Rapid Constrained Object Motion Estimation based on Centroid Localization of Semantically Labeled Objects

Carol Young[1], Calvin Stahoviak[2], Raymond Kim[1], Jonathon E. Slightam[1]

***Abstract*—Autonomous and semi-autonomous robot manipulation systems require fast classification and localization of objects in the world to realize online generation of motion plans and manipulation waypoints in real-time. Furthermore, constraints and estimated plausible motions of objects of interest in space is paramount for autonomous manipulation tasks. For non-grasping tasks like pushing a box or opening an unlatched door, physical properties such as the center of mass and location of constraints like hinges or bearings must be considered. This paper presents a methodology for rapidly inferring constraints and motion plans for objects of interest to be manipulated. This approach is based on a combination of object detection, instance segmentation, localization methods, and algebraically relating different semantically labeled objects. These methods for motion estimation are implemented on a color-depth camera (RGB-D) and a 7 degree-of-freedom serial robot arm. The algorithm's performance is evaluated through different arm poses, assessing both centroid accuracy and estimation speed, and motion estimation performance. Algorithms are tested on an exemplar problem consisting of a block constrained on a dual linear rail system, i.e., constrained linear motion. Experimental results showcase the scalability of this approach to multiple classes with sublinear slowdowns and linear motion plan direction errors as low as $1.23E{-4}$ [rad]. The manuscript also outlines how these methods for rapid constrained object motion estimation can be leveraged for other applications.**

I. Introduction

Constraint and motion estimation of objects in unstructured environments is a fundamental challenge in mobile autonomous manipulation. Core to these estimation approaches are the classification, segmentation, and localization of constitutive components in mechanical systems. Centroid estimation of semantically labeled objects provides information that can be leveraged for manipulation applications. One manipulation scenario involves determining the center of mass of an object, which enables the manipulator to interact with the object in ways that result in tipping, twisting, or translational motions. Another scenario focuses on identifying the relative locations of objects in relation to specific constraints, such as those imposed by bearings or hinges. Understanding these physical properties and constraints is essential for contact-rich manipulation in unstructured environments. Inappropriate interactions can cause damage to the object, the manipulator, or the constraints themselves. This necessity drives the development of a method for rapidly estimating constrained object motions based on centroid localization and semantic labeling of objects.

This work was supported by Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

[1] Unmanned Systems & Autonomy R&D group at Sandia National Laboratories, Albuquerque, NM 87185 USA. [emails: ccyoung@sandia.gov, rkim@sandia.gov, jslight@sandia.gov]

[2] Author Calvin Stahoviak is a student at University of New Mexico [email: cstahoviak@unm.edu]

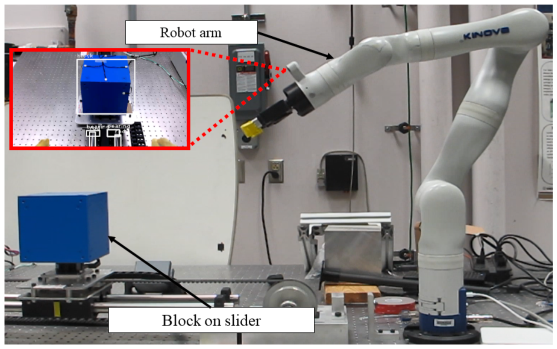

Fig. 1: Experimental setup for the rapid instance based multi-object centroid estimation with detection, segmentation, and localization to determine permissible motions of a block on a slider. The highlighted section depicts the RGB-D image with labeled objects.

II. Background

We present background on segmentation, localization, and model estimation methods and describe what the main contributions of this work are in the following subsections.

1) Segmentation: Great effort has been taken to develop 3D segmentation algorithms for use in cluttered and unknown environments.

Edge-based algorithms [1]–[5], begin by determining edges using differences in depth, angle, and color of neighboring pixels. Regions between edges are associated with surfaces and further filtered. While this approach effectively achieves real-time object segmentation with high accuracy under the assumptions that one surface of an object is at the forefront [1], [3] or objects are generally convex [2], it has limitations when dealing with irregularly shaped or partially occluded objects. Area-based algorithms determine surface patches on an image by making comparisons on adjacent pixels in order to grow regions of similarity. Overall, these approaches were found to be less capable than their edge-based counterparts in direct comparisons [5].

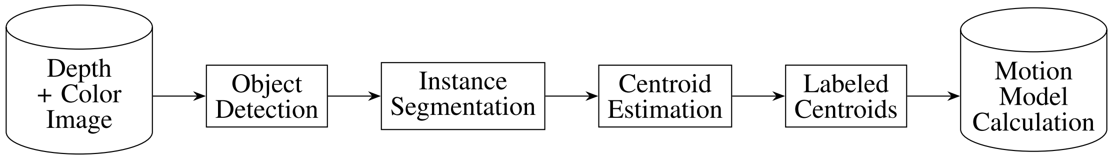

Fig. 2: Flowchart of the proposed algorithm.

Graph-based algorithms [3], [6], [7] use graph theory to group surfaces into object hypotheses. Typically, clustering methods are concurrently employed, as clustering regions of an image serves as an effective prerequisite to constructing a graph. While this approach demonstrated efficiency in segmenting irregular shapes and occluded objects, running in real-time, and occasionally outperforming learning-based counterparts, it exhibits weaknesses in handling complex scenes.

Model-based algorithms [4], [8] utilize object models to match features or shapes identified after segmentation. Many works employing model-based algorithms achieve segmentation and pose estimation simultaneously, as known models aid the segmentation process while simplifying pose estimation. On the other hand, learning-based algorithms [9], [10] leverage a deep learning module to assist in segmentation. A significant drawback of this approach is the considerable effort and data required for effective model training. Recent advancements have improved the efficiency of machine learning models and reduced the work needed to build a large dataset [11]. However, due to the utilization of multiple learning models, this approach is too slow for real-time applications.

2) Localization: Methods for localization of objects depend on segmenting out the objects of interest from the background, the available sensor data, and prior information. In cases when depth information is unavailable, 2D bounding boxes can be combined with a series of learning models to regress those 2D bounding boxes to 3D [12] but this does require large training sets for the best results. When depth information is available, 2D bounding boxes can be projected into 3D clouds, and the 3D bounding box regression is done by training a multi-layer perceptron (MLP) network. However, because the use of multiple learning models, this method is too slow for real-time applications [10]. When models are available, such as when using model-based segmentation algorithms [4], [8], the placed model can be leveraged to identify position and pose. Finally, when a global point cloud is available, position is able to come directly from it [1], [2].

3) Model Estimation: Understanding the motion model parameters of an object, particularly the permissible motions, is critical for optimizing interaction and motion strategies in object manipulation. Previous studies have demonstrated the feasibility of estimating object motion from sequential image data using various techniques. Many of these works exploit the information redundancy of pairwise motions in sequential images to estimate the kinematic and inertial parameters of the objects in motion. These information redundancies can then be further processed with adaptive Kalman filters [13], linearly fitting global models [14], recursive least-squares and linear-quadratic tracking algorithms [15], [16], or nonlinear methods [17]. In recent years, a time granularity-based visual measurement and prediction framework has also been developed to forecast the uncertain trajectory of moving objects for robotic manipulation. [18]. While considerable research has focused on estimating the motion of moving objects using image sequences, limited attention has been given to estimating the permissible motion of static objects of interest.

4) Contributions: This work proposes a fast algorithm for using RGB-D data to label, estimate, and combine the centroids of objects to determine the motion model of an object of interest. The key contributions of this paper are 1) presenting an algorithm for rapid detection, segmentation, and centroid estimation of objects using RGB-D, 2) experimentally demonstrating the scalability of the approach, 3) and illustrating how a motion model can be generated to identify permissible motions on the desired object of interaction. This proposed approach allows for a fast motion estimation of constrained objects for autonomous manipulation applications.

III. Motion Model Estimation Algorithm

This section outlines the proposed algorithm, visualized in Figure 2, and covers in detail the subsequent steps for quickly estimating object motion models. The methodology can be separated into detection, segmentation, estimation of the volumetric centroid, and motion model calculation. The entire process is divided into four different subsections to allow for easy experimentation with various methodologies or strategies. This structure supports targeted improvements in processing speed where they are most impactful. For instance, dedicating GPU resources specifically to segmentation and detection can optimize performance by reducing the search space required for segmentation.

A. Object Detection

Our algorithm utilizes the You Only Look Once version 3 (YOLOv3) convolutional neural network model to generate bounding boxes for detected objects [19]. We employ Darknet for training the YOLOv3 model, which performs detection on color images captured by the manipulator. Specifically, the model is trained to recognize three classes of objects—blocks, bearings, and rods—which are critical for identifying both the movable object and its constraints. An example of this detection output is illustrated in Figure 3. Using object detection as the initial step allows us to minimize the image size needed for subsequent segmentation, thereby enhancing the overall speed of the algorithm.

B. Instance Segmentation

The instance segmentation algorithm, outlined in Algorithm 1, is influenced by methods previously described in [1] and [20]. This algorithm leverages the current depth map and the bounding boxes output from object detection, which are depicted from various angles in Figure 3. Additionally, the implementation has been optimized for CUDA/GPU processing, significantly enhancing the system's overall speed.

Algorithm 1: Instance Segmentation

```
Data: Depth map $D$, Bounding box dimensions $B$
Result: Segmented Point Cloud $P$
1: Determine coordinate map, $C \leftarrow D$
2: Determine surface normal map, $N \leftarrow C$
3: Determine edge map, $E \leftarrow N$
4: Determine threshold-ed edge map, $T \leftarrow E$
5: Perform region growing on $T$ to generate surface patch
   map $S$
6: Perform surface grouping algorithm 2
7: Generate mask $M$ as all points in all surfaces of $R$
8: Generate point cloud $P$ by masking $D$ with $M$
```

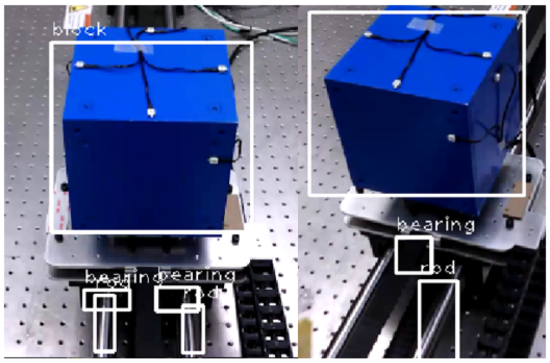

Fig. 3: Left: straight view of the block with classification labels of the block, two rods, and two bearings. Right: negative isometric view of the block with classification labels of the block, rod, and bearing.

The current depth map D_t, as seen in Figure 4, serves as the starting point of our algorithm. Each pixel in D_t aligns with a point in the vertex map V_t, which contains the 3D coordinates of each depth point.

As shown in [1], we calculate a dot product at each point to generate the surface normal map N_t composed of vectors $\vec{n}$ (Figure 5). $\vec{a}$, $\vec{b}$, and $\vec{c}$ represent vectors between three points surrounding the point of interest.

$$\vec{n} = (\vec{b} - \vec{a}) \cdot (\vec{c} - \vec{a}) \tag{1}$$

As shown in [1], we generate an edge map, E_t, as depicted in Figure 6. This involves using the normal vector, $\vec{n}$, at each point to compute the dot products. These calculations are conducted between the center pixel and its three nearest neighbors in each cardinal direction, denoted as $j = (N, E, S, W)$, and labeled $\vec{m_{i,j}}$ where $i = (1, 2, 3)$. After computing these dot products, we average the results for each direction. The final

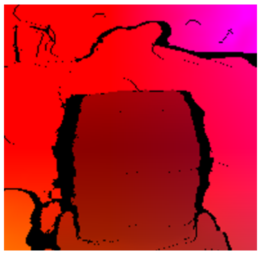
Fig. 4: Depth image

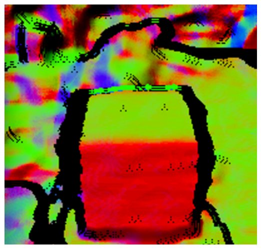
Fig. 5: Surface Normal Map

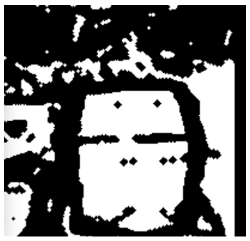
Fig. 6: Edge Map

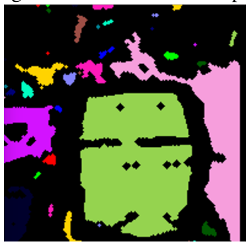
Fig. 7: Resulting Surfaces

value e for each point in E_t is determined by selecting the minimum dot product value—which corresponds to the maximum angle difference—from these averages in each cardinal direction.

$$e = \text{argmin}_j \left(\frac{\sum_{i=1}^{3} \vec{m_{i,j}} \cdot \vec{n}}{3} \right) \tag{2}$$

Based on [2], we use a threshold-operator Φ on each point e in E_t to generate a binary edge map T_t.

$$\Phi = \begin{cases} 1, & \text{e} > 0.5 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

We then apply a surface growing algorithm to T_t to generate a map of surface patches S_t (Figure 7). Pixels belonging to the same surface share the same positive integer value, while edges are denoted by -1. We accomplished this by using the method outlined in [20].

Next, we apply the surface grouping algorithm described in Algorithm 2. $SumOriginal$ is an array of patches whose index is its identifying integer and whose value is the area within the bounding box B. $SumExpanded$ is identical except for utilizing B' which is the bounding box B expanded by 20%. By comparing these areas, we can determine which surfaces were at least 90% within the original bounding box. We associate those surfaces with the object.

Finally, we generate a point cloud by using the resulting surface patches R to mask the depth image. This point cloud P is then employed in the next module for centroid estimation.

C. Centroid Estimation

This module is broken into two sections. The instantaneous centroid is computed during every iteration of the instance segmentation, and the filtered centroid is computed at a constant rate.

Algorithm 2: Surface Grouping

```
Data: Surface patch map $S$, Bounding box submap $B$
Result: Set $R$ is the set of surfaces associated with the
        object
$SumOriginal \leftarrow []$;
$SumExpanded \leftarrow []$;
for pixel $p$ in $B$ do
    Increment $SumOrignal[S_p]$
end
Expand $B$ by 20%;
for pixel $p$ in $B'$ do
    $SumExpanded[S_p]$ += 1;
end
for element $i$ in $SumOriginal$ do
    if $SumOriginal[i] >= 0.9 * SumExpanded[i]$
     then
        $R \leftarrow i$;
    end
end
```

1) Instantaneous Centroid: The accuracy of centroid estimation depends on the accuracy of the instance segmentation. Precautions can be taken to address potential misidentifications of object surfaces during instance segmentation. However, if misidentified surfaces are not filtered out, the centroid estimation could be severely affected.

Our initial method for rapidly estimating the centroid involved calculating the average of all points associated with the object. Although this approach is quick, it suffers from significant drawbacks. First, the inclusion of background surfaces erroneously associated with the object can cause the centroid to drift. Second, even when all visible surfaces are accurately grouped, the estimated centroid tends to shift toward the camera rather than aligning with the true centroid. This bias occurs because of a higher density of points closer to the camera and fewer points on the occluded sides of the object. Fortunately, the modular design of our system enabled us to swiftly replace this method with a more accurate approach to overcome these challenges.

To counteract the centroid drift toward the camera, we implemented a minimum-oriented bounding box around all identified points. We estimate the centroid by calculating the center of this box. However, it is important to note that this method's accuracy can be compromised if surfaces are misidentified, potentially making it less precise than the previously described averaging approach. The minimum-oriented bounding box around the segmented point cloud is illustrated in Figure 8.

In the final step, we use the previous centroid estimates, from c_{t-1} to c_{t-10}, to mitigate the impact of outliers that could cause drastic shifts in the object's centroid within a single frame. The steady centroid for the current time, c_t, is determined by averaging these last 10 positions, as outlined in Equation 4, provided that they are available.

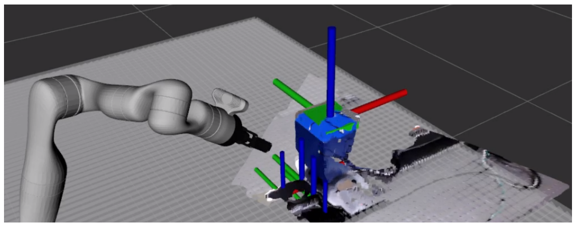

Fig. 8: The location of the centroid (large blue and red axes) and minimum orientated bounding box (green) with the arm in a negative isometric pose, as depicted in RViz.

$$c_t = \frac{1}{10}\sum_{n=1}^{10} c_{t-n} \tag{4}$$

2) Filtering: Once the instantaneous centroid is generated, we apply a threshold to exclude centroids associated with inappropriately sized bounding cubes. Specifically, centroids enclosed within a minimum-oriented bounding box with any side length exceeding $300mm$ are disregarded. Following this, Kalman filtering is applied to refine the centroid estimations. This filtering ensures that the centroids are maintained even when the object is occluded or temporarily out of the frame.

Upon receiving a new centroid and an associated YOLO label, the filtering process assesses whether this represents a new object. It scans through all currently recognized objects with the same label to identify the one closest to the new centroid. If the closest object is beyond a specified threshold distance ($5m$ for a block, $0.3m$ for a bearing, and $0.5m$ for a rod), a new object is instantiated. If the distance from the new centroid to the nearest known object is below the threshold, then the centroid's position and velocity (assumed to be zero) are used as inputs for the Kalman filter.

The Kalman filter uses three tuning parameters, the state covariance matrix **P**, measurement covariance matrix **R**, and process noise covariance matrix **Q**. The state covariance matrix is designed such that all states have the same covariance, and that each state is independent. The state covariance matrix is shown in Equation 5.

$$\mathbf{P} = 1000 I_6 \tag{5}$$

The measurement covariance matrix is also designed such that all measurements have the same covariance and such that each measurement is independent. The measurement covariance matrix is shown in Equation 6.

$$\mathbf{R} = 0.25^2 I_3 \tag{6}$$

Finally, the process noise covariance matrix is designed to use the Discrete Constant White Noise model for a two dimensional system with a variance of 1 and a timestep of 0.1. The output of this is shown in Equation 7. Currently, there is no forgetting factor in the filtering.

$$\mathbf{Q} = 10^{-4}\begin{bmatrix} 0.25 I_3 & 5 I_3 \\ 5 I_3 & 100 I_3 \end{bmatrix} \tag{7}$$

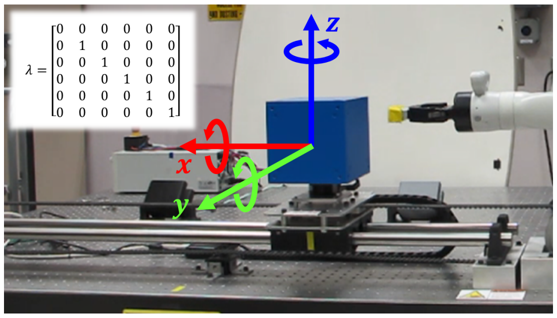
$\lambda = \begin{bmatrix} 0&0&0&0&0&0\\ 0&1&0&0&0&0\\ 0&0&1&0&0&0\\ 0&0&0&1&0&0\\ 0&0&0&0&1&0\\ 0&0&0&0&0&1 \end{bmatrix}$
z
x
y

Fig. 9: Visual representation of the motion model of our hardware setup. The cube on the linear rail is restricted on all directions except the x axis. Motion model represented as λ is shown on the top left. Note that λ is a diagonal matrix, each representing the permissibility of motion from 0 to 1 in each axes.

D. *Motion Model*

The motion model is represented by the diagonal matrix $\boldsymbol{\lambda}$. The first three values correspond to translation along the XYZ axes, while the last three represent rotation around these axes. The values in $\boldsymbol{\lambda}$ range from 0 to 1, where 0 indicates an axis fully open to movement, and 1 indicates one that is completely constrained. For example, a block on a rail system, as shown in Figure 9, would have $\boldsymbol{\lambda} = \text{diag}(0, 1, 1, 1, 1, 1)$ since it can move freely along the x-axis but is constrained in the y and z axes by the rails and gravity, respectively.

Using centroid estimation and classification, we generate a permissible motion model for each identified block. This process utilizes the list of all bearings and rods identified by the filter. Once at least one bearing and one rod are detected, we generate a constraint ID. This ID operates under the assumption that all bearings are associated with the block. For systems involving multiple blocks or multiple objects of interests, we could introduce a threshold to accurately assign constraints to specific blocks/objects.

We encounter several edge cases in our model. When there is only one bearing and one rod, the angle of motion is determined by the line connecting the two. If there is one bearing and multiple rods, the angle of motion is defined by the line between the bearing and the first rod observed. Conversely, if multiple bearings and only one rod are present, the angle of motion is determined by averaging the lines connecting the rod to each bearing.

The most common case consists of multiple bearings and rods. In this case, we iterate through all possible combinations of the two sets of unique bearing and rod pairs. Each bearing is then assigned a rod so that the total angle difference between all pairs is minimized as shown in Equation 8. The average line for this assignment is found using Equation 9. This line gives the angle of achievable motion of the block, it can also be represented as a lambda constraint using Equation 10.

$$m = \arg\min_{k \in S} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \arccos\left(\frac{(b_i^k - r_i^k) \cdot (b_j^k - r_j^k)}{\left|b_i^k - r_i^k\right| \left|b_j^k - r_j^k\right|} \right) \quad (8)$$

$$v = \frac{1}{n} \sum_{i=1}^{n} b_i^m - r_i^m \quad (9)$$

$$\boldsymbol{\lambda} = \text{diag}(1 - |v_1|, 1 - |v_2|, 1 - |v_3|, 1, 1, 1) \quad (10)$$

IV. Experimental Approach

The experimental setup is illustrated in Figure 1. It features a block positioned at $(915, 0, 265)mm$ from the base of the Kinova Gen3 7-DOF manipulator, alongside a linear rail that allows motion along a 1D axis parallel to the x-axis of the Kinova arm base. We tested our algorithmic centroid estimation approach using multiple end-effector poses, including a straight-on view and both positive and negative isometric views of the block. The coordinates for each of these poses are detailed in Table I. The manipulator arm sequentially moved through these positions, beginning with the positive isometric view and concluding with the negative isometric view. It remained stationary for 4 seconds at each pose and took 10 seconds to transition between poses. Images of the straight and negative isometric views captured by the camera, along with the YOLO output, are displayed in Figure 3.

During the experiment, we estimated the centroid using YOLO-classified RGB-D images of the block. We measured the error between the estimated centroid and the actual block centroid, as well as the time required for centroid estimation. The experiment was initially conducted using two YOLO classification labels (block and rods) and subsequently repeated with three labels (block, rods, and bearings). This iterative approach allowed us to assess how the algorithm scales with an increasing number of classified objects, which is crucial for accurately computing motion models. Finally, we calculated the motion model of the block using results from the three-class setup. This enabled us to compare the estimated permissible motion of the block with the actual movement along the linear rail hardware.

V. Experimental Results

A. *Centroid Estimation Accuracy*

The complete statistics for the instantaneous and filtered centroid estimation errors are presented in Tables II and III. A comparison of these tables reveals that while applying the filter did not markedly alter the average error, it did reduce the standard deviation, thereby enhancing the system's overall precision. An exception to this trend was observed in the positive isometric view with three classes on the z-axis, where there was an increase in the average standard deviation.

The statistics for the filtered centroid estimations are illustrated in the whisker plots in Figures 10 and 11. The similarity between these plots indicates that the presence of additional classifiable objects does not significantly impact the error in centroid estimation. These plots also highlight that the z-axis error is the predominant source of error. Following this, the x-axis error shows a unique trend: unlike the y and z-axis errors, it increases when the arm transitions from the 'Positive Isometric' to the Straight' position.

The 3D error throughout the entire experiment is depicted in Figure 12. This figure illustrates that significant fluctuations in error occur when the arm moves between positions, specifically from 'Leave Pos Iso' to 'Arrive at Straight', and from 'Leave Straight' to 'Arrive at Neg Iso'. These large changes in error during these transitions indicate a loss of centroid placement precision.

The 2D error, which reflects the accuracy of the estimated position on the table excluding height estimation, for all identified objects is presented in Figure 13. For all objects, a loss of precision is observed during transitions between positions, and the accuracy varies depending on the object's orientation.

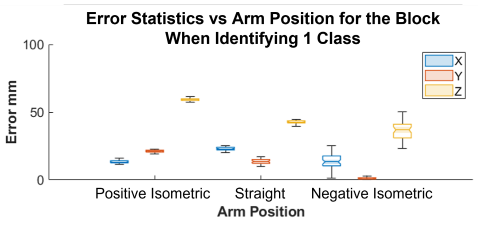

Fig. 10: Statistics, excluding outliers, for error in centroid estimation of the block with one class active.

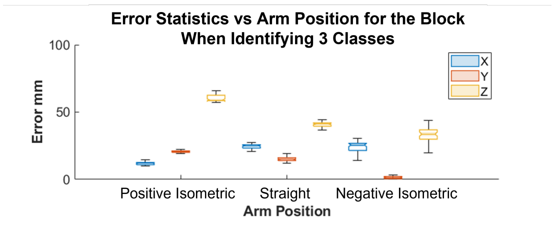

Fig. 11: Statistics, excluding outliers, for error in centroid estimation of the block with three classes. No significant change from the one class results.

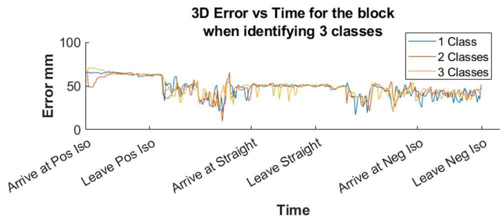

Fig. 12: 3D error of the centroid estimation over the runtime of the experiment in the case of one, two, and three classes.

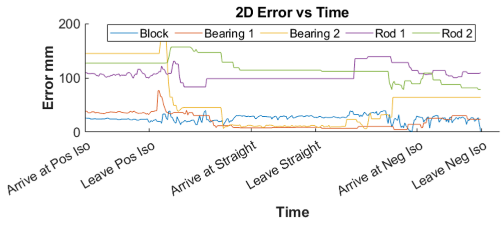

Fig. 13: Error of the centroid estimation on the XY plane of each instance of each class over time.

B. Centroid Estimation Speed

Table IV displays the runtime statistics for both the instantaneous centroid and the rate-limited filtered centroid calculations. The average time for each step of the filtered centroid estimation was consistently within $0.1ms$ of the target time set in the software. The speed statistics for the instantaneous centroid estimation are particularly noteworthy, as they are influenced by the performance of instance segmentation and object detection, as detailed in Figure 14. Notably, as the number of classes increased, the average computation time rose by $10.3ms$ when moving from one to two classes, and by $3.0ms$ from two to three classes, indicating a sub-linear growth rate.

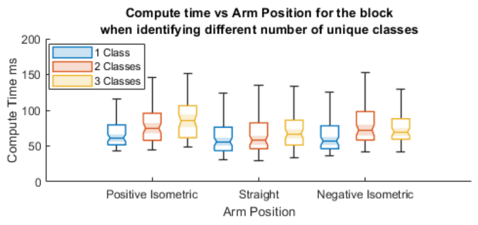

Fig. 14: Computation time for a single measurement in the case of one, two, and three classes over all three positions.

C. Motion Model Estimation Accuracy

The accuracy of the motion model is evaluated by comparing the calculated angle of block motion to the ground truth angle of the linear rail motion. The error of the angle of motion with respect to time is shown in Figure 15. Note that the error is minimal (1.23×10^{-4} radians) when the manipulator is in straight view pose. At this view, YOLO can clearly identify both bearings and rods, as shown in Figure 3. However, this accuracy is not consistent across other views. The highest peaks of error occur when the arm is moving, shown during times between 'Leave' and 'Arrive'. In addition, the negative isometric pose, or between 'Arrive at Neg Iso' and 'Leave Neg Iso', has the maximum error of the stationary poses.

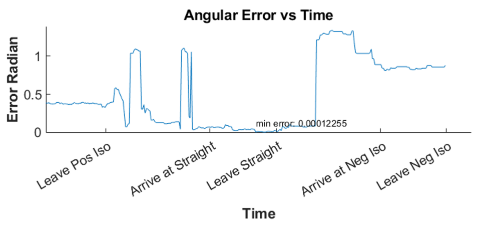

Fig. 15: Error of the angular representation of the available motion over time.

D. Correlation of Error

To identify the cause of motion model error, we correlated the errors of the various labeled centroids. Figure 16 illustrates the correlations at the time between 'Arrive at Neg Iso' and 'Leave Neg Iso'. The results shown in the two figures indicate that the errors are highly dependent on 'Rod 1', which was the initial rod identified during the experiment. 'Rod 1', as shown in Figure 3, did not have consistent YOLO identification in the negative isometric view.

VI. Discussion

This paper introduced an algorithm for estimating motion models by identifying centroids using RGB-D data, coupled with rapid detection and segmentation techniques. The experimental results highlighted that the centroid detection algorithm maintained high performance levels, with only a marginal increase in runtime as additional classes were identified. This suggests a sub-linear relationship in computational demand.

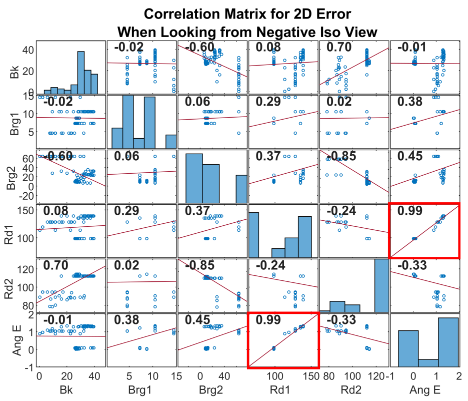

Fig. 16: Error correlations between each class for the Negative Isometric View.

The findings suggest that the algorithm can be expanded to predict and estimate the motions of various other objects, such as slides, hinges, and wheels.

The dominant factor contributing to the overall centroid error was the Z error, which likely arose from positional limitations that resulted in the loss of extreme points along the z-axis. This issue became particularly noticeable when the camera's view was restricted. While this level of error was manageable for determining the X and Y constraints of an object sliding on a surface, additional viewing strategies would be necessary to minimize z-axis point loss if other motion modes, such as tipping or ramps, are considered. Similarly, the X error became more pronounced when the block was viewed head-on, potentially due to point loss along the x-axis caused by camera positioning. Nonetheless, despite the increased data noise during movement, the mean error of the block centroid remained within acceptable limits.

Disappearing and incomplete objects are detrimental to the performance of the motion model, and a challenge going forward. A potential solution is to implement a tracking algorithm that maintains better estimates of component locations.

VII. Conclusion

In this manuscript, we developed an algorithm for fast constrained object motion estimation, achieving an average end-to-end speed of $13.7Hz$ and directional motion estimate errors as low as 1.23×10^{-4} radians. The algorithm exhibits a sub-linear slowdown in centroid estimation speed when additional identifiable classes are present, indicating scalability. However, when the robot is unable to observe all constraints from any given position, the performance of the motion model estimation deteriorates, underscoring the need for improved object tracking. This work demonstrates that a motion model for identifying permissible motions can be effectively generated through rapid centroid estimation of semantically labeled objects.

References

[1] A. Ückermann, C. Elbrechter, R. Haschke, and H. Ritter, "3d scene segmentation for autonomous robot grasping," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1734–1740.

[2] K. Tateno, F. Tombari, and N. Navab, "Real-time and scalable incremental segmentation on dense slam," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4465–4472.

[3] A. Ückermann, R. Haschke, and H. Ritter, "Realtime 3d segmentation for human-robot interaction," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 2136–2143.

[4] D. Holz and S. Behnke, "Fast edge-based detection and localization of transport boxes and pallets in rgb-d images for mobile robot bin picking," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*. VDE, 2016, pp. 1–8.

[5] A. Harati, S. Gächter, and R. Siegwart, "Fast range image segmentation for indoor 3d-slam," *IFAC Proceedings Volumes*, vol. 40, no. 15, pp. 475–480, 2007.

[6] M. Geetha and R. Rakendu, "An improved method for segmentation of point cloud using minimum spanning tree," in *2014 International Conference on Communication and Signal Processing*. IEEE, 2014, pp. 833–837.

[7] S. Christoph Stein, M. Schoeler, J. Papon, and F. Worgotter, "Object partitioning using local convexity," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 304–311.

[8] A. Ahmed, A. Jalal, and K. Kim, "Rgb-d images for object segmentation, localization and recognition in indoor scenes using feature descriptor and hough voting," in *2020 17th international Bhurban conference on applied sciences and technology (IBCAST)*. IEEE, 2020, pp. 290–295.

[9] Z. Wang, Y. Xu, J. Yu, G. Xu, J. Fu, and T. Gu, "Instance segmentation of point cloud captured by rgb-d sensor based on deep learning," *International Journal of Computer Integrated Manufacturing*, vol. 34, no. 9, pp. 950–963, 2021.

[10] J. Lahoud and B. Ghanem, "2d-driven 3d object detection in rgb-d images," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 4622–4630.

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[12] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 7074–7082.

[13] F. Aghili and K. Parsa, "Motion and parameter estimation of space objects using laser-vision data," *Journal of guidance, control, and dynamics*, vol. 32, no. 2, pp. 538–550, 2009.

[14] V. M. Govindu, "Combining two-view constraints for motion estimation," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 2. IEEE, 2001, pp. II–II.

[15] R. J. Crinon and W. J. Kolodziej, "Adaptive model-based motion estimation," *IEEE Transactions on Image Processing*, vol. 3, no. 5, pp. 469–481, 1994.

[16] T. J. Broida and R. Chellappa, "Estimation of object motion parameters from noisy images," *IEEE transactions on pattern analysis and machine intelligence*, no. 1, pp. 90–99, 1986.

[17] S. Soatto, R. Frezza, and P. Perona, "Motion estimation via dynamic vision," *IEEE Transactions on Automatic Control*, vol. 41, no. 3, pp. 393–413, 1996.

[18] C. Xia, C.-Y. Weng, Y. Zhang, and I.-M. Chen, "Vision-based measurement and prediction of object trajectory for robotic manipulation in dynamic and uncertain scenarios," *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 11, pp. 8939–8952, 2020.

[19] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[20] A. Parikh, M. W. Koch, T. J. Blada, and S. P. Buerger, "Rapid autonomous semantic mapping," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 6156–6163.

APPENDIX

TABLE I: End-Effector Positions for Hardware Experiment in m and rad

	X	**Y**	**Z**	**Roll**	**Pitch**	**Yaw**
Positive Isometric	-0.1	0.25	0.13	0.232	0.7	-0.425
Straight	-0.1	0.0	0.13	0.0	0.7	0.0
Negative Isometric	-0.1	0.25	0.13	-0.232	0.7	0.425

TABLE II: Error statistics for the instantaneous centroid in mm

		X Error				**Y Error**				**Z Error**			
		mean	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**
Filter Block 1 class	**Pos Iso**	13.3	2.03	7.54	18.4	21.0	1.32	19.0	24.4	58.7	1.78	54.0	61.9
	Straight	23.1	2.22	17.9	27.3	13.9	2.80	8.37	20.0	42.5	1.81	37.0	45.7
	Neg Iso	12.9	9.97	0.06	26.8	2.09	1.66	0.18	5.96	37.4	10.3	19.8	59.8
Filter Block 2 classes	**Pos Iso**	11.6	2.62	8.17	19.7	19.8	1.45	17.7	22.9	57.4	8.50	18.7	60.8
	Straight	25.2	1.61	19.9	28.7	13.6	2.18	9.68	20.4	41.3	2.32	36.0	45.7
	Neg Iso	21.6	7.45	0.12	30.2	1.97	1.55	0.02	6.27	36.1	8.24	19.2	59.5
Filter Block 3 classes	**Pos Iso**	11.9	1.92	7.91	14.5	20.7	1.45	17.9	22.7	58.4	1.12	56.4	60.5
	Straight	23.6	5.66	0.84	27.5	15.0	2.86	9.54	27.4	39.2	9.00	5.06	45.7
	Neg Iso	23.1	9.05	1.35	31.6	2.00	1.60	0.02	5.78	34.1	8.13	19.6	60.0
Combined		18.5	5.70	0.06	12.2	2.00	1.08	0.02	27.4	45.0	6.70	5.06	61.9

TABLE III: Error statistics for the filtered centroid in mm

		X Error				**Y Error**				**Z Error**			
		mean	**std**	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**	**min**	**max**
Filter Block 1 class	**Pos Iso**	13.0	1.66	6.67	16.1	21.1	1.03	19.1	23.0	59.2	1.17	56.1	61.4
	Straight	23.0	1.30	20.4	25.2	13.7	1.74	10.0	17.0	42.7	1.20	39.7	44.8
	Neg Iso	13.3	5.15	1.18	25.4	1.33	1.11	0.07	4.45	36.5	6.07	23.4	50.1
Filter Block 2 classes	**Pos Iso**	12.4	1.79	10.2	16.7	19.9	0.77	18.4	21.1	56.7	5.63	41.6	61.5
	Straight	25.3	0.82	23.6	27.3	13.6	1.50	10.8	16.2	41.5	1.35	38.6	44.3
	Neg Iso	21.8	4.77	6.19	28.8	1.29	0.98	0.03	3.51	35.8	4.24	27.3	48.4
Filter Block 3 classes	**Pos Iso**	12.4	1.78	9.81	19.5	20.7	0.79	19.4	22.5	60.5	2.90	57.2	66.0
	Straight	23.8	3.02	14.7	27.3	15.1	1.51	12.3	19.2	39.3	4.76	23.5	44.4
	Neg Iso	24.0	4.60	7.20	30.3	1.45	1.08	0.10	4.82	33.2	5.42	19.6	44.0
Combined		18.7	3.10	1.18	30.3	12.0	1.20	0.03	23.0	45.0	4.10	19.6	66.0

TABLE IV: Instantaneous centroid and filtered centroid code runtime statistics in ms

		Instantaneous Centroid Runtime				**Filtered Centroid Runtime**			
		mean	**std**	**min**	**max**	**mean**	**std**	**min**	**max**
Filter Block 1 class	**Pos Iso**	67.6	20.0	43.1	127	99.9	1.34	97.0	105
	Straight	61.9	23.7	30.5	127	100	1.65	95.6	104
	Neg Iso	66.2	25.7	36.6	130	100	1.45	96.4	104
Filter Block 2 classes	**Pos Iso**	79.8	25.8	44.5	146	100	0.80	96.4	102
	Straight	66.5	26.2	29.8	135	100	0.87	97.5	104
	Neg Iso	80.4	28.3	41.2	152	100	0.54	97.9	101
Filter Block 3 classes	**Pos Iso**	87.3	27.9	49.1	179	100	1.24	94.0	104
	Straight	71.2	26.6	34.1	152	100	2.71	86.6	113
	Neg Iso	77.0	23.9	41.5	157	100	1.83	94.4	106