Single and Dual Arm Manipulator Motion Planning Library

Behnam Asadi[1]

Abstract— **In this work a library for solving manipulator motion planning problems has been developed and an algorithm for imposing Cartesian constraints in single arm and dual arm operation has been proposed. The main algorithm is based on RRT [1]. The code has been tested with several single arm and dual arm robots. Our method is able to find collision free paths in environments occupied with obstacles. An example of such an environment is the shelf from the Amazon picking challenge. The implemented code also enabled us to perform complicated dual arm operations such as rotating a hand wheel or opening a drawer.**

Keywords: Manipulator motion planning, dual arm operation, planning under Cartesian constraint, robotics.

I. INTRODUCTION

Nowadays, robots are no longer just machines in factory production lines and are gradually entering human everyday life and taking part in our activities. One of the key challenges for robots is to solve motion planning problems. The task of motion planning is to find a path for moving the robot from the start state to the goal state such that the robot does not collide with itself or any other object in the environment. Various constraints including joint, torque, velocity and acceleration limits might be imposed to the task. A classical example of motion planning is the piano mover's problem. In this problem, a piano (3D rigid object) exists in a room in the presence of some obstacles. The task is finding a path for moving the piano from start pose to the goal pose while avoiding obstacles. It has been shown that the piano mover's problem is PSPACE-hard [2]–[5]. Motion planning has many applications in robotics including manipulation, autonomous cars and also in other fields such as analysis of biological molecules (protein folding), drug design and computer animation. The rest of this paper is organized as follows: In section II, we review different approaches for solving a motion planning task. In section III, the main pipe line of the library is introduced. In section V, our main algorithm for dual arm operations and our approach for imposing constraints is explained. We explain the library functionalities for motion planning in dynamic environment in section VI. Finally in section VII, the results of our approach tested on different robots is reported.

II. AN OVERVIEW OF CURRENT APPROACHES FOR MOTION PLANNING TASK

Different methods have been suggested for solving a motion planning task, including combinatorial planning, search based planning, sample based planning, optimization based planning and potential fields. In combinatorial motion planning, free space is decomposed into regions called cells. One popular algorithm for this purpose is trapezoidal decomposition [6], [7]. Cells with a common boundary are adjacent. An adjacency graph representing connected cells (via shared boundary) is being created based on that. Next, the planner determines the start and the goal cells and then searches the graph for a path connecting these two cells. As an advantage, the approach is complete and for problems with low dimensionality it is extremely efficient. A disadvantage is that planning becomes impractical as the dimensionality of the configuration space increases [8], [9]. In search based planning, the planner discretizes the work space and determines free and occupied cells. Then it generates a graph representing the planning problem and searches the graph for a solution. Several methods have been used for discretization and searching the graph including *SBPL* [10], A^*, D^* [11] and R^* [12]. Advantages of search based algorithms are their ability to incorporate various cost functions and trajectories with natural looking behaviour. The drawbacks are proper discretization of configuration space (which might be a complicated task) and defining an appropriate heuristic function. In sample based method, during the planning phase, the samples are drawn randomly in configuration space. If a sample is valid (i.e. no collisions or any other criteria set by the user) and close enough to the other generated samples (or start and goal states) it will be added to a tree. The planner continues making samples and adding them to the tree, until it can find a path from the start state to the goal state. Trees can be grown from both start and goal states (bidirectional based planner) and checking for state validation might be done when a path has been found (lazy evaluation). Some sample based algorithms are: PRM, EST, RRT, SRT and their variants. Sample based planners are very robust for finding a path even in high-dimensional configuration spaces. On the other hand, these algorithms are probabilistic and the generated path is usually jerky with redundant movements and requires smoothing and optimization. A motion planning task can be seen as leading a particle in a potential field with the same electric charge towards a goal with opposite charge. The obstacles will repulse the particle (robot) and the goal will attract it. This idea has been first introduced in [13] and [14]. The advantage of potential fields is their simplicity to implement and real-time computation. The drawbacks are oscillations in an unstable state and trapping in local minima [15], [16]. The most important issues that most of the motion planners have taken into account are joint limits and obstacle avoidance while in many applications there are other criteria that should be satisfied including constraints handling, path smoothing and minimization of torque, jerk and energy consumption. In optimization based planning, the planner defines a cost function in which the cost is weighted according to collision, path length, path smoothness, energy consumption and joint/ torque limits. Then the planner tries to reduce the cost by calling an optimizer over the generated path. In CHOMP (Covariant Hamiltonian Optimization for Motion Planning) [17] the environment is modelled with signed distance field. The planner at first step creates a naive initial trajectory, from the start to the goal state. Then it runs a modified version of gradient descent on a cost function which is a sum of cost of trajectory smoothness and collision with obstacles (penetration depth). Afterwards the planner tries to improve the initial path in every iteration. The advantages of this method is a smooth and natural looking trajectory. Due

[1]Behnam Asadi is a researcher at mathematics and computer science department of university of Bremen, Germany. behnam.asadi@gmail.com

This work was supported through two grants of the German Federal Ministry of Economics and Technology (BMWi, FKZ 50 RA 1216 and FKZ 50 RA 1217).

to the use of gradient descent, the answer might lay on a local minimum. STOMP (Stochastic trajectory optimization for motion planning) [18] relies on an optimizer that can handle cost functions where their gradients are not available. In STOMP the cost function is the combination cost of colliding with obstacles (penetration depth), constraints and torque. The optimizer algorithm explores the space around an initial trajectory by generating a series of noisy trajectories. These trajectories are then combined to produce an updated trajectory with lower cost. Since the STOMP doesn't depend on gradients, it can deal with general constraints (i.e. pose or torque constraints) and it overcomes the local minima problem of the CHOMP.

III. Main Algorithm and Pipe Line for Motion Planning

We have created our sample based motion planning library based on **OMPL** [19] and we have been mainly inspired by **MoveIt!** [20], [21] for the main pipeline and architecture, however we have added support for dual arm operation, imposing Cartesian constraints, self filtering and planning in a dynamic environment.

OMPL is an open source motion planning library that covers most of the state of the art sample based motion planning algorithms and MoveIt! is software for mobile manipulation including tools for 3D perception, collision checking, solving inverse and forward kinematics queries and control of the robot.

Due to the dependency of MoveIt! on the ROS framework, it was not possible to integrate it with our robotic framework, *ROCK* [1].

As it has been briefly discussed in section II, sample based motion planners generate random states in configuration space. Then by applying forward kinematics, we check the validity of the state (self collision and collision with environment). In the case of validity of the newly drawn sample, it will be added to the RRT. This process continues until a path between start and goal configuration can be found or some termination condition are met (i.e planning time exceeds maximum allowed time).

A. IK Solver With Minimum Movements and Jerk for High Degree of Freedom

One of the most important parts of the motion planning problem is solving inverse kinematic queries. Two main approaches for solving inverse kinematic problems are numerical solutions and analytical (closed form) solutions.

KDL [22] (Kinematics and Dynamics Library)[2] is a library for computing forward and inverse kinematic queries with numerical solutions. Due to recursive nature of numerical solutions for solving IK problems, they might be very slow or trap in local maxima. *Openrave* (Open Robotics Automation Virtual Environment) [23] is a software for simulating and deploying planning algorithm. It provides a tool called *IKFast* which can generate *C++* code (independent of any library) representing the model of the manipulator for solving forward and inverse kinematic queries. The generated code is able to solve the request on the order of 4 microseconds [24].

One key issue with using IKFast for solving IK queries is finding proper values for manipulators free joints (redundant joints). Furthermore, we would like to have a hierarchical structure for free joints such that the joints that are closer to the root of kinematic chain have less movements relative to those that are closer to end effector. The motivation for such a decision is that we like to have minimum movements and jerk during the manipulation. For instance, consider a kinematic chain in a humanoid robot from knee to the right (or left) wrist. In the case we want to use this kinematic chain for grasping an object, we like to use the joints in the lower part less (in torso or knee) and use more joints in the arm.

To overcome this issue, we discretize the space between the upper and lower joint limits for every free joint and search for possible values starting from either a given configuration (configuration specified by user or from the IK solution of the previous Cartesian way point in the trajectory or the middle point between joint lower and upper bound). This starting point policy is very crucial since starting from joint lower limit and checking values for solution toward upper limit will bias the IK solution to the joint limits, specially when there is more than one free joint in the chain. The code has been implemented recursively so it is capable of handling arbitrary number of free joints. Algorithm 1 describes the recursive function for finding free joints parameter.

```
Input  : Robot model, Vector of free joints, Cartesian pose
Output: Values for free joint, IK solution
Recursive_free_joints_value (values_for_free_joints_vector,
index)
if (index > 0) then
    while (joint lower bound < pivot + step) ||
    (pivot + step < joint upper bound) do
        step← step + step_size
        values_for_free_joints_vector.push_back(pivot+step)
        if recursive_free_joints_value
        (values_for_free_joints_vector, index-1) then
            return true
        end
        values_for_free_joints_vector.pop_back()
        values_for_free_joints_vector.push_back(pivot-step)
        if recursive_free_joints_value( values_for_free_joints,
        index-1) then
            return true
        end
        values_for_free_joints_vector.pop_back()
    end
else
    while (joint lower bound < pivot + step) ||
    (pivot + step < joint upper bound) do
        step ← step + step_size
        if IKFast(Cartesian pose,values_for_free_joints_vector,
        pivot+step, index)|| IKFast(Cartesian
        pose,values_for_free_joints_vector, pivot-step, index)
        then
            return true
        end
    end
end
return false
```

Algorithm 1: Algorithm for finding free joint parameter for IK queries with IKFast.

IV. Motion Planning Under Cartesian Constraints

Motion planning tasks always come with various constraints. These constraints make some states impermissible for the robot. Primitive examples of such constraints are collision avoidance and joint limits.

In many manipulation tasks some other constraints should also imposed on the robot to satisfy the problem conditions. Examples

[1] http://rock-robotics.org

[2] http://www.orocos.org/kdl

of these are constraints on the orientation of the end effector or on the X,Y,Z Axis of the end effector.

Various solutions proposed by researchers for imposing constraints on EEF in [25]–[28]. These algorithms are either not efficient or developed for a specific constraint representation [29]. The authors in CBiRRT [30] developed a framework for imposing constraints represented by TSRs (task space regions). The algorithm has a constraint satisfaction strategy (projection/ rejection) and general planning algorithm (sample based planner). CBiRRT employs the robustness of RRT planners to explore the configuration space and enforce samples with projection/ rejection policy based on gradient decent [29].

In the motion planning library that we developed, we add the capability of setting Cartesian constraints for single arm and dual arm tasks. The main algorithm for motion planning under constraints is RRT based. The planner always checks if any constraints have been set during the initialization of planner. In the case a constraint has been set, the planner will change the sampling space from joint space to Cartesian space and it plans for the manipulated object. In other words, the drawn sample represents the object position. The samples are drawn within the upper and lower bound of imposed constraint to satisfy the task requirements. The Cartesian constraint which are imposed to the planner are similar to TSRs in CBiRR. By having the grasp pose for the object and object pose, the end effector pose for the arm can be determined. Afterwards, we search for a valid inverse kinematic solution (by taking into account joints limit, self collision and collisions with environment). When a valid solution is found, it will be added to the RRT. If the planner can find a path between start and goal pose within the time limit for planning (or before the maximum number of iteration reaches), the trajectory will be published.

```
Input  : Robot model, motion planning request, Cartesian
         Constraint
Output: Collision free trajectory
while Time has left for planning do
    Draw a random sample from Cartesian space Between
    upper and lower constraint limits;
    if There is IK solution for the drawn sample then
        Update the robot state with IK Solution
        if robot is in self collision state then
            if Collisions are not in the allowed collisions list
            then
                next
            end
        else if robot is not in Collision with the environment
        then
            Add the newly drawn sample to the RRT
    else
        next
    end
    if There is a valid path between start and goal in RRT
    then
        publish the path
        return
    end
end
```

Algorithm 2: Motion planning under Cartesian constraints algorithm

V. Dual Arm Motion Planning

While in the domain of robotics, manipulation can be interpreted as an interaction with an object, dual arm manipulation doesn't have a common and general accepted definition [31]. "In fact, many authors do not distinguish between multi-agent or multi-arm systems" [31]. Employing two arms for manipulation has several advantageous. Using two arms gives the robot more strength (loading a heavy object for example) and for the tasks that are initially designed for human, it gives robot better dexterity compared to one arm [31].

The Authors in [32], classified dual arm operation into **non-coordinated manipulation**, where each arm is doing separated motions (manipulating an object with one arm while moving another object with a second arm for instance), and **coordinated manipulation** (opening a bottle of wine for example), in which the arms are working on the same task. Coordinated manipulation can be subdivided into **goal-coordinated** (playing piano, typing with keyboard) and **bimanual** manipulation.

In goal-oriented manipulation, in which both arms are solving the same task, the arms are not physically interacting with each other. In bimanual operations motion of one arm is widely based on motions of the other one which might be *symmetric, asymmetric, congruent or non-congruent* [32].

In this work, we have designed three scenarios for performing dual arm operations which requires bimanual manipulation. We have achieved planning for these tasks by using same code and only changing the parameters of imposed constraints. The first task is rotating a hand wheel, the second one is opening/closing a drawer and the third one is moving an object with both arms while keeping the orientation. The interaction between EFFs and hand wheel/ drawer is performed via a grasping component which is based on vision and force control, developed by DFKI [3] research center.

```
Input  : Robot model, motion planning request, Cartesian
         Constraint
Output: Dual arm collision free trajectory
while Time has left for planning do
    Draw a random sample representing object from Cartesian
    space Between upper and lower constraint limits;
    if There is IK solution for the drawn sample for the first
    arm then
        Update the robot state with IK Solution;
        if There is IK solution for the second arm from
        updated robot state then
            if robot is in self collision state then
                if Collisions are not in the allowed collisions
                list then
                    next
                end
            else if robot is not in Collision with the
            environment then
                Add the newly drawn sample to the RRT.
        end
    else
        next
    end
    if There is a valid path between start and goal in RRT
    then
        publish the path return
    end
end
```

Algorithm 3: Dual Arm Motion Planning algorithm

[3] http://robotik.dfki-bremen.de/en/startpage.html

A. Pre-Scripted vs Constraint Task

In order to perform some dual arm operations, which requires imposing constraints (for example carrying a glass of water while keeping the orientation or opening a drawer with both arms) we have examined two different approaches:

- **Pre-scripted planning**
- **Constrained based planning**

In the pre-scripted approach, at first a Cartesian path consisting of way points (for example a half circle in the case of rotating a hand wheel or a straight line in the case of opening a drawer which represent the end effector path) is being calculated. Then by calling the IK solver, the corresponding joint values are retrieved. Generalizing tasks with pre-scripted approaches requires calculating and rewriting new way points for new tasks which limits the functionality of the code. In the constraint based approach, after specifying the start and the goal pose, we can define constraints on X,Y,Z axis and orientation (roll, pitch, yaw) of the object frame. By changing the values of the parameters, it is possible to reuse the code for different task.

B. Constraint Based Dual Arm Motion Planning Algorithm

The algorithm for dual arm motion planning under constraints is similar to the approach that was introduced in section IV, except that the kienmatic reachability is checked for both arms.

VI. Dynamic Environment

During a motion planning task, the robot senses the environment via its sensors and after finding a path, it executes the trajectory. In many situations, the state of the environment during the execution of the trajectory might change, therefore following the computed trajectory might cause collision. Various approaches have been proposed for modelling obstacles in dynamic environments (completely unpredictable obstacles, partial predictable obstacle) with different strategies for planning (probabilistic models, bounded uncertainty models, game theoretic models and dynamic replanning) [33]. In a methd called "Time-bounded lattice for planning in dynamic environments" [34], the algorithm tries to predict the position of dynamic obstacles by modelling their trajectories. Then it uses a time-bounded lattice (a short-term planning in time with longterm planning without time) for planning and obstacle avoidance. In some problems, the pose of the obstacles might be dependent on the pose of the robot (i.e non cooperative game with two players). For solving such problems, models from game theory have been used. In [35], the author solved the the problem of capturing an omnidirectional evader using a DDR (differential drive robot) by employing such models. The algorithms in [36] and [37] try to compute an appropriate velocity for the robot such that it dodges obstacles in a short time step. The D^* [11] algorithm can handle the dynamic obstacles found during the execution of the trajectory so it can be used in unpredictable environments. The algorithm in [38], combines replanning algorithms with optimization-based planning to handle dynamic obstacles. The algorithm utilizes the parallelization on multi GPUs to optimizes multiple trajectories to generate a high-quality path. Here in this work since our main focus was not planning in dynamic environments, we have used a simple yet workable approach. The planner checks the validity of each state in the trajectory during execution of that trajectory. If some states are broken, the planner will try to find a collision free path right before and after the broken state (replanning only over the invalid states and not whole the path) and update the trajectory.

Input : Robot model, motion planning request,robot environment point cloud, initial trajectory
Output: Updated collision free trajectory
while *execution of trajectory is not finished* **do**
  read the trajectory from the planner obtain the point cloud from sensors, create OctoMap from point cloud update robot collision checker with acquired OctoMap **for** *every state in the trajectory* **do**
    check the validity of state and mark down invalid states
  end
  for *every invalid state in the trajectory* **do**
    start state ← state right before the invalid state goal state ← state right after invalid state call the planner for planning between start and goal state **if** *planning succeed* **then**
      update initial trajectory with newly computed trajectory
    else
      cancel the execution of trajectory return false
    end
  end
end

Algorithm 4: Replanning in dynamic environment algorithms.

VII. Evaluation and Experiments

To evaluate the robustness and performance of our planner, we have tested the planner on several robots with 6, 7 and 11 DOF for single arm and dual arm operation. We also designed several scenarios and experiments to check the success rate and recorded the required time for planning in a simulation environment. These experiment have been done on a PC with 64-bit architecture equipped with Intel Core i7-3770 CPU (3.40GHz) and 7.5 GiB of ram.

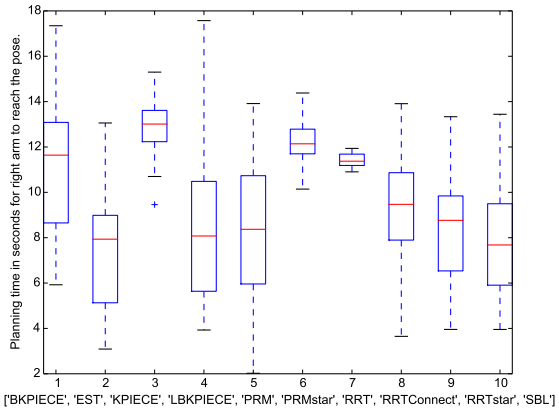
A. AILA

The robot AILA [39], is a dual arm mobile robot developed at DFKI [4] research center with a total of 70 degrees of freedom: 2 x 7-DOF arms, 4-DOF torso, 2-DOF head, 12-DOF mobile base, 2 x 18-DOF hands. We defined several planning groups including right arm, left arm and right arm plus foot, knees and torso. The combination of the two latter groups, make a bigger planning group with the form of a tree which gives more flexibility for dual arm operations. In the following we describe the experiments and outcomes.

1) Bookshelf desk: In this task the planner will try to place the right arm of the AILA in the middle of a bookshelf desk. We examined different planners with the maximum time limit of 10 seconds for planning. Note that required time for the whole task is more than 10 seconds (i.e path smoothing, interpolation, IK solving). Figures 1 and 2 show the box plot and success rate of different planners.

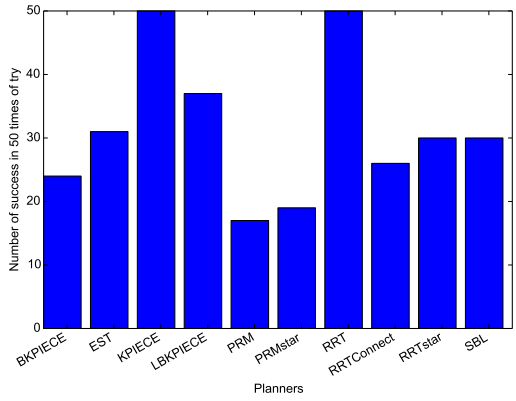
2) Amazon Picking Challenge : For this task, first robot places itself in front of the shelf by using its mobile base, such that the base link of the robot is in the middle of the shelf columns that needs to be reached. Then the planner will try to find a collision free path to place the right arm in the shelves from A to L. For this task we haven't set any time limits for planning. Figures 4 and 3 show the the box plot of required time for finding collision free trajectory from the start pose to the shelves B,E,H,K.

[4]http://robotik.dfki-bremen.de/en/startpage.html



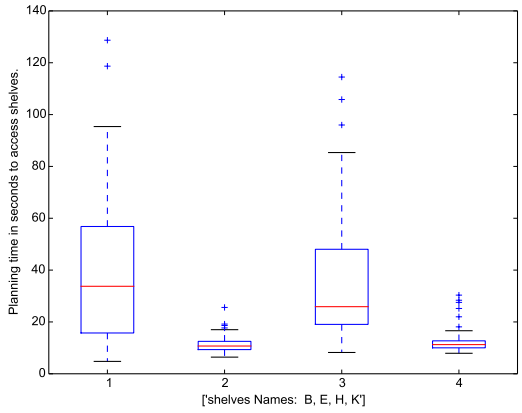
(a)

Fig. 1: The box plot of planning time for accessing middle point of a book shelf with different planners.



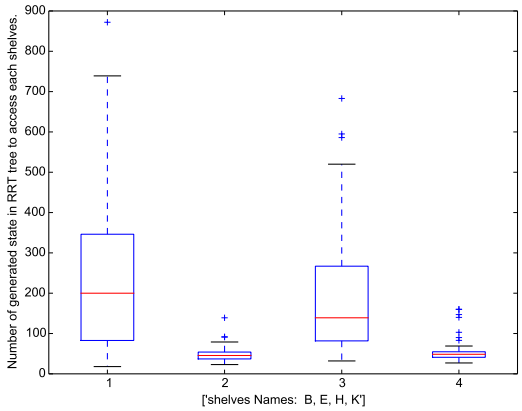
(a)

Fig. 2: Success rate of different planners to access book shelf within 50 times of trial.



(a)

Fig. 3: The box plot of planning time for the shelves B,E,H,K of Amazon picking challenge.



(a)

Fig. 4: The box plot of total number of generated state in RRT for accessing each shelf of Amazon picking challenge.

B. ARTEMIS

The ARTEMIS[5] robot developed by DFKI, consists of a rover with 6 wheels and 6-DOF manipulator mounted on top of the rover. We examined the task of carrying an object while keeping the orientation of the EFF from one side of robot to the other side. The task achieved by using the algorithm explained in section IV. Figure 5 illustrates ARTEMIS arm carrying an object while keeping the orientation.

C. SemProM

The dual arm SemProM-Robot[6] which is the predecessor of AILA, designed for controlling, transportation and gripping of objects. The robot consists of two 7-DOF schunk lwa, a flexible head and a base frame. Here we completed the task of opening a hand wheel and opening (pulling) a drawer by imposing different constraints and the algorithm explained earlier in section IV. For example, in order to open a hand wheel, it should rotate $\pi/2$ over its **x** axis. By imposing the constraints in equation 1 , the robot was able to open a hand wheel.

For the task of opening a drawer, since the only desired movement is in the **y** axis, simply by setting the constraints on **y** and no rotations the task could be accomplished. equation 2 represents such a constraint.

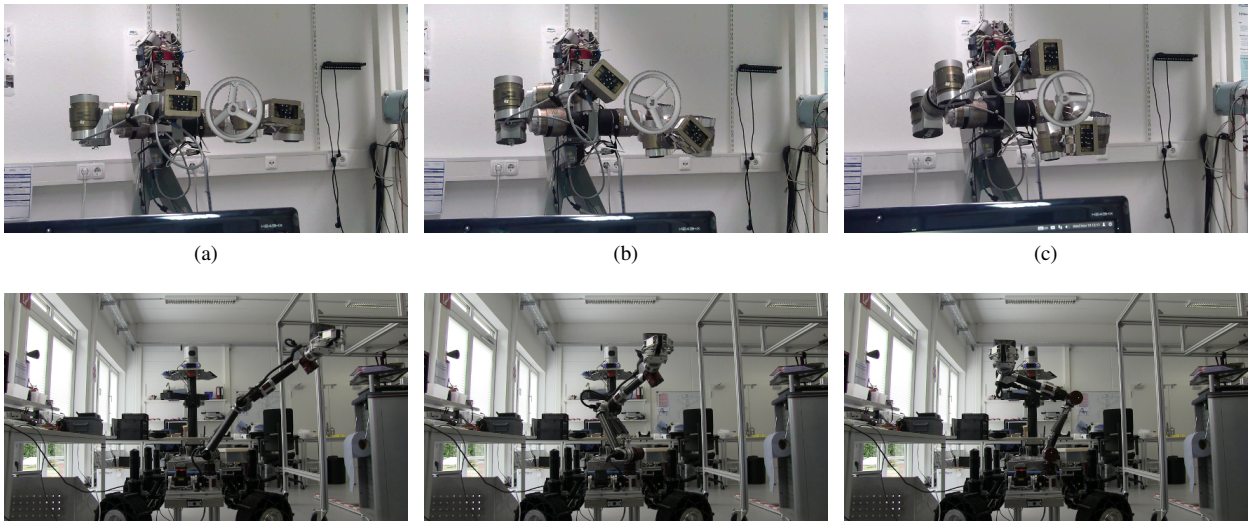
$$\begin{pmatrix} x:0.70 \\ y:0.00 \\ z:0.00 \\ roll:\pi/2 \\ pitch:0.0 \\ yaw:0.0 \end{pmatrix}, \begin{pmatrix} x:0.70 \\ y:0.00 \\ z:0.00 \\ roll:0.0 \\ pitch:0.0 \\ yaw:0.0 \end{pmatrix} \quad (1)$$

$$\begin{pmatrix} x:0.0 \\ y:1.00 \\ z:0.00 \\ roll:0.0 \\ pitch:0.0 \\ yaw:0.0 \end{pmatrix}, \begin{pmatrix} x:0.0 \\ y:-1.00 \\ z:0.00 \\ roll:0.0 \\ pitch:0.0 \\ yaw:0.0 \end{pmatrix} \quad (2)$$

Figure 5 illustrates opening a hand wheel by imposing constraints

[5]http://robotik.dfki-bremen.de/de/forschung/robotersysteme/artemis.html.

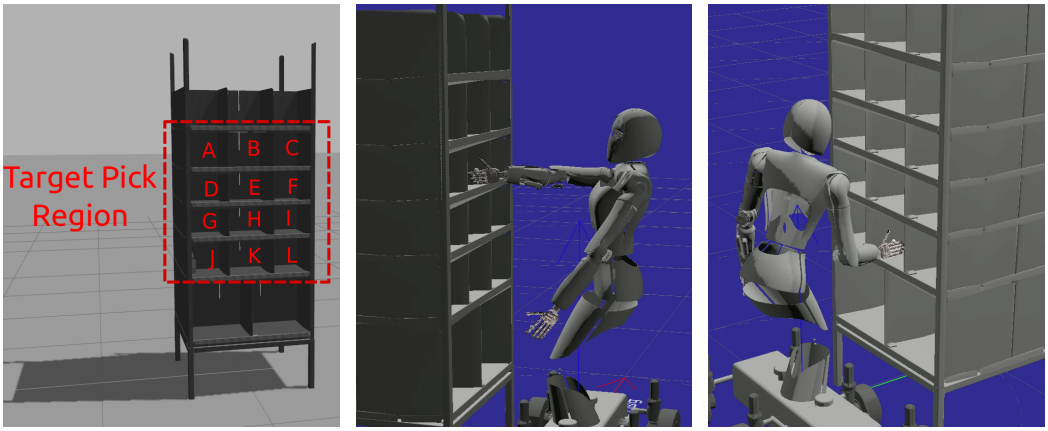
[6]http://robotik.dfki-bremen.de/en/research/robot-systems/mr-semprom-1.html.



(a) (b) (c)

(d) (e) (f)

Fig. 5: The robot SemProM (on the top) rotates a hand wheel, the robot ARTEMIS (on the bottom) carries an object while keeping the orientation. The tasked achieved by employing the algorithm 2 and the constraints from equation 1 and 2.



(a) (b) (c)

Fig. 6: The robot AILA access the shelves B,E,H,K form "Amazon Picking Challenge Shelf" respectively.

with **Mr Semprom**.

VIII. Analysis and Conclusion

One common parameter in all RRT based planers is "range". Range is the threshold at which, newly drawn samples with smaller distance value than that (distance to the closest point in the tree) will be added to the tree. Selecting big values for range make the planning time shorter but often creates jerky motions. Selecting smaller values, will generate more points in the tree and make the planning time longer. Also a collision might occur in the space between way points and extra collision checking is required. Depending on the task, we set different values for this parameter (maximum 0.2, minimum 0.02). In bookshelf desk experiment introduced in VII-A.1, although RRT planner requires more time in comparison with other planners but it provides better success rate. Since the environment of bookshelf desk experiment was similar to Amazon Picking Challenge Shelf (cluttered environment, limited workspace), therefore we used the RRT planner as the main algorithm for the Amazon picking challenge. In Amazon picking challenge, accessing the shelf E and K was easier for the robot (short time for planning, small number of generated states). This was mainly caused due to specific kinematic model of the robot, joint limits and collisions around the robot.

For a continues path in Cartesian space the planner may not generate continues trajectories in joint space. For instance a joint might reach the joint limits and as a result for the next way point in Cartesian space, it has to jump to a point which is not along the previous points. This might violate the consistency of imposed constraints. To overcome this issue, For start and goal pose we pick the IK solution that has maximum distances form the joint limits (for all the joints) and for the way points in the path we pick the IK solution that has shortest distance to its predecessor.

We also achieved to perform dual arm operation such as rotating hand-wheel or opening drawer within time limit of 10 to 20 (depending on the robot and imposed constraints). So far by using RRT based planners, we managed to find a collision free path between the start and the goal pose. This path usually doesn't look

natural and contains redundant movements. Furthermore we might be interested to get a safe trajectory (i.e by keeping the manipulator links far away as much as possible from obstacles or keeping EFF away from obstacle in the case of carrying a hazardous or fragile object) or we might be interested to minimize the planned path or combination of both (shortest path while keeping distances from obstacles). This can be achieved by using RRT based optimizing planners and defining a proper objective function to satisfy the problem conditions. These objective functions might need specific approaches for modelling the obstacles in the environment (such as inflation map or euclidean distance transform) rather than just dividing the space into free and collision zones. For our future work, we use optimizing planners and we define new metric for checking the safety of the trajectory while we try to minimize the planned path.

REFERENCES

[1] D. Hsu, *Randomized Single-Query Motion Planning In Expansive Spaces*. PhD thesis, Department of Computer Science, Stanford University, 2000.

[2] J. H. Reif, "Complexity of the movers problem and generalizations extended abstract," in *Proceedings of the 20th Annual IEEE Conference on Foundations of Computer Science*, pp. 421–427, 1979.

[3] "Open motion planning library:a primer." `ompl.kavrakilab.org/OMPL_Primer.pdf/`, 2006.

[4] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[5] J.-C. Latombe, *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers, 1991.

[6] F. P. Preparata, M. I. Shamos, and F. P. Preparata, *Computational geometry: an introduction*, vol. 5. Springer-Verlag New York, 1985.

[7] R. Seidel, "A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons," *Computational Geometry*, vol. 1, no. 1, pp. 51–64, 1991.

[8] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[9] S. LaValle, *Planning algorithms*. Cambridge New York: Cambridge University Press, 2006.

[10] B. Cohen, S. Chitta, and M. Likhachev, "Search-based planning for manipulation with motion primitives," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 2902–2908, 2010.

[11] J. Van Den Berg, D. Ferguson, and J. Kuffner, "Anytime path planning and replanning in dynamic environments," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2366–2371, IEEE, 2006.

[12] M. Likhachev and A. Stentz, "R* search," *Lab Papers (GRASP)*, p. 23, 2008.

[13] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2, pp. 500–505, Mar 1985.

[14] J. R. Andrews, *Impedance control as a framework for implementing obstacle avoidance in a manipulator*. 1983.

[15] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pp. 1398–1404, IEEE, 1991.

[16] H. Safadi, "Local path planning using virtual potential field," *McGill University School of Computer Science, Tech. Rep*, 2007.

[17] N. Ratliff, M. Zucker, J. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pp. 489–494, 2009.

[18] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 4569–4574, 2011.

[19] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 72–82, December 2012. `http://ompl.kavrakilab.org`.

[20] I. Sucan, M. Moll, and L. Kavraki, "The open motion planning library," *Robotics Automation Magazine, IEEE*, vol. 19, pp. 72–82, Dec 2012.

[21] I. A. Sucan and S. Chitta, "Moveit!." `http://moveit.ros.org/`.

[22] R. Smits, H. Bruyninckx, and E. Aertbeliën, "Kdl: Kinematics and dynamics library," *Avaliable: http://www. orocos. org/kdl*, 2011.

[23] R. Diankov and J. Kuffner, "Openrave: A planning architecture for autonomous robotics," Tech. Rep. CMU-RI-TR-08-34, Robotics Institute, Pittsburgh, PA, July 2008.

[24] "Openrave, ik fast module, openrave documentation." `http://openrave.org/docs/latest_stable/openravepy/ikfast/#ikfast-the-robot-kinematics-compiler`, Cited January 2014.

[25] M. Stilman, "Task constrained motion planning in robot joint space," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 3074–3081, IEEE, 2007.

[26] Y. Koga, K. Kondo, J. Kuffner, and J.-C. Latombe, "Planning motions with intentions," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, (New York, NY, USA), pp. 395–408, ACM, 1994.

[27] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 1874–1879, IEEE, 2006.

[28] E. Drumwright and V. Ng-Thow-Hing, "Toward interactive reaching in static environments for humanoid robots," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 846–851, IEEE, 2006.

[29] D. Berenson, S. S. Srinivasa, and J. Kuffner, "Task space regions: A framework for pose-constrained manipulation planning," *The International Journal of Robotics Research*, p. 0278364910396389, 2011.

[30] D. Berenson and S. S. Srinivasa, "Probabilistically complete planning with end-effector pose constraints," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 2724–2730, IEEE, 2010.

[31] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic, "Dual arm manipulationa survey," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1340 – 1353, 2012.

[32] D. Surdilovic, Y. Yakut, T.-M. Nguyen, X. B. Pham, A. Vick, and R. Martin, "Compliance control with dual-arm humanoid robots: Design, planning and programming," in *Humanoid Robots (Humanoids), 2010 10th IEEE-RAS International Conference on*, pp. 275–281, 2010.

[33] S. M. LaValle, "Steven m. lavalle." `http://msl.cs.uiuc.edu/~lavalle/icra12/`, May 2012.

[34] A. Kushleyev and M. Likhachev, "Time-bounded lattice for efficient planning in dynamic environments," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pp. 1662–1668, IEEE, 2009.

[35] U. Ruiz and R. Murrieta-Cid, "A homicidal differential drive robot," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3218–3225, IEEE, 2012.

[36] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[37] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," *The International Journal of Robotics Research*, vol. 17, no. 7, pp. 760–772, 1998.

[38] C. Park, J. Pan, and D. Manocha, "Real-time optimization-based planning in dynamic environments using gpus," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 4090–4097, IEEE, 2013.

[39] J. Lemburg, J. de Gea Fernandez, M. Eich, D. Mronga, P. Kampmann, A. Vogt, A. Aggarwal, Y. Shi, and F. Kirchner, "Aila - design of an autonomous mobile dual-arm robot," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5147–5153, May 2011.