was completed in 78 seconds.

Among the three proposed approaches, only the Attraction point method was unable to systematically reach the bottom of the glass without touching the sides. A reason for these failures might reside in the logic of Algorithm 9, where a decision is made on which IK output postures have the minimum distance based on the attraction point angle. From what we could determine, a small change in the attraction point angle could sometimes lead to bigger change in the posture than expected. The investigation performed in Section 5.3 helped mitigate this problem by minimizing the dissimilarity gap between the individual postures in the set of IK

output postures through better coverage. However, finding the next IK output posture with minimum distance relative to the attraction point is not guaranteed to be close to the current posture. Very dissimilar postures could both be close to a minimal distance for a specific attraction point angle. To decrease this possibility, we use the properties of the Greedy search algorithm to limit our search to the closest (local) optima solution in the neighborhood of the current posture. All these tactics help in obtaining a smoother transition between postures with the Attraction point method. However, based on the results of this test, there is still a need for improvement. In comparison, the control of the end-effector through direct use of the inverse kinematic (FABRIK) does not have this problem, hence the perfect results obtained with the Browse and Select and Tracing method.

In Section 5.5, we performed the moving block experiment during which a maximum number of objects were picked and dropped by a user in a limited time while avoiding a moving obstacle. The time taken between each pick was recorded. Using the Attraction point, this resulted in a total of 47 successful picks and drops with no encounters between the arm and the obstacle. On average, it took 10.92 seconds between picks with the longest being 23 seconds while the shortest was 5 seconds. Longer times were observed when the arm had to go over the obstacle, moving between the recipients, to drop the object. In comparison, using the other methods (No posture control (EE control only), Browse and Select and Tracing), the robotic arm was consistently hitting the obstacle. The other approaches were either unable to adjust its posture trajectory to avoid the obstacle (no posture control) or too slow in adjusting to the constant movement of the obstacle (Browse and Select and Tracing). Based upon those results, the Attraction point method is the only approach fast enough in controlling the posture to react in time to an environment with moving obstacles.

In Section 5.6, we performed an experiment using a real physical robotic arm (AL5D) placed in an environment with real static constraints. This time of course, constraints were applied on the robot joint angles. Using each method for posture control, the user was tasked to retrieve a pen located on the other side of an obstacle and place it at a designated location. A series of 15 attempts were executed with the Attraction point method. From the results presented in TABLE

XXIII, the gripper's jaw of the robotic arm was able to get around the marker in 87% of the tests by passing through the obstacle. The robotic arm was able to pick up the marker in 67% of the tests and put the marker down on the right side of the obstacle in 60% of the tests. In the end, 47% of the attempts were able to successfully accomplish the pick and place task by releasing the marker upright. The biggest challenge with this method was to effectively grab the marker with the gripper's jaw, as this accounts for the source of most of the failures. Overall, it took an average of 170 seconds to successfully complete the entire task. The no posture control approach method was tried but failed each time to pass through the opening after a reasonable amount of time (3 min). In all, three attempts were made with this method. The posture was often blocked by a joint reaching its angle limits. Then, three real-world demonstration attempts were performed with the other two posture control approaches: the Browse and Select and Tracing methods. At each attempt for both methods, the robotic arm failed at passing through the narrow space opening without consistently hitting the sides (obstacle). The problem is caused by a lack of key features that required further development in both the Browse and Select and Tracing methods. Indeed, in the Browse and Select approach, the user selects a possible posture from which they can accomplish a specific task or sub task without posture control.

The tall glass, moving obstacle experiment and real-world demonstration attempts revealed that constantly adjusting the posture from the selection of possible postures is a rather tedious process that is tiresome for the user. Hence, minimizing the number of selections during a task means choosing postures further apart in similarities which, in turn, become hard to determine when looking at a simulation that does not include a representation of the obstacles from the real-world environment. We do not want to accidentally choose a possible posture that goes through one of the real-world obstacles. The same problem arises with the Tracing approach. The benefit of using these approaches would be severely compromised if accomplishing a real-world experiment with these two posture control approaches requires a pre-analyzed 3D representation of the real-world environment constraints to avoid selecting or tracing a posture that goes through obstacles. A solution with an architecture type closer to supervisory control paired with an automatic obstacle avoidance system would be better suited in this case. An

alternative solution which could be an interesting project for further development would be to use our two proposed approaches with additional sensors for analyzing the environment constraints in real time.

6.4 Evaluation of the proposed approach relative to common problems found in the literature

Based on the characteristics of the proposed methods presented in Chapter 4 and the results analysis presented in this chapter, we evaluate in **TABLE** XXIV the level at which our solution addresses each of the problems or limitations presented in Section 1.2 and commonly found in other approaches in the literature.

TABLE XXIV Evaluation of the proposed solution relative to the problems and limitations presented in Section 1.2.

	Problem or limitation	Evaluations
1	Complex installation or usability	Each proposed solution requires the installation of only two components: (1) a software application, (2) a depth sensor. No calibration or adjustment is needed. They can easily be carried from site to site. The complexity of the usability varies between solutions, but since they all use natural human gestures with close to real-time posture calculation, the control of the overall posture and the end effector is both natural and intuitive. Note: Less than 5 minutes of training was conducted prior to the series of tests performed by each method in the experiments presented in Section 5.4, 5.5 and 5.6.
2	Costly in terms of resources or scalability	Each proposed solution uses a vision-based gesture recognition system which can recognize the features identification of interest (hands and fingers) of any user body type. Loose pieces of clothing around the arms and hands could however decrease the accuracy of the results.
3	Prone to errors due to complexity or design	Each proposed solution only uses one sensor, which reduces the source of errors to a minimum. The accuracy of the result is not subject to drifting errors as no calibration is needed. The user is not in contact with the sensor or any type of trackers. Consequently, errors due to

		undesired displacements are not an issue as opposed to solutions like the exoskeleton, data glove or marker system.
4	Restricted compatibility in terms of type of robotic arm configuration	As discussed in Section 6.2, the use of the FABRIK algorithm and a global posture control approach (instead of specific joints) enables our proposed solutions to be extended to more complex robotic arm solutions.
5	Restricted applications due to limited control	Each proposed solution enables the user to control both the EE position and the posture of the robotic arm. This is the highest level of control over a robotic arm. Hence the proposed solutions are not restricted in terms of application due to limited control.
6	Tedious, unnatural or non intuitive control	Each proposed solution uses natural human gestures with close to real-time posture calculation; the control of the overall posture and the end effector is both natural and intuitive. However, the manual selection of posture used in the Browse and Select and Tracing methods is tedious to perfom when continuous posture changes are required. The Attraction point offers a continuous posture change control by the user's index finger direction. This approach allows for easy and simple posture changes while reducing the repetitive strain of performing this task to a minimum. Moving the direction of one finger is indeed a lot less tiresome than moving an arm or a combination of fingers.
7	Requirement of regular calibration	None of the proposed solutions involve calibration.
8	Invasive to the user	All proposed solutions are contactless. The user does not wear any type of sensors or trackers.

Chapter 7 Conclusion

7.1 Summary

The three posture control solutions proposed in this thesis were able to satisfy to some degree the common problems and limitations presented in Section 1.2. Indeed, by using a vision recognition system instead of a mechanical device commonly found in master slave system, our solutions are easier to install and use. They are more scalable than solutions involving exoskeletons or other wearables that might be difficult to have in a one-size-fits-all device with consistent results. They require no calibration since all positions are directly mapped to the joints space instead of being controlled through incremental movement commands that might be subject to drifting problems. The proposed solutions are touchless and use only one sensor (Kinect IR camera) which helps decrease errors due to bad sensors or sensor displacement. However, vision-based recognition systems usually bring other kinds of errors like occultation or false positive/negative recognition. A lot of efforts have been included in our solutions to avoid or minimize those issues with distinct field of view workspace for two-hand gesture cases and filters to smooth sensor data and detect data outliers.

Although we limited most of our experiments on the proposed three posture control solutions to a three-link planar robotic arm and demonstrated a 3D posture control on the AL5D, we see no reason for them not to be scalable to a 3D arm with n joints. Three-dimensional robotic arms using other types of joints such as spherical, prismatic or Hooke's joints can equally potentially be considered since the FABRIK IK algorithm can be applied to such configurations according to [39] and implemented in [42]. This scalability is possible because the user controls the overall posture of the robotic arm, not specific joints. Of course, the more complex the robotic arm configurations, the higher each IK calculation cost. However, we were able to control the posture of a real 3D robotic arm in real-time using modest computer equipment.

By adding a control over the posture, our solutions open a broader range of applications, for robotic arms not restricted to pick and place tasks [28] or unconstrained environments

[8][9][10][11][33][35]. The Attraction point method requires only one hand, opening new application opportunities to use the second hand for controlling a second remote robotic arm such as in complex medical operations. Relying on a vision-based human gesture recognition system, our three posture control solutions can be a lot more natural, intuitive and faster to use compared to approaches controlling each robotic joint independently. One downside of gesture control, however, is the amount of energy spent by the user performing long series of extensive gestures, which can rapidly be tiresome. To minimize this issue, we use very small gestures that involve more finger than arm actions.

Looking back at the objectives described in Section 1.4, we were able to reach most of them in the pursuit of the overall goal of successfully investigating, designing and implementing different solutions using gesture-based visual recognition to control the posture of a remote arm that overcome the many limitations listed in Section 1.2. The experiment performed in Section 5.5 with a moving obstacle revealed that only the Attraction Point posture control approach was fast enough to avoid moving objects. In the real-world experiment (Section 5.6), we used an initial posture set of varying $\vartheta 2$:$\vartheta 3$ joints with a resolution of 100:100:0 angular steps tried/(joint maximum range in radians) with the 8 neighbors exploration technique. For this configuration, from TABLE XVIII, the average time for posture control calculation is 60.4 ms (16.5 Hz), which supports real-time operation. Since the hand and finger gesture recognition system is faster to process (~30 Hz), the bottleneck is the posture control calculation with a latency that is unfortunately lower that our ideal objective of 24 Hz stated in Section 1.5

To evaluate our three different solutions, it would have been interesting to find results to compare to in the literature. From the limited work involving posture control, we only found results focused on accurately reproducing the human arm or hand movement on the remote robotic arm joints movement [20] [32] [35]. Unlike them, our approach is not restricted to robotic arm configurations similar to the human arm because it performs a control over the general posture instead of directly mapping the human to the robotic arm posture. Hence, for now, we limited our comparisons to the three solutions proposed in this thesis. We hope that

our experimental results constitute a first base reference for other authors interested in developing similar robotic arm posture control approaches.

During the real-world 3D posture control demonstrations, we realized that the Browse and Select and Tracing methods, although allowing more postures than the Attraction Point method, were not adapted for obstacles requiring continuous posture changes to navigate around. Contrary to the Attraction point, they are not only tedious processes but also perilous ones because the user has no idea whether or not a selected or traced possible posture line is passing through an obstacle. The current software simulation described in Section 5.1.2 is limited to a 2D representation of the current and possible potential posture(s) of the robotic arm and does not give any information about the environmental constraints of the physical world.

7.2 Contributions

This thesis makes the following contributions to the field of telerobotics with a focus on natural human-computer interaction, gesture-based manipulation and real-time robotic posture control:

- Design and implementation of three different real-time posture control approaches (browse and Select; Attraction point; and Tracing) that use vision-based gesture recognition and an inverse kinematic exploration approach. The combination of these approaches enables complete and natural control over the joints of the robotic arm in real time. They are all user scalable and can be applied to most existing robotic arm configurations.
- Several search strategies are proposed to optimize the speed and the exploration coverage of the generated postures in the co-domains of the inverse kinematic given a specific target position. These proposed strategies combine the use of the Greedy Best First search approach and an efficient selection of input postures that utilize the FABRIK's unique characteristics. These optimizations allow the solutions to find better matching postures to the user's desired postures during the inverse kinematic exploration, which in turn translates to smoother and more accurate posture transitions of the robotic arm.

- Adaptation and integration of a vision-based gesture recognition system to support posture control and operation of a robotic arm.

So far, this work led to one publication [90].

7.3 Future work

An interesting future work to remedy the limited telepresence given by the current software simulation on the environmental constraints would be to integrate these applications in an augmented reality environment where the possible posture lines overlay the physical real-world objects through a VR headset. Such a system would require additional sensors for analyzing the obstacle dimensions in real time. As stated above, another compelling project would be to apply our posture control approaches to a 3D robotic arm with more complex configuration. That would give a further proof on the range of compatible robotic arm configurations capability and potentially uncovered unexpected limitations when applied to our proposed posture control approaches.

References

[1] G. Niemeyer, C. Preusche and G. Hirzinger, "Telerobotics, " in Siciliano B and Khatib O (eds.) Springer Handbook of Robotics. Springer, 2008, pp. 741–757.

[2] D. Lester and H. Thronson, "Low-Latency Lunar Surface Telerobotics from Earth-Moon Libration Points," in AIAA SPACE Conference and Exposition, 2011. [DOI: 2011. 10.2514/6.2011-7341].

[3] G. Plouffe and A.-M. Cretu, "Static and Dynamic Hand Gesture Recognition in Depth Data Using Dynamic Time Warping," IEEE Trans. Instrum. and Measurement, Vol. 65, No. 2, Feb. 2016, pp. 305-316.

[4] J. S. Heunis, C. Scheffer, K. Schreve, "A user interface for a seven degree of freedom surgical robot," in 5th Robotics and Mechatronics Conference of South Africa Year, 2012, pp. 1 – 6.

[5] T. Ma, C. Fu, H. Feng, Y. Lv, "A LESS robotic arm control system based on visual feedback," in IEEE International Conference on Information and Automation, 2015, pp. 2042 – 2047.

[6] N. N. Mansor, M. H. Jamaluddin, A. Z. Shukor, C. C. Lok, "A Study of Accuracy and Time Delay for Bilateral Master-Slave Industrial Robotic Arm Manipulator System," in MATEC Web of Conferences 150, 01015, 2018.

[7] A. C. A. Dione, S. Hasegawa, and H. Mitake, "Stable posture control for planar hyper-redundant arms using selective control points," Advanced Robotics, 17 December 2017, Vol.31(23-24), p.1338-1348.

[8] A. A. Syed, X. Duan, X. Kong, M. L. Yonggui-Wang and Q. Huang, " 6-DOF Maxillofacial Surgical Robotic Manipulator Controlled By Haptic Device," in 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Daejeon, Korea, Nov. 26-29, 2012.

[9] K. Ibrahim and A. S. Ali, "Development a Force Feedback Control of Robot Manipulator," in The 2nd International Conference on Control, Automation and Robotics, 2016.

[10] D. Dhivin, J. Jose and R. R. Bhavani, "Bilateral Tele-haptic Interface for Controlling a Robotic manipulator," in International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), 2017.

[11] G. A. D. Anjos, G. A. Santos, J. A. D. Amado, J. E. S. Marques, " Haptic Glove Development for Passive Control of a Robotic Arm," in IEEE XXIII International Congress on Electronics, Electrical Engineering and Computing (INTERCON), 2016, pp. 1 – 6.

[12] M. R. Huertas, J. R. M. Romero and H. A. M. Venegas, "A Robotic Arm Telemanipulated through a Digital Glove," Electronics, Robotics and Automotive Mechanics Conference (CERMA 2007), Morelos, 2007, pp. 470-475. [doi: 10.1109/CERMA.2007.4367731]

[13] D. Gong, J. Zhao, J. Yu and G. Zuo, " Motion mapping of the heterogeneous master–slave system for intuitive telemanipulation, " in International Journal of Advanced Robotic Systems, 2018, pp. 1–9.

[14] A. Chai and E. Lim, "Conceptual design of master-slave exoskeleton for motion prediction and control," 2015 International Conference on Advanced Robotics and Intelligent Systems (ARIS), Taipei, 2015, pp. 1-4.

[15] S. Kumra, R. Saxena and S. Mehta, "Design and development of 6-DOF robotic arm controlled by Man Machine Interface," 2012 IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, 2012, pp. 1-5.

[16] R. Tadakuma, Y. Asahara, H. Kajimoto, N. Kawakami and S. Tachi, "Development of anthropomorphic multi-D.O.F master-slave arm for mutual telexistence," in IEEE Transactions on Visualization and Computer Graphics, vol. 11, no. 6, pp. 626-636, Nov.-Dec. 2005.

[17] S. Kumra, S. Mehta and R. Singh, "Development of anthropomorphic multi-D.O.F master-slave manipulator," 2013 3rd IEEE International Advance Computing Conference (IACC), Ghaziabad, 2013, pp. 1631-1635.

[18] G. S. Gupta, S. C. Mukhopadhyay, C. H. Messom and S. N. Demidenko, " Master–Slave Control of a Teleoperated Anthropomorphic Robotic Arm With Gripping Force Sensing, " in IEEE Transactions on Instrumentation and Measurement, Vol. 55, no. 6, dec. 2006.

[19] L. Barbieri, F. Bruno, A. Gallo, M. Muzzupappa and M. L. Russo, " Design, prototyping and testing of a modular small-sized underwater robotic arm controlled through a Master-Slave approach," in Ocean Engineering 158 (2018) 253–262.

[20] H. Lin, and X. Nguyen, " A manipulative instrument with simultaneous gesture and end-effector trajectory planning and controlling, " in Review of Scientific Instruments, Vol 88, no 5, May 2017, 055107.

[21] G. Vasiljevic, N. Jagodin and Z. Kovacic, " Kinect-based Robot Teleoperation by Velocities Control in the Joint/Cartesian Frames, " in 10th IFAC Symposium on Robot Control International Federation of Automatic Control September 5-7, 2012.

[22] S. Shirwalkar, A. Singh, K. Sharma and N. Singh, " Telemanipulation of an Industrial Robotic Arm using Gesture Recognition with Kinect, " in International Conference on Control, Automation, Robotics and Embedded Systems (CARE), 2013, pp. 1-6.

[23] R. K. Megalingam, D. Menon, N. Ajithkumar and N. Saboo, " Implementation of Gesture Control in Robotic Arm using Kinect Module, " in Applied Mechanics and Materials, ISSN: 1662-7482, Vol. 786, 2015, pp 378-382.

[24] H. Jiang and J. P. Wachs, " Integrated vision-based system for efficient, semi-automated control of a robotic manipulator, " in International Journal of Intelligent Computing and Cybernetics, Vol. 7, No. 3, 2014, pp. 253-266.

[25] R. Gonçalves Lins, E. Wisniewski Marcondes Gomes, M. Corazzim and A. Beaulieu, "Development and implementation of a natural interface to control an industrial hydraulic robot arm," 2015 Annual IEEE Systems Conference (SysCon) Proceedings, Vancouver, BC, 2015, pp. 766-772.

[26] M. M. F. M. Fareed, Q. I. Akram, S. B. A. Anees and A. H. Fakih, "Gesture Based Wireless Single-Armed Robot in Cartesian 3D Space Using Kinect," 2015 Fifth International Conference on Communication Systems and Network Technologies, Gwalior, 2015, pp. 1210-1215.

[27] I. Benabdallah, Y. Bouteraa, R. Boucetta and C. Rekik, "Kinect-based Computed Torque Control for lynxmotion robotic arm," 2015 7th International Conference on Modelling, Identification and Control (ICMIC), Sousse, 2015, pp. 1-6.

[28] Y. Yang, H. Yan, M. Dehghan and M. H. Ang, "Real-time human-robot interaction in complex environment using kinect v2 image recognition," 2015 IEEE 7th International

Conference on Cybernetics and Intelligent Systems (CIS) and IEEE Conference on Robotics, Automation and Mechatronics (RAM), Siem Reap, 2015, pp. 112-117.

[29] F. Marić, I. Jurin, I. Marković, Z. Kalafatić and I. Petrović, "Robot arm teleoperation via RGBD sensor palm tracking," 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, 2016, pp. 1093-1098.

[30] M. Fuad, "Skeleton based gesture to control manipulator," 2015 International Conference on Advanced Mechatronics, Intelligent Manufacture, and Industrial Automation (ICAMIMIA), Surabaya, 2015, pp. 96-101.

[31] S. Filiatrault and A. Cretu, "Human arm motion imitation by a humanoid robot," 2014 IEEE International Symposium on Robotic and Sensors Environments (ROSE) Proceedings, Timisoara, 2014, pp. 31-36.

[32] R. C. Luo, B. Shih and T. Lin, "Real time human motion imitation of anthropomorphic dual arm robot based on Cartesian impedance control," 2013 IEEE International Symposium on Robotic and Sensors Environments (ROSE), Washington, DC, 2013, pp. 25-30.

[33] H. Wu, M. Su, S. Chen, Y. Guan, H. Zhang and G. Liu, "Kinect-based robotic manipulation: From human hand to end-effector," 2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA), Auckland, 2015, pp. 806-811.

[34] W. F. Cueva C., S. H. M. Torres and M. J. Kern, "7 DOF industrial robot controlled by hand gestures using microsoft kinect v2," 2017 IEEE 3rd Colombian Conference on Automatic Control (CCAC), Cartagena, 2017, pp. 1-6.

[35] G. Du, & P. Zhang, J. Mai and Z. Li, "Markerless Kinect-Based Hand Tracking for Robot Teleoperation, " in International Journal of Advanced Robotic Systems. 9. 1. 10.5772/50093, 2012.

[36] F. Chen, J. Deng, Z. Pang, M. B. Nejad ; H. Yang and G. Yang, " Finger Angle-Based Hand Gesture Recognition for Smart Infrastructure Using Wearable Wrist-Worn Camera," in Applied Sciences, March 2018, Vol.8(3), p.369.

[37] S. Lian, Y. Han, Y. Wang, Y. Bao, H. Xiao, X. Li and N. Sun, "Dadu: Accelerating Inverse Kinematics for high-DOF robots," 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, 2017, pp. 1-6.

[38] A. Reiter, A. Müller and H. Gattringer, "On Higher Order Inverse Kinematics Methods in Time-Optimal Trajectory Planning for Kinematically Redundant Manipulators," in *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1681-1690, April 2018.

[39] A. Aristidou and J. Lasenby, " FABRIK: A fast, iterative solver for the Inverse Kinematics problem, " in Graphical Models, 2011, 73(5), 243-260.

[40] A. Aristidou, Y. Chrysanthou and J. Lasenby, "Extending FABRIK with model constraints, " in Computer Animation and Virtual Worlds, 2016, 27(1), 35-57.

[41] R. Poddighe, "Comparing FABRIK and neural networks to traditional methods in solving Inverse Kinematics, " RobotLab, Departement of Data Science and Project Engineer, Maastricht University, 2013.

[42] A. Lansley, P. Vamplew, P. Smith and C. Foale, "Caliko: An Inverse Kinematics Software Library Implementation of the FABRIK Algorithm, " in Journal of Open Research Software, 2016, 4(1), E36-e36.

[43] M. A. Y. Abdallah, M. S. Baziyed, R. Fareh and T. Rabie, "Tracking control for robotic manipulator based on FABRIK algorithm," 2018 Advances in Science and Engineering Technology International Conferences (ASET), Abu Dhabi, 2018, pp. 1-5.

[44] A. Jha, S. S. Chiddarwar and M.V. Andulkar, "Application of human arm kinematics for robot path programming using imitation, "in Proceedings of the 2015 Conference on Advances In Robotics, Article No. 15, 2015.

[45] D. Tolani, A. Goswami and N. I. Badler, "Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs, " Graphical Models, 2000, 62(5), 353-388.

[46] M. M. F. M. Fareed, Q. I. Akram, S. B. A. Anees and A. H. Fakih, "Gesture Based Wireless Single-Armed Robot in Cartesian 3D Space Using Kinect," 2015 Fifth International Conference on Communication Systems and Network Technologies, Gwalior, 2015, pp. 1210-1215.

[47] A. Dione, S. Hasegawa, and H. Mitake, "Stable posture control for planar hyper-redundant arms using selective control points, " in Advanced Robotics, 2017, 31(23-24), 1338-1348.

[48] A. R. Várkonyi-Kóczy, and B. Tusor, “Human-Computer Interaction for Smart Environment Applications Using Fuzzy Hand Posture and Gesture Models”, IEEE Trans. Instrumentation and Measurement, vol. 60, no. 5, pp. 1505-1514, 2011.

[49] Q. Chen, N. D. Georganas, and E.M. Petriu, “Hand Gesture Recognition Using Haar-Like Features and a Stochastic Context-Free Grammar”, IEEE Trans. Instrum. and Meas., vol. 57, no. 8, pp. 1562-1571, 2008.

[50] C.-C. Hsieh, and D.-H. Liou, “Novel Haar features for real-time hand gesture recognition”, J. Real Time Image Processing, vol. 10, pp. 357-370, 2015.

[51] S. Mitra, and T. Acharya, “Gesture Recognition: A survey,” IEEE Transactions on Systems, Man, and Cybernetics, pp. 311–324, 2007.

[52] J. Suarez, and R.R. Murphy, “Hand Gesture Recognition with Depth Images: A Review”, IEEE Int. Symp. Robot and Human Interactive Communication, pp. 411-417, Paris, France, 2012.

[53] F. Trapero Cerezo, 3D Hand and Finger Recognition using Kinect, Universidad de Granada, Spain, 2013.

[54] Z. Ren, J, Yuan, and Z. Zhang, “Robust Hand Gesture Recognition Based on Finger-Earth Mover’s Distance with a Commodity Depth Camera”, Proc. ACM Int. Conf. Multimedia, pp. 1093-1096, 2011.

[55] R. P. Mihail, N. Jacobs, and J. Goldsmith, “Static Hand Gesture with 2 Kinect Sensors”, Int. Conf. Image Processing, Computer Vision, and Pattern Recognition, vol. 2, pp. 911-917, 2012.

[56] D. Uebersax, J. Gall, M. Van den Bergh, and L. Van Gool, “Real-time Sign Language Letter and Word Recognition from Depth Data”, IEEE Int. Conf. Computer Vision, pp. 383-390, Barcelona, 2011.

[57] K. Kollorz, J. Penne, and J. Hornegger, “Gesture Recognition with a Time-Of-Flight Camera”, Int. Journal of Intelligent Systems Technologies and Applications, vol. 5, pp. 334-343, 2008.

[58] K. Rimkus, A. Bukis, A. Lipnickas and S. Sinkevicius, ”3D Human Hand Motion Recognition System”, IEEE Int. Conf. Human System Interaction, pp. 180-183, Sopot, Poland, 2013.

[59] M. Correa, J. Ruiz-del-Solar, R. Verschae, J. Lee-Ferng, and N. Castillo, “Real time Hand Gesture Recognition using a Range Camera“, J. Baltes et al. (Eds.), RoboCup 2009, LNAI5949, pp. 46-57, 2010.

[60] M. Van den Bergh, and L. Van Gool, “Combining RGB and ToF Cameras for Real-time 3D Hand Gesture Interaction”, IEEE Workshop on Applications of Computer Vision, pp. 66-72, Kona, 2011.

[61] D.J. Ryan, "Finger and gesture recognition with Microsoft Kinect", Master Thesis, University of Stavanger, Norway, 2012.

[62] M. Tang, “Recognizing Hand Gestures using Microsoft’s Kinect”, Univesity of Stanford, 2011.

[63] I. Oikonomidis, N. Kyriazis and A.A. Argyros, “Efficient Model-Based 3D Tracking of Hand Articulations Using Kinect”, in Proc. British Machine Vision Conference, University of Dundee, UK, 2011.

[64] Y. Zhu, and B. Yuan, “Real-Time Hand Gesture Recognition with Kinect for Playing Racing Video Games”, Int. Joint Conf. Neural Networks, pp. 3240-3246, Beijing, China, 2014.

[65] Z. Ren, J. Meng, and J.Yuan, “Depth Camera Based Hand Gesture Recognition and its Applications in Human-Computer-Interaction”, IEEE Conf. Information, Comm. and Signal Proc., pp. 1-5, 2011.

[66] Z. Ren, J. Yuan, Ji. Meng, and Z. Zhang, “Robust Part-based Hand Gesture Recognition Using Kinect Sensor”, IEEE Trans. Multimedia, vol. 15, no. 5, 2013.

[67] M. Van den Bergh, D. Carton, R. De Nijs, N. Mitsou, C. Landsiedel, K. Kuehnlenz, D. Wollherr, L. Van Gool and M. Buss, “Real-time 3D Hand Gesture Interaction with a Robot for Understanding Directions from Humans”, IEEE Int. Symp. Robot and Human Interactive Communication, pp. 357-362, Atlanta, USA, 2011.

[68] K. Otiniano-Rodriguez, and G. Camara-Chavez, “Finger Spelling Recognition from RGB-D Information using Kernel Descriptor“, IEEE Conf. Graphics, Patterns and Images, Arequipa, pp. 1-7, 2013.

[69] M. Schroder, C. Elbrechter, J. Maycock, R. Haschke, M. Botsch, and H. Ritter, “Real-Time Hand Tracking with a Color Glove for the Actuation of Anthropomorphic Robot Hands”, IEEE-RAS Int. Conf. Humanoid Robots, pp. 262-269, Osaka, Japan, 2012.

[70] H. Takimoto, S. Yoshimori, Y. Mitsukura, and M. Fukumi, “Classification of Hand Postures Based on 3D Vision Model for Human-Robot Interaction”, IEEE Int. Symp. Robot and Human Interactive Communication, pp. 292-297, Italy, 2010.

[71] Y. Li, ”Multi-Scenario Gesture Recognition Using Kinect”, IEEE Int. Conf. Computer Games, pp. 126-130, 2012.

[72] X. Liu and K. Fujimura, “Hand Gesture Recognition Using Depth Data”, IEEE Conf. Automatic Face and Gesture Recogn., pp. 529-534, 2004.

[73] N. Pugeault, and R. Bowden, “Spelling it out: Real-time ASL fingerspelling recognition“. Conf. Consumer Depth Cameras for Computer Vision, Barcelona, Spain, 2011.

[74] F. Pedersoli, S. Benini, N. Adami, and R. Leonardi, “XKin: An Open Source Framework for Hand Pose and Gesture Recognition”, Visual Computer, vol. 30, pp.1107-1122, 2014.

[75] M. R. Abid, F. Shi and E. M. Petriu, “Dynamic Hand Gesture Recognition from Bag-of-Features and Local Part Model“, IEEE Conf. Computational Intelligence and Virtual Environments for Measurement Systems and Applications, Ottawa, ON, Canada, pp. 12-17, 2014.

[76] X. Wu, C. Yang, Y. Wang, H. Li, and S. Xu, “An Intelligent Interactive System Based on Hand Gesture Recognition Algorithm and Kinect”, IEEE Int. Symp. Computational Intelligence and Design, pp. 294-298, Huangzhou, China, 2012.

[77] N.H. Dardas and N. D. Georganas, “Real-Time Hand Gesture Detection and Recognition Using Bag-of-Features and Support Vector Machine Techniques”, IEEE Trans. Instrumentation and Measurement, vol. 60, no. 11, pp. 3592-3607, 2011.

[78] M. Vanko, I. Minarik, and G. Rozinaj, “Evaluation of Static Hand Gesture Algorithms”, Int. Conf. Systems, Signals, and Image Processing, Dubrovnik, Croatia, pp. 83-86, 2014.

[79] M.R. Abid, E.M. Petriu, and E. Amjadian, "Dynamic Sign Language Recognition for Smart Home Interactive Application Using Stochastic Linear Formal Grammar", IEEE Trans. Instrumentation and Measurement, vol. 64, no. 3, pp. 596-605, 2015.

[80] J. M. Palacios, C. Sagues, E. Montijano and S. Llorente, "Human-Computer Interaction Based on Hand Gestures Using RGB-D Sensors", MDPI Sensors Journal, vol. 13, pp. 11842-11860, 2013.

[81] Z. Li, and R. Jarvis, "Real time Hand Gesture Recognition using a Range Camera", Australasian Conference on Robotics and Automation, Sydney, Australia, pp. 529-534, 2009.

[82] Y. Yao, and Y. Fu, "Contour Model based Hand-Gesture Recognition Using Kinect Sensor", IEEE Trans. Circuits and Systems for Video Technology, vol. 24, no.11, pp. 1935-1944, 2014.

[83] F. Kobayashi, and F. Kojima, "Handover Motion Based on Human Hand Posture with Hand/Arm Robot", IEEE Int. Symp. Micro-Nano Mechatronics & Human Sciences, pp. 1-6, Nagoya, Japan, 2013.

[84] P. Payeur, G.M.G. Nascimento, J. Beacon, G. Comeau, A.-M. Cretu, V. D'Aoust, M.-A. Charpentier, "Human gesture quantification: An evaluation tool for somatic training and piano performance," in Proc. IEEE Symp. Haptic Audio-Vis. Environ. Games, Dallas, TX, USA, Oct. 2014, pp. 100–105.

[85] E. Keogh and C. A. Ratanamahatana, "Exact indexing of dynamic time warping," Knowl. Inf. Syst., vol. 7, no. 3, pp. 358–386, 2005.

[86] M. Müller, "Dynamic time warping," in Information Retrieval for Music and Motion. New York, NY, USA: Springer-Verlag, 2007, pp. 69–82.

[87] Lynxmotion. "Lynxmotion SSC-32U Servo Controller Board – Electronic Guide," Lynxmotion.com. Web Site: *http://www.lynxmotion.com/s-4-electronics-guides.aspx*. [Accessed on: January 24th, 2019]

[88] Math.net. "Math.NET Numerics" numerics.mathdotnet.com. Web Site: *https://numerics.mathdotnet.com* [Accessed on: February 21th, 2019]

[89] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. "Robotics: Modelling, Planning and Control". Springer, 2009.

[90] G. Plouffe, P. Payeur and A.-M. Cretu. (2018). Real-Time Posture Control for a Robotic Manipulator Using Natural Human-Computer Interaction. Proceedings. 4. 5751. 10.3390/ecsa-5-05751, 2018.

Appendix

A. Median smoothing of the gripper opening

Algorithm 11 gives the details of the median smoothing used to decrease the variations in the measure of the distance between the user's index and middle fingertip to determine the opening of the gripper's mouth.

Algorithm 11 : Median smoothing on the gripper opening

Input:

D = Current distance between user's fingers (mm)

$\mathbf{Q_D}$= Queue of previous distances, where $\mathbf{Q_D}$.count ≤ **SF** (Smooth factor)

Output:

$\mathbf{D_S}$= New distance with smoothing applied

```
Q_D.EnQueue(D)    // Add new distance to the queue
if(Q_D.Count> SF) {
   Q_D.Dequeue()    // Remove the oldest distance in the queue
}
Q_s = Q_D.sort()     // Sort the element of the list in order of magnitude
D_S= Q_s[Q_s.Count/2]
return D_S
```

B. Details of the results obtained during the directional exploration experience

This section gives the detailed results behind some of the analysis presented in Section 5.3.

TABLE XXV Converging solution results from the experiments done in Section 5.2.3 and presented in TABLE V where a set of initial postures varying only by their ϑ2 joint angle is given as input to the FABRIK algorithm.

Posture # in initial posture set	Initial posture joints value (ϑ2, ϑ3, ϑ4) in radian	IK output posture
1	0, 0, 0	New distinct
2	0.0317322591271696, 0, 0	Already found
3	0.0634645182543393, 0, 0	Already found
4	0.0951967773815089, 0, 0	Already found
5	0.126929036508679, 0, 0	Already found
6	0.158661295635848, 0, 0	Already found
7	0.190393554763018, 0, 0	Already found
8	0.222125813890187, 0, 0	Already found
9	0.253858073017357, 0, 0	Already found
10	0.285590332144527, 0, 0	Already found
11	0.317322591271696, 0, 0	Already found
12	0.349054850398866, 0, 0	Already found
13	0.380787109526036, 0, 0	Already found
14	0.412519368653205, 0, 0	Already found
15	0.444251627780375, 0, 0	Already found
16	0.475983886907545, 0, 0	New distinct
17	0.507716146034714, 0, 0	New distinct
18	0.539448405161884, 0, 0	New distinct
19	0.571180664289053, 0, 0	New distinct
20	0.602912923416223, 0, 0	New distinct
21	0.634645182543393, 0, 0	New distinct
22	0.666377441670562, 0, 0	Already found
23	0.698109700797732, 0, 0	New distinct
24	0.729841959924901, 0, 0	New distinct
25	0.761574219052071, 0, 0	New distinct
26	0.79330647817924, 0, 0	New distinct
27	0.82503873730641, 0, 0	New distinct
28	0.85677099643358, 0, 0	New distinct
29	0.888503255560749, 0, 0	New distinct
30	0.920235514687919, 0, 0	New distinct
31	0.951967773815088, 0, 0	New distinct
32	0.983700032942258, 0, 0	Already found
33	1.01543229206943, 0, 0	New distinct
34	1.0471645511966, 0, 0	New distinct

35	1.07889681032377, 0, 0	New distinct
36	1.11062906945094, 0, 0	New distinct
37	1.14236132857811, 0, 0	New distinct
38	1.17409358770528, 0, 0	New distinct
39	1.20582584683245, 0, 0	New distinct
40	1.23755810595962, 0, 0	New distinct
41	1.26929036508679, 0, 0	New distinct
42	1.30102262421396, 0, 0	New distinct
43	1.33275488334113, 0, 0	New distinct
44	1.3644871424683, 0, 0	New distinct
45	1.39621940159547, 0, 0	New distinct
46	1.42795166072264, 0, 0	New distinct
47	1.45968391984981, 0, 0	New distinct
48	1.49141617897698, 0, 0	New distinct
49	1.52314843810414, 0, 0	New distinct
50	1.55488069723131, 0, 0	New distinct
51	1.58661295635848, 0, 0	New distinct
52	1.61834521548565, 0, 0	New distinct
53	1.65007747461282, 0, 0	New distinct
54	1.68180973373999, 0, 0	New distinct
55	1.71354199286716, 0, 0	New distinct
56	1.74527425199433, 0, 0	New distinct
57	1.7770065111215, 0, 0	New distinct
58	1.80873877024867, 0, 0	New distinct
59	1.84047102937584, 0, 0	New distinct
60	1.87220328850301, 0, 0	New distinct
61	1.90393554763018, 0, 0	New distinct
62	1.93566780675735, 0, 0	New distinct
63	1.96740006588452, 0, 0	New distinct
64	1.99913232501169, 0, 0	New distinct
65	2.03086458413886, 0, 0	New distinct
66	2.06259684326603, 0, 0	New distinct
67	2.0943291023932, 0, 0	New distinct
68	2.12606136152037, 0, 0	New distinct
69	2.15779362064754, 0, 0	New distinct
70	2.18952587977471, 0, 0	New distinct
71	2.22125813890188, 0, 0	New distinct
72	2.25299039802905, 0, 0	New distinct
73	2.28472265715622, 0, 0	New distinct
74	2.31645491628339, 0, 0	New distinct
75	2.34818717541056, 0, 0	New distinct

76	2.37991943453772, 0, 0	New distinct
77	2.41165169366489, 0, 0	New distinct
78	2.44338395279206, 0, 0	New distinct
79	2.47511621191923, 0, 0	New distinct
80	2.5068484710464, 0, 0	New distinct
81	2.53858073017357, 0, 0	New distinct
82	2.57031298930074, 0, 0	New distinct
83	2.60204524842791, 0, 0	New distinct
84	2.63377750755508, 0, 0	New distinct
85	2.66550976668225, 0, 0	New distinct
86	2.69724202580942, 0, 0	New distinct
87	2.72897428493659, 0, 0	New distinct
88	2.76070654406376, 0, 0	New distinct
89	2.79243880319093, 0, 0	New distinct
90	2.8241710623181, 0, 0	Already found
91	2.85590332144527, 0, 0	Already found
92	2.88763558057243, 0, 0	Already found
93	2.9193678396996, 0, 0	Already found
94	2.95110009882677, 0, 0	Already found
95	2.98283235795394, 0, 0	Already found
96	3.01456461708111, 0, 0	Already found
97	3.04629687620828, 0, 0	Already found
98	3.07802913533545, 0, 0	Already found
99	3.10976139446262, 0, 0	Already found
100	3.14149365358979, 0, 0	Already found

TABLE XXVI Converging solution results from the experiments done in Section 5.2.5 and presented in TABLE IXa where a set of initial postures varying by their ϑ2 and ϑ3 joint angles is given as input to the FABRIK algorithm.

Posture # in initial posture set	Initial posture joints value (ϑ2, ϑ3, ϑ4) in radian	IK output posture
1	0, -1.5707963267949, 0	No solution
2	0, -1.22173147639603, 0	No solution
3	0, -0.872666625997165, 0	No solution
4	0, -0.523601775598299, 0	No solution
5	0, -0.174536925199433, 0	No solution
6	0, 0.174527925199433, 0	New distinct
7	0, 0.523592775598299, 0	Already found
8	0, 0.872657625997165, 0	Already found
9	0, 1.22172247639603, 0	Already found
10	0, 1.5707873267949, 0	Already found
11	0.349064850398866, -1.5707963267949, 0	No solution
12	0.349064850398866, -1.22173147639603, 0	No solution
13	0.349064850398866, -0.872666625997165, 0	No solution
14	0.349064850398866, -0.523601775598299, 0	Already found
15	0.349064850398866, -0.174536925199433, 0	Already found
16	0.349064850398866, 0.174527925199433, 0	Already found
17	0.349064850398866, 0.523592775598299, 0	Already found
18	0.349064850398866, 0.872657625997165, 0	Already found
19	0.349064850398866, 1.22172247639603, 0	Already found
20	0.349064850398866, 1.5707873267949, 0	No solution
21	0.698129700797732, -1.5707963267949, 0	No solution
22	0.698129700797732, -1.22173147639603, 0	No solution
23	0.698129700797732, -0.872666625997165, 0	New distinct
24	0.698129700797732, -0.523601775598299, 0	Already found
25	0.698129700797732, -0.523601775598299, 0	Already found
26	0.698129700797732, -0.174536925199433, 0	Already found
27	0.698129700797732, 0.174527925199433, 0	Already found
28	0.698129700797732, 0.523592775598299, 0	Already found
29	0.698129700797732, 0.872657625997165, 0	New distinct
30	0.698129700797732, 1.22172247639603, 0	No solution
31	0.698129700797732, 1.5707873267949, 0	No solution
32	1.0471945511966, -1.5707963267949, 0	No solution
33	1.0471945511966, -1.22173147639603, 0	Already found
34	1.0471945511966, -0.872666625997165, 0	Already found

35	1.0471945511966, -0.523601775598299, 0	New distinct
36	1.0471945511966, -0.174536925199433, 0	New distinct
37	1.0471945511966, 0.174527925199433, 0	New distinct
38	1.0471945511966, 0.523592775598299, 0	No solution
39	1.0471945511966, 0.872657625997165, 0	No solution
40	1.0471945511966, 1.22172247639603, 0	No solution
41	1.0471945511966, 1.5707873267949, 0	No solution
42	1.39625940159546, -1.5707963267949, 0	New distinct
43	1.39625940159546, -1.22173147639603, 0	New distinct
44	1.39625940159546, -0.872666625997165, 0	New distinct
45	1.39625940159546, -0.523601775598299, 0	New distinct
46	1.39625940159546, -0.174536925199433, 0	No solution
47	1.39625940159546, 0.174527925199433, 0	No solution
48	1.39625940159546, 0.523592775598299, 0	No solution
49	1.39625940159546, 0.872657625997165, 0	No solution
50	1.39625940159546, 1.22172247639603, 0	No solution
51	1.39625940159546, 1.5707873267949, 0	No solution
52	1.74532425199433, -1.5707963267949, 0	No solution
53	1.74532425199433, -1.22173147639603, 0	No solution
54	1.74532425199433, -0.872666625997165, 0	No solution
55	1.74532425199433, -0.523601775598299, 0	New distinct
56	1.74532425199433, -0.174536925199433, 0	No solution
57	1.74532425199433, 0.174527925199433, 0	No solution
58	1.74532425199433, 0.523592775598299, 0	No solution
59	1.74532425199433, 0.872657625997165, 0	No solution
60	1.74532425199433, 1.22172247639603, 0	No solution
61	1.74532425199433, 1.5707873267949, 0	No solution
62	2.0943891023932, -1.5707963267949, 0	No solution
63	2.0943891023932, -1.22173147639603, 0	No solution
64	2.0943891023932, -0.872666625997165, 0	No solution
65	2.0943891023932, -0.523601775598299, 0	No solution
66	2.0943891023932, -0.174536925199433, 0	No solution
67	2.0943891023932, 0.174527925199433, 0	No solution
68	2.0943891023932, 0.523592775598299, 0	No solution
69	2.0943891023932, 0.872657625997165, 0	No solution
70	2.0943891023932, 1.22172247639603, 0	No solution
71	2.0943891023932, 1.5707873267949, 0	No solution
72	2.44345395279206, -1.5707963267949, 0	No solution
73	2.44345395279206, -1.22173147639603, 0	No solution
74	2.44345395279206, -0.872666625997165, 0	No solution
75	2.44345395279206, -0.523601775598299, 0	No solution

76	2.44345395279206, -0.174536925199433, 0	No solution
77	2.44345395279206, 0.174527925199433, 0	No solution
78	2.44345395279206, 0.523592775598299, 0	No solution
79	2.44345395279206, 0.872657625997165, 0	No solution
80	2.44345395279206, 1.22172247639603, 0	No solution
81	2.44345395279206, 1.5707873267949, 0	No solution
82	2.79251880319093, -1.5707963267949, 0	No solution
83	2.79251880319093, -1.22173147639603, 0	No solution
84	2.79251880319093, -0.872666625997165, 0	No solution
85	2.79251880319093, -0.523601775598299, 0	No solution
86	2.79251880319093, -0.174536925199433, 0	No solution
87	2.79251880319093, 0.174527925199433, 0	No solution
88	2.79251880319093, 0.523592775598299, 0	No solution
89	2.79251880319093, 0.872657625997165, 0	No solution
90	2.79251880319093, 1.22172247639603, 0	No solution
91	2.79251880319093, 1.5707873267949, 0	No solution
92	3.14158365358979, -1.5707963267949, 0	No solution
93	3.14158365358979, -1.22173147639603, 0	No solution
94	3.14158365358979, -0.872666625997165, 0	No solution
95	3.14158365358979, -0.523601775598299, 0	No solution
96	3.14158365358979, -0.174536925199433, 0	No solution
97	3.14158365358979, 0.174527925199433, 0	No solution
98	3.14158365358979, 0.523592775598299, 0	No solution
99	3.14158365358979, 0.872657625997165, 0	No solution
100	3.14158365358979, 1.22172247639603, 0	No solution
101	3.14158365358979, 1.5707873267949, 0	No solution