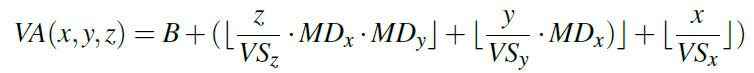
**Skeleton Swept Volume Lit Review**

1. **Trajectory generation algorithm for safe human-robot collaboration based on multiple depth sensor measurements.**
   1. Safety constraints set to ensure that the robots speed multiplied by the worst-case braking time is less than the minimum allowable separation distance.
      1. Robot and obstacle dimensions are accounted for with “clearance” which is subtracted from the minimum distance.
   2. Multiple depth sensors merged with a sensor fusion algorithm used to get 3D skeletal points position and velocity of the human.
      1. Fusion algorithm designed to be robust to sensor noise and occlusion.
      2. Implements a Linear Kalman Filter to monitor joint positions and kinematic relationships between each joint from frame to frame.
      3. Confidence levels for each sensed joint position are generated. A point is valid if its confidence is above a threshold.
   3. Predictions of the joint locations in the immediate future are made via swept volumes.
      1. Modeled human is very rough. Consists of a one DOF shape that encompasses the head, neck, torso, waist, and legs, and two four DOF shapes for each arm.
      2. One DOF shape moved around by specifying an angle and distance (polar coordinates)
      3. 4 DOF arms use a ball joint at shoulder and revolute joint at elbow
      4. It looks like they’re moving the human along a trajectory and generating the human’s general shape with convex polyhedral at a series of steps (not actually sweeping anything)
      5. Swept volumes created by sweeping convex polyhedral one body shape at a time.
      6. Collision detection consists of a minimum distance calculation.
      7. Their approach is very low resolution, approximating the human as very crude shapes and drastically limiting the human’s DOFs
2. **Adaptive swept volumes generation for human-robot coexistence using Gaussian Processes.**
   1. Extension to [1] but an attempt to use Gaussian processes to learn limiting parameters better in order to get a less conservative estimate of the human’s swept volume. This would in turn make the robot more productive and just as safe. The gaussian process (basically the same as the machine learning technique) allows model parameters to be customized to specific observed humans rather than assuming worst case scenario (most conservative) parameters.
   2. Human treated as a manipulator (model) with orientation and posture of torso and each arm determined from depth sensing cameras.
   3. The gaussian process is used to determine the reachable space around each body part in the next time step. A high delta means more space around the body part must be maintained (and is thus conservative).
   4. This approach can adapt to the human being measured to provide an appropriate amount of safety to the human depending on how unpredictable or quick their actions are.
   5. The control structure is simply to modulate robot velocity in order to avoid collisions.
   6. Run time: Mean time was 0.31 seconds with a standard deviation of 0.153 seconds.
3. **Optimal Proactive Path Planning for Collaborative Robots in Industrial Contexts**
   1. Approach is to proactively plan robot paths to achieve tasks in a way that minimizes risk to human. This is done ahead of time based on the human’s scheduled motions to avoid sub optimal reactive replans.
   2. Store a number of human actions sensed with a depth sensor. Process: read in actions, segment them, store them.
   3. During execution, reference this storage and compute the probability that the human is executing a particular action.
   4. Swept Volume to model human.
      1. Only the arms are modeled as they are the most likely to interfere with the robot.
      2. Only one are at a time is moved for an action.
      3. Swept volume between two poses is found with Minkowski sum of a sphere.
      4. Skeletal points are used to define the boundaries.
      5. Swept volume stored in an OctTree (0 if unoccupied, 1 if occupied)
      6. These volumes don’t appear to take into account time.
   5. All volume sweeps for all human actions are overlaid. The probability of the action is used to set the strength of a potential field around the volume’s occupied points.
   6. The robot optimizes its path around this aggregate potential field. STOMP is used to modify the pre-existing path and bend it around the potential field.
   7. Computation of the updated point cloud and updated robot trajectory took 1.23 seconds.
4. **Collision-free Trajectory Planning in Human-robot Interaction through Hand Movement Prediction from Vision**
   1. This paper focuses on integrating a module for tracking a collaborator and a module for planning collision free trajectories for interacting with that collaborator.
   2. CNN used to extract features and track collaborator and RNN is used to predict hand motions.
   3. This work develops 1) a perception module to predict human’s hand motion, 2) robot trajectory adaptive planning module that is robust to noisy data, and 3) integration of 1 and 2, 4) a new human manipulation dataset or motion capture data.
   4. An attention module is used to focus the camera frame on the human’s hand by cropping the frame around the hand’s color signature. These cropped images are fed into the CNN model.
   5. In training the RNN, LSTM is used to combat the vanishing gradients problem.
   6. After predictions are made, the predicted hand position and robot trajectory (in 3D) is projected onto the work table (2D surface) for collision detection. A critical distance is then calculated between robot and the hand to identify collisions.
   7. New robot trajectories are found with optimization criteria and an iterative approach.
5. **Fenceless obstacle avoidance method for efficient and safe human–robot collaboration in a shared workspace. ( DON’T CITE THIS WORK if it can be helped)**
   1. Real time prediction collision model for stationary robots and moving targets based on a cost function. Rather than replanning to avoid the human, the robot’s motion is slowed to give the human time to make a corrective action.
   2. Collision is detected between the worker and the end effector. Their positions are predicted linearly to investigate intersection (constant acceleration assumed).
   3. Everything done in 2D: an area around the robot and human at multiple time steps in predicted trajectory is drawn. The ratio of overlapping area is used to modify robot speed.
   4. Interesting cost function. They use 4 different measures of risk and scale them all by their own coefficient to calculate overall cost.
   5. Robot velocity inversely proportional to cost.
6. **Toward Safe Human Robot Collaboration by Using Multiple Kinects Based Real-Time Human Tracking**
   1. This work is centered around using multiple Kinect sensors to get the best possible understanding of the human’s pose in the work cell.
   2. Analysis is done to figure out the optimal placement of the Kinect sensors for full visibility of the human, but also minimization of the interference between multiple Kinect sensors.
   3. This work found that there was no additional benefit from placing more than four Kinect sensors in the workspace.
   4. Each joint position is found by running the Kinect data through the joint’s own Kalman filter. The outputs of the filter (the joint position) are then run through a particle filter.
   5. Results from all Kinect sensors are aggregated into a final prediction
   6. Accuracy was asses by having the human move their hand to a known position in space and seeing where the algorithm’s output predicted the hand was.
   7. Distance between the human’s joints and the robot were used to detect collision.
   8. The model of the human was only waist up.
   9. If an imminent collision was detected, the robot simply stopped.
   10. A rollout strategy was used to detect collisions. The human’s location was sensed. This instantaneous pose is checked with the robot’s projection into the future. The robot’s [projection consisted of a set of 10 poses of the robot distributed amongst 3 seconds.
   11. It only performs interference detection for motion briefly into the robot’s future (in this case as far as the braking distance). There is no prediction of human motion. A crude model of the human pose is made with spheres at an instantaneous time. This instantaneous pose is checked with the robot’s projection into the future. If the human is moving rapidly towards the robot, this could cause an issue.
7. **Real-time swept volume and distance computation for self-collision detection**
   1. THIS WORK AND THE FOLLOWING TWO WORKS ARE A GREAT SOLUTION TO A PROBLEM DIFFERENT THAN OURS. THIS SOLUTION OFFERS THE ABILITY FOR THE ROBOT TO LOOK JUST FAR ENOUGH AHEAD IN TIME TO STOP BEFORE A COLLISION WITHIN ITS BRAKING CAPABILITIES. SWEEPS ARE NOT TEMPORAL, AND THEY DON’T NEED TO BE SINCE THE LOOK AHEAD PERIOD IS JUST THE SWEPT VOLUME THAT IS POSSIBLE IF THE ROBOT WERE TO START BRAKING NOW. THE 0.4 MS EXECUTION TIME MAKES THEM A TERRIFIC ONLINE COLLISION DETECTION MODULE THAT COULD BE USED IN CONJUNCTION WITH A MORE COMPUTATIONALLY INTENSE METHOD THAT COULD LOOK FURTHER AHEAD AND ATTEMPT TO OPTIMIZE COLLISION AVOIDANCE.
   2. Four step collision detection (0.4ms): 1) Compute the joint interval required for braking, which is a function of velocity, latency, angles, accelerations, and uncertainty, 2) Compute swept volume of each joint on robot, 3) Compute distance of sequences of joints, 4) Brake if any distances are zero.
   3. This robot is checking self collision but is analogous to two bodies because it has two independent arms that can collide.
   4. Time is not encoded: Thus, any intersection of the swept volumes constitutes a collision.
   5. “Self-Collision”: Assumes that all kinematic chains are linked. This allows them to simplify the calculation.
   6. Robot is modeled with a series of convex polygons.
   7. Swept Volume Computation
      1. Bodies are defined in their own reference frame and transformed to a common frame.
      2. Collision detection can be computed in any frame, so they selected the least common ancestor (lca) along the kinematic chains (the torso). This allows joint rotations that affect both kinematic chains, such as the hip, to not introduce extra computation. If the fixed frame were used, the swept volumes of the arms would be larger since the hip sweeps them more, but since the collision is checked in the torso’s frame, the hip doesn’t add this extra computation.
   8. Checking distance between volumes is streamlined by maintaining a lower bound on the distance that is updated every iteration of the GJK algorithm (used for computing shortest distance between convex hull polynomials).
      1. 1) Compute changes for swept volumes, 2) update bounds for distance between two volumes, 3) Run GJK to converge on new distance between polynomials
8. **Real-time Continuous Collision Detection for Mobile Manipulators – A General Approach**
   1. Extension of previous result to mobile manipulators and various other joint types: This step is needed to be comparable in terms of model accuracy and dexterity (we don’t need to remain convex however).
   2. Sphere Swept Convex Hulls (SSCH) used to represent volume by a finite set of points and a radius.
   3. Additional transformations introduced to accomplish goal in (a).
   4. Generating points: points used along the shapes trajectory to serve as feed in points to the SSCH algorithm that generates a bounding volume around the points.
      1. THESE NOTES APPLY TO REVIEW OF PREVIOUS PAPER
      2. These generating points are selected at various points along a modeled objects trajectory. They are the points used to define a shape that wraps around the modeled objects entire swept motion.
      3. Different numbers of points can be used to get more complex motions. 1 point would represent a sphere that circumscribes the entire shape, 2 points is more of an ellipse or 3D slot shape with the points as the centers of the circles at either end of the shape. 3 points can be used to define a triangle around the entire sweep. It looks like this is as complex as they go.
      4. This kind of formulation only holds if motions are relatively linear and the sweep only projects out a short distance in time.
      5. For more curvature in the trajectories, this approach will either cut off some of the motion (in the case of 2 generating points)or drastically overestimate the swept volume (in the case of 3 generating points)
      6. Sever Limitations: This approach can only be used if the trajectories are all straight (which is not natural movement for a human or robot). If this is not the case, only very short sweeps as a time can be taken. To use this approach in the way we need it, we would need to apply it iteratively. In our case, we would need to reapply it every time we get a new frame or two of data.
      7. INDEED, IN THEIR SUPPLEMTARY VIDEOS, ALL THEY DO IS TEST HIGHLY LINEAR MOTIONS (ARMS AND JOINTS NOT MOVING) AND PROJECT IMMEDIATELY IN FROM TO THE ROBOT ONLY.
9. **A New Library for Real-time Continuous Collision Detection**
   1. This work basically describes the library used to perform calculations in the above papers dealing with the SSCH.
   2. In past work, a library was developed (in C++) for continuous collision detection based on fast swept volume computation.
   3. Library makes extensive use of sphere swept convex hulls.
   4. Calculations take short cuts when possible (e.g. not computing distances that are known to be larger than other distances that are calculated to determine collision)
   5. Volumes are the Minkowski sum of a convex polyhedron and a ball of radius r.
   6. User defines a kinematic chain that describes their configuration for the library to compute on.
   7. The library is capable of taking input for revolute and prismatic joints. It is currently (as of 2012) being extended to mobile manipulators.
   8. A collision volume generator (which creates the SSCH) leverages CAD models of the robot to facilitate calculations.
   9. The authors have also put a lot of work into developing a good visualization of the collision volumes, overlaid on top of a visual of the robot. They discus this visualization with emphasis in all three papers that they published around this time on this subject.
   10. DEPENDING ON OUR NEEDS IN THE FUTURE, THIS LIBRARY MAY BE OF GREAT USE. IT PROVIDES A VERY QUICK ONLINE COLLISION DETECTION MODULE. THIS WOULD BE USEFUL IF WE FOUND THAT THE SURFACE SWEEP WAS USEFUL FOR ADJUSTING FUTURE SEGMENTS, BUT BY THE TIME WE ARE IN ONLINE EXECUTION, IT IS TOO LATE TO TRY TO REPLAN. IN THIS CASE WE ARE FINDING THAT THE BEST THING TO DO WHEN A COLLISION IS DETECTED DURING A SINGLE MOVEMENT IS TO JUST IDENTIFY THE COLLISION AND STOP. GIVEN THE SPEED OF EDO, HOWEVER, THIS PROBABLY WONT BE THE CASE.
10. **Real-time Collision Detection for Manipulators Based on Fuzzy Synthetic Evaluation**
    1. Collision detection method based on fuzzy synthetic evaluation.
    2. This paper devises a method of determining the braking distance for each joint.
    3. Swept volumes are then generated to encompass the entire robot throughout its braking distance.
    4. Similar to papers discussed previously they use GJK algorithm to check distances between the convex hulls encompassing the sweeps. The volumes are represented as SSCH’s.
    5. Three factors are used to develop a risk assessment to allow the robot to understand how dangerous a motion is: 1)security distance, 2) Time till collision, and 3) Radius of curvature.
    6. Five grade assessment used: very danger, danger, normal, safety, very safety.
    7. This work builds on the similar swept volume approach but has the same “shortcomings”. It only sweeps motion briefly into the future (in this case as far as the braking distance). The convex hulls used to model the sweep are only accurate for very linear motion and have other inherent inaccuracies. Finally, there is no prediction of human motion. A crude model of the human pose is made with spheres at an instantaneous time. This instantaneous pose is checked with the robot’s projection into the future. If the human is moving rapidly towards the robot, this could cause an issue.
11. **Deep Prediction of Swept Volume Geometries: Robots and Resolutions**
    1. Use of a neural network to approximate the geometry of a swept volume.
    2. While error cannot be bounded, most error happens at the boundaries of the volume. Additionally, the type of error incurred can be tuned.
    3. Specifically, the goal of the NN is to predict the swept volume between two configurations of a robot in the form of a voxel grid.
    4. Training data: two configurations are sampled uniformly at random. Inverse kinematics was completed on each configuration to same this occupancy grid.
    5. A decoder network is used as a guide for the model as output dimension is larger than the input dimension. This is required since the output is an occupancy grid.
    6. Four total layers are used along with ReLu activation functions. 90,000 training samples, and 10,000 evaluation samples were used with a mini batch size of 100.
    7. F1 Accuracy was used to assess performance. Accuracy for a planar robot ranged between 93 and 99 % for different parameter choices. It should be noted that some of the voxels are always occupied and some always unoccupied. This bolsters the accuracy.
    8. The best accuracy for more complex robots ranged between 80 and 90%.
    9. For one of the larger, more complex robots (with a higher number of voxels and DOF’s) prediction time was on average 3.9 ms.
    10. POTENTIALLY, IF ERROR CAN BE FURTHER REDUCED, THIS WOULD BE A GOOD METHOD FOR DETERMINING THE SPATIAL SWEPT VOLUME. FOR EFFECTIVE PREDICTIVE COLLISION DETECTION, HOWEVER, SPATIO-TEMPORAL SWEPT VOLUMES ARE REQUIRED.
12. **GPU-based Real-Time Collision Detection for Motion Execution in Mobile Manipulation Planning**
    1. This work investigates a collision detection strategy involving the overlay of two voxel maps processed on different GPU’s. This work is developed for highly articulated, two armed, mobile robots.
    2. 3D measurements are used to update a probabilistic map via a Bayesian process. This allows consistent computational time for updates.
    3. The ROS Collider Package’s Voxel Collision Check is implemented to perform the check.
    4. A Voxelmap is used to store positional data to allow for quick random access. An equation is given here to map geometric positions to Voxels. 
    5. The robot’s point cloud is generated offline, loaded at program run time, and transformed to match the robot’s joint locations. After transformation, they are voxelixed.
    6. THIS APPROACH DOESN’T COVER PURELY SENSED BODIES (LIKE THE HUMAN) FOR WHICH JOINT TRAJECTORIES ARE NOT KNOWN. IN THIS FRAMEWORK, THERE IS NO WAY TO CONSTRUCT THE TRANSFORMATION MATRICES REQUIRED TO MODEL THE HUMAN WITH THE LOADED POINTS.
    7. OUR APPROACH IS SIMILAR, EXCEPT, BY LEVERAGING THE COONS PATCH SHELL, WE DRASTICALLY MINIMIZE THE NUMBER OF POINTS WE NEED TO TRANSORM, WE HAVE A BUILT IN WAY TO CODIFY TIME INTO THE SWEPT VOLUME, AND WE INTERPOLATE BETWEEN POSES RATHER THAN HAVING TO USE MANY CLOSELY SPACED CONFIGURATIONS.
    8. Computational efficiency is best when it is designed as a pipeline. This requires all steps to be running on the GPU, and data to be kept on the device’s memory. This way the CPU does not need to maintain any memory.
    9. This work extensively focuses on optimizing the GPU’s threads for computation time.
13. Total execution time on their machine (NVidia Titan GTX with 6 GB GDDR5 frame buffer memory and a bandwidth of 288.4 GB/s. The GPU combines 2688 CUDA processing units with a warp size of 32 threads each)
    * 1. Inserting new data via the Kinect sensor took 0.9833ms.
      2. Loading and transforming saved data took 0.936ms for a 9-link robot.
      3. Collision check in the Voxel grid took 9.58ms (for a voxel map size of 400 x 400 x 400).
    1. These times allow for the checking of an arbitrary number of configurations at once (limited only by the GPU memory)
    2. Their approach for map updating was nearly 10 times faster for data insertion than an OctMap.
14. **Safety-aware trajectory scaling for human-robot collaboration with prediction of human occupancy.**
    1. A safety criterion is developed based on the robots braking ability. The product of the braking time and the velocity must be less than the distance to the closest obstacle. In previous works this is completed for only the endpoints of every link on the robot. This work extends the approach to apply to complex convex shapes.
    2. Basically this work uses a distance proximity to scale the robots velocity.
    3. THIS WORK USES A PRETTY SIMPLISTIC MODEL OF THE HUMAN. Full DOF’s for arm but not nearly enough for the head and doesn’t include legs.
    4. They use the same SSCH approach seen many other places.
15. **Collison Avoidance using Point Cloud Data Fusion from Multiple Depth Sensors: A Practical Approach**
    1. Determination of collision free paths is usually done globally with “globally capable” path finding algorithms. On the other hand, local path planning strategies, such as potential fields are typically done online because they are computationally cheaper.
    2. Online strategies should aim to be computable in less than 33ms to match depth sensing rates.
    3. This work seeks to merge point clouds from multiple different depth sensors for use in collision avoidance.
    4. Two Kinect V2 are used on two different dedicated computers (i7-6700 wuth 32 GB RAM).
    5. Transfer of the data between processors (especially due to Matlab) was computationally expensive (20 ms). This was sped up with custom TCP/IP Matlab Jave interface.
    6. Translation and Rotation matrices used to convert data from camera frame to cartesian frame. Since they are using depth sensing, they needed to account for the fact that their frame is radial from the camera. The skeleton tracking SDK however is outputting cartesian positions of the joints in X,Y,Z coordinates already so this consideration is not required.
    7. Both Kinect sensor’s point clouds are merged to get a complete picture
    8. Various grid steps were attempted. It was notice that large grid steps, while decreasing computation time, had little influence on the robot’s trajectory. Thus, in this work, it was decided that a larger grid was preferable for online use.
    9. It looks like they went with a grid step of 0.02. I could not find units for grid step at any point in the paper (I couldn’t find any length units in the paper actually). From the images though it looks like they were dealing in meters, and that a grid step of 0.02 corresponds to a point spacing of 2 cm.
    10. Two collision detection approaches were used: 1) Distance between every point on the human, and a few control points on the robot was checked and 2) Triangulation is used to create a convex hull around the point cloud. (this does not work if the robot attempted to pass between the humans’ hands as the entire human is enclosed in a convex hull.
    11. In the first approach, the distance between each control point and each human point is used to generate a repulsive velocity. All repulsive velocities for a control point are converted to joint space and used (as done in potential fields approaches) to modify joint behavior. This approach took 29.3ms for a grid spacing of 0.02.
    12. This is only completing instantaneous collision detection. It doesn’t look at the future. It is, however, also taking care of the robot response. While it is a good approach, it is solving a problem different than ours. We seek to look into the future to find collisions. Not to simply avoid proximity to instantaneous situations.
    13. The second approach (convex hull) had similar computation time.
16. **Continuous-Time Collision Avoidance for Trajectory Optimization in Dynamic Environments**
17. **Time Distance: A Novel Collision Prediction and Path Planning Method**
18. **A Manipulator’s Safety Control Strategy based on Fast Continuous Collision Detection**
    1. Use of a pre and post contact safety controller. Precontact focusses on collision avoidance and post contact focuses on minimizing forces upon collision.
    2. **Framework:** They use continuous collision detection and joint limits on velocity and acceleration to maintain safety.
    3. A smooth trajectory planner is implemented.
    4. This work appears to use the same collision detection technique as 8-10.
    5. Collision detection was very fast (fastest “best case scenario” was 0.05 ms) but this only modeled a three link robot and a static known object in the environment.
    6. The primary contribution of this work was a smooth trajectory planning technique
19. **Unified GPU Voxel Collision Detection for Mobile Manipulation Planning**
    1. Discusses many requirements of data structures for environment modeling with the goal of finding the ideal data structure to represent a volume such that it can be overlaid with another volume representation to check for collision.
       1. Large sparse static structures
       2. Highly dynamic but local structures
       3. Randomly dispersed dense structures (the kind of objects you want to make swept volumes out of)
       4. Static subvolume moving in map
    2. Their work deals mainly with point clouds sensed with 3D scans of the room. Point clouds are read in, outliers filtered out, and a 3D Bresenham ray casting kernel determines distance between object and camera.
    3. Bayesian occupancy representation is used.
    4. Robot point cloud is generated offline and transformed appropriately to represent a configuration.
    5. Motion planning completed with the OMPL which allows them to superimpose multiple maps.
    6. Voxel maps were determined to work well for dynamic objects (swept volume type obstacles).
    7. Voxel lists are goof for representing motion primitives.
    8. Octrees are optimized in this work to reduce memory footprint.
    9. On average, their collision check took 1 ms, this however doesn’t include any of the auxiliary processes such as constructing the swept volumes or planning new trajectories, or even sensing the environment. This is just the time required to overlay the swept volumes and detect intersection. Also, point clouds are only spatial, not temporal. Finally, none of their experiments demonstrated implementation on dynamic environments. Only static environments and pre-saved motion primitives for the robot were used.
20. **Mobile Manipulation Planning Optimized for GPGPU Voxel Collision Detection in High**
21. **Resolution Live 3D-Maps**
22. **Recurrent neural network for motion trajectory prediction in humanrobot collaborative assembly**
    1. RNN’s look at the influence of the motion state on the next state at each time step.
    2. A state is determined from both the observed current position and the state at the previous step.
    3. Since normal RNNs typically fail due to the interconnectedness of the body, the RNN structure has been modified in this work to include two new things:
       1. Component: investigates trajectory of a specific body part (5 total: two arms, two legs, and spine).
       2. Coordination: investigate interaction among body parts (4 total: arm-arm, arm-spine, leg-leg, leg-spine).
    4. Back propagation used to train the model.
    5. Probability that a specific motion is being predicted correctly is found by implementing Monte Carlo dropout where weights are selected randomly using a Bernoulli distribution.
       1. This in effect makes multiple predictions and averages them to get a probability of the most likely prediction.
    6. Testing procedure involved making predictions while the human assembled a car engine.
    7. The RNN was trained with the objectives to 1) predict end location of a motion trajectory at each time step, ) evaluate transition probability to determine if a transition is to occur such that the robot’s proactive motion should be triggered. (robot can pose for a) handover, b) standing, or c) installation)
       1. This is in effect predicting a future joint location and predicting the current task being completed.
    8. **Input length is 30 frames (1 second)**
    9. **Output is the end positions of the joints based on samples from all 30 frames.**
    10. Execution time for the RNN prediction is 0.02 – 0.03 seconds **.**
23. **Real-Time Human Collision Detection for Industrial Robot Cells**
    1. OctTrees or GPUs are common methods for handling depth point cloud data, storing them efficiently and doing collision detection.
    2. This work builds off of a combination of OctTrees and GPUs by additionally removing all static items in the environment before the collision check to make the collision check faster.
    3. This work specifies real time as 10 FPS
    4. This approach breaks down into three components:
       1. Registration: Sensor fusion from multiple 3D cameras getting depth fields
       2. Segmentation: Removal of static objects
       3. Collision Detection: Use of standard algorithms to do this
    5. ABB IRB 1600 -6kg/1.45m 6 DOF robot used and three 3D depth sensors used.
    6. To run the three cameras a PCU Excess Card was used for computation.
    7. An intel core i7 – 6700k CPU and NVIDIA GeForce GTX 1080 graphics card with a CUDA programming platform for parallel computing.
    8. **Registration:** 
       1. Find a set of correspondences between datasets an compute a transformation matrix that minimizes the distance between the corresponding points.
       2. A series of operations are performed on each point cloud: filtering with a downsampling filter to extract common areas in the point clouds, scale invariant feature transform used on a few keypoints.
       3. Vectors are made to each keypoints. Instances where keypoint vectors from different depth fields point to the same thing are used to estimate the transformation matrix.
       4. An algorithm is used to get the estimated transform to converge. ICP (Iterative Closest Point) is a brute force method for this that minimizes a cost function.
       5. Run time of this approach is 2.5-3.4s. This step is thus an initialization step.
    9. **Segmentation:**
       1. Walls of the room are removed by cutting off known positions of the walls (cropping the workspace)
       2. The robot is removed
       3. by aligning the robots point cloud (via the same registration technique) with the camera’s depth fields and removing those points. This is done at every new set of joint variables throughout time.
       4. Static objects are removed by looking at a short run of time (1 second). All of the points for this time window are run through a noise reduction filter. All of these points are then removed as well.
       5. To remove unfiltered noise, a clustering algorithm is used to assign noise to static objects so that it can be removed along with the object. The clusters identify what segment a noisy point belongs to.
       6. Run time is 40 – 50 ms (within their definition of real time). There is a small amount of corruption to the data. This can be fixed with more extensive filtering, but at the expense of point cloud density.
    10. **Collision Detection:**
        1. Octree implementation allows for an initial rejection offar apart points. This narrows the search space and a finer grain look can be implemented on the data subset.
        2. Benchmarked with an exhaustive search (10.1 FPS on CPU, 9.5 FPS with GPU programming)
        3. Using GPU programming and Octree structure, the fastest performance was 0.04 seconds.
        4. Robot point cloud contained 1140 points and target point cloud contained 1543 points.
    11. ROS not a great platform for computational speed. If real time performance is desired, this author suggests using a different system. The reason is ROS is not reliable for giving computation to these functions and can sometimes get bogged down. Its not suitable for safety critical functions. Another version of ROS, ROS2, is under development to ameliorate this.
24. **Visual Marker Guided Point Cloud Registration in a Large Multi-Sensor Industrial Robot Cell**
    1. Using a combination of 6 Kinect sensors, average point cloud positional error from frue position was on the order of cm. Approximately 3 cm for a sensing distance of 3 meters and 6-10 cm for 5 or more meters of sensing distance. These were computed for sensing a highly distinguishable target.
25. **FCL: A General Purpose Library for Collision and Proximity Queries**
    1. FCL is a unified interface that can handle a wide class of models, from continuous bodies to noisy point clouds.
    2. Types of collision detection
       1. Convex polyhedral:
          1. Based on Minkowski formulation of GJK algorithm.
          2. Can be used to get penetration depth.
       2. Bounding Volume Hierarchy:
          1. Commonly use axis aligned or oriented boxes, spheres, polytypes or swept spheres
       3. Continuous Collision detection:
          1. Can miss collisions and still be computationally intensive.
          2. Swept volumes can be approximated by interpolating between static poses.
       4. Point Cloud Collision Detection:
          1. Can handle collision detection between noisy unstructured point clouds or between one such point cloud and a mesh.
       5. GPU (parallel) collision detection: as discussed in other paper reviews above.
    3. FCL: C++ library supports triangle meshes, common primitive shapes.
    4. Structure set up to efficiently store and perform queries on a number of different types of input object.
    5. Tringle meshes only represent boundaries. If one is inside the other, no collision is reported. Need to use a primitive for this.
    6. Collision check completion time (moving opjects): 5.5 ms for a single primitive.
    7. Collision check with point clouds (or with mesh): 43 ms (average)
    8. Collision check for robot made of triangle mesh: 0.24 ms (grows quickly for other forms)
    9. Performance for point clouds can be “quite slow”