Trajectory planning in Dynamics Environment : Application for Haptic Perception in Safe HumanRobot Interaction

A Gutierrez, V K Guda, S Mugisha, C Chevallereau, Damien Chablat

▶ **To cite this version:**

A Gutierrez, V K Guda, S Mugisha, C Chevallereau, Damien Chablat. Trajectory planning in Dynamics Environment : Application for Haptic Perception in Safe HumanRobot Interaction. Vincent G. Duffy. Digital Human Modeling and Applications in Health, Safety, Ergonomics and Risk Management, Springer, Cham, pp.313-328, 2022, 10.1007/978-3-031-06018-2_22 . hal-03583589

HAL Id: hal-03583589
https://hal.science/hal-03583589v1

Submitted on 22 Feb 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Trajectory planning in Dynamics Environment: Application for Haptic Perception in Safe Human-Robot Interaction

A. Gutierrez, V.K Guda, S. Mugisha, C. Chevallereau, D. Chablat

Laboratoire des Sciences du Numérique de Nantes, UMR CNRS 6004, Nantes 44300, France
damien.chablat@cnrs.fr

Abstract. In a human-robot interaction system, the most important thing to consider is the safety of the user. This must be guaranteed in order to implement a reliable system. The main objective of this paper is to generate a safe motion scheme that takes into account the obstacles present in a virtual reality (VR) environment. The work is developed using the MoveIt software in ROS to control an industrial robot UR5. Thanks to this, we will be able to set up the planning group, which is realised by the UR5 robot with a 6-sided prop and the base of the manipulator, in order to plan feasible trajectories that it will be able to execute in the environment. The latter is based on the interior of a vehicle, containing a user (which would be the user in this case) for which the configuration will also be made to be taken into account in the system. To do this, we first investigated the software's capabilities and options for path planning, as well as the different ways to execute the movements. We also compared the different trajectory planning algorithms that the software is capable of using in order to determine which one is best suited for the task. Finally, we proposed different mobility schemes to be executed by the robot depending on the situation it is facing. The first one is used when the robot has to plan trajectories in a safe space, where the only obstacle to avoid is the user's workspace. The second one is used when the robot has to interact with the user, where a mannequin model represents the user's position as a function of time, which is the one to be avoided.

Keywords: Trajectory planning, Human safety, Haptic interface, Intermittent contact interface

1 Introduction

In human-robot interaction systems, knowing how to compute a path for the robot to follow, while taking into account the human position, is a crucial task to ensure the safety of the individuals around the robot. This is where path and trajectory planning plays its role in the field of robotics, where achieving real-time behaviour is one of the most challenging problems to solve. The result is

a constant demand for research into more complex and efficient algorithms that allow robots to perform tasks at higher speeds, reducing the time they need to complete them, resulting in increased efficiency. But this also comes at a cost: to achieve higher speeds and shorter times, robot actuators must work under more demanding conditions that can shorten their overall life or even damage their structure. High operating speeds can also affect the accuracy and repeatability of manipulators. Therefore, it is important to generate well-defined trajectories that can be executed at high speeds without generating high accelerations (to avoid robot wear or end effector vibrations during stopping). Path planning is the generation of a geometrical path from an initial point to an end point and the calculation of the crossing points between them. Each point of the generated trajectory is supposed to be reached by the robot end effector through a specific movement. When the robot is supposed to interact with a human, its velocity and acceleration must be zero at the end of the trajectory. Another important element to take into account is the environment in which the task or the movement is going to be performed. This is what allows the system to identify the robot's environment and the colliding objects that might be present, thus determining the areas in which the robot must be constrained or limited to ensure the safety of the user.

The Lobbybot project is a project that allows interaction between a user and a cobot. These interactions allow for the creation of a touch-sensitive interface or intermitant contact interface (ICI). The scenario used allows the user to be inside a car with the possibility to interact with its environment by getting a sensory feedback of the different surfaces thanks to a 6 faces prop providing the different textures. Due to the immersion of the user via a VR headset, the system must ensure the safety of the user, as he cannot see the location of the robot. Therefore, it is necessary to implement trajectory planning techniques to be able to avoid unwanted interactions between the robot and the user. To do this, the system must take into account the obstacles present (environment or user). A virtual mannequin is modelled using data from the HTC Vive trackers which provide an estimate of the user's position, and will give the system a model to plan the movements. Thus, the goal of the LobbyBot project is to provide an immersive VR system that is safe for the user and gives them the ability to interact with the environment at different locations, providing a new level of interaction between VR environments and the real world.

2 State of the art

2.1 Intermittent contact interface

In the area of human-robot interaction and haptic perception, the ability to reproduce the sense of touch to appreciate different textures and motion sensations through the use of cobots has been addressed in [1], where a rotatable metaphorical accessory approach (ENTROPiA) has been proposed to provide an infinite surface haptic display, capable of providing different textures to render multiple infinite surfaces in VR (virtual reality). Studies in [2] [3] have focused

on the perception of stiffness, friction, and shape of tangible objects in VR using a wearable 2-DoF (degrees of freedom) tactical device on a finger to alter the user's sense of touch. In [4, 5], a 6-DoF cobot is used in a VR environment to simulate the interior of a car, where interaction between the robot and the user is expected just at specific, instantaneous points. This proposal is to use ICIs (Intermittent Contact Interfaces) [6] to minimise the amount of human-robot interactions to increase safety. In order to use the proposed implementations in this study in a real-time environment that involves human movement, it is important to ensure the safety of both the user and the robot to avoid potential collisions or accidents. This is where it is necessary to implement proper path and trajectory planning, in order to determine a feasible path to the desired goal, while avoiding interaction with the human until said goal is reached, generating a human-robot interaction just at the desired time.

2.2 Path Planning

Path planning refers to the calculation or generation of a geometric path, which connects an initial point to an end point, passing through intermediate via-points. These trajectories are intended to be followed by the end effector of a robot in order to execute a desired task or motion. This geometric calculation is based on the kinematic properties of the robot as well as its geometry (included in its workspace). In the simplest case, path planning is performed within static and known environments. However, this problem can also be generated for robotic systems subjected to kinematic constraints in a dynamic and unknown environment.

Path planning can be done using a previously known map. This is called global planning. This method is commonly used to determine the possible paths to follow to reach the final position. It is used in the case of a known and static environment, where the position of the obstacles does not change. This operation can be performed offline, as it is based on previously known information. In the case of dynamic environments, it is necessary to perform local path planning, which relies on sensors or any other type of interface providing data to obtain updated information about the robot's environment. This planning can only be done in real time, as it depends on the dynamic evolution of the environment. Figure 1 presents the main differences between local and global path planning [7, 8].

There have been multiple proposals on path planning algorithms over the years. In [9], one can find a review of the basics and workings of the most common algorithms most commonly found in the robotics literature. The main methods are the following:

- The Artificial Potential Fields (APF) approach [8] introduced by O. Khatib in 1985 and further developed by [10] [11].
- The Probabilistic Road-maps approach [7] consists in generating random nodes in the configuration space (C_{space}) in order to generate a grid (so called, the road-map).

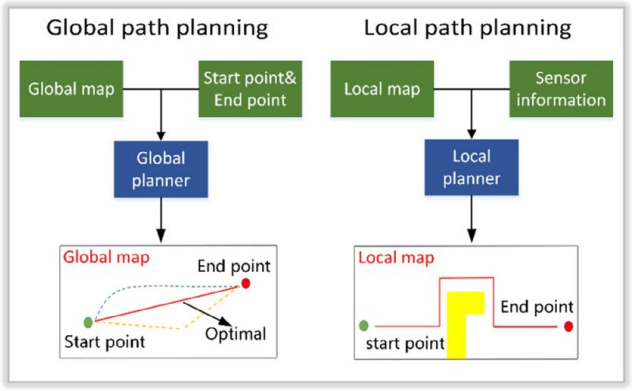

Fig. 1. Global [7] and local [8] path planning.

- The Cell Decomposition algorithms [12].
- The Rapidly Exploring Random Trees ($RRTs$) [13], introduced by S. LaValle in 2001 as an optimisation from the classical Random Trees algorithm.

2.3 Algorithm Comparison

In robotics, path planning is one of the most difficult tasks in real-time dynamic environments. Among the presented algorithms, APF and its variations offer a good adaptation to path planning in dynamically changing environments, where any obstacle entering the C_{space} generates a new repulsive field that can be taken into account to generate a new path. But the local minima problem requires the use of alternative algorithms to overcome it.

The case of PRM, it is well known for its ability to find a path without needing to explore the whole C_{space}, but it is also a graph based algorithm, which requires the use of shortest path method like A^*. It works well in static environments and can handle initial and final configuration changes, but if the objects in C_{space} change position, the connections between the nodes must be redone. Some alternatives propose to keep the previously generated nodes and recheck whether they belong to C_{free} or C_{obs}, then rebuild the graph based on this information and find a new path. This is also the case for cell decomposition methods, where the graph search has to be reconstructed again. Nevertheless, these methods have proven to be viable options in real time, capable of adapting to a dynamic environment.

Finally, regarding the RRT and RRT* methods and their alternatives, they are known to be good path planning methods, with the limitations that the generated trees are related to the initial configuration and have high computational demands. The proposal of the different alternatives allows to obtain very optimal real-time path planners. The limitations of this type of algorithms are that they require a large memory capacity, as the entire tree must be stored at all times,

and that they only work in bounded environments, with unbounded and long distance environments remaining a challenge.

2.4 Setup of the experimentation

In this section, we will present the tools used in the development of the project, such as the laboratory system, the software used, a description of the system environment as well as the laboratory setup.

System Architecture The architecture of MoveIt is based on two main nodes, the node *move_group* and the node *planning_scene*, which is part of the first one. The *move_group* node is responsible for obtaining the parameters, configuration and individual components of the robot model being used, in order to provide the user with services and actions to use on the robot.

Within the planners available in the *OMPL* library there are:

- PRM methods (PRM [7], PRM* [14], LazyPRM [15], LazyPRM* [15] [14]),
- RRT methods (RRT [13], RRT* [16], TRRT [17]), BiTRRT [18], LBTRRT [19], RRTConnect [20],
- Expansive Spacial Trees (EST) methods (EST [21], BiEST [21]).

Collision detection Collision checking in MoveIt is configured within a planning scene using the CollisionWorld object. Collision checking in MoveIt is performed using the Flexible Collision Library (FCL) package - MoveIt's main collision checking library.

Kinematics MoveIt uses a plugin infrastructure, specifically designed to allow users to write their own inverse kinematics algorithms. Direct kinematics and Jacobian search are built into the RobotState class itself. The default inverse kinematics plugin for MoveIt is configured using the KDL numerical solver [22] based on Jacobians. This plugin is automatically configured by the MoveIt configuration wizard.

ROS-Industrial ROS-Industrial is an open-source project that extends the advanced capabilities of ROS software to industrial hardware and applications. For this project, we used the ROS-Industrial-Universal-Robots metapackage [23], which provides and facilitates the main configuration files for the use of Universal Robots cobots in the ROS environment, providing the different descriptions of the robot, configuration files such as joint boundaries, UR kinematics, etc.. This package also facilitates the use of the robot in MoveIt, providing the setup for its use in simulation or in real implementations.

HTC Vive The HTC Vive is a motion tracking system that allows users to be immersed in a VR system [24]. It consists of trackers, which can attach to any rigid object, and work with the VR headset. The tracker creates a wireless connection between the object and the headset and then allows the user to represent the objects movements in a virtual world.

Laboratory Setup The laboratory setup consists of a UR5 robotic system and a car chair in a face-to-face configuration (Figure 2). The location and height of the robot was determined by [4] to be 75 cm above the floor. This position is optimal enough for the robot to reach all the interaction points that the system is interested in reaching. For the user, the VR headset and trackers are attached to the body (the humerus and palms), in order to obtain data and locate the user's location in the VR environment (Figure 3).

Fig. 2. Laboratory Setup

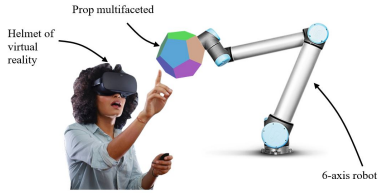

Fig. 3. Conceptual scheme of the experimental platform

3 Selection of the optimal trajectory planning and its application

We present the setup associated with the choice of the optimal trajectory generator available within the MoveIt software and its application for the LobbyBot project.

3.1 MoveIt Setup

The installation of MoveIt consisted of configuring and defining the planning group, as well as making it compatible to work in Gazebo. The start-up phase was very important to analyse the behaviour of the different movement alternatives found in the MoveIt API. For this, it was important to configure the simulation environment in Gazebo so that we could test without compromising the real robot.

3.2 Planning group

The planning_group is defined as the group of elements that make up the entire robotic system. These are the UR5 robot, the 6-faced prop and the robot support. These three elements are the ones that the trajectory planning algorithms must consider in order for them to avoid any collision state existing with one of these elements. The robot support was modelled to match the size of the real system that was optimally defined [4]. For the configuration of the plannig_group, MoveIt has an integrated graphical interface to create all the configuration files related to the kinematics, controllers, Semantic Robot Description Format (SRDF) and other files for the usage of the robot in ROS. This interface is called *MoveIt Setup Assistant*. The MoveIt Setup Assistant creates all the mentioned files based on the robot description given to it, in this case the UR5 robot description files provided by [23] where taken and modified to include the robot support (included in the URDF definition of the robot) and also the mesh file for the prop.

3.3 User's Model

To model the user, a mannequin was defined in a URDF robot model. The main torso of the model is fixed, while the arms are structured as a serial robot with seven revolute joints, where the first three constitute the shoulder, the fourth joint represents the elbow, and the last three revolute joints represent the wrist of the arm. In the model, two small dots have been created in the humerus and palm links, which represent the location of the sensors in the user, as shown in Figure 4. Regarding the movement of the mannequin model, a kinematic model has been developed in parallel to this project in [25], where the connection between the sensor data and the model is defined. This will allow the system to recognise the user's movements and represent it in the simulation 4.

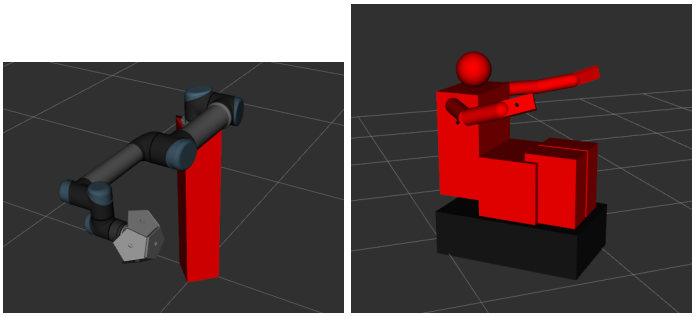

Fig. 4. Planning group and mannequin model of the user

3.4 Motions

To generate collision-free trajectories, the different algorithms implemented in the MoveIt API have been analysed. All the tasks related to planning group movement are handled by the move_group class. By specifying the planning group we want to consider, we are able to use all the different functions that the class offers for it, such as getting information about the current values of the joints, the target, configuring the planning algorithm we intend to use, and performing the planning and execution of the movements in the environment.

Types of movement The move_group class has the ability to perform path planning through different types of movements. These options can be chosen according to the nature of the task. For example, we can define a given pose in workspace or a desired joint value as the goal. Given the nature of the system, we will work with joint value goals, as we hope to achieve the different points in a specific configuration that provides a higher level of safety to the user (elbow up configuration for the UR5).

Another important feature is the ability to specify whether one wants to achieve each of the requested objectives or not. As the implementation will receive constantly changing goals, the best implementation is to plan and move towards said goal by allowing the system to replan if the goal changes, meaning that we do not need to reach the initial goal. To do this, the move_group class relies on the $move\_group.execute(my\_plan)$ function to strictly reach the goal and on the $move\_group.asyncExecute(my\_plan)$ function to execute the planned path with the possibility of re-planning during this execution.

In Figure 5, two trajectories are calculated from an initial configuration, to an intermediate goal, and then to a final goal. In this case, by using the function $move\_group.execute(my\_plan)$, we ensure that the robot will completely execute each of the trajectories and achieve both goals. This is illustrated in Figure 5, where the speeds drop to zero as the robot comes to a stop.

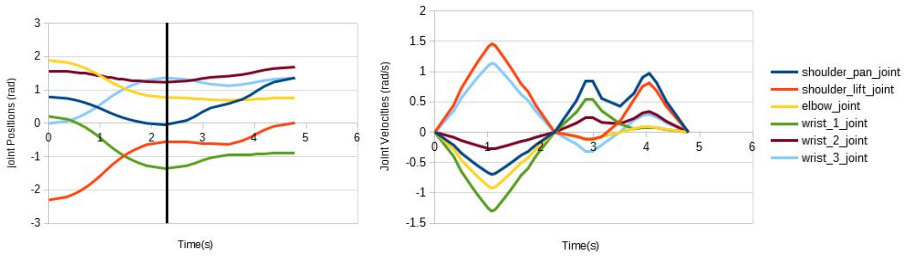

Fig. 5. Planned paths using $move\_group.execute(my\_plan)$: Back-to-back plans (left) and Velocities for both plans (right).

In the case of figure 6, we have calculated the same two trajectories as before, but using the function $move\_group.asyncExecute(my\_plane)$, which allows

replanning during the execution of the first plane. In this case, in figure 6(a), we can see the two plans one after the other, while in figure 6(b), we show the representation of the segment that was not executed from the first plan, because a replanning scenario was set up. In this case, the current positions of the first plan were taken as the initial positions for the second plan, resulting in Figure 6(c), showing the two plans that were executed.

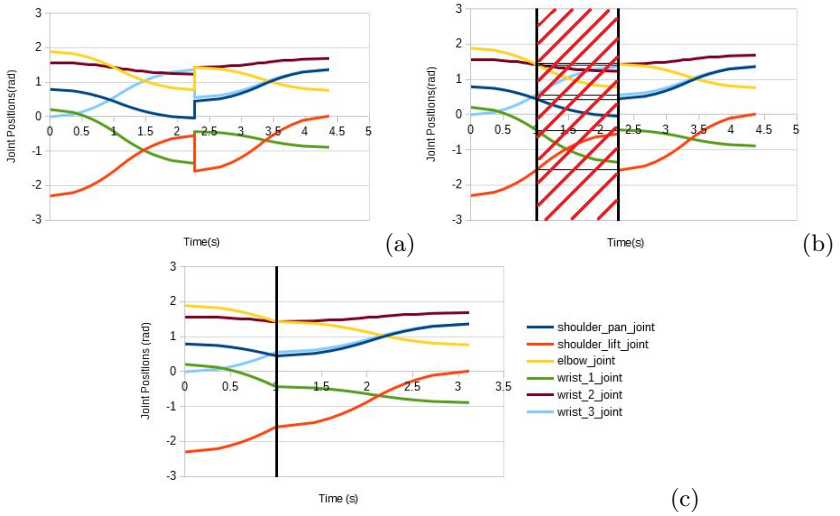

Fig. 6. Re-planned paths using *move_group.asyncExecute*(*my_plan*)
(a) Back-to-back plans. (b) Segment not executed due to re-planning. (c) Final executed plan.

Algorithm Selection Another parameter to select was the planning algorithm that best suited the task. As mentioned earlier, MoveIt has several built-in path planning algorithms that can be used. In order to determine the best option, we went through all the available options and performed a planning task to a desired target configuration, measuring the time required for each algorithm and recording the data. We ran each of the 12 available planning algorithms five times through nine different paths. We then took the average time it took them to find a solution, to simplify the trajectory (only for the algorithms that had this feature) and calculated the total average time. Using this data, we were able to select the algorithms that performed best with the shortest planning times (Figure 7).

After performing these calculations, given the large difference in planning times for some of the algorithms, we select the six best algorithms to compare them on 12 trajectories (Figure 8).

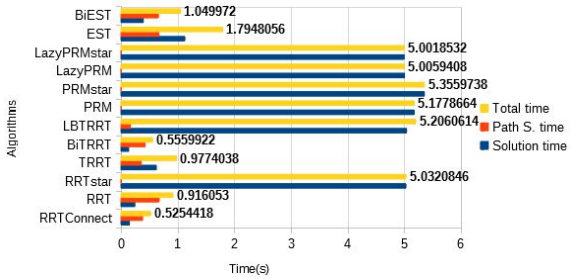

Fig. 7. Comparison of all planning algorithm's times for nine trajectories

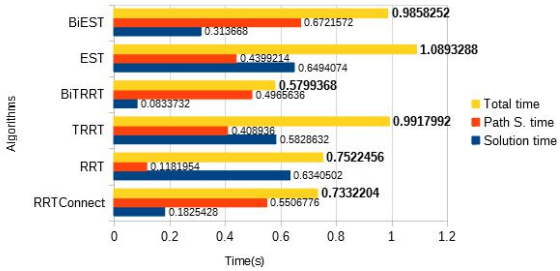

Fig. 8. Comparison of best planning algorithm's times for 12 trajectories

Another analysis that allowed us to select the algorithm which behaved the best for the implementation, was to perform an analysis on the generated trajectories with each one of the algorithms for a fixed task. Based on the six best algorithms from the previous analysis as a starting constraint, we computed the average execution time and via-points number for a set of trajectories. The BiTRRT algorithm wins for both comparisons.

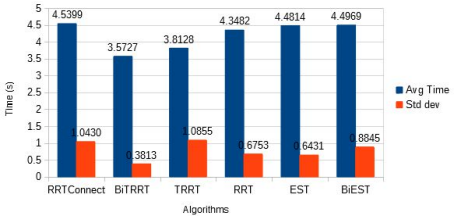

Fig. 9. Comparison of average execution times of the algorithms.

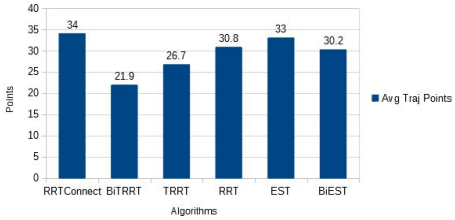

Fig. 10. Comparison of average amount of generated via-points.

This analysis was performed for the same trajectories as in the previous graphs for a total of ten iterations for each algorithm, but instead of considering only the computation time (Figure 9), we also took into account the amount of

via-points generated (Figure 10). Between each via-point, a linear interpolation is performed in the joint space. For theses trajectories, the mannequin was placed in its seat so that it was avoided in the calculation, in order to test each algorithm's ability to plan around it. This also allowed us to see how consistent the behaviour of each algorithm was.

Unity's Virtual Environment In parallel to the development of the project, and to better explain the developed implementation, it is important to specify how it will fit into the project. The system will receive a desired goal configuration which will be the q_{goal} for the planning algorithm, from the current q_{init} configuration. This goal selection is done in Unity by a *Point selection algorithm* which determines the interaction point the user intends to reach [26] (Figure 11).

3.5 Planning Scene

For the definition of the planning environment and scene, MoveIt has instances that allow the manipulation and monitoring of the scene to keep it up to date. These instances are :

- PlanningSceneInterface: Is responsible for adding and removing objects in the scene.
- PlanningSceneMonitor: Takes care of keeping track of the planning_scene in order to keep it updated.

The last of these instances is absolutely necessary to perform the collision check, as we need to ensure that the scene being processed is the last one available.

3.6 Mobility Schemes

Based on the Unity information, two different motion or mobility schemes and scenarios has been proposed depending on the nature of the task we want to achieve at the moment. One for which no interaction with the user is required, and another one for when it is. These two scenarios have their own environment to consider, presenting in general two different behaviours.

Movement outside user's workspace The first scenario is based on [26] where a distinction for velocity zones is made and where a plane divides the environment (space with the user and space where the user cannot go).Based on the same idea, we represented the effective working space of the mannequin as a sphere surrounding the model (Figure 12).

The mobility scheme consists of alternating between different *"Safe positions"*. These positions are so called because they are points out of reach of the user, which means that there is no need to constrain the robot's speeds. Therefore, the movement from one point to another just has to take into account the defined sphere, as we do not want to "collide" with it. Following this idea, we

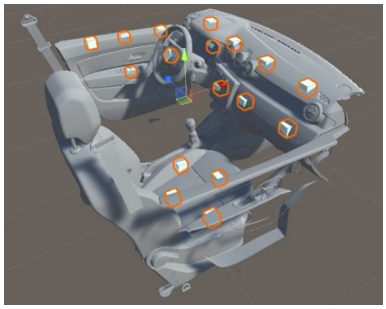

Fig. 11. Unity VR system and representation of interaction points

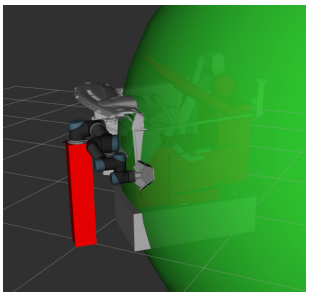

Fig. 12. Representation of the user's effective workspace as a sphere

have performed a calculation of all existing trajectories between the different *"Safe positions"* and stored them in a data file. This allows us to perform offline path planning, and then at runtime, depending on the initial and desired goal, we can access the pre-calculated paths to execute them directly, eliminating the computational time that would otherwise be required by performing online planning. The algorithm 1 allows the storage of the trajectory.

Then, the second part of the device consists of loading the pre-registered data and being able to use them on demand (Algorithm 2). We wait until we know the position we want to reach. Unlike [26], we have used a spherical surface here to divide the two areas of the space instead of a plane, as this allows greater flexibility for the planning group to consider more configurations when calculating the path between points. It also allows for more feasible trajectories for the robot.

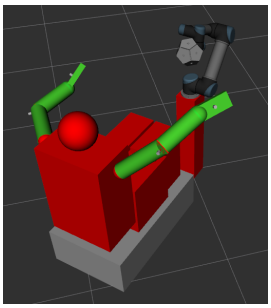

Fig. 13. Moving obstacles

Movement inside user's workspace The second mobility scheme has been proposed for the scenario where the robot end effector has to go inside the user's

Algorithm 1 Trajectory computation and storage

Require: Number of points no_p. A counter for start point i. A counter for final point j

```
start_name[no_p] ← {Store id of the points}
final_name[no_p] ← init_names[no_p] {Same id's as we are iterating through all points}
for i < 0 ; i < no_p ; i++ do
    for j < 0 ; j < no_p ; j++ do
        if i ≠ j then
            Ø ← Plan_and_Exec_to(points[i]) {Move to initial point of the plan}
            plan_array[i][j] ← Plan_and_Exec_to(points[j]) {Move to desired point and keep the planned trajectory}
        end if
    end for
end for
{Store all as a structured message}
for i < 0 ; i < no_p ; i++ do
    for j < 0 ; j < no_p ; j++ do
        if i ≠ j then
            init_pos_id ← start_name[i]
            goal_pos_id ← final_name[j]
            plan ← plan_array[i][j]
        end if
    end for
end for
```

workspace, which means that the movements have to take into account the user's model in order to avoid any collision with him. We also have to take into account that the speed of these movements must be limited, in order to ensure safety.

Unlike the first scheme, in this case the environment consists of moving obstacles, which requires constant updating of the scene and constant tracking of the objects in it (Figure 13). For this reason, we used the images of the mannequin model to obtain its current positions and orientations in order to track their movement and link it to the objects created in the scene.

We also need to be able to determine whether a computed plan will collide or not, which requires taking several aspects into account. First, based on the calculated path to the desired goal, we check whether the path remains valid during the execution of the plan. To do so, we check for all calculated via-points of the path, whether the respective configurations are currently colliding with any other object present in the scene. If there are no collisions, we continue the execution. In the case of a collision present in any of the remaining states of the path, we instruct the robot to stop the execution of the computed path and replan it based on the updated scene information.

To test our framework, we performed an initial trajectory planning. Then, during the execution, we created an obstacle. Then, by checking the validity of the trajectory, we are able to detect that an object is in collision with the

Algorithm 2 Trajectory upload and execution

Require: A desired frame to go to des_frame. A home pose $home$. Number of elements no_e. Initial positions $init\_pos\_id$. Goal positions $goal\_pos\_id$. Planned trajectories $plan$. A counter i.

for $i < 0$; $i < no_e$; $i++$ **do**
  {Extract the data from the file}
  $start\_name[i] \leftarrow init\_pos\_id[i]$
  $final\_name[i] \leftarrow init\_names[i]$ {Same id's as we are iterating through all points}
  $plan\_array[i] \leftarrow plan[i]$
end for
$\emptyset \leftarrow$ Plan_and_Exec_to($home$) {Move to home pose}
{Reference to home position as current}
$init\_frame \leftarrow$ "$home$"
$aux\_des\_frame \leftarrow$ "$home$"
while running **do**
  if $des\_frame == init\_frame$ **then**
    {The robot is in position.}
  else
    $aux\_des\_frame = des\_frame$ {Update the desired position}
    for $i < 0$; $i < no_e$; $i++$ **do**
      {Search in the list of plans the one that matches the init and final frames}
      if ($start\_name[i]$ == $init\_frame$) **and** ($final\_name[i]$ == aux_des_frame) **then**
        $execute(plan\_array[i])$
        $init\_frame \leftarrow aux\_des\_frame$
      end if
    end for
  end if
end while

planned trajectory. We then instruct the robot to stop the current execution and replan towards the same goal, taking into account the updated planning scene. This work is intended to be extrapolated to work according to the size of the mannequin. Thus, we can take into account the user moving in the environment as an obstacle to be avoided (Figure 14).

4 Conclusions

In this paper, we have presented motion generation algorithms that can be used by a cobot to create an intermittent contacts interface. A framework was presented including a UR5 cobot, ROS nodes, HTC Vive sensors and a car chair. Taking into account the objects present in the environment, a comparison of trajectory planning algorithms is presented. The selected algorithm is then used in two examples. An experimental validation is in progress and will be presented in the final version of the paper.

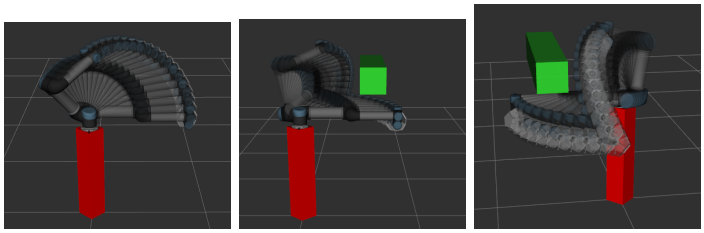

Fig. 14. Original planned path (left) and Re-planned path from detected collision (middle and right)

Acknowledgements

This research is part of LobbyBot: Novel encountered type haptic devices, a French research project funded by ANR.

References

1. V. Mercado, M. Marchal, and A. Lécuyer, "ENTROPiA: Towards Infinite Surface Haptic Displays in Virtual Reality Using Encountered-Type Rotating Props," *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 2019.
2. S. Salazar, C. Pacchierotti, X. de Tinguy, A. Macie, and M. Marchal, "Altering the Stiffness, Friction, and Shape Perception of Tangible Objects in Virtual Reality Using Wearable Haptics," *IEEE Transactions on Haptics*, 2020.
3. X. de Tinguy, C. Pacchierotti, M. Marchal, and A. Lécuyer, "Enhancing the Stiffness Perception of Tangible Objects in Mixed Reality Using Wearable Haptics," *IEEE Conference on Virtual Reality and 3D User Interfaces*, 2018.
4. V. Guda, D. Chablat, and C. Chevallereau, "Safety in a Human Robot Interactive: Application to Haptic Perception," *Chen J.Y.C., Fragomeni G. (eds) Virtual, Augmented and Mixed Reality. Design and Interaction. HCII 2020. Lecture Notes in Computer Science*, vol. 12190, pp. 562–574, 2020.
5. CLARTE, CNRS/LS2N, INRIA/Hybrid, and Renault, "Lobbybot project," https://www.lobbybot.fr/, 2020.
6. O. De la Cruz, F. Gosselin, W. Bachta, and G. Morel, "Contribution to the Design of a 6 DoF Contactless Sensor Intendedfor Intermittent Contact Haptic Interfaces," *2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM)e*, pp. 130–135, 2018.
7. L. Kavraki, P. Svetska, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces." *IEEE Transactions on Robotics and Automation*, vol. 12(4), pp. 566–580, 1996.
8. O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *IEEE International Conference on Robotics and Automation*, 1985.
9. A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path Planning and Trajectory Planning Algorithms: A General Overview," *Motion and Operation Planning of Robotic Systems*, vol. 29, 2015.
10. R. Volpe, "Real-time obstacle avoidance for manipulators and mobile robots," Ph.D. dissertation, The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA, 1990.

11. R. Volpe and P. Khosla, "Manipulator control with superquadric artificial potential functions: theory and experiments," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20(6), pp. 1423–1436, 1990.
12. N. Sleumer and N. Tschichold-gurman, "Exact Cell Decomposition of Arrangements used for Path Planning in Robotics," *Technical Report*, 2000.
13. S. LaValle and J. Kuffner, "Randomized Kinodynamic Planning," *The International Journal of Robotics Research.*, 2001.
14. S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.
15. R. Bohlin and L. Kavraki, "Path planning using lazy prm," *IEEE International Conference on Robotics and Automation*, pp. 521—528, 2000.
16. S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research.*, 2011.
17. L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Transactions on Robotics*, vol. 26, no. 4, August 2010.
18. D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the Transition-based RRT to Deal with Complex Cost Spaces," *Proceedings - IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4120–4125, 2013.
19. O. Salzman and D. Halperin, "Asymptotically near-optimal rrt for fast, high-quality motion planning," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 473–483, April 2016.
20. J. Kuffner and S. Lavalle, "Rrt-connect: An efficient approach to single-query path planning," *Proceedings of the 2000 IEEE International Conference on Robotics & Automation*, pp. 995–1001, April 2000.
21. D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces," *International Journal of Computational Geometry & Applications*, vol. 9, no. 4-5, pp. 495–512, 1999.
22. "Kinematics and dynamics library," http://wiki.ros.org/kdl, Online.
23. F. Messmer, K. Hawkins, S. Edwards, S. Glaser, and W. Meeussen, "Ros-indutrial-universal-robots," https://github.com/ros-industrial/universal_robot, Online.
24. HTC and Valve., "Htc vive," https://www.vive.com/fr/, Online.
25. V. Guda, D. Chablat, and C. Chevallereau, "Lobbybot Deliverable Report on Methogology for Motion Capture," Ecole Centrale de Nantes, Laboratoire des Sciences de Numérique de Nantes, Tech. Rep., 2021.
26. S. Mugisha, M. Zoppi, R. Molfino, V. Guda, C. Chevallereau, and D. Chablat, "Safe Collaboration Between Human And Robot In A Context Of Intermittent Haptique Interface," *Proceedings of the ASME 2021 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, 2021.