BACHELOR

Modelling a table using the Hero robot

Beurskens, Thijs

Award date:
2022

Modelling a table using the Hero robot

T.M.H. Beurskens 1310909

Supervisor: dr.ir. M.J.G. van de Molengraft
Coach: ir. P. van Dooren

Eindhoven, January 19, 2022

Contents

1 Introduction

This report provides a description and evaluation of a method to generate a table model, using the sensors of the Hero robot. The Hero robot is a standard Toyota HSR robot, which runs custom code developed by Tech United Eindhoven. For the purpose of this project, the most important aspects of the robot are the depth camera at the top of its head, which outputs RGBD images, the laser depth sensor on its base, which outputs 2D depth information and the localisation of the robot, which provides estimates for the coordinates and rotations for each part of the robot. The localisation of the robot is based on a custom made world representation system called the environment descriptor, which is also developed by Tech United Eindhoven [7].

1.1 Features and limitations

The model provided by the proposed method will be used by the robot to interact with the real world table. These interactions include manipulating objects on the table, navigating around the table, and using the table as a reference for localisation. To achieve this goal, two separate parts of the table are to be modelled, namely the surface of the table, and the legs of the table. The surface of the table is what the robot will be dealing with the most, such as when it has to lift objects off of the table or place objects on top of it, so it is important that this area is modelled as accurately as possible. For object manipulation, only modelling the surface would be sufficient, however, the model must also work to help with localisation. The large flat surface of the table can also help with this of course, but the robot also has an infrared distance sensor in its base, which is mostly used for localisation. In order for the table to be recognisable for this sensor, the legs must also be modelled.

The surface of the table will be modelled as a parametric 2D shape, parallel to the ground, at a certain height. If a parametric shape cannot properly describe the table surface, a 2D mesh will be used instead. The legs of the table should be modelled as a flat shape at the height of the laser sensor, defined only by the laser sensor data. This is meant to help the robot place the table into the environment descriptor, as well as localising the robot once the table is placed.

To limit the scope of the project, the prototype will deal only with generating the top surface of the table. Furthermore, the area around the table should be clear, to simplify segmentation. In a real environment, this requirement may not always hold, so future research may look into ways to drop this requirement. A table is always assumed to have a flat top surface. A table that is not flat on the top is extremely rare, and in the case that the robot does encounter one, it should probably not try to place or manipulate objects on it anyway, since this introduces a high risk of objects falling over. Since the robot cannot detect transparent objects, a glass table cannot be considered.

1.2 Method

To generate a table model, the robot captures several depth images of the table from different viewpoints. The location estimate is also recorded for each depth image. The depth images are then converted into point clouds. The point clouds are individually segmented using a combination of Euclidean clustering and RANSAC plane fitting, so that only the surface of the table is left. Using the location estimates as a starting point, iterative closest point is used to align the different point clouds into a single cloud. This cloud is then flattened by removing the height coordinates, and a 2D concave hull is generated around the cloud. RANSAC is then used to fit primitives onto this concave hull. If the fit is not good enough, a mesh of the concave hull is used as a backup.

2 Literature

In this section, different pre-existing techniques are explored, which may be useful for the project. The sensors of the Hero robot output depth images. These are images where each pixel has an additional depth value. In order to merge different viewpoints into the same 3D space, and to easily manipulate the data in different coordinate systems, the depth images are converted to point clouds. This point cloud data has to be processed to generate a 3D model. Several different types of techniques are required to achieve this:

- Fitting primitives
- Registration
- Segmentation
- Filtering

2.1 Fitting primitives

The most important technique is fitting the primitives that will make up the model. A primitive is a simple shape like a cuboid or a sphere in 3D, or a rectangle or circle in 2D. Primitives come in all shapes and sizes, and are usually defined by their location, rotation and some internal variables that determine shape and size. In the case of the table, since the table surface geometry is flat, the point cloud can be collapsed into 2D to fit a 2D shape instead. [1] fitting a 2D shape is much easier because there are only 2 translational dimensions and 1 rotational dimension instead of 3 translational and 3 rotational dimensions. Before collapsing the point cloud, it must be certain that the cloud is oriented properly with respect to the ground. This may be achieved by fitting a plane to the ground, and using the rotation of that plane to rotate the cloud. Another option is to use the internal position encoders of the robot to determine the camera rotation with respect to the ground.

2.1.1 "Click & Clear" procedure

This technique takes a seed point and a basic shape as an input to generate a primitive around that point. First, an initial guess is generated using the points closest to the seed point. The model parameters of this initial guess are used to determine a safety margin, which is used to create a region of interest. Only points within this region of interest are considered, to reduce compute time. Of the points inside the region of interest, a set of potential inliers is created, based on their distance to the current guess. Based on these new inliers, a new guess is generated, along with a new safety margin and region of interest. This process is repeated until the model converges. It is possible to change the primitive type along the way, if a slightly different primitive better describes the data. The model parameters are computed by minimising an error function based on the distances from the inliers to the model. [2]

2.1.2 RANSAC

RANSAC stands for random sample consensus. The technique works by selecting a small random sample of the data, and defining a primitive based on those points. The size of each sample is determined based on which primitive should be fitted, for example, a 2D circle can be defined by three points, so each sample for a 2D circle model would contain three points. Then, the amount of points within a certain threshold of the primitive, also known as inliers, is counted. This threshold is taken from the surface of a 3D shape, or the outline of a 2D shape. This process is repeated a set number of times, after which the model with the largest amount of inliers is chosen. [3]

2.1.3 Local Hough transform

In a local Hough transform, a set of possible primitives is generated based on point pair features. As the name implies, these features are based on point pairs, and they contain the distance between the two points, the angles between the normal vectors and the difference vector, and the angle between the normal vectors. The point pairs are generated by sampling the cloud uniformly, and for each sample, a point pair is created with every other point. The point pair features are then used to cast votes on which parameters are possible. After removing duplicates and refining the possible parameters, the set of parameters with the highest score is chosen. [4]

2.1.4 Comparison

While the "Click & Clear" procedure supports many different types of primitives, it also requires that an initial guess is provided. Since this initial guess cannot be given by a human, the basic shape would have to be determined in some other way. Another problem is that a seed point must be provided which is guaranteed to be on the shape. determining this seed point is also meant to be done by a human. These issues make this approach problematic for use with an autonomous robot. Using a local Hough transform, an initial guess is not required, however this approach only supports planes, spheres and cylinders. RANSAC also works without initial guesses, and while the selection of possible primitives is still limited, there is still a lot more freedom in this aspect than using the local Hough transform. Therefor, RANSAC seems to be the best candidate.

2.2 Registration

Registration refers to stitching multiple point clouds together to create a larger combined point cloud. This is useful when an object can't be fully scanned from a single viewpoint, and multiple viewpoints are required. Most of the time, the exact relative position of these viewpoints cannot be determined beforehand, in which case registration is required. The Hero robot keeps track of its location, but this data is not entirely accurate, so some registration is still required.

2.2.1 Iterative closest point

As the name implies, iterative closest point is an iterative registration method, in which a source cloud is aligned to a static target cloud, by finding the closest point in the target cloud for each point in the source cloud, and minimising the distance between each point pair. This process is repeated until the difference between each iteration approaches zero. This method has an additional parameter, called the maximum correspondence distance. This parameter limits the distance in which the closest point is searched. If there are no points within the maximum correspondence distance, the point is ignored for that iteration. [9]

This technique is useful when prior correspondence data is not available, and it can produce highly accurate results. The main drawback of this technique is that it is very prone to local minima when the initial guess is not good enough.

2.3 Segmentation

Segmentation is the act of splitting up the point cloud into several regions. These regions can then be evaluated separately. It is useful because instead of taking the whole point cloud and trying to find regions that form a single shape, you can focus on fitting the shape that fits best on a particular region. Segmentation is sometimes achieved by fitting course primitives, but other approaches look for sharp edges in the data to cut out points. It is also possible to group together points that are close to each other. [11]

2.3.1 Euclidean clustering

Euclidean clustering is a very simple technique that groups points together based on their distance to each other. Given a maximum distance, and a maximum and minimum cluster size, this method returns a set of clusters, where two points that are closer to each other than the maximum distance will be in the same cluster. In effect, a list of clusters is created with a distance between them that is larger than the maximum distance.

2.3.2 Primitive fitting

It is possible to segment simple shapes by fitting primitives into the point cloud. This approach requires a fitting method that is very robust to outliers, since the subject is likely a very small part of the whole cloud. Different fitting techniques are discussed in section 2.1. This method is very useful for extracting large planes such as floors and walls.

2.3.3 Comparison

Euclidean clustering is a technique that is easy to understand, and it can be used on any shape, making it a versatile option when the exact shape of the object is hard to define or unknown. It does, however, require physical separation and the required cluster may not be immediately apparent. Using primitive fitting, physical separation is not required, and the objects are retrieved from the scene in a predictable way, though not every shape can be found in this way. Euclidean clustering is useful when the exact shape is unknown, and primitive fitting is required when physical separation cannot be guaranteed.

2.4 Filtering

Filtering is applying operations to each point in the cloud individually. This step is used to prepare the point cloud for further manipulation by removing outliers and smoothing the data. [8] It can also be used to estimate normal vectors for each point, which is required for use with a local Hough transform (section 2.1.3), and can help improve results when used with RANSAC (section 2.1.2), if the model supports it. [5]

3 Approach

In this section, the approach chosen to model the surface is described. The approach consists of several different steps. First, the robot uses its sensors to scan the table into a set of point clouds. This data is then segmented, such that only the table itself is left in the data set, and all the other information, such as walls, floors and other objects are discarded. Next, the different snapshots are stitched together into a single point cloud. Once all the data is combined, a fit can be made on the surface of the table.

Segmentation is done before registration, because this removes a the data in the background. This background data does not overlap a lot between the different clouds shots, so if an attempt is made to match them, this can result in many mistakes.

The Point Cloud Library (PCL) will be used as a framework for much of the project. It contains an easy way to store point clouds, as well as a myriad of implementations for filtering, segmentation and registration, including normal estimation, euclidean distance segmentation, sample consensus algorithms and iterative closest point. [10]

3.1 Collecting data

To collect the data for the surface of the table, an infrared depth camera is used, which is located at the top of the robot. This camera returns the distance from each pixel to a normal plane at the position of the camera. This makes it relatively easy to extract 3D coordinates from the depth image, by using the known focal distance of the camera. Each pixel in the depth image is translated into a 3D point, along with a colour value. If there is no depth value for a particular pixel, which can happen when the distance to the pixel is larger than what the camera can record, the location fields will be set to NaN (not a number). The points are stored in a grid, arranged by their original pixel coordinates. As such, it is possible to find out which point corresponds to which pixel in the original image. Figure 2 shows an example of an input point cloud generated in this way. Figure 1 shows only the colour information for this cloud. A metadata file is also generated during the creation of the depth image, which holds the estimate for the location and rotation of the camera when the image was captured. The pitch, roll and height of the camera are determined by the internal position encoders in the robot. As such, these values should be relatively accurate. The yaw and horizontal location of the camera are determined by recording the route that the robot takes, and combining this data with environmental clues, such as where the walls are. This estimate is less accurate, since it cannot be directly measured inside the robot.

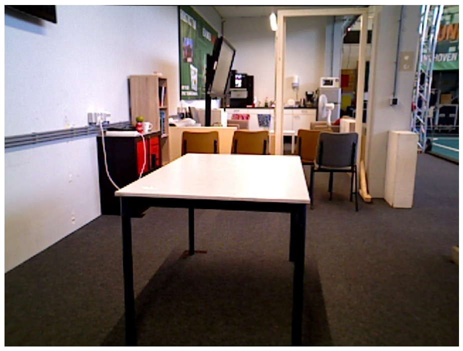

Figure 1: A single snapshot from the depth camera, colour information only

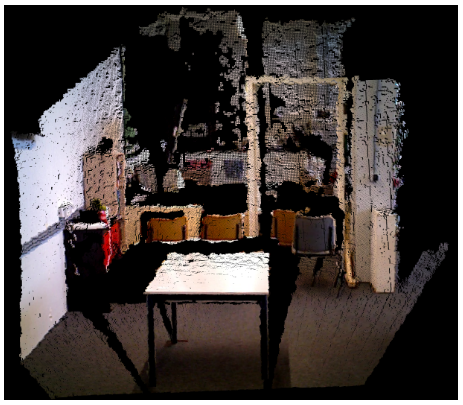

Figure 2: A single snapshot from the depth camera, converted into a point cloud

3.2 Segmentation

Segmentation refers to dividing the data set into coherent groups. In this case, the group we are interested in is the group of points that lie on the table. Finding these points is achieved by first translating each point cloud from camera coordinates to global coordinates, using the metadata file. Then, the ground plane is removed, by discarding every point below 0.1 metres. This threshold was chosen to make sure every ground point is removed, while preserving the table surface, even when the location data is inaccurate. With the ground plane removed, the cloud is cut up into clusters of points that lie close to each other. Of these clusters, the cluster closest to the camera is chosen. If there are no obstacles between the camera and the table, this will always result in successfully retrieving the table. The result of this approach can be seen in Figure 3. As an additional step, RANSAC (section 2.1.2) is used to fit a horizontal plane on the retrieved cluster. An inlier threshold of 3 centimetres is used for this, so that even with a slight misalignment in the rotation of the table surface, all surface points can fall within the threshold. Points below this plane are discarded from the cluster. This is done, because rotational errors in registration result in protrusions in the final result, caused by the table legs. The protrusions happen because the point cloud is An example of this happening is shown in Figure 5. By removing the legs, these protrusions disappear. The result of removing the legs is shown in Figure 4.

The height of this plane is also used to determine the height of the table. The recorded heights in each of the input clouds are averaged together, and this value is taken as the height estimate. This approach is used, because it is robust to objects on the table. If, for example, the highest point in each of the clouds is taken, any objects on the table would inflate the height estimate.

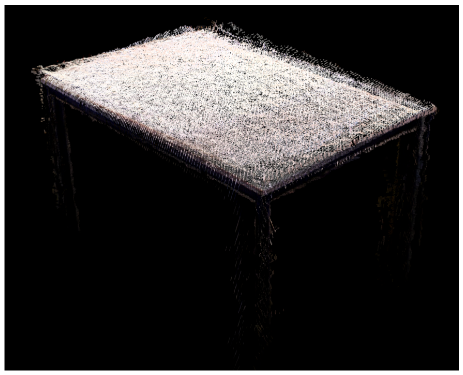
Figure 3: The result of individually segmenting the point clouds using euclidean clustering

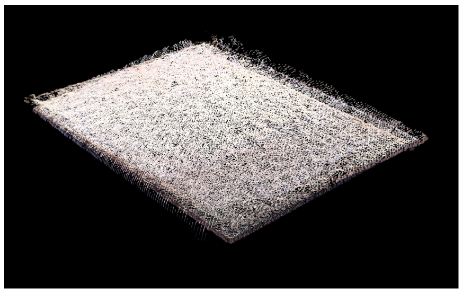
Figure 4: The result of individually fitting planes and removing the legs

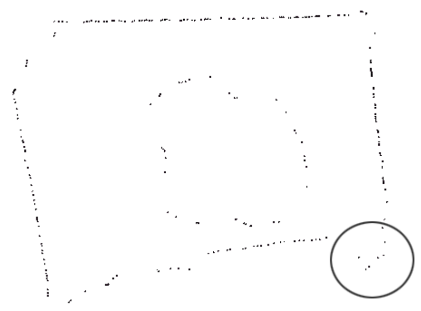
Figure 5: An example of bad registration showing up as a protrusion in the final result, when leg data is not removed

3.3 Registration

During registration, the individual clouds are brought into alignment with each other. The location metadata is used as a starting point for this. Because this data is not entirely accurate, a final alignment step must be performed. This final step is performed using the iterative closest point method (section 2.2.1). This method is prone to local minima, so it is important to have a good estimate as a starting point. Since this method can only process two clouds at a time, the clouds are added one at a time, using the previous result as a target for the next cloud. The result of aligning multiple point clouds is shown in Figure 6

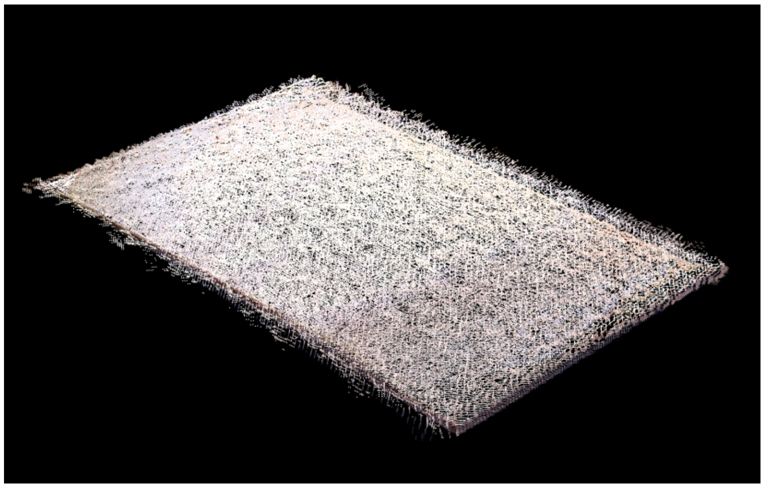

Figure 6: The result of applying iterative closest point, note that the edges are slightly more defined than in Figure 4

3.4 Fitting

The final step of generating the surface of the table is fitting a model onto the full point cloud. Since the surface of the table is assumed to be flat, the height coordinates are discarded at this point. This step also collapses any objects on top of the table into the surface. RANSAC (section 2.1.2) was chosen as the fitting method for it's versatility, and because it is widely used. Since we are now dealing with a 2D shape, the outline of the point cloud is required. This is because the points in the middle of the cloud give very little information about the overall shape. It is therefor more efficient to focus on the points near the edge of the cloud. The outline is generated by making a 2D concave hull around the shape, and taking only the points that lie on this hull. The resulting point cloud can be seen in Figure 7. Once again, an inlier threshold of 3 centimetres is used, which allows for small undulations in the edges of the table, while still eliminating more obvious outliers.

RANSAC is then used to fit a set of primitives onto the dataset, and the primitive type with the highest amount of inliers is chosen as the best fit. This metric is used because it ignores outliers. If, for example, the distance to each point was used to determine the correct primitive, an outlier with a large distance from the shape could skew the result significantly. It is also easy to normalise by dividing the amount of inliers by the total number of points, so that a score from 0 to 1 can be assigned to each primitive. 0 means that there are no inliers, and 1 means that all points are inliers. Since not every table can have a primitive made for it, This score can be used to assess whether or not a certain primitive is good enough, otherwise, a fallback model must be used. The fallback model is a mesh based on the concave hull. This model was picked because the concave hull requires no prior knowledge about what shape might be encountered. Also, by definition, the concave hull always fits the data. A drawback of this approach is that storing the concave hull requires more memory than storing primitive parameters. The concave hull also stores all of the noise in the input, which is unwanted.

In the prototype, two primitives have been implemented, namely a circle and a parallelogram. These shapes were chosen to account for most standard tables. Many more shapes exist, but circles and parallelograms are by far the most common. A circle is very easy to model, since any circle is defined by any three points on its outer edge, so any selection of points on the circle can theoretically be used to correctly model the whole shape. A parallelogram, on the other hand, presents an issue, in that by sampling points on one side, it is impossible to know what the other side looks like. An initial attempt to model a rectangle directly was deemed insufficient, since it required two corner points in a sample of three points, as illustrated in Figure 8. This approach resulted in wrong dimensions, because it is very unlikely that the exact points required are chosen as a sample, and these points are not guaranteed to exist in the first place. Instead, a primitive was created that represents two parallel lines. This primitive only requires that the first two points lie on the same line, and the third point lies on the second line. given the amount of sample sets, this approach is likely to return a large amount of valid models. By removing the inliers from the cloud, and fitting the primitive a second time, two sets of parallel lines are found. The height, width and internal angle of the parallelogram can then be easily extracted.

Figure 7: An example of a concave hull for a rectangular table

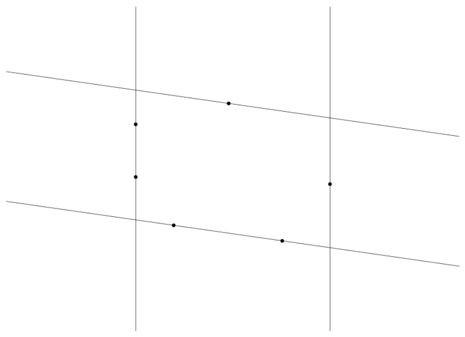
Figure 9: The parallelogram primitive, with defining points for both sets of lines

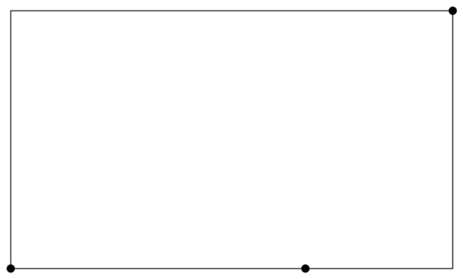
Figure 8: The initial rectangle primitive, with defining points

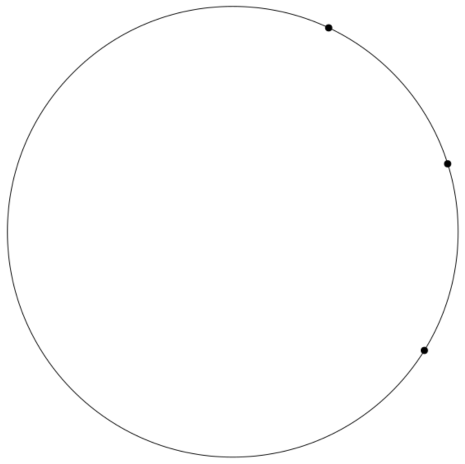
Figure 10: The circle primitive, with defining points

4 Testing

In this section, the prototype will be tested on the basis of the estimated table height and the achieved fit. The method for determining which fit is chosen is also tested. Then, the maximum correspondence distance for iterative closest point is varied (section 2.2.1), and the effects of removing the table legs and/or clutter on the table are tested.

The dataset consists of four different tables. Figures 11 to 14 show schematic images of what each table looks like. Each table is scanned with and without a small amount of clutter on it. Figure 15 shows an example of clutter on a table. The first table is a rectangular table, which is scanned both from the default robot height as well as an extended position. The second table is a small circular table, which is only scanned from the default position. The third table is a small square table, which is also only scanned from the default position. The fourth and final table is a rectangular desk with a cutout on one side, which is only scanned from an extended position. The datasets for the rectangle, circle and square each consist of eight viewpoints, at roughly 45 degree intervals around the table. This ensures that there is significant overlap between the viewpoints, especially around the edges of the tables. Because the desk is larger, ten viewpoints are used for those datasets, where an additional viewpoint is introduced on both the long sides of the desk. This is done so the whole desk can be scanned. The viewpoints were recorded in a clockwise fashion around the table, and the viewpoints are processed in the same order. This is important, because this ensures that neighbouring viewpoints are processed one after the other, and the overlap can be used in the iterative closest point algorithm.

Figure 11: Rectangular table top view

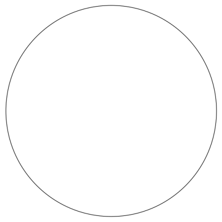
Figure 12: Circular table top view

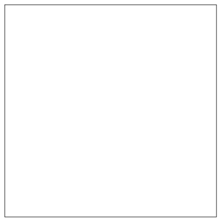
Figure 13: Square table top view

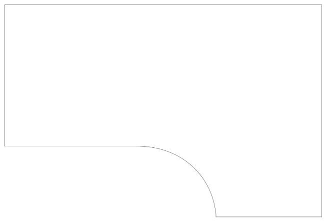
Figure 14: Desk top view

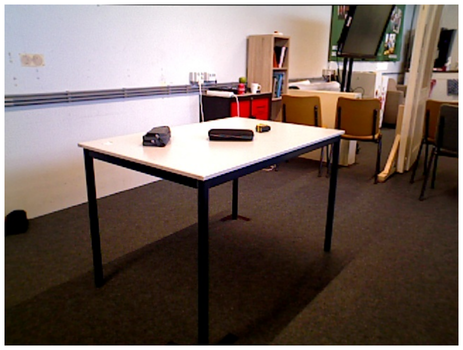
Figure 15: A table with a small amount of clutter on it

4.1 Table height

The obtained height estimates for each dataset are shown in Table 1, along with the measured height of each table. from the data, it seems that when the robot is looking down onto the table, the height estimate is overestimated by roughly 1 to 2 cm. The estimated heights of both the rectangular table at default extension and the circular table are almost completely correct. The height estimates of the square table are also 1 to 2 cm higher than expected.

The difference between the different errors could be explained by the fact that the robot looks at the different tables from different angles. When the robot looks from a shallow angle, it will see more of the side of the table, and try to include these points into the plane fit. This pulls the estimate down. When the robot looks down at a high angle, the side of the table is less visible, which means the height estimate is pulled down less. The fact that the height estimates from the shallow angle datasets are better indicates that there might be a constant error in the height localisation of the robot, which pushes all points up slightly. This would result in the data from shallow viewing angles to be pushed up to the correct height, while data from high viewing angles (which should be more correct) is pushed up away from the correct height.

All estimates are within 2 cm of the ground truth, which should be sufficiently accurate for the robot to use.

The height estimate is unaffected by the maximum correspondence distance and the removal of the legs or clutter on the table, since it is generated before the legs or clutter are removed, and before registration. Therefor, it will no longer be included in the next subsections.

Table 1: Estimated and measured heights for all datasets

	Extended Rectangle clear	Extended Rectangle cluttered	Default Rectangle clear	Default Rectangle cluttered		
Estimated height (m)	0.779	0.776	0.755	0.758		
Ground truth (m)	0.755	0.755	0.755	0.755		
	Circle clear	Circle cluttered	Square clear	Square cluttered	Desk clear	Desk cluttered
Estimated height (m)	0.619	0.615	0.472	0.473	0.736	0.736
Ground truth (m)	0.620	0.620	0.455	0.455	0.720	0.720

4.2 Maximum correspondence distance

One of the variables that was tested is the maximum correspondence distance for the iterative closest point registration. This variable determines how far a point pair can be apart to be considered as a correspondence. A lower value makes the algorithm quicker to compute, while a higher value may yield a higher accuracy, up to a point. This is because the maximum correspondence distance limits the area in which correspondences can be found. When this value is low, a lot of correspondences are skipped, which decreases compute time, but those skipped correspondences might contain useful data. A higher value means more points are considered, which increases compute time, but the chances of missing an important correspondence is lower. In Appendix A, Table 3, each dataset is processed with a set of maximum correspondence distances, and the estimated model parameters are shown. A fail indicates that the wrong type of model was chosen as a fit. The desk is not included in this table, since there is no correct fit for those datasets. The processing time was not recorded, because the current prototype is not optimised for speed at all. The focus here lies in maximising accuracy first.

As expected, a high correspondence distance gives the best results. At a maximum correspondence distance of one metre, the widths and lengths of all rectangles are within 2 or 3 centimetres of the ground truth. The sizes of the circle and square are consistently overestimated by 5 to 10 centimetres. This indicates a suboptimal registration. When the registration is incorrect, there is some spread in the point clouds. when an outline is drawn around this spread, the outline will be larger than any of the individual clouds.

The angles are mostly within 3 degrees of the ground truth. The default height rectangular table without clutter is a notable exception where the angle is surprisingly high. This can also be explained by a suboptimal registration. In this case, one or more of the clouds are rotated with respect to each other. When taking the outline of the combined point cloud, some of the sides may be rotated. The effect of slight clutter on the table parameters is very small. Increasing the maximum correspondence distance any further will not have any effect on the smaller tables, since all points can already correspond to any other point.

Dropping to a lower maximum correspondence distance of 0.1 metres still yields mostly the same results. Some of the length and width estimates are slightly higher, which corresponds to a slightly worse registration. The default height rectangular table without clutter is once again an outlier, but this time the length estimate is significantly higher. This trend continues for 0.05 metres and no registration. It seems like this dataset has a particularly bad initial guess, which makes it difficult to achieve a satisfactory registration

Dropping down to 0.05 metres, The lengths and widths once again increase by a few centimetres. The error is now up to roughly 5 centimetres for rectangle and square datasets, and closer to 10 centimetres for circle datasets.

When no registration is performed at all, so only the pre-existing location data is used, there is once again a similar size increase. Notably, the circle is no longer recognised at this point. The algorithm determined that the circular table fits closer to a parallelogram. The square is also hit particularly hard by this step, increasing up to 10 centimetres with regards to the previous step. This indicates that the initial guesses for the square and circle datasets are worse than for the rectangle datasets, which might be the reason why those have underperformed so far. Comparing the size parameters in the case where no registration is performed to the ground truth, shows that the spread of the localisation can reach up to roughly 20 centimetres.

4.3 Removing legs/clutter

During the height estimate step, the legs of the table are removed, by removing every point underneath 1 cm below the height estimate. This threshold was chosen to minimize the amount of points underneath the surface of the table, while avoiding cutting out the surface as well. It is also possible to remove the objects on top of the table using a similar method. In this test, there are reference values, using the default behaviour of only removing the legs. In addition, there are values for removing both legs and clutter, and removing nothing. All tests use a maximum correspondence distance of 1 metre. The full results are shown in Appendix A, Table 4.

When cutting both the legs and the clutter, a few results become noticeably better, but many results become completely wrong. Looking at the point clouds reveals why. Because the height estimate is slightly lower than the actual cloud data, the whole table plane is cut out. The effect this has on registration is shown in Figure 16.

While not cutting any data does not completely break the algorithm, it does not seem to help in any of the test cases in any meaningful way. In some cases, the table legs do also interfere with registration. This happens because there is very little data on the legs, which means that each side of the legs is only seen once, and only partially. When trying to align this incomplete data, a mistake is easily made. A good example is shown in Figure 17. In this case, the incomplete leg data has caused parts of the table to rotate upwards, to try to fit the legs together. This also explains why the angle and size estimates for this dataset are completely wrong.

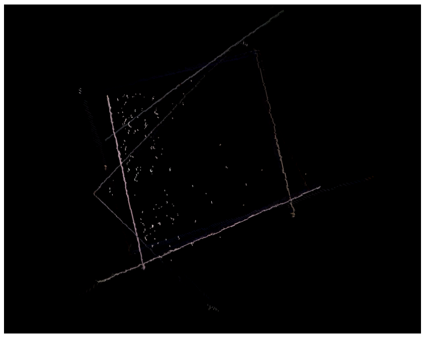
Figure 16: Rectangular table with clutter, with legs and clutter cut out

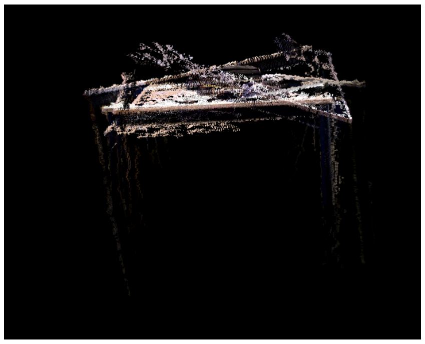
Figure 17: Rectangular table with clutter, with nothing cut out

4.4 Model selection

The current method to select which model to use, fits both models, and simply counts the amount of inliers. The amount of inliers is then divided by the total number of points in the concave hull, to determine a normalised score. The model with the highest score is chosen as the correct model. For most datasets, this approach works quite well. Table 2 shows the scores for each dataset, using default settings. The approach works for every dataset where a "correct" model exists. The desk is determined to be a parallelogram with this algorithm. This makes sense, since it has some long straight edges, but it doesn't really fit in either category.

To determine whether or not the shape can be fitted using the existing models, this score does not seem entirely reliable. This is because the desk is deemed to be a better rectangle than the rectangular table, when measured from a shallow angle. However, looking at the concave hulls used to fit those rectangles, it becomes clear why they have such a low score. Because of the shallow angle, only the front part of the table surface could be recorded for each viewpoint. In the case of the clear table, this has resulted in an apparent hole in the convex hull. Every point that lies on the boundary of that hole is considered an outlier. For the cluttered table, the lack of data has resulted in one or more of the input clouds rotating by 90 degrees. The resulting protrusions also account for a lot of outliers. Based solely on those concave hulls, it would not be easy to say if those datasets correspond to rectangles or not. In these cases, better registration is still required. All the other shapes reach a score of at least 0.900, which is higher than the maximum of 0.868 of the desk. Setting the threshold between these values would have worked for this particular dataset, but the margin is very small.

Table 2: Score, determined by dividing the amount of inliers by total amount of points for both the parallelogram and circle model

	Extended Rectangle clear	Extended Rectangle cluttered	Default Rectangle clear	Default Rectangle cluttered		
Parallelogram score	0.956	0.990	0.763	0.756		
Circle score	0.400	0.416	0.371	0.308		
	Circle clear	Circle clutter	Square clear	Square clutter	Desk clear	Desk clutter
Parallelogram score	0.867	0.891	1.000	1.000	0.868	0.805
Circle score	0.900	1.000	0.627	0.628	0.374	0.345

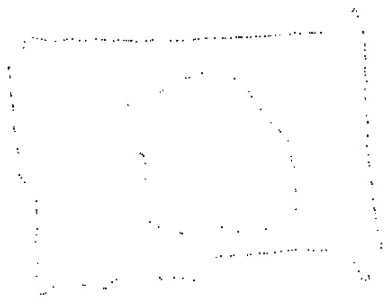
Figure 18: The concave hull for "Default Rectangle clear"

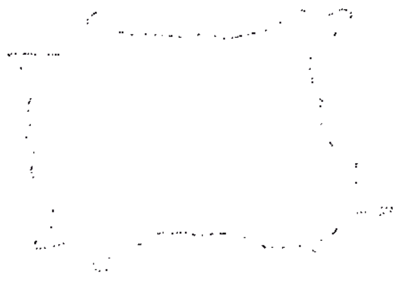
Figure 19: The concave hull for "Default Rectangle cluttered"

5 Discussion

This section will start with a brief reflection on the quality of this report. This is then followed by some suggestions for future work, regarding segmentation, registration, fitting, and the modelling of the legs.

5.1 Reflection

The method was developed in a somewhat disorganised manner. While a clear step by step plan was made beforehand, the exact implementation of each step was not thought out enough. In hindsight, more time should have been spent reading literature before designing the method. If I had a better understanding of the limitations of each method before starting, many problems could have been avoided.

5.2 Segmentation

While the current segmentation method works well for completely free standing tables, it breaks down when a table is against a wall, or surrounded by other objects such as chairs. One possible way to get around this issue, is to use the colour information from the depth camera, and use a neural network to recognise when and where a table is in the image. Since each point in the point cloud corresponds to a pixel on the depth image, it should be possible to find which points lie on the table. This approach may be combined with the current method in case there are multiple tables in the image or the neural network makes any small mistakes.

5.3 Registration

The current registration method, iterative closest point, relies heavily on a good initial position estimate. This estimate is currently provided by the localisation of the robot. This position estimate has proven to be somewhat unreliable, which has resulted in several mistakes in the registration process. There are several ways to help remedy this issue:

- Use more intermediary points
- Add an additional coarse registration step
- Use of normal vectors
- Use of colour information

5.3.1 Intermediary points

By integrating data collection more closely into the method, it is possible to create snapshots at set time intervals. These snapshots may be labelled to be exclusively used to help in registration. by applying iterative closest point to each pair of snapshots in the order they are created, a total transformation from one data point to the next can be computed. If the time interval is short enough, it becomes unnecessary to use the existing localisation, since the difference between the snapshots is already very small. This approach does come with a significant computation time penalty. This approach also introduces the risk that small registration mistakes compound into very large errors.

5.3.2 Additional coarse registration

Another approach would be to use a different registration method, that is more robust to local minima, such as the method proposed by Lin et al, which relies on matching 2D image features [6]. A method such as this could be combined with iterative closest point, by using the output of the new method as a starting point for ICP, in which case the new method does not have to be as accurate as ICP, as long as it improves the registration to some extent. More research is required to determine a suitable method.

5.3.3 Normal vectors

It is possible to compute an estimate of the normal vector for each point. Using this normal vector, a more accurate correspondence may be found. In Figure 20, an example of a dataset that has encountered a local minimum is shown. In this case, one of the clouds is aligned incorrectly, because the wrong side of the pencil case is aligned. By adding the normal vector data, these correspondences could be eliminated, since it is obvious that the normal vectors are pointing in opposite directions. By eliminating such obviously wrong correspondences, many local minima can be eliminated.

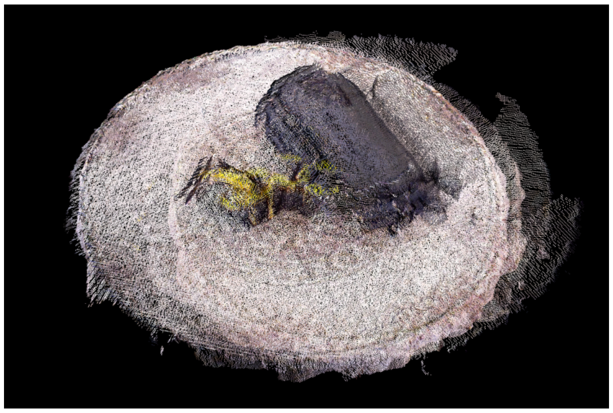

Figure 20: Example of incorrect registration in the case of a circular table with clutter. Note the dark line to the right of the pencil case, which looks like the left side of the pencil case

5.3.4 Colour information

Currently, the colour information of the point cloud is unused. By limiting correspondences between points with different colours, it is possible to use patterns on the table to help with alignment. This will not help in all cases, but it may help on any table with some sort of table cloth, such as in the situation in Figure 20.

5.3.5 Comparison

Using a combination of normal vector and colour information would be a good way to improve the ICP algorithm itself, by removing wrong correspondences and thus removing some of the local minima. If this does not solve the problem, an additional registration step using a different algorithm may provide better results. Using many intermediate points is the least desirable option, because it takes significantly longer to compute, and small registration mistakes can add up to reduce overall accuracy.

5.4 Fitting

When presented with a properly registered dataset, the current method of applying a concave hull and fitting a primitive using RANSAC works quite well. A drawback of this approach is that a single misaligned point cloud can dramatically alter the finished result. This happens because a concave hull is an outline around the whole cloud, and a few points are enough to move the boundary of this outline. It might be a good idea to apply a concave hull to each input cloud individually, after determining the transformation matrices. This would result in a cloud with the outlines of each input cloud overlaid, which would then be the input cloud for RANSAC. Because RANSAC is robust to outliers, any misaligned clouds would be mostly ignored, and a fit would be made on the area with the highest point density. This would create a lot of outliers for some datasets, where each input cloud does not show the whole table. In these cases, there would be additional lines across the table. This might make it more difficult to determine when a table cannot be described using one of the implemented models. It may also be possible to determine when an edge is a result of a lack of data, such as when a point is on the edge of the image, or the depth information is missing for the neighbouring points. By eliminating those incorrect edge points, additional lines across the table could be avoided.

5.5 Table legs

The data required to model the legs of the table can be acquired from the laser sensor at the base of the robot. This sensor outputs a 2D point cloud, which shows objects around the robot at roughly ground level. By taking a snapshot of this sensor at the same time as the depth camera, and recording the position of this sensor in the metadata, the transformation matrices from the surface registration can be reused. This is because the internal encoders of the robot are very accurate, which means that the pose of the laser sensor with respect to the camera is also known very accurately. It might also be possible to use the laser sensor data to improve the registration of the surface data.

To generate the leg models, a similar approach can be used as with the surface, namely a primitive fit using RANSAC. Since the laser sensor can only see the outline of the leg, and not the cross sectial area, it is unnecessary to generate a concave hull for this dataset. The method should be applied multiple times to find all the legs, until a certain percentage of points has been classified as inlier. The current parallelogram model is sadly incompatible with multiple shapes, since it relies on two separate line fits, which are not guaranteed to originate from the same shape. The circle model does not have this limitation. Alternatively, the data can be split up into the individual legs before applying RANSAC, by using a euclidean distance segmenter. This makes it possible to use the current parallelogram model, and each leg can be fitted in the same way as the table surface. Since the exact shape of the table legs is less important, it may not be necessary to use a mesh as a fallback option. This may save space in the final model.

6 Conclusion

In this report, a method for generating a table model using the sensors of the Hero robot has been described and tested. The method uses the depth camera at the top of the robot to generate point cloud data. The points corresponding to the table are then isolated, and different viewpoints of the same table are overlaid to create one point cloud. After flattening this combined cloud and generating an outline of the table, a fit can be made on this outline.

The estimated surface parameters using the proposed method range from within 1 or 2 centimetres of the ground truth, to an error of over 5 centimetres, depending on how well the registration is performed. The heights of the tables are all estimated within 2 centimetres of the ground truth.

While the proposed segmentation method works well for the current datasets, a different method may be required when dealing with more realistic scenarios, where the table is not completely free standing. The registration method also requires improvements, especially when dealing with inconsistent localisation. These problems may be the subject of future work.

A Estimated model parameters

Table 3: Estimated model parameters with varying correspondence distances

Maximum correspondence distance	Parameters	Extended Rectangle clear	Extended Rectangle cluttered	Default Rectangle clear	Default Rectangle cluttered
1 m	Width (m)	0.843	0.830	0.734	0.771
	Length (m)	1.22	1.23	1.20	1.12
	Angle (degrees)	89	91	99	93
0.1 m	Width (m)	0.844	0.837	0.710	0.779
	Length (m)	1.21	1.24	1.37	1.19
	Angle (degrees)	92	89	93	93
0.05 m	Width (m)	0.853	0.823	0.804	0.781
	Length (m)	1.25	1.23	1.35	1.22
	Angle (degrees)	89	89	93	91
no registration	Width (m)	0.865	0.846	0.861	0.814
	Length (m)	1.26	1.25	1.41	1.33
	Angle (degrees)	92	88	90	89
ground truth	Width (m)	0.800	0.800	0.800	0.800
	Length (m)	1.20	1.20	1.20	1.20
	Angle (degrees)	90	90	90	90

Maximum correspondence distance	Parameters	Circle clear	Circle cluttered	Square clear	Square cluttered
1 m	Width (m)	0.608	0.581	0.606	0.591
	Length (m)	-	-	0.607	0.628
	Angle (degrees)	-	-	87	93
0.1 m	Width (m)	0.604	0.582	0.623	0.591
	Length (m)	-	-	0.623	0.616
	Angle (degrees)	-	-	88	93
0.05 m	Width (m)	0.610	0.598	0.596	0.603
	Length (m)	-	-	0.603	0.634
	Angle (degrees)	-	-	87	86
no registration	Width (m)	fail	fail	0.622	0.600
	Length (m)	-	-	0.735	0.692
	Angle (degrees)	-	-	91	96
ground truth	Width (m)	0.500	0.500	0.550	0.550
	Length (m)	-	-	0.550	0.550
	Angle (degrees)	-	-	90	90

Table 4: Estimated model parameters with varying amounts of cutting

	Parameters	Extended Rectangle clear	Extended Rectangle cluttered	Default Rectangle clear	Default Rectangle cluttered
Cut only legs	Width (m)	0.843	0.830	0.734	0.771
	Length (m)	1.22	1.23	1.20	1.12
	Angle (degrees)	89	91	99	93
cut legs and clutter	Width (m)	0.831	0.790	0.634	0.774
	Length (m)	1.26	1.20	1.19	0.808
	Angle (degrees)	105	89	61	78
Don't cut	Width (m)	0.829	0.835	0.226	0.873
	Length (m)	1.21	1.23	1.28	1.13
	Angle (degrees)	90	88	89	77
ground truth	Width (m)	0.800	0.800	0.800	0.800
	Length (m)	1.20	1.20	1.20	1.20
	Angle (degrees)	90	90	90	90

	Parameters	Circle clear	Circle cluttered	Square clear	Square cluttered
Cut only legs	Width (m)	0.608	0.581	0.606	0.591
	Length (m)	-	-	0.607	0.628
	Angle (degrees)	-	-	87	93
Cut legs and clutter	Width (m)	0.570	fail	0.583	0.587
	Length (m)	-	-	0.604	0.615
	Angle (degrees)	-	-	87	87
Don't cut	Width (m)	0.672	0.678	0.583	0.596
	Length (m)	-	-	0.646	0.615
	Angle (degrees)	-	-	85	92
ground truth	Width (m)	0.500	0.500	0.550	0.550
	Length (m)	-	-	0.550	0.550
	Angle (degrees)	-	-	90	90

References

[1] Ahsan Abdullah, Reema Bajwa, Syed Rizwan Gilani, Zuha Agha, Saeed Boor Boor, Murtaza Taj, and Sohaib Ahmed Khan. 3d architectural modeling: Efficient ransac for n-gonal primitive fitting. In *Eurographics (Short Papers)*, pages 5–8, 2015.

[2] Sung Joon Ahn, Ira Effenberger, Sabine Roth-Koch, and Engelbert Westkämper. Geometric segmentation and object recognition in unordered and incomplete point cloud. In Bernd Michaelis and Gerald Krell, editors, *Pattern Recognition*, pages 450–457, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[3] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4(1):2–3, 2010.

[4] Bertram Drost and Slobodan Ilic. Local hough transform for 3d primitive detection. In *2015 International Conference on 3D Vision*, pages 398–406, 2015.

[5] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2684–2689, 2012.

[6] Chien-Chou Lin, Yen-Chou Tai, Jhong-Jin Lee, and Yong-Sheng Chen. A novel point cloud registration using 2d image features. *EURASIP Journal on Advances in Signal Processing*, 2017(1):5, Jan 2017.

[7] R.P.W. Appeldoorn L.L.A.M. van Beek J. Geijsberts L.G.L. Janssen P. van Dooren H.W.A.M. van Rooy A. Aggarwal S. Narla M.F.B. van der Burgh, J.J.M. Lunenburg and M.J.G. van de Molengraft. Tech united eindhoven @home 2022 team description paper. Paper, Tech United Eindhoven, 2022.

[8] Bradley Moorfield, Ralf Haeusler, and Reinhard Klette. Bilateral filtering of 3d point clouds for refined 3d roadside reconstructions. In George Azzopardi and Nicolai Petkov, editors, *Computer Analysis of Images and Patterns*, pages 394–402, Cham, 2015. Springer International Publishing.

[9] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[10] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE.

[11] H. Woo, E. Kang, Semyung Wang, and Kwan H. Lee. A new segmentation method for point cloud data. *International Journal of Machine Tools and Manufacture*, 42(2):167–178, 2002.