(Supplementary Material) Intention-Aware Cobots: Revolutionizing Object Packing with Human-Centric Synchronization

I. Transformer Encoder Design

We propose an encoder-only task transformer design that inputs a sequence of trajectories from several subtasks of a complete task and predicts the next subtask trajectory. The block diagram of the transformer in Fig. 8(M2), shows three key modules - the self-attention unit, the Softmax function unit and the feed forward unit. The mathematical model follows. We provide the derivation to construct the key modules. Let $X, Y \in \mathbb{R}^{n\times m}$, $R, S \in \mathbb{R}^{k\times n}$ and $\lambda \in \mathbb{R}$ for representation R, sub-task trajectory S and the complete task trajectory T. The function to derive is,

$$f(X,Y) = \text{softmax}\left(\lambda \cdot X^T \cdot R^T \cdot S \cdot Y\right) \cdot T \cdot X \quad (1)$$

Let $b_k \in \mathbb{R}^m$ denote the basis vectors. Using b_k we access the each columns of a matrix as $c_k = Hb_k$ where $H = \begin{bmatrix} h_1 & h_2 & \cdots & h_m \end{bmatrix}$ and each row as $c_k = H^T b_k$. The matrix expansion is expressed as,

$$H = \sum_{k=1}^{m} (Eb_k)b_k^T = \sum_{j=1}^{m} b_j (b_j^T E) \quad (2)$$

For each row-wise matrix element we take the derivative of the Softmax function as,

$$\begin{aligned} \text{m} = softmax\left(H^T b_j\right) \quad &\implies \quad M_j = \text{Diag}(m_j) \\ dm_j = \left(M_j - m_j m_j^T\right)&\left(dH^T b_j\right) \end{aligned} \quad (3)$$

To apply a function element wise to H's rows, let us denote,

$$E = \sum_{k=1}^{m} Hb_k b_k^T = \sum_{j=1}^{m} b_j H^T {b_j}^T \quad (4)$$

To this we apply the derivative of the Softmax function as,

$$\begin{aligned} \text{dE} &= \sum_{j=1}^{m} b_j \left[\left(M_j - m_j m_j^T\right) dG^T b_j\right]^T \\ &= \sum_{j=1}^{m} b_j b_j^T dG \left(M_j - m_j m_j^T\right) \end{aligned} \quad (5)$$

The self-gradient of each element in the matrix where ε_ℓ are the basis vectors for $\mathbb{R}^n$ is given as $\frac{\partial Y}{\partial Y_{\ell k}} = E_{\ell k} = \varepsilon_\ell b_k^T$. Using this equation, we find the gradient for Y as,

$$H = \lambda X^T A^T BY \in \mathbb{R}^{m\times m},\ dH = \lambda X^T A^T B dY \quad (6)$$

The objective function of the attention block is $O = FTX \in \mathbb{R}^{m\times m}$. Differentiating O w.r.t F and substituting dH,

$$\begin{aligned} dO = dF(TX) &= \sum_{j=1}^{m} b_j b_j^T dG \left(M_j - m_j m_j^T\right) TX \\ &= \sum_{j=1}^{m} b_j b_j^T \left(\lambda X^T A^T B dY\right) \left(M_j - m_j m_j^T\right) TX \end{aligned} \quad (7)$$

Taking partial derivative

$$\begin{aligned} \frac{\partial O}{\partial Y_{\ell k}} &= \sum_{j=1}^{m} b_j b_j^T \left(\lambda X^T R^T S E_{\ell k}\right) \left(M_j - m_j m_j^T\right) TX \\ \frac{\partial O_{im}}{\partial Y_{\ell k}} &= \sum_{j=1}^{m} e_i^T b_j b_j^T \left(\lambda X^T R^T S E_{\ell k}\right) \left(M_j - m_j m_j^T\right) CXb_m \\ &= \sum_{j=1}^{m} \delta_{ij} b_j^T \left(\lambda X^T R^T S E_{\ell k}\right) \left(M_j - m_j m_j^T\right) CXb_m \\ &= b_i^T \left(\lambda X^T R^T S E_{\ell k}\right) \left(M_i - m_i m_i^T\right) TXb_m \\ &= \lambda \left(x_i^T R^T b_\ell\right) b_k^T \left(M_i - m_i m_i^T\right) Tx_m \\ &= \lambda \left(b_\ell^T R x_i\right) b_k^T \left(M_i - m_i m_i^T\right) Tx_m \end{aligned} \quad (8)$$

where $\{b_\ell, x_m\}$ are the columns of $\{S, X\}$ respectively. Next, we take the gradient w.r.t X. We see the following changes in $dH = \lambda dX^T R^T SY$. The self gradient of transpose of X is $\frac{\partial X^T}{\partial X_{\ell k}} = E_{k\ell} = b_k \varepsilon_\ell^T$. We get $O's$ differential as,

$$dO = dE(TX) + (ET)dX \quad (9)$$

Solving steps similar to Eq. (8), we finally arrive at,

$$\frac{\partial P_{im}}{\partial X_{\ell k}} = \lambda \left[a_\ell^T SY \left(M_k - m_k m_k^T\right) Tx_m\right] \delta_{ik} + \left(e_i^T F c_\ell\right) \delta_{mk} \quad (10)$$

where $\{a_\ell, c_\ell\}$ are the columns of $\{R, T\}$.

II. Trajectory Generation

The first step is to check the state of the object's source C_{src} (from STOP) and the targetted global solution state (from COPA). If there is a mismatch, we trace the correct pose C_{dest} of the object's centroid in the targetted global solution. The first step is to connect the neighbouring points with the common dominant intention. Let l_{I_1} and l_{I_2} be two non-intersecting lines corresponding to intentions I_1 and I_2 respectively. The aim is to find the shortest path between C_{src} and C_{dest} which intersects both $l_{I_1}$and l_{I_2}, for which we minimize a convex function of two real variables. The input consists of: C_{src}, C_{dest}, l_{I_1} represented by the point P_{I_1} and direction vector v_1 and l_{I_2} by P_{I_2} and v_2. The path length we aim to minimize is $f(s,t) = ||C_{src} - P_{I_1} - tv_1||_2 + ||P_{I_1} + tv_1 - P_{I_2} - sv_2||_2 + ||P_{I_2} + sv_2 - M_2||_2$. None of the terms can be zero, which implies that f is differentiable. Its partial derivatives are $\frac{\partial f}{\partial s} = \left\langle \frac{P_{I_2}+sv_2-C_{dest}}{||P_{I_2}+sv_2-C_{dest}||_2} - \frac{P_{I_1}+tv_1-P_{I_2}-sv_2}{||P_{I_1}+tv_1-P_{I_2}-sv_2||_2}, v_2 \right\rangle$, $\frac{\partial f}{\partial t} = \left\langle \frac{P_{I_1}+tv_1-P_{I_2}-sv_2}{||P_{I_1}+tv_1-P_{I_2}-sv_2||_2} - \frac{C_{src}-P_{I_1}-tv_1}{||C_{src}-P_{I_1}-tv_1||_2}, v_1 \right\rangle$. Solve for s and t.

For the trajectory parametrization, CILA learns to split, transform and glue trajectories from the wrist trajectory database 'WristNET' for the intention it is maximizing. We describe each step below.

A. Splitting the trajectories

This section describes the algorithm for the subtrajectory splitting module in online trajectory generator module (M4). We use the re-parametrized functions in the previous step to find the control points. For each time interval, we check if the intention with the maximum value. If a new intention attains the maximum value at any interval, we mark it as a control point. For the derivation, let (C_{x1}, C_{y1}) and (C_{x2}, C_{y2}) be two control points, (x_s, y_s) is the start and (x_e, y_e) is the end point of the of the trajectory. t_0 and t_1 are the curves parameter space representing the instantaneous motion of the trajectory, which in our case is a time interval. t_{max} controls the maximum time interval of the wrist actions to sample, using which we define two splitting parameters as $h_0 = t_{max} - t_0, h_1 = t_{max} - t_1$. We derive the parameters a_1, b_1, c_1 and d_1 as $a_1 = x_s h_0^2 + 2C_{x1} t_0 h_0 + C_{x2} t_0^2, b_1 = x_s h_1^2 + 2C_{x1} t_1 h_1 + C_{x2} t_1^2, c_1 = C_{x1} h_0^2 + 2C_{x2} t_0 h_0 + x_e t_0^2, d_1 = C_{x1} h_1^2 + 2C_{x2} t_1 h_1 + x_e t_1^2$. And , the parameters a_2, b_2, c_2 and d_2 as $a_2 = y_s h_0^2 + 2C_{y1} t_0 h_0 + C_{y2} t_0^2, b_2 = y_s h_1^2 + 2C_{y1} t_1 h_1 + C_{y2} t_1^2, c_2 = C_{y1} h_0^2 + 2C_{y2} t_0 h_0 + y_e t_0^2, d_2 = C_{y1} h_1^2 + 2C_{y2} t_1 h_1 + y_e t_1^2$. From the found parameters we find the splitted curve's new endpoints as $x_a = a_1 h_0 + c_1 t_0, y_a = a_2 h_0 + c_2 t_0, x_b = a_1 h_1 + c_1 t_1, y_b = a_2 h_1 + c_2 t_1, x_c = b_1 h_0 + d_1 t_0, y_c = b_2 h_0 + d_2 t_0, x_d = b_1 h_1 + d_1 t_1, y_d = b_2 h_1 + d_2 t_1$. We split the trajectory in two parts between points (x_a, y_a), (x_b, y_b) and (x_c, y_c), (x_d, y_d) into two subtrajectories.

B. Transformations on the subtrajectories

This section describes the algorithm for the subtrajectory transformation module in online trajectory generator module. Each subtrajectory is administered transformations to fit into the suboptimal solutions. Table I summarizes the transformations we apply to a subtrajectory to make it fit into the suboptimal solution. The resultant trajectory is transformation invariant, inspired by the work [1].

TABLE I: Transformations applied to the splitted curves

TF	New coordinate	Geometric significance
$f(x)+k$	$(x,y) \mapsto (x, y+k)$	Shift up by k
$f(x)-k$	$(x,y) \mapsto (x, x-k)$	Shift down by k
$f(x+k)$	$(x,y) \mapsto (x-k, y)$	Shift left by k
$f(x-k)$	$(x,y) \mapsto (x+k, y)$	Shift right by k
$\lambda f(x)$	$(x,y) \mapsto (x, \lambda y)$	Vertical expansion by λ
$\frac{1}{\lambda} f(x)$	$(x,y) \mapsto (x, \frac{1}{\lambda} y)$	Vertical contraction by λ
$f(\lambda x)$	$(x,y) \mapsto (\frac{1}{\lambda} x, y)$	Horizontal contraction by $\frac{1}{\lambda}$
$f(\frac{1}{\lambda} x)$	$(x,y) \mapsto (\lambda x, y)$	Horizontal expansion by λ
$-f(x)$	$(x,y) \mapsto (x, -y)$	Reflection about x-axis
$f(-x)$	$(x,y) \mapsto (-x, y)$	Reflection about y-axis

C. Gluing isomorphic affine trajectories

Let $\{R_i\}$ be a family of trajectory representations maps. For each $i \neq j$, suppose we are given a trajectory subtask's subset $S_{ij} \subseteq R_i$. We are given for each $i \neq j$ an isomorphism of affine trajectories $A_{ij} : S_{ij} \to S_{ji}$ such that for each i, j we have $A_{ij} = A_{ji}^{-1}$ and for each i, j, k we have $A_{ij}(S_{ij} \cap S_{ik}) = S_{ji} \cap S_{jk}$, and $A_{ik} = A_{jk} \circ A_{ij}$ on $S_{ij} \cap S_{ik}$. Whenever these conditions are satisfied, we find that there exists a representation map R, together with morphisms $\mathscr{A}_i : R_i \to R$ for each i, such that

(a) $\mathscr{A}_i$ is an isomorphism of R_i onto the family of trajectory representations of R, **(b)** $\mathscr{A}_i(R_i)$ covers R, **(c)** $\mathscr{A}_i(S_{ij}) = \mathscr{A}_i(R_i) \cap \mathscr{A}_j(R_j)$ and **(d)** $\mathscr{A}_i = \mathscr{A}_j \circ A_{ij}$ on R_{ij}. Thus, we obtain R by gluing the representations in R_i along isomorphisms $\mathscr{A}_i = \mathscr{A}_j \circ A_{ij}$.

Example: Let T_3: $y^2 - x^4 + 7 = 0$ and T_4: $v^2 - 1 + 7u^4 = 0$ be two affine trajectories. We aim to make a smooth trajectory by glueing the affine parts of T_3 and T_4. The solution maps for glueing the two curves are $(x, y) \mapsto (1/x, y/x^2)$ and $(v, v) \mapsto (1/u, v/u^2)$.

D. Algorithm for gluing two affine isomorphic trajectories

(1) In this algorithm we show the steps to constraint the trajectory in the projective space from trajectories in the affine spaces. The projective curve $\mathbb{C}^1$ denotes the gluing of two copies of $\mathbb{A}^1$ via the map $x \mapsto \frac{1}{y}$,, where x is the coordinate of the first copy of $\mathbb{A}^1$ and y is the coordinate of the second $\mathbb{A}^1$. For the algorithm, we shall assign an algebraically closed field k.
(2) $X_1 = \operatorname{Spec} k[x]$ and $X_2 = \operatorname{Spec} k[y]$ $U_{12} = D(x) = \operatorname{Spec} k[x, x^{-1}] \subseteq X_1$ $U_{21} = D(y) = \operatorname{Spec} k[y, y^{-1}] \subseteq X_2$
(3) We define the morphisms $\phi_{12} : \operatorname{Spec} k[x, x^{-1}] \to \operatorname{Spec} k[y, y^{-1}]$ and $\phi_{21} : \operatorname{Spec} k[y, y^{-1}] \to \operatorname{Spec} k[x, x^{-1}]$
(4) The above equations induce the isomorphism $f_{12} : k[y, y^{-1}] \to k[x, x^{-1}]$, $y \mapsto 1/x$ and $f_{21} : k[x, x^{-1}] \to k[y, y^{-1}]$ and $x \mapsto 1/y$.
(5) Generalization: If $\mathfrak{m}$ is a prime ideal of $k[x]$ (respectively $k[y]$), denote by $[\mathfrak{m}]$ the associated point in X_1 (respectively X_2). If we denote the embedding of $X_1 \subseteq \mathbb{P}^1$ on closed points by $[(ax - b)] \mapsto [b : a]$,, the equality $(ax - b) = (x - b/a)$ forces us to have $[b : a] = [b/a : 1]$ for nonzero a. More generally, this means that the homogeneous coordinates $[b : a]$ on $\mathbb{P}^1$ must satisfy the familiar rule $[bc : ac] = [b : a]$ for $c \in k^\times$.
(5) Extension: Moreover, if $\mathfrak{m} = (ax - b) \subseteq k[x]$ then $\phi_{12}([\mathfrak{m}]) = [(by - a)]$. This means that the embedding of X_2 into $\mathbb{P}^1$ must be given by $[(by - a)] \mapsto [b : a]$ for $a, b \in k^\times$. We may extend this compatibly to all of X_2 by saying that $[(by - a)] \mapsto [b : a]$ for all $a \in k, b \in k^\times$. This implies that if we simply view a point $[(ax - b)] \in X_1$ as the coordinate b/a, and if $b \neq 0$, then b/a is identified with the coordinate a/b from X_2 inside $\mathbb{P}^1$ via $[b/a : 1] = [1 : a/b]$.

$$\frac{1}{N} \sum_{i,j} c_{i,j} y_{i,j} \log x_{i,j} \tag{11}$$

where i runs over items/documents, and j runs over classes, x is the prediction, y is the binary label (1 if item i has class j), and $0 < c < 1$ is your confidence. This is a simple modification of the cross entropy. When the confidence c is low, the value of the prediction matters less.

III. Intention classifier evaluation

n is the number of examples, Y_i is the ground truth intention assignment of the i^{th} sub-trajectory. x_i is the i_{th} trajectory. $h(x_i)$ is the predicted intention for the i^{th} trajectory.
We use the following metrics to evaluate the intention classifier model. Firstly, for each trajectory's predicted intention we calculate its score. Then these scores are aggregated over all the trajectories. We calculate precision as the ratio of how

much of the predicted intention is correct using,

$$\frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap h(x_i)|}{|h(x_i)|} \tag{12}$$

The numerator calculates how many of the predicted intentions vector is common with the ground truth, and the ratio computes, how many of the true labels are there in the ground truth. We compute recall as the number of actual intention labels that were predicted,

$$\frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap h(x_i)|}{|Y_i|} \tag{13}$$

The numerator calculation is same as Eq (12) and the ratio is the fraction of the actual intention labels that were predicted.

Next, we compute the precision and recall for each label over the entire dataset and then averaged. We consider averaging of all the metrics for the labels as

$$\frac{1}{q}\sum_{k=1}^{q} C(TP_k, FP_k, TN_k, FN_k) \tag{14}$$

where TP_k, TN_k, FP_k, FN_k are the true positive, true negative, false positive and false negative counts respectively for the k^{th} label and C is the confusion matrix. Next, we consider averaging of each metric individually across the labels as

$$C(\sum_{k=1}^{q} TP_k, \sum_{k=1}^{q} FP_k, \sum_{k=1}^{q} TN_k, \sum_{k=1}^{q} FN_k) \tag{15}$$

IV. Collaborative Object Packing Algorithm ($COPA$) - Description of each constraint in the paper

This section starts with an overview of the COPA steps. Subsequent subsections describe each step in detail. **Input:** Robot initial positions, container initial positions and properties (volume, weight, density), boundary positions of the map where objects are to be packed. **Optimal output:** Object packing poses requiring minimum containers.

The proposed bin packing algorithm has the following characteristics. Online: objects' properties and locations are unknown beforehand, and new objects are added at any time. A batch-sized packing: The algorithm executes separately for every batch, and global optimization solutions from individual batches are merged to minimize the global objective. This may require readjusting and swapping objecting positions. Notations: $s_{i,j}$ is a variable indicating that object i was packed in container j at least once, $x^k_{i,j}$ denotes if object i remained packed in container j through time deadline k, I_i is the set of objects that are incompatible with object i to avoid breakage, A_i is the set of similar class (compatible) objects that must be packed adjacent to object i, W is the set of human-cobots workers (h_1 humanoids, r omni-platforms and m_1 manipulators, h_2 humans), O is the set of the objects to be packed, Q is the set of containers, K is the set of time deadlines, K_o is the amount of time object o was seeking to be packed, R_w is the set of objects assigned to the worker w, d_{o_1,o_2} is the distance between objects o_1 and o_2 ($d = 1$ if they are are adjacent, else $d = 1 + d_{o_1,o_2}$,). $c_{i,j}$ indicates that container having object i contains item j. $p_{i,j}$ is the penalty to pack two incompatible object i, j together. $c_{i,j,k}$ indicates that the container container object i also contains object j and k. V_i and V_c is the volume of the object i and container resp. **Our global objective is to minimize the number of bins and also minimize the number of object readjustments**. The objective function G_ϑ we minimize is: $\sum_i c_{i,i}$ and $\min_{o\in O, q\in Q} s_{o,q}$.

Under the following set of constraints: $\min \sum_{i\le j} c_{i,j} = 1\,(\forall j)$ **(1)**, $c_{i,j} \le c_{i,i}\,(\forall i < j)$ **(2)**, $c_{i,j} + c_{i,k} - 1 \le t_{i,j,k}\,(\forall i \le j < k)$ **(3)**, $\sum_{j\ge i} V_j c_{i,j} + \sum_{i\le j<k} p_{j,k} c_{i,j,k} \le V_c c_{i,i}\,(\forall i)$ **(4)**, $\sum_{q\in Q} x^k_{o,q} = 1, \forall o \in O, k \in K_o$ **(5)**, Meaning: Each object in the current batch must be packed into a container $\sum_{o\in O} x^k_{o,q} \le 1, \forall q \in Q, k \in K$ **(6)**, Meaning: Forbid the collaborating coworkers to place more than one object in the same position the same allocated time deadline. $x^k_{o,q} \le s_{o,q}, \forall o \in O, q \in Q, k \in K_o$ **(7)**, Meaning: Guarantee that an empty container slot can only be packed by the given object on a given time deadline if the variable stating that the similar empty slot of another container was most recently packed by an object with a similar property is set to 1. (Collaborative task experience sharing) $x^k_{o,q} = 0, \forall o \in O, q \in Q, k \in K \backslash K_o$ **(8)**, Meaning: It states that when an object is not packed within K deadlines, it is not assigned to any container immediately. (Prevents a single object from sabotaging the global packing time) $x^k_{o,q} + x^k_{o',q'} \le 1, \forall o \in O, o' \in I_c, k \in K_o \cap K_{o'}, q \in Q, q' \in Q | d_{q,q'} = 1$ **(9)**, $x^k_{o,q} \le \sum_{q'\in Q | d_{q',q}=1} x^k_{o',q'}, \forall o \in O, o' \in A_o, k \in K_o \cap K_{o'}, q \in Q$ **(10)**, Meaning: For contraints (9) and (10), we guarantee that object property incompatibility and proximity constraints such as are respected. Each container has dimensions described in terms of 3D bounding box with height, width and depth dimensions as $\mathbb{H} \times \mathbb{W} \times \mathbb{D}$. Each item also has dimensions a $h_i \times w_i \times d_i$. The container can hold items until no empty space is left. The problem of calculating the empty space even for two items isn't trivial. Given a container and the first item, there are three ways to pack it in the container or fewer ways if the dimensions do not fit. In the available empty space, we have similar ways to pack the second item and so on. We introduce the check constraints before placing the item based on it property. Constraints when no tilting is allowed are $\Sigma^N_{i=1} w \le \mathbb{W}$, $\Sigma^N_{i=1} h \le \mathbb{H}$ and $\Sigma^N_{i=1} d \le \mathbb{D}$. Constraints when tilting is allowed are $w_1 + d_2 \le \mathbb{W}$, $d_1, w_2 \le \mathbb{H}$ and $h_1, d_2 \le \mathbb{D}$. A constraint to avoid breakage when packing items on top of one another is $\Sigma^k_{i=1} w \le w_1$. There are a total of 15 constraints we implement depending on the respective item properties, orientations and their contact faces to fix the packing problems, avoid damages and provide optimal packing. Apart from the three listed above, the remaining 12 constraints implemented in code are: (a) Prevent placing a very small area face on a very large area face, (b) Prevent matching similar area faces to contact, (c) Prevent putting a abrasive surface on a smooth surface (glass), (d) Prevent placement with no stable gripping points, (e) Prevent placing liquid upside-down, (f) To start placing items from the corners, (g) Prevent specific item-item interaction, such as (h) cleaning products pack separate from food, (i) balancing weights of containers (so that a few containers are very heavy and others are very less), (j) Prevent

squishing of items during transit, (k) Over-packing leading to breakage during cobot lifting and (l) reserve space for sufficient cushioning.
$\sum_{o \in R_w} x^k_{o,q} + x^k_{o,q'} \leq 1, \forall s \in S, q \in Q, q' \in Q, k \in K, d_{q,q'} > |R_w|$, Meaning: Ensure that the objects that are the responsibility of the same coworker for packing members are placed close together. (To preserve object packing locality of reference) **(11)**.

V. Proof for readjustment algorithm using hierarchical tree ordering

Lemma 1: For each object O and its parent P, after we visit O from P and union O with P, P and all vertices in the subtree of root O (i.e. P and all descendents of O, including O) will be in one disjoint set represented by P (i.e. ancester[root of the disjoint set] is P).

Proof. Proof: Suppose the tree has height h. Then proceed by induction in object height, starting from the leaf nodes. □

Lemma 2: For each object O, right before we mark it as visited, the following statements are true:
Each O's parents pi will be in a disjoint set that contains precisely pi and all vertices in the subtrees of pi that pi has already finished visiting.
Every visited object so far is in one of these disjoint sets.

Proof. Proof: We proceed by induction. The statement is vacuously true for the root (the only object with height 0) as it has no parent. Now suppose the statement holds for every object of height k for $k0$, and suppose O is a object of height k + 1. Let P be O's parent. Before P visits O, suppose it has already visited its children $c_1, c_2, ..., c_n$. By Lemma 1, P and all vertices in the subtrees of root $c_1, c_2, ..., c_n$ are in one disjoint set represented by P. Furthermore, all newly visited vertices after we visited P are the vertices in this disjoint set. Since P is of height k, we can use the induction hypothesis to conclude that O indeed satisfies 1 and 2. □

We are now ready to prove the algorithm.
Claim: For each query (U, O), the algorithm outputs the lowest common ancester of U and O.

Proof. Proof: Without loss of generality suppose we visit U before we visit O in the DFS. Then either O is a descendent of U or not.
If O is a descedent of U, by Lemma 1 we know that U and O are in one disjoint set that is represented by U, which means ancestor[findSet(O)] is U, the correct answer.
If O is not a descendent of U, then by Lemma 2 we know that U must be in one of the disjoint sets, each of them represented by a parent of O at the time we mark O. Let P be the representing object of the disjoint set U is in. By Lemma 2 we know P is a parent of O, and U is in a visited subtree of P and therefore a descendent of P. These are not changed after we have visited all O's children, so P is indeed a common ancestor of U and O. To see P is the lowest common ancestor, suppose q is the child of P of which O is a descendent (i.e. if we travel back to root from O, q is the last parent before we reach P; q can be O). Suppose for contradiction that U is also a descendent of q. Then by Lemma 2 U is in both the disjoint set represented by P and the disjoint set represented by q, so this disjoint set contains two O's parents, a contradiction. □

References

[1] R. S. Sharma, S. Shukla, H. Karki, A. Shukla, L. Behera, and V. K. Subramanyam, "Dmp based trajectory tracking for a nonholonomic mobile robot with automatic goal adaptation and obstacle avoidance," *International Conference on Robotics and Automation (ICRA)*, 2019.
[2] Github, "Supplementary material," https://github.com/rospackage/TRO1.