# Task-driven Perception and Manipulation for Constrained Placement of Unknown Objects (Supplementary material)

Chaitanya Mitash, Rahul Shome, Bowen Wen, Abdeslam Boularias and Kostas Bekris

## I. Algorithmic Outline

The framework proposed for *integrated perception and manipulation planning* is outlined in terms of algorithmic steps. The high-level pipeline is first described in Algorithm 1. The manipulation planning primitive is represented by Algorithm 2. Additional details of the subroutines are specifically addressed.

### A. Integrated Perception and Manipulation Planning

Algorithm 1 goes over the sequence of steps for the high-level pipeline. The inputs to the subroutine are the sensing information derived from a single RGB-D sensor. The object segmentation mask $T_{\text{mask}}$ refers to the segment of the target object in the RGB-D image. $R_{\text{place}}$ is also obtained from the point cloud at the placement location to recover the workspace bounds for the placement region. The algorithm proceeds over these inputs.

***GetPlacement:*** This subroutine generates a target placement pose that is fully contained within $R_{\text{place}}$. Though $P_{\text{target}}$ denotes a single target pose, in the implementation a set of candidate targets can be generated based on how constraining the dimensions of $R_{\text{place}}$ are.

***InitializeShape:*** The shape initialization uses the RGB-D data from a calibrated camera and the segment for the target object to determine the object representation $O$ using considerations for what part of the object is already visible $\mathcal{S}$, and the region that is currently unseen $\mathcal{U}$ but might belong to the object. The initial pose is also attached to the (arbitrarily chosen) local coordinate frame common to the $\mathcal{S}$ and $\mathcal{U}$ points.

***ManipulationPlanning:*** This step represents the open-loop manipulation planning primitive and is further described in Algorithm 2.

In line 3 *ManipulationPlanning* is called with the initial and target poses, and a flag that forces the method to attempt the initial pick in the solution. If a solution is already found then the method can go straight to executing the solution in Line 18. Otherwise, the failure to find the solution must be caused because

- either the problem is not solvable given the current placement region bounds, or
- the object representation $O$ is too conservative.

The authors are with the Computer Science Department of Rutgers University in Piscataway, New Jersey, 08854, USA. Email: {cm1074, rs1123, bw344, ab1544, kb572}@rutgers.edu

In Line 6, another invocation of *ManipulationPlanning*, this time with the *optimistic* estimate $\mathcal{S}$ validates whether the problem is solvable. In the most optimistic scenario all of the currently unseen points might be resolved to not belong to the object. This means that the problem is not solvable if planning fails with this optimistic representation.

***PlanPick:*** This subroutine computes the motion of the picking arm from its current configuration to a grasping configuration for the object at the passed pose. The grasps are generated over the input object representation.

As an implementation detail, the call to *PlanPick* on Line 7 effectively uses the picks that were discovered during the optimistic planning step in Line 6 to narrow down a pick that works for $O$ and solved the problem with $\mathcal{S}$.

Line 8 passes over the computed motions to the robot and executes the pick action, at the end of which the object is grasped by the end-effector.

The loop outlined between Lines 11-17 repeatedly attempts to update the object representation by exposing new viewpoints to the camera, and then trying to solve the manipulation planning problem over the update shape. The loop continues till a plan is found or up to a maximum number of attempts.

***GetNextBestView:*** This function returns a viewpoint $P_{\text{next}}$ to take the object to next, based on what pose would be expected to expose most of the unseen points.

***PlanToPose:*** Given a target pose, this computes the arm motion to take a currently grasped object to that pose. In Line 12, the *PlanToPose* subroutine is called to take the object to the next view point, and then subsequently executed in Line 13. Once the object has reached the next viewpoint, the new sensing information can be used to update its shape.

***UpdateShape:*** The RGB-D data and the outcome of pose tracking are used to update the object representation $O$ by augmenting the seen subset $\mathcal{S}$ and reducing the unseen set $\mathcal{U}$. This operation reduces the conservative nature of $O$. This increases the possibility that this relaxed representation will be sufficient to compute a solution.

Line 15 goes on to pass the updated object representation, and attempts manipulation planning from to the target object pose. The $pickFlag$ is set to false here since the object has already been picked up. If a solution is discovered at this step the motions can be executed.

***ClosedLoopExecute:*** In order to ensure robust execution of the solution trajectory, careful attention must be paid to any errors that might arise from imprecise execution, non-

**Algorithm 1:** IntegratedPerceptionAndManipulationPlanning

**Input:** $I_{\text{rgbd}}$: RGB-D image of the scene
$T_{\text{mask}}$: Object segmentation mask
$R_{\text{place}}$: Target placement region

$\{O, \mathcal{S}, \mathcal{U}, P_{\text{init}}\} \leftarrow \texttt{InitializeShape}(I_{\text{rgbd}}, T_{\text{mask}})$ ▷ the object representation and pose are initialized
$P_{\text{target}} \leftarrow \texttt{GetPlacement}(O, R_{\text{place}})$ ▷ a computed placement pose for the object inside the region
$\Pi \leftarrow \texttt{ManipulationPlanning}(O, P_{\text{init}}, P_{\text{target}}, \mathbb{T})$ ▷ plan manipulation actions over the conservative object estimate $O$
$P'_{\text{target}} \leftarrow \texttt{GetPlacement}(\mathcal{S}, R_{\text{place}})$ ▷ a computed placement pose for the optimistic estimate
▷ if no solution was found, check if the problem is solvable when using the optimistic shape $\mathcal{S}$
**if** $\Pi = \emptyset$ ***and*** $\texttt{ManipulationPlanning}(\mathcal{S}, P_{\text{init}}, P'_{\text{target}}, \mathbb{T}) \neq \emptyset$ **then**
  $\Pi \leftarrow \texttt{PlanPick}(O, P_{\text{init}})$ ▷ compute motion to pick object at the initial pose
  $\texttt{Execute}(\Pi)$ ▷ execute picking action
  $\Pi \leftarrow \emptyset$
  ▷ keep trying to get more views, and solve the problem with updated shape
  **while** $\Pi = \emptyset$ ***or*** $\texttt{MaxAttempts}()$ **do**
    $P_{\text{next}} \leftarrow \texttt{GetNextBestView}()$ ▷ $P_{\text{next}}$ keeps track of the next viewpoint for the object to try
    $\Pi \leftarrow \texttt{PlanToPose}(P_{\text{next}})$ ▷ compute motion to next viewpoint
    $\texttt{Execute}(\Pi)$ ▷ execute motion to next viewpoint
    $O \leftarrow \texttt{UpdateShape}(I_{rgbd})$ ▷ using current sensing data, update the object representation
    $P_{\text{target}} \leftarrow \texttt{GetPlacement}(O, R_{\text{place}})$ ▷ a computed placement pose for the object inside the region
    $\Pi \leftarrow \texttt{ManipulationPlanning}(O, P_{\text{next}}, P_{\text{target}}, \mathbb{F})$ ▷ attempt solution using updated shape, and *pickFlag* false
▷ if a solution was found
**if** $\Pi \neq \emptyset$ **then**
  $\texttt{ClosedLoopExecute}(\Pi, O)$ ▷ execute the solution with online adjustments to correct errors

**Algorithm 2:** ManipulationPlanning

**Input:** $O$: Object representation
$P_{\text{start}}$: Starting pose
$P_{\text{goal}}$: Goal pose
$pickFlag$, $\mathbb{T}/\mathbb{F}$ toggles the pick computation

**Output:** $\Pi$, Solution trajectory

$\Pi \leftarrow \emptyset$ ▷ initialize solution as empty
▷ *pickFlag* toggles whether the pick action is performed
**if** $pickFlag$ **then**
  $\Pi \leftarrow \texttt{PlanPick}(O, P_{\text{start}})$ ▷ compute motion to pick at the start pose, and save to solution
▷ proceed if pick worked or *pickFlag* is true
**if** $\Pi \neq \emptyset$ ***or*** $pickFlag$ **then**
  $\Pi' \leftarrow \texttt{PlanPlace}(O, P_{\text{goal}})$ ▷ compute motion to directly place the object at goal pose
  ▷ if a direct placement failed
  **if** $\Pi' = \emptyset$ **then**
    $\Pi' \leftarrow \texttt{PlanHandoffPlace}(O, P_{\text{goal}})$ ▷ compute motion to handoff and place at goal pose
  ▷ if a motion to reach the object placement was found
  **if** $\Pi' \neq \emptyset$ **then**
    $\Pi \leftarrow \Pi \oplus \Pi'$ ▷ stitch together the solution
  **else**
    $\Pi \leftarrow \emptyset$ ▷ reset solution to be empty
**return** $\Pi$ ▷ return solution

prehensile physical interactions, and sensing noise. To that effect, at specific discrete moments the solution motions are readjusted to account for any discrepancy between where the object was expected to be, and where it is sensed. During this execution, the objective is to reuse the motions generating in the initial solution while compensating for errors, to achieve robust placement of the object at the target pose.

### *B. ManipulationPlanning*

The steps involved in manipulation planning are described in the *ManipulationPlanning* subroutine. This planning step occurs solely on the information provided to it as its input, and does not interact with sensor data. The manipulation actions computed here are also only returned, and it is left to the high-level pipeline to make use of the output from this function.

***Inputs:*** The object representation, starting pose of the object, goal object pose are passed as input arguments. An additional *pickFlag* parameter toggles whether the planning computes the pick over the initial object pose or whether the object has already been grasped, and the initial pick can be skipped.

Line 3 checks the flag to invoke the pick computation through *PlanPick*. If no such pick is possible, then the problem cannot be solved. Line 6 checks this and returns failure in Line 16. If picking succeeded or was skipped, the placement computation can proceed in Lines 7-15.

***PlanPlace:*** This subroutine computes the arm motions for the *picking arm* to directly attempt the placement of the object at the target pose. If a direct placement was not possible as checked in Line 9, handoffs are attempted.

***PlanHandoffPlace:*** Handoffs make use of a second available arm to effectively enhance reachability of the target pose. Handoff are computed over the input object representation, and the subroutine attempts to find motions that can hand off the object to the second arm, and then place it using the second arm. If this works, this can be added to the solution in Line 13.

Line 16 returns the solution that would have been populated successfully if the planning worked.

### *C. Generalization*

The steps followed in the algorithmic outline correspond to the pipeline that has been evaluated in the current work to solve the constrained placement problem. The specific implementation choices for each of the subroutines described here would depend on the specific design choices, like the availability of a second arm etc.

The specific underlying planning algorithms used for the computations in *ManipulationPlanning* can be chosen based on the nature of the planning problem. *PlanHandoffPlace* can also be replaced with other object manipulation primitives, like reorientation and regrasp, if a second arm is not available.

Operations like *GetNextBestView* and motions to these views, can be replaced by more intelligent, and application-specific strategies.

The objective of the current work is to identify the key considerations in designing an algorithmic framework capable of solving the constrained placement problem for unknown objects. The evaluations performed over real experiments demonstrate the efficacy of these design choices.