SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER, CONTROL AND MANAGEMENT ENGINEERING

Kinodynamic motion planning for steerable WMRs

AUTONOMOUS AND MOBILE ROBOTICS

Professors:
Giuseppe Oriolo

Teaching Assistant:
Michele Cipriano

Students:
Dario Basile
Lorenzo Ciarpaglini
Luca Del Signore

Academic Year 2023/2024

Contents

1 Introduction

In recent years, the field of autonomous mobile robotics has witnessed substantial advancements, particularly in the areas of kinematic modeling and motion planning algorithms. These developments have paved the way for robots to navigate and operate more effectively in increasingly complex environments. This report focuses on the development and implementation of the "Kinodynamic RRT*" (KRRT*) planning algorithm for the Neobotix MPO-700, a mobile robot equipped with 4 off-centered steering wheels.

The Neobotix MPO-700 is designed to handle a variety obstacles, making it an ideal subject for exploring advanced kinodynamic constraints in robotic path planning. Classic differential drives and car-like robots are characterized by low production costs and simple designs but suffer from limited maneuverability due to the well-known inability to move laterally without preliminary maneuvers (nonholonomic constraints).

The Neobotix MPO-700 addresses this issue as an omnidirectional platform, capable of translating and rotating independently, albeit at a higher production cost.

The KRRT* algorithm, known for its efficiency in handling dynamic constraints while ensuring rapid convergence, has been adapted to meet the specific navigational needs of the Neobotix MPO-700. Unlike traditional path planning techniques, KRRT* integrates the robot's kinematic constraints directly into the path search algorithm, facilitating more realistic and practical navigation strategies.

In this report, we detail the kinematic model of the Neobotix MPO-700 and describe how the KRRT* algorithm was tailored and implemented to exploit these characteristics fully. Our focus is solely on the planning phase, where the primary objective is to generate feasible and efficient paths through diverse environmental setups.

Subsequent sections will present our methodology for developing the algorithm, followed by a series of simulations conducted on different maps to validate our approach. These simulations demonstrate the algorithm's effectiveness and limitations.

The remainder of this report is organized as follows:

- **Model:** describes formally the kinematic model of a off-centered steering robot.
- **Planner** provides a complete description of the KRRT* algorithm, and how we exploit it for our purpose.
- **Simulations** discuss our experiment set-up and the respective results.
- **Conclusions** concludes the report.

2 Model

The development of the kinematic model is based on a set of fundamental assumptions aimed at simplifying the representation of the system. These assumptions allow for a purely kinematic analysis, ignoring the complexities introduced by external forces. By constraining the model to a purely kinematic level, the focus lies primarily on the geometric relationships governing motion, facilitating a clearer understanding of the system's behavior. In particular:

- **Localization Precision in Simulation**: Given that the analysis is conducted within a simulated environment rather than on a physical robot, the challenges associated with robot localization through sensor measurements can be disregarded. In simulation, the precise state and location of the robot is known with certainty, eliminating uncertainties related to sensor/actuators inaccuracies or environmental conditions. This ideal mapping scenario provides a controlled environment for studying the system's kinematics, allowing for precise and deterministic modeling of the robot's motion.

- **Absence of Friction or External Forces**: The model operates in an idealized environment where frictional effects and external forces are disregarded. This simplification enables a more straightforward analysis of motion without the complicating factors introduced by frictional interactions or external disturbances.

- **Pure Rolling Constraints**: Each wheel adheres to the perfect rolling constraint, meaning there is no longitudinal or lateral slipping during motion. This assumption ensures that the model accurately represents the kinematic behavior of the system without the complexities introduced by wheel slippage.

- **Rigid Body Assumption**: The platform chassis is modeled as a perfectly rigid body moving on a horizontal plane. This simplifying assumption facilitates the representation of the system's motion without the need to account for deformations or flexibility within the chassis.

- **A Completely Known Environment**: The robot knows a priori where the obstacle are. So we don't need sensors to intercept them.

2.1 MPO-700 Kinematics

The Neobotix MPO-700 is a steerable wheeled mobile robot (SWMR) equipped with four independently actuated off-centered steering wheels. These wheels can roll on the ground and rotate around an off-center vertical axis (hereinafter referred to as the joint) which does not pass through wheels' center, hence the designation "off-centered."

This configuration ensures, through appropriate coordination, the ability to achieve arbitrary linear/angular velocities, providing omnidirectionality.

For the kinematic model, we follow the analysis conducted in [1] for robots with n_s independently actuated centered steering wheels. The nonholonomic constraints are equivalent because the zero motion line of a wheel stays invariant regardless of the displacement value from the joint,and kinematically it is as if the wheel is in the same position as the joint is.

Let $\mathcal{F}_\mathbf{0} : \{\mathbf{O}, \mathbf{X_0}, \mathbf{Y_0}\}$ be a fixed inertial frame and $\mathcal{F}_\mathbf{B} : \{\mathbf{O_B}, \mathbf{X_B}, \mathbf{Y_B}\}$ a moving frame attached to the platform body. We consider a SWMR equipped with $n_s \geqslant 2$ independent steerable wheels W_i, each characterized by angle ϕ_i representing the orientation of the wheel's sagittal axle with respect to $\mathbf{X_B}$ (also referred to as the steering angle). Each wheel's joint S_i is located at position $\mathbf{P_i}$ in frame $\mathcal{F}_\mathbf{B}$.

The vector

$$\boldsymbol{\xi} = \begin{bmatrix} x & y & \theta \end{bmatrix}^T \in SE(2)$$

represents the pose of the mobile base in $\mathcal{F}_0$, with (x, y) denoting the position of $\mathcal{F}_B$ and θ the angle between axes $\mathbf{X_B}$ and $\mathbf{X_0}$. The complete configuration space of the mobile robot is given by

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{\xi}^T & \boldsymbol{\phi}^T \end{bmatrix},$$

where $\boldsymbol{\phi} = \begin{bmatrix} \phi_1 & \phi_2 & \dots & \phi_n \end{bmatrix}$ represents the steering angles of all wheels.

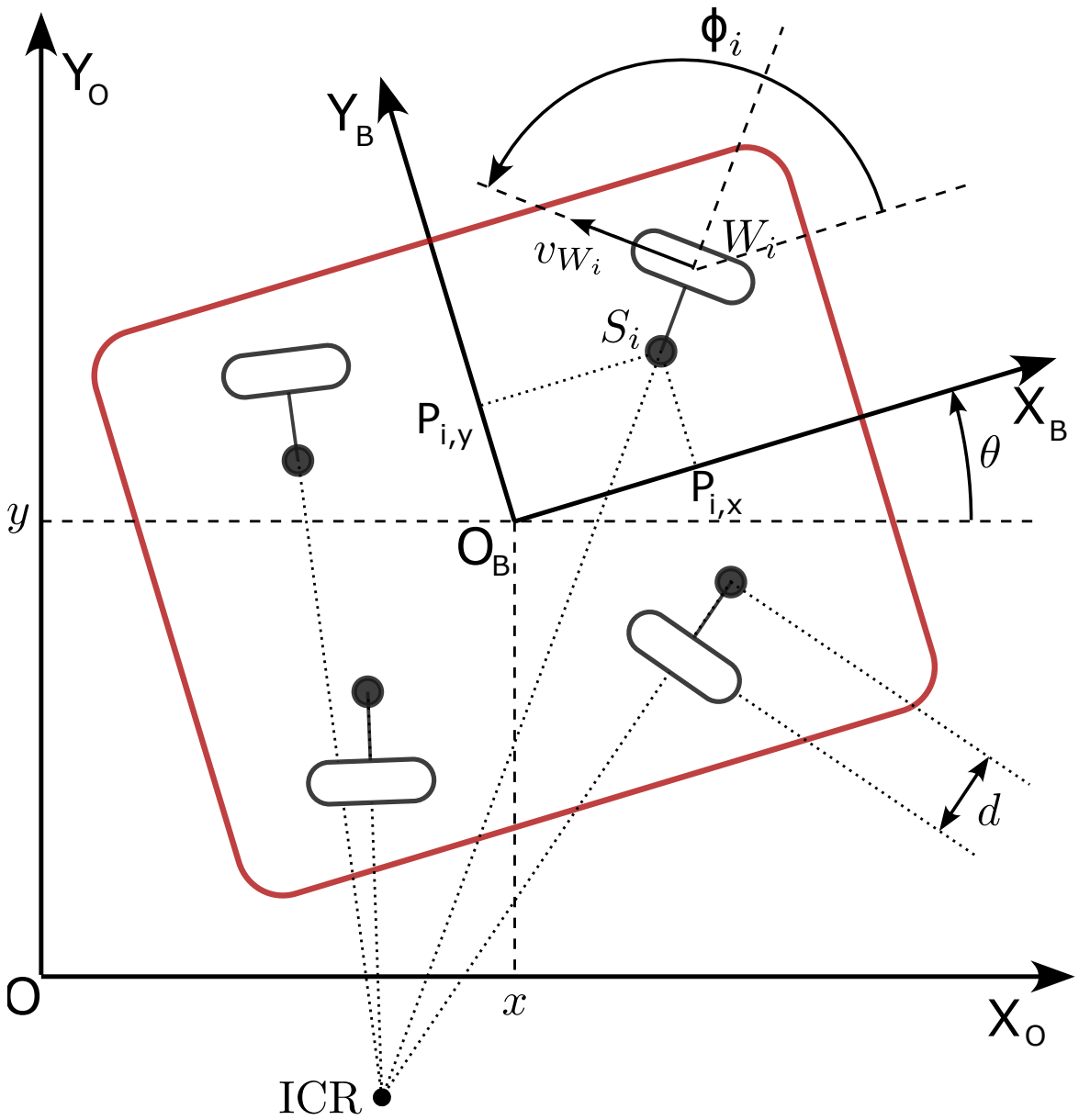


Figure 1: Schematic model of Neobotix MPO-700

The position of the i-th steering joint S_i is defined as

$$\mathbf{o}_{S_i} = \begin{bmatrix} x \\ y \end{bmatrix} + \mathbf{R}(\theta) \begin{bmatrix} P_{i,x} \\ P_{i,y} \end{bmatrix}, \tag{2.1}$$

and the position of the i-th wheel W_i is defined as

$$\mathbf{o}_{W_i} = \mathbf{o}_{S_i} + \mathbf{R}(\theta + \phi_i) \begin{bmatrix} 0 \\ -d \end{bmatrix}, \tag{2.2}$$

where $\mathbf{R} \in SO(2)$ is a planar rotation matrix, and d is the displacement.

Due to the pure rolling constraint (i.e., the velocity of the contact point of the wheel must be orthogonal with respect to the zero motion line of the wheel itself),

each wheel is subject to the Pfaffian constraint:

$$\begin{bmatrix} -\sin(\theta+\phi_i) \\ \cos(\theta+\phi_i) \end{bmatrix}^T \dot{\mathbf{o}}_{W_i} = 0.$$

All n_s constraints can be consolidated into the following matrix form:

$$\begin{bmatrix} -\sin(\theta+\phi_1) & \cos(\theta+\phi_1) & \Delta_1 & 0 & \cdots & 0 \\ -\sin(\theta+\phi_2) & \cos(\theta+\phi_2) & \Delta_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -\sin(\theta+\phi_{n_s}) & \cos(\theta+\phi_{n_s}) & \Delta_{n_s} & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi}_1 \\ \vdots \\ \dot{\phi}_{n_s} \end{bmatrix} = \begin{bmatrix} \mathbf{A_p}(\mathbf{q}) \ \mathbf{0} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\xi}} \\ \dot{\boldsymbol{\phi}} \end{bmatrix} = \mathbf{A}(\mathbf{q})\dot{\mathbf{q}} = 0 \tag{2.3}$$

where $\Delta_i = P_{i,x}\cos(\phi_i) + P_{i,y}\sin(\phi_i)$, $\mathbf{A_p}(\mathbf{q}) \in \mathbb{R}^{n_s \times 3}$, and $\mathbf{A}(\mathbf{q}) \in \mathbb{R}^{n_s \times (3+n_s)}$. The model implies that the instantaneous velocity of the platform must satisfy the Pfaffian constraints.

For Descartes' principle all axles of the wheels must intersect at a single point, the Instantaneous Center of Rotation (ICR), or the platform will not move. When they intersect at infinity, it means that all the wheels are parallel and the platform moves straight in the direction of the saggittal axle of each wheel. Thus not all $\dot{\boldsymbol{\xi}}$ are feasible instantaneously, but we need first to reorganize the wheels through steering angles.

2.2 A General Kinematic Model

The existence of a unique Instantaneous Center of Rotation (ICR) can be viewed as a geometric constraint that requires all wheel axles to converge at a single point through steering coordination. Rather than having $\dot{\boldsymbol{\xi}}$ depend on this intersection, the ICR (and the corresponding ϕ_i for each wheel) is determined by the desired geometric path of $\boldsymbol{\xi}$, namely $\boldsymbol{\xi}(\boldsymbol{t})$.

Considering the i-th Pfaffian constraint and solving it for ϕ_i, we find that it depends on $\boldsymbol{\xi}$ and $\dot{\boldsymbol{\xi}}$:

$$\phi_i = h_i(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}) = \arctan\left(\frac{-\sin\theta\dot{x} + \cos\theta\dot{y} + P_{i,x}\dot{\theta}}{\cos\theta\dot{x} + \sin\theta\dot{y} - P_{i,y}\dot{\theta}}\right) \tag{2.4}$$

As mentioned earlier, we can use this to simplify the model (2.3) by using only one wheel as a kinematic constraint, coordinating the orientation of all other wheels to achieve the desired path using (2.4).

The reduced configuration space of the robot is then $\boldsymbol{q} = \begin{bmatrix} x & y & \theta & \phi_1 \end{bmatrix}^T$. The

reduced implicit kinematic model is:

$$\begin{bmatrix} -\sin(\theta+\phi_1) & \cos(\theta+\phi_1) & \Delta_1 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi}_1 \end{bmatrix} = 0, \tag{2.5}$$

combined with these geometric constraints:

$$\phi_i = h_i(\boldsymbol{\xi}, \dot{\boldsymbol{\xi}}), \quad i = 2, \ldots, n_s.$$

Considering the implicit model (2.5), $\dot{q}$ can be explicitly written as:

$$\dot{q} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi}_1 \end{bmatrix} = \begin{bmatrix} \cos(\theta+\phi_1) \\ \sin(\theta+\phi_1) \\ 0 \\ 0 \end{bmatrix} v_{S_1} + \begin{bmatrix} P_{1,x}\sin(\theta) + P_{1,y}\cos(\theta) \\ -P_{1,x}\cos(\theta) + P_{1,y}\sin(\theta) \\ 1 \\ 0 \end{bmatrix} \omega + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} v_{\phi_1}$$

or alternatively:

$$\begin{aligned} \dot{x} &= v_{S_1}\cos(\theta+\phi_1) + \omega(P_{1,x}\sin\theta + P_{1,y}\cos\theta) \\ \dot{y} &= v_{S_1}\sin(\theta+\phi_1) + \omega(-P_{1,x}\cos\theta + P_{1,y}\sin\theta) \\ \dot{\theta} &= \omega \\ \dot{\phi}_1 &= v_{\phi_1} \end{aligned} \tag{2.6}$$

where v_{S_1}, ω, and v_{ϕ_1} are the inputs of the system. While v_{ϕ_1} is a physical input of the wheel (representing steering velocity of W_1), v_{S_1} is the velocity of the point S_1 in the wheel W_1 frame and ω is the angular velocity we want to impose on the platform. It's important to underline here that the velocity v_{S_1} is the velocity that the wheel would have if the displacement were zero. Instead, in our case, the wheel is in a different location of S_1 due to the displacement, and then the velocity of the wheel v_{W_1} is different from v_{S_1} but they are correlated as we can see in (2.2). We will show later how to retrieve this value.

This model must be used together with the geometric constraints (2.4) to set ϕ_i to a specific value in order to maintain a unique ICR, located on the line of the axle of W_1 at distance v_{S_1}/ω from the joint S_1. When $\omega = 0$, the ICR is at infinity. If the ICR is located on one point belonging to the robot, it means that the robot is rotating around a vertical axis passing through this point and thereby this point has zero velocity.

Substituting the explicit model (2.6) into (2.4), we obtain:

$$\phi_i = h_i(v_{S_1}, \omega, \phi_1) = \arctan\left(\frac{v_{S_1}\sin\phi_1 + \omega(P_{i,x} - P_{1,x})}{v_{S_1}\cos\phi_1 + \omega(P_{1,y} - P_{i,y})}\right) \tag{2.7}$$

which we will now refer to as the coordinating functions. Before proceeding, we need to note that:

1. The relation (2.7) returns two solutions. We need to select the one closer to the previous value of ϕ_i.

2. The coordinating functions present a structural singularity when the ICR is located on one of the joints S_i. In such cases, any orientation ϕ_i satisfies its nonholonomic constraint and (2.7) returns undefined values; physically it means that the robot is pivoting on S_i. Moreover, when ICR is near to joint S_i, the corresponding steering velocity $\dot{\phi}_i$ can reach high values. This dangerous condition is met when the joint velocity is zero or passes through zero while following the path $\boldsymbol{\xi(t)}$.

Therefore, we must avoid this situation. To view the velocity of S_i as a scalar rather than a vector, we consider the frame attached to the W_i wheel and write $\dot{\mathbf{o}}_{S_i}$ as:

$$\begin{bmatrix} v_{S_i} \\ 0 \end{bmatrix} = \mathbf{R}(\theta + \phi_i)^T \dot{\mathbf{o}}_{S_i} \tag{2.8}$$

and then enforcing:

$$v_{S_i} = \begin{bmatrix} \cos(\theta + \phi_i) \\ \sin(\theta + \phi_i) \end{bmatrix}^T \dot{\mathbf{o}}_{S_i} \neq 0 \tag{2.9}$$

which also means never passing through zero during the path (like changing sign). It's important to notice that these functions are undefined also when the platform is at rest (so that v_{S_1} and ω are equal to zero), and it's necessary to take into account this problem before starting the path.

2.3 A Dynamic Extension of the General Kinematic Model

We have extended the kinematic model (2.6) dynamically by introducing the following integrators:

$$\begin{aligned} \dot{v}_{S_1} &= a_{S_1}, \\ \dot{\omega} &= a_\omega \end{aligned} \tag{2.10}$$

The complete model, combined with the coordinating functions (2.7), now becomes:

$$\begin{aligned} \dot{x} &= v_{S1} \cos(\theta + \phi_1) + \omega(P_{1,x} \sin\theta + P_{1,y} \cos\theta), \\ \dot{y} &= v_{S1} \sin(\theta + \phi_1) + \omega(-P_{1,x} \cos\theta + P_{1,y} \sin\theta), \\ \dot{\theta} &= \omega, \\ \dot{\phi}_1 &= v_{\phi_1}, \\ \dot{v}_{S_1} &= a_{S_1}, \\ \dot{\omega} &= a_\omega \end{aligned} \tag{2.11}$$

where $\boldsymbol{u} = \begin{bmatrix} v_{\phi_1}, a_{S_1}, a_\omega \end{bmatrix}^T$ represents the new input vector with a_{S_1} and a_ω being the acceleration of S_1 (in the wheel W_1 frame) and angular acceleration of the platform,

respectively. The robot's dynamic configuration space now extends to $\boldsymbol{q} = \left[x, y, \theta, \phi_1, v_{S_1}, \omega\right]^T$, incorporating the velocity of S_1 and angular velocity of the platform. The main difference with the previous model is that (2.6) directly controls velocities and doesn't have free evolution of the state (resulting in the platform remaining immobile without directly giving inputs).

From now on, we will consider only this model for the following reasons:

1. As we will see in the next section, the steering velocity of each wheel is necessary for calculating the driving velocity of the wheel itself, and the derivative of the coordinating functions $\dot{\phi}_i = \frac{d}{dt}h_i(v_{S_1}, \omega, \phi_1)$ (i.e., the steering velocity of W_i) depend on the derivative of v_{S_1} and ω. By incorporating these values directly in the model as inputs, the computation becomes easier.

2. As we will see in the planner chapter (3), KKRT* requires a linearized controllable model to work, which is the case of the linearization of this dynamic extension.

2.4 Driving Velocity and Steering Velocity of Each Wheel

The Driving velocities of the wheels are significant in terms of physical inputs for the real robot Neobotix MPO-700. We will demonstrate how to explicitly calculate these velocities to use them in future applications. Similar to what we have done in (2.8), the scalar velocity of W_i in the W_i frame can be obtained by:

$$v_{W_i} = v_{S_i} + d(\dot{\theta} + \dot{\phi}_i), \tag{2.12}$$

where the steering velocities $\dot{\phi}_i$ can be obtained as previously mentioned by differentiating (2.7).

3 Planner

3.1 Motion planning

Motion planning refers to the problem of finding a collision-free motion that takes the robot from an initial to a final configuration, in a workspace populated by obstacles. Within the classes of methods for motion planning we can distinguish probabilistic methods, which are sampling-based methods, with the basic concept of randomly sampling from the configuration space new samples and check whether they can be added to a roadmap of the space.

KRRT* [2] is an incremental sampling-based approach for asymptotically optimal motion planning for robots with linear dynamics. It is an extension of the RRT* algorithm [3], which is itself an extension of the RRT algorithm [4].

3.2 RRT

The RRT (Rapidly-exploring Random Tree) algorithm demonstrates effectiveness in addressing motion planning challenges within high-dimensional state spaces in a reasonable time.

It is probabilistically complete, which means that if a solution exist the probability of finding it tends to 1 as the execution time tends to infinity, and it is single query, so a new initial and final configuration pair requires to run the algorithm again.

The concept behind RRT is to utilize random sampling to construct a roadmap of the configuration space as a Tree $\mathcal{T}$ structure where the root is the initial configuration $\mathbf{q}_\text{s}$. The algorithm iteration follow this simple steps:

1. Sample a random configuration $\mathbf{q}_\text{rand}$ from the configuration space using random uniform probability.
2. Search within the Tree $\mathcal{T}$ the closest node $\mathbf{q}_\text{near}$ to $\mathbf{q}_\text{rand}$.
3. Choose $\mathbf{q}_\text{new}$ at a distance δ from $\mathbf{q}_\text{near}$ in the direction of $\mathbf{q}_\text{rand}$.
4. If $\mathbf{q}_\text{new}$ and the segment between $\mathbf{q}_\text{near}$ and $\mathbf{q}_\text{new}$ do not collide with any obstacle, then add $\mathbf{q}_\text{new}$ to the Tree $\mathcal{T}$.

The choice of a metric plays an important role in identifying the nearest node to the random sampled.

The expansion of the Tree is biased toward larger Voronoi regions, so toward larger unexplored areas.

3.3 RRT*

The extension RRT* (Optimal RRT) introduces a cost function C^* for each node in the Tree, representing the cost of traversing the tree from the root to that specific node. An optimal solution aims to minimize this cost along the path from the initial configuration to the final one. RRT* is considered asymptotically optimal, meaning that the likelihood of finding an optimal solution approaches 1 as execution time increases indefinitely.

To achieve this, when a new node joins the tree, all existing nodes are reassessed to determine if the new node provides a more optimal path. In practice, only the nearest nodes to the newly added one undergoes this rewiring process. The selection of $\mathbf{q}_{\text{new}}$ is the same as the base algorithm. After that the iteration follow this steps:

1. Select the closest nodes $\mathbf{Q}_{\text{neighbors}}$ to $\mathbf{q}_{\text{new}}$ in the Tree $\mathcal{T}$, using a fixed radius r.
2. Choose the parent $\mathbf{q}_{\text{parent}}$ of $\mathbf{q}_{\text{new}}$ between $\mathbf{Q}_{\text{neighbors}}$ as the node with lower cost= $C^*(\mathbf{q}_{\text{parent}})$ + c($\mathbf{q}_{\text{parent}}$,$\mathbf{q}_{\text{new}}$) and such that the segment between $\mathbf{q}_{\text{parent}}$ and $\mathbf{q}_{\text{new}}$ is free of collisions.
3. For each node $\mathbf{q}_{\text{near}}$ in $\mathbf{Q}_{\text{neighbors}}$ if its current cost is higher than the cost of reaching it passing from the newly added node (and the segment between the two is free of collision) update its parent and its cost.
4. Add $\mathbf{q}_{\text{new}}$ to $\mathcal{T}$.

As before choice of a metric plays an important role in identifying the nearest node to the random sampled, but also to identify the set of neighbors.

3.4 KRRT*

The RRT* algorithm does not inherently consider dynamic constraints and tries to find the optimal geometric path that connects two configurations. Thus, it is primarily suited for use with holonomic systems. However, model with more complex dynamics requires the consideration of both geometric constraints and the robot's dynamic constraints to find a feasible and optimal trajectory.

KRRT* (Kinodynamic RRT*) approach specifically addresses kinodynamic systems with controllable linear dynamics. It achieves this by optimally connecting any pair of states, with a fixed-final-state free-final-time controller, ensuring asymptotic optimality, with respect to a cost function that is expressed as a trade-off between the duration of the trajectory and the expended control effort.
We can have this kind of controller by extending the well-studied formulation for a fixed-final-state and fixed-final-time optimal control problem for linear and controllable

systems of the type in (3.1) to derive an optimal, open-loop, fixed-final-state free-final-time control policy as was done in [2].

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{A}\boldsymbol{q}(t) + \boldsymbol{B}\boldsymbol{u}(t) + \boldsymbol{c} \tag{3.1}$$

3.5 Optimal trajectory between two states

From now on, we will refer to the general linear model in (3.1), which is assumed to be controllable.

Let $\boldsymbol{Q} = \mathbb{R}^n$ and $\boldsymbol{U} = \mathbb{R}^m$ be the state space and control input space where $\boldsymbol{q}(t) \in \boldsymbol{Q}$ is the state of the robot at time t, $\boldsymbol{u}(t) \in \boldsymbol{U}$ is the control input of the robot at time t, and $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B} \in \mathbb{R}^{n\times m}$, and $\boldsymbol{c} \in \mathbb{R}^n$ are constant and given.

A trajectory of the linear system is defined by a tuple $\boldsymbol{\pi} = (\boldsymbol{q}(), \boldsymbol{u}(), \tau, c)$, where τ is the arrival time or duration of the trajectory, c is the cost of the trajectory, $\boldsymbol{u} : [0, \tau] \to \boldsymbol{U}$ defines the control input along the trajectory and $\boldsymbol{q} : [0, \tau] \to \boldsymbol{Q}$ are the corresponding states.

The cost $c(\boldsymbol{\pi})$ of a trajectory $\boldsymbol{\pi}$ is defined by the function:

$$c(\boldsymbol{\pi}) = \int_0^\tau (1 + \boldsymbol{u}(t)^T \boldsymbol{R}\boldsymbol{u}(t))\, dt \tag{3.2}$$

which penalizes both the duration of the trajectory and the expended control effort, where $\boldsymbol{R} \in \mathbb{R}^{m\times m}$ is positive-definite, constant and given that weights the cost of the control inputs relative to each other and to the duration of the trajectory.

We need to find the best trajectory $\boldsymbol{\pi^*(q_0, q_1)}$ (and its cost $c^*(\boldsymbol{q_0}, \boldsymbol{q_1})$) between any two states $\boldsymbol{q_0} \in \boldsymbol{Q}$ and $\boldsymbol{q_1} \in \boldsymbol{Q}$ that minimizes the function of cost. It is well-known what the optimal control policy is when the final arrival time τ is fixed and decided a priori, but now we want to modify this control policy algorithm in order to find also what is the best arrival time τ^* that minimize the function of cost. This modification introduces an additional degree of freedom in the minimization of (3.2),where both $\boldsymbol{u}(t)$ and τ are involved in a trade-off relationship.

We will start with the analysis of fixed-final-state fixed-final-time controller and then illustrating the modification in order to obtain a fixed-final-state free-final-time controller, emphasizing the practical implementation employed in the project. Having this Lyapunov equation:

$$\dot{\boldsymbol{G}}(t) = \boldsymbol{A}\boldsymbol{G}(t) + \boldsymbol{G}(t)\boldsymbol{A}^T + \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T, \quad \boldsymbol{G}(0) = \boldsymbol{0}. \tag{3.3}$$

we can obtain the value of $\boldsymbol{G}(t)$ (referred to as the *weighted controllability Gramian matrix*), at time t, either through closed-form solution or numerically by integrating (3.3) using the 4th-order Runge-Kutta method. In our case, we have opted for numerical computation. Moreover, as a result of the controllability of the linear system, $\boldsymbol{G}(t)$ is positive definite for $t > 0$ and then its inverse always exists. Now let's declare

$$\dot{\bar{\boldsymbol{q}}}(t) = \boldsymbol{A}\bar{\boldsymbol{q}}(t) + \boldsymbol{c}, \quad \bar{\boldsymbol{q}}(0) = \boldsymbol{q_0}. \tag{3.4}$$

the dynamics of free evolution of the state. In order to obtain the value of $\bar{\boldsymbol{q}}$ at time t we can use both closed form or numerically by integrating (3.4) using 4th-order Runge-Kutta method as said before for $\boldsymbol{G}(t)$.

Then, the optimal control policy for the fixed final state $\boldsymbol{q_1}$, generic fixed final time τ is given by:

$$\boldsymbol{u}(t) = \boldsymbol{R}^{-1}\boldsymbol{B}^T \exp\Big(\boldsymbol{A}^T(\tau - t)\Big)\, \boldsymbol{d}(\tau) \tag{3.5}$$

where $\boldsymbol{d}(\tau)$ is:

$$\boldsymbol{d}(\tau) = \boldsymbol{G}(\tau)^{-1}(\boldsymbol{q_1} - \bar{\boldsymbol{q}}(\tau)) \tag{3.6}$$

which is an open-loop control policy.

Now, we aim to find the optimal arrival time τ^* that minimize the cost function, and to achieve this, we will proceed as follows: by substituting (3.5) into (3.2) and evaluating the integral, we derive a closed-form expression for the cost function that depends on final time τ:

$$c(\tau) = \tau + (\boldsymbol{q_1} - \bar{\boldsymbol{q}}(\tau))^T \boldsymbol{d}(\tau) \tag{3.7}$$

To find the optimal arrival time τ^* we can study its derivative in respect to τ, but it can may have multiple local minimum.

By noting that $c(\tau) > \tau$ for all $\tau > 0$ as $\boldsymbol{G}(\tau)$ is positive-definite, instead of studying the derivative directly, we monitor, step-by-step during iterative forward integration of $\dot{\boldsymbol{G}}(t)$ and $\dot{\bar{\boldsymbol{q}}}(t)$ for increasing value of τ, the minimal cost c^* value of (3.7) found with it's corresponding time τ^*, and we can terminate the loop when the iterative variable τ is $\geq c^*$ found until now to ensure that c^* is the global minimum of $c(\tau)$. Below there is the pseudo-code of the algorithm just described (1). We estimate the values of the functions every $k\Delta t$ seconds (where k is a positive integer and Δt is an hyperparameter), and h is the interval of time used in every step of the 4th-order Runge-Kutta algorithm. At the end of the loop we also have $\boldsymbol{d}(\tau^*)$ that will be used later.

Now that we have the optimal final time τ^*, we can substitute it in (3.5) in order to obtain the optimal control policy in terms of time and control effort in a closed form, but we will continue in estimating it numerically as will follow.

In order to have a more compact notation, let's define:

$$\boldsymbol{y}(t) = \exp\Big(\boldsymbol{A}^T(\tau^* - t)\Big)\, \boldsymbol{d}(\tau^*) \tag{3.8}$$

that is the solution to the differential equation:

$$\dot{\boldsymbol{y}}(t) = -\boldsymbol{A}^T\boldsymbol{y}(t), \quad \boldsymbol{y}(\tau^*) = \boldsymbol{d}(\tau^*) \tag{3.9}$$

and we can rewrite the optimal control policy as:

$$\boldsymbol{u}(t) = \boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{y}(t) \tag{3.10}$$

Algorithm 1 Forward Integration of $\dot{\boldsymbol{G}}(t)$ and $\dot{\bar{\boldsymbol{q}}}(t)$

$\tau, k = 0$
$\boldsymbol{G}[0] = \boldsymbol{0}$
$\bar{\boldsymbol{q}}[0] = \boldsymbol{q_0}$
$c^* = \infty$
$\Delta t = 0.1$
h=Δt
while $\tau < c^*$ **do**
    $\tau \leftarrow \tau + \Delta t$
    k $\leftarrow$ k +1
    $\bar{\boldsymbol{q}}$[k] $\leftarrow \bar{\boldsymbol{q}}$[k-1] + RK4($\dot{\bar{\boldsymbol{q}}}, \bar{\boldsymbol{q}}$[k-1], h)
    $\boldsymbol{G}$[k] $\leftarrow \boldsymbol{G}$[k-1] + RK4($\dot{\boldsymbol{G}}, \boldsymbol{G}$[k-1], h)
    c[k] $\leftarrow \tau + (\boldsymbol{q_1} - \bar{\boldsymbol{q}}[\text{k}])^T \boldsymbol{G}[\text{k}]^{-1} (\boldsymbol{q_1} - \bar{\boldsymbol{q}}[\text{k}])$
    if c[k] $< c^*$ **then**
        $c^* \leftarrow c$[k]
        $\tau^* \leftarrow \tau$
        $k^* \leftarrow k$
    end if
end while
$\boldsymbol{d}(\tau^*) = \boldsymbol{G}[k^*]^{-1} (\boldsymbol{q_1} - \bar{\boldsymbol{q}}[k^*])$

To derive the optimal state trajectory between $\boldsymbol{q_0}$ and $\boldsymbol{q_1}$ (which lasts τ^* seconds) and it's corresponding optimal control policy, we substitute (3.10) into the linear dynamics described by (3.1), yielding the following differential equation:

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{A}\boldsymbol{q}(t) + \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T\boldsymbol{y}(t) + \boldsymbol{c}, \quad \boldsymbol{q}(\tau^*) = \boldsymbol{q_1} \tag{3.11}$$

and then, combining this with (3.9), we obtain the composite differential equation:

$$\begin{bmatrix} \dot{\boldsymbol{q}}(t) \\ \dot{\boldsymbol{y}}(t) \end{bmatrix} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B}\boldsymbol{R}^{-1}\boldsymbol{B}^T \\ \boldsymbol{0} & -\boldsymbol{A}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{q}(t) \\ \boldsymbol{y}(t) \end{bmatrix} + \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{q}(\tau^*) \\ \boldsymbol{y}(\tau^*) \end{bmatrix} = \begin{bmatrix} \boldsymbol{q_1} \\ \boldsymbol{d}(\tau^*) \end{bmatrix} \tag{3.12}$$

By integrating the differential equation (3.12) backward in time for $0 \leq t \leq \tau^*$ through 4th-order Runge-Kutta method, we numerically obtain estimates of $\boldsymbol{q}(t)$ and $\boldsymbol{y}(t)$ every $k\Delta t$ seconds. The latter is then substituted into (3.10) to determine $\boldsymbol{u}(t)$. Finally we have the complete optimal trajectory $\boldsymbol{\pi^*(q_0, q_1)} = (\boldsymbol{q}(t), \boldsymbol{u}(t), \tau^*, c^*)$, where $\boldsymbol{q}(0) = \boldsymbol{q_0}$ and $\boldsymbol{q}(\tau^*) = \boldsymbol{q_1}$.

In conclusion, we now have an algorithm that, given an initial state $\boldsymbol{q_0}$ and a final state $\boldsymbol{q_1}$, can determine the optimal trajectory connecting these states where the total traversal time is not arbitrarily chosen by us but it is selected by the algorithm in order to minimize the cost function, making it a fixed-final-state, free-final-time solution.

3.6 KRRT* algorithm

In the KRRT* algorithm, once able to compute the optimal trajectory between two configurations, it can be used following the exact same steps of RRT*.

The full algorithm follow this steps:

1. Sample a random configuration $\mathbf{q}_{\text{new}}$ from the configuration space using random uniform probability.
2. Select the closest nodes $\mathbf{Q}_{\text{neighbors}}$ to $\mathbf{q}_{\text{new}}$ in the Tree $\mathcal{T}$.
3. Choose the parent $\mathbf{q}_{\text{parent}}$ of $\mathbf{q}_{\text{new}}$ between $\mathbf{Q}_{\text{neighbors}}$ as the node with lower cost= $C^*(\mathbf{q}_{\text{parent}})$ + c($\mathbf{q}_{\text{parent}}$,$\mathbf{q}_{\text{new}}$) and such that the optimal trajectory between $\mathbf{q}_{\text{parent}}$ and $\mathbf{q}_{\text{new}}$ is free of collisions.
4. For each node $\mathbf{q}_{\text{near}}$ in $\mathbf{Q}_{\text{neighbors}}$ if its current cost is higher than the cost of reaching it from the newly added node (and the optimal trajectory between the two is free of collision) update its parent and its cost.
5. Add $\mathbf{q}_{\text{new}}$ to $\mathcal{T}$.

The algorithm builds a tree of trajectories in the free configuration space rooted in the initial configuration. To ensure a trajectory is collision-free, for each configuration in the trajectory, the entire footprint of the robot is calculated using forward kinematics,

Algorithm 2 Kinodynamic RRT*

$\mathcal{T} \leftarrow \{\mathbf{q}_{\text{start}}\}$
$\mathbf{q}_{\text{start}} \in \mathbf{Q}_{\text{free}}, \mathbf{q}_{\text{goal}} \in \mathbf{Q}_{\text{free}}$
for $i = 1, \infty$ **do**
    Randomly sample $\mathbf{q_i} \in \mathbf{Q}_{\text{free}}$
    $\mathbf{q} \leftarrow \arg\min_{\{\mathbf{q} \in \mathcal{T} | c^*[\mathbf{q}, \mathbf{q_i}] < r \wedge \text{COLLISIONFREE}[\pi^*[\mathbf{q}, \mathbf{q_i}]]\}} (\text{cost}[\mathbf{q}] + c^*[\mathbf{q}, \mathbf{q_i}])$
    $\text{parent}[\mathbf{q_i}] \leftarrow \mathbf{q}$
    $\text{cost}[\mathbf{q_i}] \leftarrow \text{cost}[\mathbf{q}] + c^*[\mathbf{q}, \mathbf{q_i}]$
    for all $\{\mathbf{q} \in \mathcal{T} \cup \{\mathbf{q}_{\text{goal}}\}\} | c^*[\mathbf{q_i}, \mathbf{q}] < r \wedge \text{cost}[\mathbf{q_i}] + c^*[\mathbf{q_i}, \mathbf{q}] < \text{cost}[\mathbf{q}] \wedge \text{COLLISIONFREE}[\pi^*[\mathbf{q_i}, \mathbf{q}]]$ **do**
        $\text{cost}[\mathbf{q}] \leftarrow \text{cost}[\mathbf{q_i}] + c^*[\mathbf{q_i}, \mathbf{q}]$
        $\text{parent}[\mathbf{q}] \leftarrow \mathbf{q_i}$
    end for
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{\mathbf{q_i}\}$
end for

Figure 2: The complete KRRT* algorithm. The optimal trajectory $\pi^*[\mathbf{q_i}, \mathbf{q}]$ is computed using the controller as described before. The COLLISIONFREE operation checks whether all the points in the trajectory are in the free configuration space, or in other words if the footprint of the robot in each configuration collides with any obstacle, whether the control inputs are in the admissible ranges and whether the constraints on v_{S_i} are satisfied, i.e. they not change sign during the path.

and then it is checked for any potential intersection with any obstacle. It is also verifyed that the control inputs are in their admissible ranges and that the velocity of the point S_i does not change in sign.

The algorithm as presented in Figure 2 utilizes also the goal configuration in the rewiring step. This allows to find a trajectory that exactly arrives at the goal configuration, rather then a close region.

In our application, instead of computing the set of neighbors we utilized the whole Tree. This approach, while it does not impact the effectiveness of the algorithm, leads to a slower converge time. Further approaches will be discussed in section 5.

3.7 Linearization of the Dynamic model

In order to use KRRT* with non-linear systems, it's necessary a first-order Taylor approximation of the system in every iteration of the algorithm.

Let's define the non-linear dynamic model of Neobotix MPO-700 (2.11) in a composite way as $\dot{\boldsymbol{q}} = \boldsymbol{f}(\boldsymbol{q}, \boldsymbol{u})$. The linearization is done by a first-order Taylor approximation

around $\boldsymbol{q} = \hat{\boldsymbol{q}}$ and $\boldsymbol{u} = \mathbf{0}$ in this way:

$$\boldsymbol{A} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{q}}(\hat{\boldsymbol{q}}, \mathbf{0}), \quad \boldsymbol{B} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}(\hat{\boldsymbol{q}}, \mathbf{0}), \quad \boldsymbol{c} = \boldsymbol{f}(\hat{\boldsymbol{q}}, \mathbf{0}) - \boldsymbol{A}\hat{\boldsymbol{q}}.$$

For our purpose the point of linearization correspond to $\hat{\boldsymbol{q}} = \boldsymbol{q_i}$ which is the configuration randomly sampled at the beginning of each iteration. Moreover, it is necessary to ensure that this linearized model is formally controllable.

For a generic approximation point $\hat{\boldsymbol{q}}$, $\boldsymbol{A}$ is equal to:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & -\hat{v}_{S_1} s(\hat{\theta}+\hat{\phi}_1) + \hat{\omega}(P_{1,x} c\hat{\theta} - P_{1,y} s\hat{\theta}) & -\hat{v}_{S_1} s(\hat{\theta}+\hat{\phi}_1) & c(\hat{\theta}+\hat{\phi}_1) & P_{1,y} c\hat{\theta} + P_{1,x} s\hat{\theta} \\ 0 & 0 & \hat{v}_{S_1} c(\hat{\theta}+\hat{\phi}_1) + \hat{\omega}(P_{1,y} c\hat{\theta} + P_{1,x} s\hat{\theta}) & \hat{v}_{S_1} c(\hat{\theta}+\hat{\phi}_1) & s(\hat{\theta}+\hat{\phi}_1) & -P_{1,x} c\hat{\theta} + P_{1,y} s\hat{\theta} \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$\boldsymbol{B}$ is equal to:

$$\boldsymbol{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

and $\boldsymbol{c}$:

$$\boldsymbol{c} = \begin{bmatrix} \hat{v}_{S_1}(\hat{\theta}+\hat{\phi}_1) s(\hat{\theta}+\hat{\phi}_1) - \hat{\omega}\hat{\theta}(P_{1,x} c\hat{\theta} - P_{1,y} s\hat{\theta}) \\ -\hat{v}_{S_1}(\hat{\theta}+\hat{\phi}_1) c(\hat{\theta}+\hat{\phi}_1) - \hat{\omega}\hat{\theta}(P_{1,y} c\hat{\theta} + P_{1,x} s\hat{\theta}) \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

where c represents the cosine function and s represents the sine function.

This new linear system is approximately valid only when $\boldsymbol{q} \approx \hat{\boldsymbol{q}}$ and $\boldsymbol{u} \approx \mathbf{0}$. If we move from the linearization point and use inputs different from zero, the linear model is no longer representing the real system. And in fact, during the development of the project, we have seen that if $\boldsymbol{q_0}$ and $\boldsymbol{q_1}$ are numerically far, the resulting optimal path generated by linearizing the system differs significantly from the path of the original non-linear system, which was determined through the evolution of (2.11) using the optimal control policy derived from the linearized system.

Studying the controllability property of the new linear system for a generic point of linearization $\boldsymbol{q} = \hat{\boldsymbol{q}}$ through the controllability matrix of this form:

$$\mathcal{C} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{AB} & \boldsymbol{A^2 B} & \cdots & \boldsymbol{A^5 B} \end{bmatrix},$$

we find that the rank of submatrix $\begin{bmatrix} \boldsymbol{B} & \boldsymbol{AB} \end{bmatrix}$ is always 6 if $v_{S_1} \neq 0$. This condition, as previously said, is necessary to prevent the robot from pivoting on S_1. Since this condition is already ensured in COLLISIONFREE-step of the algorithm by discarding any path that fails to meet it, the controllability matrix consistently maintains full rank. This guarantees the system's controllability at all times, and then the possibility to use the planner described above. As mentioned before in the model chapter, for the model of type (2.6) the matrix $\boldsymbol{A}$ would be a matrix of zeros for any point of approximation $\hat{\boldsymbol{q}}$, so would be difficult to study controllabilty and then using the controller.

4 Simulations

This section is dedicated to discussing various simulations conducted in Python to validate the motion planning algorithm outlined in Section 3. The simulations were performed across environments that vary in the number of obstacles—ranging from none to several—and in map sizes: small, medium, and large.

4.1 Maps

Figures 3, 4, and 5 illustrate the maps used in the Python simulations. Additionally, we provide a detailed description of each map's characteristics, which are summarized in Table 1.

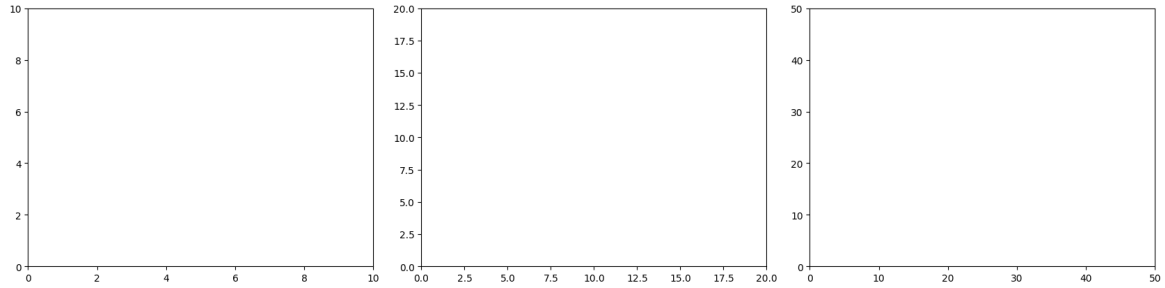


Figure 3: Small, medium and big empty maps

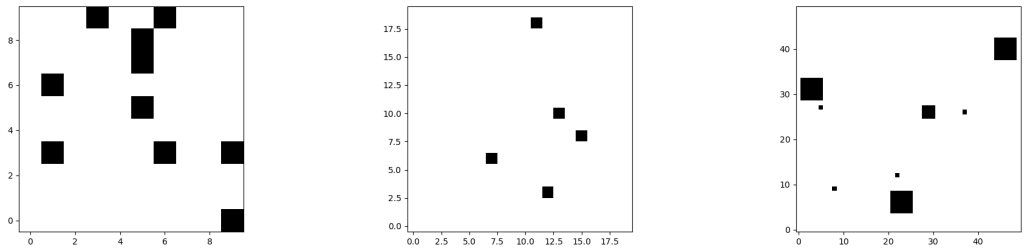


Figure 4: Small, medium and big maps with few obstacles

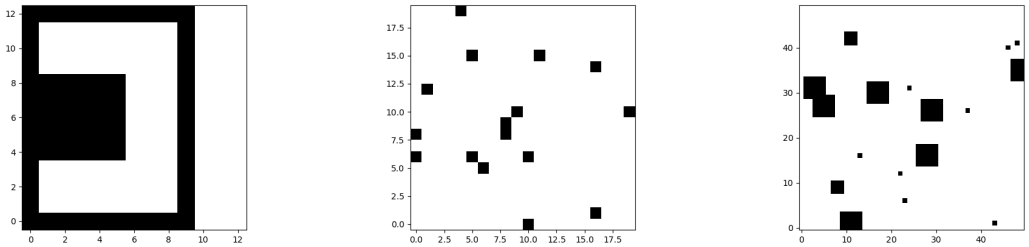


Figure 5: Small, medium and big maps with many obstacles

Map	Dimension	Resolution	Obstacle-probability
Small-empty	10 x 10 (m)	1 (m)	0.0
Small-few	10 x 10 (m)	1 (m)	0.12
Small-many	13 x 13 (m)	1 (m)	custom
Medium-empty	20 x 20 (m)	1 (m)	0.0
Medium-few	20 x 20 (m)	1 (m)	0.01
Medium-many	20 x 20 (m)	1 (m)	0.04
Big-empty	50 x 50 (m)	1 (m)	0.0
Big-few	50 x 50 (m)	1 (m)	0.005
Big-many	50 x 50 (m)	1 (m)	0.008

Table 1: Maps details

In addition to variable-sized environments, we have generated maps containing obstacles of diverse sizes, ranging randomly from smaller to larger. We have also generated custom maps, like in the case of the small map with many obstacle.

4.2 KRRT* Hyperparameters

Selecting appropriate hyperparameters was crucial to optimize the planner's performance. Initially, we experimented with various configurations of the matrix $\boldsymbol{R}$. We began with $\boldsymbol{R} = \boldsymbol{I}$, the identity matrix, but observed that this setting led the robot to prefer solutions with a high value of ω, causing the mobile base to spin excessively. To mitigate this issue, we adjusted $\boldsymbol{R}$ to:

$$\boldsymbol{R} = \begin{bmatrix} w_1 & 0 & 0 \\ 0 & w_2 & 0 \\ 0 & 0 & w_3 \end{bmatrix},$$

where $w_1 = w_2 = 1$ and $w_3 = 10$, to heavily penalize high values of ω.

Furthermore, we configured the planner to reject any trajectory that approached too close to obstacles or the boundaries of the map, setting a safety distance tolerance of 0.5 meters.

Another critical hyperparameter was the number of iterations N, which serves as a bottleneck for the algorithm. While the algorithm is theoretically guaranteed to find the optimal solution asymptotically, achieving this in practice would require an impractical amount of time. To obtain efficient solutions within a reasonable timeframe, we set different time limits depending on the map type being used.

4.3 Robot Hyperparameter

As explained in Equation 2.9, the joint velocity v_{s_1} should never assume a value of zero, thus we must prevent the joint velocity from changing sign, that is, from shifting between positive and negative values or vice versa. In our simulations, we have set the initial velocity v_{s_1} of the starting node to always be positive, therefore, it must maintain a positive sign throughout the trajectory. To achieve this, we have imposed the following constraint:

$$v_{s_1} > 0 \, \frac{m}{s} \tag{4.1}$$

Therefore each node sampled randomly will always have a positive velocity v_{s_1}. We also need to take into account the dimensions of the Neobotix MPO-700 robot for both modeling purposes and collision detection. According to the datasheet:

- d is 0.045 meters.
- $P_{1,x}$ is 0.24 meters.
- $P_{1,y}$ is 0.19 meters.
- The robot's length is 0.741 meters.
- The robot's width is 0.590 meters.

4.4 Numerical Results

In this section, we present the outcomes of the simulations carried out in Python.

In Table 2, we display the results obtained from various simulations. **Cost** refers to the definition in 3.2, which quantifies the total cost of all trajectories within the optimal path. **Iterations** refers to the number of iterations executed by the planner. **KRRT* Time** denotes the time required by the planner to complete all iterations of the algorithm. **Trajectory Time** is the cumulative time of all τ^*, representing the duration of each trajectory connecting the nodes along the optimal path. **Nodes** represents all the random nodes added to the Random Tree, including the start and goal nodes.

Map	**Cost**	**Iterations**	**KRRT* Time**	**Traj. Time**	**Nodes**
Small-empty	84.59	100	42 (s)	33 (s)	19
Small-few	26.8	200	301 (s)	8 (s)	37
Small-many	249.69	500	1362 (s)	34 (s)	55
Medium-empty	37.51	100	367 (s)	28 (s)	53
Medium-few	13.66	500	2659 (s)	12 (s)	109
Medium-many	66.79	500	1491 (s)	28 (s)	56
Big-empty	18	1	0.12 (s)	15 (s)	2
Big-few	61.82	100	426 (s)	21 (s)	84
Big-many	64.691	500	9349 (s)	27 (s)	324

Table 2: Maps simulations results

In all three **empty** map scenarios, the algorithm can identify the best trajectory with just few iterations, in particular for very simple trajectories it can requires even just 1 iteration as shown for the Big-empty map, as it can directly link the goal node to the start node in the absence of obstacles.

For maps with **few** and **many** obstacles, the algorithm requires many iterations to find the optimal path, partially due to the peculiarities of the sampled nodes, which may have advantageous positions but unusual frame orientations or ϕ angles.

Since we have focused more on the model of the mobile platform than on optimizing the KRRT* algorithm, the algorithm often requires too many iterations to reach the goal node and find an optimal path.

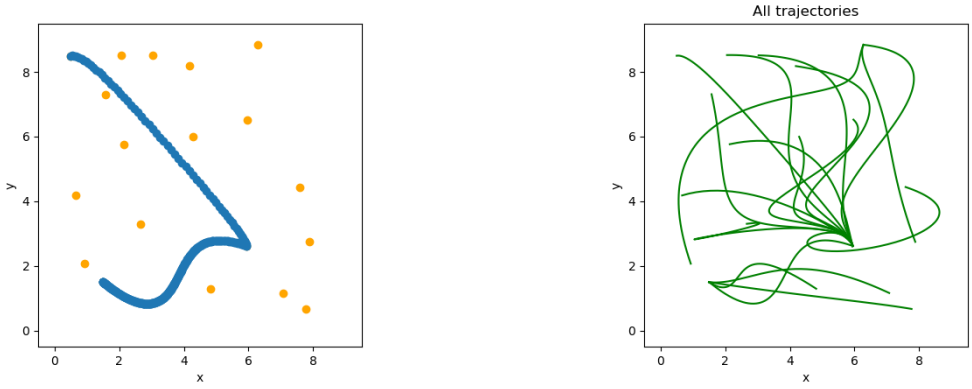
To aid the algorithm in finding a viable path, we have *proposed* certain nodes to the algorithm. These nodes are selected *a priori* by examining the map and identifying states that appear favorable. Occasionally, this approach has helped the algorithm discover efficient plans, especially when the random sampling process might only generate nodes that are either unfeasible or too costly, prompting the algorithm to opt for the proposed nodes.

In other instances, the sampling process is exceptionally *lucky*, generating nodes that are even better than the proposed ones.

4.5 Simulations Conducted

Finally, we present the graphs from the simulation conducted.
In the following graphs we will refers to v_{s_1} as to v_1 and to a_{s_1} as to a_v_1.

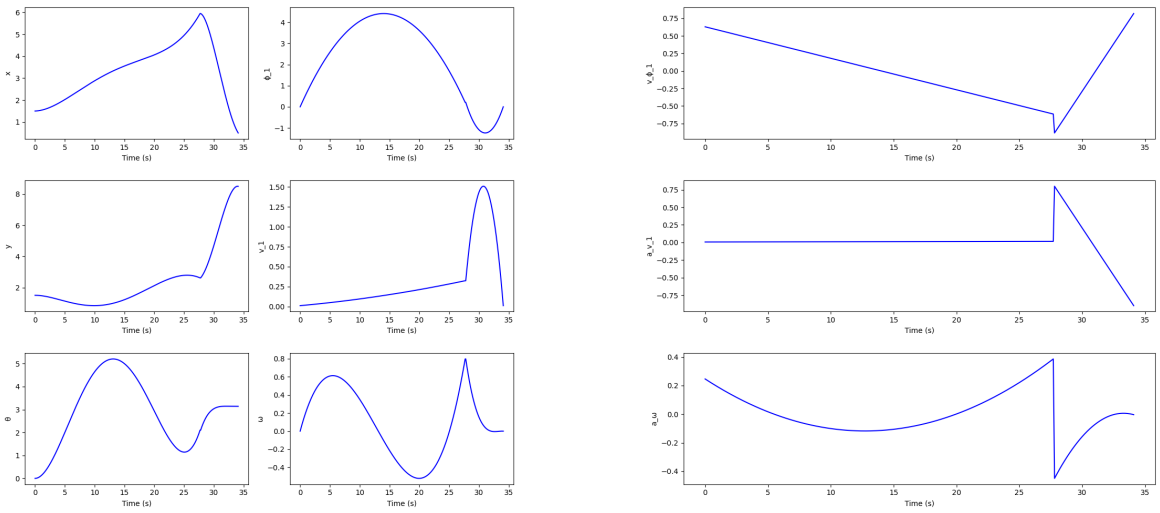
Start state = [1.5, 1.5, 0, 0, 0.01, 0], Goal state = [0.5, 8.5, π, 0, 0.01, 0].



(a) Optimal path retrieved by KRRT*

(b) All trajectories calculated among all nodes

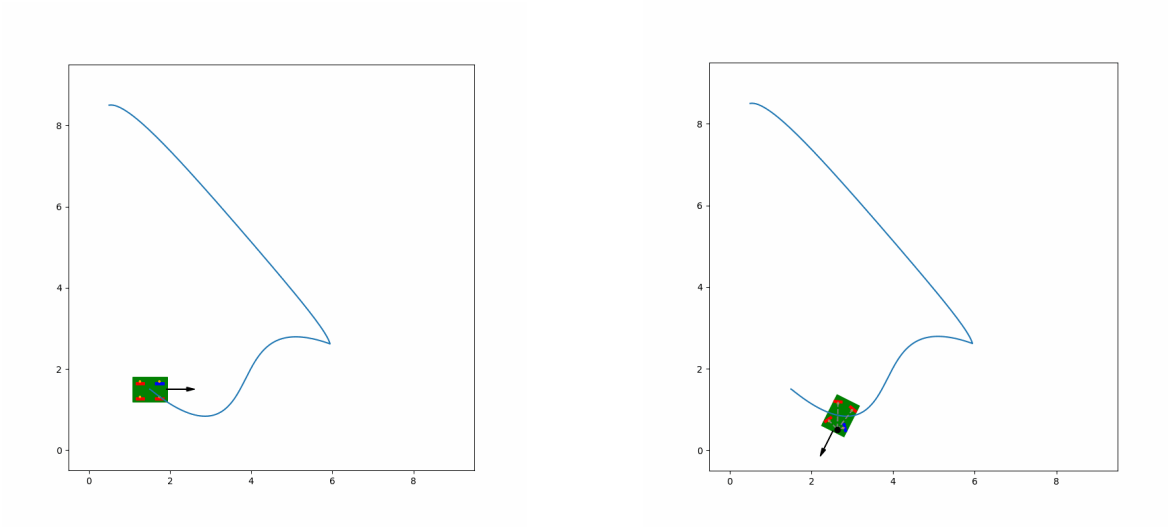
Figure 6: These images showcase a simulation conducted on the *Small-empty* map, highlighting all nodes selected by the algorithm along with all possible trajectories, particularly emphasizing the optimal trajectory.



(a) State evolution during the Optimal Trajectory

(b) Inputs used by the model during the Optimal Trajectory

Figure 7: We would like to highlight that the acceleration a_{v_1} appears to be 0 $\frac{m}{s^2}$ for the first 28 seconds, but the actual value is 0.009 $\frac{m}{s^2}$.



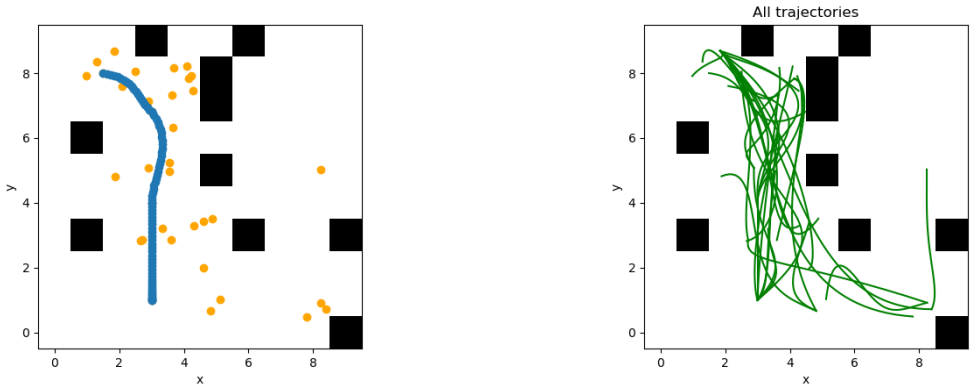
(a) (b)

(c) (d)

(e) (f)

Figure 8

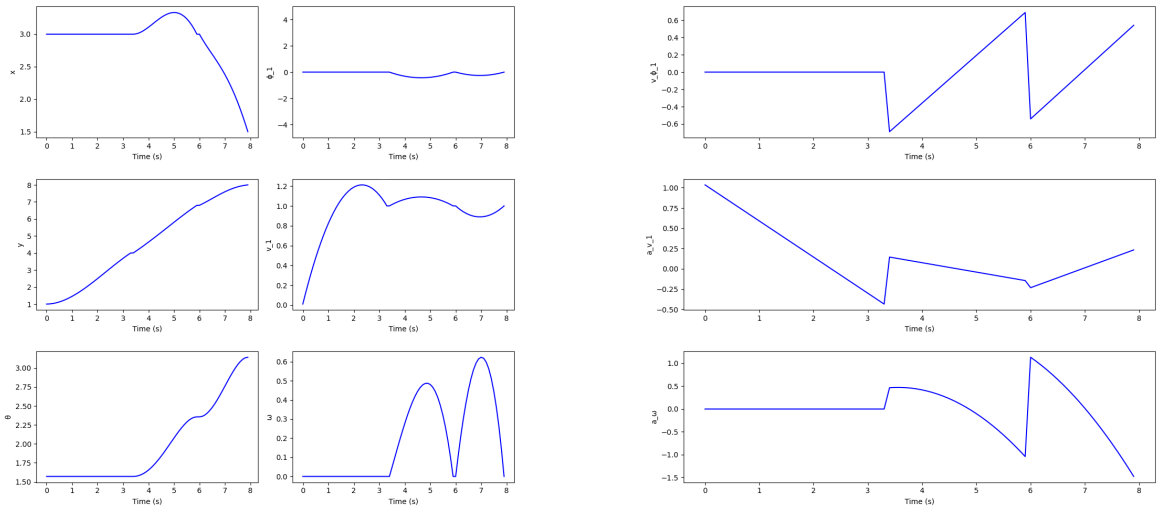
Start state = [3, 1, $\frac{\pi}{2}$, 0, 0.01, 0], Goal state = [1.5, 8, π, 0, 1, 0].



(a) Optimal path retrieved by KRRT*

(b) All trajectories calculated among all nodes

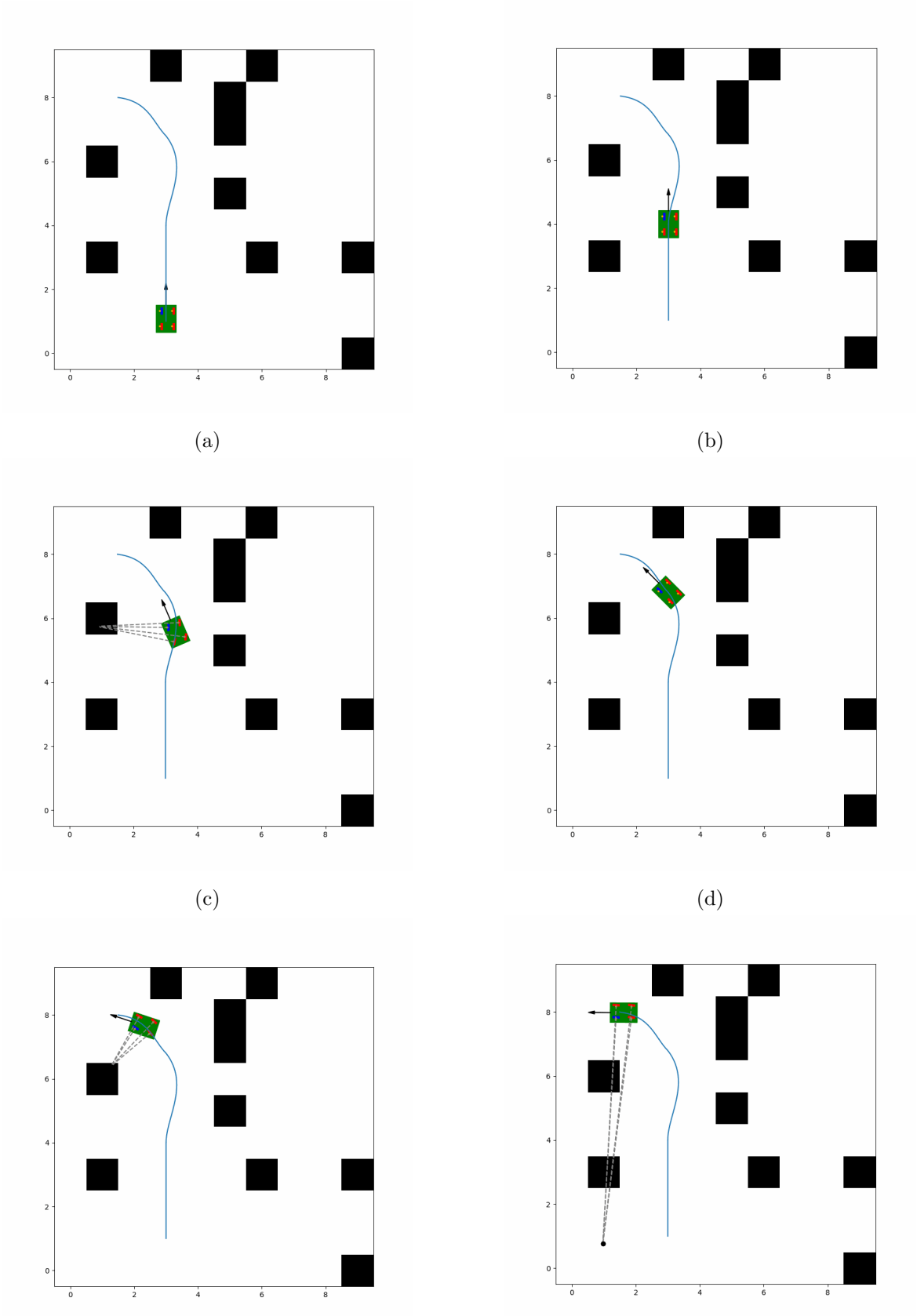
Figure 9: These images showcase a simulation conducted on the *Small-few* map, highlighting all nodes selected by the algorithm along with all possible trajectories, particularly emphasizing the optimal trajectory



(a) State evolution during the Optimal Trajectory

(b) Inputs used by the model during the Optimal Trajectory

Figure 10



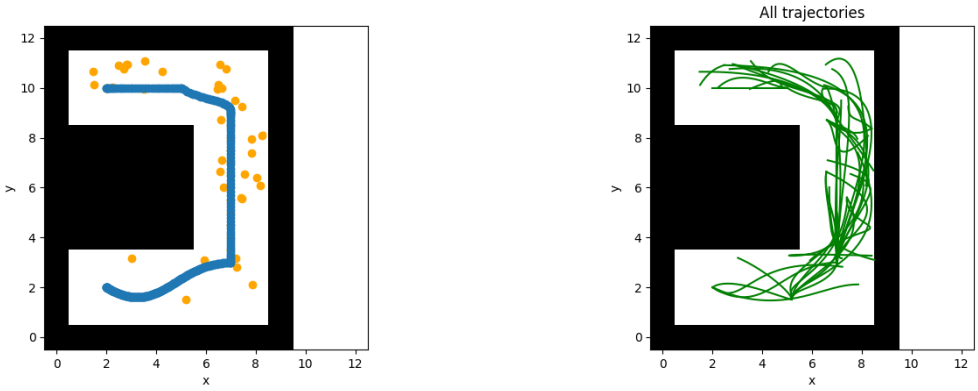
(a) (b)

(c) (d)

(e) (f)

Figure 11

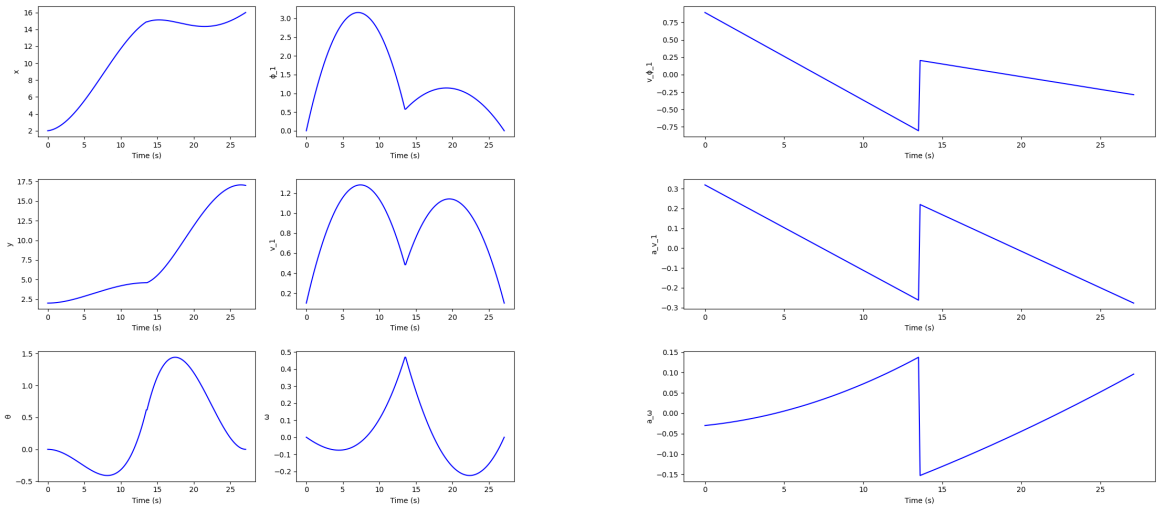
Start state = [2, 2, 0, 0, 0.01, 0], Goal state = [2, 10, 0, 0, 0.01, 0].



(a) Optimal path retrieved by KRRT*

(b) All trajectories calculated among all nodes

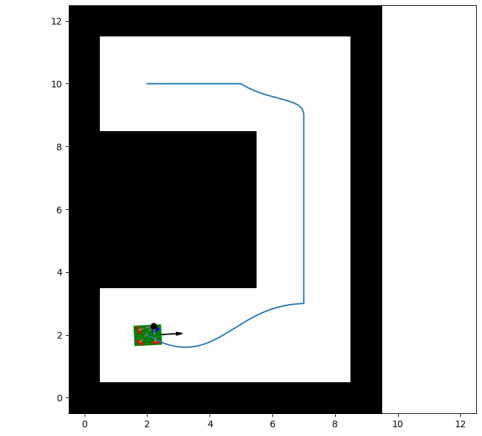
Figure 12: These images showcase a simulation conducted on the *Small-many* map, highlighting all nodes selected by the algorithm along with all possible trajectories, particularly emphasizing the optimal trajectory



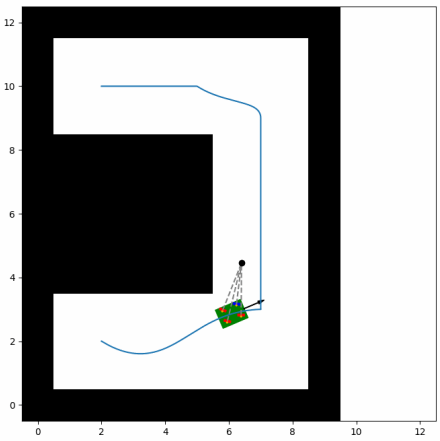
(a) State evolution during the Optimal Trajectory

(b) Inputs used by the model during the Optimal Trajectory

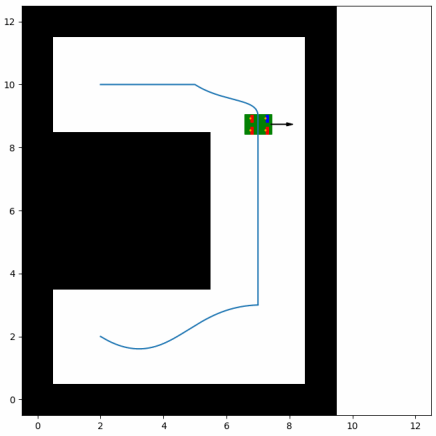
Figure 13



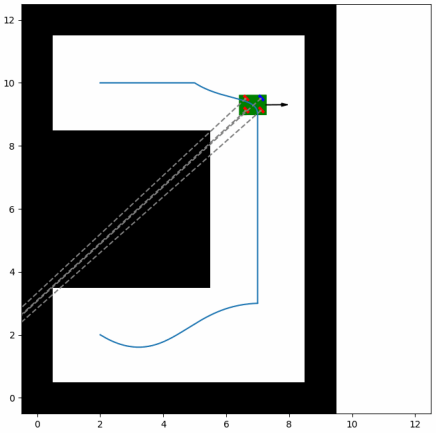
(a)



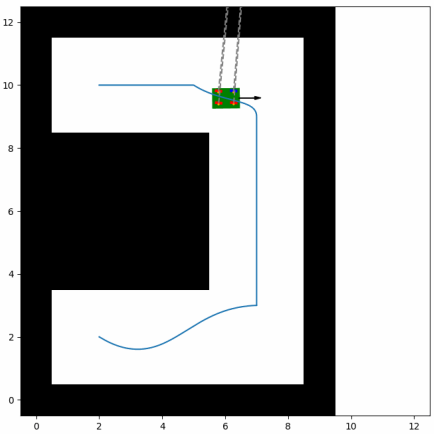
(b)



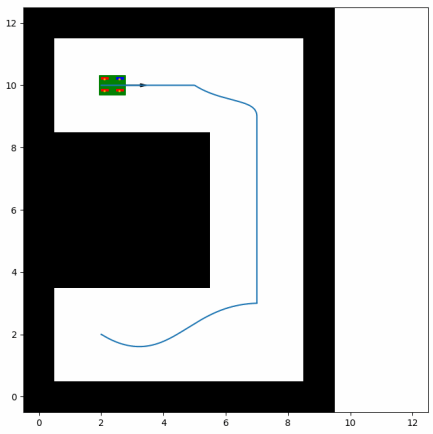
(c)



(d)



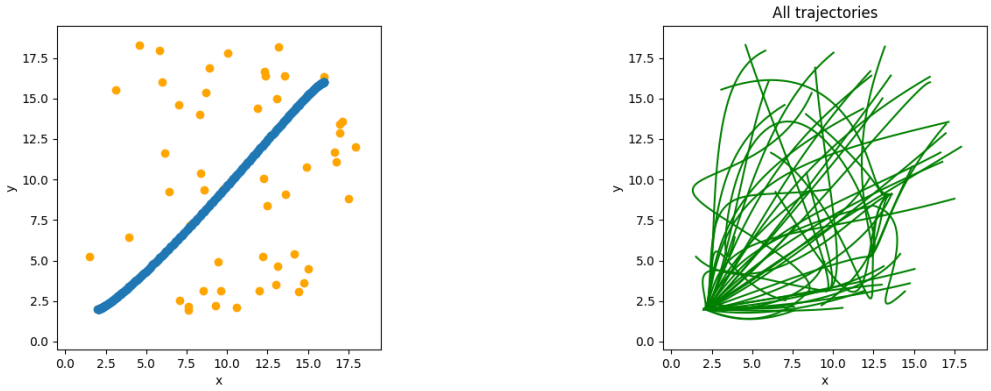
(e)



(f)

Figure 14

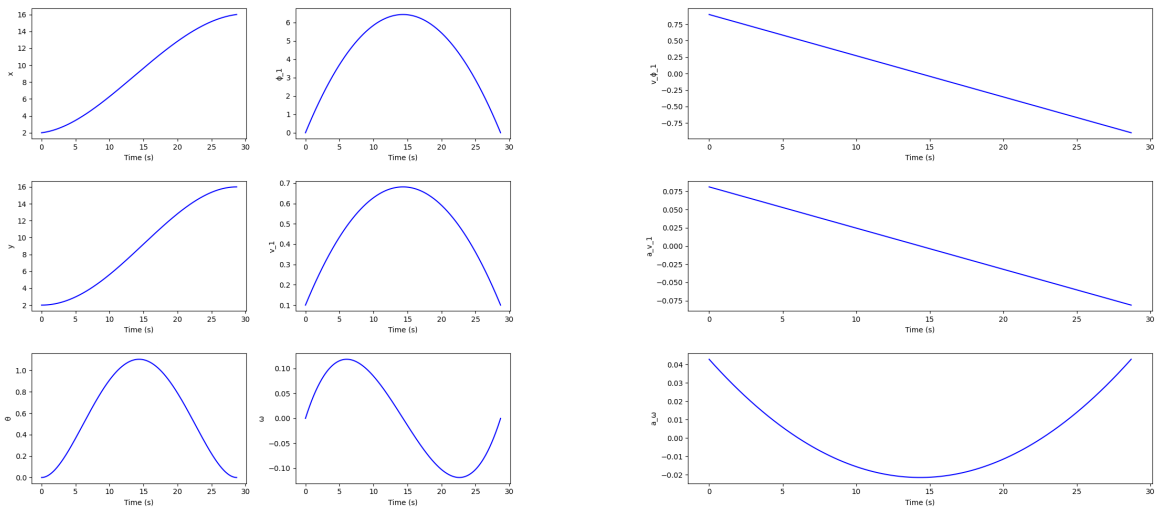
Start state = [2, 2, 0, 0, 0.1, 0], Goal state = [16, 16, 0, 0, 0.1, 0].



(a) Optimal path retrieved by KRRT*

(b) All trajectories calculated among all nodes

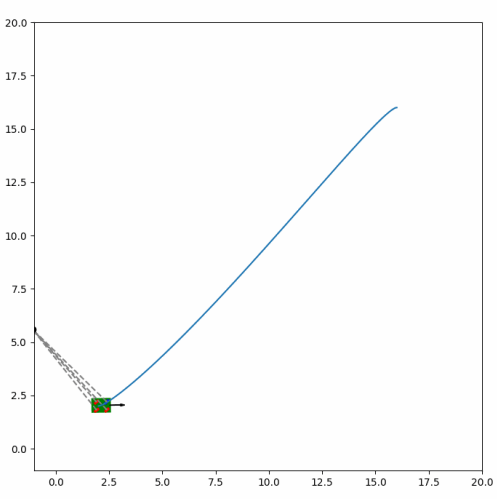
Figure 15: These images showcase a simulation conducted on the *Medium-empty* map, highlighting all nodes selected by the algorithm along with all possible trajectories, particularly emphasizing the optimal trajectory



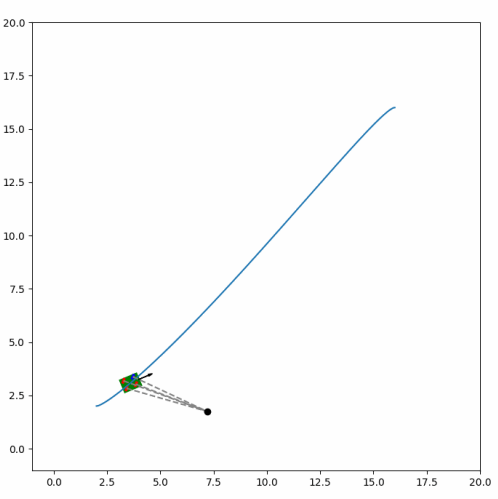
(a) State evolution during the Optimal Trajectory

(b) Inputs used by the model during the Optimal Trajectory

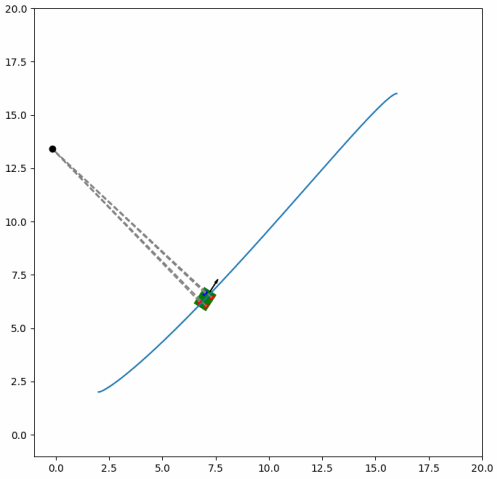
Figure 16



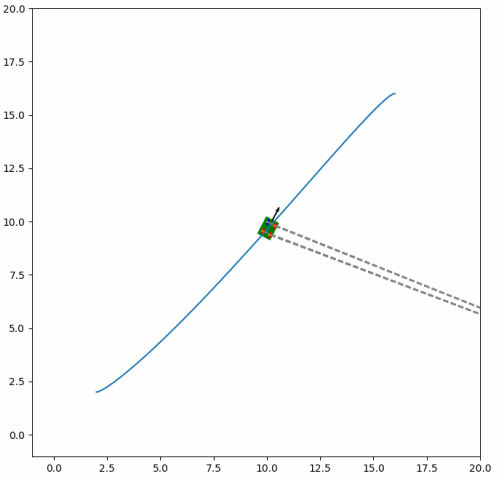
(a)



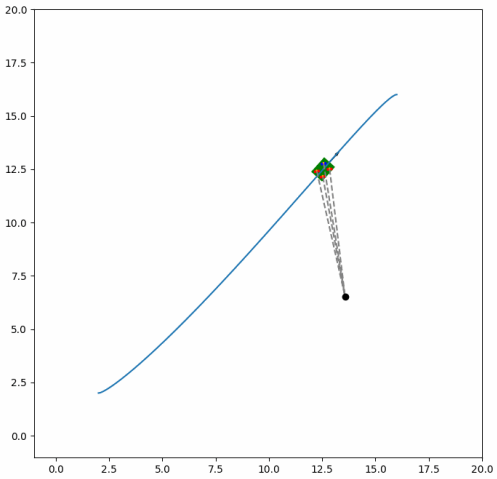
(b)



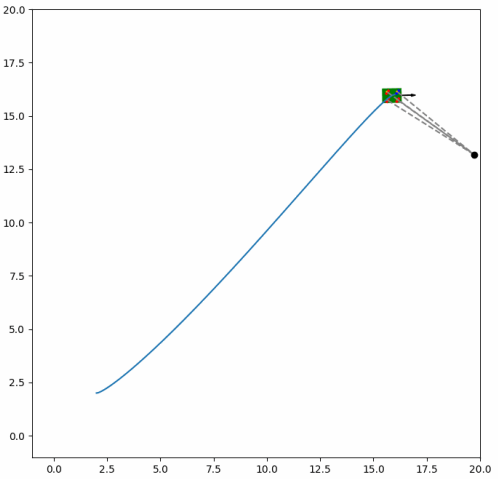
(c)



(d)



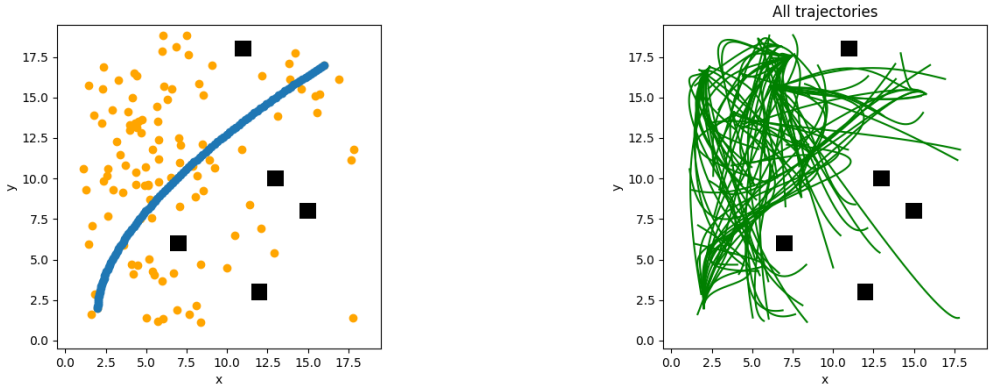
(e)



(f)

Figure 17

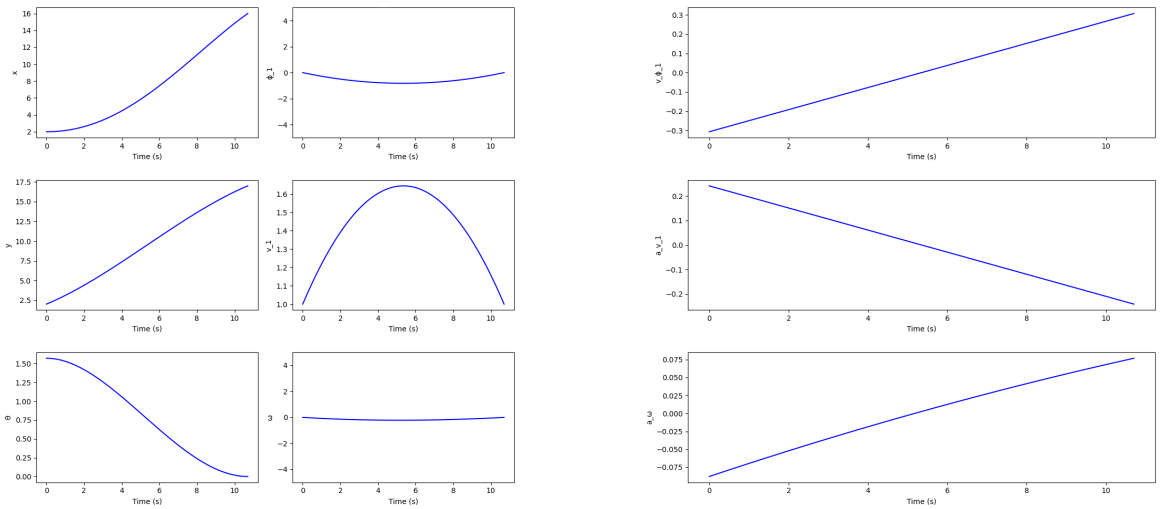
Start state = [2, 2, $\frac{\pi}{2}$, 0, 1, 0], Goal state = [16, 17, 0, 0, 1, 0].



(a) Optimal path retrieved by KRRT*

(b) All trajectories calculated among all nodes

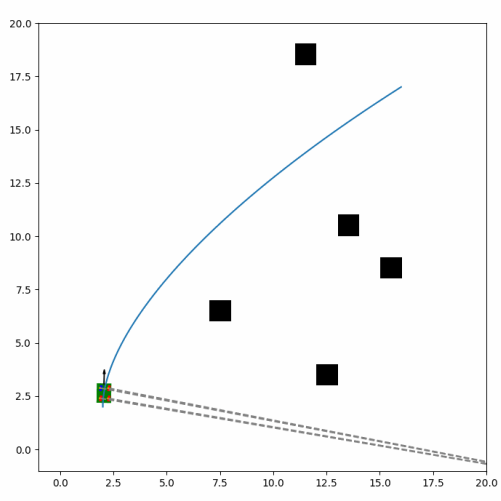
Figure 18: These images showcase a simulation conducted on the *Medium-few* map, highlighting all nodes selected by the algorithm along with all possible trajectories, particularly emphasizing the optimal trajectory



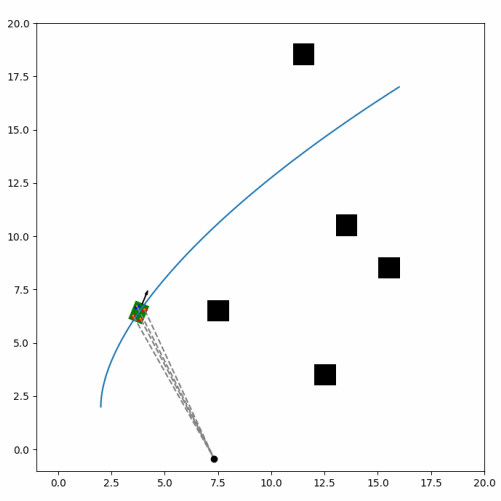
(a) State evolution during the Optimal Trajectory

(b) Inputs used by the model during the Optimal Trajectory

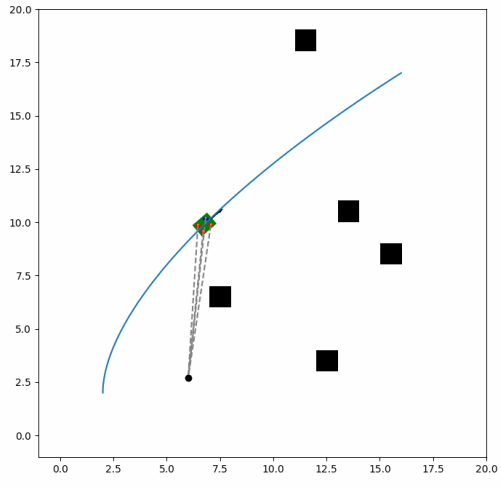
Figure 19



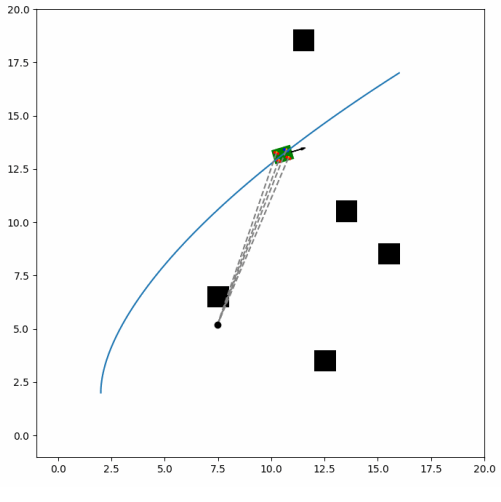
(a)



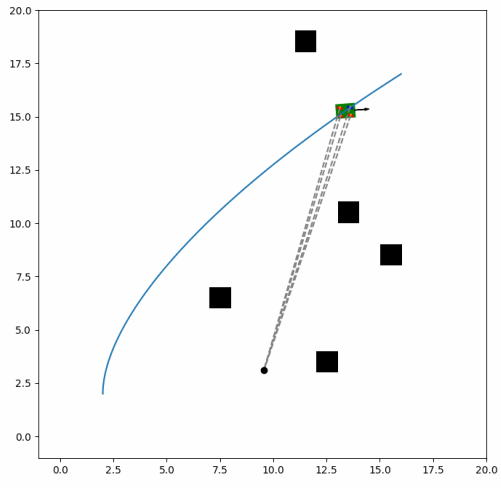
(b)



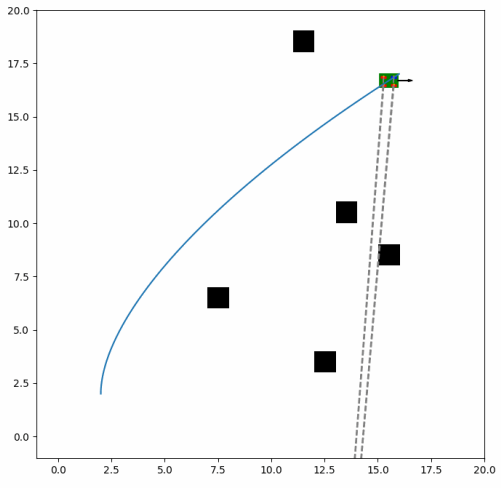
(c)



(d)



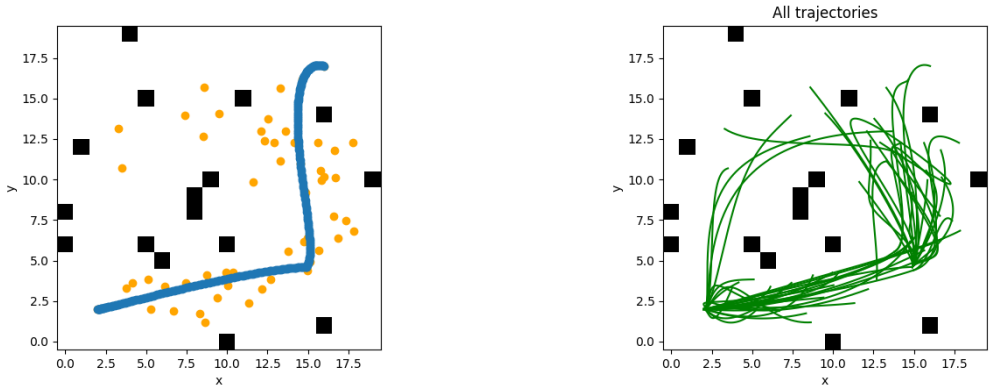
(e)



(f)

Figure 20

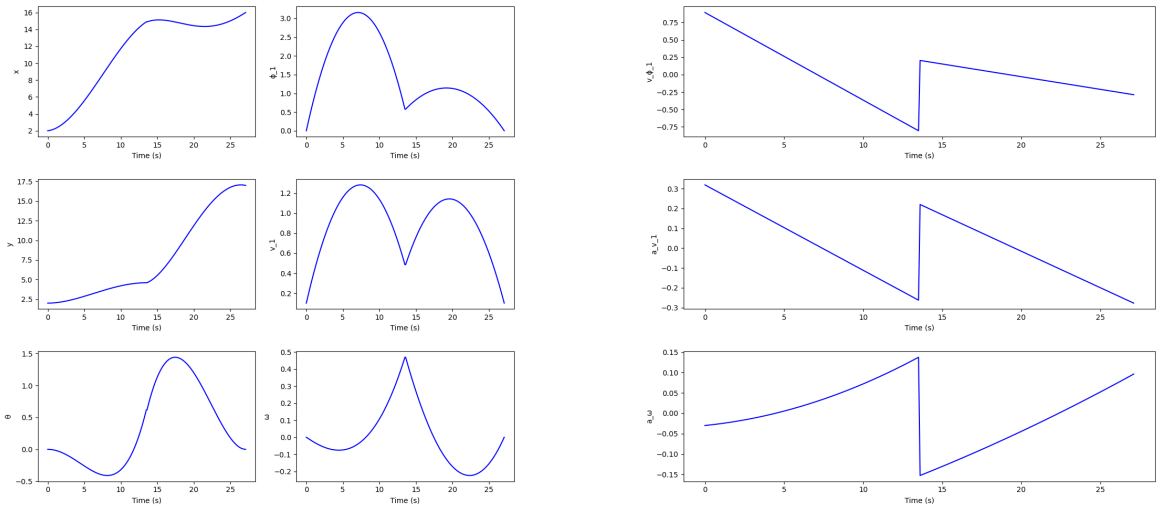
Start state = [2, 2, 0, 0, 0.1, 0], Goal state = [16, 17, 0, 0, 0.1, 0].



(a) Optimal path retrieved by KRRT*

(b) All trajectories calculated among all nodes

Figure 21: These images showcase a simulation conducted on the *Medium-many* map, highlighting all nodes selected by the algorithm along with all possible trajectories, particularly emphasizing the optimal trajectory



(a) State evolution during the Optimal Trajectory

(b) Inputs used by the model during the Optimal Trajectory

Figure 22

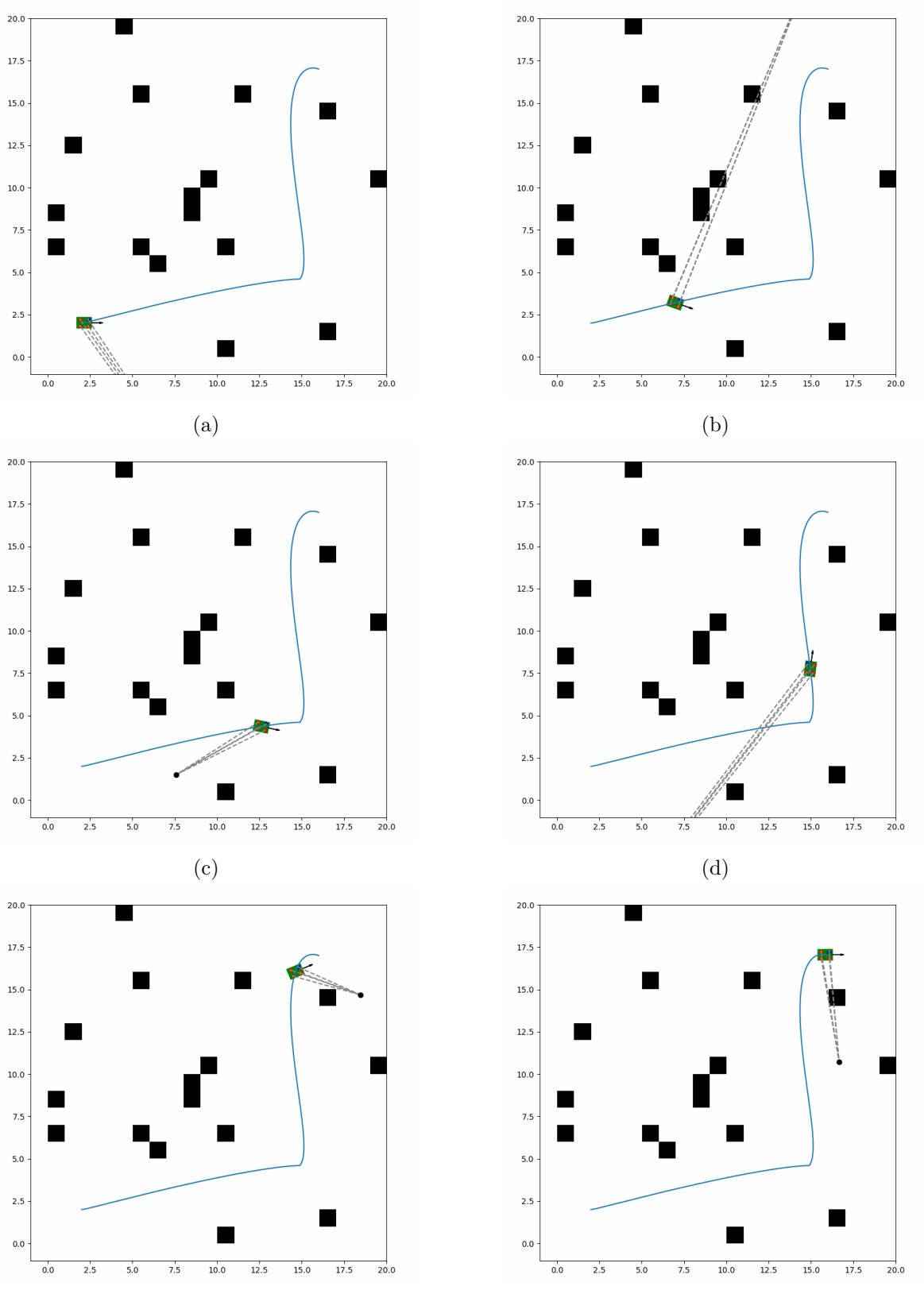
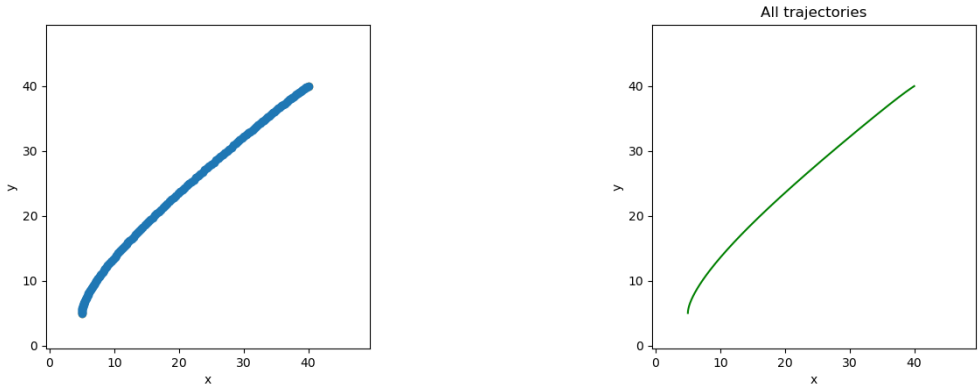


Figure 23

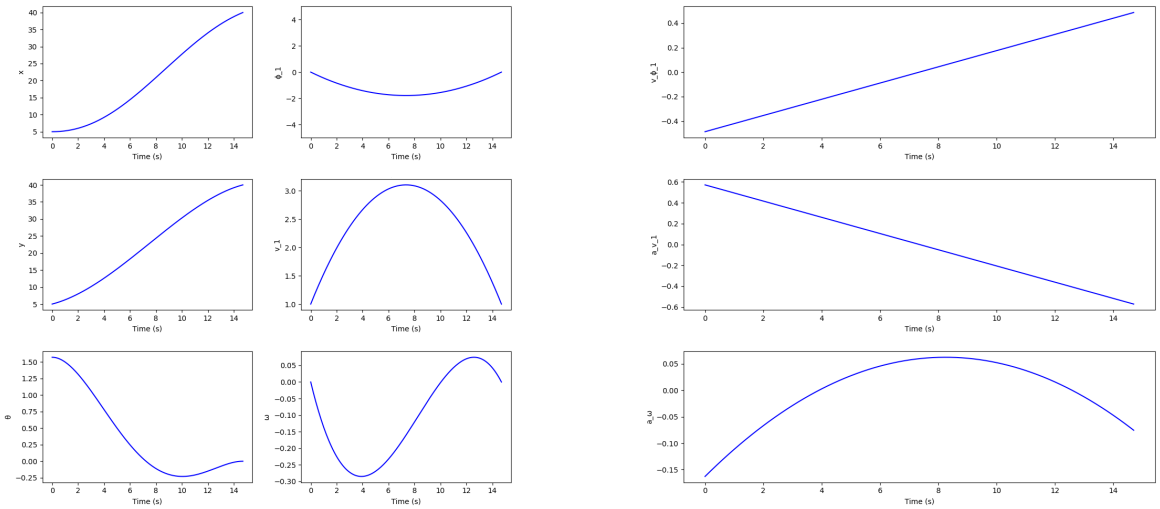
Start state = [5, 5, $\frac{\pi}{2}$, 0, 1, 0], Goal state = [40, 40, 0, 0, 1, 0].



(a) Optimal path retrieved by KRRT*

(b) All trajectories calculated among all nodes

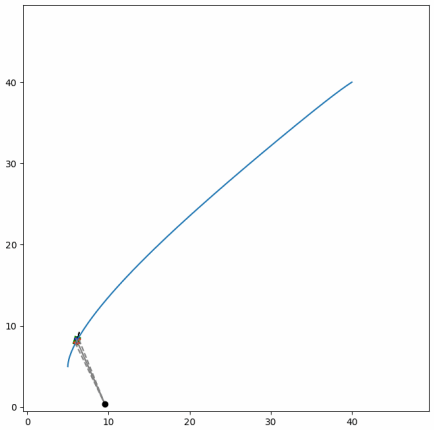
Figure 24: These images showcase a simulation conducted on the *Big-empty* map, highlighting all nodes selected by the algorithm along with all possible trajectories, particularly emphasizing the optimal trajectory



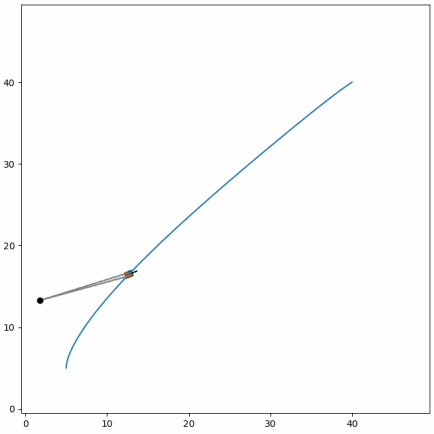
(a) State evolution during the Optimal Trajectory

(b) Inputs used by the model during the Optimal Trajectory

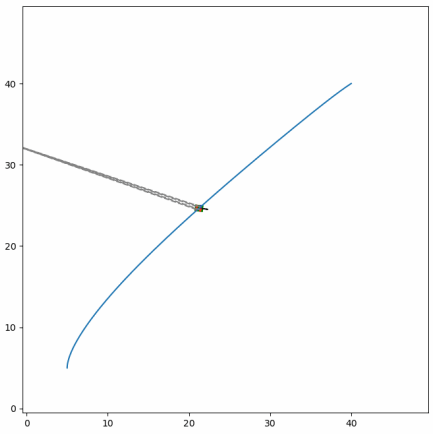
Figure 25



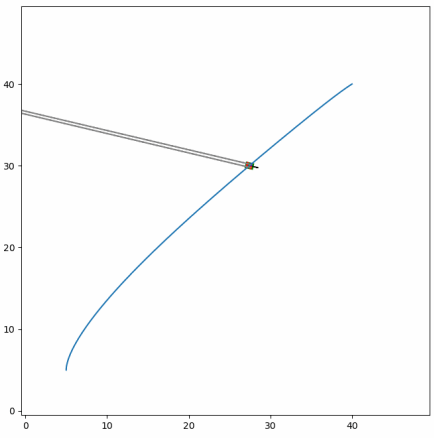
(a)



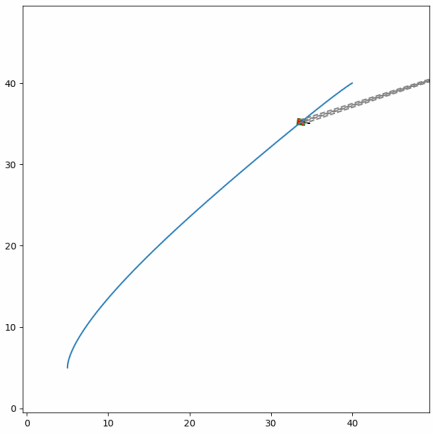
(b)



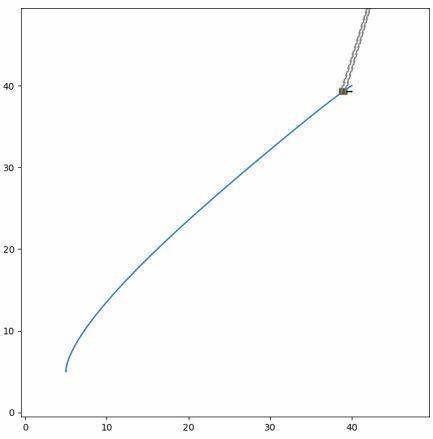
(c)



(d)



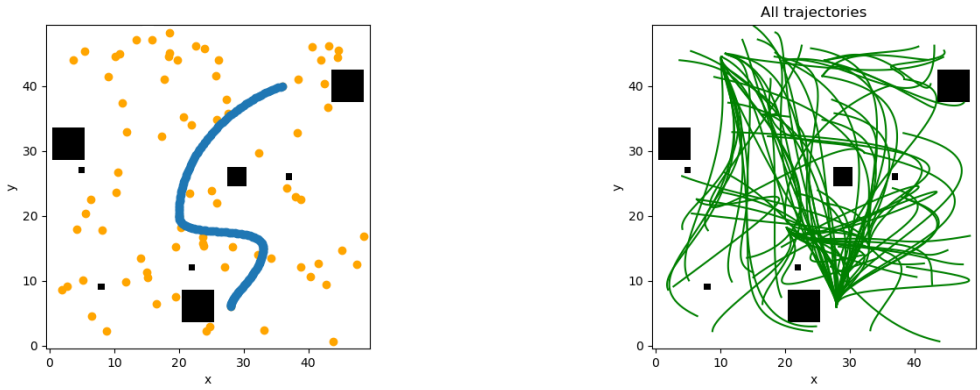
(e)



(f)

Figure 26

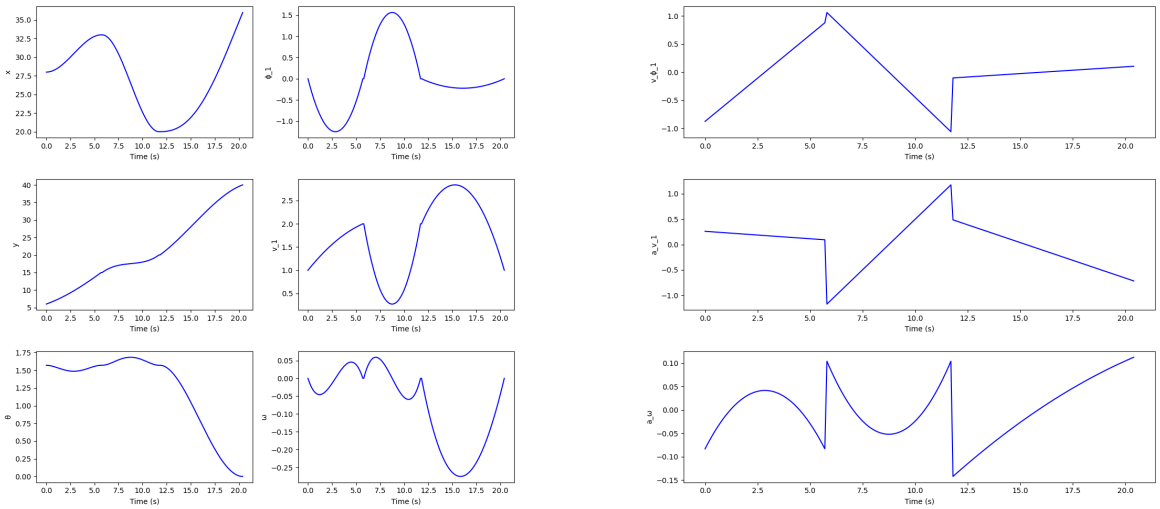
Start state = $[28,\ 6,\ \frac{\pi}{2},\ 0,\ 1,\ 0]$, Goal state = $[36,\ 40,\ 0,\ 0,\ 1,\ 0]$.



(a) Optimal path retrieved by KRRT*

(b) All trajectories calculated among all nodes

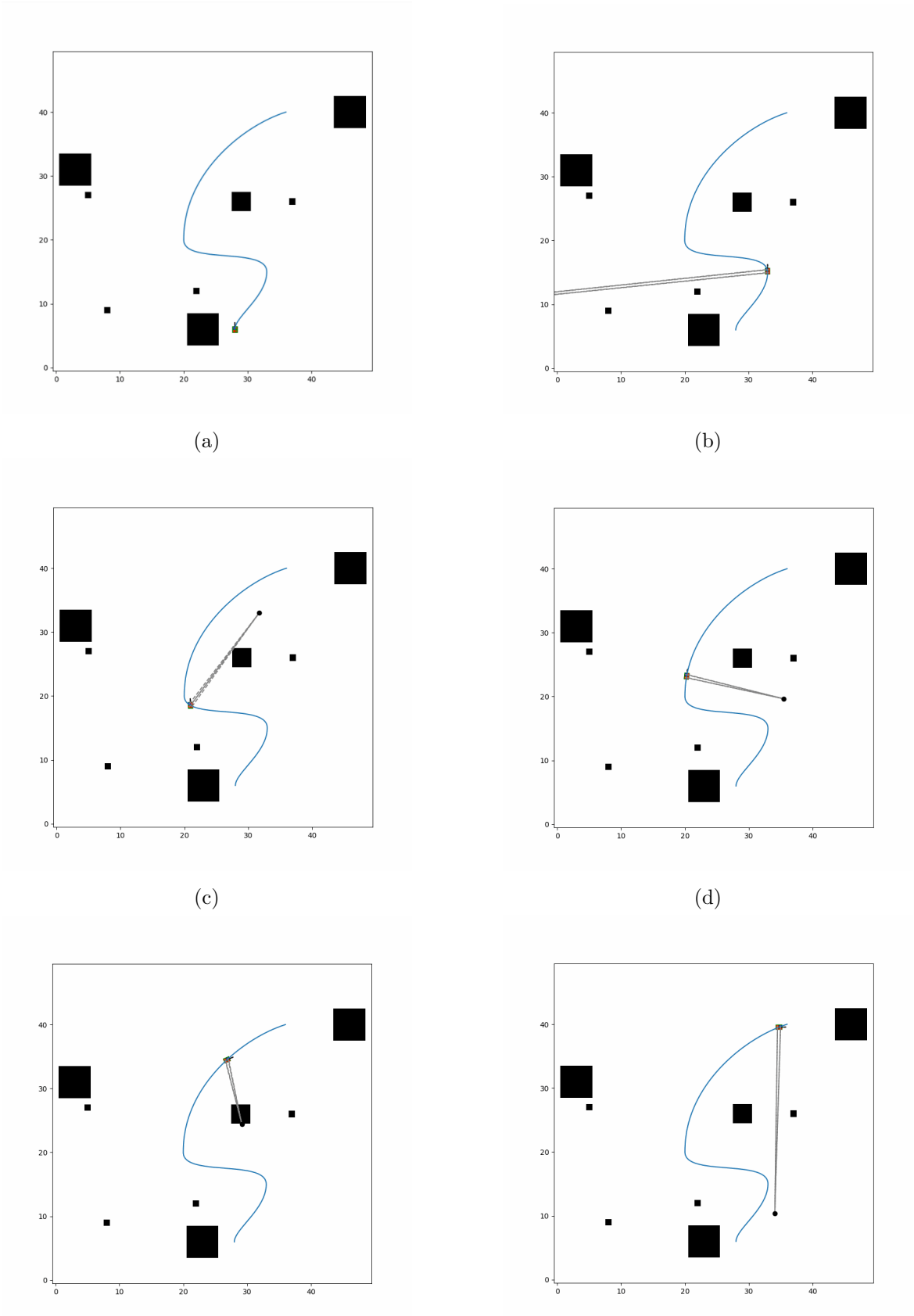
Figure 27: These images showcase a simulation conducted on the *Big-few* map, highlighting all nodes selected by the algorithm along with all possible trajectories, particularly emphasizing the optimal trajectory



(a) State evolution during the Optimal Trajectory

(b) Inputs used by the model during the Optimal Trajectory

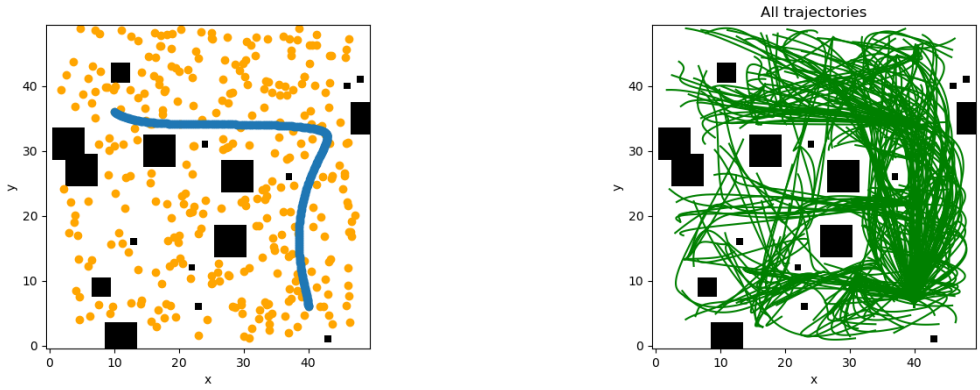
Figure 28



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

Figure 29

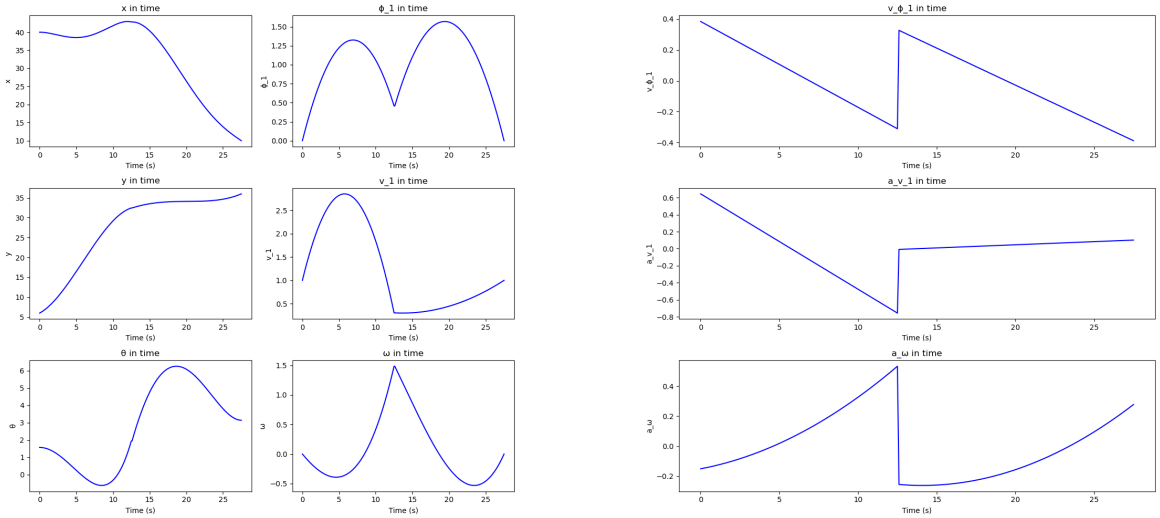
Start state = $[40,\ 6,\ \frac{\pi}{2},\ 0,\ 1,\ 0]$, Goal state = $[10,\ 36,\ \pi,\ 0,\ 1,\ 0]$.



(a) Optimal path retrieved by KRRT*

(b) All trajectories calculated among all nodes

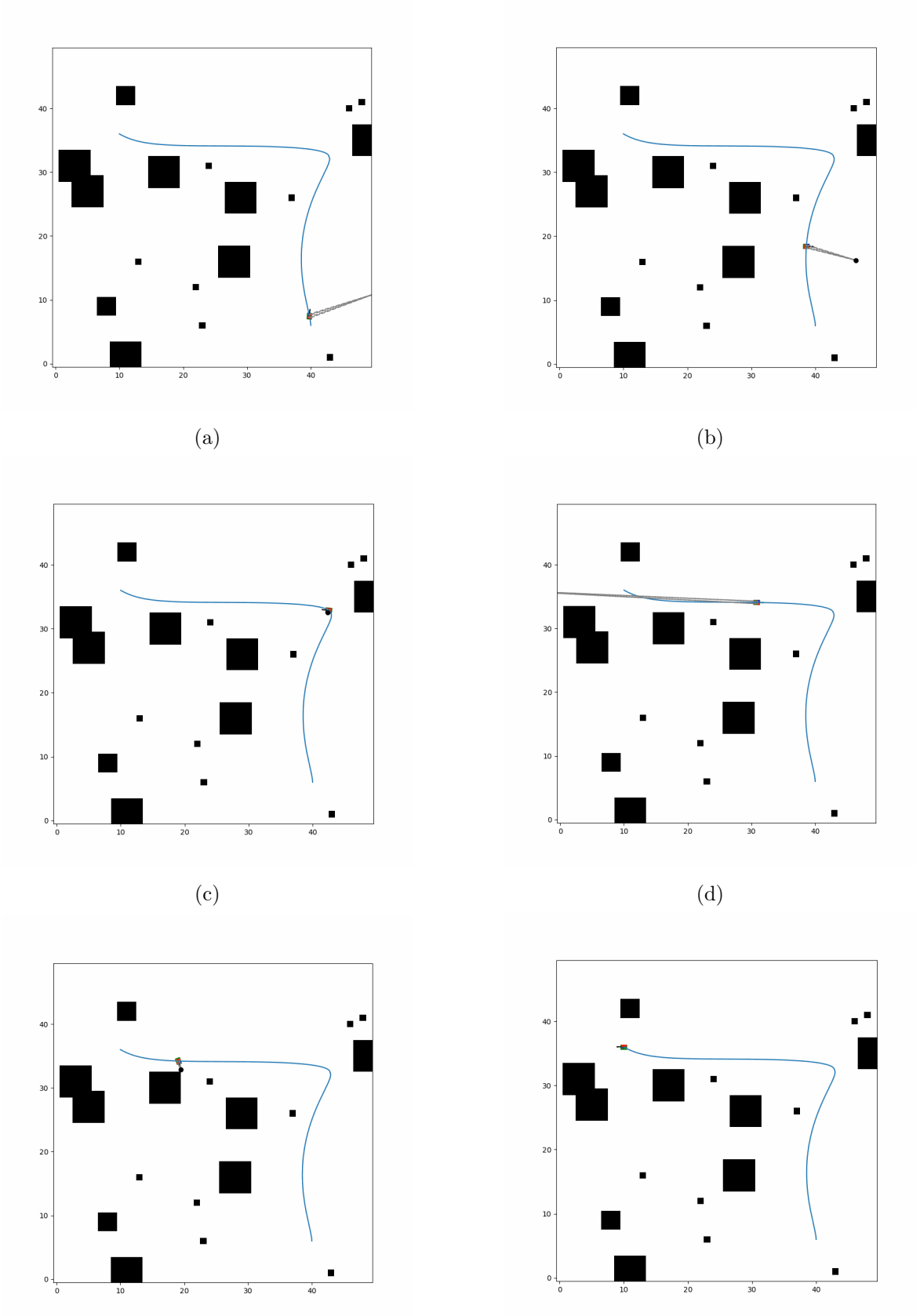
Figure 30: These images showcase a simulation conducted on the *Big-many* map, highlighting all nodes selected by the algorithm along with all possible trajectories, particularly emphasizing the optimal trajectory



(a) State evolution during the Optimal Trajectory

(b) Inputs used by the model during the Optimal Trajectory

Figure 31



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

Figure 32

5 Conclusions

In this project, we demonstrated the efficacy of the KRRT* algorithm in generating feasible trajectories for an off-centered steering robot navigating environments characterized by obstacles. The KRRT* algorithm maps out nodes that the robot must traverse to reach the goal, while producing the ideal commands for the robot to execute in order to navigate between these nodes. These commands are tailored to meet the dynamics of the off-centered steering robot, incorporating the nonholonomic constraints. The algorithm proved highly effective in finding optimal solutions quickly in scenarios where the map was clear of any obstacles. This efficiency is pivotal for real-time application in autonomous navigation. We demonstrated the potential to limit specific control inputs by adjusting the structure of the matrix $\boldsymbol{R}$. This adaptability allows the robot to follow smoother trajectories. Despite his strength, a significant limitation of this algorithm lies in the high number of iterations required to achieve an optimal solution. This aspect can impede the practical deployment in environments where rapid decision-making is crucial.

For future developments, integrating dedicated methods and data structures for selecting neighbor nodes could significantly reduce the number of iterations needed to reach the optimal path. These enhancements aim to refine the search radius for neighboring nodes and enhance the efficiency of this search process. Such advancements could significantly reduce the computational overhead and improve the real-time capabilities of the navigation system.

These conclusions not only underscore the robustness of the implemented algorithm but also highlight critical areas for improvement and future research that could further elevate the operational proficiency of autonomous robotic systems.

References

[1] Paolo Giordano, M. Fuchs, Alin Albu-Schäeffer, and G. Hirzinger. On the kinematic modeling and control of a mobile platform equipped with steering wheels and movable legs. pages 4080 – 4087, 06 2009.

[2] Dustin J. Webb and Jur van den Berg. Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*, pages 5054–5061, 2013.

[3] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. 2011.

[4] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.