Proceedings of the ASME 2021
International Mechanical Engineering Congress and Exposition
IMECE2021
November 1-4, 2021, Virtual, Online

IMECE2021-68054

HIGH FIDELITY HUMAN MODELING VIA INTEGRATION OF SKELETON TRACKING FOR PREDICTIVE HRC COLLISION DETECTION

Gabriel Streitmatter
University of Florida
Gainesville, FL

Jared Flowers
University of Florida
Gainesville, FL

Gloria Wiens[1]
University of Florida
Gainesville, FL

ABSTRACT

This paper develops a predictive collision detection algorithm for enhancing safety while respecting productivity in a Human Robot Collaborative (HRC) setting that operates on outputs from a Computer Vision (CV) environmental monitor. This prediction can trigger reactive and proactive robot action. The algorithm is designed to address two key challenges: 1) outputs from CV techniques are often highly noisy and incomplete due to occlusions and other factors, and 2) human tracking CV approaches typically provide a minimal set of points on the human. This noisy set of points must be augmented to define a high-fidelity model of the human's predicted spatial and temporal occupancy. A filter is applied to decrease sensitivity of the algorithm to errors in the CV predictions. Kinematics of the human are leveraged to infer a full model of the human from a set of, at most, 18 points, and transform them into a point cloud occupying the swept volume of the human's motion. This form can then quickly be compared with a compatible robot model for collision detection. Timed tests show that creation of human and robot models, and the subsequent collision check occurs in less than 30 ms on average, making this algorithm real-time capable.

Keywords: *Predictive Collision Detection, Human Robot Collaboration, Skeleton Tracking*

NOMENCLATURE

U_i	General direction vector in i^{th} direction
u_i	i^{th} direction vector of torso frame
h_i	i^{th} direction vector of head frame
P_n	The n^{th} joint from the Skeleton Tracking SDK
BP_n	The n^{th} point on a human's boundary curve
T_b^a	Transformation matrix from system *b* to *a*

1. INTRODUCTION

Trends in Industry 4.0 show human robot collaboration (HRC) becoming ever prevalent in pushing manufacturing towards greater flexibility and intelligence. HRC can pave the way for humans and robots to work in highly collaborative teams which possess both the humans' capacity for creativity and flexibility and the robot's speed and precision. A key focus in HRC is augmenting robots' intelligence with the necessary "situational awareness" so they can safely interact with humans. For HRC to be viable for more industries, depth cameras and computer vision (CV) techniques can be used to permit workspace monitoring at a lower cost than laser scanners or motion capture systems. A challenge is fusing data from a sensor suite and augmenting this data, riddled with noise, occlusion, and sparsity, to transform it into more complete environmental data a robot can use to make decisions and ensure safety of humans.

The work presented in this paper develops a predictive collision detection algorithm to serve as a real-time watchdog process during robot execution and preemptively predict collisions with a human, enhancing safety while respecting productivity in an HRC workspace. First, a human skeleton tracking algorithm is implemented to provide live positional data of up to 18 points on a sensed human. This data is highly noisy and often missing important keypoints of the skeleton under suboptimal measurement conditions. To ensure this skeleton data is reliable for use in HRC algorithms, a filter is applied to the skeleton tracking output to ensure that only realistic and physically relevant predictions are used. Next, human dynamics and kinematics are leveraged to define boundary curves around the estimated human. By completing this process for every new frame of camera data, a sequence of boundary curves defining

[1] Corresponding author: Gloria J. Wiens, University of Florida, gwiens@ufl.edu

the human's poses at multiple time steps is defined. The boundary curves between each pose are then connected by surfaces, each populated with a point cloud including temporal information indicating the time each point is occupied. These surfaces model the human, interpolating motion of the human between sensed poses. The resulting point cloud encodes spatial and temporal data of the human's swept volume. After completing a similar process for the robot's trajectory, the two volumes can be overlaid and checked for spatial and temporal intersection, i.e., a predicted collision.

2. RELATED WORKS

In the context of HRC, one of the primary goals of integration of a suite of sensed data is to enhance the robot's situational awareness. One essential application of this situational awareness is collision detection, a necessary ability of a robot if any sort of interaction with a human is to be permitted. A fast method for collision detection is to select a representative set of human and robot poses throughout time and check them for instantaneous collision, as done in [1]. This approach is risky in that collisions can be missed if sampling frequency is not sufficient. Much work has been done to learn how to best define this sampling frequency. In [2], a Gaussian process was used to determine the reachable space of a modeled body to adaptively set the sampling frequency. This type of collision check is fast for simple models. For complex models, computational costs can be prohibitive, and collisions can still be missed.

The primary alternative to interference detection is swept volume interference. When first implemented, computation of swept volumes was too expensive for real-time application. Since then, developments have been made to ameliorate the costs. In [3], a Neural Network was trained to approximate the swept volume of an articulated body between two poses resulting in quick run times but decreasing accuracy for increasing complexity of the articulated body. Another attempt to simplify computation has been to represent bodies with a set of spheres and perform a Minkowski sum of the spheres between successive poses [4, 5, 6]. This swept volume approach has been extended to convex hulls for which distance computations are cheap. In [7], convex hulls are used to encapsulate point clouds, such that distance between the hull and other nearby convex objects can be found quickly. This approach, however, provides overly conservative estimates of the volume. In [8], spheres were used to model end effecters of a humanoid robot. As the robot moved, each sphere was projected in the direction of motion far enough that it gave the robot the lookahead required to completely stop before future collisions. The area swept out between the current position and the projected position was encompassed with a convex hull. The GJK algorithm was used to identify collisions with other convex hulls. In [9], this work was extended to model the entire humanoid robot. Detailed application of this algorithm for revolute and prismatic joints is outlined in [10]. These methods only look far enough ahead to encompass the required braking distance, which can be a complex parameter to determine. One approach for doing so is outlined in [11].

A disadvantage of using convex hulls is limitation in model accuracy of geometrically complex volumes. Also, applications with convex hulls are typically only applied to the fully known robot traversing a determined trajectory. Modeling the human with this approach is much more challenging. Point clouds are desirable, as environmental data in this form can easily be ascertained with depth sensing devices. While brute force methods in comparing sets of point clouds for collisions are computationally prohibitive, much work has been done to optimize this computation. GPU programming to remove computational load from the CPU has met successes. In [12] GPU programming was implemented on a collision checker to find collisions between voxel maps in real-time. In [13] this approach was used to perform online collision checking between an environment and a robot's precomputed swept volume in voxel form. Efficiency was improved by removing unnecessary depth points (e.g., static objects) [14]. A resource that integrates many of these approaches is the Flexible Collision Library (FCL). The FCL provides a framework that can check various forms of modeled objects, (bounding volumes, mesh models, point clouds, etc.). The FCL, however, notably performs poorly in terms of computational speed on point clouds [15].

What many of these methods omit, is the temporal aspect of motion. For swept volumes, true collision occurs when volumes intersect in locations on the respective sweeps that represent the same time. Without checking spatial and temporal information, collision detection algorithms become excessively conservative. Temporal information is often ignored when 1) sweeps represent a short enough time window, or 2) collecting, modeling, and comparing this information is computationally prohibitive. As human sensing and prediction abilities are improved by advances in computer vision and sensing, predicted motion corresponding to longer future time windows will require temporal information. Additionally, most works in this field operate on fully deterministic models, which don't typically exist in dynamic environments. Predictive collision detection methods must be designed to accommodate noisy sensor data and longer-term forecasts of this data. In this paper, long-term noisy predictions of human motion are used. An approach is proposed by which noisy minimal environmental data is received, repaired, and used to model a human with point cloud data that closely approximates the true human's motion.

3. METHODS

The proposed method for dynamic environment monitoring for the purpose of preemptively identifying collisions is an integration of multiple techniques. First, a robust CV method is used to generate basic data about the environment. The Skeleton Tracking SDK for Intel ® RealSense™ Depth Cameras Software (denoted herein ST SDK) is implemented to track key points on the human. Next, a filtering technique is applied to the data to evaluate key distances between consecutive sensed points and reject frames of data that suggested impossible orientations of the human. This filtering step is necessary to provide clean human data as input to a human motion prediction algorithm.

A key assumption of this paper is that the future locations of key points on the human for a short time horizon can be predicted. This assumption is not baseless in that already, collaborating researchers have developed a Recurrent Neural Network (RNN) that is capable of such predictions for a human head and arm [16]. Such algorithms rely on a clean and accurate real-time human skeleton to make accurate predictions. Until integration with the RNN is possible, the most recent frame read in by the ST SDK herein constitutes an emulation of predicted motion, while the previous frame is held to be the current state of the human. This assumption leaves the algorithm in this paper open to future integration with the motion prediction RNN [16].

After the human motion prediction is generated, kinematics of the tracked points on the human are leveraged to draw a minimal set of boundary curves around the human to define the human's volume at each new pose sensed by the ST SDK. Finally, a previously developed surface sweep algorithm is implemented to patch together the boundary curves at each pose with Coons patches [17]. This procedure effectively augments the information sensed from the ST SDK, transforming 18 key points on the human tracked throughout time into a temporal point cloud closely approximating the swept human volume throughout the entire runtime (Fig. 1). This point cloud can be directly used in predictive collision detection with any other temporal point cloud by locating spatio-temporal intersections.

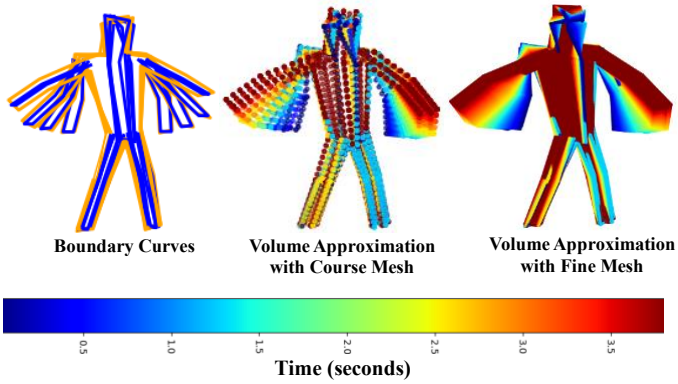


FIGURE 1: HUMAN MODELING PROCESS

3.1 Human Skeleton Tracking

The Skeleton Tracking (ST) SDK for Intel ® RealSense™ Depth Cameras software is a deep-learning based computer vision tool for full body tracking of humans. This software can perform real-time tracking of up to 18 points on the human: the ankles, feet, knees, hands, elbows, shoulders, eyes, ears, nose, torso, and hips. These key points are shown in the left side of Fig. 2 as blue dots along with the respective point nomenclature. As with all CV approaches, weaknesses exist. The primary challenge is occlusions. When key points on the human are not visible to the ST SDK, it can't track them and will inaccurately set their positions. A more difficult challenge, however, is when key points are just beginning to leave the line of sight. These points are difficult for the ST SDK to process and are often assigned highly inaccurate positions. This issue motivates the filtering technique presented in this paper. Another challenge is that, while the ST SDK can track up to 18 points, the points tracked from frame to frame can be inconsistent depending on human orientation, lighting, etc. For generation of boundary curves, a consistent number of points is necessary for continuity.

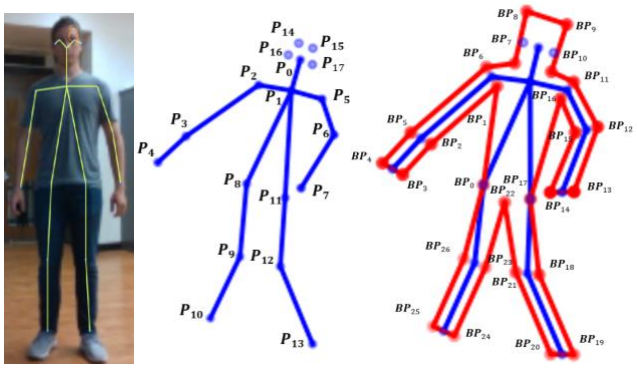


FIGURE 2: ANNOTATED DRAWING OF SKELETON WITH BOUNDARY POINTS (LABELED BY INDEX)

3.2 Filtering

As with most CV applications, error in the predicted location of sensed points drastically increased when occlusions began to occur. In these situations, the ST SDK would attempt to ignore occluded points but would run into difficulties when a point was just leaving the camera's line of sight. It would interpret these points on the human to be located far from the actual human. Such points caused the swept surface to stretch drastically to reach these unrealistic points.

These situations needed to be identified and avoided. To do this, a filter was designed to evaluate a number of key distances between points on the sensed human. The human's neck, arms, forearms, torso, thighs, and shins for each frame of data were investigated. The predicted measurements for each of these distances were compared to nominal dimensions on the tested human. If the error between these measurements was too large for any joint, the entire data frame for that instant in time was discarded. The allowable error was determined by investigating the algorithm's performance. This was done to represent a tradeoff between the desire to use only the best data to create the most accurate model, and the necessity of accepting some less-than-ideal data so that the algorithm could run in real-time.

To define this acceptable error, the ST SDK was run for a large number of poses under ideal conditions. For each pose, the distances between points were computed. The averages of these distances were compared to the actual human to ensure the data was good. The standard deviations of the measurements for each distance were used to select the acceptable error. Histograms in Fig. 3 show this analysis for five different measurements. The histograms suggest that some of the points on the extremities, such as the hands, feet, and head, had much more variability.

This variability was generally far from a normal distribution and was the result of the sensed points approaching occlusions.

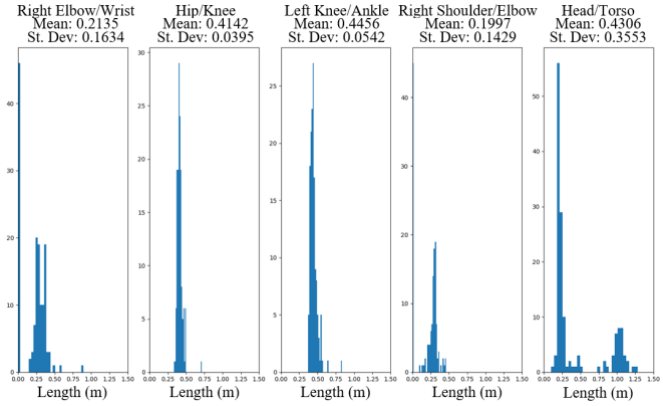


FIGURE 3: HISTOGRAMS OF BODY PART DIMENSIONS MEASURED BY THE ST SDK

In order for the algorithm to be successful, it was necessary to allow greater error tolerance for these sensed points than for some of the more reliable points, like the torso, or, as seen in Fig. 3, points on the leg. Due to points like the hands and feet, sacrifices in data quality had to be made in order to strike a tradeoff between model perfection and real-time capability. It is expected that in future works, when an additional camera is integrated from a different perspective, the number of occluded points will decrease, allowing tolerances to become tighter.

3.3 Definition of Human Boundaries

Given a good set of points on the human, the next step was to determine a boundary around the human in two orthogonal directions to approximate the human's volume. Orthogonal curves were defined such that regardless of the direction of motion of the human, some component of the curves would be orthogonal to the direction of motion. The following discussion focuses on the definition of the primary boundary curve (the one in plane with the human's orientation). This curve provides the most information about the human's occupied volume. A visual is given on the right side of Fig. 2 as a red curve along with associated nomenclature for each point. The other orthogonal curves are explained by extension of the approach used for the primary boundary curve.

First, the human's general orientation was defined as the direction orthogonal to the chest and calculated from the locations of the detected torso and waist points. This direction was selected as 1) chest orientation was usually a strong indicator of direction of motion, and 2) it relies on detection of the torso and waist, which were the most reliably detected. To establish this direction, two vectors, U_{1-8} (1) and U_{1-11} (2) were created between the torso, P_1, and the two waist points respectively, P_8 and P_{11}, where subscript indicates index value in a storage array. Then, U_{1-8} and U_{1-11} were crossed to obtain u_z, the normal vector of the torso plane (3). In this work, all direction vectors were unitized immediately after their calculation.

$$U_{1-8} = P_8 - P_1 \quad (1)$$

$$U_{1-11} = P_{11} - P_1 \quad (2)$$

$$u_z = U_{1-8} \times U_{1-11} \quad (3)$$

Next, u_x was defined as the vector going from the torso to the midpoint of the waist, P_{mp} (4-5). A third orthonormal direction was found using u_z and u_x (6). The origin of this coordinate system was placed at the torso. Finally, a transformation matrix between the camera and the torso was defined (7).

$$P_{mp} = (P_8 + P_{11})/2 \quad (4)$$

$$u_x = P_{mp} - P_1 \quad (5)$$

$$u_y = u_z \times u_x \quad (6)$$

$$T_{torso}^{camera} = \begin{bmatrix} U_x & U_y & U_z & P_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

Within this coordinate system, three boundary points were defined, in units of meters: the left and right side of the neck, BP_{10} and BP_7 respectively, and the pelvis, BP_{22}. Dimensions were selected to define general human body sizes. These points were made homogeneous and pre-multiplied by T_{torso}^{camera} to place them in the camera's coordinate system (8-10).

$$BP_7 = T_{torso}^{camera}[-0.07, -0.08, 0, 1]^T \quad (8)$$

$$BP_{10} = T_{torso}^{camera}[-0.07, 0.08, 0, 1]^T \quad (9)$$

$$BP_{22} = T_{torso}^{camera}[0.6, 0, 0, 1]^T \quad (10)$$

Next, a similar approach was taken to calculate the head boundary points. A coordinate system was established in the center of the head. To do this, a decision structure was designed to pick two appropriate detected points on the head with which to define the system because, in most cases, some head points were not detected. Five different sets, ordered by preferable choice, were considered: both eyes, both ears, the nose and left ear, the nose and right ear, or the neck points previously defined. If the nose, P_n, was not detected, it was approximated by the neck, ear, and eye points. If this approximation was not possible, the data from the current camera coordinate system was deemed insufficient and discarded for the next data frame. Next, two direction vectors were established between the nose and the torso, h_i, and between the leftmost point, P_L, and the rightmost point, P_R, (from the human's perspective), h_x (11-12). These were crossed to find the direction normal to the head, h_z (13). The h_z and h_x were crossed to obtain h_y (14). The origin of the coordinate system placed at the nose, along with the direction vectors, were used to transform boundary points defined on the head, BP_8 and BP_9, to the camera coordinate system with the transformation matrix T_{head}^{camera} (15-17).

$$h_i = P_1 - P_n \quad (11)$$

$$h_x = P_L - P_R \quad (12)$$

$$h_z = h_i \times h_x \quad (13)$$

$$h_y = h_z \times h_x \quad (14)$$

$$T_{head}^{camera} = \begin{bmatrix} h_x h_y h_z P_n \\ 0\ 0\ 0\ 1 \end{bmatrix} \quad (15)$$

$$BP_8 = T_{head}^{camera}[-0.2, 0.15, 0, 1]^T \quad (16)$$

$$BP_9 = T_{head}^{camera}[\ 0.2, 0.15, 0, 1]^T \quad (17)$$

Next, the boundary points around the arms and legs were calculated. This calculation had two flavors: calculating the boundary points for 1) a joint (shoulder, elbow or knee) and 2) an end effector (hand or foot). Only the calculations for the joints are presented since end effectors represent a simplifying special case. For each joint, P_j, first the joint's parent and child were established. The parent/child structure is outlined in section 3.4 but can be thought of as the joint's two neighboring points, P_- and P_+, which refer to the preceding and following joints in the human's kinematic chain, respectively. Direction vectors U_- and U_+ were found between the joint and its neighbors (18-19).

$$U_- = P_{j+1} - P_j \quad (18)$$

$$U_+ = P_{j-1} - P_j \quad (19)$$

These vectors were then projected into the torso plane (Fig. 4). This was necessary because two consistently orthogonal boundary curves were required. In order to establish the first direction, a common reference was needed. The torso plane was the most fitting and convenient choice to use for a reference direction since 1) it was the most likely to indicate the human's direction of motion and 2) it was formed from the most reliably sensed points. For the primary boundary curve, the in-plane direction was specified. The remaining orthogonal curves were found by dotting vectors with the normal of the torso plane. The projection of each vector, U, was computed by subtracting out the portion of the vector perpendicular to the plane of projection (20), where U' represents the vector U after projection into the plane defined by its perpendicular u_z. The U_- and U_+ were projected into the torso plane and used to determine a line of action that ran though the joint. The boundary points for the joint were to be placed along this line. The ideal placement of these points such that the two neighbor joints were accommodated was along the vector, b, that bisected U'_- and U'_+ (21). This choice is shown in the section blow up in Fig. 4.

$$U' = U - \frac{dot(U, u_z)}{\|u_z\|^2} * u_z \quad (20)$$

$$b = \frac{U'_-}{\|U'_-\|} + \frac{U'_+}{\|U'_+\|} \quad (21)$$

In the special case of the end effector, only the parent joint was needed, and rather than a bisection vector, only a vector normal to the direction to the parent projected into the torso plane was needed. This is because the boundary points on the end effector only needed to accommodate the preceding joint's boundary points. With this procedure, all remaining boundary points were calculated such that no matter the configuration of the human's joints, the boundary points were positioned such that they were indicative of the actual shape of the human.

To accommodate lateral motion of the sensed human with sets of orthogonal boundary curves, five different curves were defined: a left leg, right leg, left arm, right arm, and one curve around the torso, head, and neck. The boundary curves for the arms and legs use the same approach as above, only instead of projecting the direction vectors into the torso plane, they were dotted with the normal of the torso plane. The orthogonal boundary curve outlining the side profile of the human's torso, neck, and head was also found in the same way as done for the primary boundary curve. The only difference was that the points defined in the torso and head systems had only Z components rather than X and Y components. A series of boundary curves can be seen in Fig. 1 (left). For the ideal case, this is how each boundary point can be assigned for each set of data. This approach fails when there is/are missing joint(s) in the data. The next section outlines an approach defined to handle this issue.

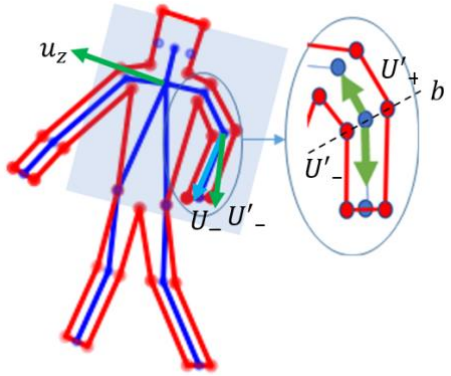
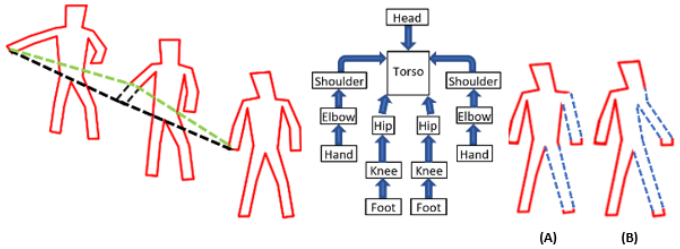


FIGURE 4: JOINT AND END EFFECTOR BOUNDARY POINTS

3.4 Occlusion Handling with Child and Parent Joints

Continuity of each tracked point on the human was vital for determination of the boundary curves and subsequent swept surface generation. This proved problematic when occlusions and other factors made it impossible for the ST SDK to assign a meaningful position to a joint. When a tracked point, the right hand for example, disappeared for a few frames, the boundary curve definition and surface sweep steps would not know how to patch together sequential poses of the human. Additionally, the boundary point definition of the missing joint's neighbors depended on determining the direction to this point. This issue is demonstrated in Fig. 5 (left), where the right hand is lost for a frame. The black dotted lines represent the connections that were expected to be made when connecting sequential poses. When this connection was not possible, the connection could not simply be terminated. If the joint showed up again, it needed to be patched to the previous pose so that the poses could be continuously connected. The green line in this image represents

the desirable best alternative connection. This solution was desirable because when a joint was lost, it was assigned to its closest neighbor, the preceding joint in the human's kinematic chain. This mapping is codified with the child parent hierarchy shown in the center of Fig. 5.



Black dashed line maps 'ideal' case (no data loss), connecting all portions of the pose. Green mapping, the next best option, is case in which portions of pose data is not detected (missing).

Blue dashed lines show mappings when parents (a) and/or grandparents (b) are not detected. The mapping method sustains the human form despite data loss.

FIGURE 5: TIME SERIES OF HUMAN POSES WITH A POSE OF MISSING DATA, MAPPING HEIARCHY, AND VISUAL OF USE OF GRANDPARENT AND GREAT GRANDPARENT MAPPINGS

Each joint was assigned a default parent. Feet were mapped to knees, and knees to the waist. Hands were mapped to elbows, and elbows to shoulders. Finally, the shoulders and waist were mapped to the torso. Grandparent and great grandparent joints were also assigned such that if a missing joint's parent was also missing, both missing joints would be mapped to the furthest existing point in the kinematic chain, the grandparent. A visual of this approach is shown on the right side of Fig. 5 for cases where a parent (a) and grandparent (b) go missing. The hierarchy was used to 1) reassign missing joint positions and 2) select the direction to a parent and child joint used to define the location of the boundary points. A child joint structure was also needed. The parent structure hierarchy was used here, except rather than work backwards along the kinematic chain, the next joint in the chain was used. Since it was more likely that joints further along in the kinematic chain would be missing, if a child could not be found, the parent joint would be located and used in place of the child.

3.5 Surface Sweep and Collision Detection with Robot

The result of the previous steps was a set of boundary curves encapsulating the human at an instance in time. This process could be iterated for an arbitrary number of time steps or predictions of future times. These boundary curves could then be patched together to create a surface that encompassed the entire swept volume of the human. This operation was completed by invoking a previously developed algorithm presented in [17]. This algorithm created a composite surface by connecting each segment of a boundary curve at one time to the corresponding segment on a boundary curve at a future time, shown in Fig. 6a and Fig. 6b. The surface was stored in the form of a point cloud containing spatial and temporal occupancy data resolved to a standardized grid spacing. The algorithm iterated this process for an arbitrary number of poses. It finally applied boundary

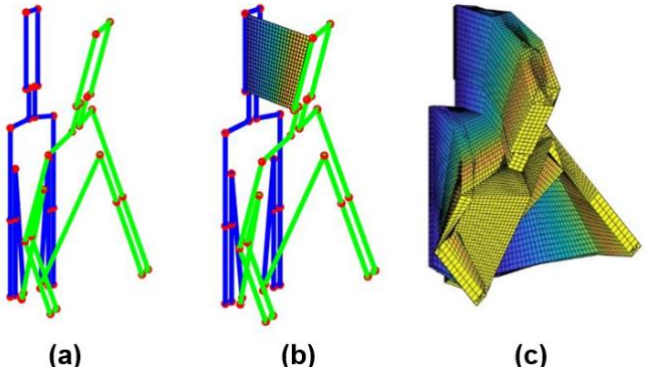


FIGURE 6: COONS PATCHES ARE GENERATED BETWEEN BOUNDARY CURVES OF THE HUMAN AT TWO TIME STEPS

conditions at the first and last pose to create patches to close off the surface. This is shown in Fig. 6c with time encoded as color. In addition to human modeling, [17] outlines application of the algorithm to modeling a robot.

4. INTEGRATION AND TESTING OF ALGORITHM

This algorithm was applied to an HRC setting in which a human and a serial link 6 DOF manipulator worked in close proximity. The overall structure of the algorithm incorporating all functions discussed in this work is as follows. First, in an offline setting, the robot's task schedule, in terms of joint trajectories, was loaded and used to generate boundary curves at a representative set of poses along the trajectories. The boundary curves were used to generate spatial and temporal point clouds. In the online portion, the sweep for each robot task was loaded sequentially. During the execution of the robot's tasks, a safety loop monitored and modeled human actions and compared them with the current robot's task trajectory to identify collisions. To emulate the output of the prediction algorithm, incorporated in the future, real-time human locations were captured using the ST SDK and then a time shift was applied so the real-time human data emulated prediction for generating the surface sweeps. The presented approach, herein, was used to generate continually updated sweeps of the human's motion, which were compared to the sweep for the robot's current task to identify collisions via spatio-temporal intersections.

The overall goals of the algorithm were to 1) precisely identify collisions between the human and robot during dynamic tasks, 2) function successfully in the face of noisy and missing data, and 3) run in real-time. The following tests were completed to evaluate the effectiveness of the algorithm with respect to these goals and establish any, and all limitations.

As an initial test to determine efficacy of the algorithm, the simulated robot was overlayed on top of the real robot from which the simulation was modeled. This was done after positioning the depth camera. The fixed coordinate system of the simulated robot was modified until it coincided with the real robot. The simulation was programmed to hold the robot at home

position for the duration of the run while the human was sensed. It was confirmed that collisions were detected whenever the human touched the robot. This process also served as calibration of the robot-camera set up. Fig. 7 shows the swept volumes for the human and robot. The larger red points indicate collisions.

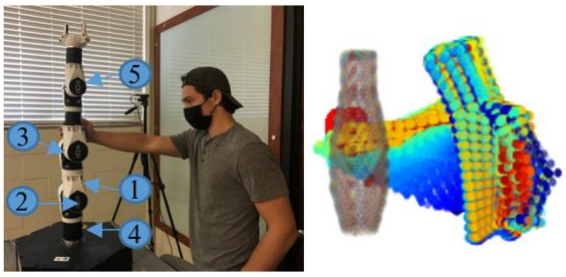


FIGURE 7: OVERLAY OF SIMULATED AND PHYSICAL ROBOT TO EVALUATE COLLISION DETECTION CAPABILITY

To understand the precision of this algorithm, it was desired to evaluate the region where the prediction of the algorithm began to dither. Dithering was defined as not making a solid prediction as to whether or not a collision was taking place and was due to variations in the perceived location of the human. The robot was set to remain still at its home position while the human moved. To evaluate this region for which dithering predictions took place, the following test was implemented. The human held a depth probe extended towards the nearest point on the robot. The human then moved their hand towards the robot until the algorithm began dithering in collision warnings. The depth probe was zeroed, and the human continued to move closer until a solid collision detection was made. The probe reading was noted. This procedure was completed for five locations on the robot (marked on Fig. 7), ten times each. Table I gives the mean and standard deviation of each location. The worst-case dither region indicates a 25.76 mm (location 4 mean plus standard deviation) thick region around the robot for which intermittent collision warnings were raised. This serves as the accuracy limit for an inexpensive collision detection system, only requiring a depth camera and processor. Increased error will yield more mis-triggers for close proximity HRC tasks.

TABLE I
PRECISION DATA FOR CAMERA PREDICTION

Location of Robot	Mean Error (mm)	St Dev Error (mm)
Location 1: Link 2	13.11	5.93
Location 2: Joint 2	18.29	4.34
Location 3: Joint 3	15.57	6.03
Location 4: Joint 1	22.28	3.48
Location 5: Joint 5	21.42	4.29

*St Dev is shorthand for standard deviation.

Next, to evaluate dynamic performance of the algorithm, robot trajectories were generated to complete a sequence of motions. Corresponding motions were planned for the human such that the human wouldn't collide with the robot. For each set of tasks, two runs were taken: one where everything went as planned, and one where the human deviated from the plan in a way that caused collision. The goal was to prove that the algorithm was 1) able to accurately identify collisions and 2) was robust to false positives in collision detection. Figs. 8-10 demonstrate either case, where the time scale on the right is the same time scale as in Fig. 1. Collisions are plotted in large red points. Each case indicated that the algorithm was effective in both identifying collisions and non-collisions accurately. Testing was not limited to this scenario, but this scenario was selected to highlight the evaluation approach.

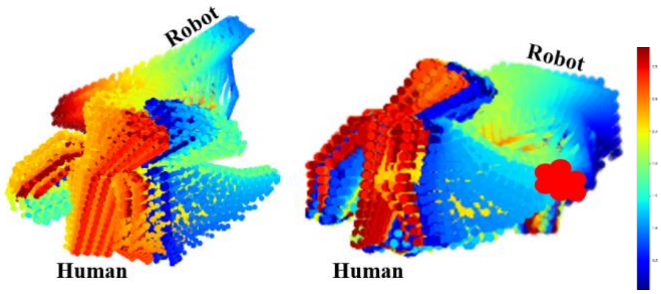


FIGURE 8: HRC TASK 1: HUMAN REACHES AROUND ROBOT WITHOUT COLLISION (LEFT) AND WITH COLLISION (RIGHT)

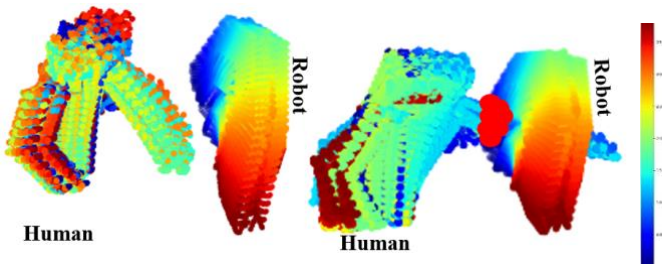


FIGURE 9: HRC TASK 2: HUMAN REACHES IN FRONT OF ROBOT WITHOUT COLLISION (LEFT) AND PAST ROBOT WITH COLLISION (RIGHT)

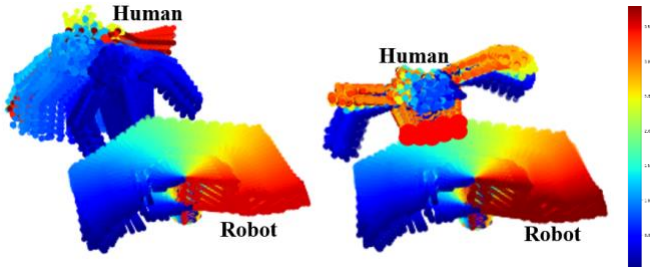


FIGURE 10: HRC TASK 3: HUMAN BACKS OUT OF THE WAY OF THE ROBOT WITHOUT COLLISION (LEFT) AND REMAINS IN THE WAY OF THE ROBOT WITH COLLISION (RIGHT)

For each scenario, execution of each submodule of the algorithm and the algorithm as a whole was timed to evaluate real-time capability. The algorithm is made up of four submodules. Submodule 1 is the ST SDK, submodule 2 is the boundary curve calculator, which is discussed in detail in section 3.3, submodule 3 is the surface sweep generator, and submodule

4 is the collision detector. Submodule 4 is discussed in detail in [17]. Fig. 11 demonstrates this time data for submodules 2 and 3 developed in this work. In the figure, the execution time for the ST SDK was not included because this module was run on a different thread and the ST SDK was a commercial product. On average, the ST SDK produced a new frame every 37 ms. The rest of the algorithm consumed data from a variable that was continuously updated by the ST SDK thread. Time data shown in Fig. 11 was typical of the execution times seen for other tests.

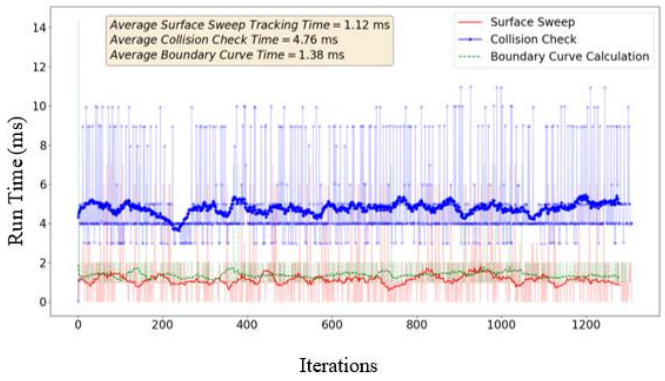


FIGURE 11: TIME DATA BY SUBMODULE. RAW DATA IS PLOTTED TRANSLUCENT. A 30 SAMPLE MOVING AVERAGE IS PLOTED BOLD

TABLE II
RUN TIME DATA FOR VARIOUS TESTS

	Boundary Curve		Surface Sweep		Collision Check		Total Time (Raw Data)		Total Time (Outliers Removed)	
Metric	Mean (ms)	St Dev (ms)	Mean (ms)	St Dev (ms)	Mean (ms)	St Dev (ms)	Mean (ms)	St Dev (ms)	Mean (ms)	St Dev (ms)
Test 1	1.39	2.75	1.59	3.16	5.44	4.49	12.60	55.27	8.69	6.39
Test 2	1.25	2.74	1.35	3.12	5.61	4.80	13.95	82.08	8.56	4.90
Test 3	1.12	2.40	1.40	2.92	5.42	4.44	14.47	59.39	8.02	4.50
Test 4	1.45	2.85	1.57	3.09	5.58	4.42	15.61	64.08	8.74	4.31
Test 5	1.44	2.84	1.35	2.89	5.94	4.61	13.79	46.81	8.81	4.58
Mean	1.33	2.72	1.45	3.04	5.60	4.55	14.08	61.53	8.56	4.94

*St Dev is shorthand for standard deviation.

Table II demonstrates time data for five runs of HRC tasks similar to the ones shown in Figs. 8-10. It provides the mean and standard deviation (St Dev) of five different tests in which both the robot and human were moving. Each reported value is taken over more than 500 iterations of the algorithm. These tests included a mix of runs with no collisions and runs with collisions. In all cases, the run's standard deviation is either close to or greater than the mean of the time data. Variation in the execution time arose from the computer's processing allocation between the algorithm and unrelated background processes. The computation time of the algorithm is low enough that the computer's fluctuation in processing power produces a noticeable effect on computation. Even still the computation time was far under the real-time threshold, which is taken to be 30 ms. The average total run time of the algorithm was typically more than sum of the submodules due to rejection of bad data by the filter. Fig. 12 shows a metric called "Frames Rejected" (right axis) along with its impact on the total run time (left axis). This metric tallied the total number of frames that had to be thrown out each iteration due to irreparable errors in the data. This was the function of the filter presented in section 3.2. Every frame rejected caused a runtime delay of, on average, 37 ms, the time required for the ST SDK to produce new data. Table II gives an estimate of what the average and standard deviation of the run time of each test would be if no frames were rejected in the Total Time (Outliers Removed) column. These values were found by removing from the "Total Time" data any recorded times greater than the mean plus three standard deviations of the raw Total Time data. Given the large relative size of the Total Time standard deviation with respect to the average total time, retaining all data within three standard deviations was a conservative estimate.

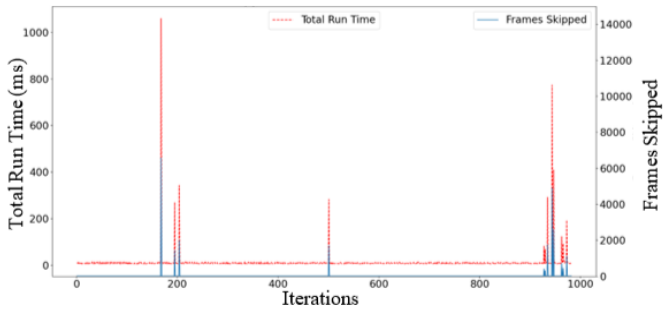


FIGURE 12: TOTAL TIME DATA OVERLAID ON THE SKIPPED FRAMES COUNT FOR EACH ITERATION

Generally, frames were rejected when the camera would begin to lose sight of a point, either due to occlusion or exiting of the sensed point from the camera line of sight. To combat this issue, attempts were first made to mend the data via the parent/child structure. If this was not possible, bad data was identified and rejected by the filter. Fig. 13 demonstrates the advantages of the parent/child mapping approach to mending the data. The output of the algorithm for a series of poses is shown on the left in red. Three instances of situations where the sensed data was highly inaccurate are shown on the right in the top row. These poses, as sensed, could not be used for construction of the model. Instead, the parent/child structure was used to map the highly inaccurate points to their closest accurate kinematic neighbor. The result of this procedure is shown on the right side of Fig. 13 on the bottom. Such repairs were able to mend a large number of poses produced by the ST SDK. Some frames, however, remained highly inaccurate even after mending. This

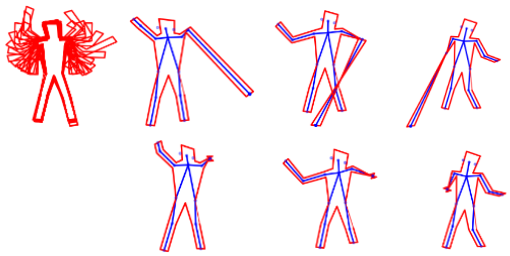


FIGURE 13: EFFECT OF THE PARENT/CHILD MAPPING

is where the filter was applied. Fig. 14 demonstrates a few examples in the same sequence of data used in Fig. 13 of poses that were too inaccurate to use for the human model. In these cases, dimensions of the human were too far from reality to create an accurate representation of the human. The parent/child structure in conjunction with the filter made the algorithm far more robust to bad data.

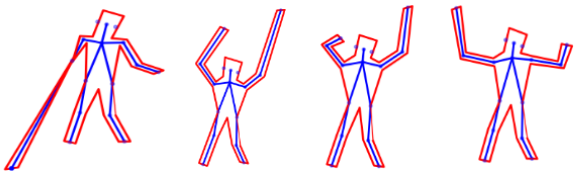


FIGURE 14: EXAMPLE FRAMES THAT WERE CAUGHT BY THE FILTER. HUMAN DIMENSIONS WERE IDENTIFIED AS NOT TRUE TO REALITY AND THUS OMMITED

Whenever the filter rejected a frame of data, the algorithm had to wait for the next frame. This was a necessary tradeoff of execution time for model fidelity. Without this, the model would be too inaccurate to make meaningful predictions. For occlusions or orientations of the human that are difficult to perceive, this could take many cycles of the ST SDK, resulting in the time spikes seen in Fig. 12. A common example is when portions of the human were occluded by the robot. It is expected that the remedy to this problem will be a second camera perspective that could augment the data that was "side profile" to the first camera. To substantiate this claim, Fig. 15 demonstrates a human orientation that has been repeated for two camera positions.

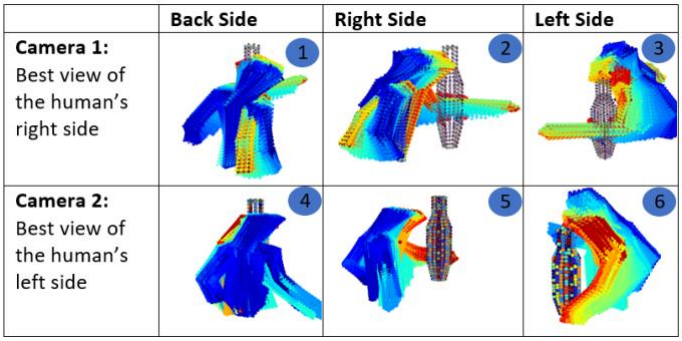


FIGURE 15: TWO CAMERA PERSPECITVES – RESULTING IN DIFFERENT TEMPORAL AND/OR SPATIAL DATA LOSS

In this scenario, the human leaned towards the robot to reach around with the left arm while moving the right arm back. Camera 1, positioned on the human's right, lost sight of the left arm as it extended behind the robot. Camera 2, positioned on the human's left, lost sight of the right arm as it fell behind the human (indicated by loss of temporal swept position data). The true motion for the right arm is shown most clearly in cells 1 and 2 of Fig. 15, while the true motion of the human's left arm can be seen most clearly in cells 5 and 6. Cell 3, which shows a distorted and stretched left arm, demonstrates Camera 1's inability to sense the human's left arm. While cells 4 and 5, which show no temporal data for the right arm during the latter half of the motion (note that the lighter color portions of the arm which indicate positions at a later time are missing), show Camera 2's inability to approximate the human's right arm. Clearly, both camera positions offer valuable and complimentary information about the human. Future work will combine these perspectives to form a higher fidelity human model.

5. CONCLUSION

Run-time tests indicate that the algorithm could filter data to avoid ST SDK error, model the human's swept volume, and check for collisions with the robot in approximately 14 ms, on average. Accuracy testing indicates that the algorithm can reliably predict collisions or non-collisions in dynamic situations, and that the region around the robot for which false positives can occur is limited to approximately 26 mm. This algorithm presents an efficient method by which a depth camera can provide reliable, real-time safety monitoring in an industrial work cell and enhance the robot's situational awareness and ability to look ahead at its trajectory and predict collisions.

Issues were noted for cases in which the human orientation caused occlusion with respect to the camera. In this case, points were highly inaccurate and had to be either repaired or filtered out, resulting in a temporary increase in run time. Future work will focus on resolving this issue by developing a sensor fusion algorithm to take data from another depth camera positioned such that its perspective is complementary to the first camera. A confidence measure will be associated with each sensed point, and between the two cameras' perspectives, the points with the greatest confidence will be fed to the rest of the algorithm.

ACKNOWLEDGEMENTS

Funding was provided by the NSF/NRI: INT: COLLAB: Manufacturing USA: Intelligent Human-Robot Collaboration for Smart Factory (Award I.D. #:1830383). Any opinions, findings and conclusions or recommendations expressed are those of the researchers and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Morato, Carlos, Kaipa, Krishnanand, Zhao, Boxuan, and Gupta, Satyandra. "Toward Safe Human Robot Collaboration by Using Multiple Kinects Based Real-Time Human Tracking." ASME. *J. Comput. Inf. Sci. Eng*. Vol. 14 No. 1 (2014) : pp. 1006-1009.
DOI: https://doi.org/10.1115/1.4025810.

[2] Casalino, Andrea, Brameri, Alberto, Zanchettin, Andrea Maria, and Rocco, Paolo. "Adaptive Swept Volumes Generation for Human-Robot Coexistence using Gaussian Processes." *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* pp. 3823-3829. Macau, China, 2019. DOI: 10.1109/IROS40897.2019.8967807.

[3] Baxter, John, Yousefi, Mohammad, Sugaya, Satomi, Morales, Marco, and Tapia, Lydia. "Deep Prediction of Swept Volume Geometries: Robots and Resolutions." *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. pp. 6665-6672. Las Vegas, NV, USA, 2020. DOI: 10.1109/IROS45743.2020.9341396.

[4] Ragaglia, Matteo, Zanchettin, Andrea Maria, and Rocco, Paolo. "Trajectory Generation Algorithm for Safe Human-Robot Collaboration Based on Multiple Depth Sensor Measurements." *Mechatronics* Vol. 55 (2018): pp. 267-281. DOI: 10.1016/j.mechatronics.2017.12.009.

[5] Casalino, Andrea. Bazzi, Davide. Zanchettin, Andrea Maria. and Rocco, Paolo. "Optimal Proactive Path Planning for Collaborative Robots in Industrial Contexts." *2019 International Conference on Robotics and Automation (ICRA)*. pp. 6540-6546. Montreal, QC, Canada, 2019. DOI: 10.1109/ICRA.2019.8793847.

[6] Ragaglia, Matteo, Zanchettin, Andrea Maria, and Rocco, Paolo. "Safety-Aware Trajectory Scaling for Human-Robot Collaboration with Prediction of Human Occupancy." *2015 International Conference on Advanced Robotics (ICAR)*. pp. 85-90. Istanbul, Turkey, 2015. DOI: 10.1109/ICAR.2015.7251438.

[7] Melchiorre, Matteo, Scimmi, Leonardo Sabatino, Pastorelli, Stefano Paolo, and Mauro, Stephano. "Collison Avoidance using Point Cloud Data Fusion from Multiple Depth Sensors: A Practical Approach." *2019 23rd International Conference on Mechatronics Technology (ICMT)*. pp. 1-6. Salerno, Italy, 2019. DOI: 10.1109/ICMECT.2019.8932143.

[8] Täubig, Holger, Bäuml, Berthold, and Frese, Udo. "Real-Time Swept Volume and Distance Computation for Self Collision Detection." *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 1585-1592. San Francisco, CA, USA, 2011. DOI: 10.1109/IROS.2011.6094611.

[9] Täubig, Holger, Bäuml, Berthold, and Frese, Udo. "Real-Time Continuous Collision Detection for Mobile Manipulators - A General Approach." *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. pp. 461-468. Osaka, Japan, 2012. DOI: 10.1109/HUMANOIDS.2012.6651560.

[10] Täubig, Holger, and Frese, Udo. "A New Library for Real-Time Continuous Collision Detection." *ROBOTIK 2012; 7th German Conference on Robotics*. pp. 1-5. Munich, Germany, 2012.

[11] Jing, Xia, Guangxin, Wu, Chongyang, Li, and Hong, Liu. "Real-Time Collision Detection for Manipulators Based on Fuzzy Synthetic Evaluation." *2016 IEEE International Conference on Mechatronics and Automation*. pp. 777-782. Harbin, China, 2016. DOI: 10.1109/ICMA.2016.7558661.

[12] Hermann, Andreas, Klemm, Sebastian, Xue, Zhixing, Roennau, Arne, and Dillmann, Rudiger. "GPU-Based Real-Time Collision Detection for Motion Execution in Mobile Manipulation Planning." *2013 16th International Conference on Advanced Robotics (ICAR)*. pp. 1-7. Montevideo, Uruguay, 2013. DOI: 10.1109/ICAR.2013.6766473.

[13] Hermann, Andreas, Drews, Florian, Bauer, Joerg, Klemm, Sebastian, Roennau Arne, and Dillmann, Ruediger. "Unified GPU Voxel Collision Detection for Mobile Manipulation Planning." *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. pp. 4154-4160. Chicago, IL, USA, 2014. DOI: 10.1109/IROS.2014.6943148.

[14] Ujkani, Ering, Eppeland, Petter, Aalerud, Atle, and Hovland, Geir. "Real-Time Human Collision Detection for Industrial Robot Cells." *2017 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. pp. 488-493. Daegu, Korea (South), 2017. DOI: 10.1109/MFI.2017.8170368.

[15] Pan, Jia, Chitta, Sachin, and Manocha, Dinesh. "FCL: A General Purpose Library for Collision and Proximity Queries." *2012 IEEE International Conference on Robotics and Automation*. pp. 3859-3866. Saint Paul, MN, USA, 2012. DOI: 10.1109/ICRA.2012.6225337.

[16] Zhang, Jianjing, Liu, Hongyi, Chang, Qing, Wang, Lihui, Gao, Robert. "Recurrent Neural Network for Motion Trajectory Prediction in Human-Robot Collaborative Assembly." *CIRP Annals* Vol. 69 No. 1 (2020) pp. 9-12. DOI: 10.1016/j.cirp.2020.04.077.

[17] Streitmatter, Gabriel, and Wiens, Gloria, "Human-Robot Collaboration: A Predictive Collision Detection Approach for Operation within Dynamic Environments." *2020 International Symposium of Flexible Automation (ISFA).* Virtual, Online. July 8-9, 2020. DOI: https://doi.org/10.1115/ISFA2020-9659

