DEVELOPMENT OF A ROBOTIC ARM–GANTRY SIMULATOR WITH PROBABILISTIC INFERENCE BASED CONTROL

By

SHAHBAZ PEERAN QADRI SYED

Bachelor of Engineering in Mechanical Engineering
Osmania University
Hyderabad, India
2019

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2022

DEVELOPMENT OF A ROBOTIC ARM–GANTRY SIMULATOR WITH PROBABILISTIC INFERENCE BASED CONTROL

Dissertation Approved:

Dr. He Bai

Dissertation Advisor and Committee Chair

Dr. Brian Elbing

Committee Member

Dr. Imraan Faruque

Committee Member

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. He Bai, for his time and patience throughout this thesis. His guidance and support helped me immensely in navigating through this research. It has been a valuable experience working with him. I have learned a lot and will continue to learn from him in the future.

I would also like to thank my committee members, Dr. Brian Elbing and Dr. Imraan Faruque, for agreeing to serve on my committee. Additionally, I would also like to thank Dr. Yujiang Xiang for the opportunity to collaborate and work on the Human-Robot Lifting experiment project.

The work reported in this thesis was partially funded by the tow-tank project at Oklahoma State University.

Lastly, I would like to thank my family for their continued support and patience over the years. I want to express my gratitude to everyone who were directly or indirectly, involved in this research for their support and encouragement.

Acknowledgments reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

Name: SHAHBAZ PEERAN QADRI SYED

Date of Degree: MAY 2022

Title of Study: DEVELOPMENT OF A ROBOTIC ARM–GANTRY SIMULATOR WITH PROBABILISTIC INFERENCE BASED CONTROL

Major Field: MECHANICAL AND AEROSPACE ENGINEERING

Abstract: Robotic arms can perform repetitive tasks with high speed, accuracy, and precision, making them highly suitable for applications that are monotonous or require a high degree of precision. Such applications include industrial applications, human-robot collaboration applications, and experimental setups for model testing. A robust simulation as prototype testing is typically required to identify potential risks of catastrophic damage to the model or the experimental setup. Robot Operating System (ROS) is a widely-used framework for creating robotic applications both in research and in industry due to its easy hardware abstraction, code re-usability, and compatibility with popular open-source libraries. This thesis is focused on a robotic system consisting of a 7-DOF robotic arm ceiling mounted on a two-axis gantry. A simulator is developed for the 9-DOF robotic system in the ROS–Gazebo framework that interfaces with Moveit API for motion planning. A well-known Approximate Inference Control (AICO) algorithm is added to the simulator to extend its functionality to perform optimal trajectory planning. This work also extends the study of the AICO algorithm to non-holonomic mobile robots. Simulation experiments are conducted on both systems to study the behavior, performance, and limitations of AICO.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

II. PRELIMINARIES 6

2.1 Stochastic Optimal Control 6

2.2 Markov Decision Process (MDP) 7

2.3 Probabilistic Graphical Models (PGM) 8

2.4 Bayes rule and inference on graphs 9

2.4.1 Message passing 10

III. METHODOLOGY 13

3.1 Review of the Approximate Inference Control (AICO) 13

3.2 Review of the constrained - AICO 21

3.3 State extension approach 23

IV. DEVELOPMENT OF THE ROBOTIC ARM-GANTRY SIMULATOR 25

4.1 Robot Operating System (ROS) 25

4.1.1 ROS Control 25

4.1.2 Moveit 26

4.2 Robotic arm-gantry simulation 27

4.2.1 Development of ROS-Gazebo framework 28

4.2.2 Configuration of Moveit for the arm-gantry system 31

4.2.3 Integration of Gazebo and Moveit 33

Chapter Page

4.2.4 Integration of AICO and the simulator 35

V. RESULTS 38

5.1 Robotic arm-gantry system 38

5.1.1 Problem formulation 38

5.1.2 Implementation details of AICO 39

5.1.3 Results 41

5.2 Unicycle model 50

5.2.1 Problem formulation 50

5.2.2 Implementation details of AICO 56

5.2.3 Results 60

VI. CONCLUSION AND FUTURE WORK 71

REFERENCES 73

APPENDICES 76

0.1 Approximate Inference Control (AICO) algorithm 76

0.2 Product of multivariate Gaussians 77

LIST OF FIGURES

Figure		Page
1	Types of Probabilistic Graphical Models. a) Bayesian Network b) Markov Network c) Factor Graph.	9
2	Inference on a Bayesian Network.	11
3	Graphical Model representation of the SOC in Section 3.1	16
4	Overview of ROS Control architecture [3]	26
5	Moveit pipeline [Source: *https://moveit.ros.org*]	27
6	CAD model of the Robotic arm-gantry system.	28
7	Front view of the Robotic arm-gantry system.	28
8	Top view of the Robotic arm-gantry system.	29
9	URDF of the x-axis of the gantry.	30
10	Kinematic chain of the arm-gantry system.	31
11	A snippet of the control configuration file for the arm-gantry system.	32
12	Gazebo simulation of the arm-gantry system.	33
13	The graphical user interface of the Moveit setup assistant.	34
14	Moveit interface of the robotic arm-gantry system.	34
15	Node graph showing the active ROS nodes and ROS topics of the simulation setup.	36
16	All the ROS nodes and ROS topics of the simulation setup.	37
17	The trajectory for target reaching in the Gazebo simulation.	41
18	End-effector trajectory for the target reaching problem start = [0.25,-0.25,3.1415,0.9494,0,2.3,0,-1.5,0] rad; goal = [2.73,-2.11,1.55] m.	42
19	Trajectory cost over iterations for the target reaching problem.	42
20	The trajectory for target reaching problem with non-uniform control costs in the Gazebo simulation.	43
21	Comparison of the end-effector trajectories for uniform and non-uniform control costs start = [0.25,-0.25,3.1415,0.9494,0,2.3,0,-1.5,0] rad; goal = [2.73,-2.11,1.55] m.	44
22	Comparison of the joint position trajectories of the XY-gantry for uniform and non-uniform control costs.	44
23	Comparison of the joint position trajectories of the robotic arm for uniform and non-uniform control costs.	46
24	The configuration of arm-gantry system at various timesteps of the trajectory for the target reaching problem using RRT algorithm.	47

Figure Page

25 The end-effector trajectories of the arm-gantry system using RRT and AICO algorithms for the target reaching problem. 48
26 The trajectory for sinusoidal trajectory tracking in the Gazebo simulation. 49
27 End-effector trajectory for sinusoidal trajectory tracking. 50
28 Comparison of reference and tracked sinusoidal trajectories. . . . 51
29 The trajectory for elliptical trajectory tracking in the Gazebo simulation. 52
30 End-effector trajectory for elliptical trajectory tracking. 53
31 Comparison of reference and tracked elliptical trajectories. 54
32 The configuration of arm-gantry system at various timesteps of the trajectory for elliptical trajectory tracking using RRT algorithm. . 55
33 Comparision of the end-effector trajectories of the arm-gantry system for elliptical trajectory tracking using AICO and RRT algorithms. 56
34 Comparision of the tracking error of the trajectories generated using AICO and RRT algorithms for elliptical trajectory tracking. . 57
35 xy-trajectory of unicycle model for the target reaching problem start = [-100,20,0,100]; goal = [-10,30] m. 61
36 The linear velocity (v = 100 m/s) of the unicycle model for the target reaching problem. 62
37 Angular velocity ($\omega \in [-3.14, 3.14]$ rad/s)of unicycle model for the target reaching problem. 62
38 xy-trajectory of unicycle model for the target reaching problem start = [-100,20,0,100]; goal = [0,0] m. 63
39 The linear velocity (v = 100 m/s) of the unicycle model for the target reaching problem. 63
40 Angular velocity ($\omega \in [-3.14, 3.14]$ rad/s) of unicycle model for the target reaching problem. 64
41 xy-trajectories in various obstacle-filled environments. 66
42 xy-trajectory in an obstacle-filled environments. 67
43 Angular velocity for the trajectory in Fig. 42 67
44 xy-trajectories with and without disturbance for $\mu = 0$ and $\gamma = -5$ m/s. 68
45 xy-trajectories with and without disturbance for $\mu = 2.5$ m/s and $\gamma = 2.5$ m/s. 68
46 xy-trajectories with (μ = 2.0 m/s and γ = 0 m/s) and without disturbance in an obstacle filled environment. 69
47 Angular velocity with (μ = 2.0 m/s and γ = 0 m/s) and without disturbance in an obstacle filled environment. 70

48 The Approximate Inference Control algorithm [22]. 76

CHAPTER I

INTRODUCTION

In the present work, a 9-DOF Robotic arm-gantry system is considered. It consists of a 7-DOF Motoman SIA10D robotic arm ceiling mounted on a 2-DOF XY-gantry. This arm-gantry system is designed as a towing mechanism for the wavemaker facility being setup at Oklahoma State University. The motivation for the design is to leverage the wide range of motion of the articulated robot (SIA10D) for model handling while having a mobile base (XY- gantry) that can traverse the entire length of the tank. In general, prototype testing is not advisable in such huge systems to avoid catastrophic failures. Hence, a simulation of the arm-gantry system was created in the ROS-Gazebo environment to test and visualize the motion plans before interfacing with the actual hardware. The simulator was coupled with Moveit to perform motionplanning. The capabilities of the simulator were further enhanced to perform stochastic optimal control using probabilistic inference.

Stochastic Optimal Control (SOC) has been a topic of interest for the past few decades due to its applications across several disciplines such as robotics, finance, and biomechanics. The objective of SOC is to solve the traditional optimal control problem in the presence of uncertainty. In the context of robotics, this uncertainty is either in the form of noisy observations or as the process noise that captures model uncertainties in the system. In practice, this uncertainty is assumed to have a known distribution such as a Gaussian distribution.

Filtering is the estimation of the time-dependent state of the system having ob-

served some noisy measurements. Filtering performed with the notion of some statistic optimality is called optimal filtering. Another form of filtering is Bayesian Filtering which uses the Bayesian paradigm for state estimation. Bayesian Smoothing is another important class of estimation methods that are used to reconstruct the smoothed estimates of states that occurred before the current time [19]. In contrast, the Bayesian filtering methods tend to estimate the current state based on the history of measurements. A wide range of algorithms has been developed in the past few decades for estimation for various applications such as Kalman Filter [7], Extended Kalman Filter, Unscented Kalman Filter [6], Particle Filter [5, 11], Unscented Particle Filter [23], Rao–Blackwellized Particle Filter [12], etc., and their smoothing counterparts.

The Kalman filter and the Linear Quadratic Regulator (LQR) are the two most fundamental paradigms in estimation and optimal control theories respectively. The most straightforward connection between the two was proposed in the Kalman duality [7, 20] which states that the Kalman filter for a linear time-varying stochastic system is equivalent to the LQR of its dual system [9]. It has been shown that there is a one-to-one correspondence between the two paradigms. But, the main limitation of this duality is that it generalizes poorly beyond the Linear Quadratic Gaussian (LQG) case. Hence, a more general duality between optimal control and estimation was proposed in [20] that replaces the Kalman/ Kalman-Bucy filter in Kalman duality with an information filter and relates the probabilities to exponential costs rather than a one-to-one correspondence. The general duality was proven to be applicable to a wide range of problems such as systems with non-linear dynamics and non-linear measurements, systems with non-quadratic costs, discrete-time and continuous-time systems, and deterministic optimal control problems.

The general duality and in particular the notion of exponentiated costs to model likelihood has motivated a new class of methods for solving SOC problems called the

'Control as Inference' methods. Some of the algorithms proposed in this area are ilqg [21], AICO [22] and its variants [15, 17, 18], I2C [24], etc.

The use of SOC with linearized dynamics has proven to be fruitful in designing optimal feedback control laws in high-dimensional non-linear systems. The iterative linear quadratic gaussian (*ilqg*) [21] performs iterative linearization of non-linear system dynamics around the current trajectory and uses the LQR paradigm to obtain update equations to compute locally optimal trajectory. This uses a linear feedback control law. The Approximate Inference Control (*AICO*) [22] uses a graphical model representation of the SOC problem. It then performs inference on the graphical model using message passing techniques analogous to the belief propagation. The update equations are derived assuming all the messages to be Gaussian. The algorithm iterates continuously through forward and backward passes until some convergence criterion is satisfied. It was shown that the update equations for the backward message are analogous to the Ricatti equations in the LQG setting. The AICO algorithm is limited by two assumptions: it assumes that the trajectory length is known and it doesn't account for the constraints on the control both of which are common considerations in real-world robotic applications. The *AICO-T* [15] algorithm was proposed to address the issue of trajectory length constraint by parametrizing time as a variable and using Expectation Maximization to simultaneously perform both temporal and spatial optimization. The *constrained - AICO* [17] algorithm was proposed to address the control constraint issue by modifying the control prior to include the upper and lower bounds on the control and prescribing a linear feedback controller to infer the optimal control distribution rather than inferring the optimal trajectory distribution as in AICO. More recently, the Input Inference for Control (I2C) [24] algorithm formulated the SOC problem as that of an input estimation problem. It performs Expectation-Maximization by treating the hyperparameters as latent variables and hence finding the parameters that maximize the expected log-likelihood

in the M-step. The E-step performs linear Gaussian message passing to find state and control distributions. It finally prescribes a linear gaussian controller to infer the optimal control distribution.

The idea of approximate inference was also connected to the field of Reinforcement Learning. ψ-Learning, Soft Q-learning are some of the algorithms developed in this direction. For a more detailed introduction to RL as inference, the interested readers are directed to [16, 10, 13, 14].

In the present work, only the AICO and the constrained - AICO algorithms are studied in detail. This thesis report is outlined as follows the preliminaries for the AICO algorithm are introduced in Chapter II, the AICO algorithm and the constrained - AICO algorithm are reviewed in detail with derivations for all the main results in Chapter III, the steps involved in the development of the Robotic arm - gantry simulator are described in detail in Chapter IV, the results of the simulation experiments performed on the robotic arm-gantry system and the unicycle model are presented in Chapter V.

The contributions of this thesis are as follows:

- Development of a Robotic arm-gantry simulator using ROS-Gazebo framework integrated with Moveit to test motion plans before interfacing with the actual hardware.

- Extending the motion planning functionality of the simulator with the addition of probabilistic inference-based control.

- Most of the literature at the time of this writing on AICO is on articulated and humanoid robots. The present work attempts to extend this study to robotic applications with prismatic joints and non-holonomic mobile robots.

- This work suggests a dual to the constrained AICO formulation by state extension of the AICO formulation to impose the control constraints in the former as

state constraints in the latter approach. Hence, enabling AICO to infer state trajectories that obey the control constraints.

CHAPTER II

PRELIMINARIES

2.1 Stochastic Optimal Control

Consider a dynamical system given by

$$x_{t+1} = x_t + \Delta t f(x_t, u_t) + d\xi_t, \tag{2.1.1}$$

where $x_t \in \mathcal{X}$ represents the state of the system at time t and $u_t \in \mathcal{U}$ represents the control given that the system is in state x_t. Let $c(x_t, u_t)$ be the cost incurred at time t. Then the optimal cost-to-go function is given by

$$J(x_t) = \min_u \left[c(x_t, u_t) + \mathbb{E}[J(x_{t+1}, u_{t+1}])] \right]. \tag{2.1.2}$$

The general solutions to this problem are confined to a small class of systems [1]. But, if the system dynamics are linear, the states evolve by an additive white gaussian noise and the costs are quadratic then there exists a closed-form solution for the optimal control problem. The solution to this control problem is the optimal feedback control that minimizes the expected value of the quadratic cost criterion. This is well known as the Linear Quadratic Gaussian (LQG) case in the optimal control literature.

2.2 Markov Decision Process (MDP)

An MDP is a discrete-time stochastic control process that models the evolution of the system in a motion planning setting. It is the extension of the idea of the Markov chains but with the addition of a finite number of possible actions in each state and a reward that is incurred for every choice of the state-action pair. An MDP satisfies the Markov property which is a memoryless property of the system i.e. the next state of the system only depends on the current state and the current action and is independent of all of the previous states of the system.

In the context of the present work, we consider the following MDP. If a system is in state s and the decision policy chooses an action u then the system evolves into a new state s' with a state transition probability governed by the system dynamics $P(s'|s,u)$. Every such transition results in a reward $r(s,u)$ which acts as a quantifiable measure of the effectiveness of the state-action pair. Strictly speaking, the reward is a term used in the Reinforcement learning literature, In the context of optimal control, this measure is called cost $c(s,u)$. A distinguishable feature of the RL and optimal control paradigms is that the objective of the action policy in RL is to maximize the rewards whereas the objective of the control policy in optimal control is to minimize the cost. In the present work, we only consider finite horizon trajectories and hence we only consider finite-horizon MDPs with a linear Gaussian assumption. Hence, the state transition probability and the cost at any time t are given as follows

$$P(x_{t+1}|x_t,u_t) = \mathcal{N}(x_{t+1}|A_t x_t + a_t + B_t u_t, Q_t), \tag{2.2.1}$$

$$c(x_t,u_t) = \frac{1}{2}x_t^T R_t x_t - r_t^T x_t + \frac{1}{2}u_t^T H_t u_t. \tag{2.2.2}$$

2.3 Probabilistic Graphical Models (PGM)

The graphical representation of probability distributions is called a Probabilistic Graphical Model (PGM). Such representation of the model provides advantages such as visualization of the model, analysis of the conditional dependence of variables and facilitates the transformation of the complex mathematical computations into relatively simpler graphical computations [2]. A PGM consists of nodes and edges where each node represents a random variable and each edge represents the dependency between the variables. Hence, a joint distribution of all random variables can be decomposed into a product of factors based on the conditional independence properties of the PGM.

Generally, PGMs can be classified into three types given as follows

- **Bayesian Networks**: These models have directed edges indicated by an arrow from the independent to a dependent random variable. These are also called directed graphical models. For instance, the Bayesian Networks shown in Fig. 1(a) represents the joint distribution $P(x, y, z) = P(x)P(y)P(z|x, y)$. These types of graphical models are generally used to model causal relationships.

- **Markov Networks**: These models are distinguished by the undirected edges between the nodes. Hence, these are also called undirected graphical models or Markov Random Fields. The joint distribution is represented as the product of the potential functions $\psi_C(w_C)$ over the maximal cliques w_C of the graph. A maximal clique w_C is the maximum subset of nodes in the graph that are fully connected. For instance, the joint distribution of the undirected graphical model shown in Fig. 1(b) is given as $P(x, y, z) = \frac{1}{Z}\psi_C(w_C)$ where Z is the normalization constant (equal to 1 in this case) called the partition function and the maximal clique $w_C = \{x, y, z\}$.

- **Factor Graphs**: A Factor graph is a bipartite graph i.e. it consists of two types

of nodes one to represent the random variables and the other to represent the factors containing those random variables. An edge in a factor graph always connects two nodes of different types. A random variable node is connected to the factor node if and only if the factor depends on the random variable. These are an important and more general class of PGMs that are popularly used for inference. The joint distribution over variables is given as the product of factors in the factor graph. For instance, the joint distribution represented by the factor graph shown in Fig. 1(c) is given by $P(x, y, z) = f(x, y, z)$.

The three types of PGMs discussed above are almost always interchangeable. As the reader must have already noticed that all the three representations in Fig. 1 correspond to the same joint distribution over the relation $\psi_C(w_C) = f(x, y, z) = P(x)P(y)P(z|x, y)$. In the present work, Bayesian Networks are the choice of PGM used and hence, the discussion that follows will only be focused on Bayesian Networks. For a more detailed introduction to PGMs, the interested readers are referred to [2, 8].

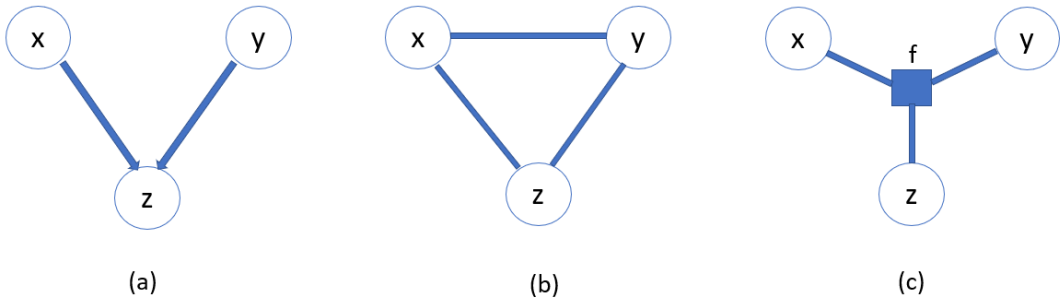


Figure 1: Types of Probabilistic Graphical Models. a) Bayesian Network b) Markov Network c) Factor Graph.

2.4 Bayes rule and inference on graphs

The underlying principle behind inference, in general, is one of the foundational concepts in probability and statistics called the Bayes theorem. It states that the proba-

bility of the joint distribution of two random variables is the product of the probability of the conditional distribution and the prior distribution of the independent variable. This can also be extended to distributions with more than two variables. Let x, y be two random variables with prior distributions $P(x)$ and $P(y)$ respectively then, according to Bayes theorem

$$P(x, y) = P(y|x)P(x) = P(x|y)P(y). \tag{2.4.1}$$

Inference in graphs deals with the computation of one or more subsets of nodes in a graph having observed some other nodes. Mathematically, it deals with the computation of the posterior distribution $P(\mathcal{X}|\mathcal{Y})$, where $\mathcal{X} = \{x_1, x_2, ..\}$ are the unobserved nodes in the graph or sometimes called as the latent variables and $\mathcal{Y} = \{y_1, y_2, ..\}$ are the observed nodes in the graph or observed variables. The inference is generally classified into two types based on the computational tractability of the posterior distribution as exact inference, approximate inference.

A common approach to performing inference on graphs is using local message passing. A message from a sender node to a receiver node is the marginalization of all the nodes connected to the sender node except the receiver node. This approach helps in reducing the computational complexity of computing marginals of the nodes by performing efficient computations exploiting the structure of the graph. The message passing can be used to derive recursive relationships thereby avoiding redundant computations. Finally, the marginal of a particular node is given as the product of all the incoming messages into the node.

2.4.1 Message passing

Consider the Bayesian Network as shown in Fig. 2. Let x_0, x_1, x_2, x_3 be the latent variables and z_0, z_1, z_2, z_3 be the observed variables in the graph. For instance, let the inference problem is to find the marginal of x_2 conditioned on the observed variables.

Then the posterior probability is given as

$$P(x_2|z_0, z_1, z_2, z_3) \propto P(x_2, z_0, z_1, z_2, z_3) \tag{2.4.2}$$

$$= \sum_{x_0,x_1,x_3} P(x_0, x_1, x_2, x_3, z_0, z_1, z_2, z_3). \tag{2.4.3}$$

From the structure of the graph in Figure 2, we further write (2.4.3) as

$$\sum_{x_0,x_1,x_3} P(x_0, x_1, x_2, x_3, z_0, z_1, z_2, z_3) = \sum_{x_0,x_1,x_3} P(x_0) \prod_{i=0}^{2} P(x_{i+1}|x_i) \prod_{i=0}^{3} P(z_i|x_i) \tag{2.4.4}$$

$$= \sum_{x_3} P(z_3|x_3)P(x_3|x_2)P(z_2|x_2) \sum_{x_1} P(x_2|x_1)P(z_1|x_1) \sum_{x_0} P(x_1|x_0)P(x_0)P(z_0|x_0). \tag{2.4.5}$$

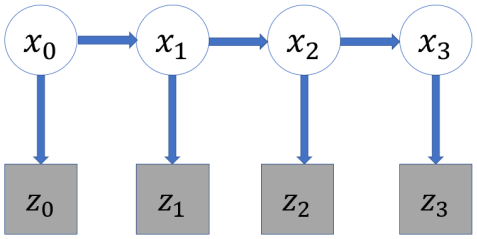


Figure 2: Inference on a Bayesian Network.

A message $\mu_{i\to j}(x_j)$ from i to j is defined as,

$$\mu_{i\to j}(x_j) = \sum_{x_i} f(x_i, x_j) \prod_{k\setminus i} \mu_{k\to i}(x_i) \tag{2.4.6}$$

Hence, substituting eq. (2.4.6) into (2.4.5) yields

$$P(x_2, z_0, z_1, z_2, z_3) = \mu_{x_3\to x_2}(x_2)\mu_{z_2\to x_2}(x_2) \sum_{x_1} P(x_2|x_1)P(z_1|x1)\mu_{x_0\to x_1}(x_1) \tag{2.4.7}$$

$$= \mu_{x_3\to x_2}(x_2)\mu_{z_2\to x_2}(x_2)\mu_{x_1\to x_2}(x_2) \tag{2.4.8}$$

Hence, the marginal distribution is given as the product of the messages coming into the node. This message passing technique is used in algorithms such as belief propagation, sum-product algorithm, loopy belief propagation, etc. The reason for the choice of this example, in particular, is that the AICO algorithm discussed in the present work uses the same graphical model as 2 and it uses the message passing technique for approximating the posterior distribution. The AICO algorithm is discussed in further detail in the Chapter III.

CHAPTER III

METHODOLOGY

In this chapter, the Approximate Inference Control (AICO) algorithm [22] and the constrained - AICO algorithm [17] are reviewed in detail. A state extension approach is also suggested to impose the control constraints as state constraints in the AICO algorithm thereby inferring state trajectories that account for the control constraints.

3.1 Review of the Approximate Inference Control (AICO)

Consider the discrete time controlled stochastic equation is given by :

$$\dot{\bar{x}} = f_t(\bar{x}_t, \bar{u}_t) + \xi_t, \tag{3.1.1}$$

where f_t is the non-linear state dependent dynamics, $\bar{x_t}$ is the zero-noise state, $\bar{u}$ is the open-loop control input, and $\xi \sim \mathcal{N}(0, Q_t)$ is the zero mean brownian motion. Using Euler forward method for discretization,

$$\bar{x}_{t+1} = \bar{x}_t + h f_t(\bar{x}_t, \bar{u}_t) \tag{3.1.2}$$

where h is a sufficiently small timestep.

Let x_t and u_t be the actual state and controls (with noise). Then, the state and

control deviations δx and δu are given by

$$\delta x = x_t - \bar{x}_t, \tag{3.1.3}$$

$$\delta u = u_t - \bar{u}_t. \tag{3.1.4}$$

Linearizing the dynamics in (3.1.2), we obtain

$$\delta x_{t+1} + \bar{x}_{t+1} = \delta x_t + \bar{x}_t + h f_t(\delta x_t + \bar{x}_t, \delta u_t + \bar{u}_t). \tag{3.1.5}$$

Subtracting the original dynamics (3.1.2) from (3.1.5) yields

$$\begin{aligned}
\delta x_{t+1} &= \delta x_t + h\big[\frac{\partial f_t}{\partial x_t}\delta x_t + \frac{\partial f_t}{\partial u_t}\delta u_t\big] && (3.1.6)\\
&= (I + h\frac{\partial f_t}{\partial x_t})\delta x_t + h\frac{\partial f_t}{\partial u_t}\delta u_t && (3.1.7)\\
&= A_t\delta x_t + B_t\delta u_t && (3.1.8)\\
x_{t+1} - \bar{x}_{t+1} &= A_t(x_t - \bar{x}_t) + B_t(u_t - \bar{u}_t) && (3.1.9)\\
x_{t+1} &= A_t x_t + B_t u_t + \bar{x}_{t+1} - \bar{x}_t - h\frac{\partial f_t}{\partial x_t}\bar{x}_t - h\frac{\partial f_t}{\partial u_t}\bar{u}_t && (3.1.10)\\
&= A_t x_t + B_t u_t + h f_t(\bar{x}_t, \bar{u}_t) - h\frac{\partial f_t}{\partial x_t}\bar{x}_t - h\frac{\partial f_t}{\partial u_t}\bar{u}_t && (3.1.11)\\
&= A_t x_t + a_t + B_t u_t && (3.1.12)
\end{aligned}$$

where

$$A_t = I + h\frac{\partial f_t}{\partial x_t}, \tag{3.1.13}$$

$$B_t = \frac{\partial f_t}{\partial u_t}, \tag{3.1.14}$$

$$a_t = h(f_t(\bar{x}_t, \bar{u}_t) - \frac{\partial f_t}{\partial x_t}\bar{x}_t - \frac{\partial f_t}{\partial u_t}\bar{u}_t). \tag{3.1.15}$$

The state transition probability is assumed to be a Gaussian distribution given

by

$$P(x_{t+1}|x_t, u_t) = \mathcal{N}(x_{t+1}|A_t x_t + a_t + B_t u_t, Q_t). \tag{3.1.16}$$

where $\mathcal{N}(x|a, A)$ represents a Gaussian distribution with mean μ and covariance Σ defined as

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{\det(2\pi\Sigma)^{\frac{1}{2}}} \exp((x-\mu)^T \Sigma^{-1}(x-\mu)) \tag{3.1.17}$$

The total trajectory cost $\mathcal{C}(\cdot)$ is given as

$$\mathcal{C}(x_{0:T}, u_{0:T}) = \sum_{t=0}^{T} c_t(x_t, u_t), \tag{3.1.18}$$

$$\text{where} \quad c_t(x_t, u_t) = \frac{1}{2} x_t^T R_t x_t - r_t^T x_T + u_t^T H_t u_t. \tag{3.1.19}$$

For approximate inference, consider a binary random task variable z_t with associated likelihood given as

$$P(z_t = 1|x_t, u_t) = \exp\{-c_t(x_t, u_t)\} \tag{3.1.20}$$

AICO [22] represents the above problem formulation given in eqns. (3.1.16) - (3.1.20) in the form of a Bayesian Network. The state x_t and the control u_t at time t are considered as latent variables of the model. The term z_t is an observed variable that corresponds to the task likelihood or optimality at time t. The probabilistic graphical model used to represent the above stochastic optimal control problem in the AICO formulation is shown in Fig. 3.

Consider the likelihood in eq. (3.1.20),

$$P(z_t = 1|x_t, u_t) = \frac{P((z_t = 1), x_t, u_t)}{P(x_t, u_t)} \tag{3.1.21}$$

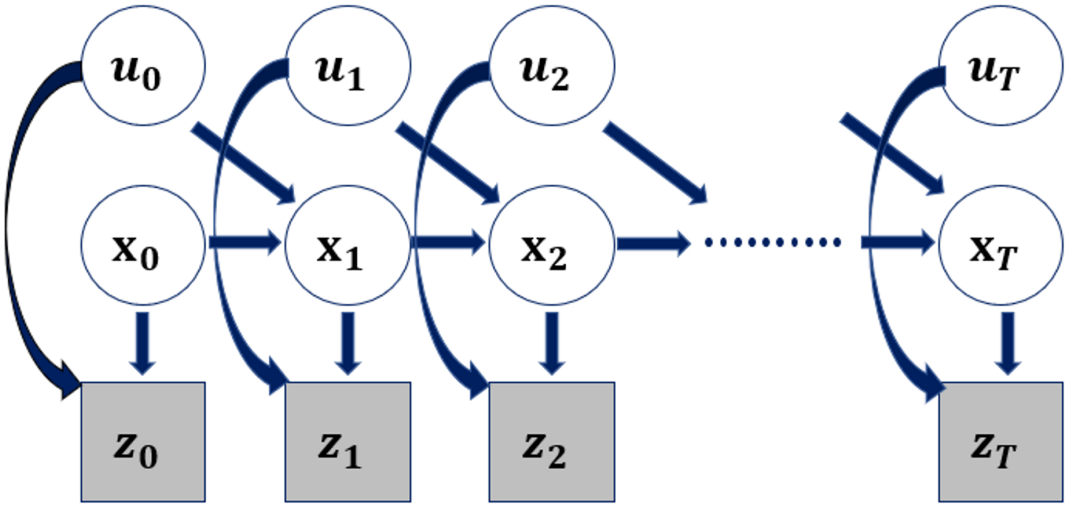


Figure 3: Graphical Model representation of the SOC in Section 3.1

Since the joint distribution of x_t, z_t is conditionally independent of u_t, eq. (3.1.21) can be rewritten as,

$$P(z_t = 1|x_t, u_t) = \frac{P((z_t = 1), x_t)P(u_t)}{P(x_t, u_t)} \tag{3.1.22}$$

$$\propto P(z_t = 1|x_t)P(u_t) \tag{3.1.23}$$

For LQG case, the cost is quadratic in x_t, u_t. Hence the eq. (3.1.23) can be written as

$$P(z_t = 1|x_t, u_t) = \exp\{-c_t(x_t, u_t)\} \tag{3.1.24}$$

$$= \exp(-\frac{1}{2}x_t^T R_t x_t + r_t^T x_T - u_t^T H_t u_t) \tag{3.1.25}$$

$$= \exp(-\frac{1}{2}x_t^T R_t x_t + r_t^T x_T)\exp(-u_t^T H_t u_t) \tag{3.1.26}$$

From eqns. (3.1.26) and (3.1.23) the likelihood can be rewritten as

$$P(z_t = 1|x_t, u_t) = \underbrace{\exp(-\frac{1}{2}x_t^T R_t x_t + r_t^T x_T)}_{\propto \mathcal{N}[x_t|r_t, R_t]} \underbrace{\exp(-u_t^T H_t u_t)}_{= \mathcal{N}[u_t|0, H_t]} \tag{3.1.27}$$

$$P(z_t = 1|x_t, u_t) \propto \mathcal{N}[x_t|r_t, R_t]\mathcal{N}[u_t|0, H_t] \tag{3.1.28}$$

Comparing with the eq. (3.1.23) yields

$$P(z_t = 1|x_t) \propto \mathcal{N}[x_t|r_t, R_t] \tag{3.1.29}$$

$$P(u_t) = \mathcal{N}[u_t|0, H_t] \tag{3.1.30}$$

where $\mathcal{N}[x|a, A]$ represents the Gaussian distribution in canonical form defined as

$$\mathcal{N}[x|a, A] = \frac{\exp(-\frac{1}{2}a^T A^{-1} a)}{|2\pi A^{-1}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}x^T A x + x^T a\right). \tag{3.1.31}$$

Marginalizing the controls in eq. (3.1.16) yields

$$P(x_{t+1}|x_t) = \int_u P(x_{t+1}|x_t, u_t)P(u_t)du_t \tag{3.1.32}$$

$$= \int_u \mathcal{N}(x_{t+1}|A_t x_t + a_t + B_t u_t, Q_t)\mathcal{N}[u_t|0, H_t]du_t \tag{3.1.33}$$

$$= \mathcal{N}(x_{t+1}|A_t x_t + a_t, Q_t + B_t H^{-1} B_t^T) \tag{3.1.34}$$

For the prior and likelihood given in eqns. (3.1.34), (3.1.30), consider the posterior distribution $P(x_{0:T}|z_{0:T} = 1)$ which represents the probability of a trajectory $x_{0:T}$ conditioned on the observing optimality $z_{0:T} = 1$ for the entire trajectory. The

posterior can be rewritten as

$$P(x_{0:T}|z_{0:T}=1) = P(x_0, x_1, x_2, ..., x_T|z_0=1, z_1=1, z_2=1, ..., z_T=1) \quad (3.1.35)$$

$$= \frac{P(x_{0:T}, z_{0:T}=1)}{P(z_{0:T}=1)} \quad (3.1.36)$$

$$= \frac{P(x_0)P(x_1|x_0)...P(x_T|x_{0:T-1}P(z_0|x_{0:T})....P(z_T|z_{0:T-1}, x_{0:T})}{P(z_{0:T}=1)} \quad (3.1.37)$$

From the graph shown in Fig. 2, z_t, x_{t+1} depend only on x_t, all the other variables are conditionally independent of each other. Hence, the eq. (3.1.37) yields

$$P(x_{0:T}|z_{0:T}=1) \propto P(x_0)P(x_1|x_0)P(x_2|x_1).....P(x_T|x_{T-1})P(z_0=1|x_0)....P(z_T=1|x_T) \quad (3.1.38)$$

$$\propto P(x_0)\prod_{t=0}^{T-1} P(x_{t+1}|x_t)\prod_{t=0}^{T} P(z_t=1|x_t) \quad (3.1.39)$$

$$\propto P(x_0)\prod_{t=0}^{T-1} \mathcal{N}(x_{t+1}|A_t x_t + a_t, Q_t + B_t H^{-1} B_t^T)\prod_{t=0}^{T} \mathcal{N}[x_t|r_t, R_t] \quad (3.1.40)$$

The posterior in eq. (3.1.40) is computationally intractable. Hence, AICO approximates the above posterior using the message passing approach as discussed in Section 2.4.1.

The messages are assumed to be Gaussian distributions of the form

$$\textbf{Forward message} : \mu_{x_{t-1}\to x_t}(x_t) = \mathcal{N}(x_t|s_t, S_t), \quad (3.1.41)$$

$$\textbf{Backward message} : \mu_{x_{t+1}\to x_t}(x_t) = \mathcal{N}(x_t|\nu_t, V_t), \quad (3.1.42)$$

$$\textbf{Task message} : \mu_{z_t\to x_t}(x_t) = \mathcal{N}[x_t|r_t, R_t]. \quad (3.1.43)$$

Using a similar message passing approach as discussed in Section 2.4.1, the messages

can be derived in recursive form given as

Forward message:

$$\mu_{x_{t-1}\to x_t}(x_t) = \mathcal{N}(x_t|s_t, S_t) \tag{3.1.44}$$

From the definition of message given in eq. (2.4.6)

$$\mu_{x_{t-1}\to x_t}(x_t) = \int_{x_{t-1}} dx_{t-1} P(x_t|x_{t-1})\mu_{x_{t-2}\to x_{t-1}}(x_{t-1})\mu_{z_{t-1}\to x_{t-1}}(x_{t-1})dx_{t-1}. \tag{3.1.45}$$

Solving which we obtain

$$s_t = a_{t-1} + A_{t-1}(S_{t-1}^{-1} + R_{t-1})^{-1}(S_{t-1}^{-1}s_{t-1} + r_{t-1}), \tag{3.1.46}$$

$$S_t = Q + B_{t-1}H^{-1}B_{t-1}^T + A_{t-1}(S_{t-1}^{-1} + R_{t-1})^{-1}A_{t-1}^T. \tag{3.1.47}$$

Backward message:

$$\mu_{x_{t+1}\to x_t}(x_t) = \mathcal{N}(x_t|\nu_t, V_t) \tag{3.1.48}$$

From the definition of message given in eq. (2.4.6)

$$\mu_{x_{t+1}\to x_t}(x_t) \propto \int_{x_{t+1}} dx_{t+1} P(x_{t+1}|x_t)\mu_{x_{t+2}\to x_{t+1}}(x_{t+1})\mu_{z_{t+1}\to x_{t+1}}(x_{t+1})dx_{t+1} \tag{3.1.49}$$

Solving which yields

$$\nu_t = -A_t^{-1}a_t + A_t^{-1}(V_{t+1}^{-1} + R_{t+1})^{-1})(V_{t+1}^{-1}\nu_{t+1} + r_{t+1}), \tag{3.1.50}$$

$$V_t = A_t^{-1}[Q + B_t H^{-1}B_t^T + (V_{t+1}^{-1} + R_{t+1})^{-1}]A_t^{-T}. \tag{3.1.51}$$

Task message:

Let $y_i \in \mathbb{R}^{p\times 1}$ represent a task variable of the system such as the end-effector position

of a robotic arm, measure of collision danger etc. and $\phi_i(x) : x \to y_i$ be a map from the state space to the task space. Let J_i denote the jacobian of the function $\phi_i(\cdot)$ for a given state x. Then for n task variables, the task message is given as,

$$\mu_{z_t \to x_t}(x_t) = P(z_t|x_t) = \mathcal{N}[x_t|r_t, R_t] \tag{3.1.52}$$

where,

$$r_t = \sum_{i=1}^{n} \rho_{i,t} \hat{J}_i^T (y_{i,t}^* - \phi_i(\hat{x}_t) + \hat{J}_i \hat{x}_t), \tag{3.1.53}$$

$$R_t = \sum_{i=1}^{n} \rho_{i,t} \hat{J}_i^T \hat{J}_i, \tag{3.1.54}$$

in which $\rho_{i,t}$ are time-dependent precision values, $y_{i,t}^*$ are the task space targets for $t \in (0, T]$, $\hat{x}_t$ is the estimate of x_t and $\hat{J}_i = J_i(\hat{x}_t)$.

Belief distribution:

The posterior distribution is approximated in AICO by computing the belief distribution as in graphical model inference but using the messages described above. The mean of the belief distribution gives the optimal state at each time t. The belief distribution is computed as

$$b(x_t) = \mu_{x_{t-1} \to x_t}(x_t) \mu_{x_{t+1} \to x_t}(x_t) \mu_{z_t \to x_t}(x_t) \tag{3.1.55}$$

$$= \mathcal{N}[b_t, B_t] \tag{3.1.56}$$

where

$$b_t = S_t^{-1} s_t + V_t^{-1} v_t + r_t, \tag{3.1.57}$$

$$B_t = S_t^{-1} + V_t^{-1} + R_t. \tag{3.1.58}$$

3.2 Review of the constrained - AICO

The AICO provides an elegant way to solve the trajectory optimization problem for a stochastic system. But, one of the major limitations of AICO is that the control prior is modeled as a zero-mean Gaussian noise which is marginalized later. Hence, the controls do not directly contribute to the trajectory inference. Hence, the AICO formulation infers trajectories that do not account for the control constraints. To address this issue the constrained-AICO [17] algorithm extends the AICO formulation by modifying the control prior to having a non-zero mean which captures the control constraints. The control cost is modified in the constrained-AICO formulation as

$$P(u_t) = c'(u_t) = u_t^T H_t u_t + (u_t - u_{max})^T H_t^U (u_t - u_{max}) + (u_t - u_{min})^T H_t^L (u_t - u_{min}) \tag{3.2.1}$$

$$= \mathcal{N}[h_t, \hat{H}_t] \tag{3.2.2}$$

where u_{max}, u_{min} are the maximum and minimum control limits respectively,
$h_t = u_{max}^T H_t^U + u_{min}^T H_t^L$,
$\hat{H}_t = H_t + H_t^U + H_t^L$.

To infer the controls, we consider that for a timestep t all the controls are marginalized except for u_t. Hence, the joint distribution is given as

$$P(x_{0:T}, u_t | z_{0:T}) \propto p(z_{0:T} | x_{0:T}, u_t) p(x_{0:T}, u_t) \tag{3.2.3}$$

$$= p(x_{t+1}|x_t, u_t) p(u_t) \prod_{l=0}^{T} p(z_l|x_l) \prod_{l=0}^{t-1} p(x_{l+1}|x_l) \prod_{l=t+1}^{T} p(x_{l+1}|x_l) \tag{3.2.4}$$

The joint distribution of x_t, u_t conditioned on all the observations $z_{0:T}$ is given as

$$p(x_t, u_t|z_{0:T}) = \int_0 \int_1 \cdots \int_{x_{t-1}} \int_{x_{t+1}} \cdots \int_T P(x_{0:T}, u_t|z_{0:T}) dx_0 dx_1 \cdots dx_{t-1} dx_{t+1} \cdots dx_T \tag{3.2.5}$$

Regrouping the terms, eq. (3.2.5) can be written as

$$\begin{aligned} &\propto \left[\int_0 \int_1 \cdots \int_{x_{t-1}} \prod_{l=0}^{t-1} p(x_{l+1}|x_l) \prod_{l=0}^{t-1} p(z_l|x_l) dx_0 dx_1 \cdots dx_{t-1} \right] \\ &\left[\int_{x_{t+1}} \int_{x_{t+2}} \cdots \int_T p(x_{t+1}|x_t, u_t) p(u_t) \prod_{l=t+1}^{T} p(x_{l+1}|x_l) \prod_{l=t+1}^{T} p(z_l|x_l) dx_{t+1} dx_{t+2} \cdots dx_T \right] \end{aligned} \tag{3.2.6}$$

Following the message-passing approach and using the notations, (3.1.41), (3.1.42), (3.1.43) yields

$$= \int_{x_{t+1}} \mu_{x_{t-1} \to x_t}(x_t) \mu_{z_t \to x_t}(x_t) p(x_{t+1}|x_t, u_t) p(u_t) \mu_{x_{t+2} \to x_{t+1}}(x_{t+1}) \mu_{z_{t+1} \to x_{t+1}}(x_{t+1}) dx_{t+1} \tag{3.2.7}$$

$$= p(x_t) p(u_t) \int_{x_{t+1}} p(x_{t+1}|x_t, u_t) \mu_{x_{t+2} \to x_{t+1}}(x_{t+1}) \mu_{z_{t+1} \to x_{t+1}}(x_{t+1}) dx_{t+1} \tag{3.2.8}$$

Using the Gaussian approximations of messages in eqns. (3.1.42), (3.1.43), we obtain

$$= p(x_t) p(u_t) \int_{x_{t+1}} p(x_{t+1}|x_t, u_t) \mathcal{N}(x_{t+1}|\nu_{t+1}, V_{t+1}) \mathcal{N}[x_{t+1}|r_{t+1}, R_{t+1}] dx_{t+1} \tag{3.2.9}$$

Using the result (A.12) we obtain

$$= p(x_t) p(u_t) \int_{x_{t+1}} p(x_{t+1}|x_t, u_t) \mathcal{N}[x_{t+1}|\bar{\nu}_{t+1}, \bar{V}_{t+1}] dx_{t+1} \tag{3.2.10}$$

where

$$\bar{\nu}_{t+1} = V_{t+1}^{-1}\nu_{t+1} + r_{t+1}, \tag{3.2.11}$$

$$\bar{V}_{t+1} = V_{t+1}^{-1} + R_{t+1}. \tag{3.2.12}$$

Using the Woodbury identity and simplifying the above expression we can obtain the distribution of $p(u_t|x_t)$ given as

$$p(u_t|x_t) = p(x_t, u_t)/p(x_t) \tag{3.2.13}$$

$$= \mathcal{N}\left(u_t|M_t^{-1}\left(B_t^T(Q_t + \bar{V}_{t+1}^{-1})^{-1}\left(\bar{V}_{t+1}^{-1}\bar{\nu}_{t+1} - A_t x_t - a_t\right)\right), M_t^{-1}\right) \tag{3.2.14}$$

where, $M_t = B_t^T(Q_t + \bar{V}_{t+1}^{-1})^{-1}B_t$

The mean of $P(u_t|x_t)$ can be written as an optimal feedback controller of the form

$$u_t = o_t + O_t x_t \tag{3.2.15}$$

$$\text{such that } o_t = M_t^{-1}\left(B_t^T(Q_t + \bar{V}_{t+1}^{-1})^{-1}\bar{V}_{t+1}^{-1}\bar{\nu}_{t+1} - B_t^T(Q_t + \bar{V}_{t+1}^{-1})^{-1}a_t\right) \tag{3.2.16}$$

$$O_t = -M_t^{-1}B_t^T(Q_t + \bar{V}_{t+1}^{-1})^{-1}A_t \tag{3.2.17}$$

$$\text{where } M_t = B_t^T(Q_t + \bar{V}_{t+1}^{-1})^{-1}B_t \tag{3.2.18}$$

3.3 State extension approach

In this work, we suggest an alternative to the constrained - AICO formulation. We suggest the state extension in AICO imposes the control constraints as state constraints for the trajectory inference. The two types of control constraints that are usually considered in robotic applications are equality constraints and a bounded constraints.

Equality constraint: An equality constraint is one in which the value of the

control is fixed to a preset value. By state extension we suggest that for each equality constraint $u = u_1$ the state space should be extended by an additional state $\dot{u} = 0$ and the corresponding u be removed from the control vector. The cost function can then be modeled to impose a high cost on the new state u for any deviation from u_1.

Bounded constraint: A bounded constraint is one in which the value of control is always bounded between an upper and lower limit i.e. $u_{min} < u < u_{max}$. By state extension we suggest that for each bounded constraint, the state should be extended by an additional state $\dot{\mathcal{F}} = u$ while keeping the control vector unchanged. The cost function can then be modeled to impose a cost if the value of $\mathcal{F}_{t+1} - \mathcal{F}_t > h\ u_{max}$ and $\mathcal{F}_{t+1} - \mathcal{F}_t < h\ u_{min}$ where h is step size for discretization.

The state constraints discussed above facilitate the inference of trajectories that obey the control constraints in the AICO setting even if the controls do not directly contribute to the trajectory inference. The state extension approach was utilized in the study of the unicycle model discussed in detail in Section 5.2 . It has proven to be effective in inferring trajectories that obey the control constraints as shown in Section 5.2.

CHAPTER IV

DEVELOPMENT OF THE ROBOTIC ARM-GANTRY SIMULATOR

4.1 Robot Operating System (ROS)

Robot Operating System(ROS) is the existing standard for creating robotic applications both in industry and research. It is an open-source suite of tools, libraries, and conventions that streamline and simplify the task of creating robotic applications. It allows the user to avoid building from scratch and rather focus on adapting the existing code to create more complex applications. The wide popularity and applicability of ROS are due to its easy hardware abstraction, code reusability, and compatibility with other open-source libraries and robotic platforms.

4.1.1 ROS Control

ROS Control [3] is one of the core packages of ROS that provides the control interface for robotic applications. It acts as a bridge between the ROS interfaces and the controllers/actuators in the simulated/real robot. It provides hardware interfaces, transmission interfaces, joint limits, and in-built controllers required to control the robot. It also comes with a set of libraries for creating custom controllers. The `joint_state_controller` and `effort_controllers/joint_trajectory_controller` were used in development of the Robotic arm-gantry simulation. An overview of the ROS Control architecture is shown in Fig. 4 . For a more detailed introduction to ROS Control, interested readers are directed to [3], and the official documentation on ROS Wiki.

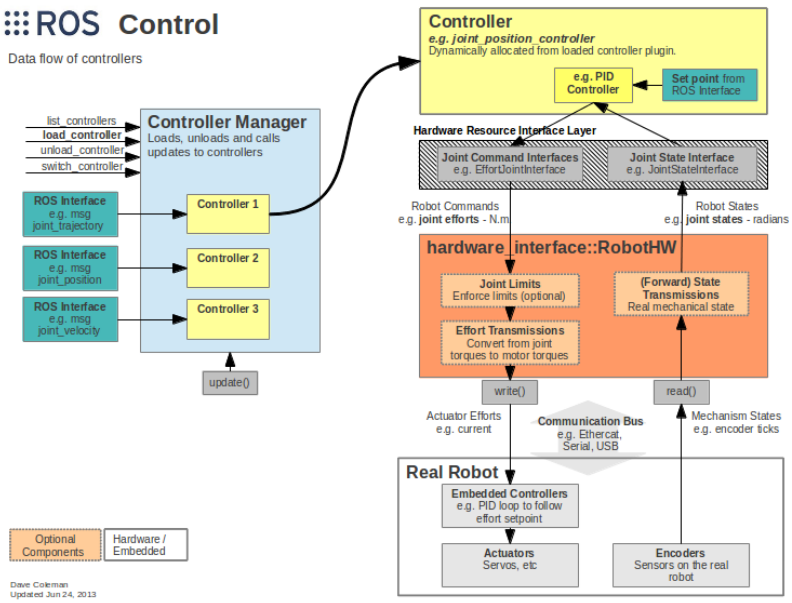


Figure 4: Overview of ROS Control architecture [3]

4.1.2 Moveit

Moveit is the primarily used perception and motion planning framework in ROS. It is widely used due to its minimal barrier to entry [4], ease of use, and huge well-maintained repository of tools for perception and motion planning in complex robotic applications.

Moveit uses ROS build and messaging system at its core, and plugins for motion planning, collision checking, and kinematics. At an architectural level, the launch of a Moveit application starts a `move_group` node that integrates all the individual components and makes them available to the user through ROS actions and services. An overview of the Moveit architecture is shown in Fig. 5. Moreover, a key feature that distinguishes Moveit from other motion planning frameworks is the GUI setup assistant that can be used for the initial setup of the robot. It can auto-generate all the configuration files that contain information about the robot poses, planning groups, planning algorithms, disabled self-collision checking pairs, etc. This immensely simplifies and streamlines the process of initial setup of Moveit for any custom robot.

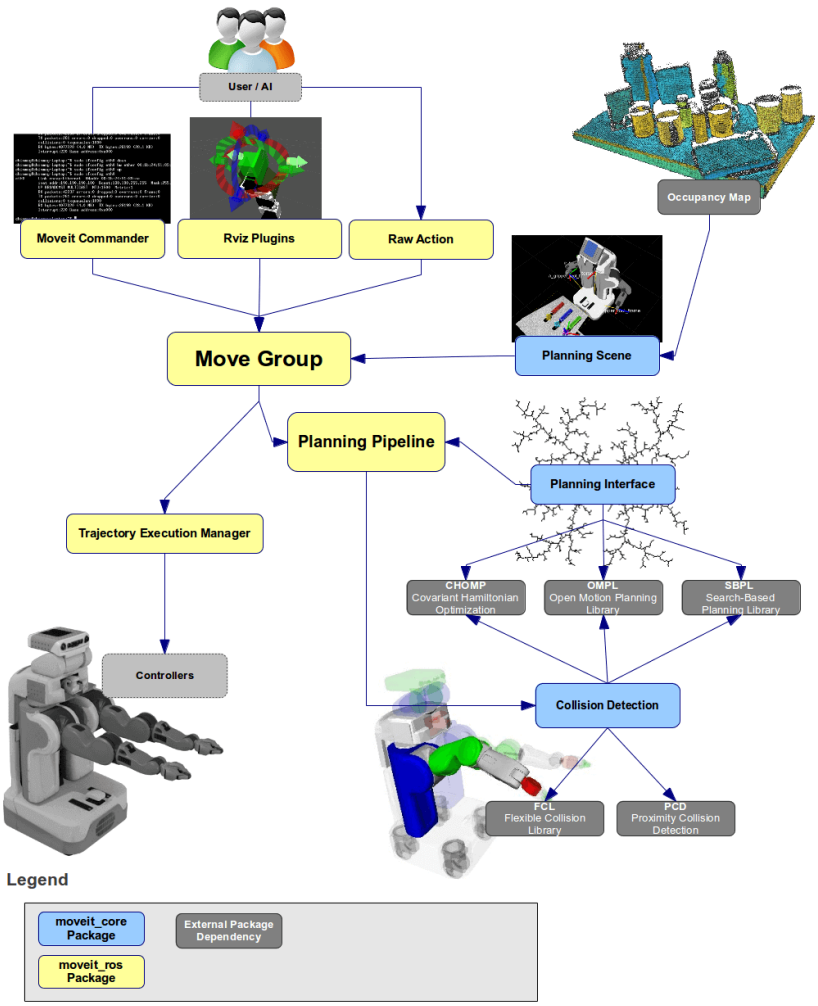


Figure 5: Moveit pipeline [Source: *https://moveit.ros.org*]

4.2 Robotic arm-gantry simulation

The development of simulation for the robotic arm-gantry system can be divided into three major steps as given below:

- Development of the ROS-Gazebo framework.
- Configuration of Moveit for the arm - gantry system.
- Integration of Gazebo and Moveit.

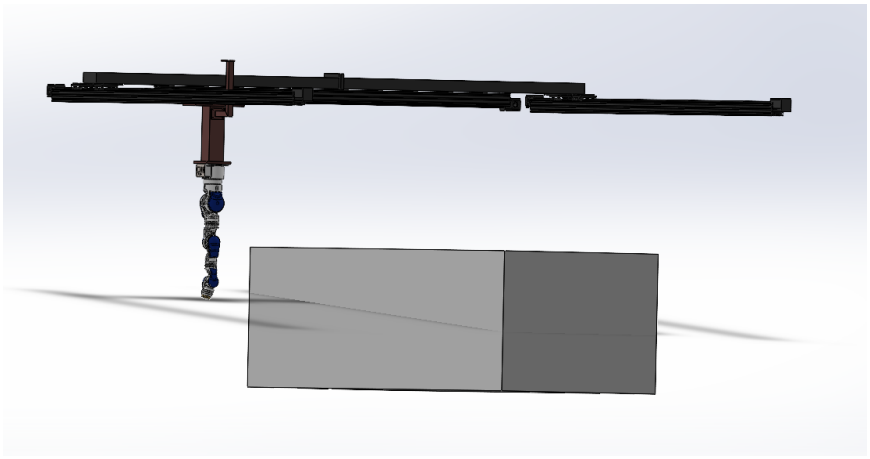


Figure 6: CAD model of the Robotic arm-gantry system.

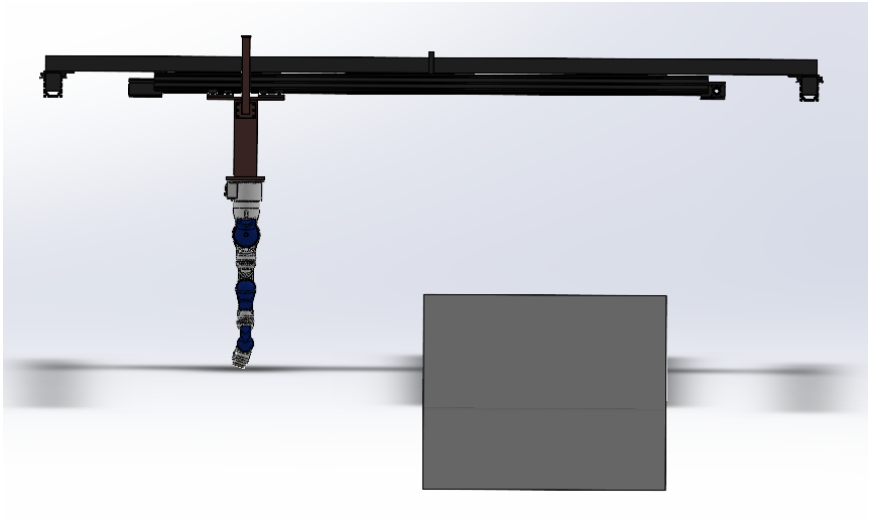


Figure 7: Front view of the Robotic arm-gantry system.

4.2.1 Development of ROS-Gazebo framework

1. **CAD model**: The first step for the development of any simulation is the creation of the 3D model of the robot. The CAD model of the arm-gantry system was created using Solidworks. The choice of Solidworks for modeling the system is two-fold: 1) It is easy to use 2) It supports the *SW2URDF* plugin that can auto-generate the URDF file from the Solidworks assembly. The Solidworks model of the arm-gantry system is shown in Fig. 6 - 8. The Solidworks model of the SIA10D arm was obtained from Yaskawa Motoman[1].

[1]https://www.motoman.com/en-us

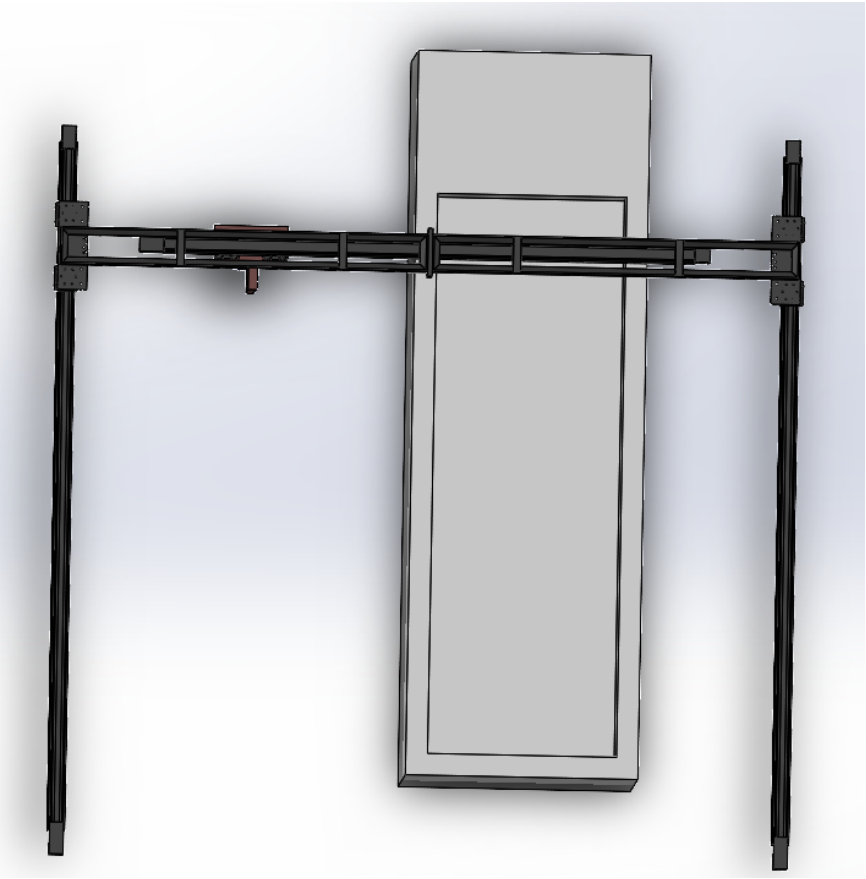


Figure 8: Top view of the Robotic arm-gantry system.

```
<link name="xystagebase_link">
  <inertial>
    <origin rpy="0 0 0" xyz="-2.4957 -2.5004 -0.031256"/>
    <mass value="76.947"/>
    <inertia ixx="126.91" ixy="6.262E-06" ixz="0.00018024" iyy="0.21414" iyz="-0.00083152" izz="126.92"/>
  </inertial>
  <visual>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://robot/meshes/xslides1.STL"/>
    </geometry>
    <material name="">
      <color rgba="0.29804 0.29804 0.29804 1"/>
    </material>
  </visual>
  <collision>
    <origin rpy="0 0 0" xyz="0 0 0"/>
    <geometry>
      <mesh filename="package://robot/meshes/xslides1.STL"/>
    </geometry>
  </collision>
</link>
<gazebo reference="xystagebase_link">
  <!--<turnGravityOff>false</turnGravityOff>-->
  <selfCollide>true</selfCollide>
  <material>Gazebo/silver</material>
</gazebo>
<joint name="xystagebase_link_to_world" type="fixed">
  <origin rpy="0 0 0" xyz="0 0 3"/>
  <parent link="world"/>
  <child link="xystagebase_link"/>
</joint>
```

```
<transmission name="transmission_x">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="xgantry_baselink">
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
  </joint>
  <actuator name="x_motor">
    <mechanicalReduction>1</mechanicalReduction>
    <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
  </actuator>
</transmission>
```

Figure 9: URDF of the x-axis of the gantry.

2. **Unified Robot Description Format (URDF)**: The auto-generated URDF file from step 1 captures all of the link and joint data but lacks the gazebo-specific information that is required to spawn and control the model in the gazebo environment. This is usually the information related to the location of the robot in the gazebo world, transmission interfaces for actuators, and additional plugins for sensors and control. The URDF of the SIA10D robot was adapted from the MotoROS Github repository[2] and modified as needed. A snippet of the URDF for the x-axis of the gantry is shown in Fig. 9. ROS also provides a tool called `check_urdf` to check the syntax and display the kinematic chain to the user. The kinematic chain for the arm-gantry system is shown in Fig. 10.

[2]https://github.com/ros-industrial/motoman

Figure 10: Kinematic chain of the arm-gantry system.

3. **Controllers**: At the end of step 2, the simulation can successfully spawn the robot model in the gazebo environment but cannot control the system. This functionality is achieved by configuring controllers for each of the joints using the `ros_control` package and tuning the PID gains for the controllers. A snippet of the control configuration file for the arm-gantry system is shown in Fig. 11.

This concludes the first stage of the development of the simulation. The gazebo simulation of the robotic arm-gantry system is shown in Fig. 12.

4.2.2 Configuration of Moveit for the arm-gantry system

The simulator at this stage has the arm-gantry system running in the gazebo environment with a controller configured to accept control commands and drive the actuators. But, the simulation is not yet equipped to handle the semantic description of the robot such as collision checking, joint limits, kinematics, sensor integration, planning groups, perception, and generation of motion plans which are key aspects of motion planning. These functionalities are added to the simulation using an external framework called Moveit discussed in Section 4.1.2. The Moveit package was created using the `moveit_setup_assistant` GUI. The graphical user interface of the Moveit setup assistant is shown in Fig. 13. The key advantage of using the setup assistant is that it auto-generates all of the required configuration files which can be manually modified later. The Moveit interface of the robotic arm - gantry system is shown in

```yaml
robot:
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  wavetank_arm_gantry_controller:
    type: effort_controllers/JointTrajectoryController
    joints:
      - xgantry_baselink
      - ytrackplate_xgantry
      - joint_s
      - joint_l
      - joint_e
      - joint_u
      - joint_r
      - joint_b
      - joint_t


    gains:
        xgantry_baselink: {p: 30000.0, i: 0, d: 8000.0}
        ytrackplate_xgantry: {p: 50000, i: 0.0, d: 1000}
        joint_s: { p: 100, d: 20, i: 0}
        joint_l: { p: 100, d: 30, i: 0}
        joint_e: { p: 100, d: 30, i: 0}
        joint_u: { p: 100, d: 30, i: 0}
        joint_r: { p: 100, d: 30, i: 0}
        joint_b: { p: 100, d: 30, i: 0}
        joint_t: { p: 1, d: 0.1, i: 0}
```

Figure 11: A snippet of the control configuration file for the arm-gantry system.

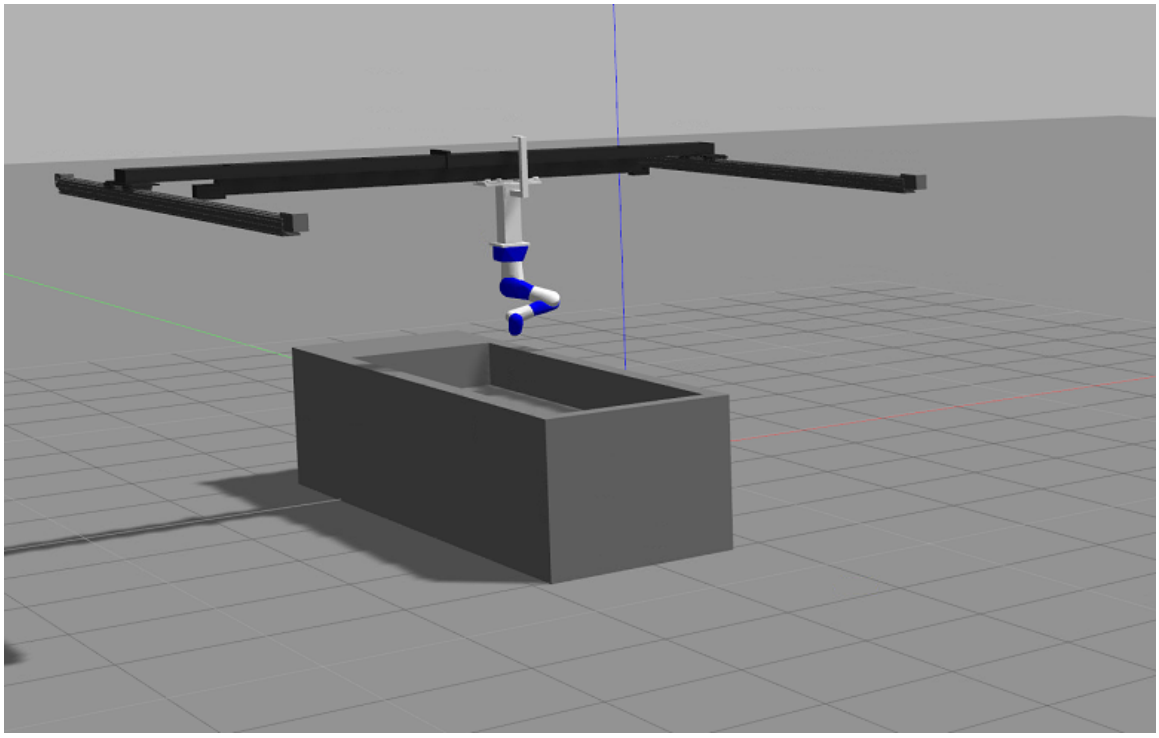


Figure 12: Gazebo simulation of the arm-gantry system.

.

Fig. 14.

4.2.3 Integration of Gazebo and Moveit

The simulator at this stage has two components the gazebo simulation which has the robot and controllers configured to command the actuators of the arm-gantry system, the Moveit ecosystem that can take a motion planning request and generate feasible motion plans without violating the task space constraints. But, both of these components are not connected yet and hence cannot communicate with each other. In this stage of the development process, the gazebo and Moveit are connected through ROS topics such that Moveit can receive the joint state data from the gazebo and generate motion plans to command the actuators using the effort controllers.

This step concludes the simulation of the robotic arm-gantry simulator. The simulator has the following functionalities

- It can take a start position and a goal position through the Moveit interface

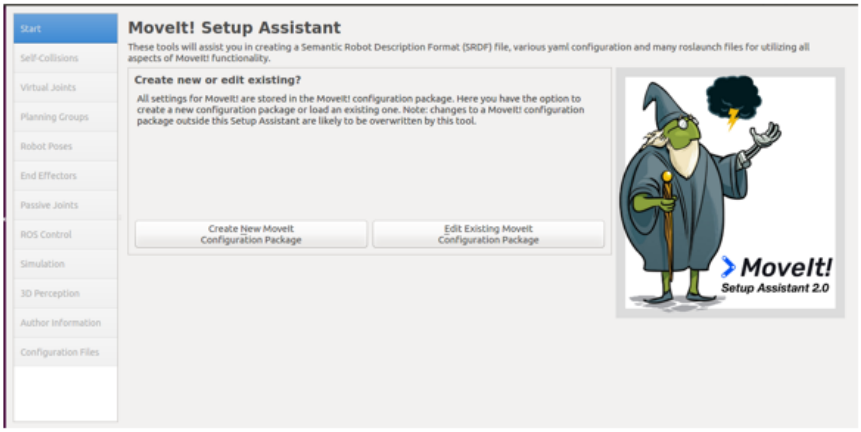


Figure 13: The graphical user interface of the Moveit setup assistant.

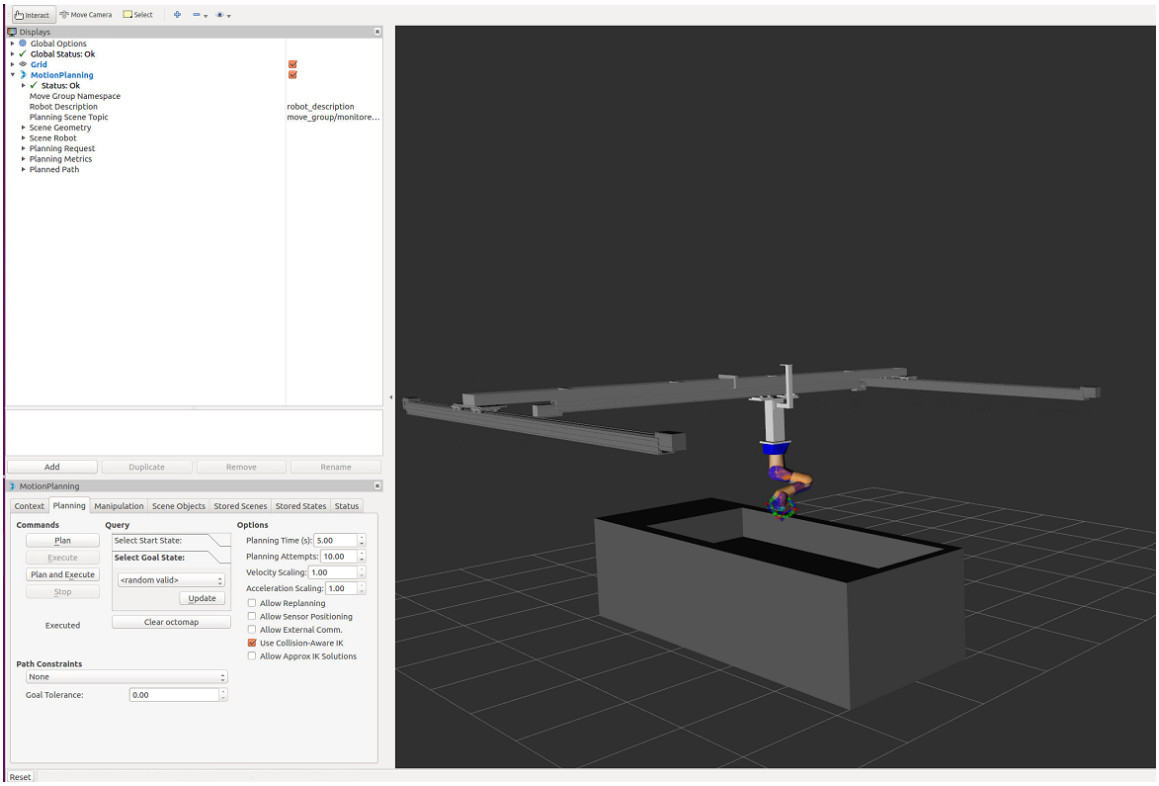


Figure 14: Moveit interface of the robotic arm-gantry system.

and then Moveit generates the motion plans and executes them in the gazebo simulation.

- The simulator can also be given custom user-defined trajectories through the `move_group python interface` to visualize them in the gazebo simulation.

The code for the simulator can be found in our github repository[3].

4.2.4 Integration of AICO and the simulator

The AICO [22] algorithm was studied on the 9-DOF Robotic arm - gantry system simulator discussed in Section 4.2.1. The simulation setup consists of a .py file that consists of the implementation of the AICO algorithm. This runs in parallel and interacts with the ROS-Gazebo simulation of the arm-gantry system. The .py file interacts with the simulation via Moveit commander to command the joint actuators in the Gazebo environment. The node graph and all nodes and topics of the simulation setup are shown in Fig. 15, 16 respectively.

[3]https://github.com/shahbazqadri/agsim.git

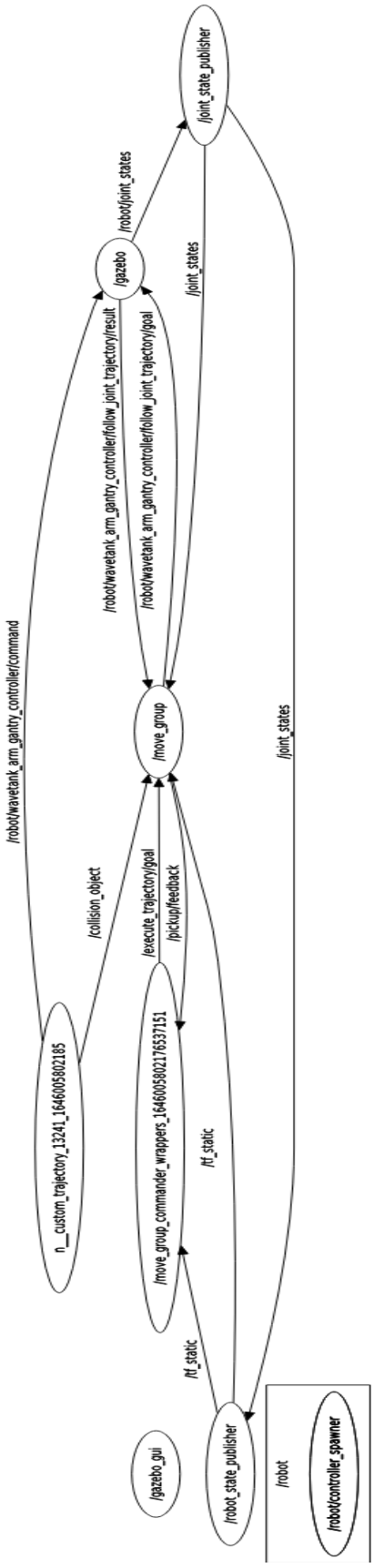


Figure 15: Node graph showing the active ROS nodes and ROS topics of the simulation setup.

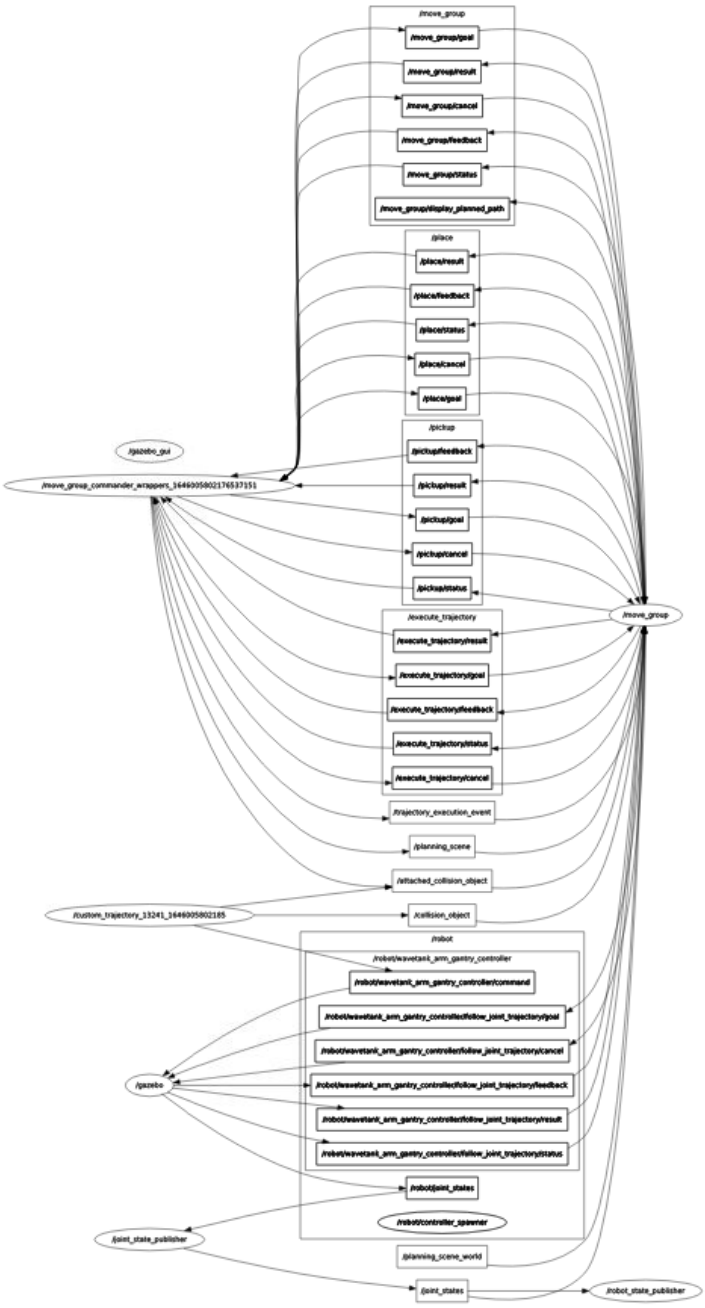


Figure 16: All the ROS nodes and ROS topics of the simulation setup.

CHAPTER V

RESULTS

5.1 Robotic arm-gantry system

5.1.1 Problem formulation

Consider, the linear discrete time dynamical system given by

$$q_{t+1} = q_t + hu_t + d\xi_t, \tag{5.1.1}$$

$$y_{1,t} = \phi_1(q_t) = \text{fwd\_kin}(q_t). \tag{5.1.2}$$

where $q_i \in \mathbb{R}^{9\times1}$ are the joint positions at the i^{th} timestep, $u_i \in \mathbb{R}^{9\times1}$ are the joint velocities at the i^{th} timestep, $y_{1,i} \in \mathbb{R}^{3\times1}$ is the task space variable that denotes the end-effector position at the i^{th} timestep, and $\phi_1(\cdot) : q \to y$ is the mapping from the state space to the task space and fwd_kin$(\cdot)$ is the forward kinematics routine that computes the end-effector position based on the given joint positions. In this work, the forward kinematics and the manipulator Jacobian of the arm-gantry system are computed using the *PyKDL* library.

Comparing the eqns. (3.1.16) and (5.1.1) we identify

$$x_t = q_t \in \mathbb{R}^{9\times 1}, \tag{5.1.3}$$

$$A_t = \mathbb{I}_{9\times 9}, \tag{5.1.4}$$

$$a_t = \mathbb{O}_{9\times 1}, \tag{5.1.5}$$

$$B_t = \mathbb{I}_{9\times 9}, \tag{5.1.6}$$

$$Q_t = \langle d\xi_t d\xi_t^T \rangle \in \mathbb{R}^{9\times 9}, \tag{5.1.7}$$

$$H_t = \mathbb{I}_{9\times 9}. \tag{5.1.8}$$

where $\mathbb{I}_{m\times n}$ and $\mathbb{O}_{m\times n}$ represent identity and null matrices of size $m \times n$.

5.1.2 Implementation details of AICO

Let $T \in \mathbb{R}$ be the trajectory length for the motion planning problem. Then, using the notations in eqns. (5.1.3) - (5.1.8) the forward, backward and task messages for the AICO algorithm [22] shown in Appendix 0.1 are computed as follows

Forward message:

For a timestep $t \in (0, T]$ the forward message is computed as

$$s_t = a_{t-1} + A_{t-1}(S_{t-1}^{-1} + R_{t-1})^{-1}(S_{t-1}^{-1}s_{t-1} + r_{t-1}),$$
$$S_t = Q + B_{t-1}H^{-1}B_{t-1}^T + A_{t-1}(S_{t-1}^{-1} + R_{t-1})^{-1}A_{t-1}^T.$$

Backward message:

For a timestep $t \in (0, T\text{-}1]$ the backward message is computed as

$$\nu_t = -A_t^{-1}a_t + A_t^{-1}(V_{t+1}^{-1} + R_{t+1})^{-1})(V_{t+1}^{-1}\nu_{t+1} + r_{t+1}),$$
$$V_t = A_t^{-1}[Q + B_tH^{-1}B_t^T + (V_{t+1}^{-1} + R_{t+1})^{-1}]A_t^{-T}.$$

Task message:

Let J_1 be the Jacobian of the function $\phi_1(\cdot)$ which corresponds to manipulator Jacobian $J_1 \in \mathbb{R}^{3\times 9}$ in this case. Since we only consider one task variable y_1 that represents the end-effector position, the task message for AICO given in eq. (3.1.43) can be rewritten as

$$r_t = \rho_{1,t}\hat{J}_1^T(y_{1,t}^* - \phi_1(\hat{x}_t) + \hat{J}_1\hat{x}_t), \tag{5.1.9}$$

$$R_t = \rho_{1,t}\hat{J}_1^T\hat{J}_1 \tag{5.1.10}$$

where $y_{1,0:T}^*$ are the task-space targets for the entire trajectory. These are considered equal to goal in the target reaching case and equal to the reference end-effector trajectory in the case of reference trajectory tracking, $\rho_{1,0:T}$ are the precisions for the entire trajectory that represent the penalty of the deviation of $y_{1,0:T}$ from $y_{1,0:T}^*$. In the case of target reaching, a high precision is required only at the final timestep. Hence, we chose $\rho_{1,0:T-1} = 1e-3$, $\rho_{1,0:T} = 1e3$. For trajectory tracking, a high precision is required throughout the trajectory and hence the precision was chosen to be $\rho_{1,0:T} = 1e4$.

Belief:

The belief at the timestep $t \in (0, T)$ is computed as

$$b_t = S_t^{-1}s_t + V_t^{-1}v_t + r_t,$$

$$B_t = S_t^{-1} + V_t^{-1} + R_t.$$

The value of b_t corresponds to the optimal value of the joint positions at time t denoted as q_t.

5.1.3 Results

In the present work, AICO [22] was used to generate the trajectories of the robotic arm - gantry system for two planning problems: the target reaching and reference trajectory tracking.

Target reaching problem

The target reaching problem refers to the problem of trajectory generation given a start state, goal state, and trajectory length while minimizing some cost function. In this simulation, the cost function penalizes the error between the final state and the goal. The arm-gantry system was tested for a set of start and goal positions with uniform and non-uniform control costs. The visualization of the trajectories, end-effector trajectories, and the total trajectory costs for both cases are shown and discussed below.

a) Uniform control cost

In the first experiment, the target reaching problem is considered with uniform control cost on all the joints. The start (joint) and goal (end-effector) positions for this simulation are as follows

start = [0.25,-0.25,3.1415,0.9494,0,2.3,0,-1.5,0] rad and the goal = [2.73,-2.11,1.55] m.

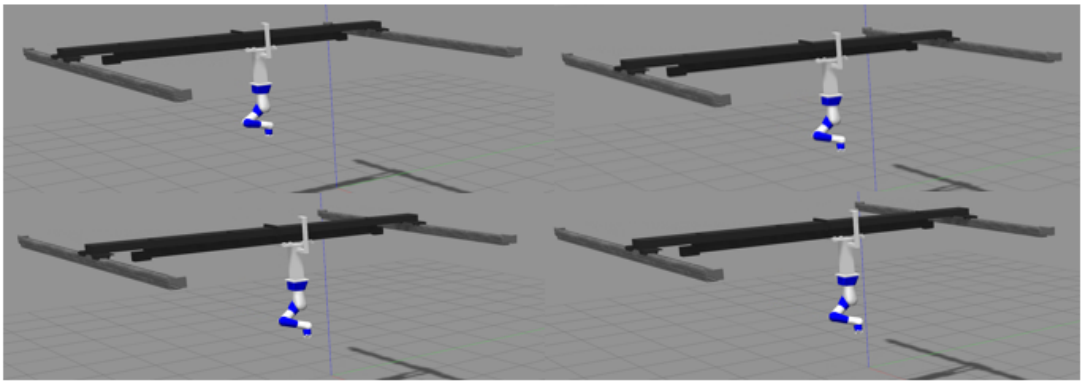


Figure 17: The trajectory for target reaching in the Gazebo simulation.

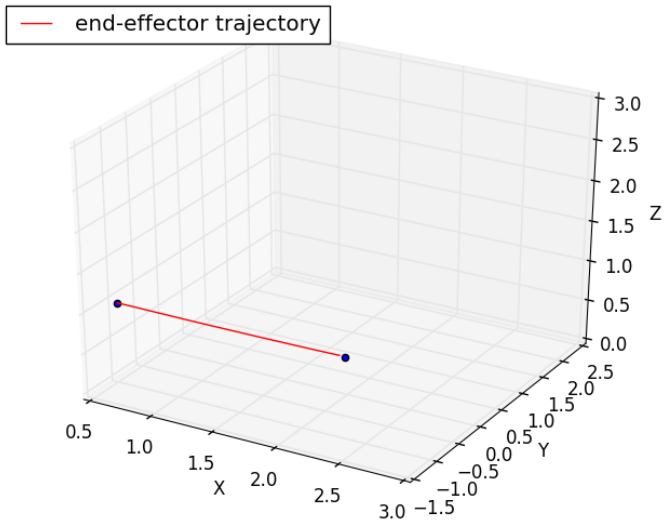


Figure 18: End-effector trajectory for the target reaching problem
start = [0.25,-0.25,3.1415,0.9494,0,2.3,0,-1.5,0] rad; goal = [2.73,-2.11,1.55] m.

The positions of the robot at various timesteps during the simulation, the end-effector

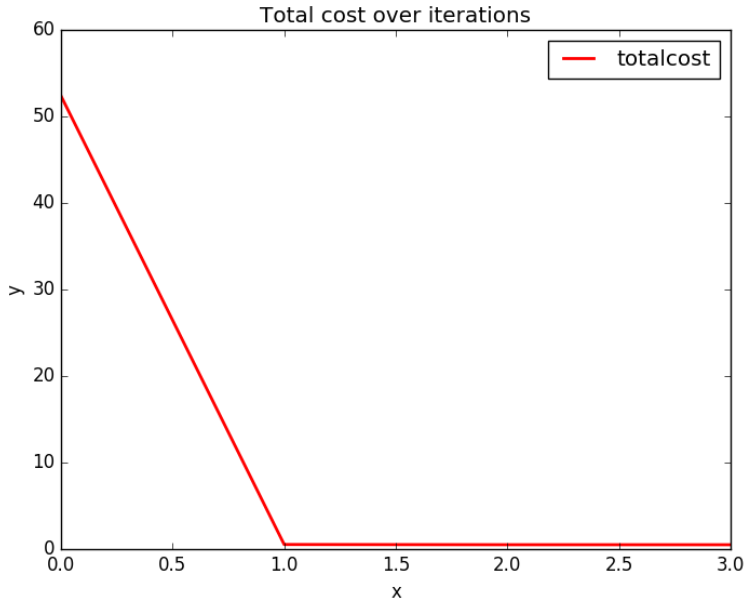


Figure 19: Trajectory cost over iterations for the target reaching problem.

trajectory, and the trajectory cost are shown in the Fig. 17, 18, 19 respectively. From the end-effector trajectory and the trajectory cost, It can be seen that the arm-gantry

system reaches the goal position in the given time.

b) Non - uniform control cost

From an application point of view, for this particular arm - gantry system the effort required to actuate the gantry is more than the effort required to actuate the joints. Hence, this simulation is conducted with a non-uniform control cost such that $H_{x\_gantry} > H_{y\_gantry} > H_{joint\_s} = H_{joint\_l} = H_{joint\_e} = H_{joint\_u} = H_{joint\_r} = H_{joint\_b} = H_{joint\_t}$ where H_i denotes the control cost on the i^{th} joint. The start (joint) and goal (end-effector) positions for this simulation are as follows start = [0.25,-0.25,3.1415,0.9494,0,2.3,0,-1.5,0] rad and the goal = [2.73,-2.11,1.55] m and the control cost H = diag(10, 5, 1, 1, 1, 1, 1, 1, 1).

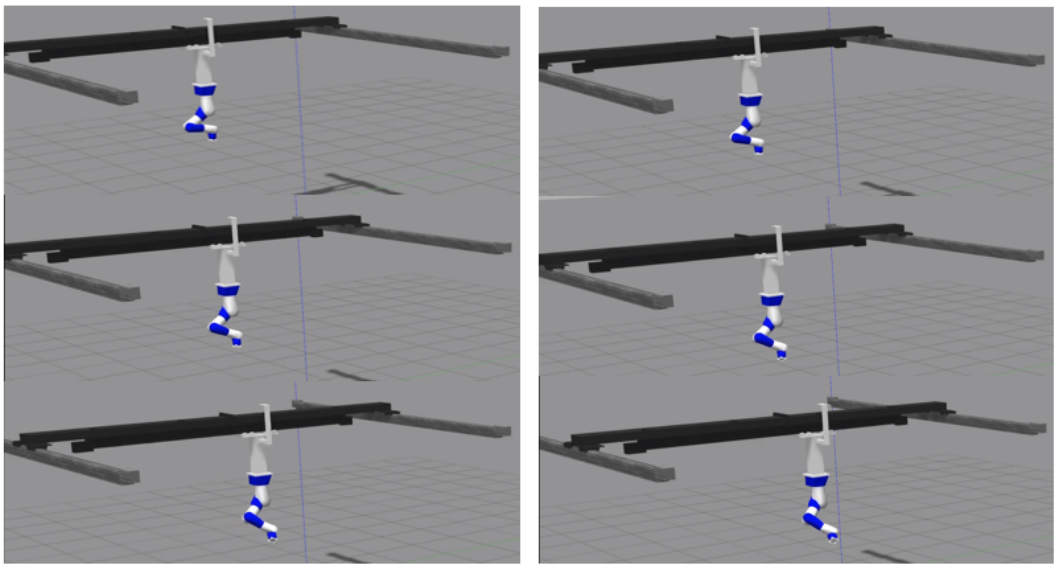


Figure 20: The trajectory for target reaching problem with non-uniform control costs in the Gazebo simulation.

The configuration of the arm-gantry system at different timesteps during the trajectory execution in the Gazebo simulation is as shown in Fig. 20. It is interesting to note that due to the non-uniform control cost, the inferred trajectories limit the motion of the gantry and leverage the range of motion of the robotic arm to reach the goal position. The end-effector and joint position trajectories for the uniform and

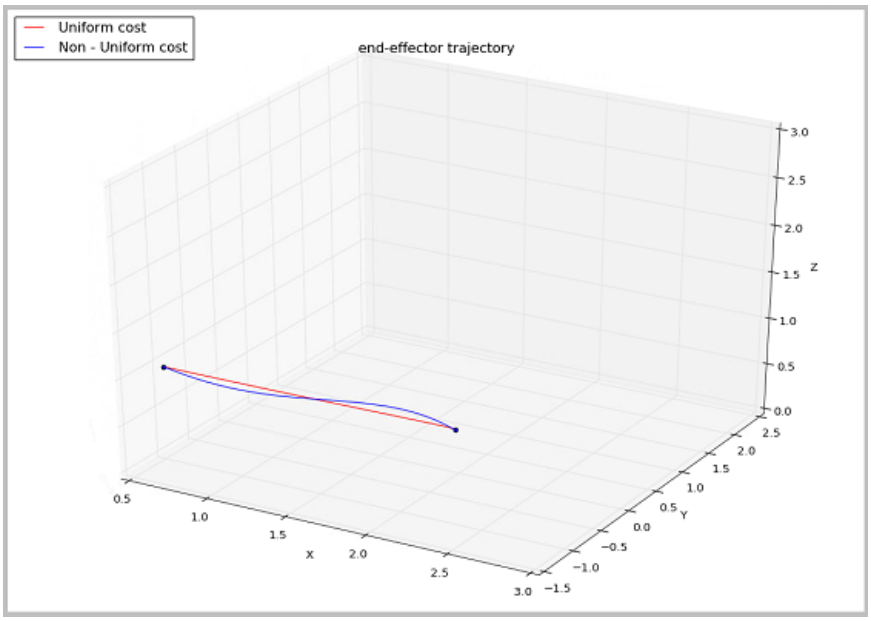


Figure 21: Comparison of the end-effector trajectories for uniform and non-uniform control costs
start = [0.25,-0.25,3.1415,0.9494,0,2.3,0,-1.5,0] rad; goal = [2.73,-2.11,1.55] m.

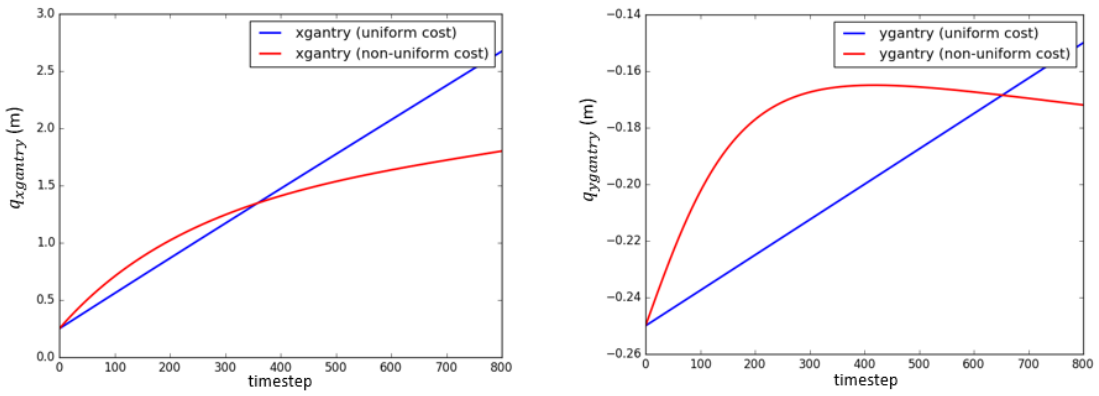
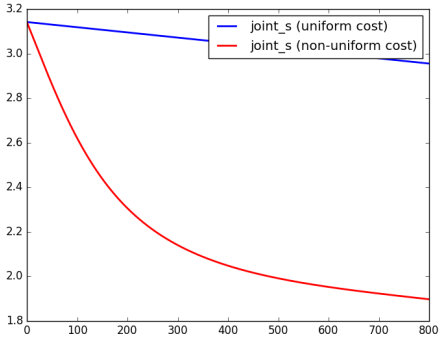


Figure 22: Comparison of the joint position trajectories of the XY-gantry for uniform and non-uniform control costs.

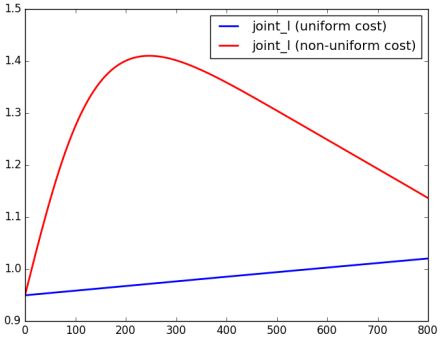
non-uniform control costs are compared in Fig. 21, 22, 23.

c) Comparison of AICO and RRT algorithms

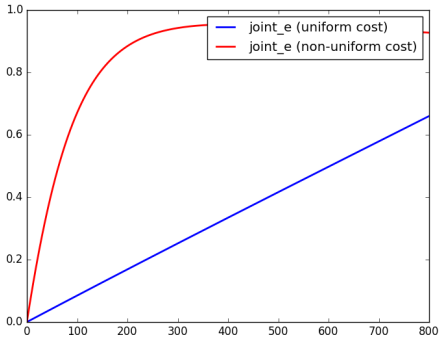
The configuration of the arm-gantry system at various timesteps of the trajectory is shown in Fig. 24 . The Rapidly-exploring Random Trees (RRT) algorithm was used in Moveit to solve the target reaching problem described in Section 5.1.3 (a). A comparison of the end-effector trajectories planned by AICO and RRT (Moveit) is



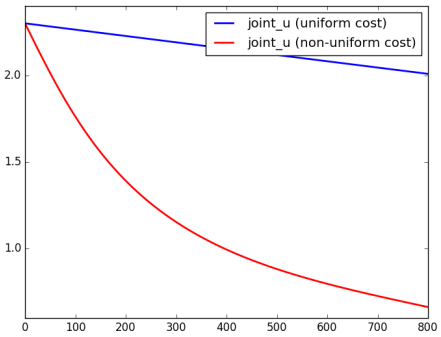
(a) Joint s.



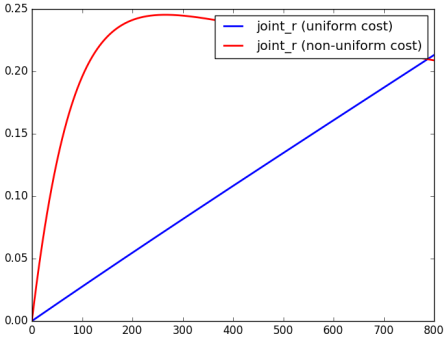
(b) Joint l.



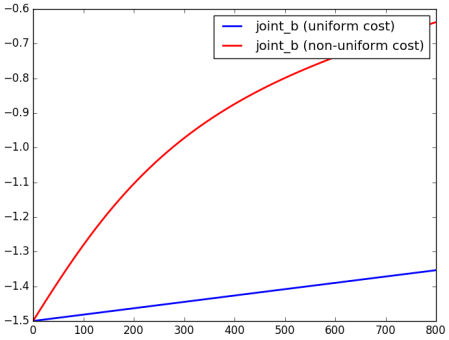
(c) Joint e.



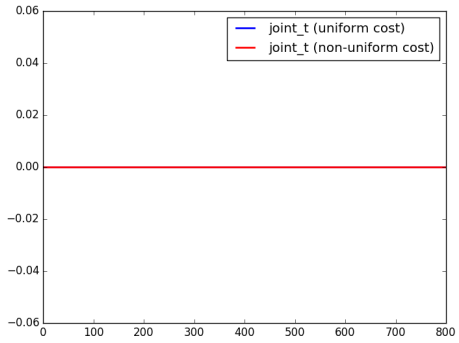
(d) Joint u.



(e) Joint r.



(f) Joint b.



(g) Joint t.

Figure 23: Comparison of the joint position trajectories of the robotic arm for uniform and non-uniform control costs.

shown in Fig. 25. It can be observed that the RRT algorithm only finds a feasible trajectory from the start position to the goal position and hence it is not optimal whereas the AICO infers the optimal trajectory which is unique for a given motion planning problem.

Trajectory tracking problem

The trajectory tracking problem refers to the problem of generating a trajectory that tracks a given reference trajectory as closely as possible. In this case, the cost function is modeled to penalize any deviations from the reference trajectory at every timestep rather than just the final timestep as in the Target reaching problem. The arm-gantry system was tested with sinusoidal and elliptical reference trajectories.

a) Sinusoidal Trajectory Tracking

In this simulation, the end-effector of the arm-gantry system is required to track a given sinusoidal trajectory. The trajectory of the arm-gantry simulation, the end-effector trajectory, and the comparison of the reference and tracked trajectories are shown in Fig. 26, 27, 28 respectively.

b) Elliptical Trajectory Tracking

In this simulation, the end-effector of the arm-gantry system is required to track

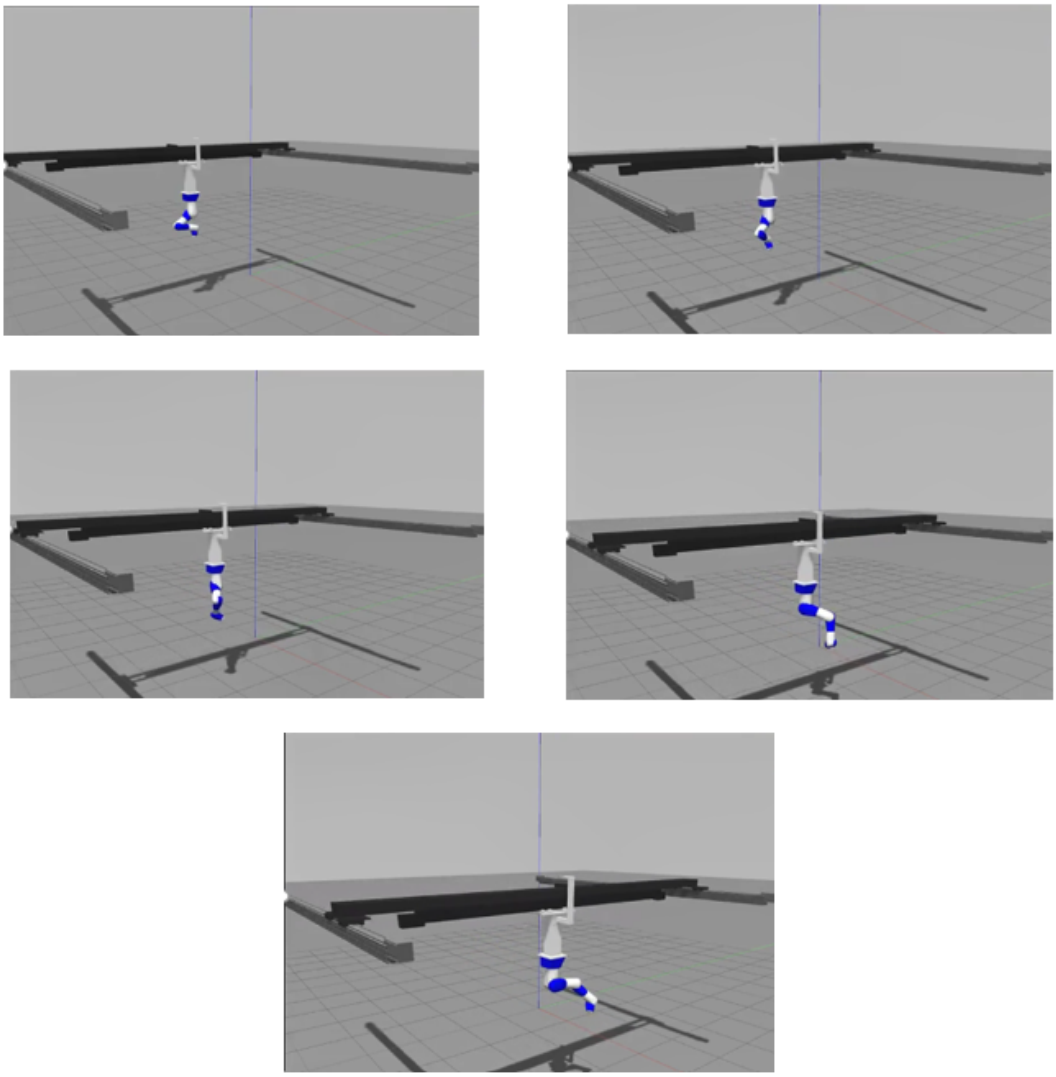


Figure 24: The configuration of arm-gantry system at various timesteps of the trajectory for the target reaching problem using RRT algorithm.

a given elliptical trajectory. The trajectory of the arm-gantry simulation, the end-effector trajectory, and the comparison of the reference and tracked trajectories are shown in Fig. 29, 30, 31 respectively.

It can be observed from the graphs that the inferred trajectories track the given reference trajectories very well and hence demonstrating the applicability of AICO for trajectory tracking problems.

c) Comparison of AICO and RRT algorithms

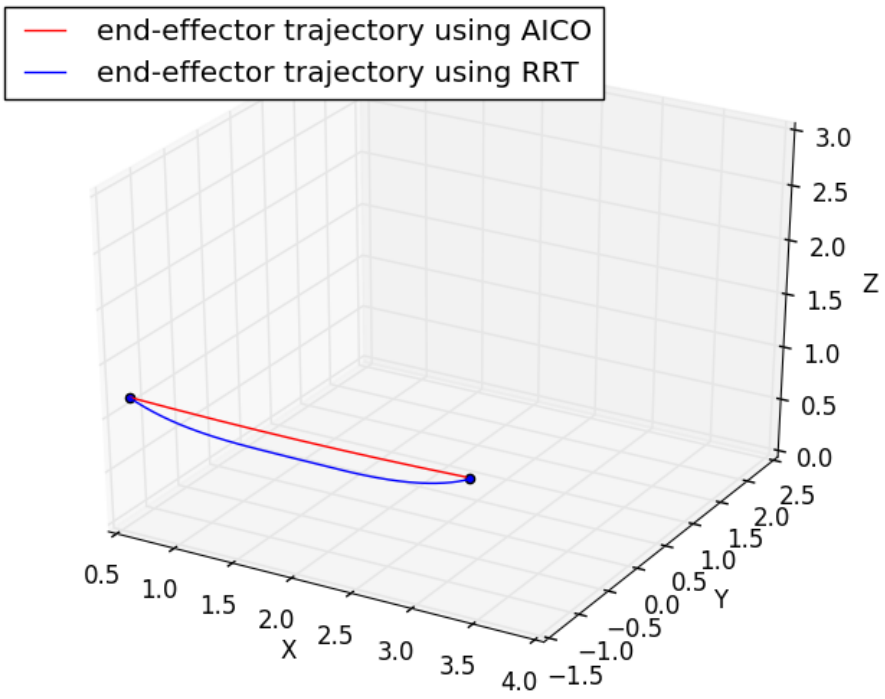


Figure 25: The end-effector trajectories of the arm-gantry system using RRT and AICO algorithms for the target reaching problem.

The tracking performance of AICO and RRT algorithms is compared for the case of elliptical trajectory tracking. Fig. 32 shows the configuration of the arm-gantry system at various timesteps of the trajectory. A comparison of the end-effector trajectories planned by AICO and RRT (Moveit) is shown in Fig. 33. Both the AICO and RRT algorithms generate trajectories based on the given reference trajectory. The key difference is that our current implementation of AICO can accept just the end-effector position (3-DOF) of the waypoints as input whereas Moveit needs the end-effector pose (6-DOF) of the waypoints as input. As the reference trajectory in this case consists only of end-effector positions, Moveit defaults the orientation of all the waypoints to that of the start state. Hence, even though both the algorithms track the reference trajectory well there is a notable difference in the generated joint

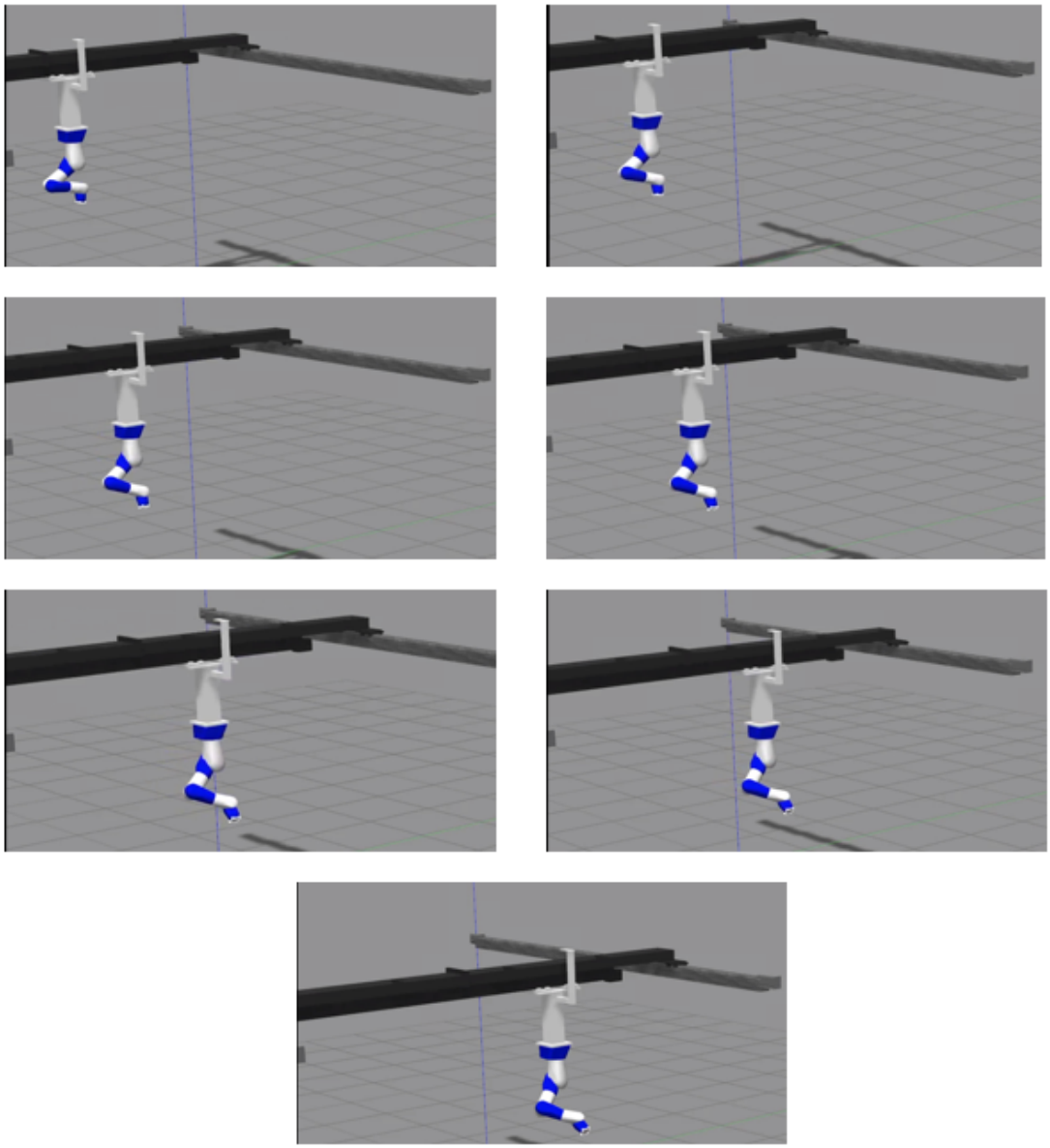


Figure 26: The trajectory for sinusoidal trajectory tracking in the Gazebo simulation.

position trajectories as shown in Figs. 29 and 32. Fig. 34 shows the comparison of the tracking error of both the algorithms. It can be observed that the errors for both AICO and RRT algorithms are on the same order. For this case of elliptical trajectory tracking AICO performs slightly better than RRT.

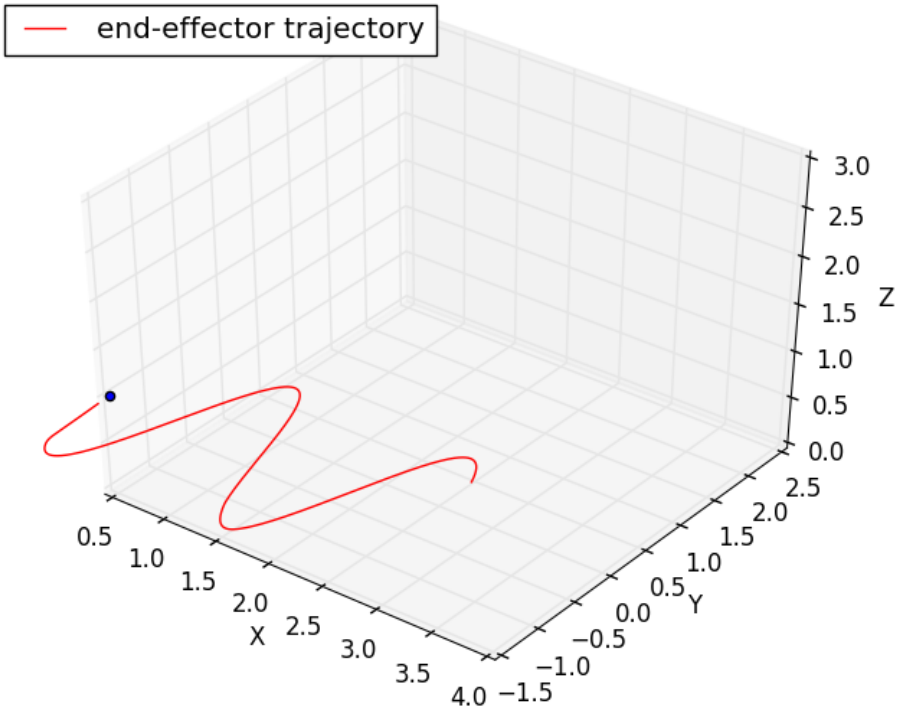


Figure 27: End-effector trajectory for sinusoidal trajectory tracking.

5.2 Unicycle model

5.2.1 Problem formulation

The unicycle model is given as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} + \begin{bmatrix} \mu(x,y) \\ \gamma(x,y) \\ 0 \end{bmatrix} + \begin{bmatrix} d\xi_x \\ d\xi_y \\ 0 \end{bmatrix}. \tag{5.2.1}$$

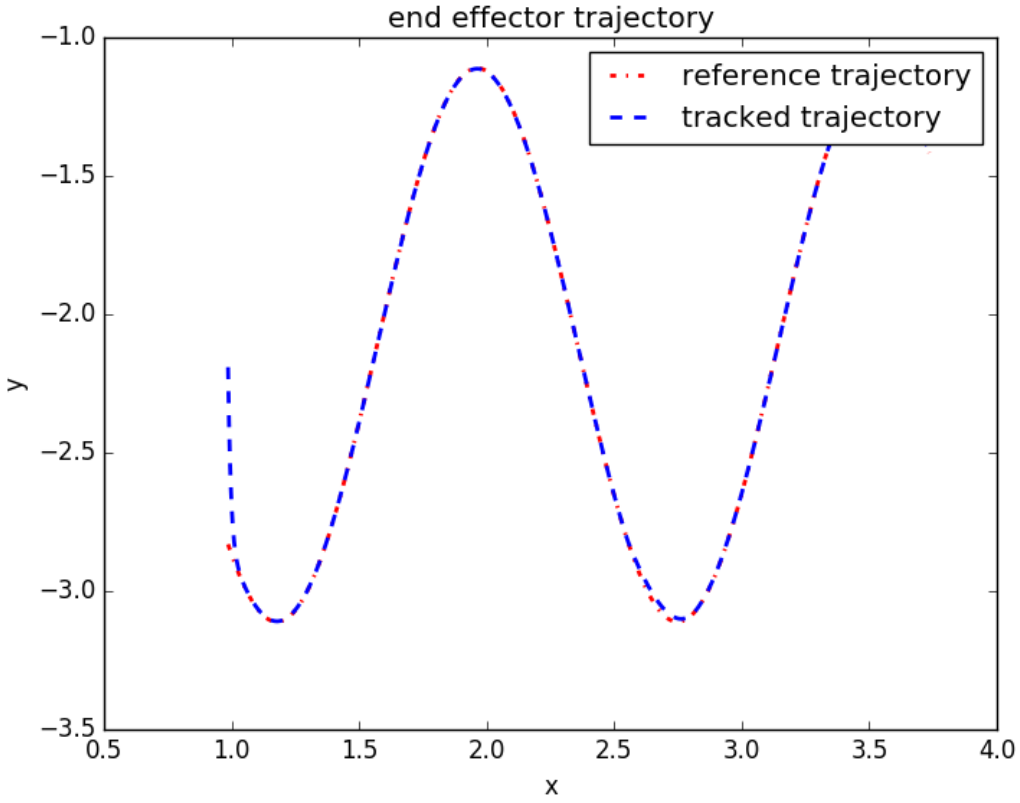


Figure 28: Comparison of reference and tracked sinusoidal trajectories.

Using the Euler's forward method of discretization, eq. (5.2.1) can be expressed as

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} + h \left(\begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + \begin{bmatrix} \mu(x,y) \\ \gamma(x,y) \\ 0 \end{bmatrix} + \begin{bmatrix} d\xi_x \\ d\xi_y \\ 0 \end{bmatrix} \right). \tag{5.2.2}$$

where h is the timestep for discretization, v_t, ω_t are the linear and angular velocities respectively, μ, γ are the external disturbances, $d\xi_x$, $d\xi_y$ represent Brownian motion in states x, y respectively. In the present work, we consider the specific case of unicycle model with an equality constraint on the linear velocity $v_t = v$ and bounded constraint on the angular velocity $\omega_{min} < \omega < \omega_{max}$. But, these constraints cannot be incorporated into the AICO algorithm and thus it infers infeasible trajectories that violate these constraints. Hence, we modify the unicycle model using the state

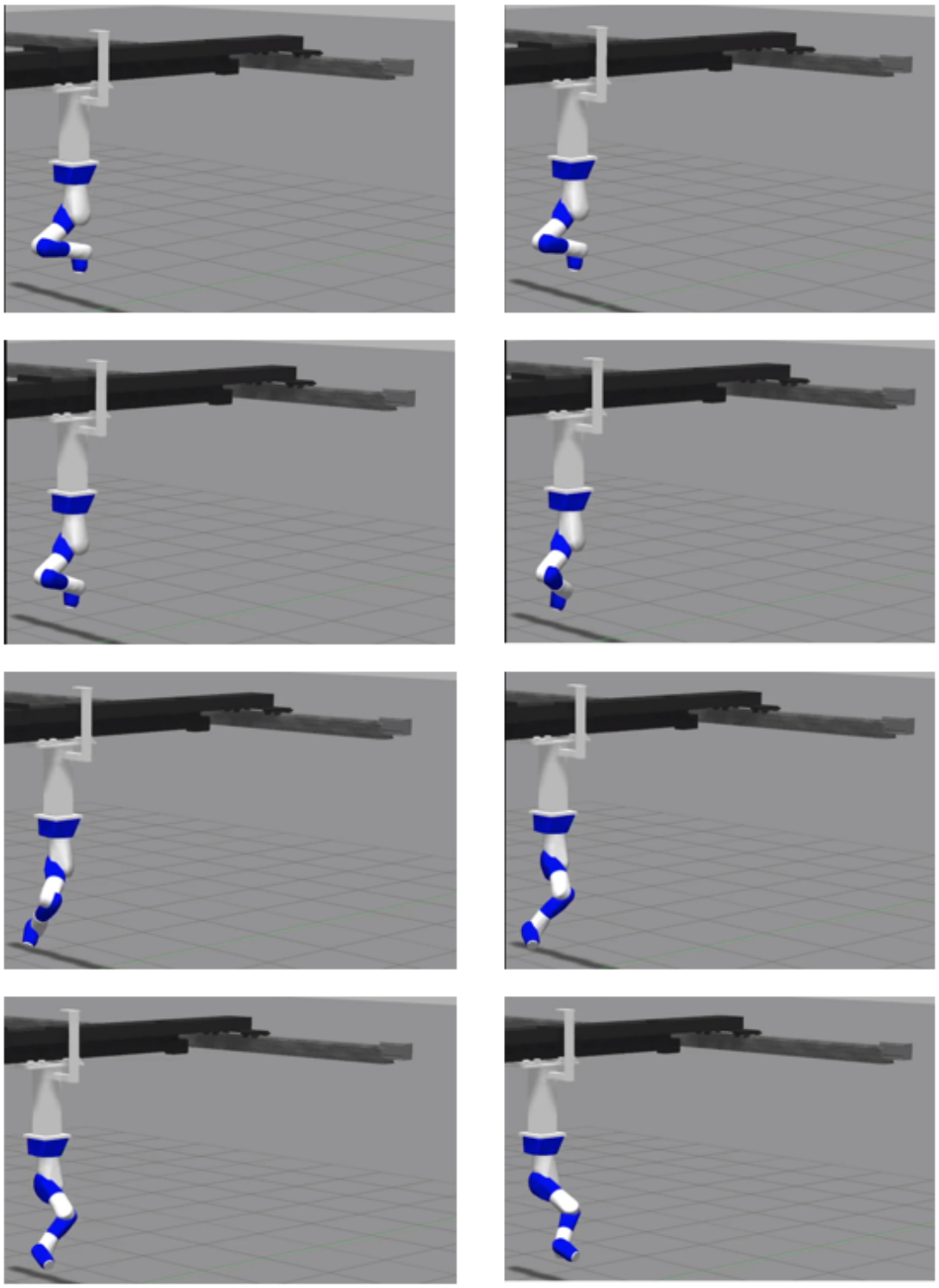


Figure 29: The trajectory for elliptical trajectory tracking in the Gazebo simulation.

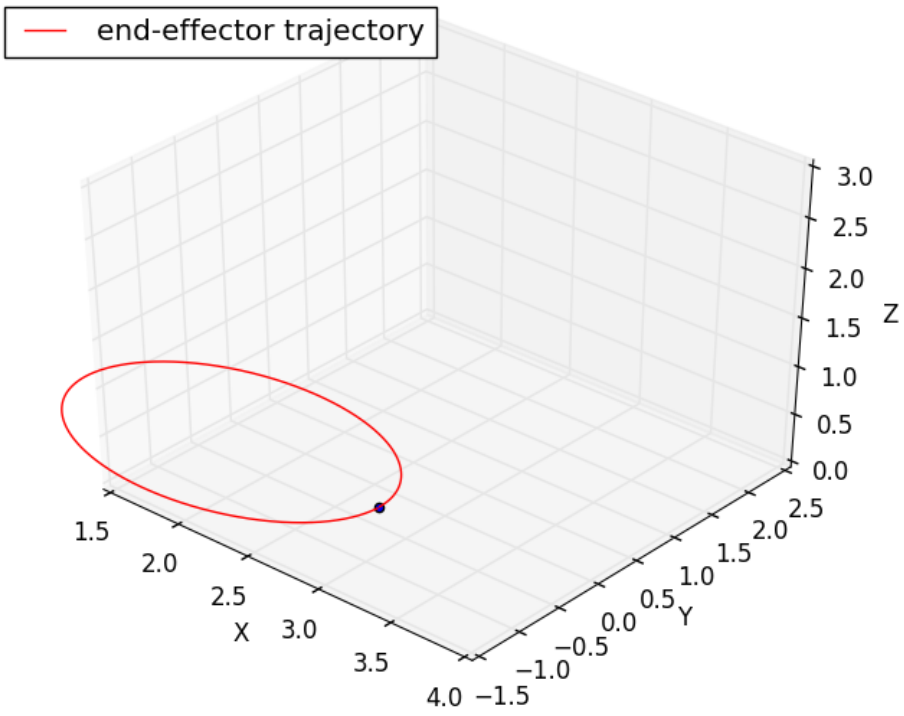


Figure 30: End-effector trajectory for elliptical trajectory tracking.

extension approach discussed in Section 3.3 and extend the state with $\dot{v} = 0$. The resulting model consists of 4 states and 1 input in contrast to the 3 states and 2 inputs in eq. (5.2.1). Linearizing the the dynamics in eq. (5.2.2) assuming that $v_t = v$(equality constraint) we obtain dynamics of the form

$$X_{t+1} = A_t X_t + a_t + B_t u_t + d\xi_t \tag{5.2.3}$$

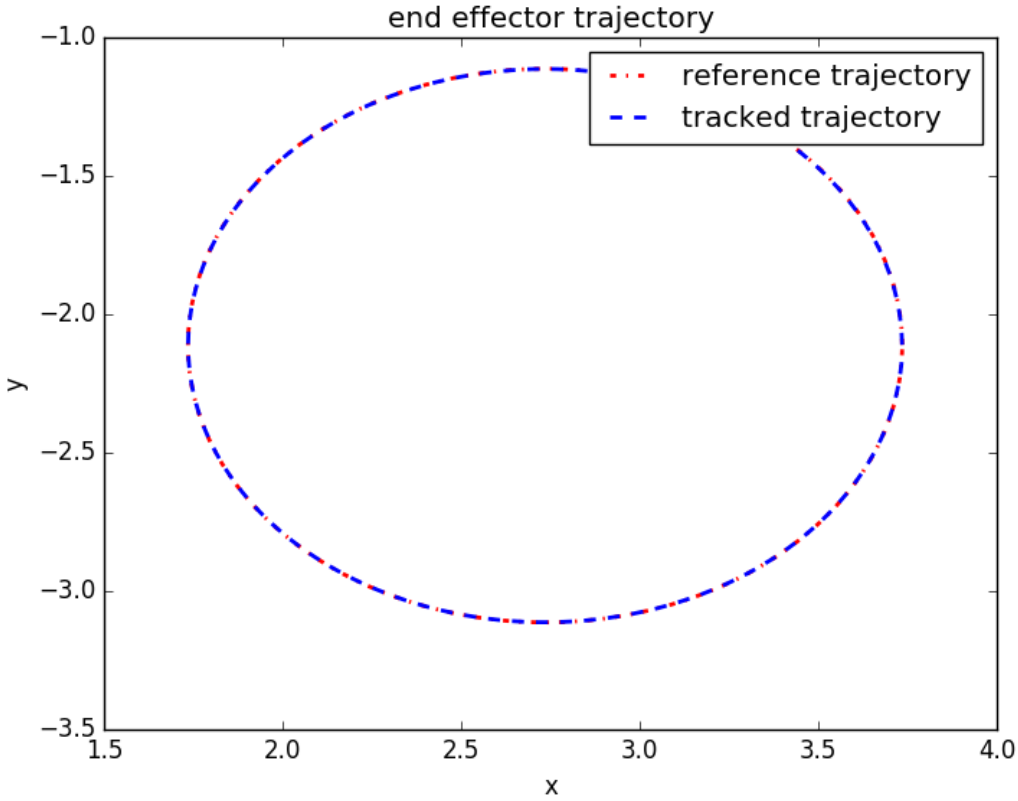


Figure 31: Comparison of reference and tracked elliptical trajectories.

Comparing eqns. (5.2.3) and (3.1.16) we identify, where

$$X_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \\ v_t \end{bmatrix}, \tag{5.2.4}$$

$$u_t = \begin{bmatrix} \omega_t \end{bmatrix}, \tag{5.2.5}$$

$$J_x = h \begin{bmatrix} 0 & 0 & -v\sin(\theta_t) & \cos(\theta_t) \\ 0 & 0 & v\cos(\theta_t) & \sin(\theta_t) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{5.2.6}$$

$$A_t = I + hJ_x, \tag{5.2.7}$$

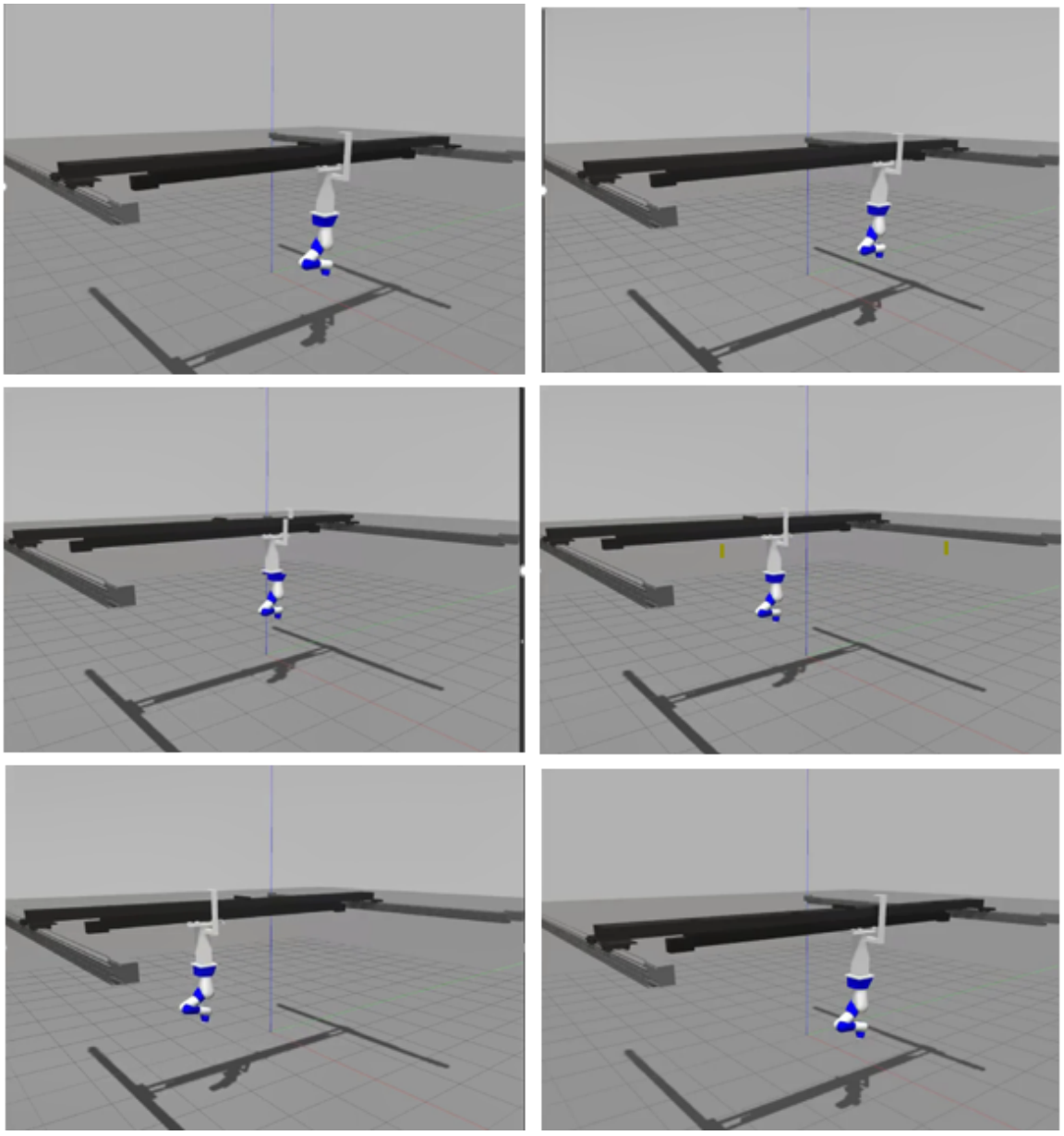


Figure 32: The configuration of arm-gantry system at various timesteps of the trajectory for elliptical trajectory tracking using RRT algorithm.

$$a_t = h \begin{bmatrix} v\cos(\theta_t) + \mu_t(x,y) \\ v\sin(\theta_t) + \gamma_t(x,y) \\ 0 \\ 0 \end{bmatrix} - hJ_x X_t, \tag{5.2.8}$$

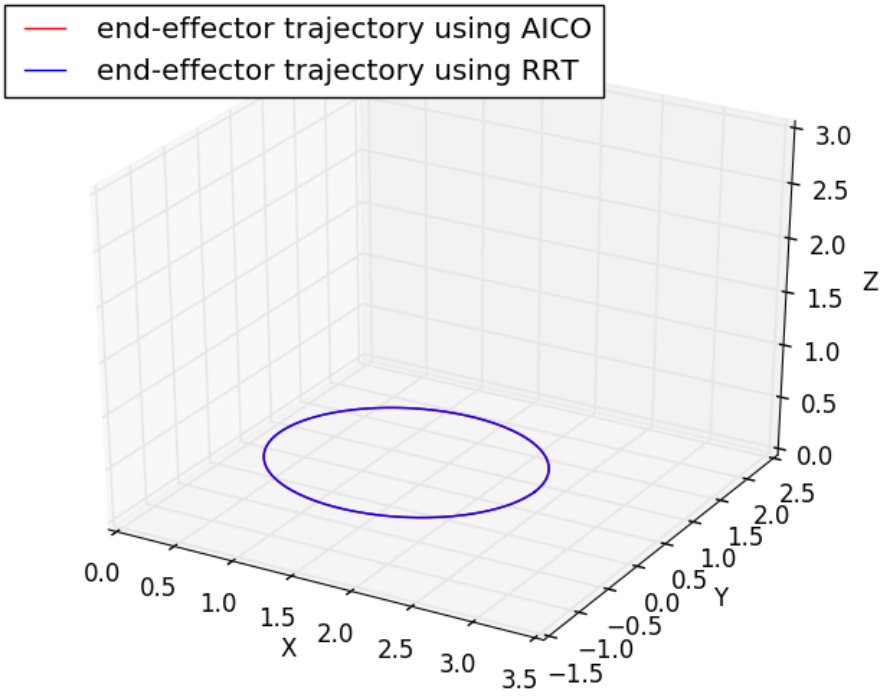


Figure 33: Comparision of the end-effector trajectories of the arm-gantry system for elliptical trajectory tracking using AICO and RRT algorithms.

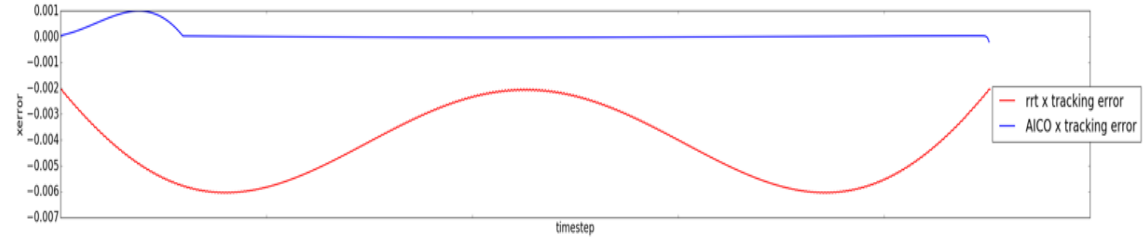
$$B_t = h \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \tag{5.2.9}$$

$$Q_t = \langle d\xi_t d\xi_t^T \rangle, \tag{5.2.10}$$

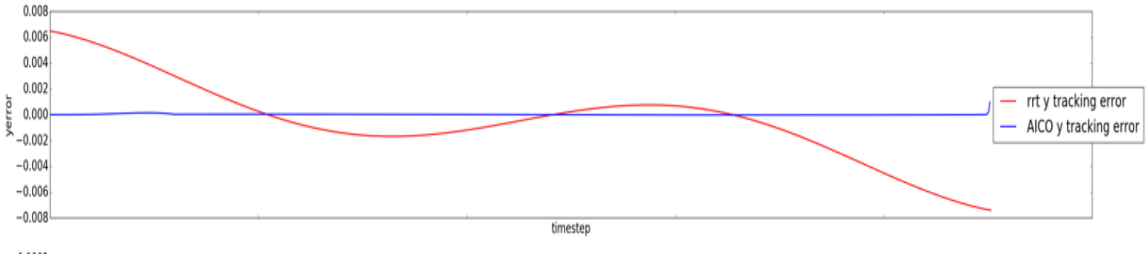
$$H_t = 1. \tag{5.2.11}$$

5.2.2 Implementation details of AICO

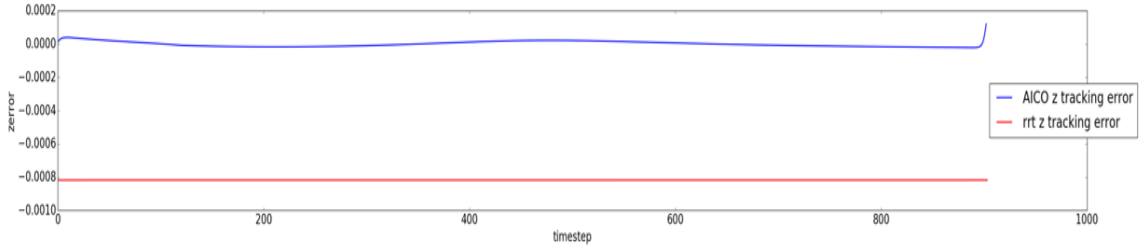
Let $T \in \mathbb{R}$ be the trajectory length for the motion planning problem. Then, using the notations in eqns. (5.2.4) - (5.2.11) the forward, backward and task messages for the AICO algorithm [22] shown in Appendix 0.1 are computed as follows



(a) x tracking error.



(b) y tracking error.



(c) z tracking error.

Figure 34: Comparision of the tracking error of the trajectories generated using AICO and RRT algorithms for elliptical trajectory tracking.

Forward message:

For a timestep $t \in (0, T]$ the forward message is computed as

$$s_t = a_{t-1} + A_{t-1}(S_{t-1}^{-1} + R_{t-1})^{-1}(S_{t-1}^{-1}s_{t-1} + r_{t-1}),$$
$$S_t = Q + B_{t-1}H^{-1}B_{t-1}^T + A_{t-1}(S_{t-1}^{-1} + R_{t-1})^{-1}A_{t-1}^T.$$

Backward message:

For a timestep $t \in (0, T\text{-}1]$ the backward message is computed as

$$\nu_t = -A_t^{-1}a_t + A_t^{-1}(V_{t+1}^{-1} + R_{t+1})^{-1})(V_{t+1}^{-1}\nu_{t+1} + r_{t+1}),$$
$$V_t = A_t^{-1}[Q + B_tH^{-1}B_t^T + (V_{t+1}^{-1} + R_{t+1})^{-1}]A_t^{-T}.$$

Task message:

Let y_1 be the task variable that defines the position of the object (x, y) at a particular timestep and $\phi_1 : X \to y_1$ be the map from X to y_1.

$$y_1 = \phi(X) = \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.2.12}$$

The value of $y_{1,0:T}^*$ is set to that of the goal position in the case target reaching and to the reference trajectory in the case of trajectory tracking.

We define y_2 to be a task variable that measures the collision danger at a particular timestep and $\phi_2 : X \to y_2$ be the map from X to y_2.

$$p_i = \begin{cases} (d_i - \epsilon)^2 & \text{, if } d_i < \epsilon \\ 0 & \text{, otherwise} \end{cases} \tag{5.2.13}$$

$$y_2 = \phi_2(X) = \sum_i p_i \tag{5.2.14}$$

The value of $y^*_{2,0:T}$ is set to zero because it is desired that the robot always maintains a distance greater than the safe distance ϵ from the obstacle.

A task variable y_3 is defined to check the violation of the equality constraint on the linear velocity at a particular timestep and $\phi_3 : X \rightarrow y_3$ be the map from X to y_3.

$$y_3 = \phi_3(X) = v \tag{5.2.15}$$

The value of $y^*_{3,0:T}$ is set to the value of the equality constraint imposed on the linear velocity. Hence, any deviation of the actual velocity y_3 from this value incurs a cost on the system.

Another task variable y_4 is defined to check the violation of the bounded constraint on the angular velocity at a particular timestep and $\phi_4 : X \rightarrow y_4$ be the map from X to y_4.

$$y_4 = \phi_4(X) = \begin{cases} h\omega_{min} - \theta_{t+1} - \theta t & \text{, if } \theta_{t+1} - \theta t < h\omega_{min} \\ \theta_{t+1} - \theta t - h\omega_{max} & \text{, if } \theta_{t+1} - \theta t > h\omega_{max} \\ 0 & \text{, otherwise} \end{cases} \tag{5.2.16}$$

The value of $y^*_{4,0:T}$ is set to zero because it is desired that the value of the angular velocity is always within bounds. Let $\rho_{i,t}$ be the precision that represents the penalty of the deviation of $y_{i,0:T}$ from $y^*_{i,0:T}$ $\forall i = [1, 4]$ at timestep t. In the case of target reaching a high precision is required only at the final timestep. Hence, we chose $\rho_{1,0:T-1} = 1e-2$, $\rho_{1,0:T} = 1e2$. For trajectory tracking, the a high precision is required throughout the trajectory and hence the precision was chosen to be $\rho_{1,0:T} = 1e2$. A more stronger cost has to be incurred for deviations in all the other task variables that represent collision avoidance, linear and angular velocity constraints requiring much larger precision on these variables. Hence, we chose $\rho_{2,3,4,0:T} = 1e5$.

From eq. (3.1.43), the mean r and precision R of the task message can be written as

$$r_t = \sum_{i=1}^{4} \rho_{i,t} \hat{J}_i^T (y_{i,t}^* - \phi_i(\hat{X}_t) + \hat{J}_i \hat{X}_t), \tag{5.2.17}$$

$$R_t = \sum_{i=1}^{4} \rho_{i,t} \hat{J}_i^T \hat{J}_i \tag{5.2.18}$$

where J_i is the Jacobian of $\phi_i(\cdot)$ defined as

$$J_i = \frac{\partial \phi_i}{\partial X_t} = \begin{bmatrix} \frac{\partial \phi_i}{\partial x_t} & \frac{\partial \phi_i}{\partial y_t} & \frac{\partial \phi_i}{\partial \theta_t} & \frac{\partial \phi_i}{\partial v_t} \end{bmatrix} \tag{5.2.19}$$

Belief:

The belief at the timestep $t \in (0, T)$ is computed as

$$b_t = S_t^{-1} s_t + V_t^{-1} v_t + r_t,$$

$$B_t = S_t^{-1} + V_t^{-1} + R_t.$$

The value of b_t corresponds to the optimal value of the joint positions at time t denoted as q_t. In the present work, the AICO [22] algorithm was studied in the context of the constrained unicycle model given by eqns. (5.2.4) - (5.2.11) for various planning problems such as target reaching, obstacle avoidance, and the effect of external disturbances.

5.2.3 Results

Target reaching problem

In this problem, the algorithm has to infer an optimal trajectory from a given start position to a given goal position. The cost function in this case penalizes the error between the (x, y) position at the final timestep and the goal position. The parameters

considered for this simulation are as follows

start = [-100,20,0,100]

goal = [-10,30] m

T = 91

h = 0.01s

$\omega \in [-3.14, 3.14]$ rad/s

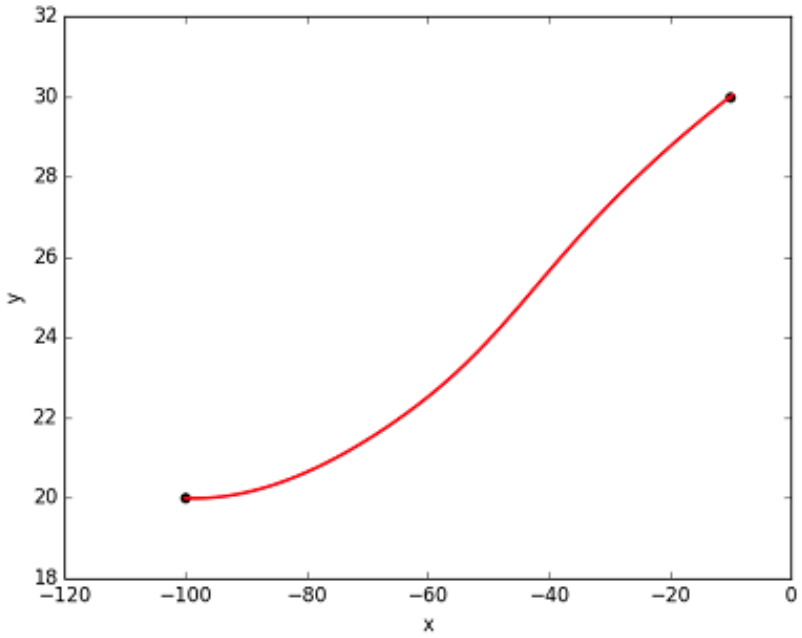


Figure 35: xy-trajectory of unicycle model for the target reaching problem start = [-100,20,0,100]; goal = [-10,30] m.

It can be observed from Fig. 35 that the robot reaches the goal state. But, it should be noted that the T is exact in this case which may not be possible always. So, we perform another simulation for the target reaching problem to study the AICO algorithm when the T is not exact. The parameters considered for this simulation are as follows

start = [-100,20,0,100]

goal = [0,0] m

T = 110

h = 0.01s

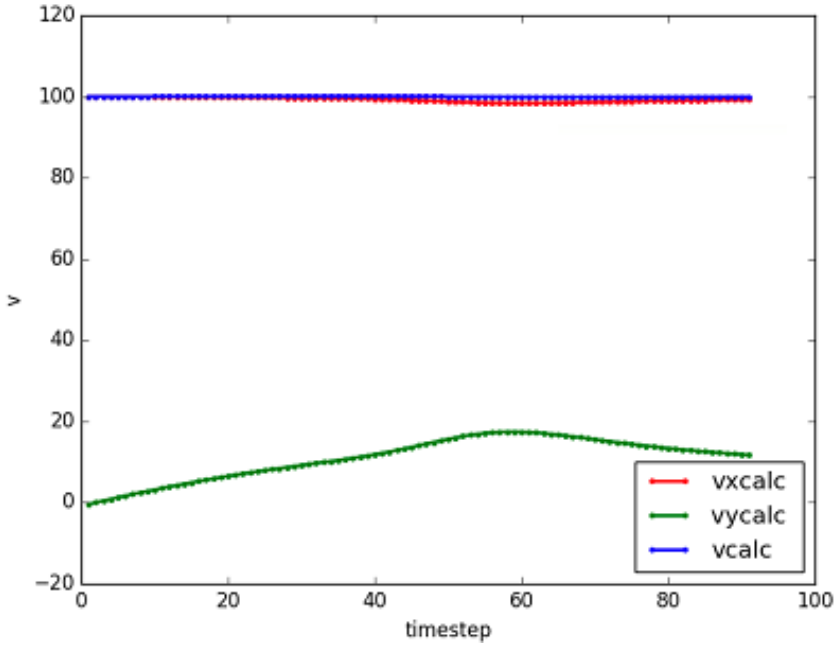


Figure 36: The linear velocity ($v = 100$ m/s) of the unicycle model for the target reaching problem.

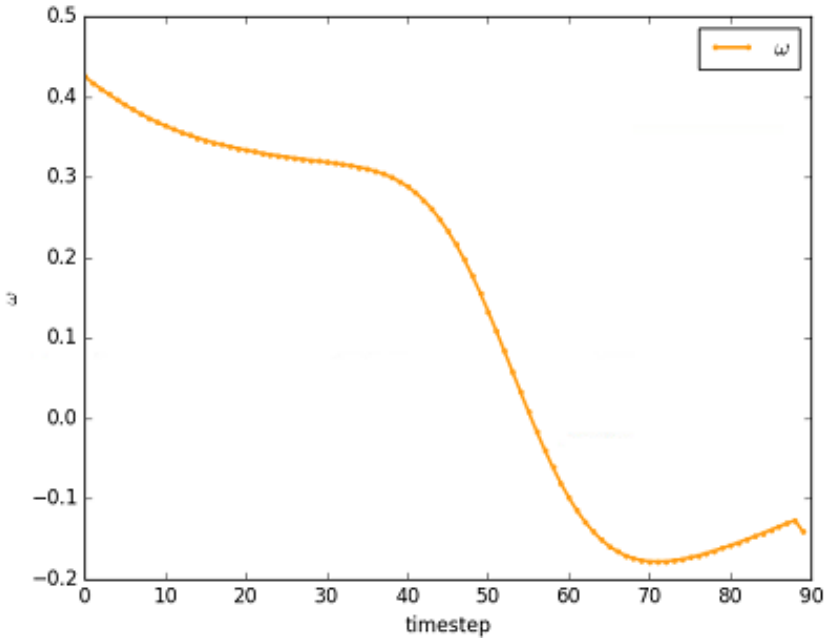


Figure 37: Angular velocity ($\omega \in [-3.14, 3.14]$ rad/s)of unicycle model for the target reaching problem.

$\omega \in [-3.14, 3.14]$ rad/s

It is interesting to note that the trajectory in Fig. 38 reaches the goal but deviates from the straight-line trajectory which is usually optimal. This is attributed to the

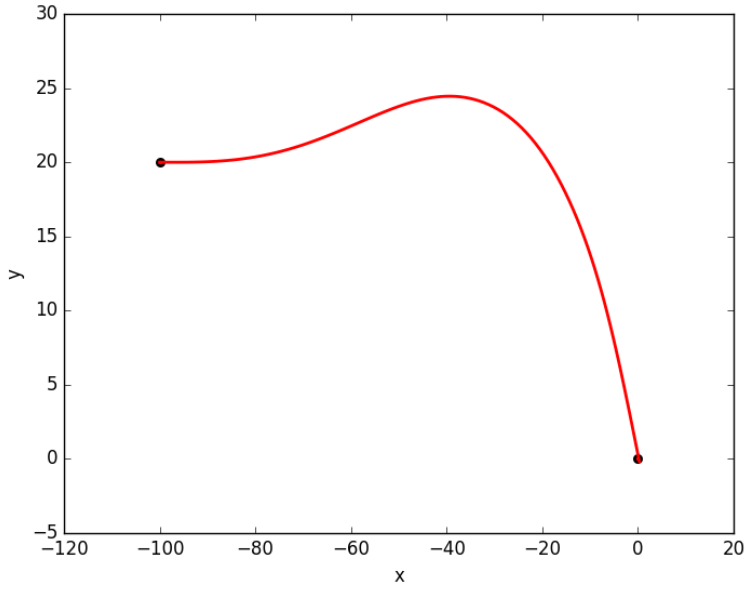


Figure 38: xy-trajectory of unicycle model for the target reaching problem start = [-100,20,0,100]; goal = [0,0] m.

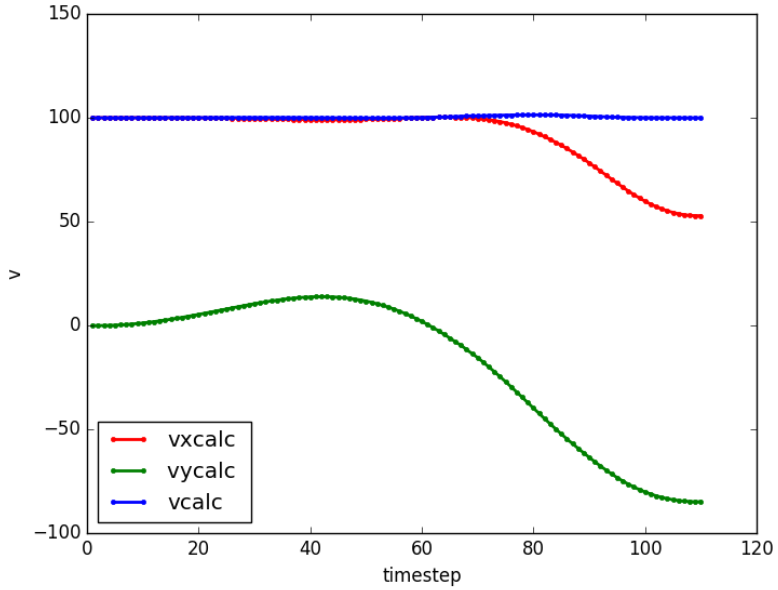


Figure 39: The linear velocity (v = 100 m/s) of the unicycle model for the target reaching problem.

fact that the cost function penalizes the error in the inferred position at the final timestep T and the given goal position. Hence, given the high linear velocity and the larger trajectory length than required, the inferred trajectory deviates from the

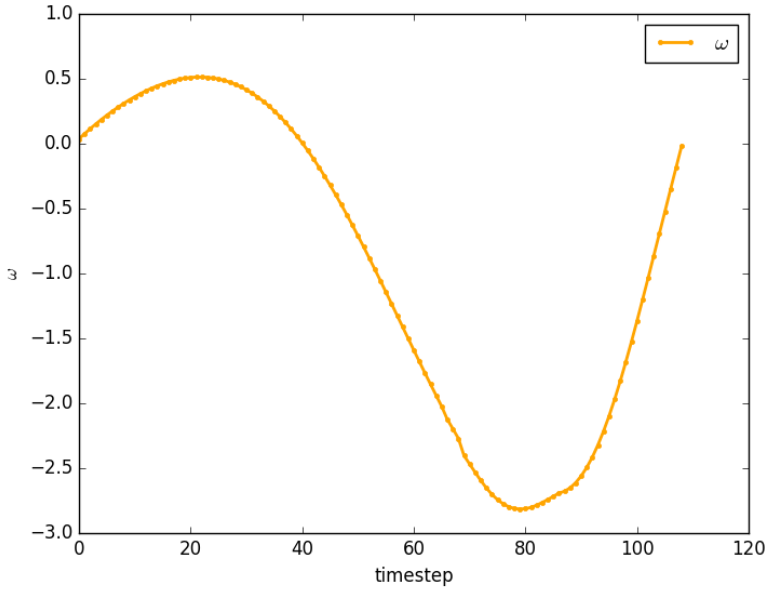


Figure 40: Angular velocity ($\omega \in [-3.14, 3.14]$ rad/s) of unicycle model for the target reaching problem.

straight-line path such that it reaches the goal position at the final timestep T.

Obstacle avoidance

In this section, we present the simulation results for the target reaching problem in an obstacle-filled environment. Here, the algorithm has to infer trajectories from the start state to the goal state while avoiding obstacles. The cost function in this case not only penalizes the goal error but also the collision with obstacles. Let, d_i be the minimum distance of the robot from i^{th} obstacle and ϵ be the safe distance for collision. The task variable is modeled as shown below.

For each obstacle i in the environment,

$$p_i = \begin{cases} (d_i - \epsilon)^2 & \text{, if } d_i < \epsilon \\ 0 & \text{, otherwise} \end{cases} \tag{5.2.20}$$

$$y = \sum_i p_i \tag{5.2.21}$$

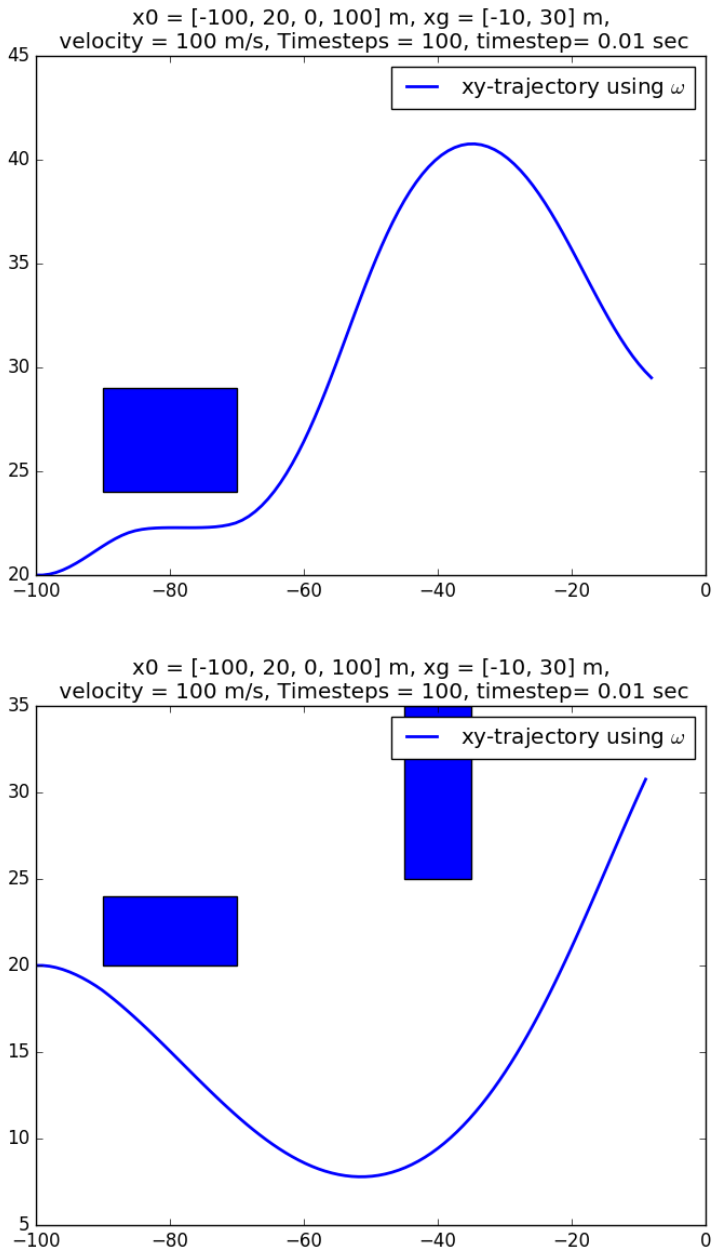


Fig. 41, 42 show the inferred trajectories in various obstacle-filled environments. It can be seen that all the trajectories reach the goal state by avoiding all the obstacles. During the simulations, it was observed that in scenarios where the problem is over-constrained i.e., if an obstacle-free trajectory does not exist with the given constraints then the AICO algorithm does not converge to a solution.

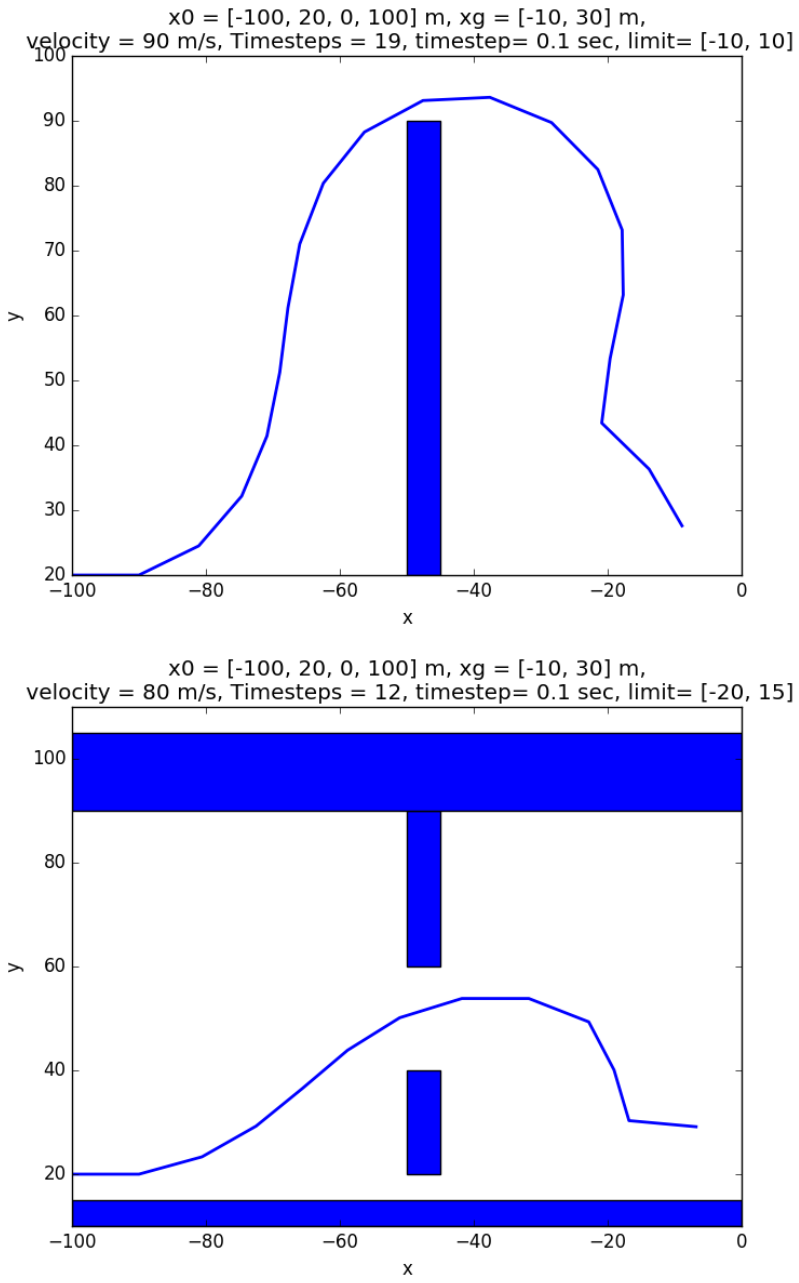


Figure 41: xy-trajectories in various obstacle-filled environments.

Effect of external disturbance

In this section, we discuss the simulation results for the target reaching problem in the presence of external disturbances. The objective of this simulation is to study the behavior of AICO for a target reaching problem in the presence of external disturbance i.e. the values of μ_t, γ_t in eq. 5.2.2 are non-zero.

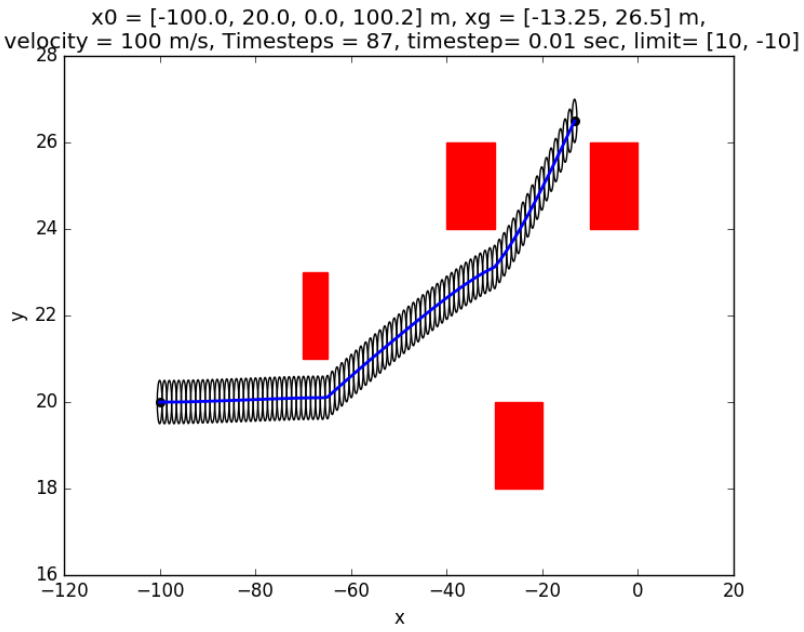


Figure 42: xy-trajectory in an obstacle-filled environments.

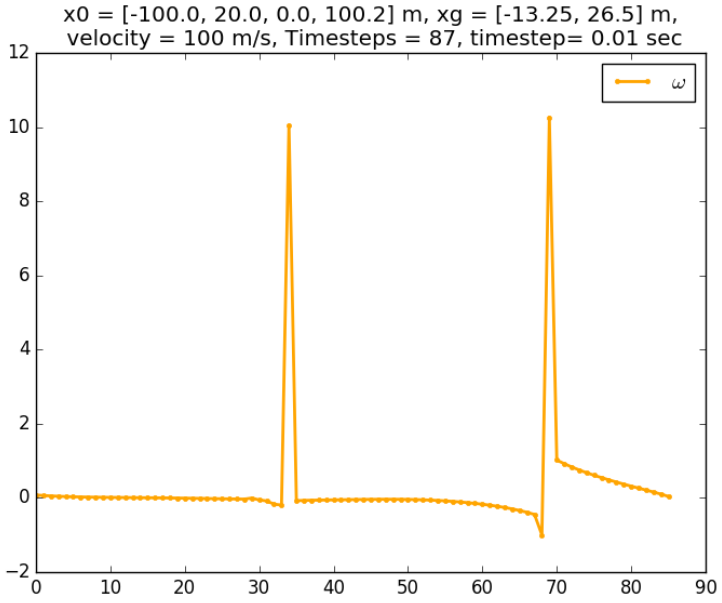


Figure 43: Angular velocity for the trajectory in Fig. 42

It can be observed from Fig. 44, 45 that the external disturbances induce a position shift in the overall trajectory in the direction of the disturbance. But, for the timesteps closer to the final time T the inferred trajectory reorients towards the goal. This can be attributed to the fact that the cost is imposed only on the final timestep. Hence,

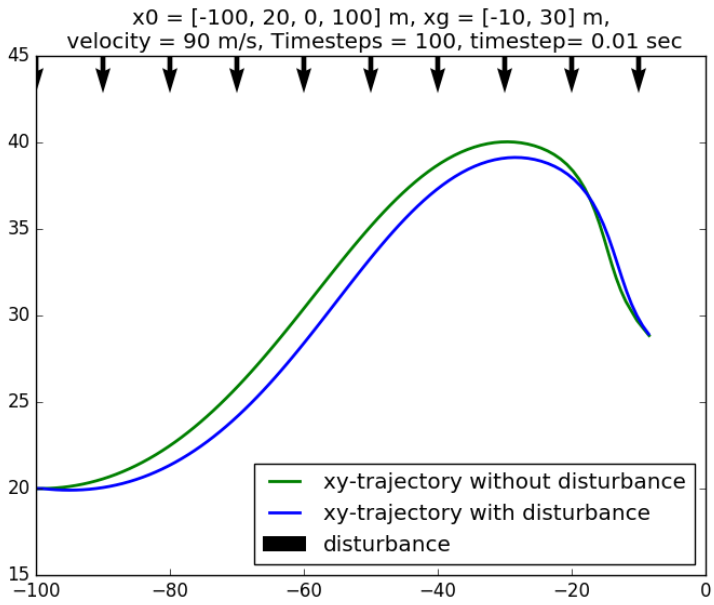


Figure 44: xy-trajectories with and without disturbance for $\mu = 0$ and $\gamma = -5$ m/s.

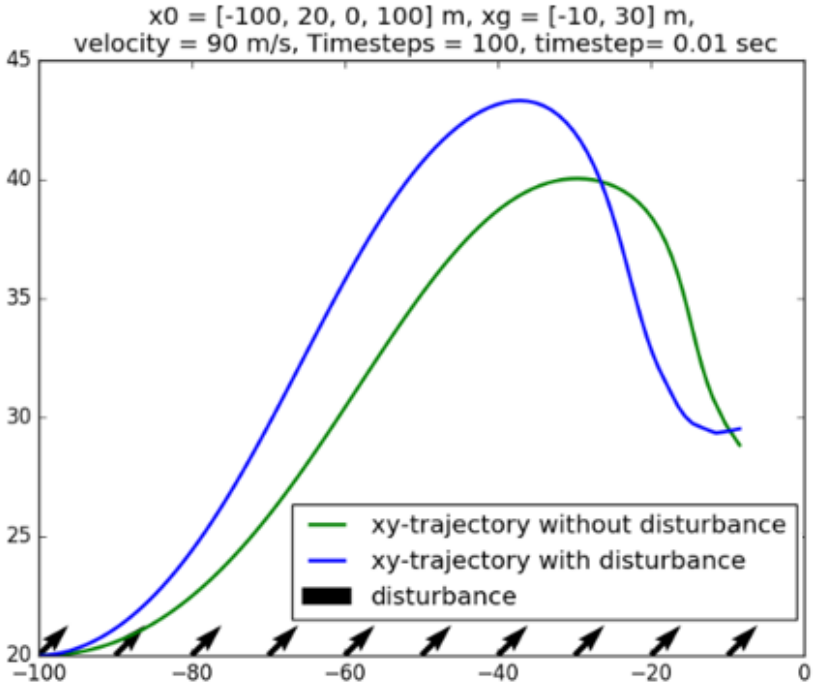


Figure 45: xy-trajectories with and without disturbance for $\mu = 2.5$ m/s and $\gamma = 2.5$ m/s.

appropriate corrections are made to the control to account for the disturbances such that the robot reaches the goal at the final timestep.

Trajectory tracking in the presence of external disturbances

In this simulation, the AICO algorithm is studied for the trajectory tracking problem in the presence of external disturbances. We consider the obstacle-filled environment shown in Fig. 42 and the trajectory therein is passed as the reference trajectory to the algorithm. The parameters for this simulation are as follows

start = [-100,20,0,100.2]

goal = [-13.25, 26.5] m

T = 87

h = 0.01s

$\omega \in [-10, 10]$ rad/s

$\mu = 2.0$ m/s and $\gamma = 0$ m/s

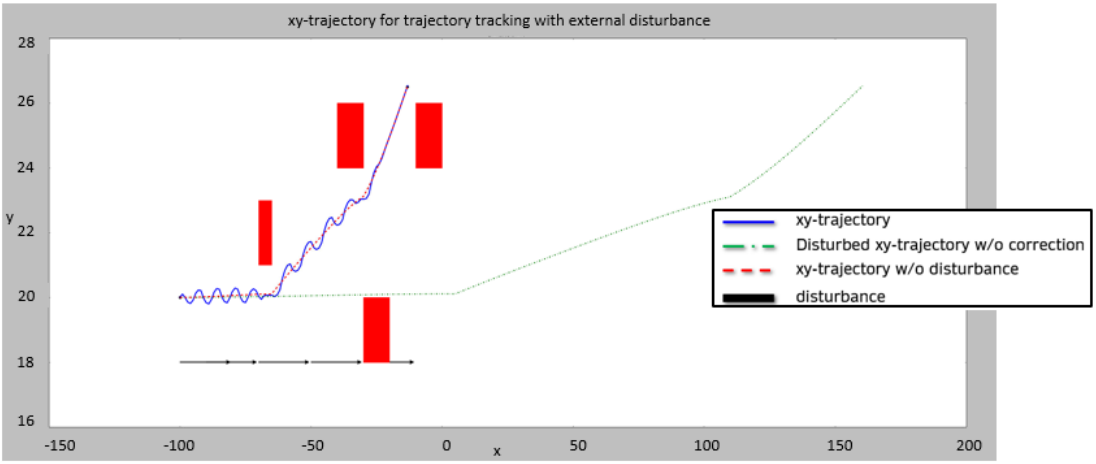


Figure 46: xy-trajectories with ($\mu = 2.0$ m/s and $\gamma = 0$ m/s) and without disturbance in an obstacle filled environment.

In Fig. 46, the red line represents the disturbance-free reference trajectory obtained in Fig. 42. The green line illustrates the disturbed trajectory that does not account for the disturbances. The blue line represents the tracked trajectory that is inferred by the AICO algorithm by accounting for the external disturbance. Given the high linear velocity and the external disturbance in the positive x-direction, the robot traverses in a sinusoidal path to stay close to the reference trajectory. The

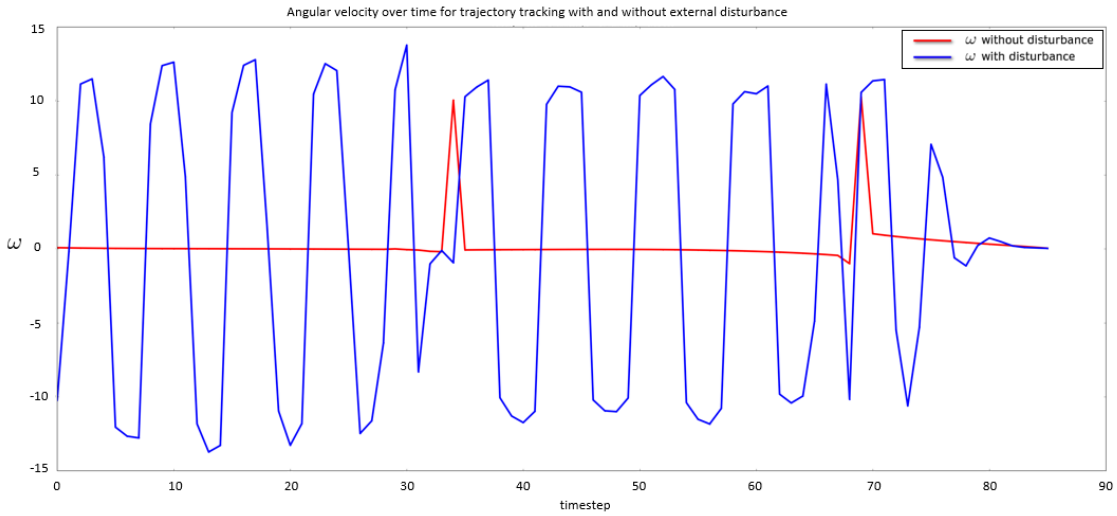


Figure 47: Angular velocity with ($\mu = 2.0$ m/s and $\gamma = 0$ m/s) and without disturbance in an obstacle filled environment.

correction to the angular velocity input in comparison to the angular velocity of the reference trajectory is as shown in Fig. 47.

CHAPTER VI

CONCLUSION AND FUTURE WORK

In this work, a robotic arm-gantry simulator is developed for the towing mechanism of a wavemaker facility. The main purpose of the simulator is to generate and validate the motion plans before interfacing with the actual hardware. The simulator is developed in the ROS-Gazebo framework coupled with Moveit API for motion planning. The simulator was demonstrated to successfully plan and visualize motion plans in target reaching and trajectory tracking settings. The functionality of the simulator was also extended to perform stochastic optimal control using the Approximate Inference Control (AICO) algorithm. The AICO algorithm was implemented in python and tested on the arm-gantry simulator for target reaching and trajectory tracking problems.

The study of AICO was extended to non-holonomic mobile robots by considering the unicycle model with control constraints. A state extension approach was suggested to infer state trajectories that obey the control constraints. Several simulations were performed on the unicycle model such as target reaching with exact and inexact trajectory length, obstacle avoidance, and effect of external disturbance on target reaching and trajectory tracking problems to validate the state extension approach and better understand the behavior of the AICO algorithm.

As part of our future work, we intend to add the functionality to interface the motionplans with the hardware using motoman drivers[1]. We also plan on integrating

[1] http://wiki.ros.org/motoman_driver

sensors into our simulator to be able to perform perception based planning.

REFERENCES

[1] H. Abdulsamad, *Stochastic optimal control with linearized dynamics*, Ph.D. thesis, 2016.

[2] Christopher M. Bishop, *Pattern recognition and machine learning (information science and statistics)*, Springer-Verlag, Berlin, Heidelberg, 2006.

[3] Sachin Chitta, Eitan Marder-Eppstein, Wim Meeussen, Vijay Pradeep, Adolfo Rodríguez Tsouroukdissian, Jonathan Bohren, David Coleman, Bence Magyar, Gennaro Raiola, Mathias Lüdtke, and Enrique Fernández Perdomo, *ros_control: A generic and simple control framework for ros*, The Journal of Open Source Software (2017).

[4] David Coleman, Ioan Alexandru Sucan, Sachin Chitta, and Nikolaus Correll, *Reducing the barrier to entry of complex robotic software: a moveit! case study*, CoRR **abs/1404.3785** (2014).

[5] Pierre Del Moral, *Nonlinear filtering: Interacting particle resolution*, Comptes Rendus de l'Académie des Sciences-Series I-Mathematics **325** (1997), no. 6, 653–658.

[6] Simon J Julier and Jeffrey K Uhlmann, *New extension of the kalman filter to nonlinear systems*, Signal processing, sensor fusion, and target recognition VI, vol. 3068, International Society for Optics and Photonics, 1997, pp. 182–193.

[7] Rudolph Emil Kalman, *A new approach to linear filtering and prediction problems*, (1960).

[8] Daphne Koller and Nir Friedman, *Probabilistic graphical models: principles and techniques*, MIT press, 2009.

[9] Dong-Hwan Lee and Jianghai Hu, *A study of the duality between kalman filters and lqr problems*, (2016).

[10] Sergey Levine, *Reinforcement learning and control as probabilistic inference: Tutorial and review*, arXiv preprint arXiv:1805.00909 (2018).

[11] Jun S Liu and Rong Chen, *Sequential monte carlo methods for dynamic systems*, Journal of the American statistical association **93** (1998), no. 443, 1032–1044.

[12] Kevin Murphy and Stuart Russell, *Rao-blackwellised particle filtering for dynamic bayesian networks*, Sequential Monte Carlo methods in practice, Springer, 2001, pp. 499–515.

[13] Brendan O'Donoghue, *Variational bayesian reinforcement learning with regret bounds*, Advances in Neural Information Processing Systems **34** (2021).

[14] Brendan O'Donoghue, Ian Osband, and Catalin Ionescu, *Making sense of reinforcement learning and probabilistic inference*, arXiv preprint arXiv:2001.00805 (2020).

[15] Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar, *An approximate inference approach to temporal optimization in optimal control*, Advances in neural information processing systems **23** (2010).

[16] ———, *On stochastic optimal control and reinforcement learning by approximate inference*, Proceedings of Robotics: Science and Systems VIII (2012).

[17] Elmar A Rückert and Gerhard Neumann, *Stochastic optimal control methods for investigating the power of morphological computation*, Artificial Life **19** (2013), no. 1, 115–131.

[18] Elmar Rueckert, Max Mindt, Jan Peters, and Gerhard Neumann, *Robust policy updates for stochastic optimal control*, 2014 IEEE-RAS International Conference on Humanoid Robots, 2014, pp. 388–393.

[19] Simo Särkkä, *Bayesian filtering and smoothing*, no. 3, Cambridge university press, 2013.

[20] Emanuel Todorov, *General duality between optimal control and estimation*, 2008 47th IEEE Conference on Decision and Control, IEEE, 2008, pp. 4286–4292.

[21] Emanuel Todorov and Weiwei Li, *A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems*, Proceedings of the 2005, American Control Conference, 2005., IEEE, 2005, pp. 300–306.

[22] Marc Toussaint, *Robot trajectory optimization using approximate inference*, Proceedings of the 26th Annual International Conference on Machine Learning (New York, NY, USA), ICML '09, Association for Computing Machinery, 2009, p. 1049–1056.

[23] Rudolph Van Der Merwe, Arnaud Doucet, Nando De Freitas, and Eric Wan, *The unscented particle filter*, Advances in neural information processing systems **13** (2000).

[24] Joe Watson, Hany Abdulsamad, and Jan Peters, *Stochastic optimal control as approximate input inference*, CoRR **abs/1910.03003** (2019).

APPENDICES

0.1 Approximate Inference Control (AICO) algorithm

Input: start state x_0, control costs $H_{0:T}$, functions $A_t(x)$, $a_t(x)$, $B_t(x)$, $R_t(x)$, $r_t(x)$, convergence rate α, threshold θ
Output: trajectory $x_{0:T}$
initialize $s_0 = x_0$, $S_0^{-1} = 1e10$, $v_{0:T} = 0$, $V_{0:T}^{-1} = 0$, $r_{0:T} = 0$, $R_{0:T} = 0$, $k = 0$
repeat
  for $t = 1 : T$ **do** *// forward sweep*
    update s_t and S_t using (20)
    if $k = 0$ **then**
      $\hat{x}_t \leftarrow s_t$
    else
      $\hat{x}_t \leftarrow (1 - \alpha)\hat{x}_t + \alpha b_t$
    end if
    access $A_t(\hat{x}_t)$, $a_t(\hat{x}_t)$, $B_t(\hat{x}_t)$, $R_t(\hat{x}_t)$, $r_t(\hat{x}_t)$
    update r_t and R_t using (22)
    update v_t and V_t using (21)
    update b_t and B_t using (19)
    if $|\hat{x}_t - b_t|^2 > \theta$ **then**
      $t \leftarrow t - 1$ *// repeat this time slice*
    end if
  end for
  for $t = T - 1 : 0$ **do** *// backward sweep*
    ..same updates as above...
  end for
  $k \leftarrow k + 1$
until convergence

Figure 48: The Approximate Inference Control algorithm [22].

0.2 Product of multivariate Gaussians

A Gaussian distribution with mean μ and covariance Σ is given as

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{\det(2\pi\Sigma)^{\frac{1}{2}}}\exp((x-\mu)^T\Sigma^{-1}(x-\mu)) \tag{A.1}$$

$$= \frac{1}{\det(2\pi\Sigma)^{\frac{1}{2}}}\exp(x^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu - \mu^T\Sigma^{-1}x + \mu^T\Sigma^{-1}\mu) \tag{A.2}$$

Similarly it can be shown that,

$$\mathcal{N}(x|\mu_3,\Sigma_3) = \mathcal{N}(x|\mu_1,\Sigma_1)\mathcal{N}(x|\mu_2,\Sigma_2) \tag{A.3}$$

$$= \frac{1}{2\pi\det(\Sigma_1)^{\frac{1}{2}}\det(\Sigma_2)^{\frac{1}{2}}}\exp\big((x^T\Sigma_1^{-1}x - x^T\Sigma_1^{-1}\mu_1 - \mu_1^T\Sigma_1^{-1}x + \mu_1^T\Sigma_1^{-1}\mu_1) + (x^T\Sigma_2^{-1}x - x^T\Sigma_2^{-1}\mu_2 - \mu_2^T\Sigma_2^{-1}x + \mu_2^T\Sigma_2^{-1}\mu_2)\big) \tag{A.4}$$

$$\propto \exp\big(x^T(\Sigma_1^{-1}+\Sigma_2^{-1})x - x^T(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2) - (\mu_1^T\Sigma_1^{-1} + \mu_2^T\Sigma_2^{-1})x + (\mu_1^T\Sigma_1^{-1}\mu_1 + \mu_2^T\Sigma_2^{-1}\mu_2)\big) \tag{A.5}$$

Comparing eqns. (A.5) and (A.2) with respect to μ_3 and Σ_3

$$\mu_3 = \Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2 \tag{A.6}$$

$$\Sigma_3 = \Sigma_1^{-1} + \Sigma_2^{-1} \tag{A.7}$$

Since $\mathcal{N}[x|\mu,\Sigma] = \mathcal{N}(x|\Sigma^{-1}\mu,\Sigma^{-1})$. Substituting in A.5 we get

$$\mathcal{N}(x|\mu_4,\Sigma_4) = \mathcal{N}(x|\mu_1,\Sigma_1)\mathcal{N}[x|\mu_2,\Sigma_2] \tag{A.8}$$

$$\begin{aligned} &= \frac{1}{\det(2\pi\Sigma_1)^{\frac{1}{2}}\det(2\pi\Sigma_2^{-1})^{\frac{1}{2}}}\exp((x^T\Sigma_1^{-1}x - x^T\Sigma_1^{-1}\mu_1 - \mu_1^T\Sigma_1^{-1}x + \mu_1^T\Sigma_1^{-1}\mu_1) \\ &\quad + (x^T\Sigma_2 x - x^T\Sigma_2\Sigma_2^{-1}\mu_2 - (\Sigma_2^{-1}\mu_2)^T\Sigma_2 x + \mu_2^T\Sigma_2\Sigma_2^{-1}\mu_2)) \end{aligned} \tag{A.9}$$

$$\begin{aligned} &\propto \exp(x^T(\Sigma_1^{-1}+\Sigma_2)x - x^T(\Sigma_1^{-1}\mu_1+\mu_2) - (\mu_1^T\Sigma_1^{-1}+\mu_2^T)x \\ &\quad + (\mu_1^T\Sigma_1^{-1}\mu_1 + \mu_2^T\mu_2)) \end{aligned} \tag{A.10}$$

Comparing eqns. (A.10) and (A.2) with respect to μ_4 and Σ_4 we get

$$\mu_4 = (\Sigma_1^{-1}+\Sigma_2)^{-1}(\Sigma_1^{-1}\mu_1+\mu_2) \tag{A.11}$$

$$\Sigma_4 = (\Sigma_1^{-1}+\Sigma_2)^{-1} \tag{A.12}$$

VITA

Shahbaz Peeran Qadri Syed

Candidate for the Degree of

Master of Science

Thesis: DEVELOPMENT OF A ROBOTIC ARM–GANTRY SIMULATOR WITH PROBABILISTIC INFERENCE BASED CONTROL

Major Field: Mechanical and Aerospace Engineering

Biographical:

Education:

Completed the requirements for the Master of Science in Mechanical and Aerospace Engineering at Oklahoma State University, Stillwater, Oklahoma in May 2022.

Completed the requirements for the Bachelor of Science in Mechanical Engineering at Osmania University, Hyderabad, Telangana, India in 2019.

Experience:

Graduate Research Assistant, Control, Robotics and Automation Laboratory, Oklahoma State University.

Graduate Teaching Assistant, School of Mechanical and Aerospace Engineering, Oklahoma State University.