REPURPOSING A SAMPLING-BASED PLANNER FOR A SIX-DEGREE-OF-FREEDOM MANIPULATOR TO AVOID UNPREDICTABLE OBSTACLES

HAFIZ IMAN[1*], MD RAISUDDIN KHAN[1]

[1]*Department of Mechatronics Engineering, Kulliyah of Engineering, International Islamic University Malaysia, Kuala Lumpur, Malaysia*

**Corresponding author: hafiz.ghazman@gmail.com*

ABSTRACT: This paper presents the use of a sampling-based planner as a reactive planning scheme to avoid obstacle between a robotic arm and a moving obstacle. Based on a planner benchmark on an obstacle-ridden environment, rapidly-exploring random tree (RRT) planner has been used to populate the trajectories of the task space and map them into a configuration space using Newton-Raphson-based inverse kinematic solver. Two robot poses are defined in a cycle of back-and-forth motion; the initial and the goal poses. The robot repeatedly moves from the starting pose to the end pose via the midpoint pose. Each set of trajectories is unique. We define this unique solution within the context of the configuration space as a cycle space. We impose periodically occurring synthetic obstacle that moves in and out of the robot arm workspace defined in a simulated environment. Within the robot's workspace, the obstacle moves and cuts through the cycle space to emulate a dynamic environment. We also ran a benchmark on the available sampling planner in the OMPL library for static obstacle avoidance. Our benchmark shows that the RRT has the lowest time planning time at 0.031 s compared with other sampling-based planners available in the OMPL library, RRT implicitly avoids singularities within the cycle space, and reactively attempts to avoid synthetic moving objects nearing the robot hardware. Going forward, this research intends to further investigate on the use of RGB-D sensor and LiDAR to track moving obstacles while abiding by the task space commands described by the initial and goal poses.

ABSTRAK: Kertas kerja ini membentangkan penggunaan perancang berasaskan persampelan sebagai skim perancangan reaktif untuk mengelakkan halangan antara lengan robot dan halangan yang bergerak. Berdasarkan penanda aras perancang pada persekitaran yang dipenuhi halangan, perancang pokok rawak (RRT) penerokaan pantas telah digunakan untuk mengisi trajektori ruang tugas dan memetakannya ke dalam ruang konfigurasi menggunakan penyelesai kinematik songsang berasaskan Newton-Raphson. Dua pose robot ditakrifkan dalam kitaran gerakan bolak-balik; pose awal dan matlamat. Robot berulang kali bergerak dari pose permulaan ke pose akhir melalui pose titik tengah. Setiap set trajektori adalah unik. Kami mentakrifkan penyelesaian unik ini dalam konteks ruang konfigurasi sebagai ruang kitaran. Kami mengenakan halangan sintetik yang berlaku secara berkala yang bergerak masuk dan keluar dari ruang kerja lengan robot yang ditakrifkan dalam persekitaran simulasi. Dalam ruang kerja robot, halangan bergerak dan memotong ruang kitaran untuk meniru persekitaran yang dinamik. Kami juga menjalankan penanda aras pada perancang pensampelan yang tersedia dalam perpustakaan OMPL untuk mengelakkan halangan statik. Penanda aras kami menunjukkan bahawa RRT mempunyai masa perancangan masa terendah pada 0.031 s berbanding dengan perancang berasaskan pensampelan lain yang terdapat dalam perpustakaan OMPL, RRT secara tersirat mengelakkan singulariti dalam ruang kitaran, dan secara reaktif cuba mengelakkan objek bergerak sintetik yang menghampiri perkakasan robot. Melangkah ke hadapan, penyelidikan ini berhasrat untuk menyiasat lebih lanjut mengenai penggunaan penderia RGB-D dan LiDAR untuk mengesan halangan bergerak sambil mematuhi arahan ruang tugas yang diterangkan oleh pose awal dan matlamat.

keywords: *mechatronics, robot manipulator, planner, motion planning, dynamic environment,*

1 . INTRODUCTION

Robot manipulators such as industrial robots work well in repetitive and heavy tasks. They are high-performing, objective, and relentless at tasks that are too difficult to complete by an operator or a group of workers. However, given their rigid and massive construction, even a small-sized industrial robot imposes significant hazards on the people that work near it. Hence, recently, robot manipulators are more compliant and are designed to work with workers cooperatively without risking their safety [1]. Regardless, it is still an issue of hazard should a compliant or cooperative robot collide with a person working close to its workspace [2]. The collision also warrants expensive maintenance and repairs. An industrial robot system implements a certain degree of planning algorithm specifically for the movement of the manipulator.

A robot motion planner can provide a collision-free motion solution for a manipulator if a solution is defined as the collection of waypoints and trajectories that avoid collision between the robot and the obstacle. In the case of the traditional definition of industrial robots, the planning is global because the robot is enclosed and isolated in a workcell. A global planner takes in a set of initial and goal positions, $t \in \mathbb{R}^3$, or as set of initial and goal poses, $p \in \boldsymbol{SE(3)}$, as its input geinput s constraint-informed trajectory as intermediate waypoints for the robot to follow. However, a global planner is offline, which implies that the trajectories are set before the task commences. The global planner also assumes a static workspace. Any unplanned changes in the workspace over the global planning scheme, such as an unplanned introduction of a stationary object or a moving object into the robot workspace, renders the offline-planned trajectory outdated and, consequently, requires replanning. In the case of a compliant robot, unplanned changes are unavoidable.

Hence, it is imperative a compliant manipulator or cooperative manipulator must have an efficient online motion planner because replanning is computationally expensive and time-consuming; sampling-based planners are the pinnacle example of efficient planning [3]. Unfortunately, sampling-based planner trade completeness and optimality with efficiency where the planner may fail to provide a solution [4]. Also, if a solution exists under its metric space, the waypoints may not be the least cost path to a goal [5].

Regardless of lack of completeness and optimality, sampling-based planner excel at maintaining reasonable usage of computational resources that pave way to the near-online planning scheme. The sampling-based planners for robot motion are a family of planners that uses probabilistic approach to generate a graph structure encoding the free space and the robot configuration space. The samplings are stochastic, such that resampling will give a unique solution to the previous sampling. Most sampling-based planners are also tractable in higher-dimensional configurations and task space. However, sampling-based planners assume a static workspace.

This paper repurposes the use of sampling-based motion planning, the rapidly-exploring random tree motion planning, to address operation task in a dynamic environment in Euclidean space, $\mathbb{R}^3$. Our method leverages the efficiency and the computationally reserved sampling-based motion planning without needing to apply a purely reactive motion planning approach so that computational resources can be delegated to other tasks, i.e., motion-tracking, state estimation, mapping, localization, and motion control. In the following sections of this report, we will assume sampling planners to provide solutions in higher dimensional configuration space, which implicates a solution with a set of poses represented by the special Euclidean group, $\boldsymbol{SE(3)}$.

2 . RELATED WORK

Kavraki et. al (1998) is the first group of researchers that used probability model for sampling the configuration space for holonomic robot motion such as a manipulator robot [4]. The planner is called the probabilistic roadmap (PRM) motion planning. The algorithm constructs a graph structure to find a path between an initial pose to a goal pose in a two-dimensional configuration space, $n = 2$. Kavraki et. al (1998) also proved a more general solution for higher dimensional configuration space, $n > 2$. With graph structure, more than one path connects the initial pose to the goal pose. Therefore, PRM is a multi-query type planner.

Kunz et. al (2010) improved PRM by redefining the distance metric of a robot manipulator so that the robot can move around a moving obstacle in real-time. Their approach performs well in an uncluttered environment [6]. They also redefined the distance function of the PRM to address dynamic objects, such as a walking person, into a two-dimensional map. Although the configuration space of the manipulator is in $\mathbb{R}^3$, the map, constructed from a two-dimensional LiDAR scan, is in, $\mathbb{R}^2$.

In retrospect, the RRT was formulated for non-holonomic motion targeting problems addressed in diffential-constrained motion such as a car on a plane [5]. However, given the model of its metric space and consequently the configuration space, RRT are tractable for higher dimensional problem such as manipulator motion in 3D space [7]. RRT assumes a static environment but Wei & Ren [7] successfully changed the way RRT samples a robot metric space so that it is fast enough to react to a changing environment. Also, unlike PRM, RRT works well in a cluttered environment because of the randomized sampling on the robot configuration space in the metric space.

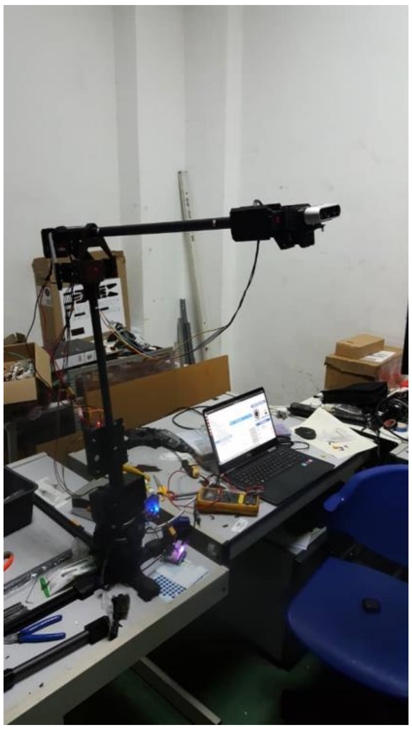
Apart from the works in [6] and [7], few have applied their planning algorithms on a robot manipulator despite both algorithms providing mathematical framework for planning for multi-body and multi-frame systems. In this paper, we will use the method demonstrated by [6] and [7] to design a moving obstacle avoidance algorithm with the implementation of the vanilla RRT to solve motion for a robot manipulator in three-dimensional space, $\mathbb{R}^3$. Our method implements the vanilla RRT where we do not represent the obstacle configuration space unlike in [7].

3 . FORMULATION AND ALGORITHMS

This paper will use the superscript notation to refer the control space and the subscript as the equivalent representation in the configuration space. For example, C^{ee}, refers to the control space of the end-effector where the controlling pipelines would take in $c^{ee} = (\theta_1, \ldots, \theta_6) \in C^{control}$, and the equivalent pose is in the configuration space, C_{ee}. Since, revolute joint topology is the 1-hypersphere, S^1, we will assume that, for the case of 6R robot, it's joints are limited to a certain range which makes, $c^{ee} \in \mathbb{R}^{6\times 1}$.

3.1. The Geometry of a Compliant Robotic Arm, r_mini

We prototype and build a 3D-printed robot called Richard Mini (*r_mini*, see Fig. 1) based on the conditioned addressed by Pieper [8], which entails three collated joints sharing the same cross point of their, *z-axes*, shown in Fig. 2.


(a) *r_mini* hardware assemblage

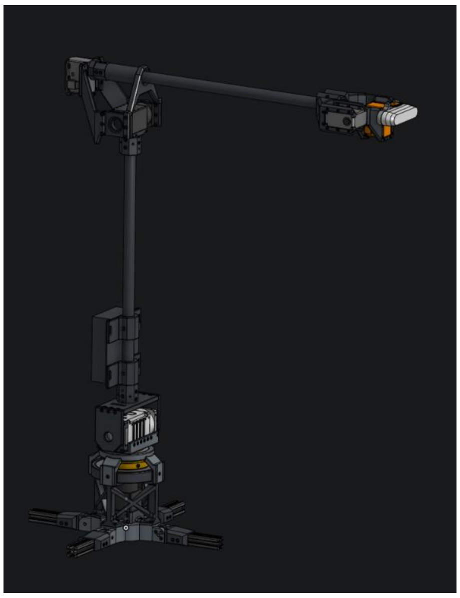

(b) r_mini Computer Aided Design (CAD) construction

Fig. 1: A 3D printed compliant manipulator, *r_mini*, designed to replicate a common industrial robot construction

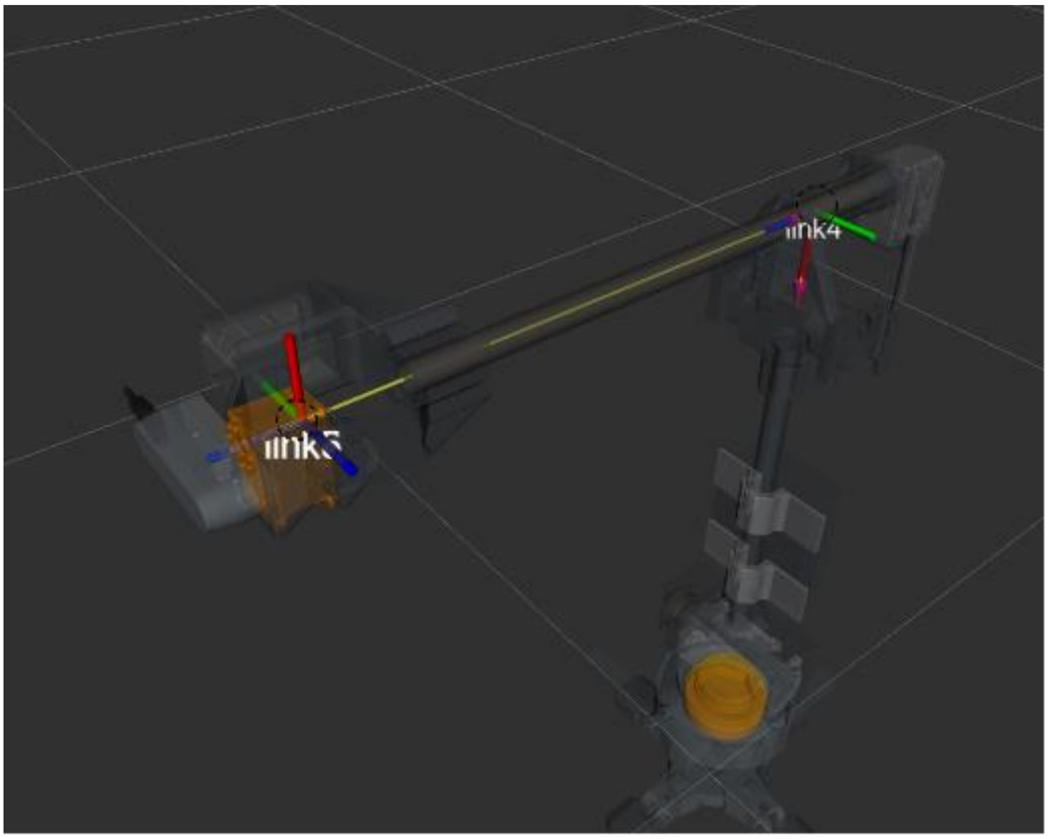


Fig. 2: *r_mini* wrist conforms to Pieper condition where axis of rotation for joint4, joint5, and joint6 share points of intercept. The dashed circles in the diagram refer to point of intercepts. Both points are valid frames for constructing the DH-table.

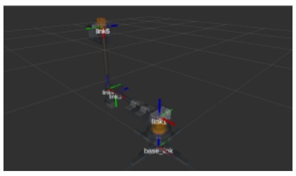
r_mini has six revolute axes, $(\theta_1, \cdots, \theta_6)$. The first three axes move the task space from one point to another representing translation vector, $t \in \mathbb{R}^3$. The last three axes of the manipulator rotate the task space representing the rotation operation about the task space frame, $R \in \mathbb{R}^{3\times3}$. Hence the complete transformation of the task space via the joint movement is represented by the homogenous transformation matrix, $T \in \boldsymbol{SE(3)}$, where $\boldsymbol{SE(3)}$ is homomorphic to $(R, \boldsymbol{t})$; $\boldsymbol{SE(3)} = \mathbb{R}^3 \times \boldsymbol{SO(3)}$. The matrix representation of the homogenous transformation is shown in Eq. (1).

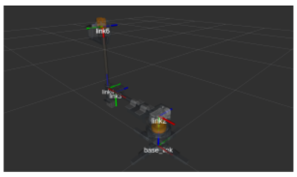
$$T = \left[\begin{array}{c:c} R & \boldsymbol{t} \\ \hdashline \boldsymbol{0} & 1 \end{array}\right] \quad (1)$$

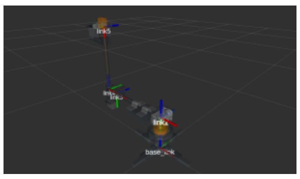
The kinematic models of the r_mini follows the Denavit-Hartenberg (DH) formulation [9]. The DH-parameters are shown in Table 1 and the visualization of these parameters in the form of frames transformation is shown in Fig. (3).

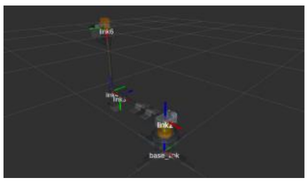
Table 1: DH-parameter table for *r_mini*

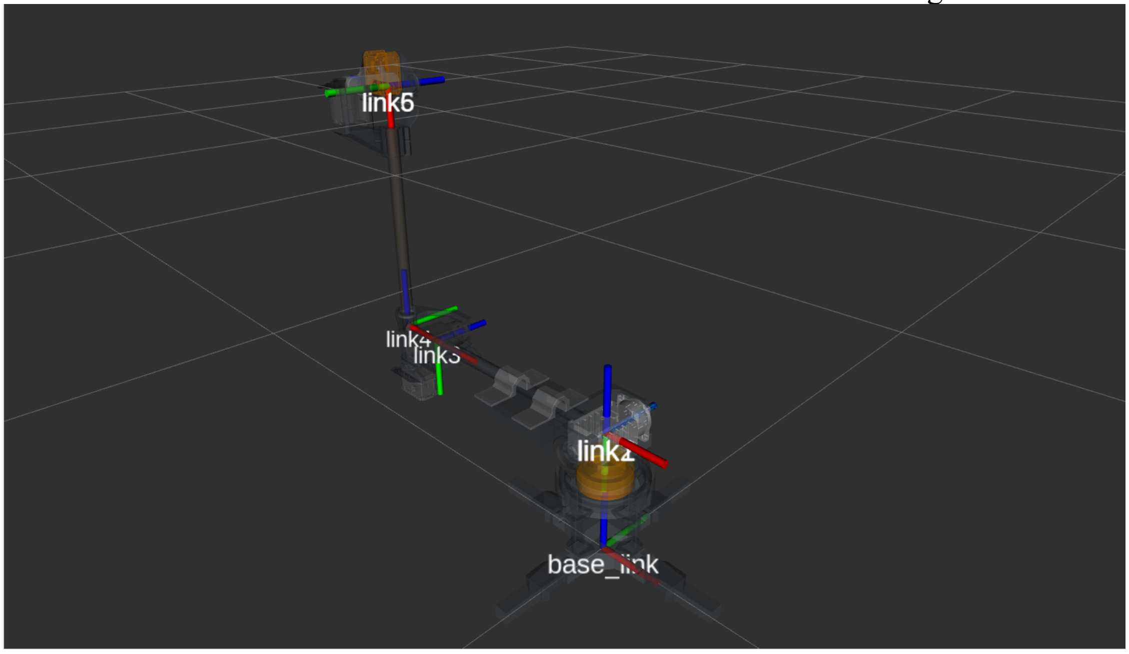
Link (i)	a_i	α_i	d_i	θ_i
1	0	0	0.196	θ_1
2	0	-90°	0	θ_2
3	-0.373	0	0	θ_3
4	-0.08	-90°	0	θ_4
5	0	-90°	0.391	θ_5
6	0	-90°	0	θ_6


(a) Frame 1


(b) Frame 2 sharing origin with Frame 1


(c) Frame 5


(d) Frame 6 sharing origin with frame 5



(e) *r_mini* build completion

Figure 3: The location and orientation of *r_mini*. The choice of the orientation for each frame is based on Denavit-Hartenberg convention. The joints's values are represented by the angle between the *x-axis* around the *z-axis* (the rotation axis of each joint) of the actuator

Following the DH-convention, each links rotates about the *z-axis* of each frame it is attached to, where $(joint_1, \ldots, joint_6)$, corresponds to $(\theta_1, \ldots, \theta_6)$ respectively. In Table 1, each row represents elements of homogeneous transformation, used in Eq. (2:

$$T_i = T(R_z(\theta_i))T(t_z(d_i))T(t_x(a_i))T(R_x(\alpha_i)) \quad (2)$$

where, $R_z, R_x \in \boldsymbol{SO(3)}$, are the rotation operations about the $z-axis$ and the $x-axis$ respectively, and $t_z, t_x \in \mathbb{R}^3$ are the translation vector on the $z-axis$ and the $x-axis$

respectively, while, $d, a, \alpha \in \mathbb{R}$, are scalars. The homogeneous transformation between the origin of the base of the robot into the end-effector of the robot, which coincide with $joint_6$ is shown in Eq. (3),

$$\begin{bmatrix} t_{ee} \\ 1 \end{bmatrix} = \left(\prod_{n=0}^{6-n>0} T_i \right) \begin{bmatrix} t_0 \\ 1 \end{bmatrix} \quad (3)$$

where $t_{ee} \in \mathbb{R}^3$, is the point location of the end effector in 3D space, and $t_0 \in \mathbb{R}^3$ is the origin of the base of the robot. Since the rotation involve in Eq. 3 includes the rotation about the origin of the local frames, the orientation of the end-effector can be represented by,

$$R_{ee} = \prod_{n=0}^{6-n>0} R_z(\theta_n) R_x(\theta_n) \quad (4)$$

Here the operation is closed. Often, to reduce computation expenses and trailing errors due to matrix-matrix multiplication, the rotation operation of *r_mini* are done over the quaternion (Eq. (5)),

$$\boldsymbol{q}_{ee} = \prod_{n=0}^{6-n>0} \boldsymbol{q}_z(\theta_n) \boldsymbol{q}_x(\theta_n) \quad (5)$$

where the Eq. (2) and Eq. (3) represent the forward kinematic solution for the end-effector of *r_mini*.

The self-collision, robot collision checking is delegated to a collision and proximity query library, the Flexible Collision library (FCL). Later in the algorithm formulation of the RRT and the cycle space, subroutine from the FCL will be invoked to check collisions between the manipulator and the moving obstacles. The robot manipulator and the obstacles are encoded inside the collision scene which reside in the planning scene () when the RRT datastructure is initialized (refer to Algorihtm 2 line 1).

We use Newton-Raphson method to find the inverse kinematic solution of *r_mini*, $\hat{c}^{ee}$. The generalization of the method uses the current value of the robot's encoder, $c^{current}$, and the termination value, $\epsilon = 0.005$, to end the iteration. Algorithm 1 delineate the method:

Table 2: The Newton-Raphson algorithm for *r_mini*'s inverse kinematic solver

```
Algorithm 1: getInverseKinematics
   Input: $c_{goal}$, $c^{current}$,$\epsilon$
   Output: $\hat{c}_{ee}$
1  $e \leftarrow$ getForwardKinematic($c^{current}$)
2  while $\|e\| \neq \epsilon$ do
3      $\hat{c}^{estimate} \leftarrow c^{current}+$ getInverseJacobian($c^{current}$)$e$
4      $e \leftarrow C_{goal}-$ getForwardKinematic($c^{estimate}$)
5      $c^{current} \leftarrow c^{estimate}$
6  $\hat{c}^{ee} = c^{current}$
7  return $\hat{c}$
```

3.2. The Rapidly-Exploring Random Trees and its Mathematical Background

This research uses RRT implementation provided by OMPL library packaged in the MoveIt software. The algorithm for the purpose of this research is shown in Algorithm 2:

Table 3: The RRT algorithm

```
Algorithm 2: generateRRT
   Input: $C_{initial}$,$C_{goal}$,$\Delta t$,$k$,$\mathcal{M}$
   Output: $\mathcal{T}$
1  $\mathcal{T}$.initialize($C_{initial}$,$C_{goal}$,$\mathcal{M}$)
2  while $c_{new} \neq C_{goal}$ do
3      $c_{random} \leftarrow$ randomState()
4      $c_{near} \leftarrow$ kNearestNeighbor($k$,$c_{random}$,$\mathcal{T}$)
5      $\boldsymbol{u} \leftarrow$ selectInput($c_{random}$,$c_{near}$)
6      $c_{new} \leftarrow$ newConfiguration($c_{near}$,$\Delta t$)
7      $\mathcal{T}$.append($c_{new}$,$c_{near}$,$\boldsymbol{u}$)
8  return $\mathcal{T}$
```

where k represent the number of nodes in the tree generated by the RRT; $\mathcal{M}$, represent the collision space of the planning scene where all RRT sampling takes place and, $\mathcal{T}$, is the tree that points to a non-colliding space. In this RRT implementation, the map is loaded or queried in line 1 each time the *generateRRT()* is invoked. Line 3 generates a random state bias towards the C_{goal}. Line 4 invokes the k-nearest neighbor to find a selection of nodes that is closes to the state configuration, c_{random}. Line 5 is the core of the RRT sampling where it represents the controlling input of the robot motion. Since, the robot is controlled in the joint-configuration space, the angular joint limit addresses the shape of the workspace. However, given the angular velocity, these limits are translated into the configuration space via the kinematic Jacobian which requires the information on the Δt. The limits implicitly ensure that the RRT, by executing Line 5 within the context of the robot‘s Jacobian, does not pass through the singularities of the robot. Hence, the configuration space of the manipulator also includes, C_{limit}, containing configuration that abides the joint-configuration space range and angular velocity limit.

The configuration space where sampling occurs is modified in this paper where, the rotation representation and its sampling is in $R \in \mathbb{H}$, such that the parameterization of the Hamiltonian-space is the quaternions, $\boldsymbol{q} \in \mathbb{R}^4$. Therefore, the representation of the robot poses and also the non-colliding poses, $(\boldsymbol{q}, \boldsymbol{t})$, are explained in Eq. 6.

$$c_{pose} = \left[\begin{array}{c} \boldsymbol{q} \\ \hdashline \boldsymbol{t} \end{array}\right] \quad (6)$$

The RRT sampling involves query into a map, that stores objects that are prone to collision. This is the planninq scene, denoted as collision map, where the RRT sampling occurs. The query is invoked when both initial pose and a goal pose are sent to the RRT planner input. The output of the pipeline is a set of non-colliding space where from another pipeline, transform the configuration space into a control space. We will define the control space in the following section.

3.3. The Cyclical Space

The cyclical space is the subset of the planner solution where the RRT algorithm is invoked twice. During the generation of the cyclical space, the RRT output a trajectory from the initial pose, $c_{initial}$ to the goal pose, c_{goal}, into a controlling pipeline. The trajectories are then mapped from the configuration space into the joint-configuration space via the Newton-Raphson inverse kinematic solver (see algorithm 1). To complete the set of the cyclical space, the entries in the initial pose and the goal pose are swapped, while invoking the query into the collision map,

$$\tau = (C^{control}, t) \quad (7)$$

which forms a cyclical motion between the initial pose and the goal pose. Here, $C_{cycle} = \overbrace{C_{initial \to goal}}^{\text{see algorithm 3 line 4}} + \overbrace{C_{goal \to initial}}^{\text{see algorithm 3 line 7}}$. Algorithm 3 block explains how the C_{cycle} space is constructed.

The control space is represented by the trajectory in the joint-configuration space $C^{control} \subset C^{cycle} \in S$, where S is the 6-hypersphere homomorphic to $\mathbb{R}^6$ because each joint is constrained to its angle limit. In Eq. (7), the joint-configuration space is equivalent to the configuration space in Eq. (6). The control space is the direct controlling parameters for the movement of *r_mini* where it only handles the control space (or joint-state space). The sampling of the RRT to generate the tree data structure, $\mathcal{T}$, are done within the $\boldsymbol{SO(3)} \times \mathbb{R}^3$ topology. The free configuration space, or the non-colliding pose, is represented by, $C_{free} = C_{workspace}/C_{obstacle}$. According to LaValle et. al [5], this also covers the physical contraint of the non-holonomic movement of the robot. However, in the case of an articulated robot arm in this research, the configuration limitations are the range of the joints and the angular velocity limits. Since, these measurements are in the 6-space, to map them into the $\boldsymbol{SO(3)} \times \mathbb{R}^3$, we use the kinematic Jacobian.

Table 4: The cycle space generator where the movement within the constraint of the cycle space (also a cyclical space) is dependent on the map, $\mathcal{M}$.

```
Algorithm 3: generateCycleSpace
   Input: $c_{initial}$,$c_{goal}$,$\Delta t$
   Output: success_status
1  Function generateCycleSpace($c_{goal}$, $c_{initial}$):
2      success_status ← false
3      while within iteration do
4          $\mathcal{T}_{initial \rightarrow goal} \leftarrow$ generateRRT($c_{initial}$,$c_{goal}$,$\Delta t$)
5          success_status ← moveRobot($\mathcal{T}_{initial \rightarrow goal}$,$\Delta t$)
6          if success_status = true then
7              $\mathcal{T}_{goal \rightarrow initial} \leftarrow$ generateRRT($c_{goal}$,$c_{initial}$,$\Delta t$)
8              success_status ← moveRobot($\mathcal{T}_{goal \rightarrow initial}$,$\Delta t$)
9      return success_status
10 Function moveRobot($\mathcal{T}$, $\Delta t$):
11     for all index in $\mathcal{T}$.vertices do
12         $c^{cycle}$(index) ← getIK($\mathcal{T}$.vertex(index))
13         $t \leftarrow \mathcal{T}.\boldsymbol{u}$.(index)$\Delta t$
14         $\mathcal{T}$.append($c^{cycle}$, $t$)
15     success_status ← TrajectoryController($\tau$)
```

4 . METHODOLOGY

The methodology starts with the benchmarking of sampling-based planners available in the OMPL library and comparing the performance of the planners with the RRT. The benchmark is followed by experimentation in the simulated environment with a simulated robotic arm (*r_mini*) followed by the coupling of the simulated environment with *r_mini* hardware. The experimentation involves the moving obstacle that is introduced synthetically in the collision space. This was necessary since, at the time of this experimentation, the feedback from the mapping sensors was unavailable.

4.1. Benchmarking of Sampling-Based Motion Planners

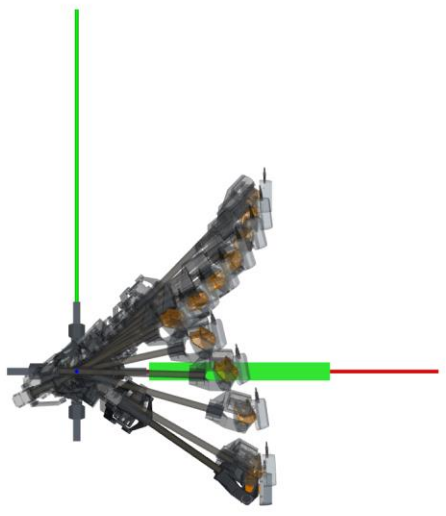
In this research, the planner for the dynamic obstacle avoidance is selected based on the performance of a benchmarking activity. Two poses were set for the benchmark, pose initial were represented in the form of Eq. (6). The following vectors explain the numerical value of these poses concerning the frame attached to the base of *r_mini*.

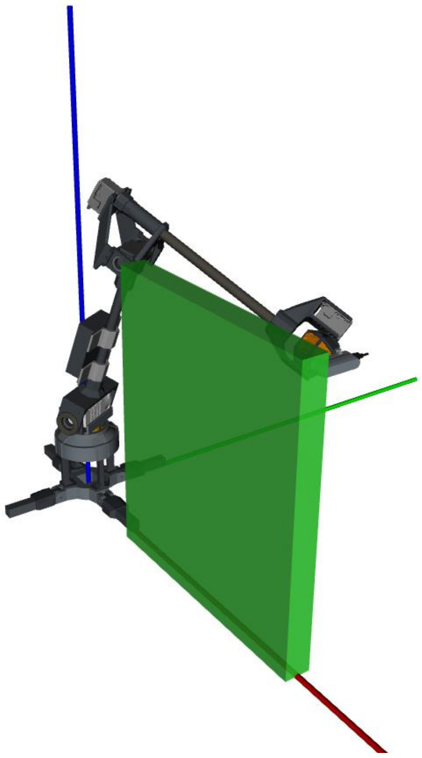
$$c_{initial} = \begin{bmatrix} 1.0 \\ 1.71 \\ 1.0 \\ 1.71 \\ \hdashline 1.43 \\ 1.25 \\ 1.42 \end{bmatrix} \quad c_{goal} = \begin{bmatrix} 1.0 \\ 1.71 \\ 1.0 \\ 1.71 \\ \hdashline 1.46 \\ 1.29 \\ 1.43 \end{bmatrix} \quad (8)$$

A box, with dimension, 0.5 m × 0.05 m × 0.575 m, was placed in front of the robot, it's pose was described by the vector in Eq. (9),

$$c_{box} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \hdashline 0.45 \\ 0 \\ 0.2874 \end{bmatrix} \quad (9)$$

Fig. (4) shows the simulation setup and the planning motion in action. The simulation was run for 50 requests from the initial pose to the goal pose. Time processing was given a 10 s limit. The memory limit was set to 1 Mb. The time limit for a request, including the motion and the processing time was set to 3637 s (about 1 hour). This paper uses these configurations and the default configuration of each planner in the MoveIt to start the benchmarking.





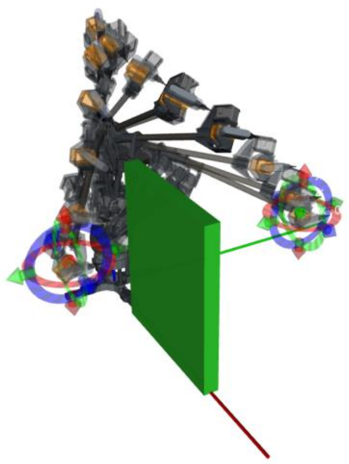


Fig. 4: The top view of the simulation shown in (top), and the isometric view of the benchmark setup (middle). *r_mini* attempts to move around the static obstacle placed in it's immediate configuration workspace (bottom).

4.2. Experiment Design

The cyclical space is populated by the RRT-Newton-Raphson pipeline where the generated trajectories are then pass to the control pipeline where the controller will spline the sparse trajectory waypoints. Two poses are defined in this experimentation which has been described in Eq. (8). A moving obstacle is placed in front-view of the robot. The obstacle is a cylinder with 0.1 m radius base at 1 m height. The obstacle moves from 0.3 m to 1.7 m away from the robot in oscillation. The period of motion is harmonic, such that, the robot follows

$1+0.7\sin(2\pi(0.0006)\Delta t)$ along the $x-axis$. Two velocities ($\mathcal{V}$) values were used: 50% and 10% scale from the maximum velocity of the end-effector.

The planner is invoked five seconds before the obstacle is placed into the planning scene. As described previously, the cylinder is directly placed into the planning scene (i.e., collision space) such that no motion tracking via mapping sensor feedback is necessary for this research. The planner is requested to provide solution for the motion described by the cycle space. Twenty iterations were done with each given a five minutes runtime. The metric used for this experiment was the time on first collision where, when the prototype touches the cylinder, the iteration is terminated. This experimentation was done, both, in simulation, and with the real robot hardware coupled with the simulated environment. To reiterate, for both the simulation and the hardware validation, the obstacle was augmented in simulated environment.

5 . RESULT AND DISCUSSION

There are two parts of the results in this paper, the first dealing with the benchmarking result to ascertain the best sampling-based planner. The second part delve into the performance of the selected planner from the benchmarking on a moving obstacle.

5.1. Benchmark Result

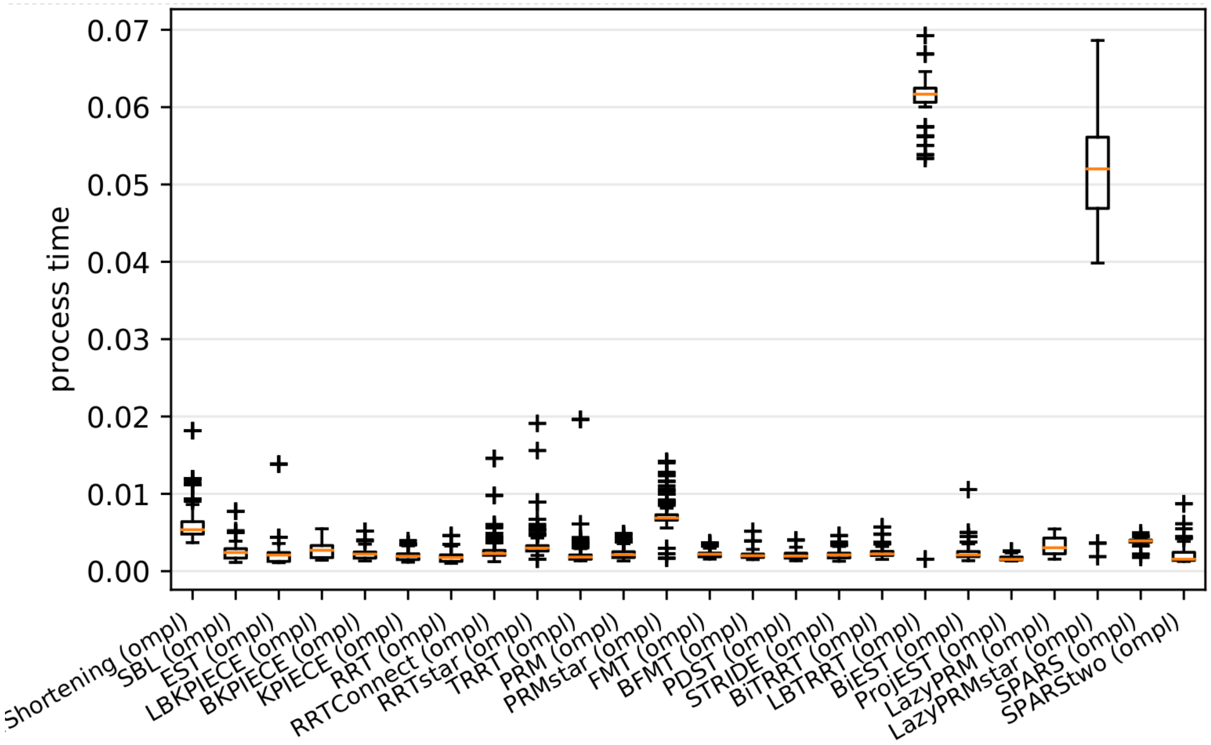


Fig. 5: The benchmark result when two configurations are defined and pass to the OMPL planner pipeline. All planners completed a 50-cycle query from an initial pose to a goal pose. RRT required the least amount of processing time at finding the motion planning solution, followed by the PRM.

Fig. (5) shows the compiled statistics of the time the solutions that were passed to the controller (in this case a virtual controller for simulation of *r_mini* in the simulated environment). RRT requires on average, 0.031 planning time while PRM requires 0.035 planning time from the initial pose to the goal pose when subjected to an obstacle close to the robot. Wei & Ren [7] explained that the improved RRT algorithms, such as the bi-RRT, and the RRT-connect, solve a query faster . However, based on our benchmarking and in the case of this experimentation setup, vanilla RRT, or base-RRT, and PRM outperformed their improved variants when completing the path query between an initial pose and a goal pose. To that end, this research uses vanilla RRT as the scheme for the high-level local planner. This result helps us select the motion planner for dynamic obstacle avoidance.

5.2. The Performance of RRT on a Dynamic Environment

Table 5: The simulated and hardware-connected result of the performance of RRT in a dynamic environment. NC stands for *No Collision* after five minute runtime

	time to 1^{st} collision,(s)				
condition\iteration	1^{st}	2^{nd}	3^{th}	4^{th}	5^{th}
$\text{simulation}_{0.5\nu}$	64	NC	NC	133	18
$\text{hardware}_{0.1\nu}$	205	16	17	134	13
$\text{simulation}_{0.5\nu}$	13	23	11	9	13
$\text{hardware}_{0.1\nu}$	17	4	15	7	10

Table 5 shows the recorded time to the collision of 20 iterations. The average time to collision was 40 s. There were two iterations where no collision was recorded. This performance is subjected to Algorithm 3, specifically in line 4 and line 7, when RRT is invoked. Within this call (refer Algorithm 2: line 1), the random tree initialization considers an obstacle map that is outdated, given the cylinder moving further toward the manipulator when the RRT is executed. Within the RRT algorithm, there are no mechanism for the robot to stop or move at a lower rate to avoid the cylinder. Fig. 6 shows the sequence when the end-effector collided with the cylinder.

Despite the obstacle collision when the moving cylinder approaches the robot, specifically when the centroid of the cylinder is nearing the $x - axis$ of the $c_{initial}$ and c_{goal}, the planner reacts to the obstacles when lines 4 and 7 in Algorithm 3 are invoked by attempting to move around the cylinder.

The planner shows reactive behavior (local planning) when the cyclical space is initialized via Algorithm 3. Fig. (7) illustrates such behavior in the simulated environment, and Fig. (8) shows the same behavior in the hardware reiteration of the experimentation. The reactive behavior is illustrated in Fig. (9), where change we observed changes in movement range and a range in movement rate.

No significant changes are observed for $joint_4$, $joint_5$ and $joint_6$. This is the implication of the Pieper-condition manipulator design where, none of the $z - axis$ from the first three joints shares the same crossing point, which suggest the actuation on these joints are not a linear transformation as the case for affine translation. Due to the offset (affine transformation) of the joints' axis of rotation, there is a bijection mapping of these joints to the task-space specifically reserved for translation changes in space. Also, changes are observed in the orientation of the frame attached to the end-effector, however, there is no bijection mapping of the three joints to the task-space's orientation.

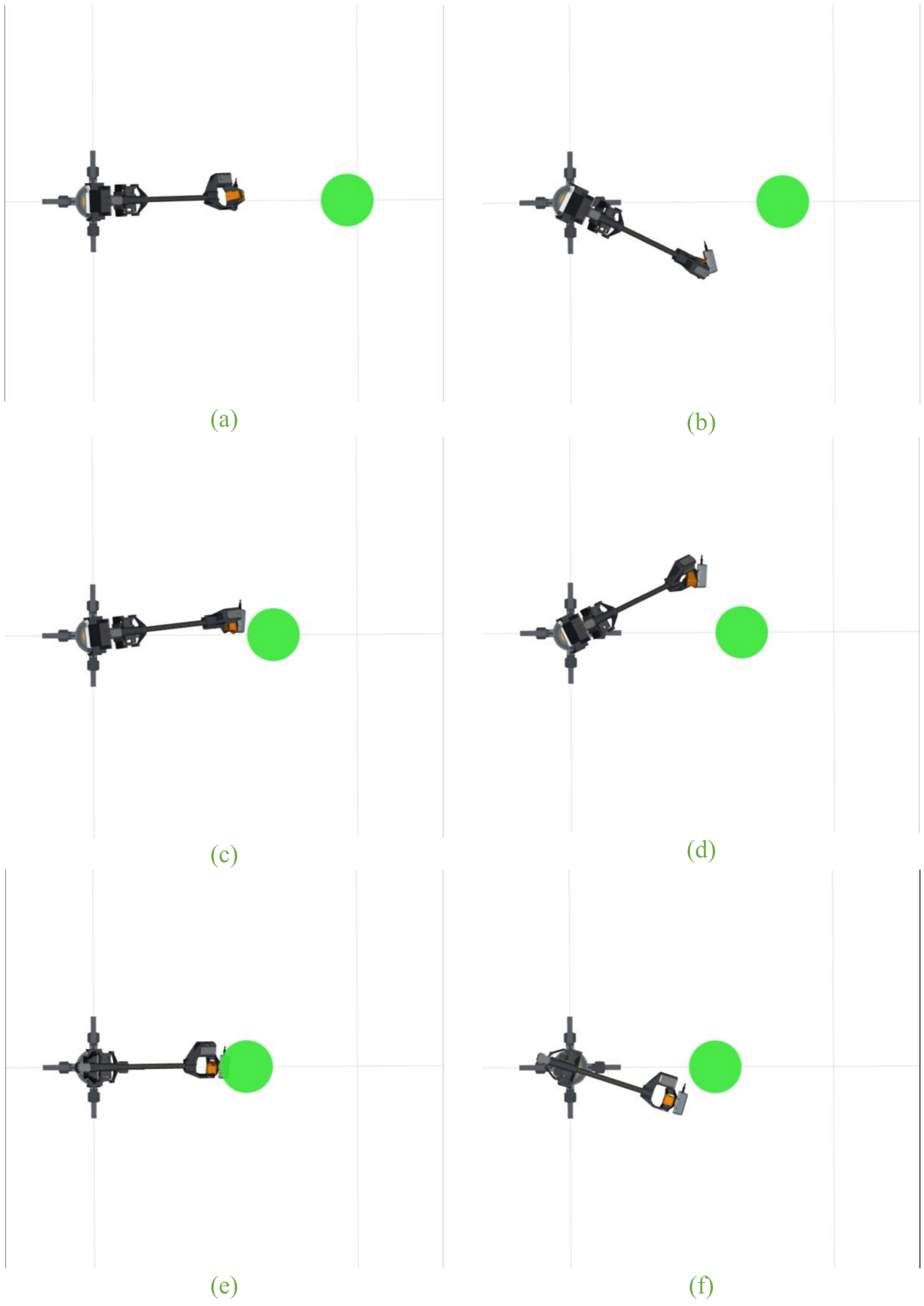
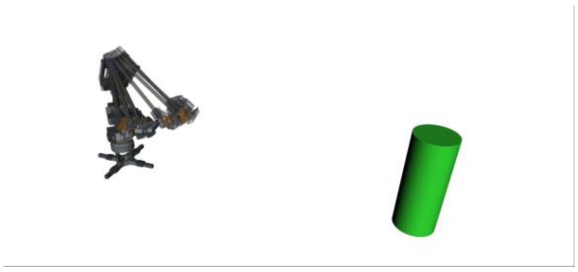
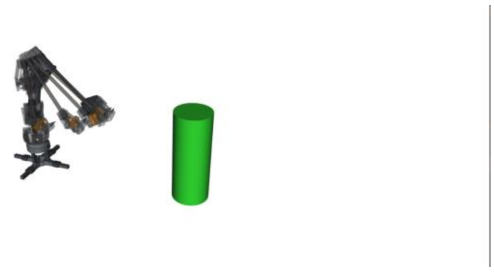
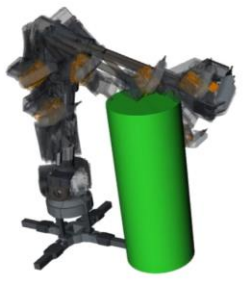
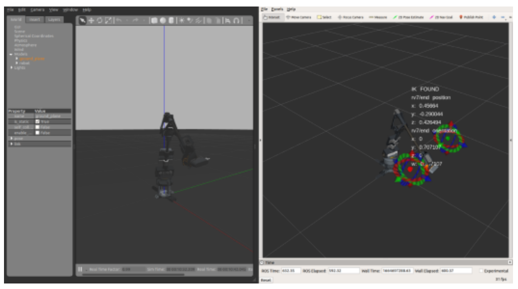

(a) (b) (c) (d) (e) (f)

Fig. 6: These sequences show the manipulator follows an outdated trajectory and collides with the cylinder despite attempt to move away from the moving cylinder.


(a)


(b)



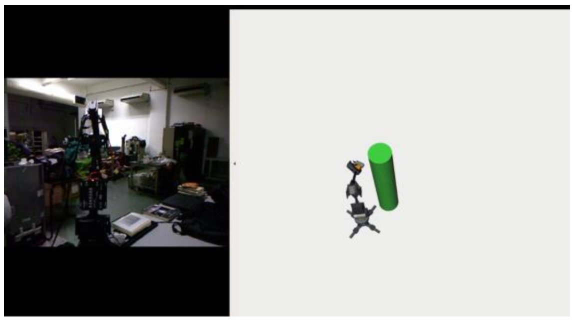


```
IK FOUND
rv7/end position
x: 0.45664
y: -0.290044
z: 0.426494
rv7/end orientation
x: 0
y: 0.707107
z: 0
w: 0.707107
```

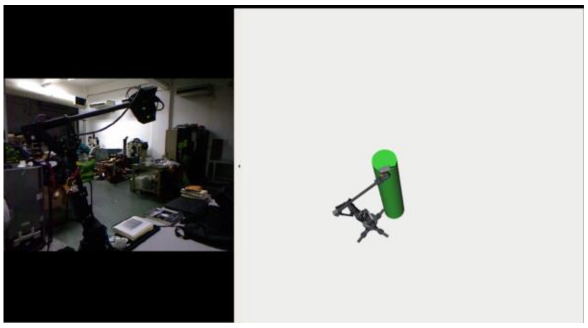
(d)

(c)

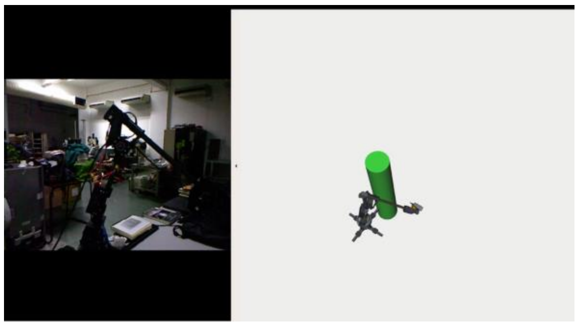
Fig. 7: The chronology of the attempt at avoiding a moving obstacle when the obstacle approaches the robot seen in (a) and (b). The planner successfully provide a non-colliding solution when the cylinder is moving away from the robot (c). This experimentation is defined in a simulated setup using Gazebo with the ODE physic engine to replicate the robot hardware and encoders feedback and the cyclical space initialization (d).



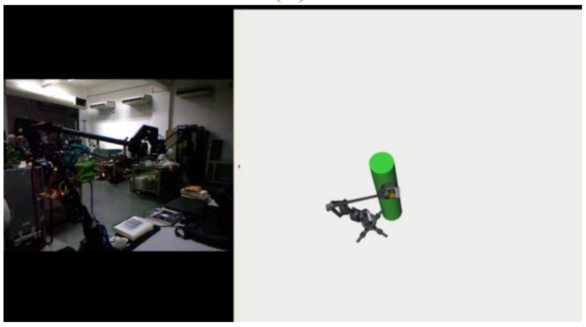
(a)



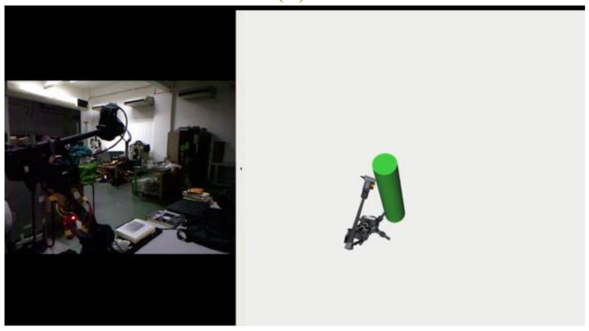
(b)



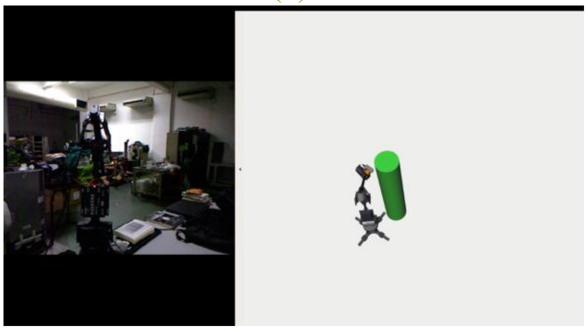
(c)



(d)



(e)



(f)

Fig. 8: The sequence of motion when *r_mini* successfully avoid a moving obstacle when the obstacle is at a turning point to move away from the hardware.

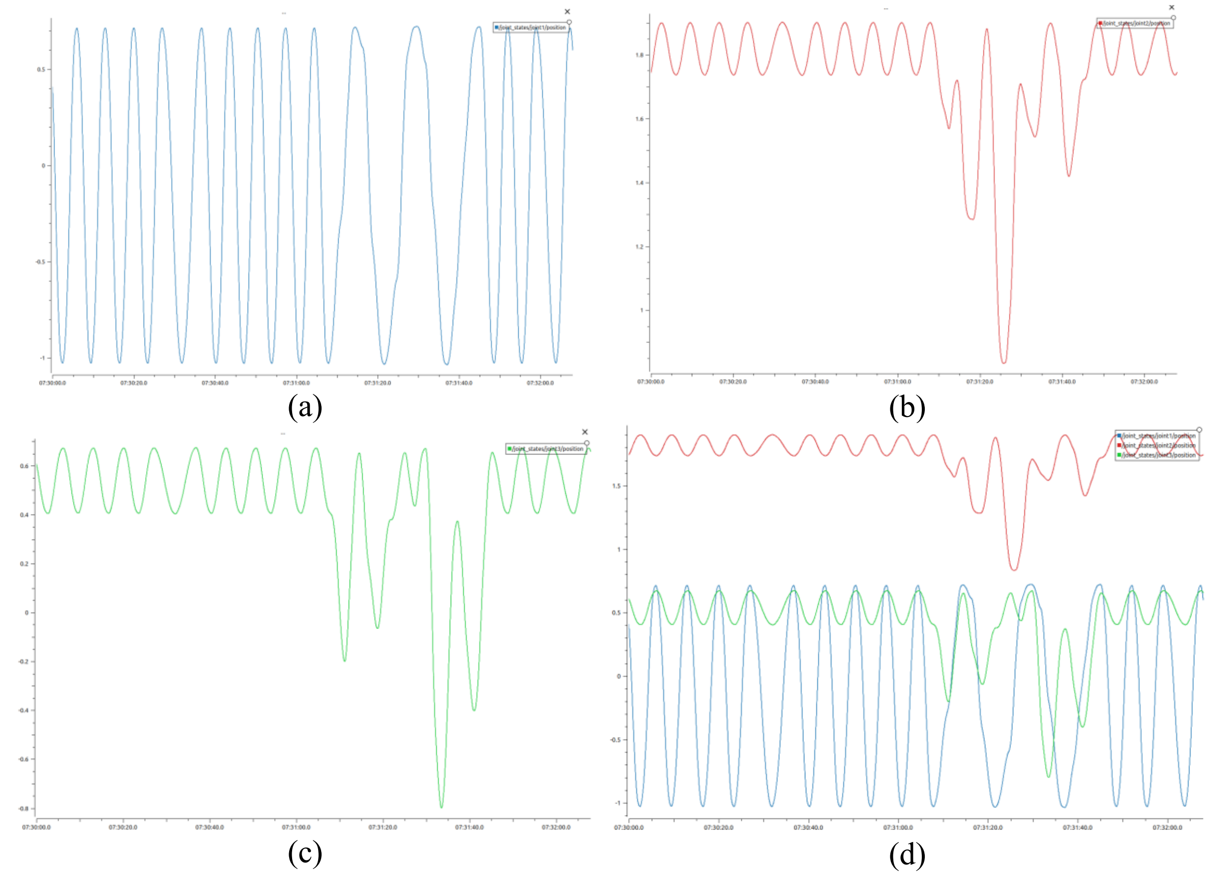


Fig. 9: Reactions from joints 1, 2, and 3, show that the planner, together with the cycle space, behave reactively towards the moving object. No rapid movement or rate on joints 4, 5, 6 are observed.

6 . CONCLUSION AND RECOMMENDATION

We conclude that the RRT outperformed other sampling-based planners when the workspace of a manipulator is subjected to static obstacle. On a dynamic setup, we observe that the RRT react to a moving object under the cycle space. We also concluded that given the velocity constrained configured within the RRT, the planners did not pass through any singularities during sampling. To increase the time-to-collision value, we proposed an obstacle-tracking pipeline via mapping sensors such as an RGB-D sensor or a LiDAR. It is also recommended that more than one intermediate pose capable of reacting with the environment between the initial pose and the goal pose should be defined in the cycle space.

BIBLIOGRAPHY

[1] M. Hägele, K. Nilsson, J. N. Pires, and R. Bischoff, “Industrial robotics,” in *Springer Handbook of Robotics*, Springer International Publishing, 2016, pp. 1385–1421.

[2] B. Matthias, S. Kock, H. Jerregard, M. Källman, and I. Lundberg, “Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept,” in *Proceedings - 2011 IEEE International Symposium on Assembly and Manufacturing, ISAM 2011*, 2011, no. 230902, doi: 10.1109/ISAM.2011.5942307.

[3] M. Elbanhawi and M. Simic, “Sampling-based robot motion planning: A review,” *IEEE Access*, vol. 2. Institute of Electrical and Electronics Engineers Inc., pp. 56–77, 2014, doi: 10.1109/ACCESS.2014.2302442.

[4] L. E. Kavraki, M. N. Kolountzakis, and J. C. Latombe, “Analysis of probabilistic roadmaps for path planning,” *IEEE Trans. Robot. Autom.*, vol. 14, no. 1, pp. 166–171, 1998, doi: 10.1109/70.660866.

[5] S. M. LaValle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” 1998. Accessed: Sep. 25, 2022. [Online]. Available: https://www.cs.csustan.edu/~xliang/Courses/CS4710-21S/Papers/06 RRT.pdf.

[6] T. Kunz, U. Reiser, M. Stilman, and A. Verl, “Real-time path planning for a robot arm in changing environments,” *IEEE/RSJ 2010 Int. Conf. Intell. Robot. Syst. IROS 2010 - Conf. Proc.*, pp. 5906–5911, 2010, doi: 10.1109/IROS.2010.5653275.

[7] K. Wei and B. Ren, “A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm,” *Sensors (Switzerland)*, vol. 18, no. 2, 2018, doi: 10.3390/s18020571.

[8] D. L. Pieper, “The Kinematics of Manipulators Under Computer Control,” Stanford University, 1968.

[9] J. Denavit and R. S. Hartenberg, “A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices,” *J. Appl. Mech.*, vol. 22, no. 2, pp. 215–221, 1955, doi: 10.1115/1.4011045.