IkeaBot: An autonomous multi-robot coordinated furniture assembly system

Citation: Knepper, Ross A., Todd Layton, John Romanishin, and Daniela Rus. "IkeaBot: An Autonomous Multi-Robot Coordinated Furniture Assembly System." 2013 IEEE International Conference on Robotics and Automation (May 2013).

As Published: http://dx.doi.org/10.1109/ICRA.2013.6630673

Publisher: Institute of Electrical and Electronics Engineers (IEEE)

Persistent URL: http://hdl.handle.net/1721.1/90593

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

IkeaBot: An Autonomous Multi-Robot Coordinated Furniture Assembly System

Ross A. Knepper, Todd Layton, John Romanishin, and Daniela Rus

Abstract— **We present an automated assembly system that directs the actions of a team of heterogeneous robots in the completion of an assembly task. From an initial user-supplied geometric specification, the system applies reasoning about the geometry of individual parts in order to deduce how they fit together. The task is then automatically transformed to a symbolic description of the assembly—a sort of blueprint. A symbolic planner generates an assembly sequence that can be executed by a team of collaborating robots. Each robot fulfills one of two roles: parts delivery or parts assembly. The latter are equipped with specialized tools to aid in the assembly process. Additionally, the robots engage in coordinated co-manipulation of large, heavy assemblies. We provide details of an example furniture kit assembled by the system.**

I. Introduction

Automated assembly was one of the earliest applications of robotics. Conventional assembly robots operate affixed to the factory floor in an environment where uncertainty is managed and engineered away by careful human design. In the coming generation, agile assembly systems will become increasingly adaptable to changing circumstances through the incorporation of mobile manipulator robots. To accommodate the additional freedom of a mobile base, uncertainty must be managed by the robots themselves.

In this paper we present a cooperative robot system capable of assembling simple furniture kits from IKEA. We describe here an entire planning and assembly system, beginning with raw parts and ending with an assembled piece of furniture. The robots perform geometric and symbolic planning, assume different roles, and coordinate actions to complete the assembly.

The robots – mobile manipulators with simple end effectors – are capable of locating parts in the workspace and performing screwing operations to connect the parts. Because the robot's own end effectors are not capable of executing a natural screwing maneuver, we developed a novel tool that can deliver continuous rotational motion to furniture components of various sizes. Many furniture assembly operations, for example screwing a table leg onto a table, require cooperation between robots for placing and holding a part and for applying the screwing tool to execute the assembly. Figure 1 shows two robots collaborating to screw a table leg into the table top. All computation occurs on board the robots in a distributed fashion. For this demonstration, we use a team of two KUKA youBots.

*This work was supported by the Boeing Company.

The authors are with the Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, 32 Vassar St, Cambridge, MA 02139 USA. {rak,tlayton,johnrom,rus} at csail.mit.edu

Fig. 1. Two robots collaborating in the assembly of a table.

The main contributions of this assembly system are as follows. First, we developed and implemented a geometric reasoning system that is capable of discovering the correct arrangement for attaching parts to one another even without knowing the final goal shape. Second, we describe a new object-oriented language for representing symbolic planning problems. Third, we describe a novel system of modular tools made to fit over the robot's end-effector. In particular, we introduce a new tool design capable of grasping and screwing a variety of objects. Fourth, we discuss a system in which robots coordinate to flip over an object that is too large and heavy for one robot to manipulate, such as furniture. We believe that this paper represents the first autonomous robotic system to assemble a piece of IKEA furniture.

II. Related Work

The application of autonomous assembly has received significant attention from the robotics community.

A number of automated assembly planning systems deduce from geometric data the correct assembly sequence [7]. These systems employ CAD models of the individual parts in order to constrain the search process and validate a choice of assembly sequence [11, 18]. Such systems are given a geometric model of the individual parts as well as of the entire assembly—how all the parts fit together. In this paper, we describe a system that does not require the full assembly as input. Instead, the system deduces the geometric relations among the parts based on possible alignments of the attach points.

One important use case for robotic assembly systems occurs in space exploration. Stroupe et al. [16] discuss a

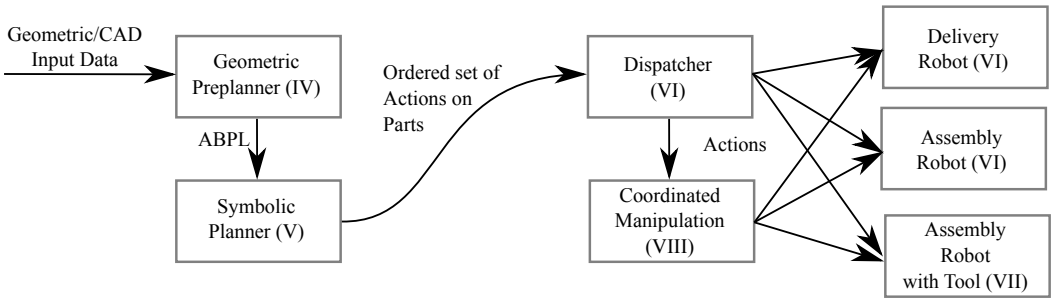

Fig. 2. System architecture. The paper section describing each module is shown in parentheses.

system architecture for assembly of structures in orbit or for planetary colonization. The authors also discuss a modular tool changing system somewhat similar to our own. Heger [4] describes a space assembly system that incorporates planning and execution, with an emphasis on error detection and recovery. Neither geometric reasoning, nor multi-robot coordination were a focus of this work.

In the furniture assembly domain, Spröwitz et al. [14] employ modular robots as smart parts to actually become furniture using decentralized computation.

Rus et al. [13] discuss the conditions for a set of mobile robots (or fingers) to induce rigid body motions on an object, such as a piece of furniture, for manipulation purposes. These principles apply both to the manipulation of parts during attachment (with fingers) and to the maneuvering of fully assembled furniture.

III. Coordinated Assembly System

The furniture assembly system uses several mobile manipulators capable of locating and connecting furniture parts. For the physical experiments in this paper, the parts are connected with screws, but the architecture and planning systems support a general class of connectors.

The architecture for the furniture assembly system, depicted in Fig. 2, is organized into several modules that direct the flow of information from basic geometric parts data into an assembled piece of furniture. In the first stage, a geometric preplanning module solves for the configuration of the completed furniture, based only on the form and quantity of the individual parts—we describe the geometric preplanner in Section IV. The output of this first stage, a symbolic blueprint, describes the final assembly configuration, but it does not provide an order of assembly. Consequently, the blueprint is then fed into stage two, a symbolic planner discussed in Section V. This planner is capable of reasoning about available robots and parts. The symbolic planner outputs a sequence of operations that lead to a correct assembled structure.

In stage three, those actions are dispatched to individual robots or teams of robots in sequence in order to execute the task. The execution of assembly actions is described broadly in Section VI. We use a modular system of custom tools to aid manipulation, as described in Section VII. In Section VIII, we discuss the implementation of coordinated manipulation involving multi-robot teams. Finally, in Section IX we describe the demonstration of the complete system for the assembly of an IKEA Lack table.

IV. Geometric Preplanning

The IkeaBot system begins the planning process by piecing together individual components based on their geometric description, similar to the work of Ambler and Popplestone [1], Thomas and Torras [17]. However, we extend that reasoning to output a blueprint for assembly comprising a set of static coordinate transforms describing the final assembled position and orientation of each part as affixed to the others. To uncover this information, the algorithm searches through possible subassemblies—that is, sets of one or more attached parts. In order to support future data acquisition by machine vision, a fuzzy match is performed between subassembly pairs. For now, augmented CAD files are supplied by the user.

A. Representation

Geometry data take the form of a set of CAD files as well as a database specifying each hole in a part. As an example, the geometric input file for the IKEA Lack table is shown in Listing 1. At present, these files are generated by hand, but in the future they could be automatically generated from stereo vision or RGB+D data.

The input file provides a list of part types as well as the number of instances of each part. Since pegs or screws are normally used to attach parts, the locations of these holes within each part are key to understanding how the parts fit together. Consequently, most of the space in the input file is dedicated to their description.

Each part type is specified in the file by three fields.

- `file`: the name of a CAD file describing the geometry (holes may be present or absent in the CAD data),
- `center`: the approximate center of mass,
- `holes`: a list of holes.

For each hole, three fields are required.

- `diameter`: either a number or the name of a fastener,
- `position`: the center of the hole in the surface plane,
- `direction`: inward-pointing vector.

The `center`, `position` and `direction` fields are expressed in part coordinates. Note that `direction` is specified as an inward-pointing vector rather than a full orientation because the fasteners used here are radially symmetric, allowing the parts to rotate around this axis with respect to each other. An optional fourth field, `pass_through`, (absent in the example) applies to holes that pass fully through the part and specifies the distance from `position` along the `direction` vector before the exit hole is reached. Such holes are usually attached by nails or conventional screws. It is important for the geometric reasoning engine to understand the correspondence between the two ends of a hole because the assignment of one end removes the other from future consideration as well.

Listing 1 Example input file for the geometric preplanner describing parts for the the IKEA Lack table.

```
parts: {table_top: 1, leg: 4}
attach_parts: {double_screw: 4}
hole_diameters: [0.006]  # meters
attach_diameters: {double_screw: 0.006} # meters

table_top: {
  file: table_top.xml, # table is 0.551 x 0.551 x 0.050 m
  center: [ 0.28, 0.28, 0.025 ],
  holes: [ { diameter: double_screw,
             position: [0.026, 0.026, 0.050],
             direction: [0, 0, -1] },
           { diameter: double_screw,
             position: [0.525, 0.026, 0.050],
             direction: [0, 0, -1] },
           { diameter: double_screw,
             position: [0.026, 0.525, 0.050],
             direction: [0, 0, -1] },
           { diameter: double_screw,
             position: [0.525, 0.525, 0.050],
             direction: [0, 0, -1] }
         ]
     }

leg: {
  file: leg.xml, # leg is 0.400 x 0.050 x 0.050 m
  center: [ 0.200, 0.025, 0.025 ],
  holes: [ { diameter: double_screw,
             position: [0.0, 0.025, 0.025],
             direction: [1, 0, 0] }
         ]
    }
```

B. Relational Inference

At each step, the algorithm attempts to discover a coordinate transform that will join two subassemblies together. One or more holes from each subassembly are used to propose candidate transforms. Then, checks are performed to validate the candidate transforms before acceptance.

The algorithm first selects a pair of holes—one hole each from two subassemblies—to associate. In joining two holes, the algorithm computes a transform such that the two subassemblies' hole positions are coincident, thus reducing the degrees of freedom by three. Furthermore, the holes are oriented such that the direction vectors are parallel and opposite, thus further reducing the freedoms by two.

A pair of holes joined in this manner permits a single degree of rotational freedom around the axis of the hole. To resolve the orientation, a second hole from each subassembly is chosen (if available) and one subassembly is rotated about its hole's axis so as to minimize error in lining up this second pair of holes. The minimum-error orientation resolves the final degree of freedom and fixes a prospective transform for mating one subassembly to another. At this point, the iterated closest point (ICP) algorithm is used to draw a correspondence among any remaining pairs of holes in the two subassemblies. The error in matching all pairs of holes figures into an overall match score for the given transform on these two candidate subassemblies. Closest holes separated by more than a threshold value are not included in the match and remain free for future use.

If one subassembly has only a single hole free, then there are no other holes to help resolve the rotational ambiguity. In this case, an orientation is selected arbitrarily. Other orientations can be tested until one is found that validates.

Several validation checks are employed to any proposed subassembly transforms before they can be accepted. First, the parts are collision-checked to verify that the proposed orientation of the two subassemblies does not lead to self-intersection. Minor penetration can occur due to imperfections in the model, thus incurring only a small penalty. More significant penetration invalidates a proposed mating transform. An additional check examines the free holes not used in a particular mating operation. If a hole in one subassembly is covered by another part, then that hole becomes unavailable for future use. Such free hole proximity queries as well as collision checks are performed using OpenRAVE [3]. Part proximity to an unused hole creates a penalty in the scoring function. A mating computed by hole matching and validated based on the above tests is considered a *plausible mating*.

The algorithm employs a number of tunable parameters to score the quality of a mating of two subassemblies. For geometry files generated by hand, the algorithm is not sensitive to the particular choice of values. For geometry files obtained from perception data, it will likely become necessary to learn a reasonable tuning of these parameters.

C. Search Algorithm

Search proceeds in a depth-first fashion through the space of plausible matings of subassemblies. The algorithm is initialized with all components in a completely unassembled state. To begin the search, the algorithm instantiates the correct number of subassemblies, each representing a singular instance of one of the basic parts.

During search, the planner exhaustively computes plausible mating transforms that lead to one or more holes lining up between adjacent parts. A *plausible assembly* has been found when two conditions are met: (1) all components are attached, and (2) no unfilled holes remain. Each plausible assembly is scored based on the scores of individual plausible matings. The highest-scoring plausible assembly is returned. The system utilizes this plausible assembly to construct a blueprint for use by the symbolic planner.

Listing 2 Machine-generated input file (blueprint) for the symbolic planner, which builds an upright IKEA Lack table.

```
type Robot {
    object arm {
        property(Object) holding = None;
    }
}
type AttachmentSpot {
    property(Object) attached_to = None;
}
type TableTop {
    group hole(AttachmentSpot)[4]{}
    property(bool) upside_down = True;
}
type Leg {
    object hole(AttachmentSpot){}
}

object table_top(TableTop){}
group leg(Leg)[4]{}
group robot(Robot)[2]{}

action pick_up_leg(robot(Robot), leg(Leg)) {
    pre {
        robot.arm.holding == None;
        leg.hole.attached_to == None;
    }
    post {
        robot.arm.holding = leg;
    }
}
action attach_leg_to_top(robot(Robot), leg(Leg), table_top(TableTop)) {
    pre {
        robot.arm.holding == leg;
        table_top.hole[0].attached_to == None;
    }
    post {
        robot.arm.holding = None;
        table_top.hole[0].attached_to = leg.hole;
        leg.hole.attached_to = table_top.hole[0];
    }
}
action flip_table(robot(Robot)[2], leg(Leg)[4], table_top(TableTop)) {
    pre {
        leg[2].hole.attached_to == table_top.hole[1];
        leg[3].hole.attached_to == table_top.hole[0];
        leg[0].hole.attached_to == table_top.hole[2];
        leg[1].hole.attached_to == table_top.hole[3];
        table_top.upside_down == True;
    }
    post {
        table_top.upside_down = False;
    }
}

goal assembly(table_top(TableTop)) {
    table_top.upside_down == False;
}
```

V. Symbolic Planning

IkeaBot determines an order of operations for assembly using a symbolic planner. The planner attempts to discover an action sequence such that the preconditions of each action are satisfied. The postconditions of an action support the preconditions of future actions, thus allowing progress toward the goal.

It should be noted that the symbolic planner ignores the order in which the geometric preplanner discovered the mating of subassemblies. Rather, the symbolic planner relies only on the blueprint generated by the geometric preplanner to identify the goal of the assembly. Assembly operations may require that components be assembled in a completely different order than that previously discovered by static geometric analysis.

Blueprints are specified using a newly-designed planning language. ABPL ("A Better Planning Language") is an object-oriented symbolic planning specification language, exemplified in Listing 2. Conceptually similar to OPL [5], ABPL describes planning problem data in a manner which respects the logical and intuitive features of the physical environment as a planning space. ABPL enables an intuitive statement of the problem by logically organizing concepts using various object-oriented programming (OOP) paradigms.

ABPL aims to overcome the necessary complexity of expressing object-oriented relations within first-order logical systems such as PDDL, the Planning Domain Definition Language [9]. PDDL relies entirely on a flat, unstructured representation of the objects in the environment and the relations between them. The burden of creating structure, such as regions of symbolic symmetry or commonalities between multiple objects, falls entirely on the user. While such systems are capable of describing OOP structures, requiring the user to express each element of those structures as a set of propositional statements would be time-consuming and burdensome to the user. ABPL, in contrast, allows the user to provide data in a more conventional object-oriented format. An ABPL problem specification can be about one-quarter the size of the equivalent PDDL specification. This simplicity improves readability and ease of ABPL problem creation, whether manual or automatic.

A. Specification Language Design and Structure

ABPL is based on the object-oriented approach to data structure design, allowing the user to hierarchically organize objects within the environment. Objects can be assigned properties that evaluate to either Boolean values or references to other objects. These values can be changed by the effects of symbolic actions to reflect those actions' consequent alterations of the environment. Objects themselves can be defined either at the top level in the global space, or as sub-elements of other objects. These sub-object relations can be syntactically referenced in the same way as object-reference properties, but are semantically different in that their values cannot be changed after declaration. This fact is meant to convey the distinction between an object's sub-objects, which represent fixed components of that object, and its object-typed properties, which represent mutable inter-object relations. ABPL also allows the user to define "types," which fulfill the role of object classes. A type can be assigned properties and sub-objects, which are then replicated in each object that is declared to be of that type. Types can also be used as action and goal predicates, for example to restrict a symbolic action variable to values of a specific type.

An important distinction from other object-oriented symbolic planners is the inclusion of groups. Groups represent the existence of multiple identical instances of a given object. For example, Lack tables have four legs, declared simply as `group leg(Leg)[4]`.

Groups enable the planner to reason efficiently about one type of symmetry. Because the four legs are identical, they are interchangeable. Thus any leg can be attached to any corner of the table, and they can be installed in any sequence. When referencing a member of the group, it is sufficient to reference the first member of that group. The planner knows that this rule extrapolates to all other group members as well. Subsequently, one may reference the second group member to indicate any table leg except the one already referenced.

B. ABPL Implementation for Assembly

IkeaBot's planning module uses the ABPL specification for its raw input and output data. At present, there is no

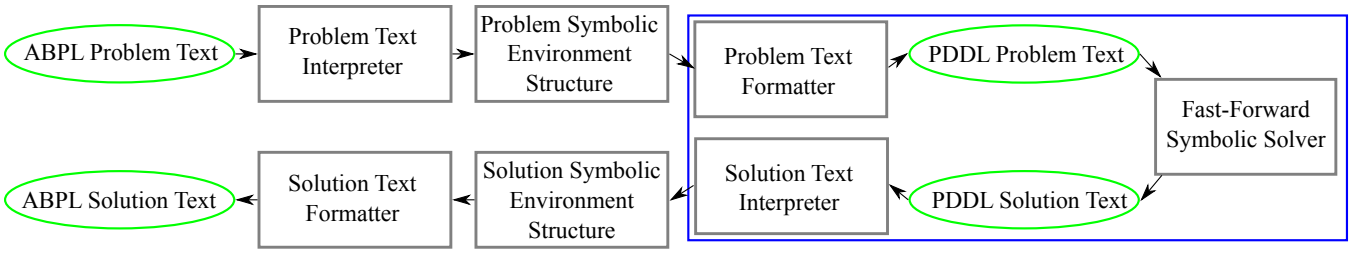

Fig. 3. Symbolic planner architecture. At present, the planner converts ABPL into PDDL and executes the Fast-Forward solver. In the future, this process (the large box at right) may be replaced with a direct ABPL solver that can incorporate algorithmic hooks and other features unavailable in PDDL.

symbolic solver that operates directly on ABPL. Rather, we convert symbolic data from one format to another using a three step process: a text interpreter (built using the SPARK little language framework [2]), a symbolic environment structure, and a text formatter. The process is outlined in Fig. 3. A planning problem is passed through this process twice; once to convert ABPL data into a PDDL format for use by the internal symbolic solver, Fast-Forward [6], and again to reformat the resulting solution sequence back into ABPL. Alternatively, the calling program, such as the main IkeaBot system, can retrieve the solution sequence in data structure form rather than as ABPL solution text.

The current design of this planning program is predicated on the assumption that any given planning problem's physical constraints can be expressed entirely symbolically. As this is a significant limitation when working with the geometric constraints inherent to the physical domain, future work includes the extension of ABPL's grammar to allow for external program calls out of the solver in order to incorporate non-symbolic data and logic into the planning process. An important example is the use of a motion planner to determine when it is feasible to attach two subassemblies.

VI. Assembly Execution

Furniture assembly is accomplished by a large set of small manipulation actions, each executed by the right robot with the right part(s) at the right time. To orchestrate this activity among an arbitrary number of robots in a distributed fashion, we utilize a modified version of the system described by Stein et al. [15].

In that system, a static blueprint describing part locations in an absolute frame of reference is passed to a partitioner (running on all assembly robots), which divides responsibility for installing each of the parts among the set of available assembly robots. Meanwhile, a set of delivery robots stands by, ready to retrieve and hand off the parts needed by the assembly robots for installation.

We adapted this framework in order to increase its flexibility for the IkeaBot assembly problem. Specific challenges we address are heterogeneous parts without fixed global coordinates, heterogeneous robot capabilities, and collaborative multi-robot manipulation activities.

We replace the partitioner with a dispatcher, in recognition of the fact that the salient assignment is over actions rather than parts. The dispatcher remains a distributed process over all assembly robots. Actions employ a fixed number of robots (one or two in our IKEA Lack table example) to one or more subassemblies. Listing 3 shows an example ABPL-encoded input to the dispatcher. In the work of Stein, et al., parts may be initially ineligible for installation, requiring that new parts get allocated to assembly robots during the course of assembly. Similarly, some actions may be initially unready for execution due to their preconditions. A partial ordering, if available from the symbolic planner, informs the dispatcher when each action becomes eligible for execution.

Listing 3 Example input file (recipe) for the dispatcher to construct the IKEA Lack table.

```
step pick_up_leg(robot[0], leg[1]);
step attach_leg_to_top(robot[1], leg[1], table_top);
step pick_up_leg(robot[0], leg[3]);
step attach_leg_to_top(robot[1], leg[3], table_top);
step pick_up_leg(robot[0], leg[2]);
step attach_leg_to_top(robot[1], leg[2], table_top);
step pick_up_leg(robot[0], leg[0]);
step attach_leg_to_top(robot[1], leg[0], table_top);
step flip_table(robot, leg, table_top);
```

In mobile manipulation, most actions require navigation and obstacle avoidance capabilities. IkeaBot achieves these capabilities using the Model-Based Hierarchical Planner [8]. It provides navigation through an environment cluttered with furniture parts and other robots. Navigating robots utilize a sampling-based reciprocal algorithm to anticipate the reaction of other robots as well as human pedestrians.

VII. Tool Usage

Some manipulation problems are best solved in hardware. In this section, we describe a modular system of hot-pluggable, dockable tools that is designed to interface with the KUKA youBot gripper and wrist design.

A. Interchangeable Gripper System

In order for the robots to use tools effectively, we designed a system that allows tools to be quickly removed or replaced with other tools. Manufacturing automation is replete with existing systems for tool interchange at scales too large for the KUKA youBot. Our system allows a single robot to accomplish a variety of manipulation activities autonomously with compact tools, thus greatly increasing the functionality of the robots.

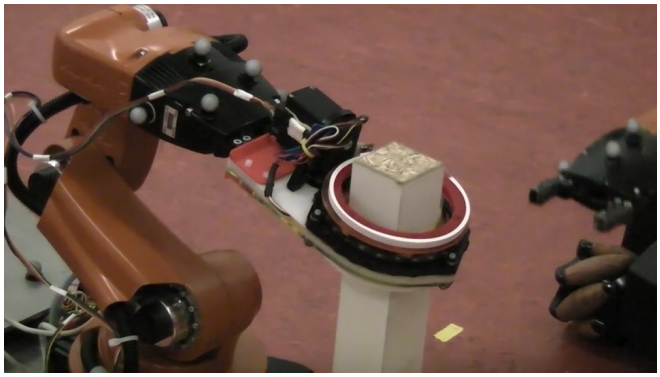

Fig. 4. The Torq gripper employed for table assembly.

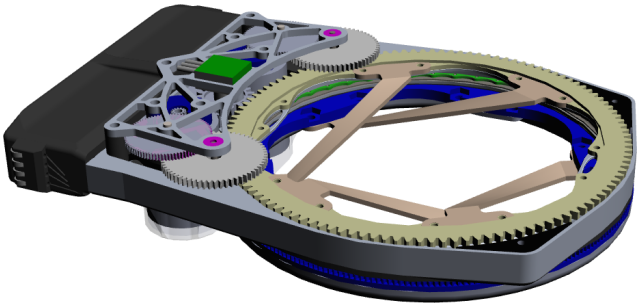

Fig. 5. Solid modeler depiction of the screwing device. The powered upper ring is shown in tan; the passive lower ring is blue. The black part on the left is the coupling point to the KUKA youBot end-effector.

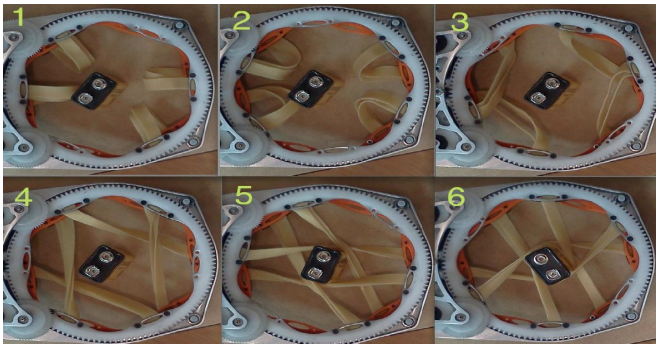

Fig. 6. A series of frames show the Torq gripper gripping and spinning an object.

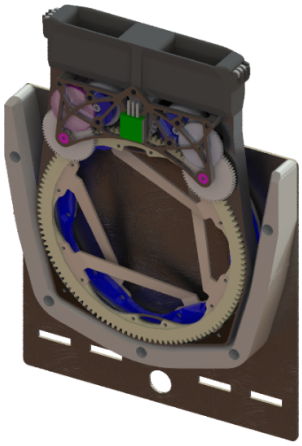

Fig. 7. The Torq gripper in its docking station. Due magnetic and gravitational force, the tool settles into a unique position.

B. Torq Gripper Design

We designed a spinning gripper tool, the Torq Gripper, which specializes in screwing-type operations. We observed that the KUKA youBot arm, having only five degrees of freedom, lacks the capability to rotate its wrist side-to-side. This limitation makes screwing operations very difficult.

Our design specification requires a versatile tool capable of applying high torque (up to 3 Nm) over many revolutions to a shape with either circular or non-circular cross section. The new robotic gripper uses multiple elastic cables to compliantly constrain and then spin an object, as in Fig. 4.

A somewhat similar design was employed by Nguyen et al. [10] in the design of the gripper for the Space Shuttle's Canadarm manipulator. The Canadarm employs a set of cables that constrict around a part as a snare. Our design contributes two primary distinctions geared for the furniture assembly task. First, the elasticity and resulting compliance of the cables allows the robot to handle uncertainty in the relative position and orientation of the part. Second, both ends of each cable connect to movable rings, which enables the device to easily rotate the part after it has been grasped.

The Torq gripper functions by the encirclement of the target object with flexible elastic members. These members are attached to two separate rings (see Fig. 5). The top ring is driven by a pair of motors, whereas the bottom ring is unactuated. The unactuated ring contains a set of magnets designed to increase the ring's static friction. As the driven ring is spun by the motors, the elastic elements begin to contract around the object in the center as can be seen in Fig. 6. After a threshold torque is reached, the bottom ring and the object begin to spin as well in a stick/slip fashion.

This design enables the rings to maintain an approximately constant gripping force on the part. The gripping strength is controlled by the combination of the elasticity of the cables and the force exerted by the magnets. Elasticity permits the tool to be non-destructive in its application of force, which is distributed almost uniformly around the circumference of object. Additionally, the elastic nature of the force application permits considerable error in position and orientation during operation. More information about the Torq gripper and its design can be found in the work of Romanishin [12]. Below, we describe the coupling mechanisms whereby the robot picks up and docks the gripper and other tools.

1) Attachment to Robot Base: Figure 7 shows the system that has been designed to hold the Torq gripper on the robot body when not in use. The design takes advantage of the magnets embedded in the unactuated ring, which attach the tool to a steel plate. The dock design guides the tool into a unique rest state for later retrieval.

2) Attachment to Robot Arm: When needed, the tool is retrieved from its dock by pulling directly away from the steel surface. The tool is shown coupled to the KUKA youBot hand in Fig. 8. Eight magnets attract flexure-based

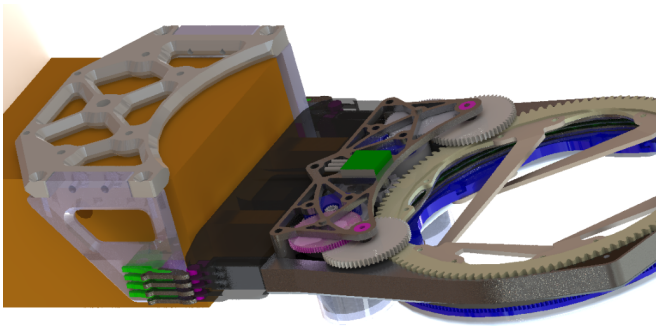

Fig. 8. A model of the Torq gripper attached to the end of the KUKA arm. Magnetic contacts are shown at left.

steel electrical contacts in order to make the necessary power and data connections. These magnets serve an additional purpose of guiding the coupling into position. The KUKA youBot gripper then securely grasps a custom handle located inside the housing of the tool. An Arduino microcontroller on the robot sends commands and interprets sensor signals.

VIII. Coordinated Manipulation

For a variety of reasons, several robots must sometimes come together in collaborative teams in order to complete the overall assembly task. Some subassemblies may be too heavy or large for a single robot to manipulate effectively, or more robots may be required in order to fixture all of the parts involved in a single attachment operation. In this section, we address two issues of coordinated manipulation: dynamic teaming and co-manipulation with decentralized coordinated control. An example of two robots cooperatively flipping a table can be seen in Fig. 9.

We address the resource allocation issue of dynamic teaming in a greedy manner. Each robot's dispatcher allocates the set of currently valid actions, including both single-robot and team actions. Assembly robots perform all part-attachment actions that can be performed without help. Delivery robots receive no singleton assignments besides delivery tasks. However, delivery robots are eligible to participate as part of a team when called upon to do so.

Because it is harder to allocate multiple resources at once, team activities take priority over singleton actions. After the dispatcher recognizes that the preconditions for a team task have been satisfied, the robots involved finish their current action and then proceed to perform the task in a coordinated fashion.

Team actions frequently possess a compound nature, and so they are implemented as a state machine. For the table flipping example, states include coarse (global-frame) navigation, precision (part-frame) navigation, opening and closing the gripper, and most importantly, following through the mechanics of a flipping motion.

The flipping motion implements a virtual hinge at one corner of the table and follows through 90 degrees of travel at a time. At the conclusion of each 90-degree interval, the position of the virtual hinge is then updated to the next corner before the action continues.

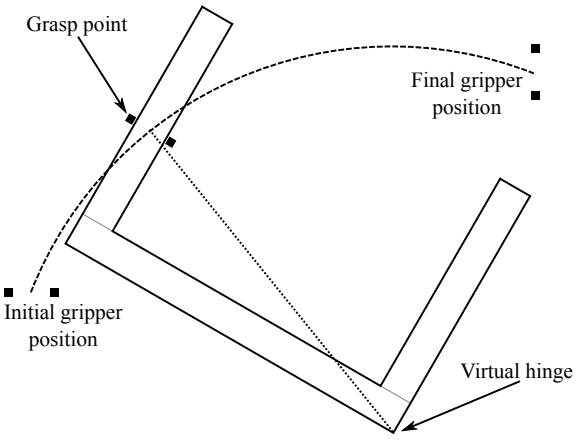

Fig. 10. The geometry of kinematic object flipping. Starting from an arbitrary grasp point, the gripper rotates 90° about a virtual hinge, maintaining a constant distance.

In order to compute the desired motion of the hand to kinematically flip an object, the robot tracks the position of the gripper with respect to the virtual hinge. The robot computes the desired velocity of its own end-effector in Cartesian coordinates in order to track an arc. See Fig. 10. Each robot adjusts the velocity of motion to match the other's pace. Joint velocities are then computed via the Jacobian. Execution terminates after the grippers sweep through 90° of rotation (the normal case) or when joint limits or singularities are reached. At present, the starting shape of the arm is carefully selected by hand to avoid these kinematic failure modes. In the future, this process will be automated.

IX. Hardware Demonstration

We demonstrated the capabilities described in this paper using two KUKA youBots, which assembled an IKEA table. Beginning from geometric descriptions of the parts, the robots automatically compute a blueprint and assembly plan.

During execution, a Vicon motion capture system provides localization for the robots and the table-top, although the table legs are unmarked. Instead, table legs are available at a fixed location in the global Vicon reference frame.

The robots expect the table-top to begin in the inverted position, as indicated in the ABPL blueprint shown in Listing 2. The robots can assemble the table flat on the ground or elevated on a platform. The latter condition, as depicted in Fig. 9, assists the robots in flipping the IKEA Lack table, whose weight exceeds the combined rated 1 kg maximum load capacity of a pair of youBots.

In addition to table assembly, the robots know how to make a stool or two-level table out of a pair of half-height Lack tables by stacking one on top of another.

The overall runtime to fully assemble the Lack table is approximately ten minutes. Of that time, virtually all is spent in execution. Geometric preplanning and symbolic planning consume only a few seconds each. The following list reports approximate times involved in executing specific tasks as part of the assembly: pick up leg: 20 s, place leg in hole: 13 s, hand off leg: 6 s, screw in leg: 45 s, flip table: 50 s.

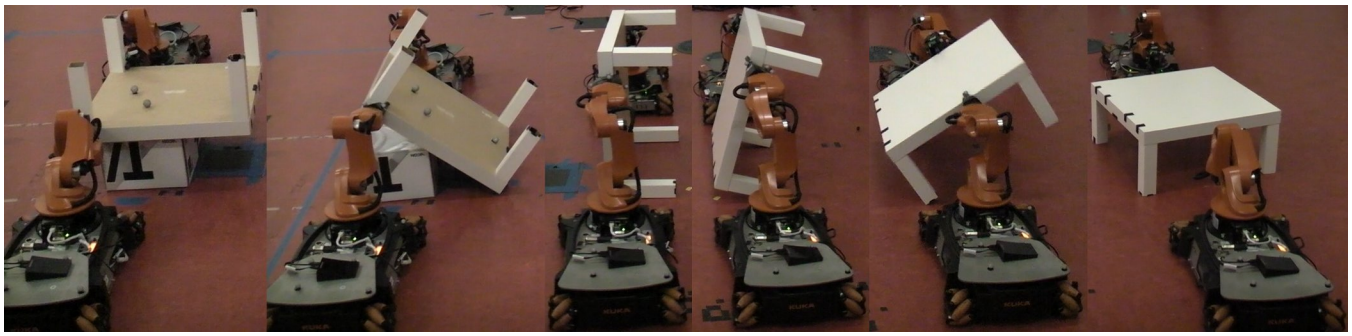

Fig. 9. This sequence of images shows the process by which two robots flip a table.

To test robustness of the implementation, we performed twelve repeated trials of the table assembly. Of those, nine were completely successful. In three trials, a screw missed the hole. Once, the screwing device failed due to software. In all, we observed 48/48 successful pickups, 44/48 successful placements, and 47/48 successful attach operations. Minor human assistance permitted all trials to run to completion.

X. Discussion and Future Work

In this paper, we describe an implementation of a furniture assembly system. Parts of the planning system are quite general in capability, such as planning "from scratch" with only the geometric form of the components as input—not even their assembled shape. We introduce a new language that is intuitive for humans and robots that efficiently expresses symbolic planning problems. We describe a modular system for powered tool use by the KUKA youBot. We discuss a distributed task allocation system for a team of robots that is capable of dynamically reassigning tasks as needed, subject to the capabilities of each robot, demonstrated through the assembly of an IKEA Lack table. Finally, we discuss an approach to multi-robot coordination for co-manipulation, illustrated through the flipping of the table.

Future work primarily revolves around making the system more generic. We plan to generalize the implementation of symbolic actions for manipulation to broaden the variety of furniture kits the system can assemble. We also intend to generalize the collaboration framework to achieve a variety of co-manipulation tasks besides object-flipping. Finally, failure detection and recovery will be added for robustness.

References

[1] A. P. Ambler and R. J. Popplestone. Inferring the positions of bodies from specified spatial relationships. *Artificial Intelligence*, 6:157–174, 1975.

[2] J. Aycock. Compiling little languages in python. In *Proceedings of the 7th International Python Conference*, 1998.

[3] R. Diankov. *Automated Construction of Robotic Manipulation Programs*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[4] F.W. Heger. *Assembly Planning in Constrained Environments: Building Structures with Multiple Mobile Robots*. PhD thesis, Carnegie Mellon University, Robotics Institute, August 2010.

[5] A. Hertle. Design and implementation of an object-oriented planning language. Master's thesis, Albert-Ludwigs-Universität Freiburg, 2011.

[6] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, month 2001.

[7] L.S. Homem de Mello and A.C. Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, 7(2):228–240, April 1991.

[8] R.A. Knepper and D. Rus. Pedestrian-inspired sampling-based multi-robot collision avoidance. In *Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication*, Paris, France, September 2012.

[9] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—the planning domain definition language. Technical Report CVC TR98003/DCS TR1165, Yale Center for Computational Vision and Control, New Haven, USA, 1998.

[10] P.K. Nguyen, R. Ravindran, R. Carr, D.M. Gossain, and K.H. Doetsch. Structural flexibility of the shuttle remote manipulator system mechanical arm. Technical report, SPAR Aerospace Ltd., 1982.

[11] J. Latombe R.H. Wilson. Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71:371–396, 1994.

[12] J. Romanishin. Development of a robotic torque application gripper for automated furniture assembly, 2012. Undergraduate Thesis.

[13] D. Rus, B. Donald, and J. Jennings. Moving furniture with teams of autonomous robots. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, August 1995.

[14] A. Spröwitz, P. Laprade, S. Bonardi, M. Mayer, R. Mckel, P. Mudry, and A. Ijspeert. Roombots-towards decentralized reconfiguration with self-reconfiguring modular robotic metamodules. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, IEEE International Conference on Intelligent Robots and Systems, pages 1126–1132, Taipei, Taiwan, 2010.

[15] D. Stein, T.R. Schoen, and D. Rus. Constraint-aware coordinated construction of generic structures. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2011.

[16] A.W. Stroupe, T. Huntsberger, B. Kennedy, H. Aghazarian, E.T. Baumgartner, A. Ganino, M. Garrett, A. Okon, M. Robinson, and J.A. Townsend. Heterogeneous robotic systems for assembly and servicing. In *Proceedings of the International Symposium on Artifical Intelligence, Robotics and Automation in Space*, Munich, Germany, August 2005.

[17] F. Thomas and C. Torras. Inferring feasible assemblies from spatial constraints. *IEEE Transactions on Robotics and Automation*, 8(2):228–239, 1992.

[18] R.H. Wilson. Minimizing user queries in interactive assembly planning. *IEEE Transactions on Robotics and Automation*, 11 (2), April 1995.