Repurposing a Sampling-Based Planner for a Six-Degree-of-Freedom Manipulator to Avoid Synthetic Moving Obstacles

Hafiz Iman[1] and Raisuddin Khan[1]

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

Abstract

This paper presents the use of a sampling-based planner as a local planning scheme to avoid obstacle between a robotic arm and a moving obstacle. Based on a planner benchmark on a obstacle-ridden environment, rapidly-exploring random tree (RRT) planner are used to populate the trajectories of the task space and map them into a configuration space using Newton-Raphson-based inverse kinematic solver. Three robot poses are defined in a cycle of back-and-forth motion; the starting, the midpoint, and the end pose. The starting pose and the end pose are equal. The robot repeatedly moves from the starting pose to the end pose via the midpoint pose. Poses between the three are a subset of a trajectory populated by the sampling-based planner. Each set of trajectory is unique. We impose periodically occuring 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 cyclical space to emulate a dynamic environment. Based on the performance of the local planning strategy, the robot has a higher success in avoiding the obstacles when the planning query starts during the presence of the obstacles in the cyclical space. However, a higher chance of collision occurs when the planning query is invoked when the obstacle nearly approaches the cyclical space.

Keywords— mechatronics, robotics, planner, path-planning, sampling-based planner, robotic arm, manipulator, dynamic environment, moving obstacles, ROS, MoveIt

1 Introduction

Robot manipulators such as industrial robots work well in repetitive and heavy tasks. They are high-performing, objective, and relentless at task 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 impose 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 (**Hagele2016**). Regardless, it is still an issue of hazard should a compliant or cooperative robot collide with a person working at close to its workspace (**Matthias2011**). The collision also warrants expensive maintenance and repairs. To address a more intelligent motion and, evidently, a safer motion planning, an industrial robot system implements a certain degree of planning algorithm specifically for the motion of the manipulator.

A robot motion planner provides a collision-free motion solution for a manipulator. Here, a solution is defined as collections of waypoints or trajectories. In the case of the traditional definition of industrial robots, the planning is global because the robot is enclosed and isolated in 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 and generates 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 imparative for a compliant manipulator or cooperative manipulator to have an efficient motion planner because replanning is computationally expensive and time-consuming; sampling-based planners are the pinnacle example of efficient planning (**Elbanhawi2014**). Unfortunately, sampling-based planner trade completeness and optimality with efficiency where the planner may fail to provide a solution (**Kavraki1998**). Also, if a solution exists under its metric space, the waypoints may not be the least cost path to a goal (**LaValle1998**). 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 sampling are stochastic, such that resampling will give a unique solution to the previous sampling. Most sampling-based planner are also tractable in higher-dimensional configuration and task space. However, sampling-based planners assume 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 leverage the efficiency and the computationally reserved sampling-based motion planning without needing to apply 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 of this report, we will assume sampling planners provide solution in higher dimensional configuration space, which implicates a solution with a set of poses represented by the special Euclidean group, $\boldsymbol{SE(3)}$.

This paper is a part of a collision avoidance system for a compliant robot manipulator design to prove an encoderless concept, and only cover the formulation of the planning approach we adopt for our robotic system.

2 Related Work

In this review, we will look into research papers that delve into motion planning in a dynamic environment for robot manipulators in three dimensional space, $\mathbb{R}^3$. We pay close attention to algorithms derived from sampling-based planner which use stochastic approach to query the configuration space. The planning algorithms for robot motion in a dynamics environment dated back to 1985, albeit mainly applied on mobile robot (**Mohanan2018**).

Some planning algorithms that is non-probabilistic for robotic arm are also observed which are based on representation space (**Su2011**; **Liu2016a**), sequential expanded Langrangian homotopy (**Dharmawan2018a**), and real-time adaptive motion planning (RAMP) (**Vannoy2008**). These works provide a new method of planning and tackle the problem at the local planning and at lower level of control to solve problems with the aid of a simulated environment. Their simulated environment, or a map model of the environment, includes dynamic objects that are then filtered and disregarded in the map registration pipeline. Their experimentation approach will be adopted in this research where, the map is informed with the presence of obstacles that are moving in the workspace without having another sub-module of the system tracking the object via motion tracking.

2.1 The Probabilistic Motion Planning

Kavraki1996 is the first group of researchers that used probability model for sampling the configuration space for holonomic robot motion such as a manipulator robot. The planner are called the probabilistic roadmap (PRM) motion planning. The algorithm construct a graph structure to find path between an initial pose to a goal pose in two-dimensional configuration space, $n = 2$. **Kavraki1996** also proof a more general solution for higher dimensional configuration space, $n > 2$. With graph structure, more than one path connect the initial pose to the goal pose. Therefore, PRM is a multi-query type planner.

Kunz2010a improve 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. **Kunz2010a** 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 (**LaValle1998**) targeting problems addressed in diffential-constrained motion such as a car on a plane. 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 (**Wei2018**). RRT assume as static environment but **Wei2018** successfully change the way RRT samples a robot metric space so that it is fast enough to react with 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.

Researchers have been modifying the PRM (**Klasing2007**; **Likhachev2005**; **Jaillet2004**; **Pomarlan2013a**) and the RRT (**Otte2015**; **Ferguson2007**; **Ferguson2006**; **Bekris2007**) to facilitate better performance. Unlike **Kunz2010a** and **Wei2018**, so few have applied their planning algorihtms on a robot manipulator despite both algorithms provides mathematical framework for .

We will use the method demonstrated by **Kunz2010a** and **Wei2018** to design our experiment of a moving obstacle collision avoidance with the implementation of the vanilla RRT to solve motion for robot manipulator in three-dimensional space, $\mathbb{R}^3$. Different from the implementation by **Wei2018** our method implement the vanilla RRT where we do not represent the obstacle configuration space.

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 *figure 1*) based on the conditioned addressed by **Pieper1968**, which entails three collated joints sharing the same cross point of their $z - axes$ shown in *figure 2*.

r_mini has six revolute axes, $(\theta_1, \ldots, \theta_6)$. The first three axes moves 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, $R \in \mathbb{R}^{3\times 3}$. 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 *equation 3.1*

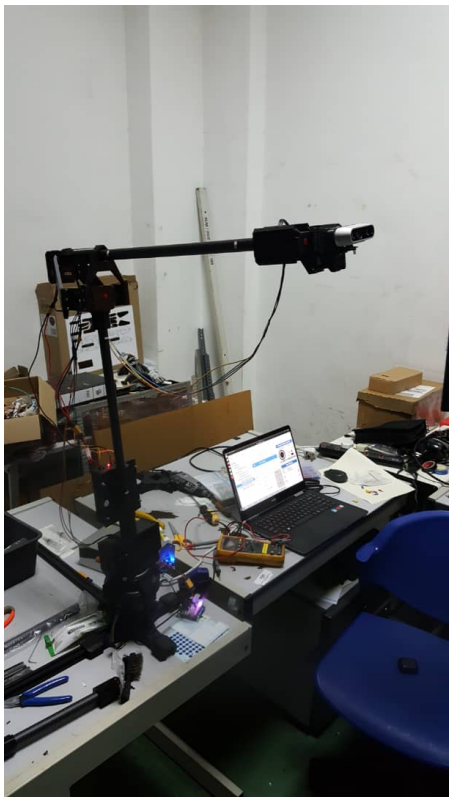
$$T = \left[\begin{array}{c:c} R & \boldsymbol{t} \\ \hdashline \mathbf{0} & 1 \end{array}\right] \tag{3.1}$$

The kinematic models of the *r_mini*follows the Denavit-Hartenberg (DH) formulation (**Denavit1955**). The DH-parameters are shown in *table 1* and the visualization of these parameters in the form of frames transformation are shown in *figure 3*.

Following the DH-convention, each links rotates about the $z - axis$ of each frames it's attached to where $(joint_1, \ldots, joint_6)$ correspond to $(\theta_1, \ldots, \theta_6)$ respectively. In *table 1*, each row of represents a homogenous transformation(*equation 3.2*)

$$T_i = T(R_z(\theta_i))T(t_z(d_i))T(t_x(a_i))T(R_x(\alpha_i)) \tag{3.2}$$

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,



(a) *r_mini*hardware assemblage.

(b) *r_mini*CAD design.

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

while, $d, a, \alpha \in \mathbb{R}$, are scalars. The homogenours 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 *equation 3.3*,

$$\begin{bmatrix} t_{ee} \\ 1 \end{bmatrix} = \left(\prod_{n=0}^{6-n>0} T_i \right) \begin{bmatrix} t_0 \\ 1 \end{bmatrix} \tag{3.3}$$

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 *equation 3.3* involves roation about the origin of the local frames, the orientation of the end-effector can also be represented by,

$$R_{ee} = \prod_{n=0}^{6-n>0} R_z(\theta_n) R_x(\theta_n) \tag{3.4}$$

where the operation is closed. Now, the rotation operation of *r_mini* can be represented

Table 1: DH-parameter table

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

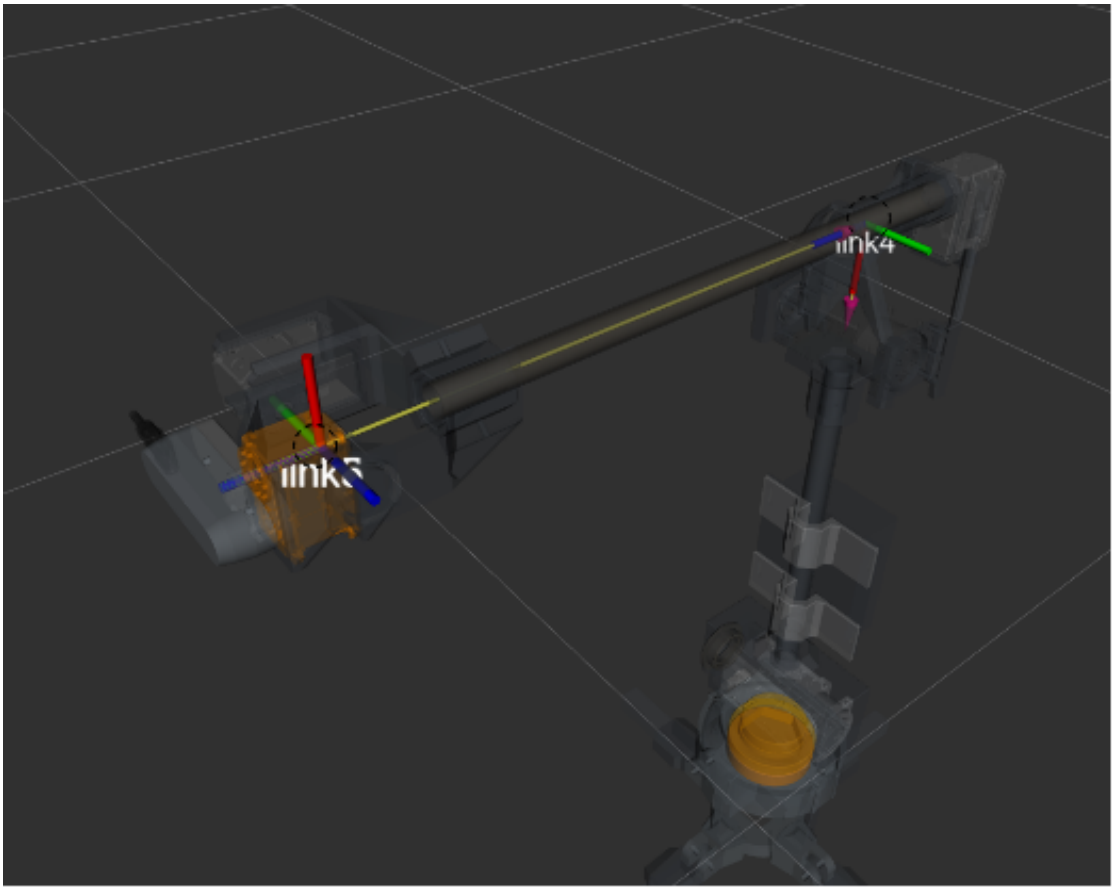
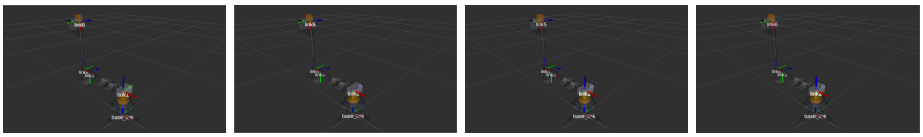
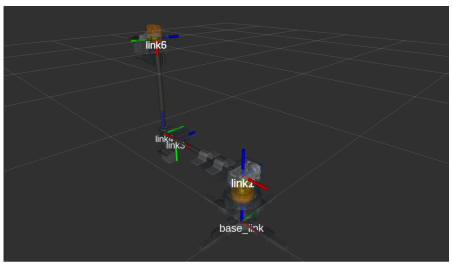


Figure 2: Rimini wrist conforms to Pieper condition where axis of rotation for joint4, joint5, and joint6 share points of intercept. The dash circles in the diagram refer to possible point of intercepts. Both points are valid for a Pieper condition



(a) Frame 1 (b) Frame 2 sharing origin with Frame1 (c) Frame 5 (d) Frame 6 sharing the 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 frames are based on Denavit-Hartenberg. The joints are values represented by the angle between two $x - axis$ around the $z - axis$ (rotation axis) of each actuators

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

Algorithm 1: getInverseKinematics

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

by quaternion,

$$\boldsymbol{q}_{ee} = \prod_{n=0}^{6-n>0} \boldsymbol{q}_z(\theta_n)\boldsymbol{q}_x(\theta_n) \tag{3.5}$$

where an *Equations 3.3* and *3.5* 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 based on the implementation by **Pan2012**. Later in the algorithm formulation of the RRT and the cycle space, subroutine from the flexible collision library (FCL) library will be invoke to check collisions between the manipulator and the moving obstacles encoded inside the collision scene inside the planning scene ($\mathcal{M}$) 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 the method.

Table 3: The RRT algorithm.

Algorithm 2: generateRRT

Input: $C_{initial}$,C_{goal},Δt,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 $\quad c_{random} \leftarrow$ randomState()
4 $\quad c_{near} \leftarrow$ kNearestNeighbor(k,c_{random},$\mathcal{T}$)
5 $\quad \boldsymbol{u} \leftarrow$ selectInput(c_{random},c_{near})
6 $\quad c_{new} \leftarrow$ newConfiguration(c_{near},Δt)
7 $\quad \mathcal{T}$.append(c_{new},c_{near},$\boldsymbol{u}$)
8 **return** $\mathcal{T}$

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*:

where k represent the, number of node in the tree generated by the RRT, $\mathcal{M}$, represent the collision map which is part 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 are 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 important part of the sampling in the RRT where it represent the controlling input of the robot motion. Since, the robot are control in the joint-configuration space, the angular joint limit address 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 ensures that the RRT, by executing line 5 within the context of the robots 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 the RRT-sampling is concerned 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 *equation 3.6*.

$$c_{pose} = \begin{bmatrix} \boldsymbol{q} \\ \cdots \\ \boldsymbol{t} \end{bmatrix} \tag{3.6}$$

The RRT sampling involves query into a map, that stores collision objects. This is the planninq scene, denoted as collision map, where the RRT sampling occurs. The query are called 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 there 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 invoke twice. During the generation of the cyclical space, the RRT output the a trajectory from the initial pose, $c_{initial}$ to the goal pose, c_{goal}, into a controlling pipeline. The trajectory are then map 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 query into the collision map,

$$\tau = (C^{control}, t) \tag{3.7}$$

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

Algorithm 3: generateCycleSpace

Input: $c_{initial}$,c_{goal},Δt
Output: success_status

1 **Function** `generateCycleSpace(`c_{goal}, $c_{initial}$`):`
2 success_status $\leftarrow$ false
3 **while** *within iteration* **do**
4 $\mathcal{T}_{initial \rightarrow goal} \leftarrow$ generateRRT($c_{initial}$,c_{goal},Δt)
5 success_status $\leftarrow$ moveRobot($\mathcal{T}_{initial \rightarrow goal}$,$\Delta t$)
6 **if** *success_status* = *true* **then**
7 $\mathcal{T}_{goal \rightarrow initial} \leftarrow$ generateRRT(c_{goal},$c_{initial}$,Δt)
8 success_status $\leftarrow$ moveRobot($\mathcal{T}_{goal \rightarrow initial}$,$\Delta t$)
9 **return** success_status
10 **Function** `moveRobot(`$\mathcal{T}$, Δt`):`
11 **for** *all index in* $\mathcal{T}$*.vertices* **do**
12 c^{cycle}(index) $\leftarrow$ 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 $\leftarrow$ TrajectoryController(τ)

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

The control space are represented by the trajectory in the joint-configuration space $C^{control} \subset C^{cycle} \in S$ where S is the 6-hypersphere. In *equation 3.7* the joint-configuration space are equivalent with the configuration space in *equation 3.6*. 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 $\mathbf{SO(3)} \times \mathbb{R}^3$ topology. The free configuration space, or the non-colliding pose, are represented by, $C_{free} = C_{workspace}/C_{obstacle}$. According to **LaValle1998**, the $C_{obstacle}$ are 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 limitation are the range of the joints and the angular velocity limit. Since, all of these measurements are in the n-hyperspace, to map them into the $\boldsymbol{SO(3)} \times \mathbb{R}^3$, we use the kinematic Jacobian.

4 Methodology

4.1 Benchmarking of Sampling-Based Motion Planners

In this research, the planner for the dynamic obstacle avoidance are selected based on the performance of a benchmarking activity. Here, the procedure is explained.

Two poses are set for the benchmark, pose initial are represented in the form of *equation 3.6*. The following vectors explain the numerical value of these poses with respect to the frame attached to the base of *r_mini*.

$$c_{initial} = \begin{bmatrix} 0.0 \\ 0.71 \\ 0.0 \\ 0.71 \\ \hdashline 0.43 \\ 0.25 \\ 0.42 \end{bmatrix} \quad c_{goal} = \begin{bmatrix} 0.0 \\ 0.71 \\ 0.0 \\ 0.71 \\ \hdashline 0.46 \\ 0.29 \\ 0.43 \end{bmatrix} \tag{4.1}$$

A box, with dimension, 0.5 m × 0.05 m × 0.575 m, are place in front of the robot, it's pose is described by the vector,

$$c_{box} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \hdashline 0.45 \\ 0 \\ 0.2874 \end{bmatrix}$$

Figure 4 shows the simulation setup and the planning motion in action. The simulation is ran for 50 request from the initial pose to the goal pose. Time processing is given a 10 s limit. The memory limit is set to 1 Mb. The time limit for a request, including the motion and the processing time is set to 3637 s. This paper use these configurations and the default configuration of each planners in the MoveIt to start the benchmarking.

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 *equation 4.1.* A moving obstacle are 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 a $1 + 0.7\sin(2\pi(0.0006)\Delta t)$. Two velocities values were used: 50% and 10% scale from the maximum velocity of the end-effector.

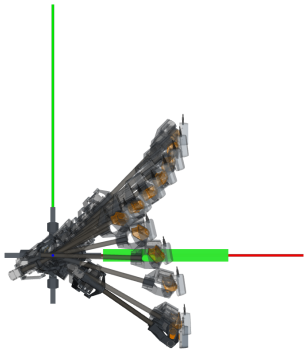
The planner is invoked 5 s before the obstacle is placed into the planning scene. The cylinder are directly place into the planning scene such that no motion tracking is necessary for this research. The planner are requested to provide solution for the motion described by the cycle space. Five iterations are done with each given a five minutes runtime. The metric use for this experiment is the time on first collision where, when the prototype touches the cylinder, the iteration is terminated. This experimentatation is done, both, in simulation, and with the real robot hardware. However, for both the simulation and the hardware validation, the obstacle are augmented in simulated environment.

5 Result and Discussion

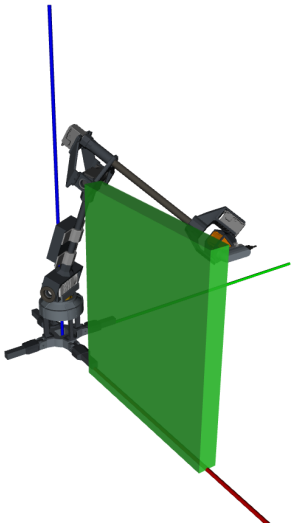
There are two part of the result on this paper, the first dealing with the benchmarking result to ascertain the best sampling-based planner. The second part delve into the performance of the selected planner from the benchmarking on a moving obstacle.

5.1 Benchmark Result

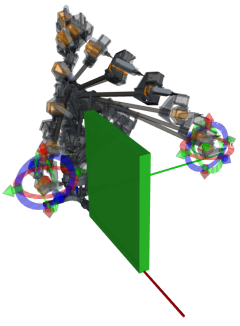
Figure 5 shows the compiled statistics of the time the solution were pass to the controller (in this case a fake controller for simulation of *r_mini* in the simulated environment). RRT requires on average, 0.031 planning time while PRM requires 0.035 planning time from the initial pose to the goal pose when subjected to an obstacle very close to the robot. **Wei2018** explained



(a)



(b)



(c)

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

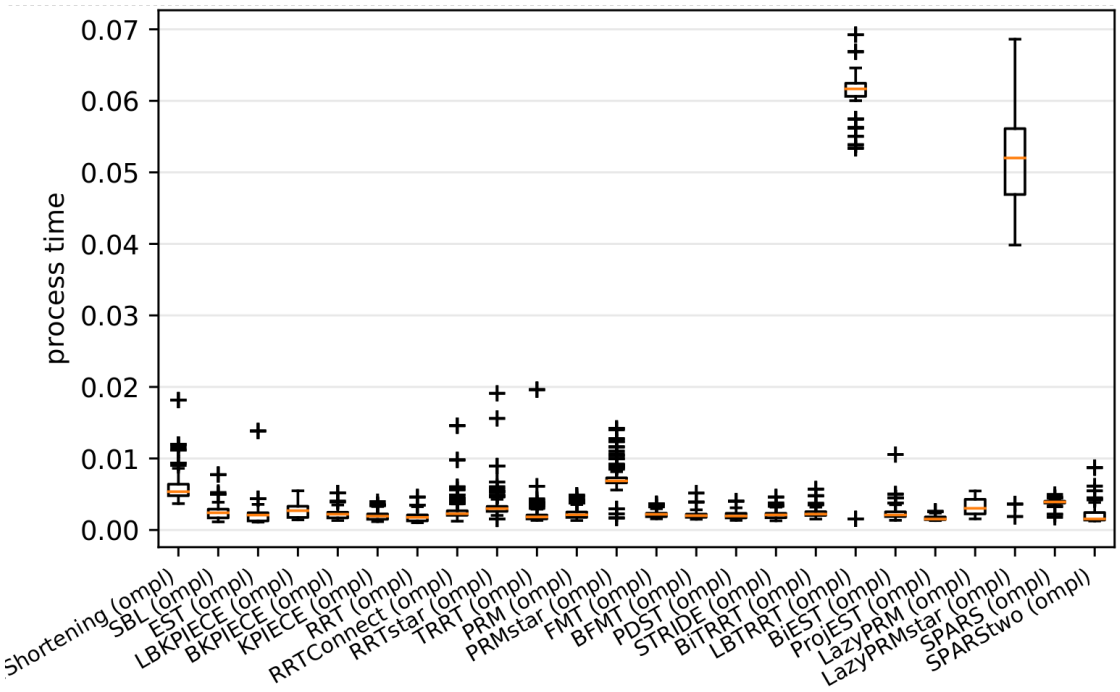


Figure 5: The benchmark result when two configurations are defined and pass to the OMPL planner pipeline. All planners completed a 50-cycle query from an initial pose to a goal pose. RRT requires the least amount of processing time at finding the motion planning solution, followed by the PRM.

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

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

the improved RRT algorithms, such as the bi-RRT, and the RRT-connect, solve a query faster. However, based on our benchmarking, vanilla RRT, or base-RRT, and PRM outperform 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 in selecting the motion planner for the dynamic obstacle avoidance.

5.2 The Performance of RRT on a Dynamic Environment

Table 5, shows the recorder time to collision of 20 iterations. The average time to collision is 40 s. There are two iterations where there are no collision recorded. This poor performance is subjected to the algorithm 3, specifically in line 4 and line 7, RRT is called. Within this call (algorithm 2, line 1 consider an obstacle map that is outdated given the cyclinder has moving further towards the manipulator when RRT: line 1. Within the RRT algorithm, there are no mechanism for the robot to stop or move at a lower rate to avoid the cylinder. *Figure 6* shows the sequence when the end-effector collide with the cylinder.

Despite the obstacle avoidance fails when the moving cylinder approaches the robot specifically when the centroid of the cylinder is nearing the $x-axes$ of the $c_{initial}$ and c_{goal}, the planner successfully avoid the obstacles when the lines 4 and 7 in algorithm 3 is invoked.

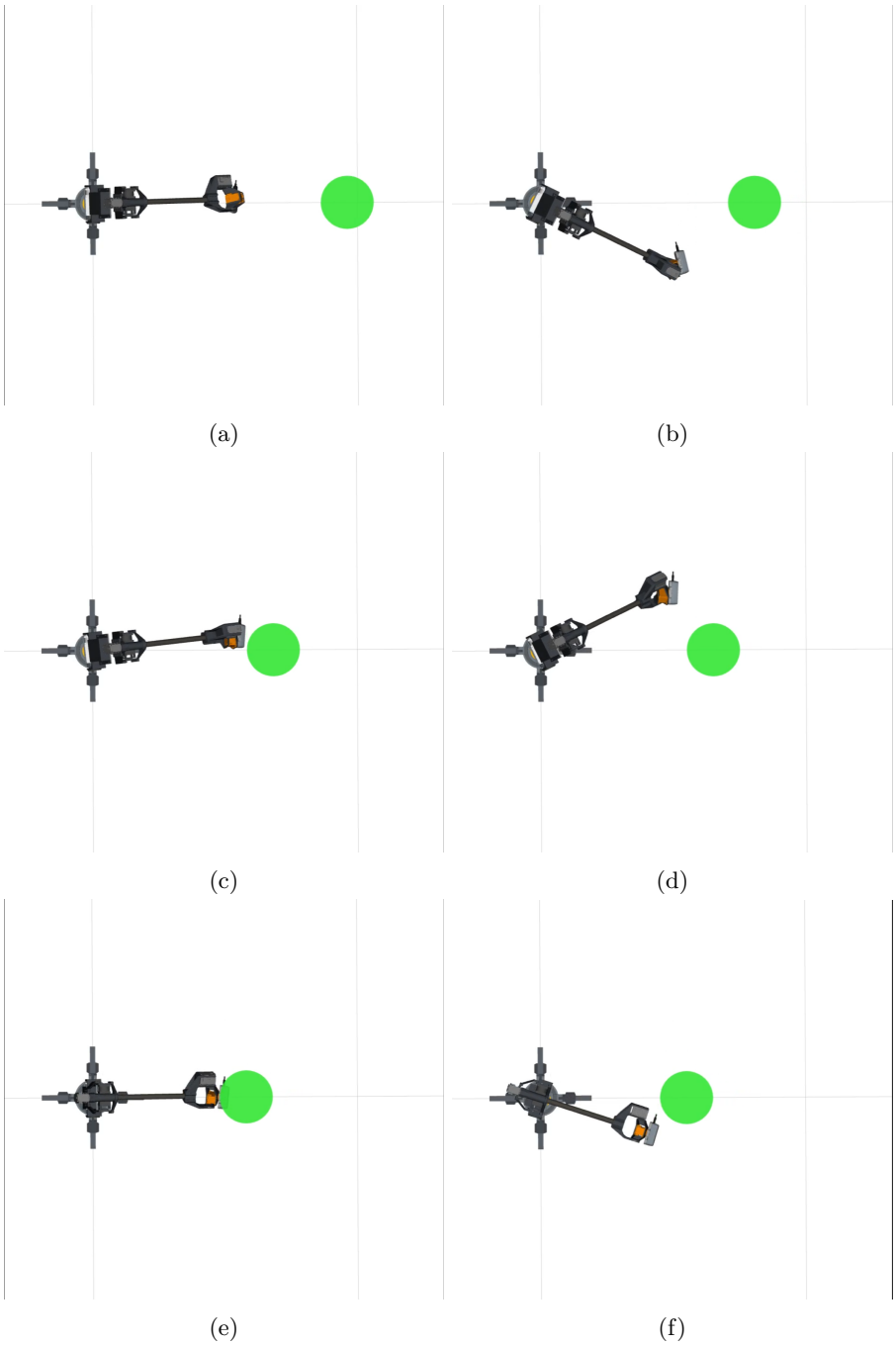
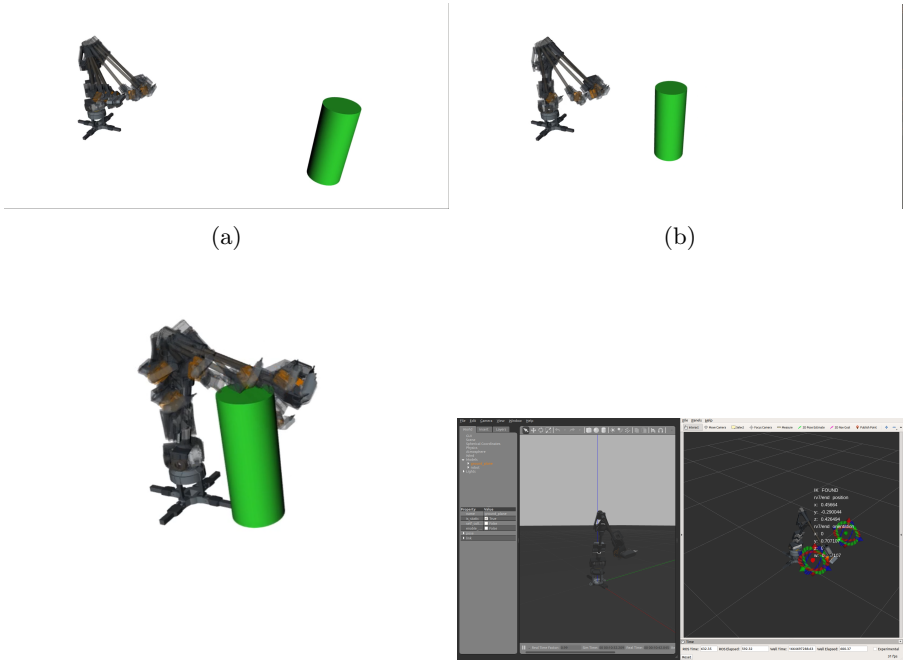


Figure 6: This sequence shows the manipulator follows an outdated trajectory and collides with the cylinder despite attempt to move away from the moving cyclinder.



(a) (b)

(c) (d)

Figure 7: The chronology of attempts at avoiding a moving obstacle when the obstacle approaches the robot. The planning algorithm fails at avoiding the cylinder before it passes the $x - location$ of the poses, c_{intial} and c_{goal}. (c) shows the planner successfully provide a non-colliding solution when the cylinder is moving away from robot. (d) shows the Gazebo as the physic engine to replicate the robot hardware and encoders feedback and the cyclical space initialization.

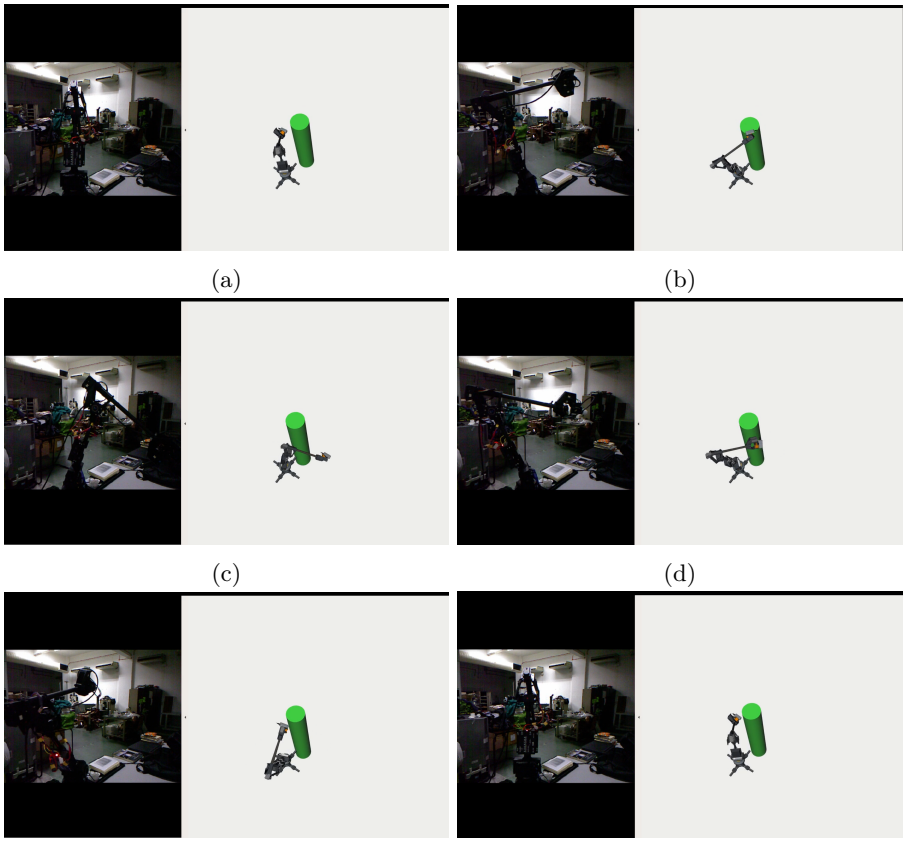


Figure 8: The sequence of motion when *r_mini* successfully avoid a moving obstacle when the obstacles at a turning point to move away from the hardware.

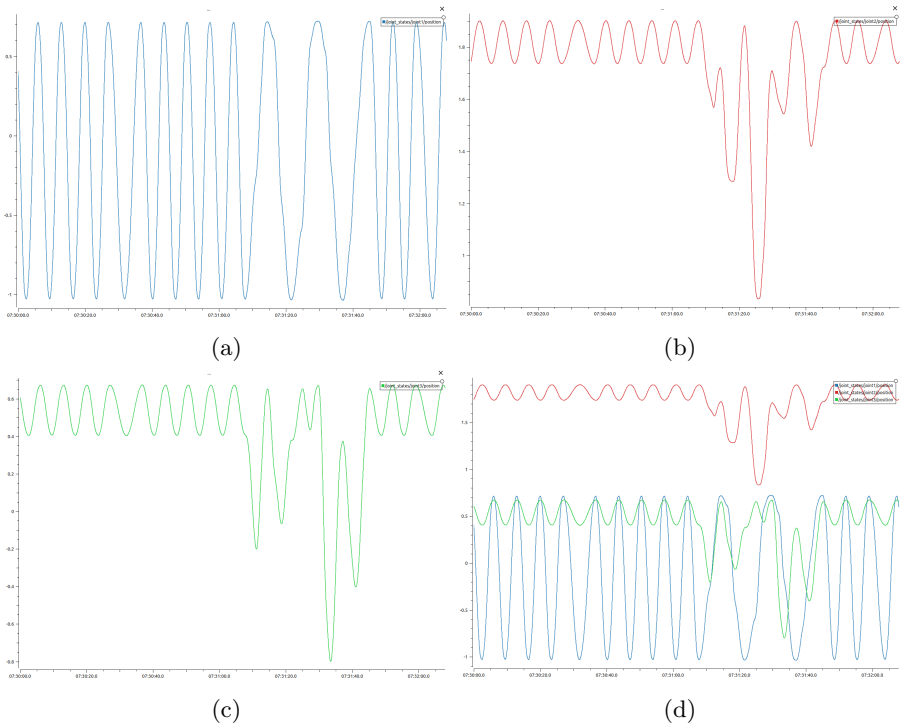


Figure 9: Reaction from $joint_1, joint_2, joint_3$ shows that the planner together with the cycle space behave reactively towards the moving object. No rapid movement or rate on the last three joints on *r_mini*

The planner shows reactive behavior when the cyclical space is initialize, via algorithm 3. .*Figure 7* illustrates such behavior in the simulated environment, and *figure 8* shows the same behavior in the hardware reiteration of the experimentation. This is illustrated in *figure 9*, where the $joint_2$ and $joint_3$ changes the range of their movement while $joint_3$ changes the rate of its movement.

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 rotation acting by these joints are not a linear transformation. Due to the offset (affine transformation) of the joints' axis of rotation, these joints' there is a bijection mapping of these joints to the task-space. Also changes are also observed on the orientation of the frame attached to the end-effector, however, there are no bijection mapping of the three joints to the task-space's orientation.

6 Conclusion and Recommendation

This concludes that, RRT on a dynamic setup is capable at reacting to an obstacle when the obstacle is moving,however, the RRT on a dynamic obstacle imposed under the cyclical space is not satisfactory. Given this reactive behavior for a repurposing a sampling based planner, it is recommended that this algorithm should be improved by including more than one intermediate poses between the initial pose and the goal pose. Also, an implementation on the tracking of the obstacle, by means of obstacle state estimation, would have helped the performance of the planner.