UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Technology
Robotics and Computer Engineering Curriculum

Iryna Hurova

Model-based planning using GPU-accelerated Simulator as a World Model

Master's Thesis (30 ECTS)

Supervisors: Karl Kruusamäe, PhD
Arun Kumar Singh, PhD

Tartu 2025

Model-based planning using GPU-accelerated Simulator as a World Model

Abstract: Manipulator robots are increasingly deployed in real-world tasks that require smooth, reactive motion and robust collision avoidance, particularly in dynamic and unstructured environments. This thesis presents a model-based, collision-free, online trajectory optimization framework tailored for such scenarios. The method involves sampling hundreds of trajectories from a multivariate normal distribution, shaping them with Bernstein polynomials, and evaluating them in parallel within a MuJoCo simulation. These trajectories are then optimized using the cross-entropy method. The system achieves real-time, in-the-loop planning by integrating a model predictive control strategy. The experiment, both in simulation and in real-world tests, demonstrated successful manipulation in an environment with multiple obstacles. In addition, the framework supports flexible task objectives by adjusting the cost function, enabling goal-driven behavior under varying conditions.

Keywords: Model-based planning, model predictive control, cross-entropy method, online planning

CERCS: T125 Automation, robotics, control engineering

Mudelipõhine planeerimine, kasutades GPU-kiirendusega simulaatorit maailmamudelina

Lühikokkuvõte: Manipulatorroboteid kasutatakse üha enam reaalses maailmas ülesannete täitmisel, mis nõuavad sujuvat, reageerimisvõimelist liikumist ja tõhusat kokkupõrgete vältimist, eriti dünaamilistes ja struktureerimata keskkondades. Käesolev magistritöö esitleb mudelipõhist, kokkupõrgetevaba, käitusaegset trajektoori optimeerimise raamistikku, mis on kohandatud just sellisteks olukordadeks. Meetod hõlmab sadade trajektooride genereerimist mitmemõõtmelisest normaaljaotusest, nende kuju määramist Bernsteini polünoomide abil ning paralleelset hindamist MuJoCo simulatsioonis. Seejärel optimeeritakse trajektoorid ristentroopia meetodil. Süsteem saavutab reaalajas, töötsüklis planeerimise, integreerides mudeliprognoosiva juhtimisstrateegia. Eksperimendid, nii simulatsioonis kui ka reaalses maailmas, demonstreerisid edukat manipuleerimist mitmete takistustega keskkonnas. Lisaks võimaldab raamistik paindlikke ülesandeid kulufunktsiooni kohandamise kaudu, võimaldades eesmärgist lähutvat käitumist erinevates tingimustes.

Võtmesõnad: Mudelipõhine planeerimine, mudelipõhine ennustav juhtimine, rist-entropia meetod, reaalajas plaanimine

CERCS: T125 Automatiseerimine, robootika, juhtimistehnika

Contents

1 Abbreviations

CEM - Cross-Entropy Method
MPC - Model Predictive Control
ROS - Robot Operating System
GPU - Graphics Processing Unit
CPU - Central Processing Unit
RRT - Rapidly-exploring Random Tree
PRM - Probabilistic Roadmaps
CHOMP - Covariant Hamiltonian Optimization for Motion Planning
DRL - Deep Reinforcement Learning
RL - Reinforcement Learning
DOF - Degrees of Freedom
PC - Personal Computer
MB-planner - Model-Based planner
MPPI - Model Predictive Path Integral

2 Introduction

2.1 Background

Motion planning and control are critical blocks of the manipulation pipeline. At the most fundamental level, planning and control require access to a world model that captures how actions/control inputs affect the manipulator and its interaction with the environment. In Reinforcement Learning (RL), a world model refers to any system that can predict the outcomes of actions in an environment, essentially, it models how the world responds to the agent's behavior. Typically, developing analytical and symbolic models for the world model requires expert knowledge of Euler-Lagrange dynamics. For non-expert users, these can be difficult to derive or prone to errors. Alternately, it is possible to learn the world model through interaction, but these have their own challenges in terms of data collection and generalization. Physics-based simulators are a form of world model as they provide a structured and predictive environment that mimics the dynamics of the real world. A physics-based simulator can be used as a world model for trajectory evaluation in the optimization process. This eliminates the need for manually deriving complex dynamics equations, and provides more realistic contact dynamics and multi-body interactions.

The optimization process is computationally intensive, requiring the evaluation of many trajectory samples within a short time, which demands massive parallelization. With increased GPU availability, tackling computationally intensive tasks became much easier. GPU-accelerated parallel programming allowed for the fast and scalable solution of complex problems that require Massively Parallel Simulation. Nowadays, it is possible to run tens of thousands of environments simultaneously using a single GPU.

2.2 Objective and Contribution

This thesis explores the use of a black-box physics-based simulator as a world model for model-based (MB) planning and control. The goal is to design both offline and online planners that utilize a GPU-accelerated physics simulator to predict system dynamics and guide decision making (Figure 1). To address this, the following contributions are made:

- **Trajectory Optimization Using Sampling-Based Methods:** A sampling-based optimizer using the Cross-Entropy Method (CEM) [10] is implemented to explore and refine feasible trajectories for goal-directed motion, including contact tasks and obstacle avoidance.
- **Integration with MuJoCo Simulation:** The MuJoCo physics engine is used to evaluate sampled trajectories in parallel, enabling efficient performance analysis under realistic physical constraints. It is also demonstrated that the simulator

model can be updated in real-time using motion capture data. This allows reactive adaptation to changes in the environment.

- **Model Predictive Control (MPC) for In-Loop Execution:** The CEM planner is used for real-time planning in a receding horizon fashion to simulate a feedback controller capable of adapting to dynamic environments.
- **Task Flexibility through Cost Function Design:** By adapting the cost function, the system can be tailored to various task objectives, including complex manipulations beyond simple point-to-point motion.
- **Real-World Validation on a UR5e Robot:** The full pipeline is tested on a real UR5e manipulator in a dynamic environment. A motion capture system is used to synchronize the simulation with the real world.

These contributions demonstrate a scalable and adaptable solution for motion planning in complex robotic environments and bridge the gap between high-performance simulation and real-world application.

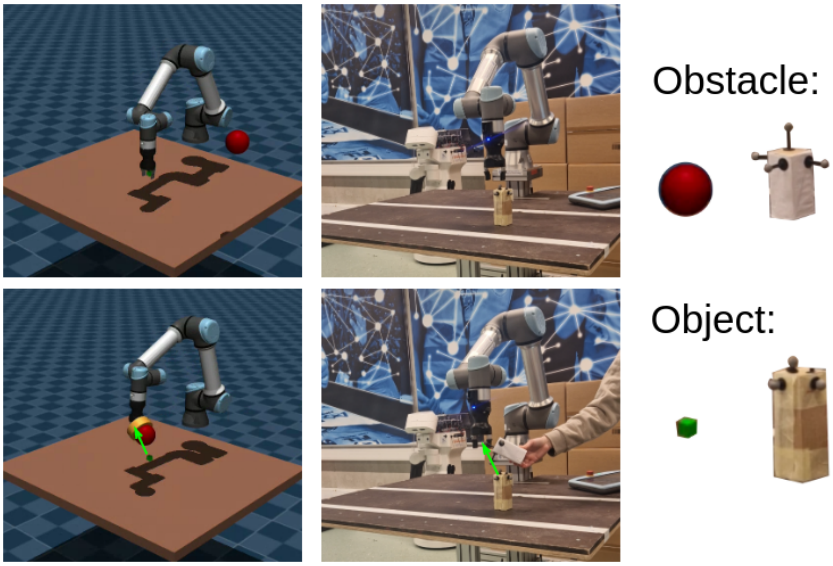


Figure 1. Images on the right show the robot controlled by the model-based planner in online fashion, and on the left are the snapshots of the simulated environment at the same time. The manipulator robot is drawn to the object (top), but backs off when the obstacle interferes (bottom).

2.3 Organization of Thesis

The Section 3 covers the current advancements and different approaches to dynamic path planning. The Section 4 describes in detail all of the approaches used in this thesis, including MuJoCo simulation, optimization with MB-planner, and real-time planning with MPC. The Section 5 presents performance evaluations along with descriptions of each test scenario and the outcomes observed during execution. In the Section 6, the interpretation of results is carried out with the mention of limitations and possible improvements. Finally, the Section 7 sums up the work that was done.

3 Literature Review

The rapid advancement in the robotics field brought a handful of challenges that are being tackled in the latest research papers [2]. The hard-coded solutions and robots executing predefined actions were good enough for Industry 3.0 when robots replaced humans on the conveyor lines. However, Industry 4.0 and Industry 5.0 require much more sophisticated solutions [25]. Mobile robots must be able to navigate through changing environments with dynamic obstacles. The humanoid and quadruped robots must be able to keep balance and navigate through complex terrain. The manipulator robots must be able to solve tasks that require fine motor skills. All the cases mentioned above must be able to complete the defined task while avoiding collisions and making quick decisions based on sensory input [7, 22]. Currently, the most widely used approaches to address these challenges are trajectory optimization and reinforcement learning, or a combination of both. Although these methods have only become available recently due to their high computational complexity, they are now commonly utilized to solve hard-to-engineer behaviors with a high degree of uncertainty [7, 28].

3.1 Manipulator Robot Path Planning

Collaborative robotic manipulators are becoming increasingly popular in fields where navigation in dynamic environments is essential. This led to many research publications exploring possible path planning strategies in complex settings [12]. Motion planning for manipulator robots refers to the process of determining a sequence of valid movements that the manipulator must perform to achieve a specific goal without collisions and within physical constraints. In other words, moving from one state (usually the current state) to another, where a state can be defined by configuration (joint positions, velocities, forces, torques, etc.) or pose (Cartesian position and/or orientation of a particular location

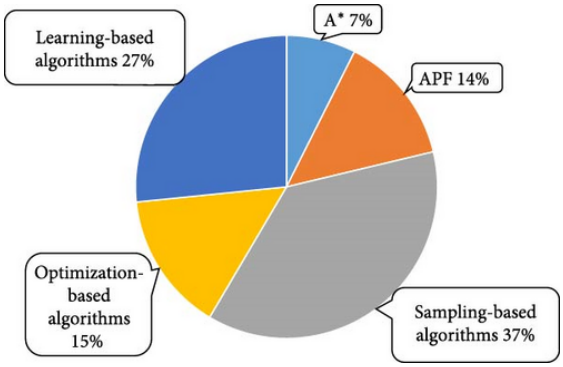


Figure 2. The most used methods for planning in a dynamic environment [12]

.

on the robot, and similar derived quantities) [6]. The planned path must be free of self-collisions and collisions with objects in the surrounding environment. Additionally, the planner must consider constraints, including limitations of joint angles, velocities, and accelerations, as well as any task-specific requirements, e.g., maintaining a particular tool orientation or specific path. In dynamic or uncertain environments, more advanced planners incorporate feedback, using techniques such as receding-horizon control to adjust trajectories in real-time [6]. According to the recent review of motion planning [12], the most used approaches for the past seven years are sampling-based algorithms, learning-based algorithms, and optimization-based algorithms (Figure 2).

3.1.1 Grid-based method

Grid-based algorithms and combinatorial approaches such as A* became computationally intractable with increased configuration space dimension [18]. Grid-based approaches discretize the world into a grid of cells and perform planning by searching over the grid to find a feasible path. In the context of manipulator robots, the space is divided along each joint's degree of freedom (DOF) into discrete intervals. For the robots with higher DOF (e.g., 6), this leads to the problem known as the curse of dimensionality, when the number of grid points along with computation time grows exponentially with the number of DOF [6]. On the other hand, the rest of the algorithms shown in Figure 2 proved to be more efficient in high-dimensional space [12].

3.1.2 Sampling-based method

Sampling-based algorithms do not require an explicit discretization of the entire space, making them well-suited for manipulator robots with many degrees of freedom. Rapidly-exploring Random Tree (RRT) works by randomly sampling the points in the robot's operation space and connecting them in such a way that the robot could follow to reach the goal. The downside of this method is that it requires postprocessing to smooth out the path [9]. Probabilistic Roadmaps (PRM) construct a roadmap of valid configurations during an offline phase and then use it during runtime. This method faces some challenges when used in dynamic environments, since it reuses the graph generated based on the static environment [13]. Both methods may struggle in environments with narrow passages and require more sophisticated solutions for better convergence in more cluttered spaces [9, 8].

3.1.3 Optimization-based method

In optimization-based motion planning, the desired trajectory is achieved by minimizing the predefined cost function. With a properly defined cost function, it is possible to achieve a smooth, time-efficient, and collision-free path [12]. There are multiple

approaches used to solve the optimization problem. Covariant Hamiltonian Optimization for Motion Planning (CHOMP) starts with an initial trajectory and iteratively improves it to minimize a cost functional. This method, however, assumes the collision-free initial state. The TrajOpt method works similarly to CHOMP, but can converge to solutions even with initial collisions [24]. The optimization-based methods are often used in combination with Model Predictive Control (MPC) to plan the trajectory in a dynamically changing environment [5]. MPC is a control strategy that uses a model of the system to predict future behavior and optimize control actions over a fixed time horizon. At each timestep, MPC solves an optimization problem, applies the first control input, and repeats the process. It can run continuously unless a stopping condition is defined. This makes MPC suitable for dynamic environments.

3.1.4 Learning-based method

The learning-based method, such as the Deep Reinforcement Learning (DRL) technique, has been actively studied since 2017. It is intended to battle the common issues of sampling- and optimization-based algorithms, such as falling into the local minima and dynamically adaptive collision avoidance actions [4]. With the use of DRL, robots learn from examples, data, and experience, enhancing their decision-making capabilities. In reinforcement learning (RL), an agent learns to interact with its environment to maximize rewards over time. Ideally, RL methods should allow agents to generalize learned behavior to new, unseen scenarios; however, this is still an ongoing challenge in the motion planning field [4, 12].

3.1.5 Hybrid approaches

Current research also shows an emerging trend of hybrid approaches that combine the methods mentioned above. This helps to balance out the speed of compilation with the quality of the path [12]. In the paper by L. Wang et al. [29], the authors use trajectory optimization to find a locally better, smoother, lower-cost trajectory for the "supervised data" for training. By using trajectory optimization during training, it is possible to acquire a lot more useful learning out of just a small amount of data. Instead of struggling to learn directly from noisy, imperfect actions, it first improves the collected data into better trajectories and then learns from those. This makes training more stable, more efficient, and requires far fewer samples, which is crucial for applying RL on real robots where collecting data is expensive. The paper by J. Leu et al. [11] combines sampling- and optimization-based approaches. This helped to conquer the cons of both methods. The first layer samples a semi-optimal path, which will not be smooth nor account for dynamic obstacles. However, this path is sufficient input to the optimization algorithm, which iteratively improves the trajectory to satisfy the constraints (e.g., collision avoidance, smoothness).

3.2 Massively Parallel Simulation

Simulations in robotics, particularly those involving reinforcement learning and trajectory optimization, are often highly computationally intensive. For example, in MPC, computing the gradient of rigid body dynamics accounts for 30% to 90% of the total computational time [21]. This is because these methods involve running multiple environments in parallel. This may come across as an issue when quick iterations and real-time performance are crucial.

The usual approach has been to combine CPU and GPU: CPU was used to simulate environment physics, calculate rewards, and run the environment, while the GPU was used for accelerating the neural networks and rendering. The downside of this approach is the constant data transfer between CPU and GPU, which leads to a high latency [14]. To address this bottleneck, popular physics engines like NVIDIA Isaac [17] and MuJoCo [15] each offered their end-to-end GPU-accelerated solutions. MuJoCo introduced MuJoCo XLA (MJX), a JAX-based branch of the MuJoCo physics engine that runs on GPU [16]. And NVIDIA Isaac introduced Isaac Gym [14].

3.3 Sampling Based Optimization

Sampling-based optimization is a gradient-free method that operates by sampling control and trajectories from a distribution and iteratively refining them based on the observed cost of the sampled trajectories and controls. This class of methods has several advantages that make them suitable for manipulation. [20]:

- the possibility of optimizing black-box functions;
- lower sensitivity to hyperparameter tuning and thus higher robustness;
- no requirement of gradient information;
- suitable for non-differentiable cost functions.

In the paper by Cristina Pinneri et al. [20], the authors were investigating whether it is possible to use CEM for real-time planning and control. They have proposed a faster, more sample-efficient, and higher-performing version of the CEM algorithm - iCEM. With the ground-truth models and CPU-parallelization, they have achieved close to real-time performance, and with learned models, the real-time performance was reached. The authors of the paper by Mohak Bhardwaj et al. [1] introduced a sampling-based MPC framework for manipulation that operates in the joint space and can achieve smooth and reactive motions while respecting constraints. To approximate the model during rollout, they use the robot's joint-space kinematics. The real-time performance was reached with a highly parallelized and GPU-accelerated control architecture. For self-collision avoidance, they trained a neural network that predicts the closest distance between robot

links given a joint configuration. The authors achieved a computation time of less than 8 ms per step. However, significant model bias was introduced by a very simplistic model approximation. In the paper by Emanuel Todorov et al. [27], the authors present a model-based control application that utilizes the MuJoCo physics engine. They demonstrate real-time performance with CPU-parallelization. However, since they use the older version of MuJoCo, parallelization capabilities are constrained by the number of CPU threads.

3.3.1 Sampling-Based Optimization Coupled With GPU-accelerated Simulators

Sampling-based optimizers are well positioned to leverage massively parallel simulators like MuJoCo-MJX. Intuitively, the rollouts of the world model for a given sequence of actions and the cost evaluation can be done in parallel. In the paper by Corrado Pezzato et al. [19], the authors propose integrating the Model Predictive Path Integral (MPPI) controller with NVIDIA's IsaacGym, a GPU-parallelizable physics simulator. They demonstrate real-time control of robots in dynamic environments. The method was validated through various simulated and real-world experiments, including mobile navigation, non-prehensile manipulation (pushing objects without grasping), and whole-body control. By relying on physics simulations, the method avoids the need for explicit dynamic modeling, making it easily adaptable to different robots and tasks. GPU acceleration enables the controller to operate in real-time, even in complex environments.

3.4 Contribution Over Existing Work

This project was built upon [19], but the sampling-based method used in the proposed work allows the adaptation of the covariance of the sampling distribution. Moreover, unlike [19], the thesis uses MuJoCo MJX as the physics engine, which was observed to provide faster parallelization and a simpler interface for coupling with a motion capture system for real-world implementation.

4 Methodology

The thesis aims to develop a model-based planner (MB-planner) for manipulation in cluttered environments and contact tasks, leveraging the GPU-parallelizable physics simulation. Two different approaches are adopted (Figure 3). In the first case, a complete trajectory from start to goal is computed and executed in open-loop on the manipulator. This approach is robust enough for simple pick-and-place tasks, even in cluttered environments. The second approach adopts a receding horizon template, which is often referred to as Model Predictive Control (MPC) in the existing literature. In this case, a short trajectory is computed based on the current state (position, velocity) of the manipulator. A small part of this trajectory (often only the first velocity in the trajectory sequence) is executed on the robot. The planner is again invoked from the new manipulator state. The underlying requirement in MPC is that the planner is computationally fast to compute/refine optimal trajectories as the manipulator is moving in the environment. To ensure this low latency in planning, often the trajectories are computed for a very short horizon (0.4s in this work).

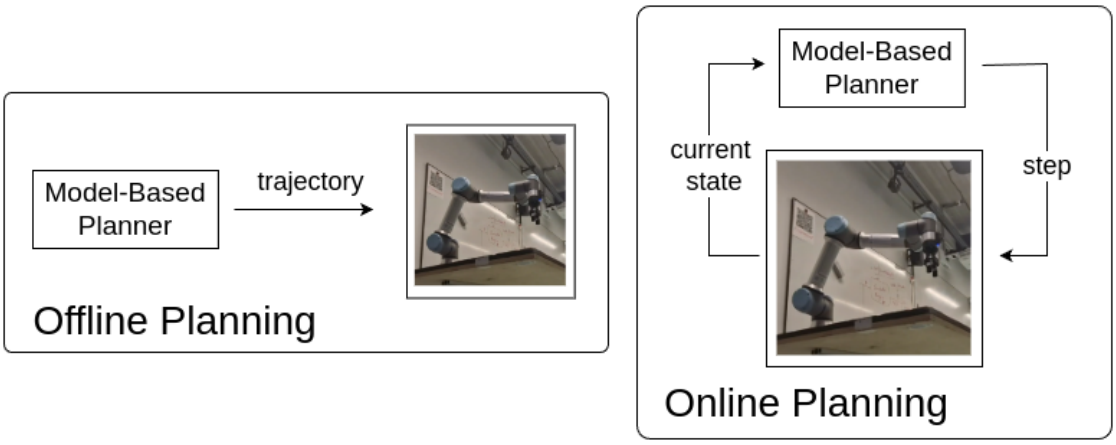


Figure 3. Two ways of using a MB-planner. Left: directly use a MB-planner to optimize the whole trajectory (offline planning); Right: use a MB-planner to find the next optimal step (online planning).

A detailed pipeline of the online planning with the MB-planner process is illustrated in Figure 4. The current state of the robot is passed to the MB-planner. This includes current joint angles, current joint velocities, and the positions of dynamic objects in the scene. This state information serves as one of the constraints during trajectory sampling. A batch of candidate trajectories is then generated and evaluated in parallel using GPU-accelerated MuJoCo simulations. The data obtained from the rollouts is used to compute costs for each sample. The set of elite samples is computed by taking the samples with the lowest cost and is used to calculate the new mean and covariance for the

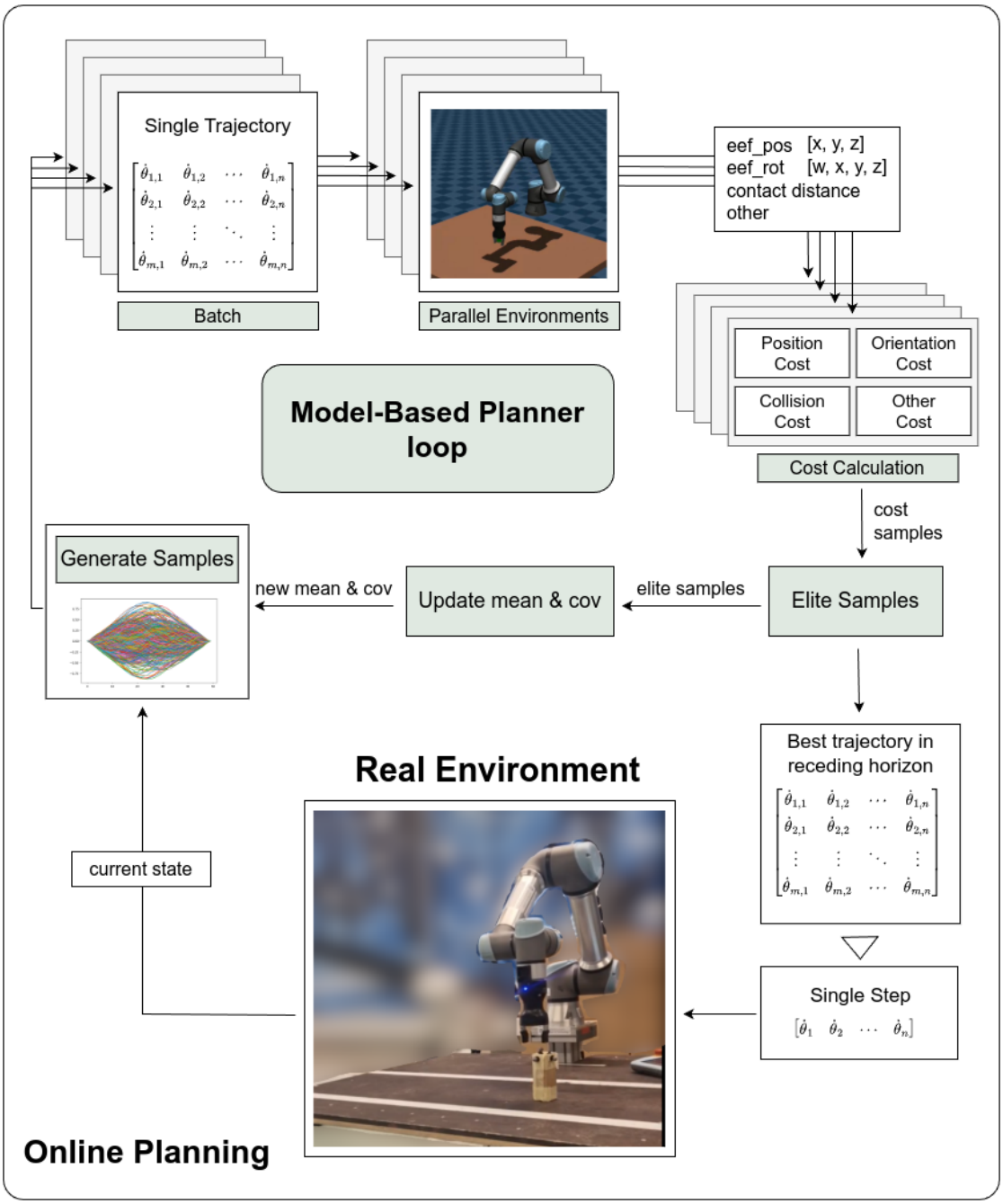


Figure 4. Model-based framework for real-time in-loop planning: the current state updates the planner, which generates and evaluates trajectory samples in parallel. Costs are computed for the entire batch to select elite samples, refining the sampling distribution. The best trajectory is applied as control to the real robot.

sampling distribution. This sampling and evaluation loop is repeated for a fixed number of iterations. Finally, the optimal trajectory is selected and used to generate joint velocity commands for the next step of the real robot.

Information related to the MuJoCo simulator can be found in Section 4.1. The sampling procedure is explained in Section 4.2.1, while the evaluation process and the update of the sampling distribution are covered in Section 4.2.2. The integration of the MB-planner with the real-world environment is described in Section 4.2.3.

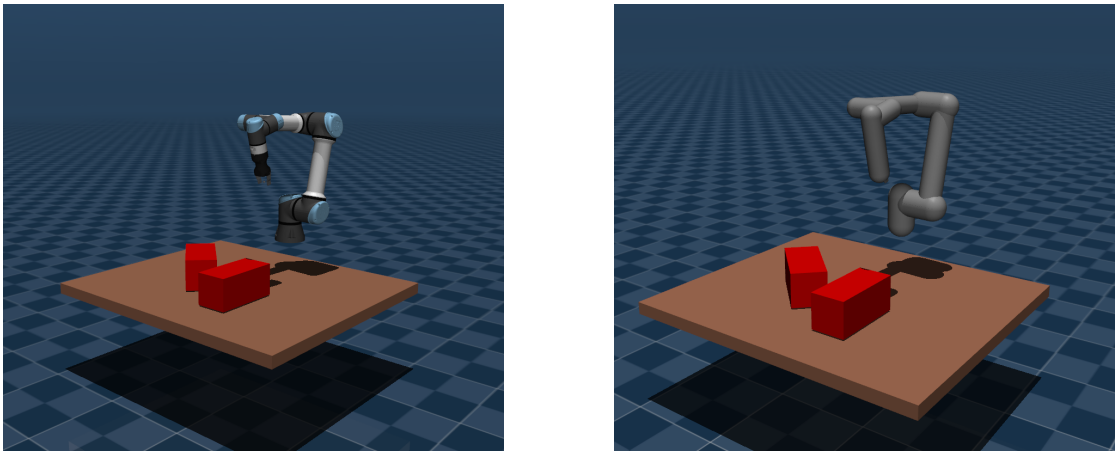
4.1 Simulation

MuJoCo is an advanced physics simulator for model-based optimization. It offers a wide variety of options that users can modify to fit the needs of their project. It can accurately handle rigid body dynamics, contact and friction, and joint constraints (e.g., limits, motors, springs). In this project, MuJoCo simulation is used as a world model for the evaluation of the generated samples. Its accurate dynamics simulation allows for a realistic prediction of how each sampled trajectory would behave in the real world. MuJoCo allows the retrieval of useful data, such as contact information for collision avoidance and the position and orientation of the end effector for cost calculation. Another key advantage of MuJoCo is the MJX, which enables the evaluation of thousands of trajectories in under a second. This capability is essential for real-time planning with MPC, where each step requires sampling and evaluating at least hundreds of trajectories to ensure sufficient exploration of the solution space.

4.1.1 Creating a Model

MuJoCo uses its own XML model format called MJCF [30]. For the XML file to be identified as an MJCF model, a <*mujoco*> top-level element has to be added to the beginning of the document. The rest of the elements are optional. Some of them can be repeated, and some can appear only once. Typically, the *scene.xml* file is created. This is where the initial settings for the compiler, simulation, and an empty world are placed. The <*compiler*> element is used to set options for the built-in parser and compiler. That's where the mesh directory can be set. The simulation options can be set with <*option*> element and can also be modified at runtime. With this element, one can set the simulation timestep, choose the solver, define the number of iterations the constraint solver will run per simulation step, and much more. The <*option*> element settings will significantly affect the simulation time and accuracy. The *scene.xml* usually contains three elements that define the initial environment: <*visual*>, <*asset*>, and <*worldbody*>. The <*visual*> tag configures the global appearance of the simulation, overall rendering behavior, and non-physical visuals. The meshes, textures, and materials are defined inside the <*asset*> element. The <*worldbody*> is the root of the kinematic tree, and all other bodies are defined inside it.

When the initial setup is done, the robot and other objects of an environment are imported with the <*include*> tag (Figure 5a). The robot's kinematic chain is created with nested <*body*> elements. Each body contains physical properties (e.g., mass, inertia), joints (e.g., rotational, sliding), and geometry (geom) for visualization and collision checking. The body can be set to be a mocap body, which makes it fixed from the viewpoint of the dynamics, and movable only with manual location modification. It is also possible to define reusable properties for the robot's joints, visuals, actuation parameters, and geometry settings. These settings can be grouped into subclasses and reused throughout the chain to maintain consistency. There are many options to define specific joint actuation mechanisms. For example, the fingers of the gripper could be set to move in a coordinated and symmetric way with <*equality*> and <*tendon*> tags.



(a) Scene with invisible collision geoms.

(b) Scene with visible primitive collision geoms.

Figure 5. MuJoCo simulated environment.

MuJoCo has specific rules on how to handle contacts. They are controlled via a combination of body geometry settings (<*geom*>), exclusion rules (<*contact*>, <*exclude*>), material and contact parameters, and settings in <*option*>. Two geoms can collide if:

- they do not belong to the same body
- they do not belong to the body of a parent or a child
- are not welded together (they have joints between them)
- contype & conaffinity is non-zero between two geoms

The contype defines what type of contacts a particular geom can generate, while the conaffinity defines what types of contacts it can receive. The robot's meshes are usually complex and can have non-convex structures. For a faster and more stable simulation,

they must be decomposed into a union of convex geoms. The primitive shapes that are available in MuJoCo (e.g., sphere, capsule, ellipsoid, cylinder, box) can be used to construct the simulation-friendly robot model (Figure 5b).

4.1.2 Utilizing MJX and JAX

Classic MuJoCo is running on a CPU, which limits its speed and makes it unsuitable for fast sampling-based planning. Methods such as MPC require evaluating many action sequences in parallel to identify the best one for the robot to execute. The GPU support is crucial to provide the necessary parallelism and acceleration. Starting with version 3.0.0, MuJoCo includes MuJoCo XLA (MJX), which benefits from running entirely on the GPU. While classic MuJoCo is implemented in C, MJX is implemented using JAX. This makes it compatible with just-in-time compilation via *jax.jit*. When a function is wrapped with *@jit*, JAX compiles it into highly optimized machine code with the XLA compiler. There is no CPU-GPU transfer bottleneck. Everything from dynamics to cost computation stays on the GPU.

Before executing MJX functions on an accelerator, the MuJoCo model and associated data must be copied to the device. All functions, including *mjx.step*, that are used multiple times must be wrapped with *@jit* (Figure 6). The first function call will begin the compilation process, incurring some overhead, while subsequent calls are highly optimized and executed much faster.

The parallelization is done with *jax.vmap* - the vectorizing map (Figure 6). *jax.vmap* transforms a function that operates on a single input into one that operates simultaneously on a batch of inputs, enabling efficient, batched computation. This function is well-suited for the evaluation of multiple sampled trajectories in parallel.

In some cases, sequential execution is necessary. For example, the evaluation of a single trajectory, where each subsequent step depends on the previous one. Even though native Python loops are jit compatible, it is not recommended to use them since they can slow down the simulation significantly. The JAX alternative to Python loops is *jax.lax.scan* (Figure 6). Unlike *jax.vmap*, which enables parallel evaluation over batches, *jax.lax.scan* performs computations sequentially over a series of inputs. A specific structure must be followed to use *jax.lax.scan*. The function accepts three main arguments:

1. A loop body function — the function to be applied sequentially over iterations.
2. The carry value — a piece of data that is passed and updated through each iteration.
3. The sequence — the data to be iterated over, analogous to a loop index or input at each time step.

The output of *jax.lax.scan* is a tuple consisting of two parts:

1. The final carry value, which represents the result of the last iteration.

2. The stacked outputs from each iteration are returned as a JAX array. This array accumulates the per-step outputs over the entire loop.

```
@jax.jit
def bar(carry_init, single_input):
    carry, output = do stuff
    return carry, output

@jax.jit
def foo(inputs):
    carry_out, outputs = jax.lax.scan(bar, carry_init, inputs)
    return outputs

batched_outputs = jax.vmap(foo)(batched_inputs)
```

Figure 6. Example of using *jax.jit*, *jax.vmap* and *jax.lax.scan*

4.2 Model-Based Planning

In this work, a model-based planning algorithm is proposed that utilizes the Cross-Entropy Method (CEM) for optimizing velocity inputs in robotic systems (Algorithm 1). The algorithm begins by initializing the system's state, including the robot's position, velocity, and the positions of surrounding objects, and sets the maximum number of iterations for the optimization. At each iteration, a set of candidate velocity inputs is sampled from a multivariate normal distribution characterized by a mean and covariance, which are iteratively refined. These candidate inputs are constrained using Bernstein basis polynomials, and the resulting trajectories are simulated to compute a cost function that reflects the system's performance. The top elite samples, those with the lowest cost, are selected, and their weighted contributions are used to update the mean and covariance of the sample distribution. This process of sampling, evaluating, and updating continues for a predefined number of iterations, allowing the algorithm to converge to an optimal set of velocity inputs. The result is an optimized control input that minimizes the cost function while respecting system constraints, making this approach suitable for tasks requiring efficient and precise trajectory planning in dynamic environments.

4.2.1 Sampling

In the sampling step, the objective is to generate valid joint velocities at each time step of the trajectory. To ensure the robot can follow the path accurately, the generated trajectories should be continuous and smooth. This can be achieved by sampling from a multivariate normal distribution, which models a vector of jointly normally distributed random variables. It is defined by a mean vector and a covariance matrix, allowing for

Algorithm 1: Model-Based Planner

Input: s_{init} :
Initial simulation state (θ_{init}, $\dot{\theta}_{init}$, *position of objects in the scene*)

Output: $\dot{\theta}$: optimized velocity inputs

$N =$ Maximum number of iterations

Initiate mean μ_i and covariance Σ_i at $i = 0$

for $i = 1, i < N, i++$ **do**

Draw n samples from the multivariate normal distribution $\mathcal{N}$

$\xi \sim \mathcal{N}(\mu_i, \Sigma_i)$

Apply constraints on ξ using Bernstein basis polynomials and derive $\dot{\theta}$

$\dot{\theta} \leftarrow \xi$

Compute rollout and cost C in parallel for n samples

$\theta \leftarrow$ `COMPUTE_ROLLOUT`$(\dot{\theta}, s_{init})$

$C \leftarrow$ `COMPUTE_COST`(θ)

Select top E samples of ξ with the lowest cost C

ξ_{elite}, $C_{elite} \leftarrow$ `COMPUTE_ELITE_SAMPLES`(C, ξ)

Calculate new mean μ and covariance Σ

$$w = \exp\left(-\frac{1}{\lambda}(C_{elite} - \min(C_{elite})\right) \tag{1}$$

$$\mu_{i+1} = (1 - \alpha_\mu)\mu_i + \alpha_\mu \cdot \frac{\sum_{k=0}^{E} w_k \xi_k}{\sum_{k=0}^{E} w_k} \tag{2}$$

$$\Sigma_{i+1} = (1 - \alpha_\Sigma)\Sigma_i + \alpha_\Sigma \cdot \frac{\sum_{k=0}^{E} (\xi_k - \mu_{i+1}) \otimes (\xi_k - \mu_{i+1})^\top w_k}{\sum_{k=0}^{E} w_k} + \varepsilon I \tag{3}$$

Return $\dot{\theta}$ with the lowest cost C from the last iteration's batch

return $\dot{\theta}$

correlated variations across time steps and joints. To make sure that the robot will not get stuck at the local minima, the initial covariance should be wide enough to cover the space of interest.

Besides smoothness and continuity, the generated trajectory must also satisfy various constraints, including limits on joint velocity, acceleration, position, and boundary conditions. For example, the initial joint positions and velocities have to match the

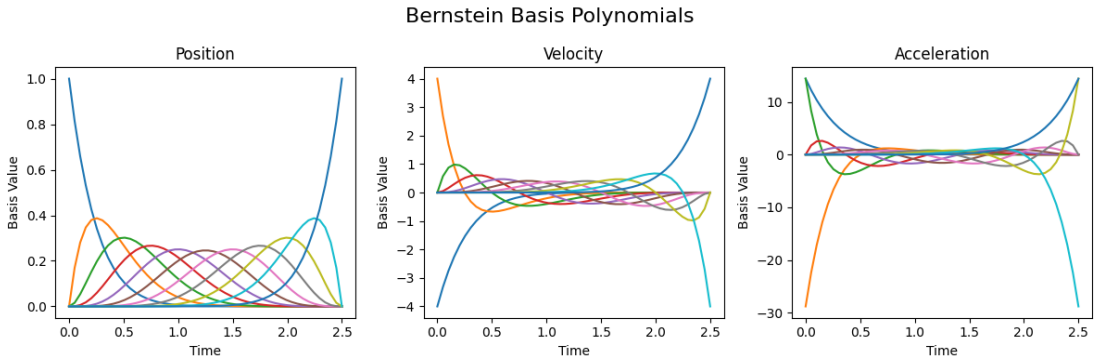


Figure 7. The 10-order Bernstein basis polynomials for position, velocity, and acceleration.

current state of the simulated robot, and final joint velocities have to be 0. These constraints can be effectively enforced using Bernstein basis polynomials. Bernstein polynomials are computed over the entire trajectory time for joint position, velocity, and acceleration (Figure 7). They are later used to calculate the coefficients that will act as the attractors for the trajectory lines, pulling the path toward preferred values. To further

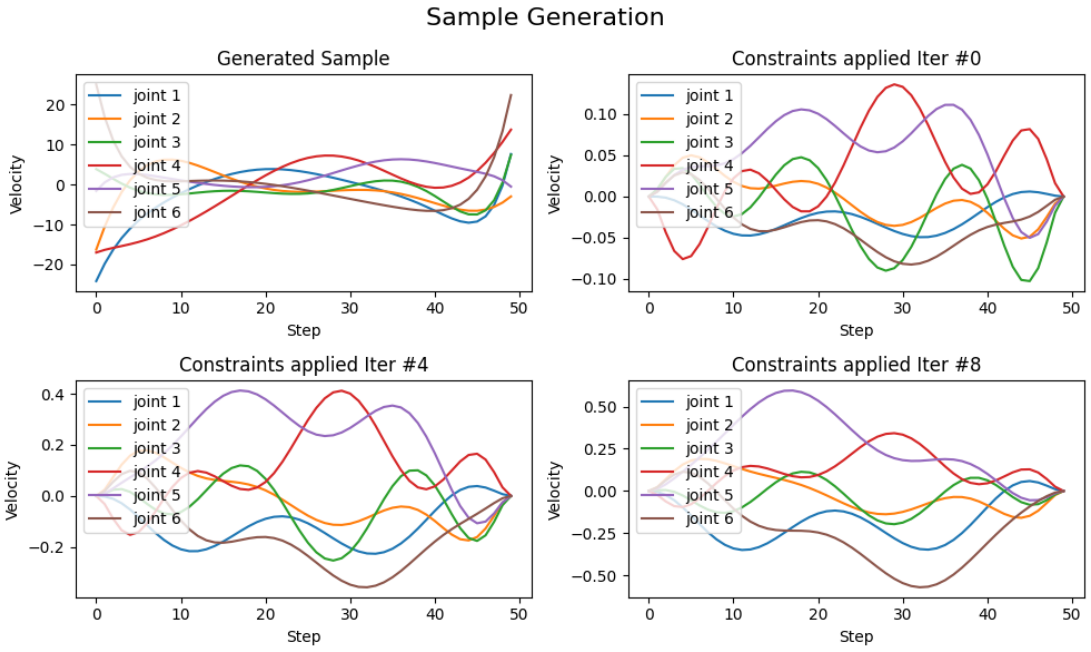


Figure 8. The process of sample generation. Top-left: the sample trajectory generated with a multivariate normal distribution; Other: iterations of the constraints application.

refine the trajectory, iterative sample modification can be applied, progressively adjusting sampled trajectories to satisfy constraints better and improve the result (Figure 8).

4.2.2 Sample evaluation and sampling distribution update

Once samples have been generated, they need to be evaluated. The rollout is computed for each trajectory (a.k.a sample, batch) in parallel with *jax.vmap*. At the beginning of each rollout, the robot's joint states are set to the initial joint position and velocity. Then, *jax.lax.scan* is used to loop through the trajectory and step up the simulation. For the following cost computation, the data needs to be collected after each step. The content of the collected data depends on how the cost function is defined. If the goal is for the robot to reach the target without collisions, the return from the rollout should be at least the position and rotation of the end effector and contact information that can be extracted from *mjx.Data*. When all the rollouts are done and the data is received, the cost can be computed. The cost computation is also done in parallel using *jax.vmap*. The cost function is built with multiple equations that evaluate how well the robot satisfied the objective with the executed trajectory. The samples with the smallest cost are claimed to be elite samples. The number of elite samples can be set during the initialization of the planner. The elite samples are then used to update the mean and covariance of the multivariate normal distribution. In the Equations 1 - 3, λ is a parameter that controls how much influence each elite sample has on the distribution update, C_{elite} is the cost of elite samples ξ_{elite} used to calculate the samples' weights w (Eq. (1)). Those weights are used to emphasize better-performing trajectories while still giving some weight to suboptimal ones. The weighted sum of current and previous values is computed for mean μ (Eq. (2)) and covariance Σ (Eq. (3)). The current mean is calculated as a weighted average of the elite samples. The updated covariance matrix is computed by calculating how much each elite sample deviates from the updated mean. The current covariance is the weighted average of outer products of the difference between elite samples and the current mean, with an added regularization term εI where ε is a constant and I is the identity matrix with the shape of Σ.

4.2.3 Sim-to-Real

The planner's functionality was tested with a real UR5e. OptiTrack was used for object and obstacle tracking. OptiTrack is a motion capture system that can track the rigid bodies with the track markers attached to them. The infrared (IR) cameras emit and detect IR light reflected from the markers. The system triangulates 3D positions by combining 2D views from multiple calibrated cameras. When multiple markers are attached to an object in a known configuration, OptiTrack tracks the pose (position + orientation) of that rigid body. To transfer the object's pose information from the computer with the OptiTrack system to the one controlling the manipulator robot, the

ROS2 package *vrpn-mocap* [23] was used. The robot is controlled with the *URX* Python library [26]. The *speedj* method from the *URX* library is used to control the speed of the robot's joint velocities. Figure 9 illustrates the setup for model-based planning in real life.

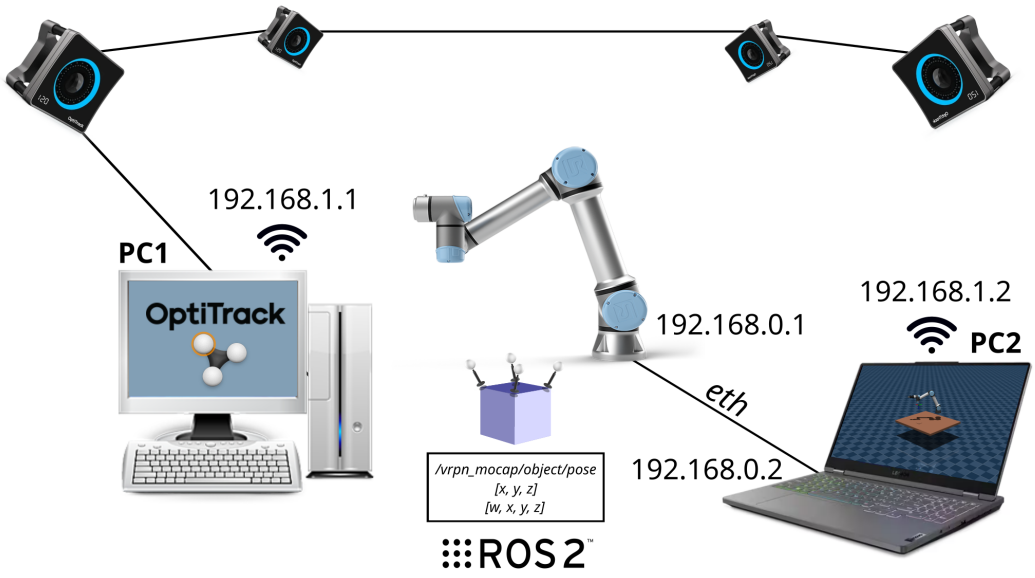


Figure 9. The setup for the real-life planning with the MB-planner. The PC1 with the OptiTrack detects the object with the markers attached. The position and orientation information is sent wirelessly to the PC2, which runs a MB-planner and controls the robot.

5 Experiments and Results

5.1 Setup

- Universal Robots Manipulator robot UR5e + HandE Gripper
- OptiTrack
- Lenovo Legion 5 15ACH6, NVIDIA GeForce RTX 3050 Ti 4GB
- Ubuntu 24.04 LTS

5.2 Performance

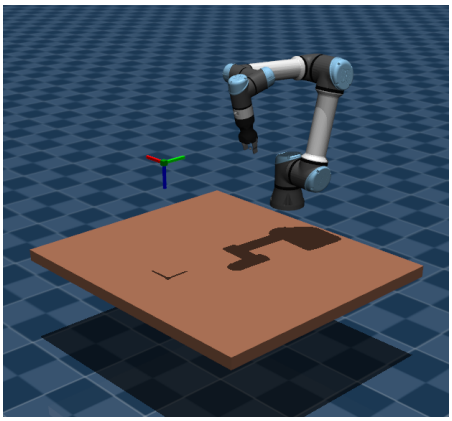


Figure 10. Scene for performance testing.

For the real-time execution, it is important to have a reasonable computation time. In this section, the performance tests were conducted to see how different parameters influence MB-planner computation time. The computation time refers to the time it takes to run the subsequent MB-planner calls after the initialization of the optimizer and computation of the first call are done. As explained in Section 4.1.2, the first call of MB-planner has to be excluded from evaluation since it is subjected to some overhead due to JAX compilation. The main parameters that affect the time are:

- Batch size - the number of samples generated and evaluated in parallel during one MB-planner iteration.
- Number of iterations - number of MB-planner iterations per one MB-planner optimization call.
- Number of steps - the length of the trajectory.

All test cases were conducted in a simple environment with one target and no obstacles (Figure 10). The cost was formulated according to Equations 4-8, which, in this case, simply makes the robot navigate towards the goal. In each test case, 10 increasing parameter values were evaluated. To test the influence of batch size on MB-planner performance, the values from 500 to 5000 with an increment of 500 were used (Table 1). The rest of the parameters were constant and chosen because they are actual valid parameters for the MPC. The effect of trajectory length (number of steps) and number of iterations was tested with an increment of 5 (Table 2, 3). Although longer trajectories would require a higher number of MB-planner iterations to converge and a larger batch size for better space exploration, for the sake of the performance test, it was set to be constant.

Table 1. Batch size influence.

Name	**Value**	**Name**	**Value**
Batch Size:	500 to 5000 (step 500)	Elite samples (%):	0.05
Number of steps:	8	Pos weight:	1
Number of iterations:	1	Rot weight:	0.5
Simulation timestep:	0.05	Collision weight:	10

Table 2. Trajectory length influence.

Name	**Value**	**Name**	**Value**
Batch Size:	500	Elite samples (%):	0.05
Number of steps:	8 to 53 (step 5)	Pos weight:	1
Number of iterations:	1	Rot weight:	0.5
Simulation timestep:	0.05	Collision weight:	10

Table 3. Number of iterations influence.

Name	**Value**	**Name**	**Value**
Batch Size:	500	Elite samples (%):	0.05
Number of steps:	8	Pos weight:	1
Number of iterations:	1 to 46 (step 5)	Rot weight:	0.5
Simulation timestep:	0.05	Collision weight:	10

For each value, MB-planner optimization was called 10 times. The plots in Figure 11 were constructed with the mean and standard deviation of the 10 recorded computation times per test value. As expected, the parameters used for sequential operations (number of steps and number of iterations) have a stronger effect on the computation time of

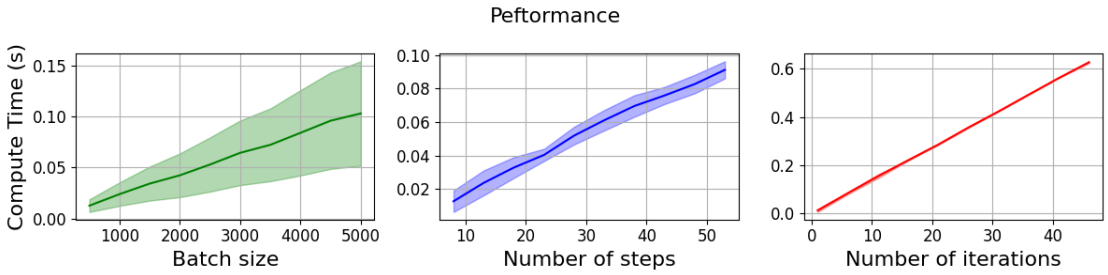


Figure 11. Performance testing results. The graphs show how the computation time depends on the change in batch size, trajectory length, and number of iterations.

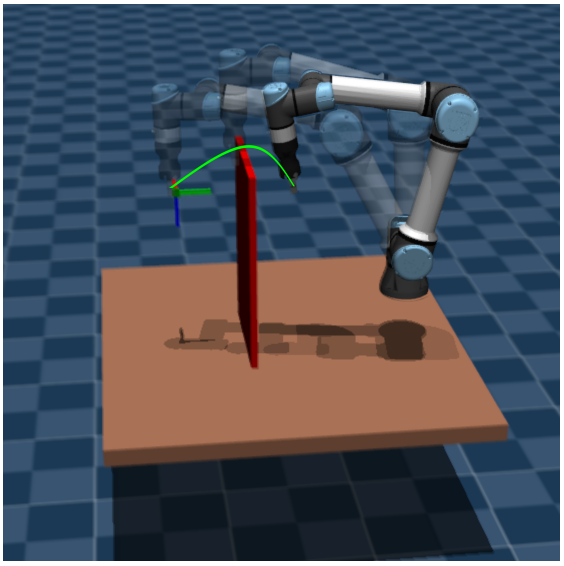
the MB-planner. The batch size, on the other hand, has a comparable influence on computation time, even though the increment step is 100 times higher.

5.3 Offline Planning

The MB-planner can be used separately from MPC for offline planning. In this case, the trajectory length is set to be longer since MB-planner will optimize the whole trajectory and not just the next step. The longer trajectory also means that the number of MB-planner iterations needs to be higher to achieve the successful convergence of the whole path. The larger batch size is also needed for better space exploration and a higher chance of having a low-cost trajectory.

For the demonstration of offline planning, an environment with one wall-type obstacle and a goal was constructed (Figure 12). The length of the trajectory was set to 50, the number of MB-planner iterations was set to 30, and the batch size was set to 2000. With each MB-planner iteration, the cost decreases until it converges. The total cost function is composed of goal, orientation, and collision costs. The goal cost is calculated by finding the distance from the end-effector position to the target position at each step of the simulation and summing it up (Equation 5). The orientation cost is found with rotation alignment [3] at each step of the simulation (Equation 6). In the case of offline planning, the additional weights could be added to decrease the influence of the long distance and incorrect alignment at the earlier steps (Equation 4). The collision cost is calculated in two parts (Equation 7). From MuJoCo, it is possible to retrieve information about the distance between geoms if they are very close to each other. In case of collision, the distance value will be negative. The first part represents the change in distance between every part of the robot and the geom that it can collide with. The second part is the number of existing collisions.

The MB-planner has successfully found the optimal and collision-free trajectory to reach the target with the correct orientation. The first cost plot (Figure 12: Total cost over CEM iterations) demonstrates how the total cost decreases with each iteration of



Name	Value
Batch Size:	2000
Number of steps:	50
Number of iterations:	30
Simulation timestep:	0.05
Elite samples (%):	0.05
Pos weight:	5
Rot weight:	1.5
Collision weight:	10

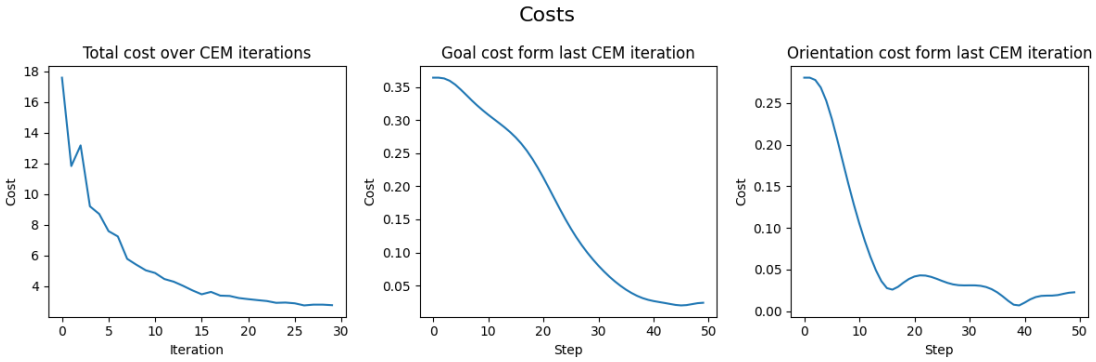


Figure 12. The results of the offline planning demo. The image shows the trajectory that the robot takes to overcome the obstacle (red) and reach the target (green). On the right of the image is the list of the planner parameters used for the demo. On the bottom three cost plots. The left plot shows the total cost over 30 MB-planner iterations. The middle and right plots show how the goal and orientation costs change with each step of the final optimized trajectory.

the MB-planner. This results from the mean and covariance update, which teilores them to have a higher chance of producing a rear low-cost trajectory. The other plots show how the cost of the optimized trajectory decreases as the robot gets closer to the target position and orientation (Figure 12: Goal cost from last CEM iteration, Orientation cost from last CEM iteration).

Table 4. Mathematical definitions for Equations 4 - 8.

Variable	Description
$N \in \mathbb{N}$	number of steps per trajectory
$y \in [0, 1]$	constant, the bigger y the smaller effect contact distance has on the total cost
$\overset{(i)}{pos}_{eef} = [x, y, z]$	end effector position at step i
$pos_t = [x, y, z]$	target position
$\hat{rot}^{(i)}_{eef} = [w, x, y, z]$	normalized rotation quaternion of end effector at step i
$\hat{rot}_t = [w, x, y, z]$	normalized rotation quaternion of target
$cost_g \in \mathbb{Q}_{>0}$	euclidean distance between end effector and target
$cost_o \in \mathbb{Q}_{>0}$	angular distance between end effector and target
$cost_{col} \in \mathbb{Q}_{>0}$	collision gradient + already existing collisions
$cost \in \mathbb{Q}_{>0}$	total cost

$$w = \left\{ \frac{i}{N-1} \;\middle|\; i = 0, 1, \ldots, N-1 \right\} \tag{4}$$

$$cost_g = \sum_{i=0}^{N} \left\| \overset{(i)}{pos}_{eef} - pos_t \right\| * w_i \tag{5}$$

$$cost_o = \sum_{i=0}^{N} 2 \arccos\left(\left| \hat{rot}^{(i)}_{eef} \cdot \hat{rot}_t \right| \right) * w_i \tag{6}$$

$$cost_{col} = \sum_{i=0}^{N} \max(-cont_{i+1} + cont_i - y * cont_i, 0) + \sum_{i=0}^{N} cont_i < 0 \tag{7}$$

$$cost = cost_g w_g + cost_o w_o + cost_{col} w_{col} \tag{8}$$

5.4 Online Planning

For the online planning, the MB-planner was used to find an optimal next step that the robot should take. When using MB-planner in an in-loop fashion, the mean of the multivariate normal distribution is initialized in the beginning and keeps updating throughout the whole execution. This makes it possible to use a minimal number of MB-planner iterations and thus decrease optimization time.

Multiple test scenarios (3) were conducted to show the wide functionality achieved with the developed planner. The Collision Avoidance demo demonstrates how the robot can navigate towards the desired position and orientation even if the target changes its location (Section 5.4.1). The Contact Task demo is intended to show how it is possible to

get the desired robot behavior by manipulating the cost function (Section 5.4.2). Finally, the Real-Life demo shows the use of the proposed MB-planner for online planning with the real UR5e. It demonstrates real-time planning and collision avoidance in a real dynamic environment (Section 5.4.3).

Table 5. Collision Avoidance demo parameters.

Name	**Value**	**Name**	**Value**
Batch Size:	1000	Elite samples (%):	0.05
Number of steps:	8	Pos weight:	5
Number of iterations:	1	Rot weight:	1.5
Simulation timestep:	0.05	Collision weight:	10

Table 6. Contact Task demo parameters.

Name	**Value**	**Name**	**Value**
Batch Size:	500	eef_to_sphere weight:	5
Number of steps:	8	sphere_to_target weight:	7
Number of iterations:	1	Eef rot weight:	2
Simulation timestep:	0.05	Collision weight:	10
Elite samples (%):	0.05	Align weight:	0.2

Table 7. Real-life demo parameters.

Name	**Value**	**Name**	**Value**
Batch Size:	500	Elite samples (%):	0.05
Number of steps:	8	Pos weight:	5
Number of iterations:	1	Rot weight:	1.5
Simulation timestep:	0.1	Collision weight:	10

5.4.1 Collision Avoidance

In the first scenario (Figure 14), the robot needs to navigate towards the goal while avoiding obstacles (two red spheres). The cost definition stayed the same as in the offline demo (Equations 4-8). After the cost is low enough, the position and orientation of the goal change, and the robot needs to navigate towards the new target. The planner parameters used for this demo are listed in Table 5.

The result showed that the robot can reach the target in a reasonable amount of time. The first two targets were reached within 5 and 7 seconds, respectively. The other three

took 16, 23, and 17 seconds, since the robot had to navigate between two obstacles to reach those targets, it was moving more slowly. The cost plots can be seen in Figure 13.

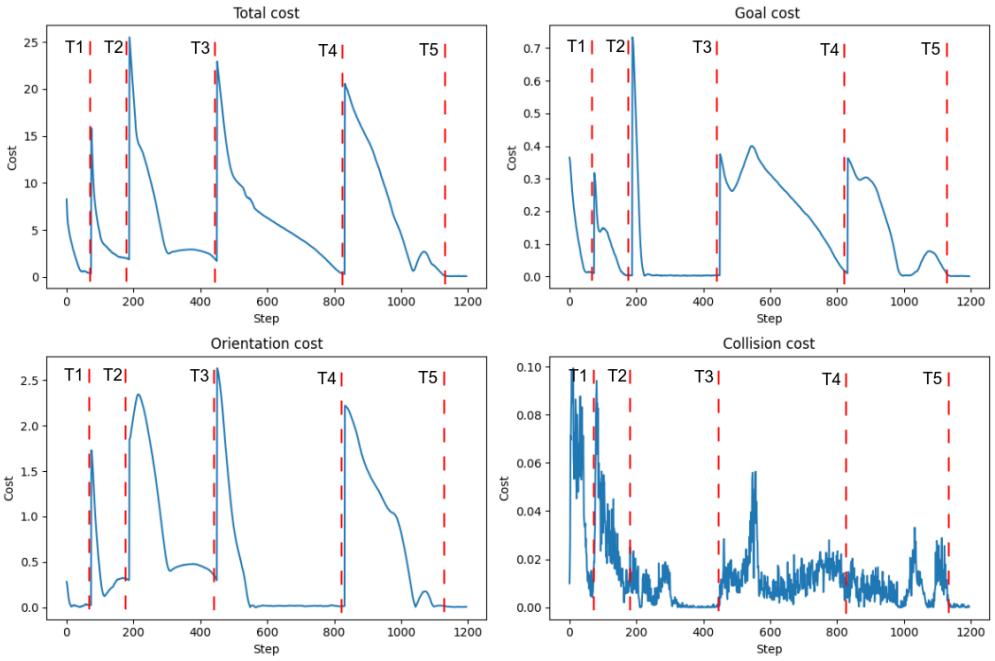


Figure 13. Cost plots of collision demo trajectory. The robot navigates through obstacles to the target. When the cost falls below a set threshold, the target is considered reached and its location changes. The red lines show when each target (Ti) was reached.

5.4.2 Contact Task

In the second scenario, the objective is to push the sphere towards the goal position (white square) (Figure 15). The planner parameters used for this demo are listed in Table 6.

In the previous demo, the target position was simply passed as coordinates without the need to update the location of a rigid body in the simulation. In this demo, however, the robot needs to engage with the physical object to satisfy the formulated task. This means that the sphere location needs to be updated after each real-life step, the same way as the robot's joint states.

The cost function was also modified to fulfill the objective. Now that the robot has to push an object, the contact between the robot and the object has to be excluded from the collision cost calculation. The *mjx.Data* structure contains all of the contact information in a fixed-size matrix. A binary mask was used to only extract contacts essential for the cost calculation. The mask includes all robot geom contacts except the contact with the sphere geom. The additional information of the sphere location had to be extracted from

mjx.Data after each step of simulation inside of MB-planner optimization loop. This information was then used in the new cost definition.

The goal cost for the end effector has an additional approach offset so that the gripper does not go directly into the object (Equation 10). The approach offset is calculated by taking the distance between the sphere and the target location and shifting the goal position for the end effector in the opposite direction. The z axis in the approach offset is constant. The second goal cost is added, which is intended to minimize the distance between the sphere and the target (Equation 11). The z axis is not included in the calculation of sphere-target goal cost. The last modification is the alignment cost, which decreases upon the end effector reaching the correct side relative to the object (Equation 9).

Table 8. Mathematical definitions for Equations 9-11

Variable	**Description**
$N \in \mathbb{N}$	number of steps per trajectory
ϵ	small constant to avoid division by 0
$\Delta d_{s\|t} = [x, y, 0]$	euclidean distance between sphere and target
$\Delta d_{eef\|s} = [x, y, z]$	euclidean distance between end effector and sphere
$cost_a \in \mathbb{Q}_{>0}$	alignment cost
$cost_g^{s\|eef} \in \mathbb{Q}_{>0}$	euclidean distance between end effector and sphere with the small offset
$cost_g^{s\|t} \in \mathbb{Q}_{>0}$	euclidean distance between sphere and target

$$cost_a = \sum_{i=0}^{N} \left| \frac{\Delta d^i_{eef|s} \cdot \Delta d^i_{s|t}}{\|\Delta d^i_{eef|s}\| \, \|\Delta d^i_{s|t}\|} - 1 \right| \tag{9}$$

$$cost_g^{s|eef} = \sum_{i=0}^{N} \left\| \Delta d^i_{eef|s} - 0.05 \frac{\Delta d^i_{s|t}}{|\Delta d^i_{s|t}| + \epsilon} \right\| \tag{10}$$

$$cost_g^{s|t} = \sum_{i=0}^{N} \left\| \Delta d^i_{s|t} \right\| \tag{11}$$

The result showed that the robot was able to navigate from its initial position closer to the sphere. Moreover, the robot approached the sphere from the correct side (opposite to the target sphere location). Throughout the whole execution, the end effector stayed perpendicular to the table and was used to push the sphere towards the target location. In addition, the robot still avoids a collision with the table.

5.4.3 Real-life Demo

In this demo, the real UR5e was controlled by the MB-planner. OptiTrack was used to track target and obstacle locations. The setup is the same as shown in Figure 9. The locations of objects are transferred wirelessly from the PC with motion capture software to the PC controlling the robot. The MB-planner receives motion capture information from the ROS2 topic published by *vrpn-mocap* package. The locations of the objects in the simulation are updated in real time. The target is defined as a box geom that can not collide with the robot, while the obstacle is defined as a mocap body with sphere geometry. In this demo, the objective is the same as in the first two experiments (Sections 5.3, 5.4.1) to reach the goal while avoiding obstacles. The planner parameters used for this demo are listed in Table 7.

The results show that the robot can navigate towards the target location, avoiding the obstacle. If the obstacle comes into collision with the manipulator robot, the robot backs off and tries to approach the target once the collision is cleared. The reaction time is less than a second from the point when the collision is introduced.

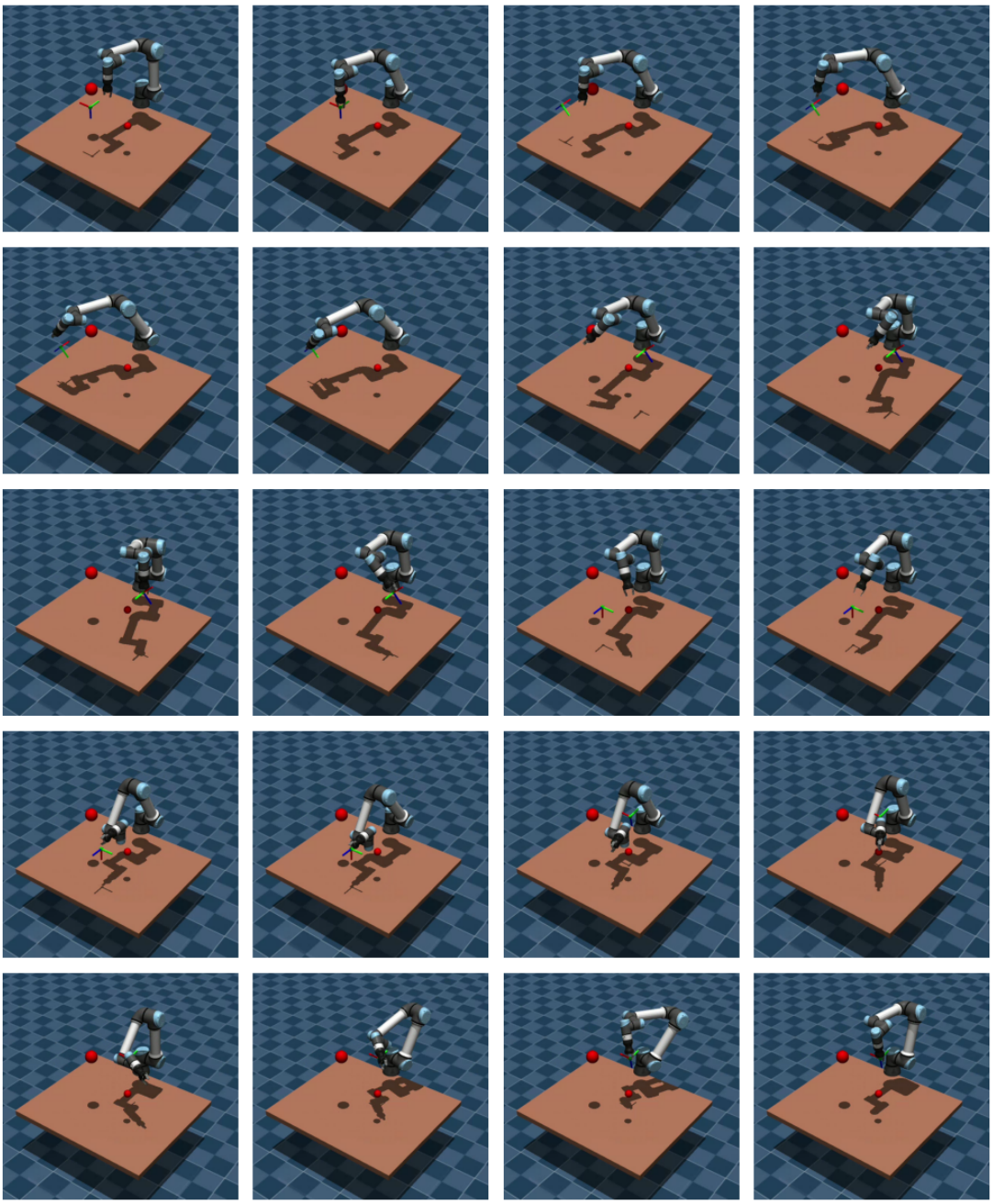


Figure 14. Collision avoidance demo. The objective is to move the robot's end effector to the target position and orientation while avoiding obstacles (red spheres). When the target is reached, the position and orientation of the target are changed.

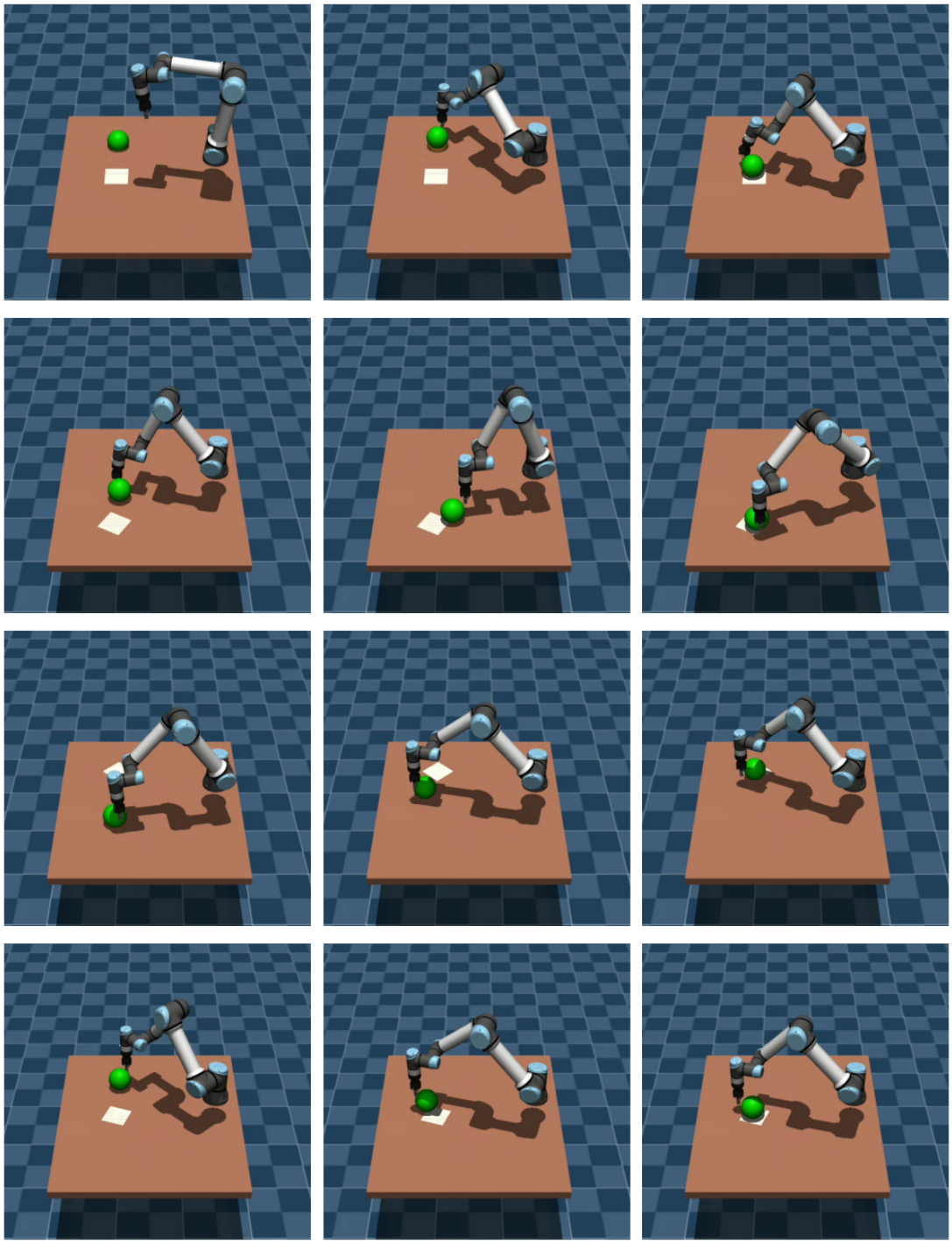


Figure 15. Contact demo. The objective is to move the sphere to the target spot (white square) by pushing. When the target is reached, the position and orientation of the target are changed.

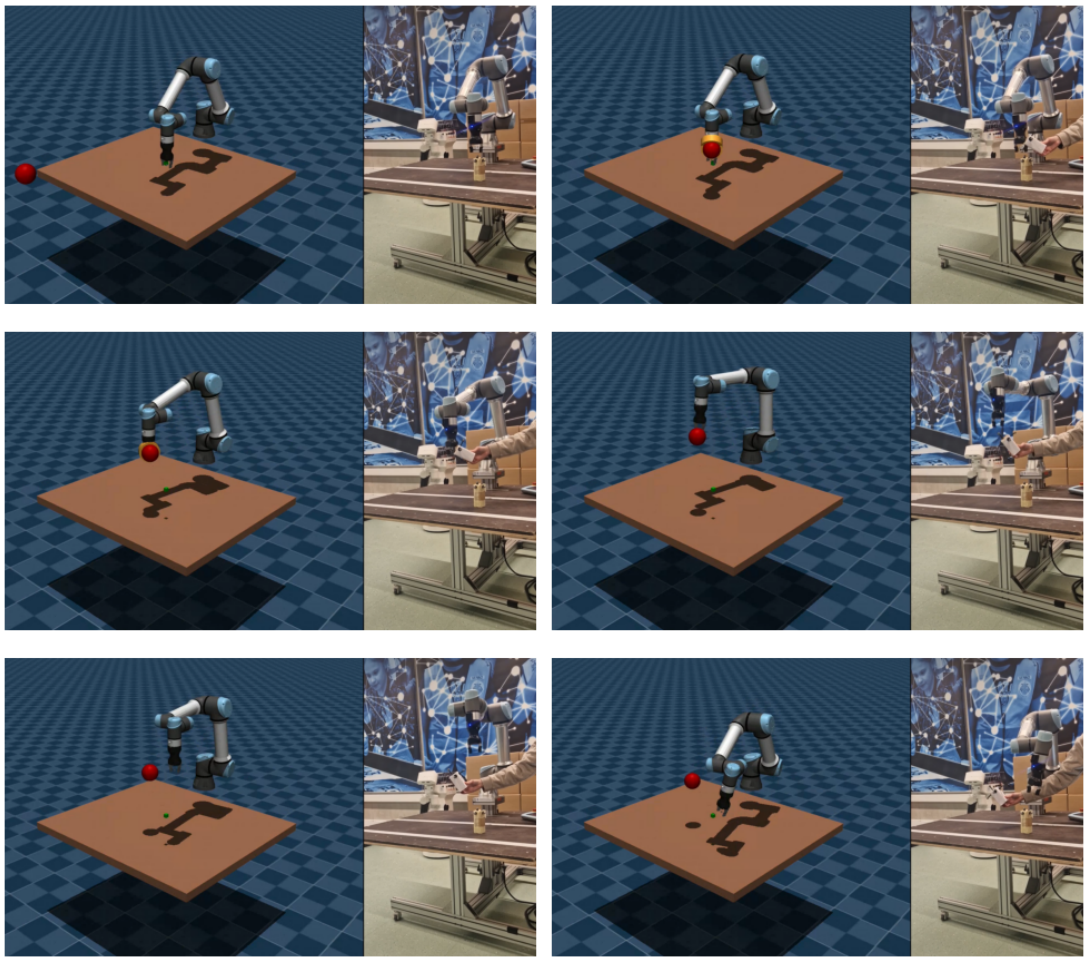


Figure 16. Real-life demo. The figure shows the snapshots of the demo (left to right, top to bottom). The robot reaches the target (green) and backs off when a collision with an obstacle (red) is introduced.

6 Discussion

6.1 Interpretation of Results

The experiments demonstrate the flexibility and efficiency of the proposed model-based planner in both offline and online planning scenarios. It is evident that physics simulation with MuJoCo provides a realistic prediction of the environmental response to given control inputs. From performance benchmarks, it is clear that the computation time scales predictably with increasing trajectory length, number of optimization iterations, and batch size. Sequential operations that inherently require more processing time are shown to have a greater impact on computation time, whereas parallel operation is more scalable, although to some extent. These findings are crucial for tuning the planner under real-time constraints.

In offline planning, the model-based planner successfully finds smooth and collision-free trajectories around obstacles when given sufficient trajectory length, iterations, and batch size. The cost gradually decreases with each optimization iteration, and the proximity cost within one trajectory shows a smooth decline throughout each timestep of the path. This shows the ability of the model-based planner to perform full-trajectory optimization without any handcrafted waypoints or path shaping.

In online planning scenarios, the planner reliably adapts to changing goals and environmental conditions in real time. The collision avoidance demo confirms that a minimal number of optimization iterations is sufficient to maintain safe and goal-directed motion. This is possible with the warm-started parameters of a sampling distribution that keep updating throughout the whole execution process. The contact-based task further demonstrates the modularity of the cost function: by selectively ignoring object contact in collision cost and introducing new task-specific costs (e.g., alignment and multi-objective proximity terms), the planner is repurposed without any structural change.

The online planning and reactive behavior with collision avoidance were also demonstrated on the real UR5e. The experiment showed that the simulation model can be updated in real time and result in a quick reaction time (less than a second) to obstacle interference.

6.2 Limitations

A large timestep can cause instabilities in the MuJoCo simulation. The timestep of the simulation is the most important parameter affecting the speed-accuracy trade-off. Smaller values result in better accuracy and stability. However, for the optimization-related applications, it is desirable to run the simulation as fast as possible. In that case, the time step should be made as large as possible. For the in-loop trajectory planning, the timestep has to be the same as the timestep of the real environment. The timestep of 0.05 s has proved to be an optimal choice for the model-based planner. However, it is

still considered a relatively large value (the default MuJoCo timestep is 0.002 seconds), leading to artifacts being observed during the simulation. This happens because MuJoCo uses a semi-implicit integration scheme and resolves contact forces at each timestep. A coarse timestep (e.g., 0.05 seconds) reduces the resolution of physical interactions. This can result in missed collisions (tunneling), inaccurate contact force estimation, and exaggerated or unstable object behavior—especially with sharp-edged geometries like boxes. Frictional sliding becomes harder to resolve accurately, often leading to jitter and object penetration. For stable and realistic simulation, especially in contact-rich tasks, a smaller timestep is typically required to ensure accurate force computation and robust convergence of the physics solver.

Some challenges were encountered with the contact detection between the robot and box-shaped objects. The contact distance between the primitive robot model (composed of capsules) and box geometries was less accurate than with spheres or capsules. This imprecision became a significant limitation when planning motions near wall-like obstacles. Moreover, in the contact-based task, the box was tested as an object that the robot had to push to the target location. However, issues with unstable contact forces between the box and the supporting surface (also modeled as a box) were observed. These instabilities are likely due in part to the relatively large simulation timestep (0.05 s), which may have contributed to numerical inaccuracies in box-vs-box contact resolution.

6.3 Future Work

Several directions can extend the current model-based planning framework. First, benchmarking the proposed planner against state-of-the-art alternatives through quantitative comparisons would provide a clearer understanding of its performance and limitations. Expanding the framework to support multi-robot planning is another important direction, particularly for tasks involving collaboration or shared workspaces. To make the planner more modular and standardized, it can be wrapped into a Gym environment, enabling easier evaluation, integration with other frameworks, and compatibility with reinforcement learning approaches. This also leads to another improvement: hybrid approaches that combine model-based planning with learning-based components. For instance, reinforcement learning can be used to learn task-specific cost function parameters, generate warm-starts to accelerate sampling-based optimization, or guide the trajectory sampling toward more promising regions of the search space. Such integration can enhance planning efficiency, adaptability to new tasks, and robustness in complex environments.

7 Conclusion

This thesis proposed a model-based trajectory optimization algorithm for manipulator robots, leveraging GPU-parallelization. The use of a MuJoCo-simulated world model provided both time-efficient and accurate prediction of system responses to control inputs sampled from a multivariate normal distribution. It was shown that the proposed planner can generate a smooth, collision-free, and goal-directed motion.

Multiple experiments with different test case scenarios have been conducted to evaluate the performance of the planner. These scenarios involved different task objectives for the robot, including reaching a specified position and orientation, and pushing an object to a designated goal location. By defining the cost function in accordance with each objective, the planner was able to adapt its behavior to task-specific requirements. The results showed that the planner could generalize across task types while maintaining stable and efficient execution.

The ability of the planner to adapt to a dynamic environment was also validated on the real UR5e robot. A motion capture system was used to track the positions of the target and obstacles in real time, and this information was continuously fed into the simulation to keep the world model up-to-date. The results demonstrated that the system could maintain goal-directed behavior and avoid collisions even under non-static conditions, highlighting the robustness and reactivity of the proposed method in practical, real-world settings.

References

[1] Mohak Bhardwaj et al. "STORM: An Integrated Framework for Fast Joint-Space Model-Predictive Control for Reactive Manipulation". In: *Proceedings of the 5th Conference on Robot Learning*. Conference on Robot Learning. ISSN: 2640-3498. PMLR, Jan. 11, 2022, pp. 750–759. URL: https://proceedings.mlr.press/v164/bhardwaj22a.html (visited on 04/27/2025).

[2] Chaithanya. *10 Robotics Challenges Every Innovator Should Know*. Industry Wired. Sept. 26, 2024. URL: https://industrywired.com/10-robotics-challenges-every-innovator-should-know/ (visited on 04/06/2025).

[3] *dist - Angular distance in radians - MATLAB*. URL: https://www.mathworks.com/help/driving/ref/quaternion.dist.html (visited on 05/04/2025).

[4] Íñigo Elguea-Aguinaco et al. "A Review on Reinforcement Learning for Motion Planning of Robotic Manipulators". In: *International Journal of Intelligent Systems* 2024.1 (2024). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/int/1636497, p. 1636497. ISSN: 1098-111X. DOI: 10.1155/int/1636497. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/int/1636497 (visited on 04/27/2025).

[5] Nigora Gafur et al. "Dynamic Collision and Deadlock Avoidance for Multiple Robotic Manipulators". In: *IEEE Access* 10 (2022), pp. 55766–55781. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3176626. URL: https://ieeexplore.ieee.org/abstract/document/9779146 (visited on 04/27/2025).

[6] *How Do Robot Manipulators Move? – Robotic Sea Bass*. June 30, 2024. URL: https://roboticseabass.com/2024/06/30/how-do-robot-manipulators-move/ (visited on 05/12/2025).

[7] *Introduction to Reinforcement Learning – A Robotics Perspective » Lamarr-Blog*. July 5, 2023. URL: https://lamarr-institute.org/blog/reinforcement-learning-and-robotics/ (visited on 04/06/2025).

[8] Ali Jamshidi and Esmaeel Khanmirza. "Improved Probabilistic Roadmap Path Planning Algorithm". In: *2024 12th RSI International Conference on Robotics and Mechatronics (ICRoM)*. 2024 12th RSI International Conference on Robotics and Mechatronics (ICRoM). ISSN: 2572-6889. Dec. 2024, pp. 362–367. DOI: 10.1109/ICRoM64545.2024.10903585. URL: https://ieeexplore.ieee.org/document/10903585/ (visited on 04/21/2025).

[9] Hongcheng Ji et al. "E-RRT*: Path Planning for Hyper-Redundant Manipulators". In: *IEEE Robotics and Automation Letters* 8.12 (Dec. 2023), pp. 8128–8135. ISSN: 2377-3766. DOI: 10.1109/LRA.2023.3325716. URL: https://ieeexplore.ieee.org/document/10287397/ (visited on 04/21/2025).

[10] Marin Kobilarov. “Cross-Entropy Randomized Motion Planning”. In: ().

[11] Jessica Leu et al. “Efficient Robot Motion Planning via Sampling and Optimization”. In: *2021 American Control Conference (ACC)*. 2021 American Control Conference (ACC). ISSN: 2378-5861. May 2021, pp. 4196–4202. DOI: 10.23919/ACC50511.2021.9483146. URL: https://ieeexplore.ieee.org/abstract/document/9483146 (visited on 04/27/2025).

[12] Jie Liu, Hwa Jen Yap, and Anis Salwa Mohd Khairuddin. “Review on Motion Planning of Robotic Manipulator in Dynamic Environments”. In: *Journal of Sensors* 2024.1 (2024). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2024/5969512, p. 5969512. ISSN: 1687-7268. DOI: 10.1155/2024/5969512. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2024/5969512 (visited on 04/21/2025).

[13] Youyu Liu et al. “Research on the Dynamic Path Planning of Manipulators Based on a Grid-Local Probability Road Map Method”. In: *IEEE Access* 9 (2021), pp. 101186–101196. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3098044. URL: https://ieeexplore.ieee.org/abstract/document/9490228 (visited on 04/21/2025).

[14] Viktor Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. Aug. 25, 2021. DOI: 10.48550/arXiv.2108.10470. arXiv: 2108.10470[cs]. URL: http://arxiv.org/abs/2108.10470 (visited on 04/06/2025).

[15] *MuJoCo — Advanced Physics Simulation*. URL: https://mujoco.org/ (visited on 04/06/2025).

[16] *MuJoCo XLA (MJX) - MuJoCo Documentation*. URL: https://mujoco.readthedocs.io/en/stable/mjx.html (visited on 04/06/2025).

[17] *NVIDIA Isaac Platform*. NVIDIA Developer. URL: https://developer.nvidia.com/isaac (visited on 04/06/2025).

[18] Luka Petrović et al. “Cross-entropy based stochastic optimization of robot trajectories using heteroscedastic continuous-time Gaussian processes”. In: *Robotics and Autonomous Systems* 133 (Nov. 2020), p. 103618. ISSN: 09218890. DOI: 10.1016/j.robot.2020.103618. URL: https://linkinghub.elsevier.com/retrieve/pii/S0921889020304589 (visited on 04/21/2025).

[19] Corrado Pezzato et al. *Sampling-based Model Predictive Control Leveraging Parallelizable Physics Simulations*. Jan. 21, 2025. DOI: 10.48550/arXiv.2307.09105. arXiv: 2307.09105[cs]. URL: http://arxiv.org/abs/2307.09105 (visited on 05/19/2025).

[20] Cristina Pinneri et al. “Sample-efficient Cross-Entropy Method for Real-time Planning”. In: *Proceedings of the 2020 Conference on Robot Learning*. Conference on Robot Learning. ISSN: 2640-3498. PMLR, Oct. 4, 2021, pp. 1049–1065. URL: https://proceedings.mlr.press/v155/pinneri21a.html (visited on 04/27/2025).

[21] Brian Plancher et al. “Accelerating Robot Dynamics Gradients on a CPU, GPU, and FPGA”. In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 2335–2342. ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3057845. URL: https://ieeexplore.ieee.org/document/9350173/ (visited on 04/06/2025).

[22] *Robotic Control Systems and Real-World Challenges*. Oct. 3, 2023. URL: https://blogs.dal.ca/openthink/robotic-control-systems-and-real-world-challenges/ (visited on 04/06/2025).

[23] *ROS Index*. URL: https://index.ros.org/r/vrpn_mocap/#rolling (visited on 05/02/2025).

[24] Elena Rubleva, Konstantin Mironov, and Aleksandr Panov. “Stabilizing Manipulator Trajectory via Collision-Aware Optimization”. In: *Interactive Collaborative Robotics*. Ed. by Andrey Ronzhin, Jesus Savage, and Roman Meshcheryakov. Cham: Springer Nature Switzerland, 2024, pp. 30–44. ISBN: 978-3-031-71360-6. DOI: 10.1007/978-3-031-71360-6_3.

[25] Tamar Segal. *From Industry 1.0 to Industry 5.0*. CoreTigo. July 2, 2024. URL: https://www.coretigo.com/industrial-revolution-from-industry-1-0-to-industry-5-0/ (visited on 04/06/2025).

[26] *SintefManufacturing/python-urx*. original-date: 2013-03-23T10:33:39Z. Apr. 30, 2025. URL: https://github.com/SintefManufacturing/python-urx (visited on 05/02/2025).

[27] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A physics engine for model-based control”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. ISSN: 2153-0866. Oct. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109. URL: https://ieeexplore.ieee.org/document/6386109/ (visited on 05/19/2025).

[28] *Trajectory optimization*. In: *Wikipedia*. Page Version ID: 1274625158. Feb. 8, 2025. URL: https://en.wikipedia.org/w/index.php?title=Trajectory_optimization&oldid=1274625158 (visited on 04/06/2025).

[29] Lei Wang et al. “Supervised Meta-Reinforcement Learning With Trajectory Optimization for Manipulation Tasks”. In: *IEEE Transactions on Cognitive and Developmental Systems* 16.2 (Apr. 2024), pp. 681–691. ISSN: 2379-8939. DOI: 10.1109/TCDS.2023.3286465. URL: https://ieeexplore.ieee.org/abstract/document/10153775 (visited on 04/27/2025).

[30] *XML Reference - MuJoCo Documentation*. URL: https://mujoco.readthedocs.io/en/stable/XMLreference.html (visited on 04/09/2025).

Appendix

The video demonstrations and source code can be found on GitHub:
`https://github.com/patsyuk03/mjx_planner.git`

I. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Iryna Hurova**,

1. grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the digital archives of the University of Tartu until the expiry of the term of copyright, my thesis

   Model-based planning using GPU-accelerated Simulator as a World Model,

   supervised by Karl Kruusamäe and Arun Kumar Singh.

2. rant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright;

3. am aware of the fact that the author retains the rights specified in points 1 and 2;

4. confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Iryna Hurova
20/05/2025