2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

A model predictive control approach to motion planning in dynamic environments*

Bernhard Wullt[+,1], Per Mattsson[2], Thomas B. Schön[2], Mikael Norrlöf[1]

***Abstract*— The current state-of-the art motion planners for mobile robots typically do not consider the future movement of moving obstacles. Instead they work by rapid replanning, which makes them reactively adapt to any changes in the environment. This can result in a sub-optimal behavior, which we address in this work by proposing a predictive motion planner that integrates motion predictions into all planning steps. We demonstrate the validity of our approach by evaluating our proposed planner in a dynamic environment where the robot moves slower than the moving obstacles. We benchmark our predictive planner with a reactive planning approach and observe better performance, both in avoiding collisions and maintaining the robots position in the goal region.**

I. Introduction

Motion planning is the problem of finding a collision-free trajectory that connects a start and goal state given some obstacle representation. The traditional setting is to consider static obstacles and under this setting the problem is well-studied with many high-performing solutions, see e.g. [1], [2]. Static environments are natural for many robots, however it is not uncommon that mobile robots are used in applications where the environment is dynamic, like environments involving humans or other mobile robots. In general, the inclusion of time-varying obstacles makes the planning problem harder, since the mobile robot needs to adapt its trajectory to the movement of the obstacles, hence requiring real-time performance. The common way to plan in a dynamic environment for mobile robots is to decompose the planner into a global and a local planner [3]. The global planner finds an initial collision-free path that at most respects the kinematics of the robot. The local planner then produces a local trajectory that tracks the global path while reacting to dynamic obstacles in the vicinity of the robot. This framework works well and has proven itself as being a robust solution. However, the current state-of-the-art (SOTA) navigation stack [4] does not include predicted obstacle motion in the planning. Instead it relies on rapid replanning to maneuver past dynamic obstacles which results in sub-optimal performance.

In this work, we follow the same idea but aim to address the reactive behavior by explicitly integrating estimated obstacle motion in the planning to obtain proactive motion. Our algorithm consists of several modules. The top module predicts the future movement of the obstacles with a constant acceleration motion model. The trajectories are then used in a warm started path planner that rapidly finds a collision-free path in a spatio-temporal volume. From this path, we construct a collision-free polygonal tube which is used in a trajectory optimization step to produce a dynamically feasible trajectory. We move along this trajectory for one time step and re-use the trajectory in next iteration, repeating the process. Our contribution is a predictive motion planner for dynamic environments with the following features:

- An online obstacle motion estimator.
- A bi-directional path planner that rapidly finds a feasible collision-free spatio-temporal path.
- A robust trajectory optimization formulation.

Our motion planner is open-sourced[1] together with a simulation environment for benchmarking motion planners in dynamic environments.

II. Related Work

In modern mobile robotic systems [4] the motion planner can be decomposed into a global planner and local planner. The global planner is responsible for producing a geometric path that takes the robot to the goal region. The path is limited to at most being kinematically feasible in order to plan efficiently. Usually, the path planners are either search or sampling-based. Common search-based methods are A* [5] and its derivative methods [6], [7], [8]. Search-based methods plan in an occupancy-grid of the environment and uses a heuristic function to speed up the planning and are attractive since these methods guarantee optimality. Core algorithms within the sampling-based methods are rapidly-exploring random trees (RRT) [9] and probabilistic road maps (PRM) [10]. RRT expands a tree from the start configuration until it reaches the goal region, while PRM builds a graph representation of the collision-free region by sampling. The convergence rate of RRT is significantly improved by bi-directional planning [11], where two trees are grown consecutively from the start and goal configurations and tested after each iteration if they can be connected. RRT has no optimality guarantees, which is something that RRT* [12] addresses by rewiring neighbouring states in the tree, such that in the limit, the optimal path is found. Spatio-temporal planning with RRT is done in [13], which introduces ST-RRT* for problems with

*This research was supported by the *Wallenberg AI, Autonomous Systems and Software Program (WASP)* funded by Knut and Alice Wallenberg Foundation.

+ Corresponding author
[1] ABB Robotics, Sweden, E-mail: name.surname@se.abb.com
[2] Department of Information Technology, Uppsala University. E-mail: name.surname@it.uu.se

[1] https://github.com/whiterabbitfollow/mpc_mp_dyn_env

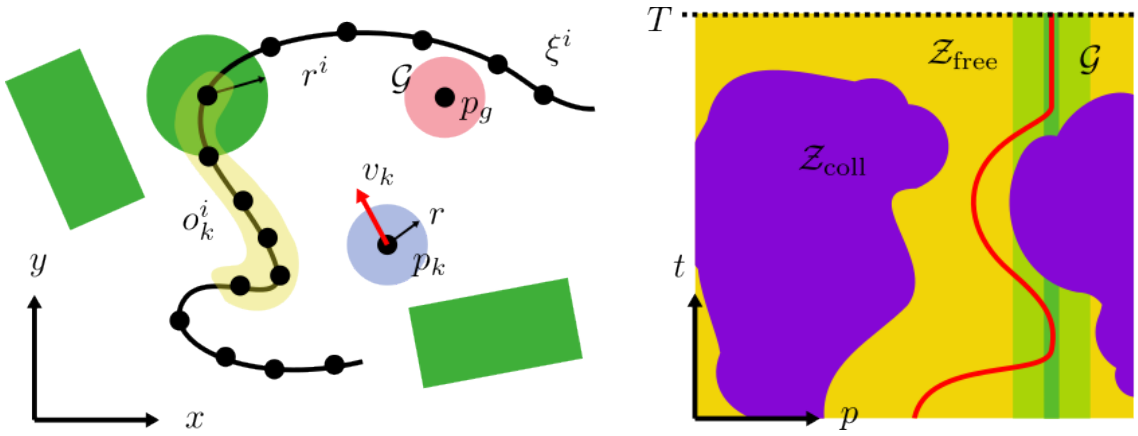

Fig. 1. (Left) The robot is represented by the blue circle with position, p_k, velocity, v_k and radius r. It wants to navigate to the goal region, $\mathcal{G}$, represented by the transparent red circle. The robot is surrounded by static and dynamic obstacles, here represented by green boxes and circles, respectively. The dynamic obstacles follows a trajectory, ξ^i and at each time step, the robot observes the last N_{o} positions, o_k^i, of the obstacle. (Right) A conceptual illustration of the problem formulation in the spatio-temporal space $\mathcal{Z}$ for a one DOF robot. The yellow region represents the set of collision free states, $\mathcal{Z}_{\text{free}}$, while the purple regions are the states when the robot is in collision, $\mathcal{Z}_{\text{coll}}$. The problem is to find a policy π, that produces a trajectory (red curve) that is collision-free and at the same time spends as much time as possible within the goal region $\mathcal{G}$ while respecting the dynamics.

unknown arrival time. Their planner is bi-directional, but since the arrival time is unknown, they instead grow a forest of goal trees along the time dimension. The arrival time is iteratively lowered by upper bounding the time dimension to the best arrival time found so far. Our global planner is similar to ST-RRT*, since we both plan bi-directionally in space-time, but we have different use cases. We plan with a fixed time horizon and with a goal tree rooted along the space-dimension, compared to ST-RRT* which has a forest rooted along the time-dimension. Furthermore, we make different assumptions on how to deal with the dynamic obstacles. In [13] the dynamic obstacles are consider in the whole time interval, while we assume that the time-varying obstacles disappear after a certain time. This allows us to speed up the online planning by warm starting the planner with a pre-expanded goal tree in the static time interval.

The task of the local planner is to track the global plan as closely as possible, while respecting the kinematics and dynamics of the robot, and also optimizing for other objectives like clearance from obstacles, path smoothness etc. A classical and well established local planner is the Dynamic Window Approach (DWA) [14], which searches over a discretized space of velocities to find the most optimal one-step velocity action. This is done in a heuristic way by first filtering out dynamically infeasible velocities and then simulate the remaining velocity actions over a certain time horizon with constant control signal. The resulting trajectories are scored based on an objective function that weights different metrics, like distance to obstacles, distance to the goal etc. Finally, the most optimal action is selected and applied to the system. DWA is simple and works well in many scenarios, but it produces sub-optimal trajectories, due to its simple search approach. To address this, modern local planners [15], [16] include more temporal information in the planning, following the model predictive control (MPC) framework. The idea behind MPC [17] is to use a dynamic model of the system being controlled to predict how it evolves in a finite time horizon and pick the control actions that leads to the best outcome according to some performance metric. The system is then actuated with the first control action for one time step, and then the process is repeated to compensate for prediction errors. Time elastic band (TEB) [15] follows this approach and formulates the trajectory optimization problem as a non-linear program, optimizing the trajectory to minimize time to reach a local via-point from the global plan while considering kino-dynamic and obstacle avoidance constraints. The minimal time formulation works well for reactive scenarios, where obstacle motion is not considered. However, for spatio-temporal planning, where the motion of the obstacles is integrated in the planning, it is not suitable to optimize for minimal time, since obstacles can appear in neighboring time steps. Model predictive path integral (MPPI) [16] instead formulates the planning as a stochastic trajectory optimization problem, relying on Monte-Carlo sampling to rapidly find a trajectory. To speed up the extensive sampling, GPUs are leveraged, which of course limits MPPI to systems where this is available.

A relevant motion planning framework outside of the mobile robotics domain is FaSTrack [18], which decouples the planning into a path planner and a tracking controller. The framework provides a safe tracking controller, that is computed offline by formulating the tracking problem as a pursuit-evasion game which is solved using Hamiltonian-Jacobi reachability analyis. The framework makes no assumption on what kind of planner is used, it only requires that the planner should run fast. Our approach is similar to FaSTrack, in that we use a rapid path planner. However, while FasTrack tracks the path found by the planner, we only use the path to construct a convex safe-region which

we optimize a trajectory to lie within.

The method that is closest to our approach is CIAO* [19], a further generalized version of CIAO [1], reformulated for dynamic environments. CIAO* includes a pre-processing step before the optimization, where the collision-free space is maximized by growing spheres around a given reference path. Collision avoidance is then obtained by constraining the trajectory to be within the spheres with some added margin for obstacle clearance. Even though CIAO* was formulated for dynamic environments, the authors evaluated their proposed method in a static setting, which makes it unclear how well it works for dynamic environments. CIAO* is also limited to trajectory optimization, while we suggest a complete motion planning solution. Our proposed trajectory optimization does not include the CIAO* pre-processing step, instead we rely on a polygonal representation of the collision-free space.

Obstacle motion prediction is a rich field in dynamic motion planning with many existing solutions for different scenarios [20]. In [21], a Gaussian process is used to predict the future movement of humans, taking into account potential goals the human is moving towards. In our work, we do not limit ourself humans, we simply make the assumtion that the obstacles move with constant acceleration, which is not a novel motion model in itself. However, our idea to disregard dynamic obstacles in the planning after a certain time period and exploit this in the planning by warmstarting is, to the best of our knowledge, novel.

III. Problem formulation

We define the robot as a single body 2 DOF robot with state, $x_k = [p^\top, v^\top]^\top \in \mathcal{X}$, defined by a position, $p \in \mathcal{W}$, in the world space, $\mathcal{W} \subset \mathbb{R}^2$, and a velocity $v \in \mathbb{R}^2$ at time t_k. The control action is denoted by $u_k \in \mathcal{U}$, with action space, $\mathcal{U} \subset \mathbb{R}^2$. The system evolves in discrete time with sample time Δ and transitions according to

$$x_{k+1} = f(x_k, u_k), \tag{1}$$

where $f : \mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$ denotes the state-transition function. The robots geometry can be described by a circle with radius r centered at the current position of the robot. The robot is surrounded by, N_{obs} obstacles, which can be divided into N_{dyn} dynamic obstacles and N_{sta} static obstacles. All static obstacles are assumed to have convex geometries. The i-th dynamic obstacle follows a trajectory, $\xi^i : \mathbb{R} \mapsto \mathcal{W}$ and its geometry can be represented by a circle with radius, r^i. The robot has no knowledge of the future movement of the obstacles, instead at time step k it is given the last N_{o} positions of each dynamic obstacle, $o_k^i = [p_{k-(N_{\text{o}}-1)}, \dots, p_k]^\top \in \mathbb{R}^{N_{\text{o}} \times 2}$, as observation, $o_k = [o_k^1, \dots, o_k^{N_{\text{dyn}}}] \in \mathcal{O}$, where $\mathcal{O}$ is the observation space. The robot has to navigate to a goal region, $\mathcal{G} \subset \mathcal{W}$, defined from a goal position, p_{g}, and stay there as long time as possible. An illustration of the problem scenario is presented in the left part of Figure 1.

We define a spatio-temporal volume as $\mathcal{Z} = \mathcal{W} \times [0, T]$, where T defines the total execution time of the problem. The set of all collision-free states is denoted by $\mathcal{Z}_{\text{free}} \subseteq \mathcal{Z}$ and the set of states that are in collision is denoted by $\mathcal{Z}_{\text{coll}} = \mathcal{Z} \setminus \mathcal{Z}_{\text{free}}$.

We now formulate our problem. Given a start state, x_0, a goal region, $\mathcal{G}$, and an execution time, T, we want to find a policy $\pi : \mathcal{X} \times \mathcal{O} \mapsto \mathcal{U}$, that solves the following problem:

$$\max_\pi \quad \sum_{t_k \le T} \mathbf{1}_{\mathcal{G}}(p_k), \tag{2a}$$

$$\text{s.t} \quad (p_k, v_k) = x_k, \tag{2b}$$

$$u_k = \pi(x_k, o_k), \tag{2c}$$

$$x_{k+1} = f(x_k, u_k), \tag{2d}$$

$$(p_{k+1}, t_{k+1}) \in \mathcal{Z}_{\text{free}}. \tag{2e}$$

In the objective, (2a), $\mathbf{1}_{\mathcal{G}}(p) : \mathcal{W} \mapsto \{0, 1\}$, is the indicator function. Hence, the problem is to stay in the goal region for as long time as possible while at the same time avoid collisions. The problem formulation is illustrated in Figure 1 (right).

IV. Method

We propose to approach the problem by integrating planning with obstacle motion prediction in an MPC fashion. Hence, we start by predicting the obstacle motion from the observations. Next, we find a feasible spatio-temporal path that is collision-free w.r.t. the predicted motion, from which we then construct a collision-free spatio-temporal region and find an optimal trajectory constrained within this region. We assume that our model predictions are only valid for a short time horizon, hence we follow our trajectory only during one time step and then repeat the process again to compensate for prediction errors. The following sections presents the steps mentioned in the sequence that they occur in. We end the section by assembling the final solution.

A. Obstacle motion prediction

We predict the future movement of the obstacles by assuming that the obstacles move according to a second order polynomial, which has the following form in one dimension,

$$p(t) = \theta_0 + \theta_1 t + \theta_2 t^2, \tag{3}$$

where θ_0, θ_1 and θ_2 denotes the model parameters. The assumed motion model is simple and reasonable for short-horizon predictions, but not accurate for long-horizon predictions. However, we compensate for prediction errors by updating our estimates each time step. The parameters of our motion model is computed by least squares, which results in parameter matrix, $\Theta \in \mathbb{R}^{3 \times 2N_{\text{dyn}}}$.

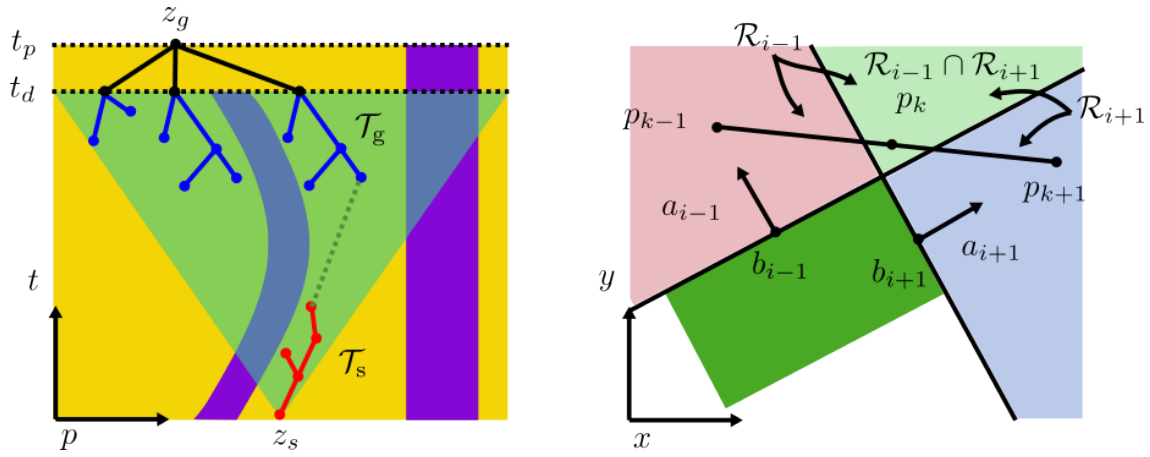

Fig. 2. (Left) Conceptual illustration of the path planning for a 1 DOF problem. The start tree, $\mathcal{T}_\mathrm{s}$, is grown from z_s and drawn in red. The goal tree, $\mathcal{T}_\mathrm{g}$, is expanded in two stages. The first stage is a pre-processing step, where we expand it offline in the static environment, drawn in black. At the next stage, which is the online planning stage, the tree is expanded from all the static nodes, which are initialized at time t_d, drawn in blue. We use the conditional sampling method from [13], by sampling states that are within a velocity cone, illustrated as a green cone, from the start state. The trees are consecutively grown and we check at each iteration if the trees can be connected. Once the trees can be connected, dashed green line, we report back the found path. (Right) Illustrating the idea how to avoid paths that tunnels through obstacles. The i:th time step needs to be in the intersection of $\mathcal{R}_{i-1}$, represented as red and green regions, and $\mathcal{R}_{i+1}$, represented as blue and green regions.

B. Spatio-temportal path planning

The path planner is supposed to find a collision-free path that connects the start z_s and goal state, z_g. At its core, the planner is a bi-directional RRT planner [11], but adapted for spatio-temporal planning and integrated with warm starting. We illustrate the planner in Figure 2 (left). The time interval for the planning is $[0, t_\mathrm{p}]$, where $t_\mathrm{p} = N_\mathrm{p}\Delta$ is the planning horizon, hence we plan N_p time steps in the future. We make the assumption that N_p is choosen such that we always can reach the goal within the time intervall. We divide the planning interval into two different intervals, one dynamic time interval, $[0, t_\mathrm{d}]$, where $t_\mathrm{d} = N_\mathrm{d}\Delta$, thus including N_d time steps, and one static interval, $(t_\mathrm{d}, t_\mathrm{p}]$. Within the dynamic time interval, we consider the motion of the obstacles in the planning. In the static time interval, the dynamic obstacles are disregarded, since our motion predictions are not valid when extrapolated too far into the future. Note that this is true for any prediction scheme that does not add any extra assumptions about how the dynamical obstacles move. This assumption allows us to warm start our goal tree before the planning takes place. If we alternatively assumed that the dynamic obstacles would keep their final predicted position, then this could in some cases result in that the dynamic obstacles would be positioned in the goal region, making it impossible to find a path to the goal. Furthermore, it would not be possible to warmstart the search, due to the introduction of new obstacles. Having motivated the removal of the dynamic obstacles, we next discuss the warmstarting, which is done by growing a tree from the goal position in the static environment, disregarding the temporal dimension, and use the rewiring method from [12] such that all the paths to the goal vertex are optimal. At run-time, we plan bi-directionally within the dynamic time interval according to Algorithm 1, which we now explain in more detail. The start tree, $\mathcal{T}_\mathrm{s}$, is initialized with root at $z_\mathrm{s} = (p_\mathrm{s}, 0)$. The goal tree, $\mathcal{T}_\mathrm{g}$, is initialized with vertices, $\mathcal{V}_\mathrm{ws}$, and edges, $\mathcal{E}_\mathrm{ws}$, from the goal tree grown in the static environment, discussed previously. Thus, the goal tree has multiple roots located at time t_d. We expand the trees in a consecutive fashion, selecting one tree for expansion, $\mathcal{T}_\mathrm{a}$, by executing STRRTGrowTree, which grows the tree according to the RRT algorithm, adapted for spatio-temporal planning with some of the techniques from [13]. We present the algorithm for this in Appendix I. If $\mathcal{T}_\mathrm{a}$ has been expanded, then we try to connect the trees, Line 10, and if successful, then we report back the found path, discretized into m states, $Z \in \mathbb{R}^{m\times 3}$, Line 12. If the connection was not successful, we swap trees, Line 15, and continue to iterate for max N_max iterations.

Algorithm 1 Bi-directional RRT planner in space-time.

```
1:  function STRRTConnect(Θ_k, p_s)
2:      T_s = ({}, {})
3:      T_g = (V_ws, E_ws)
4:      z_s = (p_s, 0)
5:      addVertex(T_s, z_s)
6:      T_a, T_b = T_s, T_g
7:      for i ∈ [1, ..., N_max] do
8:          z_new = STRRTGrowTree(T_a | Θ_k)
9:          if z_new not NULL then
10:             status = ConnectTrees(T_a, T_b, z_new | Θ_k)
11:             if status is connected then
12:                 return ReturnPath(z_new, T_a, T_b)
13:             end if
14:         end if
15:         Swap(T_a, T_b)
16:     end for
17:     return NULL
18: end function
```

C. Trajectory optimization

We use optimization to produce a sequence of optimal control actions, $U \in \mathbb{R}^{N_\text{p} \times 2}$, and states, $X = [P, V] \in \mathbb{R}^{N_\text{p}+1 \times 4}$. As input we are given a spatio-temporal path, Z, containing m states. For each space-time state in the path, $(\tilde{p}_i, t_i) \in Z$ and each obstacle j, we compute the closest point, $b_{ij} \in \mathbb{R}^2$, and boundary normal, $a_{ij} \in \mathbb{R}^2$. This gives us a linear approximation to the distance of the j:th obstacle and the boundary of the robot, $g_{ij}(p) = a_{ij}^\top (p - b_{ij}) - r$. Since all obstacles are convex, we can construct a collision-free region by combining all distance functions to form a polygon region with positive distance, $\mathcal{R}_i = \{p \mid g_{ij}(p) \geq 0,\ \forall j \in \{1, \ldots, N_\text{obs}\}\}$. Next, we define an index tuple as follows

$$\mathcal{I}_k = \begin{cases} (\lceil \frac{k}{2} \rceil) & \text{if } k \text{ odd} \\ (\frac{k}{2}, \frac{k}{2}+1) & \text{otherwise} \end{cases}, \tag{4}$$

which results in the sequence, $\mathcal{I} = ((1), (1,2), (2), \ldots, (m-1, m), (m))$, when $k = 1, \ldots, N_\text{p}+1$. We constrain the positions in the trajectory to lie within the following regions

$$p_k \in \bigcap_{i \in \mathcal{I}_k} \mathcal{R}_i,\ k = 1, \ldots,\ N_\text{p}+1, \tag{5}$$

which is a convex constraint. It stops the solver from producing paths that tunnel through obstacles in between the discretized points, illustrated in Figure 2 (right). To be robust to prediction errors in our motion estimation, we penalize the N_d positions from the dynamic time interval which are within a distance margin d_dyn of the N_dyn dynamic obstacles. We do this by taking the minimum over all distance functions computed from the dynamic obstacles for time step k and the distance margin, this results in

$$\begin{aligned} \sigma_k^\text{dyn}(p_k) = -\min(g_{i1}(p_k),\ \ldots,\ g_{iN_\text{dyn}}(p_k), d_\text{dyn}), \\ k = 1, \ldots, N_\text{d},\ \forall i \in \mathcal{I}_k, \end{aligned} \tag{6}$$

which is a convex function since we negate the min operator. To gain general obstacle clearance from the N_sta static obstacles, we repeat the process but now for all time steps and with a static distance margin, d_sta, resulting in

$$\begin{aligned} \sigma_k^\text{sta}(p) = -\min(g_{i1}(p),\ \ldots,\ g_{iN_\text{sta}}(p), d_\text{sta}), \\ k = 1, \ldots, N_\text{p}+1,\ \forall i \in \mathcal{I}_k. \end{aligned} \tag{7}$$

Having defined our collision-free regions and distance functions, we define our optimization problem as

$$\min_{X,\,U} \quad \sum_{i=1}^{3} h_i(P) + h_4(U) \tag{8a}$$

$$\text{s.t} \quad x_1 = x_\text{s}, \tag{8b}$$

$$x_{N_\text{p}+1} = x_\text{g}, \tag{8c}$$

$$x_{k+1} = f(x_k, u_k), \qquad k = 1,\ \ldots,\ N_\text{p}, \tag{8d}$$

$$u_k \in \mathcal{U}, \qquad k = 1,\ \ldots,\ N_\text{p}, \tag{8e}$$

$$p_k \in \bigcap_{i \in \mathcal{I}_k} \mathcal{R}_i, \qquad k = 1, \ldots,\ N_\text{p}+1, \tag{8f}$$

where $x_\text{g} = [p_\text{g}^\top, 0_2^\top]^\top$. Our objective functions are defined as

$$\begin{aligned} h_1(P) = \sum_{i=1}^{N_\text{p}+1} \alpha_i ||p_i - p_\text{g}||_2,\ h_2(P) = \sum_{i=1}^{N_\text{d}} \beta_i \sigma_i^\text{dyn}(p_i) \\ h_3(P) = \sum_{i=1}^{N_\text{p}+1} \omega_i \sigma_i^\text{sta}(p_i),\ h_4(U) = \sum_{i=1}^{N_\text{p}} ||u_i||_2. \end{aligned} \tag{9}$$

We inspect our objectives in more detail. The first objective, h_1, is added to obtain paths which are within the goal region as much as possible. Hence, it is needed to fulfill the objective specified in the problem formulation, (2a), which is to stay in the goal region as long as possible. It is weighted with increasing values for increasing time steps, $\alpha_i \leq \alpha_{i+1}$. The second objective and third objective, h_2 and h_3 are for obstacle clearance. The obstacle clearance weights are decreasing with increasing time steps. Thus, $\beta_i \geq \beta_{i+1}$, and $\omega_i \geq \omega_{i+1}$. The last term, h_4, is added to penalize large control actions.

Our proposed optimization problem is in its most general form a non-linear program. However, if the dynamics, (8d), is linear and the actuator constraint, (8e), is convex, then the optimization problem becomes a convex optimization problem.

D. Assembly

Having introduced all steps in the process, we now present the whole process in Algorithm 2.

Algorithm 2 Outline of the complete planning process.

Require: x_0, $\mathcal{G}$, T

1: $k = 0$
2: Z_0 = NULL
3: Warm start $\mathcal{T}_\text{g}$ in static environment from $\mathcal{G}$
4: **while** $t_k \leq T$ **do**
5: o_k = ObserveObstacleMotion()
6: Compute Θ_k by least square from observation o_k
7: **if** Z_k is NULL or IsCollision(Θ_k, Z_k) **then**
8: Z_k = STRRTConnect(Θ_k, p_k)
9: **end if**
10: X^*, U^* = Solve optimization problem, Eq. (8)
11: Apply first control action u_1^*
12: Z_{k+1} = Overwrite with positions from X^*
13: $k = k + 1$
14: **end while**

Before the planning is started, we warm start the goal tree $\mathcal{T}_\text{g}$, Line 3. At run-time, we first observe the environment and then estimate the future motion, Line 6. We check if our current path, Z_k, is valid w.r.t. our most recent motion predictions. If this is not the case or the path has not been initialized, then we query the path planner. Otherwise, we re-use our current path. We solve the optimization problem, Line 10, and apply the first control action. We shift the

optimized path and reuse it for next iteration, repeating the process according to the MPC procedure.

V. Experiments

In order to show the validity of our proposed algorithm, a custom environment was developed that randomly generates obstacles and their corresponding trajectories. An example environment instance is illustrated in Figure 3. The static obstacles are modeled as boxes with randomized dimensions and orientations while the dynamic obstacles are modeled as circles. The number of dynamic obstacles in the environment ranges between one and three. We define the world space as $\mathcal{W} = [-1, 1] \times [-1, 1]$. We limit the actuation of the robot so that its dynamics is slower than the moving obstacles. We define the action space as $\mathcal{U} = \{u \in \mathbb{R}^2 \mid ||u_k||_2 \leq u_{\max}\}$ where $u_{\max} = 0.01$. We set the sample time to $\Delta = 1\ s$. The robot dynamics is described as a discrete-time double integrator dynamics. Thus, we have a system with following form

$$x_{k+1} = Ax_k + Bu_k, A = \begin{bmatrix} I_2, & I_2 \\ 0_{2,2}, & I_2 \end{bmatrix}, \ B = \begin{bmatrix} I_2 \cdot \frac{1}{2} \\ I_2 \end{bmatrix} \quad (10)$$

where I_2 is the 2×2 identity matrix and $0_{2,2}$ is a 2×2 matrix with zeros. Since we have linear dynamics and our action space is convex, our trajectory optimization problem, (8), becomes a convex optimization problem.

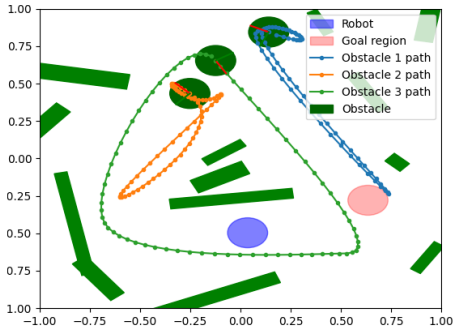

Fig. 3. Sample instance of simulation environment with three dynamic obstacles, illustrated as green circles with corresponding trajectory. Robot is illustrated as the blue circle and the goal region is represented by the red circle.

A problem instance is generated by placing the robot at a random collision-free position within the environment and randomizing a collision-free goal position, p_g, from which we define the goal region as

$$\mathcal{G} = \{p \mid p \in \mathcal{W},\ ||p - p_g||_2 \leq 0.1\}, \quad (11)$$

If the robot collides with any obstacle then the problem instance is defined as a failure and is aborted. If the solver is not able to find a solution, then we introduce slacks to the distance inequalities, (8f), to relax the problem.

To show the importance of integrating motion predictions in the planning, we benchmark our approach with reactive planning, where we run the same suggested planning steps, but assume that the dynamic obstacles are static during the dynamic time interval and then disappears afterwards. As an upper bound to the problem, we evaluate the performance of our motion planner if we plan with the exact obstacle trajectories. We run a grid search to select the best parameters for all three cases, which is described in Appendix II. The performance is evaluated based on two metrics, success rate and goal region rate. The former metric is defined as the fraction of problem instances that were executed without collision, hence fulfilling (2e). The latter metric is defined as fraction of time spent in the goal region, which is our objective, (2a). Once the best performing parameters are found, we continue with an extensive evaluation where we test the planners on 100 problem scenarios, executing each scenario for 200 time steps, $T = 200\ s$.

VI. Results and Discussion

The best parameters found from the grid search are listed in Appendix II. We present the success rates and goal region rates in Table I. By inspecting the results, we see that our suggested planner is not performing with perfect success rate, which is reasonable in this environment, since the robot is deliberately limited to be slower than the obstacles, hence in some cases collisions become unavoidable due to estimation errors in the motion predictions. However, we get both a higher success and goal region rate compared to the reactive planner, showing that it pays off to plan proactively. If we compare our planner with the case where we use the exact obstacle trajectories, we see that we could achieve a higher success and goal region rate if we improved our motion predictions.

TABLE I

Statistics over success rate and fraction of time spent in the goal region. An asterisk as suffix indicates that the exact obstacle motion where used in the planning.

	Success rate	Goal region rate
Reactive	69 %	24 %
Predictive	83 %	36 %
Predictive*	100 %	42 %

We continue by presenting statistics over computation times for the path planning and trajectory optimization steps in Table II. In general, the trajectory optimization is what takes the longest time in the process. This is due to the fact that we in some cases have to iteratively increase the relaxation, which results in a large max computation time. However, the proposed solution is naively implemented in Python, thus it should be possible to lower the times with a more optimized implementation.

TABLE II

Statistics over computation times in collision-free cases.

	Path planner	Trajectory optimization
mean [s]	0.02	0.37
max [s]	0.13	2.26

VII. Conclusion and Future Work

We have proposed a new predictive motion planning algorithm for dynamic environment. It consists of three steps, obstacle motion prediction, which is done by least squares, spatio-temporal path planning, which we do with a warm started RRT planner, and finally trajectory optimization. We showed the validity of our approach by evaluating our motion planner in a dynamic environment and benchmarked our planner with reactive planning. We observed higher collision-free rate compared to the reactive case, 83 % versus 69 %, and spent more time steps in the goal region, 36 % versus 24 %.

For future work we want to expand our motion planner to multi-body robots. This will require improvements of our path planner and the procedure to find collision-free regions, which becomes challenging when the planning in done in a high-dimensional space and should be executed online. Moreover, we aim to enhance our motion estimation by incorporating additional constraints into our predictions, such as ensuring that obstacles do not intersect with one another.

References

[1] T. Schoels, L. Palmieri, K. O. Arras, and M. Diehl, "An NMPC approach using convex inner approximations for online motion planning with guaranteed collision avoidance," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[2] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[3] S. Macenski, T. Moore, D. V. Lu, A. Merzlyakov, and M. Ferguson, "From the desks of ROS maintainers: A survey of modern & capable mobile robotics algorithms in the robot operating system 2," *Robotics and Autonomous Systems*, vol. 168, p. 104493, 2023.

[4] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The marathon 2: A navigation system," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[6] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 1994.

[7] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning A*," *Artificial Intelligence*, vol. 155, no. 1, pp. 93–146, 2004.

[8] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[9] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[10] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[11] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.

[12] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[13] F. Grothe, V. N. Hartmann, A. Orthey, and M. Toussaint, "ST-RRT*: Asymptotically-optimal bidirectional motion planning through space-time," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[14] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[15] C. Rösmann, F. Hoffmann, and T. Bertram, "Kinodynamic trajectory optimization and control for car-like robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[16] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[17] S. V. Rakovic and W. S. Levine, "Handbook of model predictive control," 2018.

[18] S. L. Herbert, M. Chen, S. Han, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Fastrack: A modular framework for fast and guaranteed safe motion planning," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 1517–1522, IEEE, 2017.

[19] T. Schoels, P. Rutquist, L. Palmieri, A. Zanelli, K. O. Arras, and M. Diehl, "CIAO*: MPC-based safe motion planning in predictable dynamic environments," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6555–6562, 2020.

[20] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 895–935, 2020.

[21] M. N. Finean, L. Petrović, W. Merkt, I. Marković, and I. Havoutis, "Motion planning in dynamic environments using context-aware human trajectory prediction," *Robotics and Autonomous Systems*, vol. 166, p. 104450, 2023.

Appendix I

The tree expansion is presented in Algorithm 3. In the tree expansion, we start by sampling a collision-free state, z_{sample}, within the dynamic time interval, Line 2, which is done by the conditional sampling technique from [13]. Concretely, this means that we only sample states, z_{sample}, which can be reached from the start state z_{s} with constant speed,

$$s_{\text{sample}} = ||p_{\text{sample}} - p_{\text{s}}||_2/(t_{\text{sample}} - t_{\text{s}}), \tag{12}$$

where $s_{\text{sample}} \in [0, s_{\text{max}}]$. The set of reachable states results in a cone in space-time with apex at the start state, illustrated in the left part of Figure 2. If the sampled state is collision-free, then it is accepted, and we find the nearest state in the tree, Line 3. In a standard RRT implementation we would return the node in the tree that has the smallest distance according to some norm. In our case however, we first need to select a set of states, $\mathcal{Z}_{\text{feasible}}$, which are temporally-coherent and are reachable from the sampled state. We use the same masking as in [13], which is if the tree is the start tree, $\mathcal{T}_{\text{s}}$, we define the set of feasible states as

$$\begin{aligned}\mathcal{Z}_{\text{feasible}} = \{(p,t) \in \mathcal{T}_{\text{s}} \mid t < t_{\text{sample}}, \\ ||p_{\text{sample}} - p||_2/(t_{\text{sample}} - t) \leq s_{\text{max}}\}.\end{aligned} \tag{13}$$

Thus, we select states that are backwards in time relative to t_{sample} and are reachable. On the other hand, if the expanding

Algorithm 3

1: **function** STRRTGROWTREE($\mathcal{T}$)
2: z_{sample} = SampleConditionally($z_{\text{s}} \mid \Theta_k$, s_{max})
3: z_{nearest} = FindNearest($\mathcal{T}$, $z_{\text{sample}} \mid s_{\text{max}}$)
4: **if** z_{nearest} **then**
5: status, z_{new} = LocalPlanner(z_{nearest}, $z_{\text{sample}} \mid \Theta_k$)
6: **if** status not collision **then**
7: addVertex($\mathcal{T}$, z_{new})
8: addEdge($\mathcal{T}$, z_{nearest}, z_{new})
9: **return** z_{new}
10: **end if**
11: **end if**
12: **return** NULL
13: **end function**

tree is the goal tree, $\mathcal{T}_{\text{g}}$, then we define the set of feasible states which are forward in time relative to t_{sample}

$$\mathcal{Z}_{\text{feasible}} = \{(p,t) \in \mathcal{T}_{\text{g}} \mid t > t_{\text{sample}}, \\ ||p - p_{\text{sample}}||_2/(t - t_{\text{sample}}) \leq s_{\text{max}}\}. \tag{14}$$

This temporal masking ensures that our tree becomes temporally coherent and that the states are reachable. After this masking, the nearest state, z_{nearest} is computed by returning the state that has the smallest ℓ^2 distance.

$$z_{\text{nearest}} = \text{argmin}\{||z - z_{\text{sample}}||_2 \mid z \in \mathcal{Z}_{\text{feasible}}\} \tag{15}$$

With z_{nearest} found, a local planner is employed, Line 5. It computes a state z_{new}, that lies along the line connecting z_{nearest} and z_{sample}. It then collision checks the straight line transition from z_{nearest} to z_{new}. We return the collision status together with z_{new}. If the transition is collision-free then the state is added to the tree and we return the state, Line 9, else we return NULL to indicate that the tree was not expanded, Line 12.

APPENDIX II

The grid search was done by evaluating the three different planners on 5 environment instances with corresponding 5 problem scenarios for 100 time steps, $T = 100\ s$. Thus, in total 25 problem instances were used for evaluation. The scoring was based on the success rate, meaning the number of problem instances which were collision-free. In case of a tie, we scored based on the goal region rate, fraction of time spent in the goal region. The result from the grid search is presented in Table III.

TABLE III

PARAMETERS USED FOR THE PLANNERS. $\exp(a, b, N)$ MEANS THAT WE COMPUTE N WEIGHTS LINEARLY SPACED BETWEEN a AND b AND APPLY THE EXPONENTIAL FUNCTION ELEMENT-WISE.

Parameter	Reactive	Predictive	Predictive*
N_{o}	-	5	5
N_{p}	40	50	50
N_{d}	9	9	15
d_{sta}	0.05	0.05	0.05
d_{dyn}	0.3	0.2	0.2
α	$\exp(-2, 0, N_{\text{p}})$	$\exp(-2, 0, N_{\text{p}})$	$\exp(-2, 0, N_{\text{p}}))$
β	$\exp(1, 0, N_{\text{d}})$	$\exp(1, -1, N_{\text{d}})$	$\exp(1, -1, N_{\text{d}})$
ω	$\exp(1, 0, N_{\text{p}})$	$\exp(1, -1, N_{\text{p}})$	$\exp(1, -1, N_{\text{p}})$
s_{max}	0.2	0.2	0.2