Asynchronous Motion Planning and Execution for a Dual-Arm Robot

Charles A. Meehan
U.S. Naval Research Laboratory
Code 5514
Washington, DC 20375
charles.meehan@nrl.navy.mil

Mark Roberts
U.S. Naval Research Laboratory
Code 5514
Washington, DC 20375
mark.roberts@nrl.navy.mil

Laura M. Hiatt
U.S. Naval Research Laboratory
Code 5514
Washington, DC 20375
laura.hiatt@nrl.navy.mil

Abstract

Asynchronous motion execution enables dual-arm robots to move their arms concurrently and independently, making them more capable and efficient. However, asynchronous motion requires coordination to avoid collisions between the two arms. Prior work has approached dual-arm motion execution by either constraining the motions of the arms to execute one motion at a time or by synchronizing both arms during planning. In contrast, we introduce an approach for asynchronous motion execution that leverages a trajectory reservation component that, when combined with off-the-shelf motion planning, prevents collisions between the arms. This approach reserves that projected space for each arm by creating collision objects in the environment that correspond to planned trajectories. In two simulated demonstrations using a DRC-Hubo humanoid robot, we show that this approach avoids collisions while also being more efficient.

1 Introduction

A challenge for dual-arm robots is ensuring that the two arms, which share a joint physical workspace, operate in a safe and collision-free way. To avoid collisions between the arms in these concurrent situations, previous methods either synchronized the planning and motion execution of both arms or made adjustments to one of the arm's velocities and accelerations to accommodate the motion of the other arm (Buhl et al. 2019; Kimmel and Bekris 2017). There are other works that coordinate robot arms if they are working on a task together (such as working together to carry objects) (Buhl et al. 2019; Salehian, Figueroa, and Billard 2018; Yu et al. 2021).

In contrast, we are interested in situations where the arms are working asynchronously on separate tasks in a shared workspace but can operate concurrently to increase the robot's efficiency. For example, an industrial robot could pick up an item from a shelf with one arm while picking up another item with the other arm as seen in Figure 1. Our work encourages *asynchronous* concurrency by opportunistically dispatching an arm's actions for execution even if the other arm is currently executing a task. To accomplish this, we introduce a trajectory reservation component that, at execution time, enables and encourages this concurrent, asynchronous motion. It does this by creating collision objects that correspond to the arm's planned trajectory, which "reserve" portions of the environment for the planned motions.

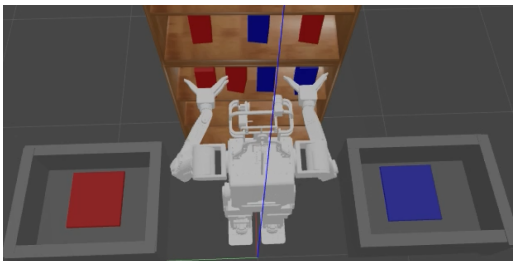


Figure 1: Dual-Arm Robot concurrently executing two pick actions.

When the planned trajectory is executed, the collision objects are then removed as the arm passes them. These collision objects prevent a plan for the opposite arm from interacting with the projected plan, which allows concurrent and safe execution of motions for the arms independently.

We demonstrate this approach on a DRC-Hubo dual-arm robot (Heo et al. 2019) operating in the Gazebo simulation environment (Koenig and Howard 2004) in two scenarios: (1) the Hubo using both arms to push colored cans on a table into their matching color bins; (2) the Hubo completes pick and place tasks in a much larger workspace with much more interaction between the objects that need to be moved. We run both scenarios for three different static motion planning algorithms (RRT-Connect, PRM, and LazyPRM). When enabled, the reservation component eliminates all collisions between the arms and it is more efficient than executing motions sequentially.

2 Related Work

Several works address dual-arm motion planning. We focus on two aspects of the literature concerning the work that addresses dual-arm motion planning and the literature on asynchronous motion planning.

2.1 Dual-Arm Motion Planning

Buhl et al. (2019) operates dual-arm robots by motion planning in static environments where nothing else is moving; for a dual arm robot, this often means planning and moving one arm at a time. While this can guarantee safety in the execution of movements, it overly constrains motion and cannot

take advantage of the full flexibility of dual-arm motion.

There are also approaches that focus on synchronizing the motions of two arms moving at once. Wang et al. (2019) describe a motion planner to find a single solution for both arms, ensuring they do not collide with each other during movement. While this provides the ability to make concurrent movements, the necessary synchronicity of the arms lessens the flexibility of the approach. For example, if one arm finishes its motion before the other, that arm would need to wait for the second arm to finish before it could begin moving again. For true movement flexibility, it is necessary to have asynchronous motion planning and execution that supports concurrency.

Other methods for dual-arm concurrent motion impose some type of time coordination between the arms during movement. For instance, there are approaches that use a velocity tuning technique that allows the arms to solve for their motion paths independently and then coordinate their movements together by finding the velocity that allows both arms to move into the same space while avoiding collisions (Kimmel and Bekris 2017).

2.2 Asynchronous Motion Planning and Execution

Asynchronous motion planning and execution for a dual-arm robot involves coordinating motion between the arms in order to avoid self-collisions. In (Salehian, Figueroa, and Billard 2018), self-collisions were avoided by using a centralized inverse kinematics solver under self-collision avoidance constraints. To calculate the self-collision avoidance constraints, a continuous and continuously differentiable function was learned from a data set of "collided" and "non-collided" multi-arm configurations (Salehian, Figueroa, and Billard 2018). Once calculated, this function represented the region of feasible and infeasible robot configurations which was used to figure out if the configuration of the arms will be in a "collided" configuration or not. Their method is a reactive method whereas, our method reserves collision-free space for the arms' trajectories in the environment before movement.

Moving two arms in the same workspace can modeled as a multi-agent problem, where each arm is an independent agent. In (Alami et al. 1998; Gravot and Alami 2001), multiple agents independently plan their motions but synchronize their motion executions with their neighboring agents. This brings in comparative work dealing with asynchronous motions of multiple agents with safety guarantees, as described by Grady et al. (2010). In that work, the agents' operation is broken into intervals called "cycles" of a constant duration. During each cycle, which are not synced between the multiple agents, an agent will plan multiple possible trajectories for its next movement and listen for broadcasts of other agent trajectories in the neighborhood, the size of which is defined by the authors. As the time approaches the end of the cycle, the agent will select a trajectory that will not cause conflicts with its neighbor's trajectory, and then will broadcast its chosen trajectory to reserve this space for itself. This work inspired part of our approach, where arms independently reserve the space of the environment that corresponds to their current movement trajectory so that the other arm does not plan a motion in that space.

3 Trajectory Reservation Component

We build on several open source packages: MoveIt™ (Coleman et al. 2014) for motion planning, Gazebo for simulated execution, and RViz as a visualization tool (Hershberger, Gossow, and Faust 2008). Asynchronous motion execution and planning is not possible within MoveIt as is, so we first describe how we modified MoveIt to support asynchronous motion planning. We then discuss how our novel reservation component works with motion planning to enable the Hubo robot to execute motions without collisions between the arms.

Modifying MoveIt. As part of our use of the MoveIt software platform (Coleman et al. 2014), we used the Open Motion Planning Library (OMPL), a library of state-of-the-art sampling based motion planning algorithms (Şucan, Moll, and Kavraki 2012), and the Flexible Collision Library (FCL), "a library for performing three types of proximity queries on a pair of geometric models composed of triangles" (Pan, Chitta, and Manocha 2012), to conduct motion planning and collision checking. To enable asynchronous motions, we created a threaded software component that at any time can receive a maximum of two joint space or pose goals (one for each arm of the Hubo robot) from our central control component. When such a goal is received, the motion planning component sends the goal, along with a snapshot of the current representation of the environment, to OMPL using MoveIt's MoveGroupInterface software class. OMPL, in conjunction with FCL, then attempts to find a collision free motion path.

Depending if a collision-free motion plan was found or not, our motion planning component will either send a failure or a success message to our central control component. The failed motion plan actions are sent back to the motion planning component for re-planning. The actions with a successful motion plan will continue on in the motion planning component and have their motion plan turned into a trajectory with velocities and accelerations for the joints by the MoveIt's time parameterization process.

Adding Trajectory Reservation. In order to ensure the safe execution of these trajectories, our reservation component updates the environment to reflect the arm's imminent execution trajectory. For each waypoint in the trajectory, it creates a set of collision objects that match the size and pose of certain links of the robotic arm at that waypoint as shown in Figure 2. Again, these collision objects serve to reserve space in the world for that arm so that if OMPL is called to find a collision-free motion plan for the other arm while this motion is being executed, it will not find a collision-free plan through this space. Therefore, the second arm will not receive a motion plan that would cause the arm to collide with the already-moving arm.

After the environment is updated by our reservation component, the trajectory is sent to the joint trajectory controller for that arm in order to move the arm in Gazebo. The joint

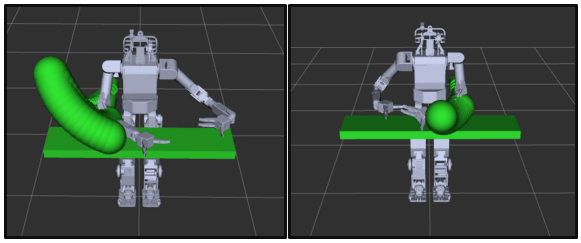
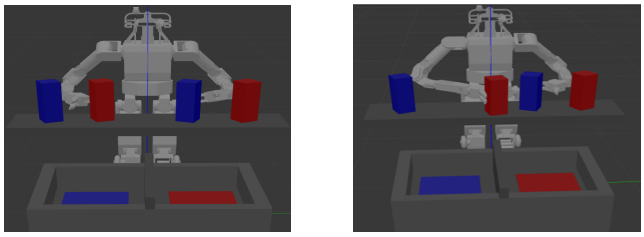


Figure 2: Planned right and left arm trajectories shown as collision objects in RViz.



(a) (b)

Figure 3: (a) Push scenario initial set up. (b) Situation where the left arm cannot start pushing the blue can due to the right arm occupying that space.

trajectory controller for each arm was set up using ROS Control (Chitta et al. 2017). The execution of these actions are tracked and once completed are removed from a list of tasks maintained by the central control component.

4 Evaluation

One claim of this work is that using our novel reservation component prevents collisions between the arms from occurring while the arms are moving asynchronously in the workspace. Further, the main benefit of asynchronous motion execution is being able to complete tasks more efficiently. We evaluated these claims first by creating two scenarios, a push scenario and a pick and place scenario. For each scenario, we used the DRC-Hubo robot (Jung et al. 2018) in the Gazebo simulator (Koenig and Howard 2004). The robot has two 7-DOF arms that are equipped with a 1-DOF gripper; note that the robot has additional degrees of freedom from the hip and leg joints which are not used for any of our evaluations. One assumption for both scenarios was that the Hubo Robot was operating in a static environment besides the motions of its arms.

For both scenarios, we investigate three motion planning algorithms. **RRT-Connect** was the default planner chosen by MoveIt for our robotic scenarios. To this we added the **Probabilistic Roadmap Method (PRM)** and **LazyPRM** as our other planners to use for this evaluation since they both differ from RRT-Connect by being multi-query planners instead of single-query planners (Kavraki et al. 1996; Bohlin and Kavraki 2000). Also, both PRM and LazyPRM return paths that are optimal versus the sub-optimal paths returned by RRT-Connect, but are slower than RRT-Connect in computing a motion planning solution.

For each scenario and motion planning algorithm, we ran 5 trials with our reservation component and 5 trials without our reservation component in order to test if our reservation component prevented collisions between the arms. In order to test the efficiency of executing motions asynchronously rather than sequentially, we recorded the total time of the scenario (push and pick and place) execution, including the time required to motion plan every task, for five trial runs while executing motions asynchronously. Then, we ran five more trials with tasks executed sequentially and again recorded the total time of scenario (push and pick and place) execution for each trial, including the time required to motion plan every task. For both sequential and asynchronous motion execution, we used our trajectory reservation system and RRT-Connect.

4.1 Hubo Robot Push Scenario

In the first scenario, the DRC-Hubo robot has the goal of pushing colored cans into their matching colored bins. There are four cans set up in pre-specified positions on a table at waist height of the robot (Figure 3a). The positions and orientations of the cans are assumed to be known by the robot. To push the cans into their corresponding bins, the robot needs to coordinate its movements so that its arms do not collide with one another. For collision planning purposes, the environment was updated with the table as a collision object set at the same height as in Gazebo. This ensures that the Hubo robot does not plan motions that would collide with the table.

During the push scenario, the Hubo robot will use our motion planning, trajectory reservation, and central control components as discussed in Section 3 until all cans are pushed off of the table into their matching bins and the arms have returned to their home position. An example of our reservation component preventing a collision from occurring during the push scenario is shown in Figure 3b. Here, the middle of the workspace was reserved by our trajectory reservation system which prevented the left arm from pushing the middle blue can since that would cause a collision with the right arm. While this space was reserved, the motion planning component would send back motion failure messages to the central control component, and the central control component continues to ask for another motion plan for the left arm until a success message is sent from the motion planning component. These procedures resulted in the left arm holding its current position until the right arm moved out of the way. Once the right arm moved out of the way, the motion planning component receives a valid motion plan and sends a success message to the central control component, and the left arm can execute its planned motion.

4.2 Hubo Robot Pick and Place Scenario

To further evaluate our work, we created a pick and place scenario. In this scenario, the Hubo robot had to complete 57 total tasks in a larger workspace, as compared to only 16 tasks for the push scenario. The main objective for the pick and place scenario was to have the Hubo Robot pick

up colored cans from shelves and place them into their corresponding colored bins. As Figure 1 shows, a red bin was placed on the left side of the Hubo, and a blue bin was placed on the right side of the Hubo. The parts of the environment such as the bookshelf and bins that could cause collision to Hubo's arms as they are moving around were generated as collision objects to avoid during motion planning. The focus of this work is motion planning and not grasping, so we did not pursue real grasping of the cans and instead attached them to the hands when in close proximity.

The motion planning, trajectory reservation, and central control components follow the procedure stated in Section 3 until all the cans are placed in the bins and Hubo returns to a home position. The initial setup for the pick and place scenario is shown in Figure 1. Similar to the push scenario, there were cases where the Hubo robot had to wait for one arm to complete its current motion in order to avoid colliding with that arm. For example, the red and blue cans in the middle of the bottom shelf are placed such that the end effectors could collide into one another if they try to complete the pick motion as shown by the left arm in Figure 4.

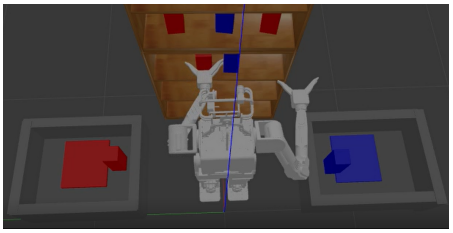


Figure 4: Situation where the right arm cannot start its pick action due to the left arm's end effector occupying that space.

5 Results

For the Hubo Push Scenario, Table 1 shows no collisions between the arms while using our reservation component. However, we had a major collision between the arms that did not have the reservation component. In each case, this collision occurred while one arm was pushing a can in the middle of the table and the other arm tried to push its can into that same space. For the trial runs testing the efficiency of asynchronous motion execution, with asynchronous motion, the average time of scenario execution, including the time taken for motion planning each task, was one minute, 19 seconds and 17 milliseconds $\pm$ two seconds and 13 milliseconds from the start of the first task to the completion of the last task. With sequential motion, the average time of scenario execution, including the time taken for motion planning each task, was one minute, 52 seconds and 3 milliseconds $\pm$ one second and 36 milliseconds. Therefore, the total time of scenario execution with asynchronous motion was 34% faster than with sequential motion.

For the Hubo Pick and Place Scenario, Table 1 shows no collisions between the arms for all runs when our reservation component was used. There were multiple collisions on the runs without our reservation component. The collisions occurred while the arms were reaching for the middle cans on the bottom shelf. For the one run with LazyPRM and not using the reservation component that did not have a collision, this was due to the left arm having larger planned motions between each task which meant that the right arm completed its middle pick and place action for the blue can on the bottom shelf before the left arm started moving into the same area.

Push Scenario Number of Collisions			
Motion Planner	runs	Baseline	TRC (ours)
RRTConnect	5	5	0
PRM	5	5	0
LazyPRM	5	5	0

Pick and Place Scenario Number of Collisions			
Motion Planner	runs	Baseline	TRC (ours)
RRTConnect	5	9	0
PRM	5	10	0
LazyPRM	5	8	0

Table 1: Number of collisions for both scenarios with our trajectory reservation component (TRC) and without our component (Baseline).

For the trial runs testing the efficiency of asynchronous motion execution, with asynchronous motion, the average time of scenario execution, including the time taken for motion planning each task, was four minutes, 14 seconds and 45 milliseconds $\pm$ 16 seconds and 29 milliseconds from the start of the first task to the completion of the last task. With sequential motion, the average time of scenario execution was 6 minutes, 11 seconds, and 51 milliseconds $\pm$ 16 seconds and 51 milliseconds. Therefore, the total time of scenario execution with asynchronous motion was 37% faster than with sequential motion execution.

6 Conclusions and Future Work

We introduced a trajectory reservation component that, combined with off-the-shelf motion planning, enables asynchronous, concurrent motion planning and execution for a dual-arm robot. Using trajectory reservations eliminated *all collisions* when the arms move concurrently and asynchronously for the two scenarios we implemented. It was also more efficient than the common dual-arm baseline case of executing motions sequentially, executing sequences of tasks more quickly.

Future work will continue to develop our asynchronous motion planning components, including accounting for dynamic obstacles in the environment. We plan to enable the motion planning algorithm to anticipate and avoid collisions with these obstacles by using the same reservation component to broadcast the trajectories of the dynamic obstacles.

Acknowledgments. The authors thank ONR and NRL for funding this research.

References

Alami, R.; Fleury, S.; Herrb, M.; Ingrand, F.; and Robert, F. 1998. Multi-robot cooperation in the MARTHA project. *IEEE Robotics and Automation Magazine* 5(1):36–47.

Bohlin, R., and Kavraki, L. 2000. Path planning using lazy PRM. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, 521–528 vol.1.

Buhl, J. F.; Grønhøj, R.; Jørgensen, J. K.; Mateus, G.; Pinto, D.; Sørensen, J. K.; Bøgh, S.; and Chrysostomou, D. 2019. A dual-arm collaborative robot system for the smart factories of the future. *Procedia Manufacturing* 38:333–340.

Chitta, S.; Marder-Eppstein, E.; Meeussen, W.; Pradeep, V.; Rodríguez Tsouroukdissian, A.; Bohren, J.; Coleman, D.; Magyar, B.; Raiola, G.; Lüdtke, M.; and Fernández Perdomo, E. 2017. ros_control: A generic and simple control framework for ros. *The Journal of Open Source Software*.

Coleman, D.; Sucan, I.; Chitta, S.; and Correll, N. 2014. Reducing the barrier to entry of complex robotic software: a moveit! case study.

Grady, D. K.; Bekris, K. E.; ; and Kavraki, L. E. 2010. Asynchronous distributed motion planning with safety guarantees under second-order dynamics. *Algorithmic Foundations of Robotics IX* 53–70.

Gravot, F., and Alami, R. 2001. An extension of the plan-merging paradigm for multi-robot coordination. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 3, 2929–2934 vol.3.

Heo, J.-W.; Lee, J.; Lee, I.-H.; Lim, J.; and Oh, J.-H. 2019. History of hubo: Korean humanoid robot. in: Goswami a., vadakkepat p. (eds) humanoid robotics: A reference. springer, dordrecht.

Hershberger, D.; Gossow, D.; and Faust, J. 2008. rviz. Accessed on: 09.14.2021.

Jung, T.; Lim, J.; Bae, H.; Lee, K. K.; Joe, H.-M.; and Oh, J.-H. 2018. Development of the humanoid disaster response platform DRC-HUBO+. *IEEE Transactions on Robotics* 34(1):1–17.

Kavraki, L.; Svestka, P.; Latombe, J.-C.; and Overmars, M. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12(4):566–580.

Kimmel, A., and Bekris, K. E. 2017. Scheduling pick-and-place tasks for dual-arm manipulators using incremental search on coordination diagrams.

Koenig, N., and Howard, A. 2004. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2149–2154.

Pan, J.; Chitta, S.; and Manocha, D. 2012. Fcl: A general purpose library for collision and proximity queries. In *2012 IEEE International Conference on Robotics and Automation*, 3859–3866.

Salehian, S. S. M.; Figueroa, N.; and Billard, A. 2018. A unified framework for coordinated multi-arm motion planning. *The International Journal of Robotics Research* 37:1205–1232.

Şucan, I. A.; Moll, M.; and Kavraki, L. E. 2012. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine* 19(4):72–82. https://ompl.kavrakilab.org.

Wang, J.; Liu, S.; Zhang, B.; and Yu, C. 2019. Inverse kinematics-based motion planning for dual-arm robot with orientation constraints. *International Journal of Advanced Robotic Systems* 16(2).

Yu, X.; Zhang, S.; Liu, Y.; Li, B.; and Yinsong Ma, G. M. 2021. Co-carrying an object by robot in cooperation with humans using visual and force sensing. *Philosophical Transactions A* 379:1–13.