2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

A Fast, GPU-based Geometrical Placement Planner For Unknown Sensor-Modelled Objects And Placement Areas

Johannes Baumgartl, Tim Werner, Per Kaminsky, and Dominik Henrich

***Abstract*— A Personal Robot should be able to handle unknown objects in unknown environments. For a manipulation task the question what to do with an object once it had been grasped is one of the most essential ones beside the grasping task itself. Moreover, the planning time should be at least as fast as the time the robot needs for its motions.**

We propose a fast placement planner for sensor-modelled objects in complex environments. The planner computes a stable position and orientation for the object in the environment. The algorithm uses only geometric information, most notably no force or torque sensor is required. In particular, we introduce a novel approach regarding the pose computation.

By means of experiments with various household objects the robustness and performance are validated. Further on, we compare our approach with a pose computation using a physics simulation framework.

I. Introduction

A flexible and skilled Personal Robot needs to perform various pick-and-place tasks in different domains like e.g. cleaning up worktops in households, laboratories, or workplaces in general. Therefore, a personal robot should be able to elaborate how and where to place objects.

However, there is a wide range of situations, objects, and environments that affect this challenging task. Hence, the robot must be able to handle unknown objects and environments whose surface models are reconstructed using sensor data. The resulting models are potentially incomplete and noisy. In addition, the planning should be done in “no time”. Meaning, the planning time should be less than the time the robot needs to do its required motions.

When it comes for the robot to place an object in the environment the robot has to be able to plan where to put down the object. An algorithm calculating stable placement poses helps the robot to perform successfully.

The input to our algorithm is a placement area and an object, both given as surface models with convex planar surface patches. To reconstruct these surface models, the sensor data from a depth camera is integrated over time. These object models include holes of non considered regions. The integration over time of the noisy sensor data dramatically reduces the noise in the resulting object models.

The output of our algorithm is a pose (position and orientation) for the object and several intermediate poses that describe constructive steps to reach a stable goal pose. The main steps of object placement planning are:

1.) **Select a location on the placement area** suitable for the object.
2.) **Orient the object** (Fig. 1(a)) with respect to its geometrical properties or semantical needs, e.g. an upright orientation [1].

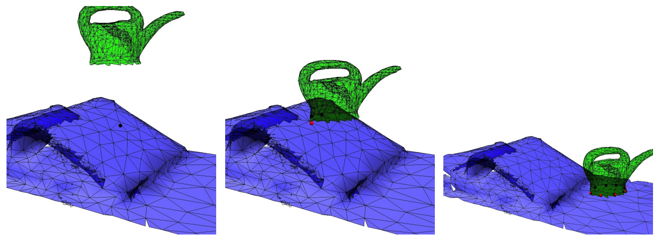

(a) Object pose after step 2.) (b) Intermediate pose during step 3.) (c) Goal pose after step 4.)

Fig. 1: Illustration of the general steps 2-4 of our algorithm.

3.) **Compute the placement pose** (Fig. 1(b)) by rotating or translating the object until it remains in a probably stable pose.
4.) **Compute the quality metric** (Fig. 1(c)) of the reached pose including an evaluation of the stability. In case of instability we apply a **local optimisation** step using step 3.

The contribution of this paper is to offer a solution considering all of the above steps for planning object placement posess in the environment while using only the geometric information gained by a depth sensor. With a special focus is put on the question of how to compute candidate poses and how to optimise them locally in case of static instability. Additionally, we consider the need of a short planning time by a GPU-based implementation.

The remainder of this paper is organized as follows: At first we review related works (Section II). Then we present in detail the steps of the proposed algorithm, starting with a short nomenclature (Section III). Next, the strategies for the selection of the suitable floor points in the placement area are introduced (Section IV). Further on, the procedures to compute appropriate candidate goal poses is explained (Section V). We continue presenting how stability and quality are evaluated in order to select a feasible goal pose and we introduce our local optimisation step (Section VI). Finally, we introduce our object modelling pipeline and the experimental results (Section VII).

II. Related Work

The field of object placement planning in the context of pick-and-place applications [2] was not as such much focused on as was the grasp planning [3]–[5] during the past years. First, we mention approaches dealing with subproblems and related approaches. Second, we review approaches considering our own problem description.

The focus of several works is on the two first steps of the placement problem. Especially, Schuster et al. [6] offers an approach to find free space on planar surfaces. Fu et al. [1] regard finding the upright orientation of man-made objects based on geometric features. And Glover [7] estimates poses of objects based on incomplete point clouds using an object database. Since these works do not consider finding a stable pose of the object on the placement area, they are not regarded any further.

Arranging objects efficiently refers to the packing problem [8]–[10] considering optimal object arrangements in a limited space even for irregular shaped objects [11], [12]. These approaches focus on generic packing problems with known objects. That is why they are complementary to our approach. These ideas could be used in future to develop local optimisation criteria for the fourth step of the algorithm overview.

In the field of robotic applications, placing is about controlling the robot arm and placing the object gently using e.g. force control and passive compliance [13], or task-level planning [14]. However, the focus is here not on finding feasible placing poses. Those approaches are subsequent operations after the placement planning.

Another related topic is push planning of objects on planar placement areas [15]. But again, the focus is not on computing placement poses. However, the results could be used in an subsequent improvement step.

An approach that targets placement planning in the sense of packing is [16]. Since this approach uses only base areas of the unknown objects and known planar placement areas, it is not applicable for general placement areas, which we are interested in. Another approach considering known objects with an unknown placement area is introduced in [17]. Here, object placement considers the footprint of the contact area. The target location of the object is given as input by a human. Consequently, they introduce an interactive approach and not an autonomous one. Both approaches only consider planar placement areas. Contrary to that, we generalise from planar to complex placement areas.

One of the first overall solutions for placing novel objects in complex placement areas is introduced by Jiang et al. [18] and [19]. They outline a learning based framework that requires an object database together with semantic labels. Our work is different from that in so far as we use only geometric information about the object and do not use any kind of learning step.

III. Nomenclature

We assume that the setting does not change during the planning phase. Here and subsequently, we take that the direction of the gravity $d_{\text{g}} \in \mathbb{R}^3$ and an approach direction $d_{\text{vis}} \in \mathbb{R}^3$ as given. Surface models and their properties are defined as follows:

Definition Let $\mathcal{S} := \left\{v_1, ..., v_k \in \mathbb{R}^3\right\}$ be a *planar convex polygonal surface patch* with v_i as vertices ordered counter clockwise, (v_i, v_{i+1}) an edge of this surface, and $|\mathcal{S}| > 2$.

Definition Let $\mathcal{M} := \{\mathcal{S}_1, ..., \mathcal{S}_n\}$ be a *polygonal surface mesh* with convex polygonal surface patches $\mathcal{S}_i$.

Definition Let $k_1^{\mathcal{M}}, k_2^{\mathcal{M}}, k_3^{\mathcal{M}} \in \mathbb{R}^3$ be the *principal axis* of a surface mesh $\mathcal{M}$, with the corresponding eigenvalues $\lambda_1^{\mathcal{M}} > \lambda_2^{\mathcal{M}} > \lambda_3^{\mathcal{M}}$.

Henceforth, we represent the gripped object $\mathcal{M}_{\text{O}}$ and the placement area $\mathcal{M}_{\text{P}}$ as polygonal surface meshes. Further on, we compute the center of mass $c_{\text{O}}^{\mathcal{M}}$ of the object using the vertices of the surface mesh assuming that all vertices have a uniform distribution over the object surface.

IV. Pre-placement Configuration

This section considers the orientation of the object in advance to the placement planning and the selection of feasible candidate locations on the placement area, called *floor points*. The whole passage refers to step 1 and 2 of the algorithm overview in Section I.

A. Object Orientation

Currently we apply a heuristic to achieve the desired object orientation. We use the dedicated principal axis $k_1^{\mathcal{M}_{\text{O}}}$ of the most dominant eigenvalue. We rotate the object such that its first dominant principal axis $k_1^{\mathcal{M}_{\text{O}}}$ is parallel to the approach direction d_{vis}.

By this procedure we are able e.g. to reach the bottom of a box filled with pens or to place a plate in a disk rack. In case of planar placement areas, a pose with $k_1^{\mathcal{M}_{\text{O}}}$ parallel to this plane is still feasible.

If the upright orientation of the object can be estimated with the help of the geometrical observations introduced by Fu et al. [1], we are able to get an idea of the semantically correct orientation and align the object accordingly.

B. Estimate Floor Points

Our algorithm requires a desired location for an object on the placement area, called *floor point*. We select those floor points according to two observations that we had made in the context of household objects. The first observation is that humans tend to place objects at visible locations. The second one is that humans tend to place objects at the lowest position possible, e.g. pens in pencil cups, flowers in vases or dishes in a dish rack.

According to the first observation, we use a ray casting approach featuring rays parallel to the approach direction d_{vis}. The intersection points between rays and placement area are the visible points. In case of a box-like placement area, we select the n lowest points (with respect to d_{g}) from the ones visible, in order to render it more likely that the object reaches the ground of the placement area. The remaining set of vertices is our floor point set.

V. Stable Pose

Based on the computed floor points, we calculate placement poses that are likely to be stable. This section refers to step 3 of the algorithm overview.

First, we explain the procedure to establish a first contact between the object and the placement area (Section V-A).

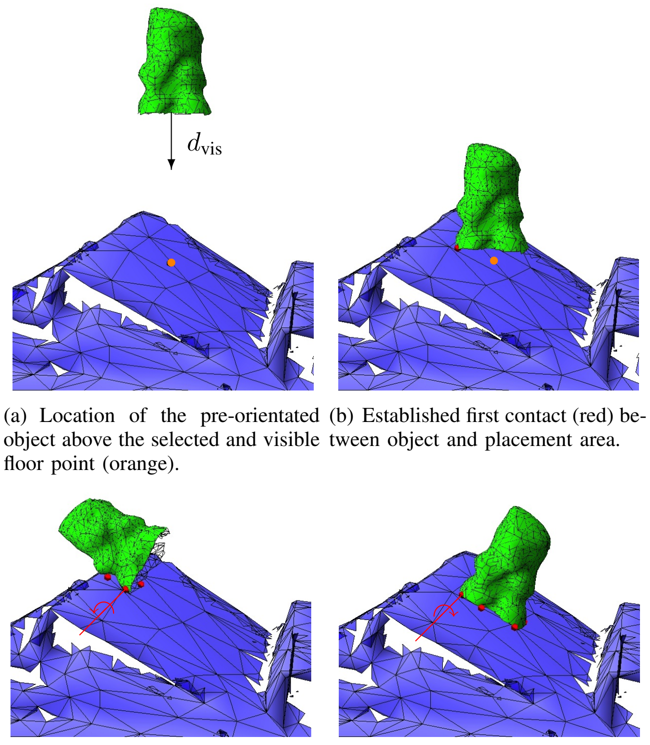
Second, we elaborate on the rotation computation run to establish more contact points (Section V-C). Those computations affect transformation matrices, which are applied to the object's center of mass and the principal axis. The resulting position of the center of mass is the position part $p^{\mathcal{C}}$ of the goal pose $\mathcal{C}$ for the object. The accumulated rotations define the rotatory part $o^{\mathcal{C}}$. Every single transformation matrix itself can be written as intermediate pose, too. Conveniently, those poses can directly be used for a motion planner to approach the goal pose.

A. *First Object Contact*

Fig. 2(a) and Fig. 2(b) show the two steps necessary to establish an initial contact between the object and the placement area.

In the first step, the object is translated such that its center of mass $c_{\mathrm{O}}^{\mathcal{M}}$ and one of the previously computed floor points define a line parallel to the direction d_{vis} and in a distance to the placement area so that no rotation of the object would cause a collision with it.

In the second step, we compute the minimal translational distance along d_{vis} needed for the object to establish contact with the placement area using the GPU-based kd-restart algorithm [20]. We build the kd-tree using the mean value as position of the separating plane of the surface centers. During a leaf refinement we select the separating plane that has the minimum intersections with the surface model.



(a) Location of the pre-orientated object above the selected and visible floor point (orange).

(b) Established first contact (red) between object and placement area.

(c) First result of the rotation computation (left-handed) based on the contact points from (b) around the red axis.

(d) Second result of the rotation computation (right-handed) based on the contact points from (b) around the red axis.

Fig. 2: Examples for the four main steps of the pose computation for the object (green) relative to the placement area (blue). Additionally shown are contact points and rotation axis (red), and current floor point (orange).

B. *Contact Point Computation*

The collision point computation of sensor modelled objects includes some challenges. Since those models may some holes, a decision regarding a penetration is not possible. Furthermore, on boundaries of holes and at sharp edges between two surface patches the computation of the contact normal is not reliable. The collision point computation based on surface models has another computational drawback. If there is just a very small gap between two surface patches, no collision is detected. And as a result some collision points might be missing.

To overcome these issues, we use a surface sphere model and a 2D uniform grid that is perpendicular to the gravity direction d_{g}. Without loss of generality we can assume that our grid is axis-aligned with the x/y-axis of the coordinate system.

We compute the sphere models for the object and the placement area by sampling every surface patch. The centers of the spheres are equally spaced over a surface patch. The distance between two sphere centers is twice the radius of the sphere. The size and position of the 2D grid are the x/y-components of the axis-aligned bounding box of the placement area $\mathcal{M}_{\mathrm{P}}$. The grid is subdivided in equally sized cells. Every sphere $s_{\mathcal{M}_{\mathrm{P}}}$ of $\mathcal{M}_{\mathrm{P}}$ is assigned to multiple grid cells that intersect with a virtual sphere using the center of $s_{\mathcal{M}_{\mathrm{P}}}$ and the sum of the sphere radii of the placement area and object model, called *collision distance*, as radius of this virtual sphere. Additionally, every grid cell stores the maximum z-coordinate plus the collision distance of all assigned spheres.

For every contact point we need to compute its position p_c and the contact normal d_c for the upcoming steps. Therefore we compute for every sphere $s_{\mathcal{M}_{\mathrm{O}}}$ of the object model the gird cells using the x/y-coordinates of the sphere center. If the center of $s_{\mathcal{M}_{\mathrm{O}}}$ has a lager z-coordinate minus the as the maximum height of a relevant grid cell, there can not be a collision. Otherwise, we compute the arithmetic average of all spheres of the relevant grid cells that collide with $s_{\mathcal{M}_{\mathrm{O}}}$ and use it as p_c. Thus the contact normal is: $d_c := s_{\mathcal{M}_{\mathrm{O}}} - p_c$. Additionally, we reorientate the normal that it points towards the object.

C. *Object Rotation Computation*

To establish possible stable contact poses, we rotate the object (Fig. 2(c) and 2(d)). Therefore, we need to compute a rotation axis based on the current contact situation (Section V-C.1). We also compute the rotation angle for this axis producing new contact points (Section V). We continue this rotating procedure until at least three independent contact points between object and placement area are established, or a maximum amount of rotations was performed in a row without success. Since we compute two rotation angles (clockwise - Fig. 2(d) and counter clockwise - Fig. 2(c)) for the object during each rotation step, the recursive applying of rotation steps results in a binary tree. It is root the first contact of the object with the placement area. We branch it at a clockwise and anti-clockwise rotation, evaluate the stability and continue branching if the pose is still unstable or end up as leaf node if the maximum amount of rotations were applied or the pose is considered as stable. Translations do not lead to a branch of the tree.

1) Tilt Axis: Based on the contact situation represented by the contact points computed in Section V-C.2.a, we compute in the following the axis for the next rotation.

If only one contact point between object and placement area exists, we distinguish between three situations. If the main principal axis of the object $k_1^{\mathcal{M}_O}$ is not parallel to the gravity direction d_g, we take $k_1^{\mathcal{M}_O} \times d_g$ as axis. Otherwise, we take the vector connecting the contact point and the center of mass of the object instead of the main principal axis of the object. If this solution is not suitable because the computed vector is also parallel to d_g, we chose an axis randomly. The reason for this rotation axis computation is that we try to rotate the object into its natural direction of dip, which is indicated by its main principal axis or its center of mass.

In case of two or more contact points we distinguish between two situations. If all contact points are located on a common line, we take this line as rotation axis. Otherwise, there is no further need for a rotation axis because the contact points are independent from each other. We then move on to the stability analysis and local optimisation (Step 4 of the algorithm overview).

2) Tilt Angle: For the computation of the rotation angle we merely need to consider vertex/surface contacts and edge/edge contacts, since we have convex surface patches.

Further on, we select the relevant surface patches using the same kd-tree as in Section V-A in order to attain a more efficient computation. These patches serve us to compute the rotation angle based on the following calculations.

a) Vertex/Polygon Rotational Contact: Given are a vertex $v \in \mathcal{M}_O$, a surface patch $\mathcal{S} \in \mathcal{M}_P$, and a rotation axis $A := s_A + \lambda_A d_A$. The point s_A is a collision point and d_A is the axis computed in Section V-C.1. Now we seek the rotation angle around A and between v and $\mathcal{S}$.

Therefore, we introduce the plane $P(\mathcal{S})$ defined by the surface patch $\mathcal{S}$. Further on, we define the auxiliary circle $D(v, A)$ by the vertex v and axis A:

$$\begin{aligned} D(v, A) := \Big\{ p \in \big\{ t \in \mathbb{R}^3 | (t - p_A) \cdot d_A = 0 \big\} | \\ \exists p_A \in A : \min d(p_A, v), \\ d(p, A) = d(p_A, v) \Big\} \end{aligned}$$

We intersect the plane and the circle: $\{p_1, p_2\} = P(\mathcal{S}) \cap D(v, A)$. Subsequently we examine both intersection points p_1 and p_2 to check whether they are located inside of the surface patch $\mathcal{S}$. Finally, we compute the angles between the intersection points inside the surface patch and the vertex v according to the rotation axis A. In case that there are two intersection points we choose the smaller angle.

We use the idea of the kd-restart algorithm in order to trace $D(v, A)$ through the kd-tree. We trace $D(v, A)$ twice: clockwise for the positive rotation angle and counter-clockwise for the negative one. First, we determine the tree

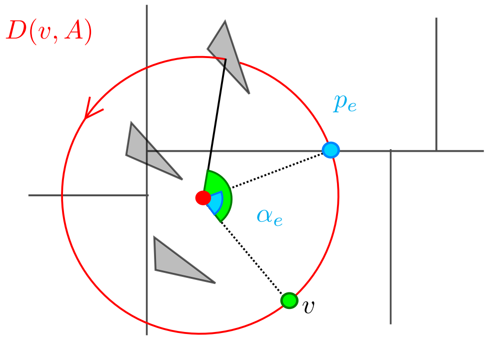


Fig. 3: 2D illustration of the kd-tree tracing using $D(v, A)$ in the counter clockwise direction. The exit angle α_e corresponds to the intersection points p_e (blue) between the kd-tree and $D(v, A)$ (red). The green angle is the rotation angle for the vertex v, we seek for.

leaf that contains the vertex v. Second, we compute the exit angle α_e, where $D(v, A)$ intersects the leaf's bounding box first, called p_e. We enforce, that p_e is located in the adjacent leaf along $D(v, A)$. Third, we find the smallest vertex-surface rotation angle that is smaller than α_e. If no valid angle is found, we continue with the first step using p_e instead of v. Fig. 3 shows a 2D example.

b) Edge/Edge Rotational Contact: Given are two given skew line segments $L_1 := s_{L_1} + \lambda_1(e_{L_1} - s_{L_1}) \in \mathcal{M}_O$, $L_2 := s_{L_2} + \lambda_2(e_{L_2} - s_{L_2}) \in \mathcal{M}_P$ and an axis $A := s_A + \lambda_A d_A$, with start points $s_{L_i} \in \mathbb{R}^3$, end points $e_{L_i} \in \mathbb{R}^3$, direction $d_A \in \mathbb{R}^3$ and $\lambda_i \in [0; 1]$. The angle α rotates L_1 around the axis A. The task is to compute α so that it brings L_1 in contact with L_2.

The computation of this rotation includes basically three steps: (1) First, we test if the claimed rotation angle does exist. Second, we compute the angle with the use of a surface of revolution constructed by the rotation of L_1 around the axis A. (2) For this task we transform the line segments and

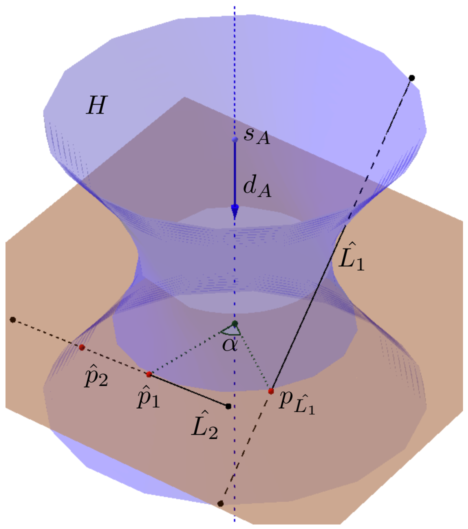


Fig. 4: The steps for the rotation angle computation between two skew line segments are: Construct a surface of revolution H using the line segment $\hat{L_1}$ as generatix and $A := s_A + \lambda_A d_A$ as axis. Compute the intersection points $\{\hat{p}_1, \hat{p}_2\} = H \cap \hat{L}_2$. Compute the intersection point $p_{\hat{L_1}}$ of the line segment $\hat{L_1}$ with the plane defined by p_1 and d_A used as normal. And compute the rotation angle α.

the axis to a more suitable coordinate system, which allows us to calculate the coefficients of the surface of revolution in an easier way. (3) At last, we compute these coefficients, and by the help of the intersection between the other line segment L_2 and the constructed surface we compute the requested rotation angle. Fig. 4 illustrates the whole process graphically. Here are the steps once more in detail:

(1) Pretest: Check if the distance of both end points of L_1 and L_2 alternates according to their distance to the rotation axis: Without loss of generality, let L_1 be closer to A than L_2. If the condition

$$\min_{p \in \{s_{L_2}, e_{L_2}\}} d(p, A) > \max_{p \in \{s_{L_1}, e_{L_1}\}} d(p, A)$$

holds, there will be no collision for the described rotation. Further on, the projected endpoints of L_1 and L_2 on A have to alternate along A such that there is no separating plane $(x - t) \cdot d_A = 0$ with $t \in A$ as an arbitrary point that separates both line segments.

(2) Transformation: For a more simple construction we transform L_1 and L_2 with the result that A is identical to the z-axis d_z. Furthermore, if there exists a single distinct point $t_0 := \text{argmin}_{t \in A}\{d(t, L_1)\}$ on the rotation axis A that has the smallest distance to the rotating line, we use t_0 to define an affine transformation that transforms t_0 into the origin and aligns the direction d_A with the z-axis d_z. If the point t_0 is not distinct, i.e. $L_1 || A$, we can take any point on A for t_0 without limitation.

Henceforth let $\hat{L_1}$ and $\hat{L_2}$ be the transformed line segments.

(3) Geometric construction: We want to compute a surface of revolution H describing the rotating segment which will later be intersected with the second line segment. We use the line segment $\hat{L_1}$ as the generatrix of the surface.

If $s_{\hat{L_1}}$ and $e_{\hat{L_1}}$ are in a plane coplanar to the x-y-plane the resulting primitive is an annulus and can therefore be written as:

$$H : (x - s_{\hat{L_1}}) \cdot d_z = 0$$

Otherwise, the surface of revolution can either be a cylinder, cone or hyperboloid of one sheet. Those surfaces are represented by a generalized quadric of the form:

$$H : \frac{x^2 + y^2}{a^2} - k \cdot \frac{z^2}{b^2} = c.$$

Hence, with the points $p_1 := \text{argmin}_{p \in \hat{L_1}}\{d(p, d_z)\}$ and $p_2 \in \hat{L_1}, p_2 \neq p_1$ and $d(p_2, d_z) \neq 0$ the quadric can be parametrized as shown in Table I.

(4) Angle computation: Since the surface of revolution is now well defined, we are able to compute the intersection points $\{\hat{p}_1, \hat{p}_2\} = H \cap \hat{L_2}$ by inserting the line segment $\hat{L_2}$ into H. We expect $\hat{L_2}$ not to lie on H. Henceforth, we outline the computation steps using $\hat{p}_1$. For each intersection point $\hat{p}_1$ and $\hat{p}_2$ we define the plane $P : (x - \hat{p}_1) \cdot d_z$. The resulting rotation angle between the two line segments is now defined by $\sphericalangle_{d_z}(\hat{p}_1, P \cap \hat{L_1})$ (angle α in Fig. 4).

Note that in case of two intersection points, the smallest rotation angle represents the correct one. In addition, there is no need for an inverse transformation of the constructed surface of revolution H back into the original space because we only applied an affine transformation, which does not affect the value of the rotation angle.

TABLE I: Parametrization of different surfaces of revolution using the general quadric using $p_1 := \text{argmin}_{p \in \hat{L_1}}\{d(p, d_z)\}$ and $p_2 \in \hat{L_1}, p_2 \neq p_1$.

$\frac{x^2+y^2}{a^2} - k \cdot \frac{z^2}{b^2} = c$	k	c	a^2	b^2
cylinder	0	1	$d(p_2, d_z)^2$	-
cone	1	0	$d(p_2, d_z)^2$	$p_{2,z}^2$
hyperboloid	1	1	$d(p_1, d_z)^2$	$\dfrac{p_{2,z}^2}{\dfrac{d(p_2, d_z)^2}{d(p_1, d_z)^2} - 1}$

VI. Configuration Selection

The previous steps computed a pose for the object, were a contact situation with a minimum of three contact points is ensured. Now we apply different quality measurements, and in some cases we adjust the computed pose (Step 4 of the algorithm overview).

The mandatory condition is that the object remains in static equilibrium. Therefore, we introduce in the following a quality metric and a local optimisation step both based on the planar case.

If all contact points are located on a common plane P, we can compute the stability based on the planar case. We compute the 2D convex hull Ω of the contact points located on P. Then we determine the projection $\hat{c}_\text{O}$ of the object's center of mass along the gravity direction d_g onto P and check whether it is inside Ω. Additionally, if the angle between the normal of P and d_g is smaller than the angle defined by the friction coefficient μ, then the object pose remains in static equilibrium.

In case the friction condition is not met and the plane defined by the collision points is not perpendicular to the gravitation, we compute the failure direction d_f parallel to P. Let $\hat{c}_\Omega$ be the center of Ω then $d_\text{f} = \hat{c}_\text{O} - \hat{c}_\Omega$. The new translation direction d_vis is d_f, which causes the object to slide down the plane P. Using the new d_vis, we re-conduct the translation of Section V-A as well as the contact computation of Section V.

In case the friction condition is not met and a slide down the plane P is not possible, we just rotate the object again into the failure direction with the rotation axis defined by $d_A = d_\text{f} \times d_\text{g}$ and s_A as the farthest contact point of Ω with respect to d_A in the direction of d_f.

Our local optimisation steps generate contact points in such a way that they stabilise the object with respect to its old failure direction. Those optimised contact situations can usually no longer be handled with the planar case. Whether those poses are in static equilibrium, is currently checked using a heuristic. It is inspired by the sliding computation used in computer games. Common approaches, like the Newton-Euler equations, the static analysis, or the finite element analysis, are not suitable for our purpose. All these approaches require the force value for every contact point, which is not available in our situation.

The heuristic assumes, that there are contact points beneath the center of mass of the object and their contact normal d_c is within the friction cone with a friction coefficient μ with $\measuredangle(d_c, -d_g) < \tan(\mu)$. We search the largest planar convex hull Ω' using these contact points and apply the computation steps of the planar case. The next step is in case of instability, to search for contact points with a contact normal pointing reverse to the resulting failure direction d'_f with $\measuredangle(d_c, -d'_\text{f}) < \tan(\mu)$. Now we apply the computation step of the planar case once again. The only modification is, that we replace the direction of gravity by d'_f and compute the 2D convex hull Ω'' with the largest area from the selected contact points. If the center of mass of the object projects into Ω'' and the angle between d'_f and Ω'' or between the x/y-plane and Ω'' is smaller than the friction angle, we consider the current pose as stable. If just the computed angle does not meet the friction requirements we use $-d'_\text{f}$ as failure direction for the optimization step. Otherwise, we use the resulting failure direction d''_f to continue the computation as outlined for the planar case. Please note that we use the same quality rating for very applied planar case and apply the arithmetic mean.

Currently this heuristic will not consider a hanging object, for example an umbrella hanging on a hall stand, as stable, since there will not be any collision points beneath the center of mass of the object.

For stable poses regarding the planar case or our heuristic, our quality metric is:

$$Q = 1 - \frac{d(\hat{c}_\text{O}, \hat{c}_\Omega)}{\hat{l}_\Omega} - \frac{d(c_\text{O}, \Omega)}{A(\Omega)}$$

with $\hat{l}_\Omega$ as distance between $\hat{c}_\Omega$ and the boundary of Ω along the failure direction and the area of the convex hull $A(\Omega)$. The best quality value is one. This is the case, if and only if the two quotients are zero. The smaller a quality value, the worse is the computed object pose.

The first quotient is a measure for the tilt turn. The closer the projected center of mass $\hat{c}_\text{O}$ is to the boundary of the convex hull Ω the greater is the distance $d(\hat{c}_\text{O}, \hat{c}_\Omega)$. A pose is more stable if the distance $d(\hat{c}_\text{O}, \hat{c}_\Omega)$ decreases and the distance $\hat{l}_\Omega$ increases. The second quotient is the ratio between footprint size and height of the object. The smaller this ratio, the larger is the footprint area at a constant height, or the smaller is the hight of the object at a constant footprint area. Both possibilities imply that the object pose is more stable at a smaller quotient-value $\frac{d(c_\text{O},\Omega)}{A(\Omega)}$.

VII. Experiments

For the experimental evaluation we initially chose $d_\text{vis} = d_\text{g}$ as negative z-axis. All computations were performed on an AMD Opteron CPU and a Nvidia GTX Titan GPU. Furthermore, we use a friction coefficient of $\mu = 0.2$. First, we describe our object modelling pipeline (Section VII-A). Afterwards, we present planning results of various object and placement areas and discuss them (Section VII-B). At last we evaluate the computation time, and compare our approach to a physics simulation framework (Section VII-C).

A. Object Model

We now outline our object modelling pipeline used for the experiments. As sensor we use a hand held depth camera. The relative registration between two camera images is accomplished by using the *iterative closest point* algorithm. Since we know the initial position of the camera we are able to register the reconstructed object model to the robot's coordinate system.

Each point cloud of a depth image is integrated into a regular grid using a *signed distance function*. We convert this grid-based model into a triangulated surface model using the *marching cube* algorithm. As a last step of our pipeline, we simplify these surface models to a certain amount of triangles (usually 1000) using the Quadric Error edge-collapse simplification* with an quality value of 0.3.

Placement area and object are modelled independently from each other.

B. Results and Evaluation

Fig. 5 and Fig. 1 show results of our placement planning for various objects and non planar placement areas. For every pair the computed pose is shown on the right and the real corresponding manual established pose. All poses are computed with respect to equally spaced floor points with a distance of 0.09m between two neighbours. Except of Fig. 5(c) the figure shows in each situation the best rated pose. The local optimisation enables the planner to compute generally stable poses. For planar and horizontal placement areas, the initial pose computation already results in stable pose.

To get an idea of the quality value of a pose, we compare the poses of Fig. 5(a) ($Q = 0.47$) and Fig. 5(b) ($Q = 0.40$). Both poses have a comparable quotient between their area of the contact point convex hull and the distance of the object's center of mass to this hull. However, the object in Fig. 5(b) is much more tilted than the other one, which causes differences in the quality measure, because the projected center of mass is closer to the boundary of the bottom point's convex hull. Our experiments show that object poses with a quality value less than 0.3 should be considered as unstable.

Fig 5(f) shows a watering can placed on stacked ring binders. Since a empty watering can is really light and the surface of the ring binders is quite smooth, the shown pose is not a stable as the quality measure lets us believe. Thus there is a slight difference between the planning result and the real placement. In case of a filled watering can the computed pose is much more stable. Since we use only geometric information and assume a friction value, the planning result and especially the quality rating are almost too good for objects that differ much from the assumed friction.

Especially Fig. 5(a),(b),(d), and (e) underline that our approach computes stable poses for complex placement areas.

*VCG Library: www.vcg.isti.cnr.it

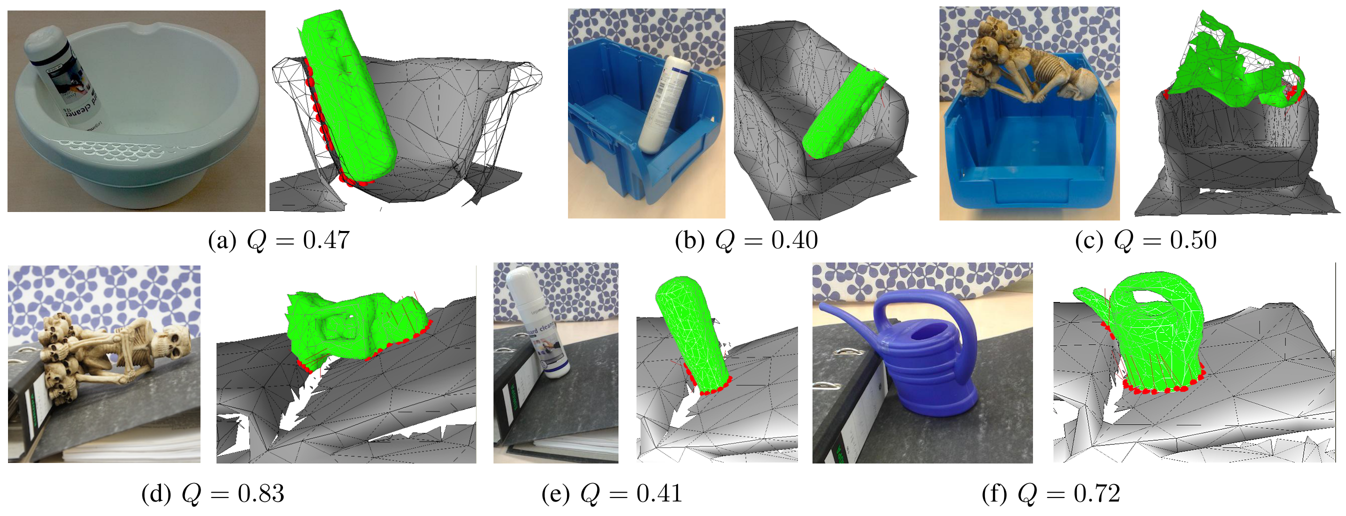


Fig. 5: Selected experimental results for four different placement areas (grey) and three objects (green) including the computed contact points (red). The value Q is the quality rating.

However, our planner is currently not able to hang objects onto the placement area. Furthermore, our optimization step could be improved, in order that slightly stable poses like in Fig. 5(c) and Fig. 5(f) are not considered as stable and are further improved, instead.

C. Runtime Analysis

Our approach has two major parameters with high impact to the computation time: On the one hand the number of surface patches used for the models of placement area and object and on the other hand the number of initial floor points. Note, for every floor point a previously mentioned binary tree of object poses is built. Since we start with uniform distributed initial floor points with a fixed distance of 0.09 m, the size of the placement area correlates with the amount of these points. During experiments, we recognized that a maximum number of five rotations is a suitable amount to achieve valid stable poses.

We implement four kernels using CUDA (distance, vertex-surface-rotation, edge-edge-rotation, and collision-point computation). We concurrently process all intermediate poses assigned to one kernel, e.g. all available collision point computations are performed at once over all binary trees. A kernel is selected for processing by the amount of available input intermediate poses. Furthermore, we launch depending on the kernel one thread for every vertex, edge, and sphere of the object model.

For the runtime measurement we use the ring binder placement area (Fig 5(d)-(f)), since it is the one that leads to the most initial floor points (140). Furthermore, we vary the amount of triangles of the object model from 200 to 1000.

Fig. 6 plots the worst case kernel computation time among the test objects gained with the skull candle (Fig 5(c),(d)) over the number of triangles of the object model.

The overall computation per kernel time grows linear with the number of triangles; only the computation time for the collision point computation is approximately constant. The reason is that the amount of collision spheres is independent from the amount of triangles, since the area of the surface

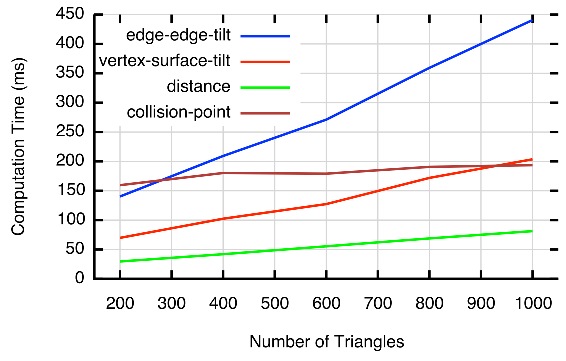


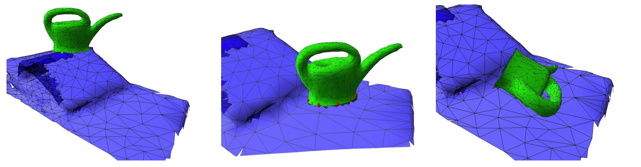
Fig. 6: Plot of computation times over number of triangles of the object model for the four GPU kernels.

is approximately constant. The increase of time for the other kernels correlates with their computational complexity. Especially the slope of the edge-edge-rotation computation time has to be mentioned. The amount of rotation computations grows more compared to the other rotation computation kernel, since the selection of relevant edges from the kd-tree is not as efficient as for the vertex-surface-rotation computation.

Our over all runtime including the sequential steps on the CPU is about 1.7 sec for 140 initial floor points dependent on the involved modells. Good stable poses are available after about 500 ms. We observed that after about 50% of the computation only a little amount of stable poses will be computed. Hence it is sufficient to terminate the planning after $500 - 900$ ms depending on the performance of the GPU.

Compared to the computation time for a physics simulation framework[*], our algorithm has a comparable runtime. The runtime of this simulation is about 200-700 ms until the object remains in equilibrium, which is mostly caused by the number of collision tests. The time needed to compute a new position and orientation in each time step is negligible. But

[*]Bullet Physics Library: www.bulletphysics.org



(a) Start pose (b) Our approach (c) Rigid body simulation

Fig. 7: Beginning with a common starting pose in (a), our placement planner outputs (b) and a rigid body simulation outputs (c).

physics simulation frameworks have one major drawback. Since the position and orientation change is affected by applying impulses or forces, it is difficult to encroach the computation as we do during the local optimisation step in Section VI. And since a robot will perform the placement, a physical correct placement process is not required, only a stable goal pose is necessary. Fig. 7 shows for example the resulting stable poses using our approach (Fig. 7(b)) and the rigid body simulation (Fig. 7(c)) with the same start pose (Fig. 7(a)). Furthermore, the reconstructed object models include holes and are especially not watertight. This fact causes unstable contact force results and, as a consequence, the computations of the rigid body simulation become unstable.

VIII. Conclusion

We introduced an improved version of our educated sample based placement planner for unknown sensor-modelled objects and placement areas [21]. It uses only geometrical information in its computations. Therewith, we introduced a novel placement pose calculation that offers the opportunity to influence the pose computation easily. We use this opportunity for a local optimisation of unstable poses, by recalculating the movement direction or rotation axis of the object. Not only planar placement areas are considered, but also for complex areas the planner performed well.

The experiments point out that the planner is able to cope with small holes in the surface of object models. However, if some parts of the object model are not reconstructed at all, the planner will not be able to overcome this issue. However, compared to a physics simulation framework our approach is more robust considering rough and noisy surfaces and holes. Further on, we indicate that different objects could be placed stable onto complex and planar placement areas. Runtime measurements show that the computation time only increases slightly for object models with more surface patches. Moreover, the performance of our GPU-based implementation matches up with one of the fastest physic-simulation frameworks.

Future work includes targeting on the conservativeness of the planner for objects with huge holes or even complete unmodelled regions of the object. Moreover, we will work on the issue that some poses have high quality value but a small stability in the real world especial for objects that differ a lot from our assumed friction value and object weight. Furthermore, we will use the bases components of this planner for a grasp planning approach and introduce a combination of both planners [22].

References

[1] H. Fu, D. Cohen-Or, G. Dror, and A. Sheffer, "Upright orientation of man-made objects," *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3, p. 42, 2008.

[2] T. Lozano-Pérez, J. L. Jones, E. Mazer, and P. A. O'Donnell, "Task-level planning of pick-and-place robot motions," *Computer*, vol. 22, no. 3, pp. 21–29, 1989.

[3] C. Papazov, S. Haddadin, S. Parusel, K. Krieger, and D. Burschka, "Rigid 3D geometry matching for grasping of known objects in cluttered scenes," *The International Journal of Robotics Research*, vol. 31, no. 4, pp. 538–553, Mar. 2012.

[4] A. Sahbani, S. El-Khoury, and P. Bidaud, "An overview of 3D object grasp synthesis algorithms," *Robotics and Autonomous Systems*, vol. 60, no. 3, pp. 326–336, Mar. 2012.

[5] M. a. Roa, M. J. Argus, D. Leidner, C. Borst, and G. Hirzinger, "Power grasp planning for anthropomorphic robot hands," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, May 2012, pp. 563–569.

[6] M. J. Schuster, J. Okerman, H. Nguyen, J. M. Rehg, and C. C. Kemp, "Perceiving clutter and surfaces for object placement in indoor environments," *2010 10th IEEE-RAS International Conference on Humanoid Robots*, pp. 152–159, Dec. 2010.

[7] J. Glover, G. Bradski, and R. B. Rusu, "Monte Carlo Pose Estimation with Quaternion Kernels and the Bingham Distribution," *Robotics: Science and Systems VII*, p. 97, 2012.

[8] S. Martello, D. Pisinger, and D. Vigo, "The three-dimensional bin packing problem," *Operations Research*, vol. 48, no. 2, pp. 256–267, 2000.

[9] F. Parreño, R. Alvarez-Valdes, J. F. Oliveira, and J. M. Tamarit, "A hybrid GRASP/VND algorithm for two- and three-dimensional bin packing," *Annals of Operations Research*, vol. 179, no. 1, pp. 203–220, Oct. 2008.

[10] E. E. Bischoff and M. S. W. Ratcliff, "Issues in the development of approaches to container loading," *Omega*, vol. 23, no. 4, pp. 377–390, 1995.

[11] W. Han, J. A. Bennell, X. Zhao, and X. Song, "Construction heuristics for two-dimensional irregular shape bin packing with guillotine constraints," *European Journal of Operational Research*, pp. 1–18, Apr. 2013.

[12] A. Pasha, "Geometric Bin Packing Alogrithm for Arbitrary Shapes," Ph.D. dissertation, University of Florida, 2003.

[13] A. Edsinger and C. Kemp, "Manipulation in Human Environments," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*. IEEE, Dec. 2006, pp. 102–109.

[14] T. Lozano-Pérez, J. L. Jones, E. Mazer, and P. A. O'Donnell, "Task-Level Planning of Pick-and-Place Robot Motions," *Computer*, vol. 22, no. 3, pp. 21—-29, 1989.

[15] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Sep. 2011, pp. 4627–4632.

[16] J. Baumgartl and D. Henrich, "Fast Vision-based Grasp and Delivery Planning for unknown Objects," in *7th German Conference on Robotics (ROBOTIK 2012)*, 2012.

[17] K. Harada, T. Tsuji, K. Nagata, N. Yamanobe, H. Onda, T. Yoshimi, and Y. Kawai, "Object placement planner for robotic pick and place tasks," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2012, pp. 980–985.

[18] Y. Jiang, M. Lim, C. Zheng, and A. Saxena, "Learning to Place New Objects in a Scene," *IJRR*, vol. 31, no. 9, Feb. 2012.

[19] Y. Jiang and A. Saxena, "Hallucinating humans for learning robotic placement of objects," in *ISER*, 2012.

[20] T. Foley and J. Sugerman, "Kd-tree acceleration structures for a gpu raytracer," in *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM, 2005, pp. 15–22.

[21] J. Baumgartl, P. Kaminsky, and D. Henrich, "A geometrical placement planner for unknown sensor-modelled objects and placement areas," in *Proceeding of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2013, pp. 364–371.

[22] J. Baumgartl and D. Henrich, "Gpu-based grasp and placement planners for sensor-modelled objects," in *Joint 45th International Symposium on Robotics (ISR'14) and the 8th German Conference on Robotics (ROBOTIK'14)*, 2014.