A Framework for CAD-Based Offline Depth-Map Preparation for Automated Assembly Tasks

Arun Dayal Udai[1], and Subir Kumar Saha[2]

***Abstract*— This paper presents an intuitive offline depth-map generation tool that can be applied for any generic assembly task using industrial robots. Such depth-maps are commonly required to be generated for any new robotic assembly in hand. Once the assembly of the peg and hole is designed in the Computer Aided Design (CAD) platform, the software framework proposed in this paper is used to generate the depth-map automatically.**

I. Introduction

A two-dimensional array of perpendicular-distance of a 'peg' with respect to its mating 'hole', while the peg is in contact with the surface of hole is termed as '*depth-map*' or '*height-map*'. Figure 1 shows the depth c_y which is used to generate the depth-map. Such maps are widely used in industrial applications of robots performing peg-in-hole operation [1], [2], robot-based localization [3], automatic adaptation [4], flexible industrial automation [5]–[7], offline programming [8], [9], etc. Depth-map is a proven tool for localization for any peg-in-hole task in an industrial assembly process. A guaranteed assembly using analytical and sampled maps was discussed in a prolix manner by [10] in his thesis in 2006. However, producing analytical depth-map requires painstaking effort to generate equations for three-dimensional volumetric intersections for each new assembly [11]. On the other hand, generating the depth-map by sampling using a robot takes considerable time if the shapes of the assembly parts are changed [12]. A clue for the usage of CAD environment for the purpose of generating the depth-map was given in [10]. With the evolution of technology, CAD systems have now become inexpensive and user friendly. This has enabled unskilled or semi-skilled users in the industry to work with CAD systems effortlessly. A wide range of robotic tasks were handled using CAD systems over last two decades. Robot path planning for spray painting using CAD-based systems was reviewed by [5], where tessellated CAD models were shown to be advantageous for surface spray painting. On a similar line, a CAD and scanner based robotic coating system was proposed by [6]. A CAD interface for the programming of automatic welding robot was discussed by [7], where Autodesk AutoCAD DXF file was used to extract the design information from the CAD using simple application. The ability of a CAD platform like Autodesk Inventor to be accessed by any programming platform via COM-API interface makes it possible to rapidly design any new system simulation [13]. A direct off-line robot programming via Autodesk Inventor was proposed in [9], where a Microsoft Visual Basic programming environment was used to extract the data from the Autodesk Inventor's active workspace containing the part drawing.

In this paper, a framework for offline depth-map preparation using a CAD package, e.g., Autodesk Inventor is proposed. Autodesk Inventor can be downloaded freely from [14]. The framework takes CAD assembly model of a mating peg and hole as input, and generates readable text data file containing the depth-map. The proposed system uses the Microsoft Visual C# .NET platform to generate the standalone application to communicate with the Autodesk Inventor having the assembly file opened.

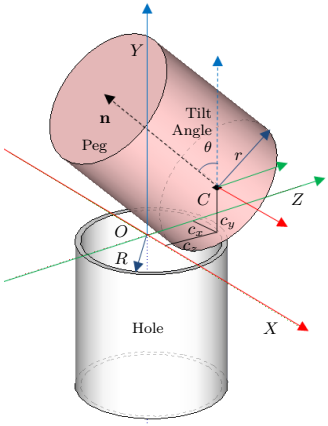

Fig. 1. A peg is in contact with a hole

The paper is organized as follows: Section II discusses the steps required for assembly design in Autodesk Inventor and organization of the proposed software framework. Section III will discuss the application of the proposed

[1]Arun Dayal Udai is a faculty in the Department of Mechanical Engineering, BIT Mesra, Ranchi. arun_udai@yahoo.com

[2]Subir Kumar Saha is a faculty in the Department of Mechanical Engineering, IIT Delhi, New Delhi. saha@mech.iitd.ac.in

978-1-5090-5203-5/16/ $31.00 ©2016 IEEE

framework for a cylindrical, square and hexagonal peg-in-hole assembly and compares it to the depth-map generated, analytically as well as with the existing literature. Section IV concludes the paper.

II. System Framework

Autodesk Inventor exposes its programming interface using a binary-interface standard introduced by Microsoft. It is known as Component Object Model (COM). Inventor supports this inter-process communication and dynamic object manipulation using various programming languages through its Application Programming Interface (API). Typically, any programming language that supports the creation of Dynamic Link Library (DLL), such as Visual C++ and .NET languages like VB.NET, C#, etc. can be used. However, in the proposed framework, the server programs to perform different graphical manipulation and data extraction were done using Microsoft Visual C#.NET. The Autodesk Inventor's API is available in the form of DLL utility files, which are included in the Software Development Kit (SDK). The SDK comes with standard installation package of Inventor and gets automatically installed when we install Autodesk Inventor. Figure 2 shows the structure of Autodesk COM-API system. The following subsections will elaborate different aspects of the proposed framework.

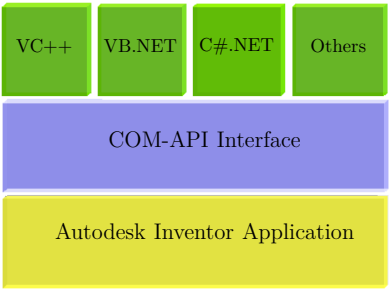

Fig. 2. Autodesk's COM-API model

A. Mechanical Assembly in Autodesk Inventor

A simplified model of the peg-in-hole assembly for which depth-map is to be generated was considered. For this, the peg and the hole on a block were generated separately in the Autodesk Inventor's part modeling environment. These parts were imported individually to the Autodesk Inventor's assembly environment and the final assembly was generated by applying the constraints. These constraints are just for representation and visualization purpose. For API programming, however, one needs to suppress these constraints so that the parts, i.e. the peg, can be moved in all the available directions. The final assembly is shown in Fig. 3, where the bottom rectangular block has the *hole* and the top block is the *peg*.

B. The software framework

The software program that connects to Autodek Inventor through COM-API is commonly known as 'Add-In' [15]. Such program typically begins with a reference to the API's DLL file '*Autodesk.Inventor.Interop.dll*'. This contains all necessary definitions of various API functions to access the properties of the CAD objects. The properties are required to perform any geometrical transformation, and to access various status like collision, etc. The proposed server code connects to Autodesk Inventor, while the peg-in-hole assembly was opened in its workspace. This is shown in Listing 1.

Listing 1. Typical C# code to connect to Autodesk Inventor

```
1  using Inventor;
2  namespace PeginHoleAddinforInventor
3  {
4    public partial class PeginHoleAddin : Form
5    { //Interface definitions
6      Inventor.Application inventorApplication;
7      Inventor.AssemblyDocument assemblyDocument;
8      Inventor.AssemblyComponentDefinition
           assemblyComponentDefinition;
9      //Assembly components definitions
10     Inventor.ComponentOccurrence HolePlate;
11     Inventor.ComponentOccurrence Peg;
12     //Object collection - variable for colliding
           objects
13     Inventor.ObjectCollection partCollectionCollision;
14     //Routine to connect to Autodesk Inventor
15     private void cmdAssembly_Click(object sender,
           EventArgs e)
16     {
17       try
18       { //Acquire assembly information
19         assemblyDocument = (Inventor.AssemblyDocument)
               inventorApplication.ActiveDocument;
20         assemblyComponentDefinition = assemblyDocument
               .ComponentDefinition;
21         //Acquire active parts (indexed objects)
               present
22         HolePlate = (Inventor.ComponentOccurrence)
               assemblyComponentDefinition.Occurrences
               [1];
23         Peg = (Inventor.ComponentOccurrence)
               assemblyComponentDefinition.Occurrences
               [2];
24         //Set the collision counter to Null
25         partCollectionCollision = inventorApplication.
               TransientObjects.CreateObjectCollection(
               null);
26         //Add the parts to the object collection
27         partCollectionCollision.Add(HolePlate);
28         partCollectionCollision.Add(Peg);
29       }
30       catch (Exception ex)
31       {
32         MessageBox.Show("Unable to connect: ", ex.
               Message);
33         return;
34       }
35     }
36   }
37 }
```

In order to have the depth-map of [10], the peg was provided with a small tilt before the peg starts moving

978-1-5090-5203-5/16/ $31.00 ©2016 IEEE

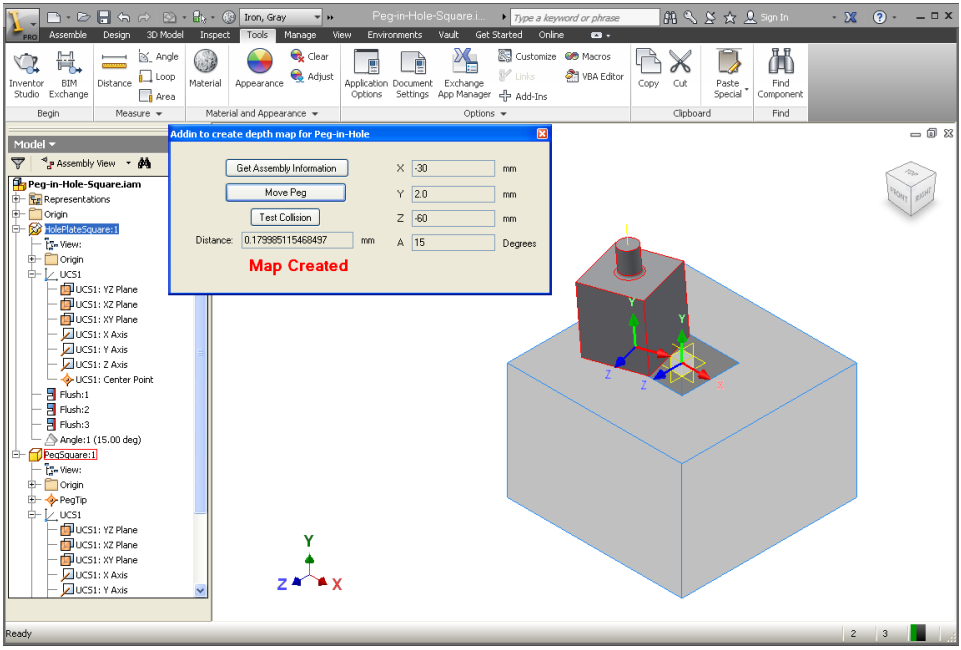

Fig. 3. CAD assembly of a square peg-in-hole with C#GUI in action.

over it while maintaining a continuous contact with the top surface of the hole. The API supports *Add-In's* with transformation matrices for translation and rotation of the geometrical parts present in the active Inventor environment. A sample code in C#.Net to perform translation and rotation of the peg in the Inventor workspace is shown in Listing 2.

Listing 2. C# code for translation and rotation of the peg

```
1  //Routine to translate the peg
2  private void SetTranslate(double MoveX_mm, double
       MoveY_mm, double MoveZ_mm)
3  { //API accepts distances in Centi-meters only
4    MoveX_mm = MoveX_mm / 10; MoveY_mm = MoveY_mm / 10;
       MoveZ_mm = MoveZ_mm / 10;
5    //Initial position of the peg
6    double HoleCentreX = 100, HoleCentreY = 110,
       HoleCentreZ = 100;
7    //Acquire assembly information to the defined
       variables
8    assemblyDocument = (Inventor.AssemblyDocument)
       inventorApplication.ActiveDocument;
9    assemblyComponentDefinition = assemblyDocument.
       ComponentDefinition;
10   //Matrix and Frame definitions for the peg
11   Matrix MatrixTransformPeg =
       assemblyComponentDefinition.Occurrences[2].
       Transformation;
12   Vector TranslationPeg = assemblyComponentDefinition.
       Occurrences[2].Transformation.Translation;
13   TranslationPeg.X = MoveX_mm + HoleCentreX;
14   TranslationPeg.Y = MoveY_mm + HoleCentreY;
15   TranslationPeg.Z = MoveZ_mm + HoleCentreZ;
16   MatrixTransformPeg.SetTranslation(TranslationPeg,
       false);
17   assemblyComponentDefinition.Occurrences[2].
       SetTransformWithoutConstraints(
       MatrixTransformPeg);
18 }
19
20 //Routine to rotate (tilt) the peg
21 private void SetTilt(double Tilt_Angle)
22 { //Acquire assembly information to the defined
     variables
23   assemblyDocument = (Inventor.AssemblyDocument)
       inventorApplication.ActiveDocument;
24   assemblyComponentDefinition = assemblyDocument.
       ComponentDefinition;
25   //Frame definitions for peg
26   Vector XAxisRot, YAxisRot, ZAxisRot;
27   Point OriginPeg;
28   Matrix MatrixTransformPeg =
       assemblyComponentDefinition.Occurrences[2].
       Transformation;
29   MatrixTransformPeg.GetCoordinateSystem(out OriginPeg
       , out XAxisRot, out YAxisRot, out ZAxisRot);
30   //Convert any input angles to radians for API.
       Rotate about X
31   MatrixTransformPeg.SetToRotation(Tilt_Angle*Math.PI
       /180, XAxisRot, OriginPeg);
32   assemblyComponentDefinition.Occurrences[2].
       SetTransformWithoutConstraints(
       MatrixTransformPeg);
33 }
```

In the proposed framework, the surface containing the hole lies in the XZ-plane and Y-axis lies verti-

978-1-5090-5203-5/16/ $31.00 ©2016 IEEE

cal to the surface, as shown in Fig. 3. The `Count` property of the object `assemblyComponentDefinition.AnalyzeInterference` was used to identify if any collision was detected in the CAD environment or not. This was invoked with the parameter `partCollectionCollision` created in the Listing 1. See line 25 of Listing 1. It was created to check collision of the objects under study. So, in order to generate the depth-map, the *peg* was translated in steps over the surface and for each step it was lowered till the state of collision was attained. The coordinate of the peg-center was recorded into the data-file when the *peg* collides with the *hole* block. This process was repeated for all the incremental grid locations at $0.5mm$ step size over the surface of the hole block so as to generate the entire depth-map. The same methodology can be used for any peg-and-hole combination. Steps to implement the proposed technique is given in Algorithm II.1 as follows:

Algorithm II.1: INPUTS($x_{Max}, z_{Max}, \Delta x, \Delta y, \Delta z$)

comment: Algorithm for implementation

START: Peg at the corner of the
bounding box of the desired map region
Use `SetTranslate(x, y, z)` and `SetTilt(Tilt_Angle)`
while $MapCreated \neq True$
do
- **if** $LowerAlongY \leftarrow False$
- **then**
  - $x \leftarrow x + \Delta x$
  - **if** $x \geq x_{Max}$
    - **then** $x \leftarrow -x_{Max}$, $z \leftarrow z + \Delta z$
  - **if** $z \geq z_{Max}$
    - **then** $MapCreated \leftarrow True$, **go to** END :
  - $MoveToNewPosition \cdots$ `SetTranslate;`
  - $LowerAlongY \leftarrow True$
- **if** $LowerAlongY \leftarrow True$
- **then**
  - $Contact\_Count$..`AnalyzeInterference;`
  - **if** $Contact\_Count > 0$
    - **then**
      - **then** **Save** x, y, z, $y \leftarrow$ Above the surface, $LowerAlongY \leftarrow False$
      - **else** $y \leftarrow y - \Delta y$
  - `SetTranslate(x, y, z);`

END: Map Generated

The depth-map was generated for the area defined by the bounding box with corners $(-x_{Max}, -z_{Min})$ and (x_{Max}, z_{Max}). The increments Δx and Δz may be chosen depending on the desired resolution of the depth-map. The peg was lowered with an increments of Δy. The values of increments were dependent on the accuracy of the depth-map required. The overlaid window on top of Inventor CAD workspace in Fig. 3 shows the Graphical User Interface (GUI) developed to interface with the Inventor software. The tool from the API, namely, `inventorApplication.MeasureTools.GetMinimumDistance(HolePlate, Peg)` was used to check the minimum distance between the peg and the hole block.

III. Results and Discussions

In order to validate the data generated by the proposed framework, a cylindrical peg of diameter $19.0mm$ was used with the hole diameter of $19.6mm$. The CAD models of the peg and the hole block were prepared with the same size as that of an existing experimental setup shown in Fig. 4. In Fig. 4, a robot is holding the peg using its gripper and the hole block is lying below. Figure 5 shows a sample depth-map generated when the peg was tilted with an angle of 10°. The steps for increments along X and Z were taken as $0.5mm$ and the resolution for depth along Y was taken as $0.1mm$. However, these increments can be lowered further in order to generate more dense map.

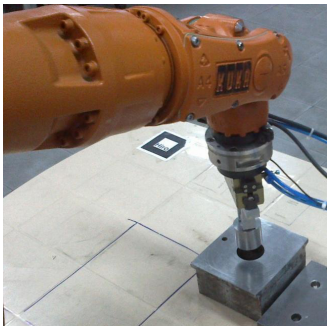

Fig. 4. Actual Peg-in-Hole setup.

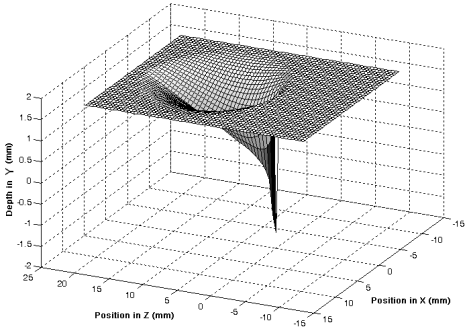

Fig. 5. A sample depth-map for cylindrical peg and hole

978-1-5090-5203-5/16/ $31.00 ©2016 IEEE

This result is similar to the one which was shown in [2] and [10] for the cylindrical peg. However, their peg diameter were different, which was $100mm$. In order to validate the accuracy of the depth-map generated with the proposed framework, a depth-map using analytical technique as discussed in [16] was generated for the same set of the peg and hole. Figure 6 shows the plot of the normal deviation of the generated depth-map compared with the analytical depth-map. The difference in the results is almost negligible in most of the region, thereby confirming the correctness of the proposed approach. Steps to generate the difference map is provided in Appendix I.

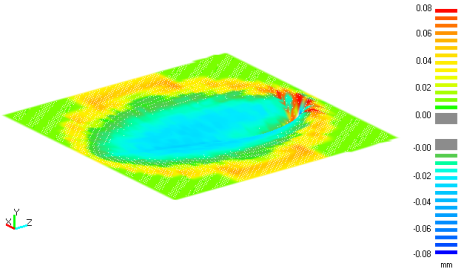

Fig. 6. Plot of differences in depth-map

The framework was also used to generate depth-map for peg and hole of two more shapes, namely, square and hexagon. This was done to test the applicability of the proposed framework to more generic shapes. The peg was rotated first about vertical $Y-axis$ by 25°, followed by a tilt of 15° about horizontal $X-axis$, as shown in Fig. 3. The depth-map was then generated similar to the cylindrical shaped peg in hole using Algorithm II.1. Figure 3 shows the CAD setup for the square shape in Autodesk Inventor. The results are shown in Fig. 7, which is similar to the one reported by [1] for the square shaped peg and hole.

For the depth-map generation of a regular hexagonal shape, which is reported for the first time, the edges were taken as $24mm$ and $24.5mm$, for the peg and hole, respectively. Figure 8 shows the depth-map for their relative orientations similar to that of the square peg-in-hole shown in Fig. 7, i.e., rotation about $Y-axis$ by 25° and tilt about $X-axis$ by 15°. For the comparison of efficiency achieved using the proposed framework, experimental depth-map using a cylindrical peg of $19.0mm$ diameter having a tilt angle of 10° was generated, where the hole was $19.6mm$ in diameter. A robotic assembly was considered using KUKA KR5 Arc robot. This was compared with the depth map generated using the proposed CAD based framework for the same setup as in

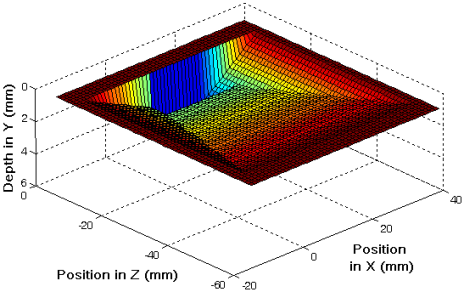

Fig. 7. Depth map for a square peg-in-hole.

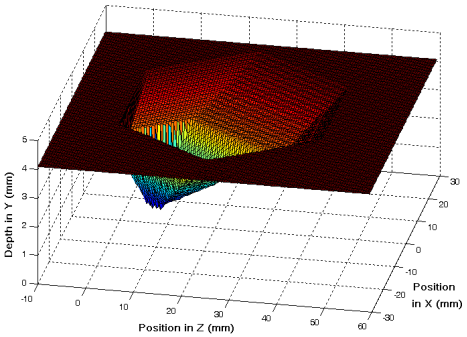

Fig. 8. Depth map for a hexagonal peg-in-hole.

the above experimental setup. The overall time taken for generating the depth-map using real robot was 320 minutes, whereas, using the proposed framework it took only 15 minutes. The resolution along the axes was taken as $0.5mm$ in both the cases. The PC used for the depth-map generation had Intel i7 processor, 2.67 GHz with 4 GB of RAM. This supports our claim of efficient evaluation of the proposed framework. Figure 9 shows the plot of the normal deviation of experimental depth-map to those generated using the proposed framework. A maximum difference at the peaks was only $0.292mm$, which may be attributed to the compliance present in the experimental setup that allows undesired sliding of the peg into the hole when the peg offset is very less.

IV. Conclusions

The paper proposed a framework for the generation of depth-map for any pair of mating parts, e.g., cylindrical, square, hexagon, etc. Depth-maps for a cylindrical and a square peg-in-hole case were compared with the existing literature, whereas the hexagon shape was produced to demonstrate a different shape. Such a generic tool

978-1-5090-5203-5/16/ $31.00 ©2016 IEEE

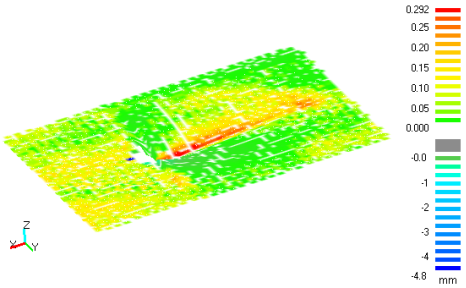

Fig. 9. Plot for the difference in depth map with experimental data.

has not been reported earlier in the literature and is considered here as the main contribution of this paper. Such maps are useful for industrial assembly tasks using dynamic coordinate measuring via robot controller or otherwise, while the assembly is in progress. The methodology to generate the depth-maps reported in this paper is also useful for automatic adaptation, flexible industrial automation, offline programming, robotic localization, etc. The examples presented in this paper may be downloaded from [17].

Acknowledgments

The reported research was carried out at IIT Delhi under the project "Setting up of Programme for Autonomous Robotics" sponsored by BARC, Mumbai.

Appendix I

Comparison of two depth-maps

A depth-map basically contains the coordinates of the peg center with respect to the hole center when a contact is established. This can be visualized in any cloud points handling software, say, using Meshlab or FreeCAD, which are free tools for such visualization. A professional software package called Imageware was used here for comparing depth-maps prepared using the proposed CAD-based technique and using analytical technique. Following are the steps to perform such analysis:

- Import analytical cloud points.
- Generate planar cloud profiles using parallel cloud cross-section tool.
- Fit curves on the planar cloud profiles.
- Reduce the knots on the curve and make the curves compatible for surface generation.
- Generate loft surface using the curve set.
- Import the generated cloud points in the same reference frame.
- Use surface to cloud difference measuring tool.
- Generate the color plot.

Due to space limitation complete description is not included here, but a detailed video for the above steps can be seen at [18].

References

[1] S. R. Chhatpar and M. S. Branicky, "Localization for robotic assemblies with position uncertainty," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, vol. 3, Oct 2003, pp. 2534–2540.

[2] K. Sharma, V. Shirwalkar, and P. K. Pal, "Intelligent and environment-independent peg-in-hole search strategies," in *IEEE Int. Conf. on Control, Automation, Robotics and Embedded Systems*, Noida, India, December 2013, pp. 1–6.

[3] R. Li, R. Platt, W. Yuan, A. ten Pas, N. Roscup, M. A. Srinivasan, and E. Adelson, "Localization and manipulation of small parts using gelsight tactile sensing," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, Illinois*, September 2014, pp. 3988–3993.

[4] J. Deiterding and D. Henrich, "Automatic adaptation of sensor-based robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, Oct 2007, pp. 1828–1833.

[5] H. Chen, T. Fuhlbrigge, and X. Li, "A review of CAD-based robot path planning for spray painting," *Industrial Robot*, vol. 36, no. 1, pp. 45–49, 2009.

[6] Z. M. Bi and S. Y. T. Lang, "A framework for CAD- and sensor-based robotic coating automation," *IEEE Trans. on Industrial Informatics*, vol. 3, no. 1, pp. 84–91, 2007.

[7] J. N. Pires, T. Godinho, and P. Ferreira, "CAD interface for automatic robot welding programming," *Industrial Robot*, vol. 31, no. 1, pp. 71–76, 2004.

[8] P. Neto, N. Mendes, R. Araujo, J. N. Pires, and A. P. Moreira, "High-level robot programming based on CAD dealing with unpredictable environments," *Industrial Robot*, vol. 39, no. 3, pp. 294–303, 2012.

[9] P. Neto and N. Mendes, "Direct off-line robot programming via a common CAD package," *Robotics and Autonomous Systems*, vol. 61, pp. 896–910, 2013.

[10] S. R. Chhatpar, "Localization for robotic assemblies with position uncertainity," Ph.D. dissertation, Case Western Reserve University, Department of Mechanical and Aerospace Engineering, University in Cleveland, Ohio, 2006.

[11] S. R. Chhatpar and M. S. Branicky, "Particle filtering for localization and in robotic assemblies with position uncertainty," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2005)*, 2005, pp. 3610–3617.

[12] Y. Kim, B. Kim, and J. Song, "Hole detection algorithm for square peg-in-hole using force-based shape recognition," in *8th IEEE Int. Conf. on Automation Science and Engineering*, Seoul, Korea, August 2012, pp. 1074–1079.

[13] S. Ivaldi, V. Padois, and F. Nori, "Tools for dynamics simulation of robots: a survey based on user feedback," http://arxiv.org/abs/1402.7050, Cornell University Library, Tech. Rep., 2014, arXiv:1402.7050v1 [cs.RO].

[14] Autodesk. (2015) Autodesk Inventor download site. http://www.autodesk.com/education/free-software/inventor-professional. [Online; accessed 07-October-2015].

[15] ——, "Autodesk Inventor API Documentation," 2013, version 17.6.

[16] A. D. Udai, R. P. Joshi, and S. K. Saha, "Depth-based localization for robotic peg-in-tube assembly," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS 2015)*, Hamburg, Germany, 2015, pp. 3538–3543.

[17] A. D. Udai. (2015) Website to download the framework with the examples presented in this paper. http://web.iitd.ernet.in/~mez108088/todownload/adu-raha2016-depth-map.zip.

[18] ——. (2015) Video link for the use of the presented framework and steps for analysis. https://www.youtube.com/watch?v=3SlgutpQ7ic.

978-1-5090-5203-5/16/ $31.00 ©2016 IEEE