ROB 590 Midterm Report
Navigating the Road Ahead: Local Path Planning and Uncovering Challenges

Nithish Kumar
University of Michigan
nithishk@umich.edu
04/28/2024

***Abstract*— Unmanned ground vehicles (UGVs) operates without the need for a human operator onboard in situations where human presence might be dangerous or impractical. In this project we assume that a satellite map of the terrain is available with the goal marked on it along with GPS coordinates. The UGV uses local sensing to detect obstacles that are not found in the satellite map. But this sometimes lead to local minima, because of the complex terrain that could cause the UGV to struggle to find a path. In this report few of the local path planning methods and insight for implementation of local path planning method in Unreal Engine are discussed.**

I. Introduction

The recent revolution in autonomous vehicles redefined mobility as we know it. Over the past few decades, mobile robots have been efficiently deployed to perform significant unmanned tasks in a variety of domains, including the military, industry, and security situations [1]. Robots have been used in more and more applications in recent years, which can save labor costs, increase production efficiency, and enhance working conditions. Automating operations like material handling in warehouses, luggage pickup in airports, and mobile security inspection robots are common uses for ground robots. More and more breakthroughs are being brought about by the development of robot applications. Robots' ability to navigate and plan on their own is what makes them useful. In order to complete tasks, robots must therefore not only understand the environment they are in but also be able to independently navigate and plan paths across certain regions.

Path planning is one of the most important problems that must be solved before mobile robots may move and explore on their own in complex environments [2]. While a UGV can autonomously generate a path, it faces three significant challenges: becoming trapped in local minima when confronted with an obstacle, difficulty navigating between obstacles, and experiencing undesirable oscillation [3]. Among these challenges, encountering local minima may particularly lead to a failed mission, as the UGV halts its navigation before reaching the goal position [4]. Therefore, predicting the local minima is increasingly important for UGV navigation.

This project aims to navigate an unmanned ground vehicle (UGV) in an unfamiliar environment using Lidar sensors to detect obstacles and its current position. The goal is to choose a local path planning strategy that navigates the UGV while avoiding detected obstacles. Techniques for detecting problems like local minima are crucial when the selected local planning approach is being used. The study examines various local path planning methods to overcome challenges and gain insights for implementation. The satellite map of the area may not be the most recent, and global knowledge may not be the most accurate. In this report, we are examining a variety of local path planning methods designed to overcome these challenges and get an insight to implement these methods.

II. Dynamic Window Approach

The Dynamic Window Approach (DWA) is a complex path planning method, since it uses the robot's detailed dynamics and kinematics to generate possible trajectories. Fundamentally, DWA establishes a "dynamic window" that includes a range of attainable velocities and accelerations that the robot is capable of achieving. Several paths are carefully analyzed inside this dynamic window according to predetermined criteria including goal proximity, safety, and smoothness. DWA can navigate through complex surroundings with efficiency, focusing on local trajectory development, by primarily functioning in the robot's velocity space instead of the standard position space. In contrast to explicit global path planning techniques, DWA sets a greater priority on local optimizations and real-time adjustments to ensure the robot's quick and secure navigation across its environment.

The initial position of the robot and target are given, and the robot only knows the outline of the global map, which means the obstacles inside the map is completely unknown. The task of robot is to reach the target while keeping the safe distance from the obstacles. The robot can reach the target with any velocity and does not need to stop at the target. Beside the outline of the global map, all the information the robot can get are obtained by the various sensors equipped. At each time step, the information required by the proposed algorithm 1 can be obtained based on these sensors. Concretely, we make the following assumptions about the navigation problem.

Algorithm and Assumptions [3]:
Assumption 1: The location, orientation and velocity of the robot in the global map are known at each time step.
Assumption 2: The position of the target in the global map is known at each time step.
Assumption 3: The distance of the nearest obstacle in each direction of a circle is known at each time step (realized by the lidar on the robot).

[x, y] is the position of the robot; θ is the orientation of the robot; v and ω are the linear velocity and angular velocity of the robot. The predicted trajectories are generated by these speed sets and the trajectory prediction time period dt.

Algorithm 1 Dynamic window Approach

```
Initialize best_traj ← None, best_cost ← ∞
while not reached_goal(robot_state, goal_pose) do
    dyn_win ← [ω_min,ω_max], [v_min, v_max]
  for v in dyn_win do
      for ω in dyn_win do
          traj ← gen_traj(robot_state, v, ω)
          cost ← eval_traj_cost(traj, goal_pose, obstacles)
          if cost < best_cost then
              best_traj ← traj
              best_cost ← cost
          end if
      end for
  end for
  exec_traj(robot_state, best_traj)
end while
return "Goal reached!"
```

The trajectory equation for the robot's motion can be represented as follows:

$$\dot{x} = v \cdot \cos(\theta)$$

$$\dot{y} = v \cdot \sin(\theta)$$

$$\dot{\theta} = \omega$$

The cost trajectory $f(T_i)$ contains three evaluation sub-functions and their coefficients c_1, c_2, c_3; σ denotes the normalization process. It can be represented as:

$$f(T_i) = \sigma[c_1 \cdot \text{heading}(v,\omega) + c_2 \cdot \text{obsdist}(v,\omega) + c_3 \cdot \text{velocity}(v,\omega)] \quad (1)$$

In the above equation, The "heading" function calculates the distance and minimum angle between the robot's orientation and the goal, indicating its proximity to the target. The "obsdist" function calculates the minimum distance from the closest obstacle (safety). The function velocity(v,ω) returns the score of the current speed, calculated by Equation 2, where σ denotes the normalization of the two speeds:

$$\text{velocity}(v,\omega) = \sigma(\omega' + v) \quad (2)$$

The scoring function of the angular speed is represented by Equation 3:

$$\omega' = \omega_{\max} - k\frac{v}{v_{\max}} \quad (3)$$

where the term $\frac{v}{v_{\max}}$ is added to measure the current linear speed, and k is a parameter which can be adjusted based on the obstacle density.

Let T_i represent a trajectory within the dynamic window of achievable velocities and accelerations. The optimization equation can be represented as:

$$T_{\text{optimal}} = \max_{T_i}\{f(T_i)\} \quad (4)$$

Where: - T_{optimal} is the optimal trajectory. - $f(T_i)$ is the cost function that evaluates trajectory T_i based on predefined objectives such as goal proximity, safety, and smoothness

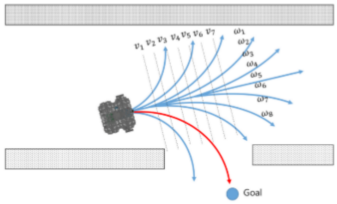


Fig. 1. Trajectory selection based on v, ω - Dynamic Window Approach. REF: C. Ovuegbe, "Parameter selection in the dynamic window approach robot collision avoidance algorithm using bayesian optimization," Ph.D. dissertation, The University of Texas at San Antonio, 2020.).

The dynamic window method is suitable for real-time path planning because of its speed, and it can also optimize the algorithm's performance by adjusting the window parameters.

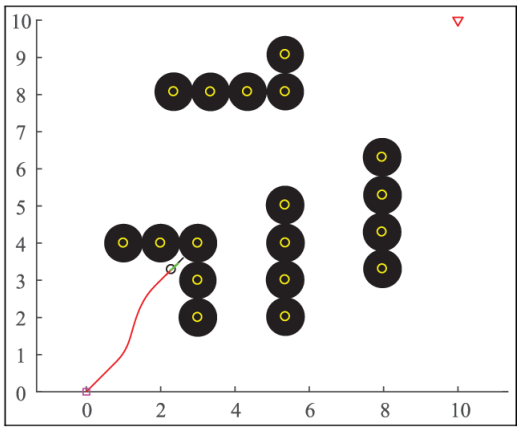
Dynamic Window Approach (DWA) exhibits certain challenges and considerations in its application. It may encounter local minima, where generated trajectories seem feasible in the short term but fail to reach the goal due to environmental constraints.

Additionally, DWA's reactive nature can sometimes lead to overly cautious or inefficient trajectories, particularly in rapidly changing environments. Tuning parameters, including the robot's maximum velocity and acceleration, is crucial for DWA's effectiveness, and improper settings can result in suboptimal or unsafe trajectories. Despite its conceptual simplicity, implementing DWA can be complex, especially with real-world robots facing non-holonomic constraints, sensor noise, and uncertain environment.

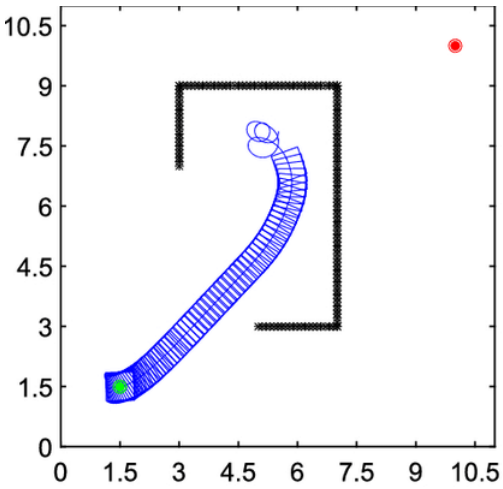
The Dynamic Window Approach is a local navigation technique that is widely utilized in Ground robots. It calculates the robot's current state and the obstacles around it to compute a "dynamic window" of suitable velocities. By considering the kinematic restrictions created by its surroundings, the robot determines the optimal velocity to achieve its goal while avoiding collisions.

III. Timed Elastic Band (TEB)

TEB is an advanced method of path planning that uses the robot's trajectory as a flexible-band to accomplish its objective and navigate through challenging settings. Its principle is similar to rubber band deformation, connecting the starting point and the target point, solving by imposing a series of constraints and designing the objective function, which is essentially a multi-objective function optimization problem. The establishment of a flexible band that encircles the nominal travel trajectory and offers the robot a margin of safety and



(a)



(b)

Fig. 2. Dynamic Window Approach local minima problem.

a) Reference: [5] —— **b)** Reference: [3]

maneuverability is a crucial element of TEB. This band adapts in response to many circumstances, including environmental limits and the proximity of obstacles, to ensure that the robot can avoid collisions and continue towards its goal.

TEB abstracts a series of pose trajectory models of the robot into a timed elastic band. [6] "The i-th pose state of the robot can be expressed as $\mathbf{S}_i = (x_i, y_i, \beta_i)$, where the pose includes configuration information x_i, y_i, and the direction angle β_i. δT_i is the transition time interval between adjacent configurations $\mathbf{S}_i$ and $\mathbf{S}_{i+1}$. (fig. 3)"

"Geometric constraints on the speed and acceleration of the robot are described by a penalty function. The average translation speed and rotation speed are calculated from the change of Euclidean distance and direction angle, respectively, from two consecutive poses $x_{\Delta t}$, $x_{(\Delta t-1)}$, and the time interval of the transition between the two poses ΔT_i, we can see that:

Line speed:

$$v_i \approx \frac{1}{\Delta T_i}\left(x_{i+1} - x_i, y_{i+1} - y_i\right)$$

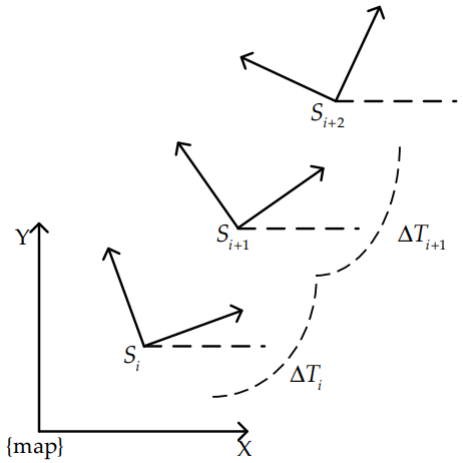


Fig. 3. Pose and time interval of UGV in Timed-Elastic Band [7]

Angular velocity:

$$\omega_i \approx \frac{\beta_{i+1} - \beta_i}{\Delta T_i}$$

Linear acceleration:

$$a_i \approx \frac{2\left(v_{i+1} - v_i\right)}{\Delta T_i + \Delta T_{i+1}}$$

Angular acceleration:

$$\alpha_i \approx \frac{2(\omega_{i+1} - \omega_i)}{\Delta T_{i+1} + \Delta T_i}$$

There are two main constraints in the TEB algorithm: the first is based on the penalty function, which constrains the velocity and acceleration. The second takes the indexes related to path optimization as constraints, such as the shortest time, shortest path, or distance from obstacles.

Algorithm 2 TEB algorithm

```
Sample trajs based on UGV's constraints.
while not converged do
    for samp_traj in samp_trajs do
        col_pred ← predict_collision(samp_traj, obs)
        eval ← eval_traj(samp_traj, col_pred, other_criteria)
        if eval is favorable then
            selected_traj ← samp_traj
        end if
    end for
    exec_traj(robot_state, selected_traj)
end while
return "Goal reached!"
```

Band adjustment involves using a proportional control law to dynamically scale the band width based on the distance to the nearest obstacle. The band width w can be adjusted based on the following equation:

$$w = w_{\text{min}} + k \cdot d_{\text{min}} \tag{5}$$

Where, w_{min} is the minimum width of the band, ensuring a minimum level of safety clearance around the robot, k is a scaling factor that determines the rate at which the band width adjusts based on the distance to the nearest obstacle, d_{min} is the distance to the nearest obstacle, measured perpendicular to the nominal path trajectory.

Moreover, TEB gives equal weight to goal-reaching abilities and collision avoidance, enabling the robot to move forward through difficult terrain and yet reach its destination.

The cost function J can be expressed as:

$$J = J_{\text{collision}} + J_{\text{deviation}} + J_{\text{goal}} \tag{6}$$

($J_{\text{collision}}$) penalty is typically based on the proximity to obstacles or the likelihood of collision. One common formulation is to use a potential field approach where the penalty increases as the distance to obstacles decreases.

$$J_{\text{collision}} = \sum_{i=1}^{n_{\text{obstacles}}} \frac{1}{d_i}$$

Where $n_{\text{obstacles}}$ is the number of obstacles, and d_i is the distance to the i-th obstacle.

($J_{\text{deviation}}$) penalty penalizes deviations from the nominal path. It can be based on differences in position, velocity, or acceleration between the planned trajectory and the nominal path.

$$J_{\text{deviation}} = \int_{t_0}^{t_f} \|S_t - S_{t_{\text{nominal}}}\|^2 \, dt$$

Where S_t is the actual trajectory, S_{nominal_t} is the nominal path, and t_0 and t_f are the start and end times of the trajectory.

(J_{goal}) penalty promotes progress towards the goal by penalizing trajectories that do not make sufficient progress towards the goal region.

$$J_{\text{goal}} = \frac{1}{d_{\text{goal}}}$$

Where d_{goal} is the distance to the goal.

Furthermore, TEB takes position and velocity into account when planning its trajectory, which helps the system make decisions and modify its movement plan in response to feedback from the environment and real-time sensor data.

The Timed Elastic Band (TEB) algorithm presents several key considerations in its application. Firstly, It lacks the constraint of considering the shortest local path, and the local path planned is not the optimal path. TEB's optimization over both spatial and temporal domains can lead to increased computational complexity compared to spatial-based approaches, potentially challenging real-time applications or robots with limited computational resources. Parameter tuning is crucial for TEB's effectiveness, involving careful adjustment of factors such as trajectory smoothness versus obstacle avoidance trade-offs, temporal discretization intervals, and penalties for violating constraints. Failure to properly tune these parameters may result in suboptimal trajectories or even the inability to find feasible solutions. TEB is susceptible to local minima, particularly in complex environments, potentially leading to trajectories that seem feasible locally but fail to reach the goal due to unforeseen obstacles or constraints.

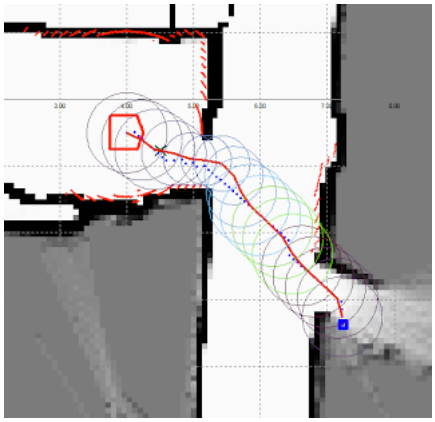


Fig. 4. local minima in Timed-Elastic Band - Global path in blue, bubbles as colored circles, and the red path is the one generated by the elastic band algorithm. [8]

Moreover, TEB typically optimizes for a single objective, such as minimizing travel time or energy consumption, yet robotic navigation often involves multiple competing objectives such as safety, comfort, and energy efficiency. Addressing these objectives simultaneously while ensuring real-time feasibility remains a challenging task. Additionally, while TEB generates dynamically feasible trajectories, it does not directly handle the low-level control of the robot to execute these trajectories accurately. Thus, integrating TEB with suitable control strategies to account for factors like actuator dynamics and disturbances is essential for achieving robust and reliable navigation performance.

In robotic motion planning, the Timed Elastic Band technique is utilized for tracking and trajectory optimization. It functions by displaying the planned path as a series of intervals, each with a distinct length. The TEB algorithm optimizes the trajectory to meet requirements including collision avoidance and smoothness by changing the length of these intervals.

IV. Receding Horizon Model Predictive Control (RHMPC) [9]

RHMPC is based on the traditional optimal control, which can be obtained for either an infinite horizon or a fixed finite horizon by minimizing or mini-maximizing a performance objective [10]. In contrast to an unlimited horizon and a fixed finite horizon, RHMPC considers receding horizon. RHMPC's fundamental idea is as follows. Currently, on a specified finite horizon from the current time, optimal controls either closed loop or open loop are obtained. Only the first of the optimal controls on the fixed finite horizon is in use as the control law at this time. After that, the process is repeated the following time, but this time with a definite, finite horizon.

A nominal path is used for the navigation of paths generated by local motion planners, the methods by which parameterized controls are generated, and trajectory optimization techniques

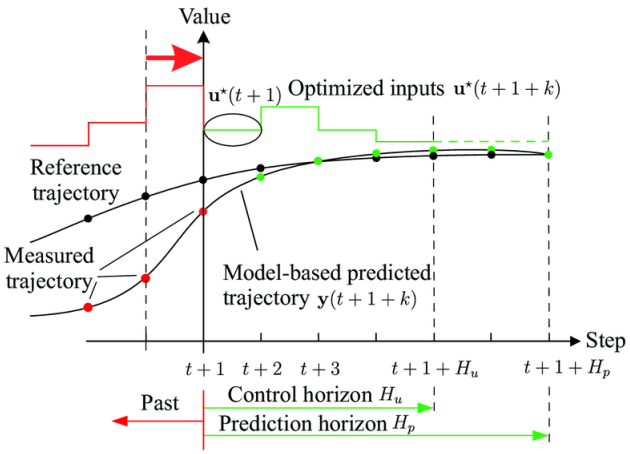


Fig. 5. Functionality of RHMPC

used to generate corrective actions. The trajectory follower is formulated as an optimal control problem, requiring actions from a set of functions (U(x,t)) to represent control inputs (u(x,t)) which, when subject to the predictive motion model (f_{PMM}(x,u,t)), that minimize a penalty function (J(x,u,t)). An additional requirement for the trajectory follower is that the resulting control must be defined for a specific period of time or distance, allowing the optimized path to be evaluated for hazards to ensure vehicle safety.

$$\begin{aligned} \text{minimize: } & J(x,u,t) \\ \text{subject to: } & \dot{x} = f_{\text{PMM}}(x,u,t) \\ & x(t_I) = x_I \\ & u(x) \in U(x), t \in [t_I, t_F] \end{aligned}$$

Control parameterization is a difficult problem in motion planning, as it involves reducing the continuous action to a manageable space to search. The trajectory following technique uses a portion of the reference controls as the initial guess, which is divided into primitives used by the motion planner.

J(x,u,t) is the cost function which depends on the state trajectory $x(t)$, the control input u, and time t. $\Phi(x(t_I), t_I, x(t_F), t_F)$ is the terminal cost, which represents the cost associated with the states of the system at the initial time t_I and the final time t_F. $\int_{t_I}^{t_F} L(x(t), u(p,x), t)\, dt$ is the integral cost, which represents the accumulated cost over the time interval $[t_I, t_F]$. $L(x(t), u(p,x), t)$ is the instantaneous cost function, which evaluates the cost associated with the current state $x(t)$, control input $u(p,x)$, and time t.

$$J(x,u,t) = \Phi(x(t_I), t_I, x(t_F), t_F) + \int_{t_I}^{t_F} L(x(t), u(p,x), t)dt \tag{7}$$

The trajectory deviation optimal control involves modifying the parameters to compensate for disturbances, approximations, and errors in the motion model. This technique seeks to minimize a cost function (J(x,u,t)) by modifying a set of control inputs. The initial corrective action is evaluated through the predictive motion model subject to the initial state constraints to obtain a cost estimate.

$$p_{\text{RHMPC}_i} = p_{\text{RHMPC}_{i-1}} - \alpha \nabla J(x,u,t), \quad i \geq 1 \tag{8}$$

Parameterized freedom vectors (p_i) control the shape of each set of inputs (u(p_i,x,t)) that define the reference trajectory. The initial guess for the parameterized control inputs (u_{RHMPC}(p_{RHMPC},x,t)) is defined by the sequence of trajectory segments Φ between the current state and the predefined fixed control horizon (possible control inputs).

Algorithm 3 Receding Horizon MPC with Optimization and Path Deviation Control [9]

procedure RECEDINGHORIZONMPC(x_0)
  Initialize N, Δt
  while Goal_not_Reached **do**
    Initialization:
    x(t) ← (robot state)
    (U(x,t)) ← (Control inputs)
    Prediction:
    for $k = 0$ **to** $N-1$ **do**
      Add x_t and ControlInput($x_t, \Delta t$) to lists
      $J(x,u,t) = \Phi(x(t_I), t_I, x(t_F), t_F) + \int_{t_I}^{t_F} L(x(t), u(p,x), t)dt$
    end for
    Optimization:
    $p_{\text{RHMPC}_i} = p_{\text{RHMPC}_{i-1}} - \alpha \nabla J(x,u,t), \quad i \geq 1$
    Update:
      $x_{t+1} \leftarrow f_{\text{PMM}}(x_t, p_{\text{RHMPC}_i}[0], \Delta t)$
      $x_t \leftarrow x_{t+1}$
  end while
end procedure

Advantages and Disadvantages [10]:
Advantages: • Applicability to a Broad Class of Systems: RHC can be applied to a wide range of systems, including nonlinear systems and time-delayed systems.
• Systematic Approach to Closed Loop Control: RHC provides closed-loop controls due to repeated computation and implementation of only the first control.
• Constraint Handling Capability: RHC can be computed using mathematical programming like quadratic programming (QP) and semidefinite programming (SDP) for linear systems with input and state constraints.
• Guaranteed Stability: RHC guarantees stability under weak conditions for linear and nonlinear systems with input and state constraints.
• Good Tracking Performance: RHC provides good tracking performance by utilizing the future reference signal for a finite horizon.
• Adaptation to Changed Parameters: RHC can adapt to future system parameters changes that can be known later, unlike infinite horizon optimal controls which are computed once in the first instance.
• Good Properties for Linear Systems: RHC possesses good properties such as guaranteed stability under weak conditions and a certain robustness.

• Easier Computation: RHC can be computed in an easy batch form for a linear system, while an optimal control on the infinite horizon is hard to compute.
• Broad Industrial Applications: RHC has broad industrial applications, particularly in industrial processes, due to its ability to efficiently handle constraints.

Disadvantages:
• RHC may have longer computation time compared to conventional nonoptimal controls, but this can be overcome by the high speed of digital processors and improvements in optimization algorithms.
• Difficulty in Designing Robust Controls for Parameter Uncertainties: System properties like robust stability and performance due to parameter uncertainties are usually intractable in optimization problems on which RHCs are based.

In "Receding Horizon Model Predictive Control," an optimization problem is repeatedly solved across a finite time horizon, and the best control inputs for the current time step are obtained from the solution. An updated state estimate is used to resolving the optimization problem at each time step. The system can react to uncertainties and disturbances while optimizing control actions over a finite prediction horizon.

V. Model Predictive Path Integral (MPPI)

(REF: [11]

The Model Predictive Path Integral (MPPI) algorithm is a complex control approach used in robotics and autonomous systems to provide optimal trajectory tracking and control. Because MPPI uses a probabilistic technique rather than a traditional one, it can effectively handle complicated, nonlinear systems with uncertainties.

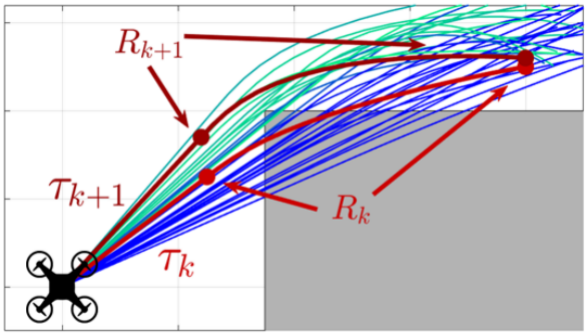


Fig. 6. Sampled trajectories in MPPI: The trajectory T_k represents the the optimal trajectory from the sample trajectories (blue) from the previous control sequences,a nd trajectory T_{k+1} represents the the optimal trajectory from the sample trajectories (green) from the new control sequences. where $R_k and R_{k+1}$ are are the position where next set of control sequences is obtained.

Essentially, MPPI samples from a population with possible trajectories to generate control trajectories using path integral formulation. These trajectories are assessed taking into account both the nominal trajectory generated by a global path planning method and system dynamics, based on their expected performance over a finite time horizon. MPPI takes control restrictions and system dynamics into account as it iteratively modifies the control inputs in an effort to minimize a cost function that represents the deviation from the nominal trajectory.

The author [11] consider stochastic dynamical systems with the state and controls at time t denoted $x_t \in \mathbb{R}^n$ and $u_t \in \mathbb{R}^m$. These dynamics are disturbed by the Brownian motion (which generates random non-negative integers) $dw \in \mathbb{R}^p$. We also define $u(\cdot) : [t_0, T] \to \mathbb{R}^m$ as the function which maps time to control inputs, and define the function $\tau : [t_0, T] \to \mathbb{R}^n$ as a trajectory of the system. In the classical stochastic optimal control setting, we seek a control sequence $u^*(\cdot)$ such that:

$$u^*(\cdot) = \arg\min_{u(\cdot)} \mathbb{E}_Q \left[\varphi(x_T, T) + \int_{t_0}^{T} L(x_t, u_t, t)\, dt \right] \quad (9)$$

where the expectation is taken with respect to the dynamics: $dx = F(x_t, u_t, t)\, dt + B(x_t, t)\, dw$. In this paper, we consider costs composed of an arbitrary state-dependent term and a quadratic control cost:

$$L(x_t, u_t, t) = q(x_t, t) + \frac{1}{2} u_t^T R(x_t, t) u_t \quad (10)$$

and dynamics of the UGV is generally a non-linear model which can be found in the reference [12] which are affine in controls:

$$F(x_t, u_t, t) = f(x_t, t) + G(x_t, t) u_t \quad (11)$$

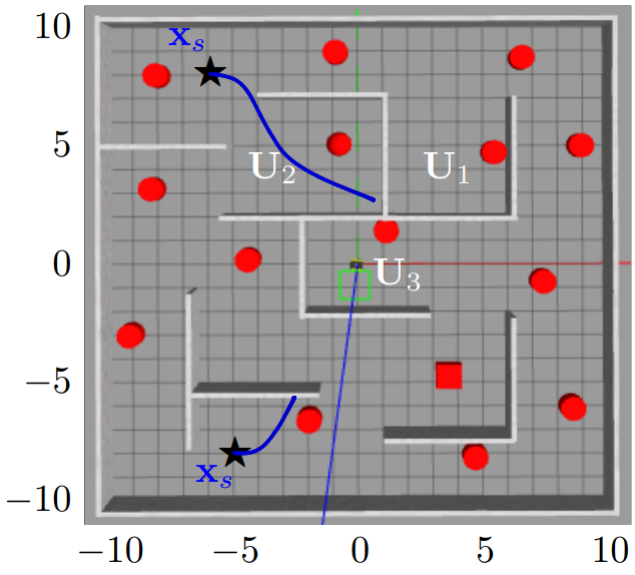


Fig. 7. Local Minima in MPPI: There are 2 situation of local minima represented in this figure: 1)The UGV got trapped in a dead end 2) Couldn't navigate through a narrow path

Model Predictive Path Integral is a stochastic optimization technique used for trajectory optimization and control in robotics. It defines the control problem as minimizing a cost function over a finite time horizon by determining a series of control inputs. Using a sampling-based methodology, MPPI creates a distribution of noise perturbations into different control trajectories. MPPI computes a control signal that maintains

Algorithm 4 Model Predictive Path Integral Algorithm

Require: K: Number of samples
N: Number of timesteps (time horizon)
$(u_0, u_1, \ldots, u_{N-1})$: Initial control sequence (based on nominal trajectory)
$\Delta t, x_{t0}, f, G, B, BE$: Robot state/dynamics
ϕ, q, R, λ: Cost parameters
u_{init}: Value to initialize new controls to
while task not completed **do**
  for $k \leftarrow 0$ **to** $K-1$ **do** (Sampling)
    $x \leftarrow x_{t0}$
    for $i \leftarrow 1$ **to** $N-1$ **do** (Time horizon)
      $x_{i+1} \leftarrow x_i + (f + Gu_i)\Delta t + BE\epsilon_{i,k}\sqrt{\Delta t}$

      Cost function:
      $\tilde{S}(\tau_k) \leftarrow \tilde{S}(\tau_k) + \tilde{q}(x_i, u_i, \epsilon_{i,k}, t_i)$
    end for
  end for
  for $i \leftarrow 0$ **to** $N-1$ **do** (Control Update)
    $u_i \leftarrow u_i + H^{-1}G\left[\frac{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_k)\right)\epsilon_{i,k}\sqrt{\Delta t}}{\sum_{k=1}^{K}\exp\left(-\frac{1}{\lambda}\tilde{S}(\tau_k)\right)}\right]$
  end for
  send to actuators(u_0)
  for $i \leftarrow 0$ **to** $N-2$ **do**
    $u_i \leftarrow u_{i+1}$
  end for
  $u_{N-1} \leftarrow u_{\text{init}}$
  Update the current state after receiving feedback
  check for task completion
end while

a balance between exploring the control space and utilizing past experiences by integrating the cost along these sampled trajectories. This allows for flexible and adaptive control in unpredictable environments.

VI. Directions for Implementing Path Planning in Unreal Engine

A. Environment Consideration

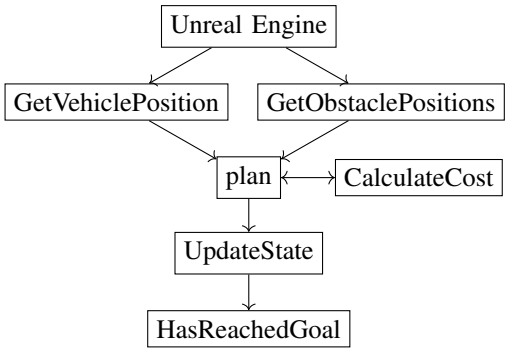
In an environment spanning A square kilometers ($l_1 \times l_2$), we'll strategically place obstacles of varying sizes and densities. The plan involves approximately distributing obstacles with size (radius/length) range of $\Psi_1 - \Psi_2$ at the rate of o_{min} to o_{max} per 100 square meters. With approximately inaccessibility of the environment rendered inaccessible by the UGV due to these obstacles, our simulation will closely mirror real-world challenges, allowing for a comprehensive assessment of the vehicle's capabilities in navigating complex terrains.

The vehicle will have a mass of M tons, dimensions of L meters, W meters, H meters. Its wheelbase will be B meters. It will have an average speed of V m/s, a sensor range of R meters, and step time of Δt seconds for each planned trajectory in simulation. Local path planning methods will be considered for managing navigation, obstacle avoidance, path planning, and trajectory optimization. Simulations and tests will be conducted within Unreal Engine, allowing the UGV to navigate through the environment using the selected local path planning method.

B. Path Planning Algorithm interaction with Unreal Engine

To implement a local path planning method in Unreal Engine, integrating Blueprint scripting with C++ functionality is a effective approach to implementing local path planning methods. BlueprintCallable functions declared within C++ classes serve as interfaces between Blueprints' visual scripting environment and the underlying C++ logic. The UFUNCTION(BlueprintCallable) macro defines these functions, making it possible for Blueprint scripts to access and use them directly. Through this integration, developers may take advantage of the advantages of both scripting frameworks, combining the flexibility and performance of C++ with the ease of visual scripting. Once the C++ functions are implemented and the code is compiled, the C++ functions accessible as nodes within the Blueprint Editor, enabling developers to seamlessly incorporate C++ functionality into their Blueprint scripts.

All actors in the game environment in Unreal Engine use 'AActor' as their basic class. An actor is defined as any object that can be placed in a level and interacts with, in this case it is the vehicle. The C++ class will be given access to all of the features and properties associated with the actors by deriving from AActor. We also include "Components/StaticMeshComponent.h", is the details of the static mesh components to represent game world objects. It is used to depict static geometry or immobile objects using static mesh components.



Some of the important functions are discussed below:

GetVehiclePosition() is responsible for fetching the vehicle's current position within the game world, providing crucial data for trajectory planning and collision avoidance. Meanwhile, GetObstaclePositions() retrieves nearby obstacle positions, essential for ensuring safe navigation during motion planning. CalculateCost() evaluates the desirability of a given state by considering factors like distance to the goal and obstacle proximity. On the action side, Plan() computes a sequence of actions to guide the vehicle toward its goal while avoiding obstacles. Lastly, UpdateState() dynamically adjusts the vehicle's state, including position, orientation, and velocity, based on received control inputs, ensuring accurate simulation of its motion over time. Together, these functions empower effective navigation and control of the vehicle in dynamic environments.

The UpdateState function plays a crucial role in modifying the state of the vehicle according to the control inputs that are decided upon during trajectory planning. This function updates the location, orientation, and velocity of the vehicle in real time using a kinematic bicycle model that has been simplified. The vehicle's motion is controlled by the kinematic bicycle model equations, which take into account variables such the vehicle's current velocity, steering angle, and time step (Δt). The following equations are used to express these:

$$\begin{aligned}
\frac{dx}{dt} &= v \cdot \cos(\theta) \\
\frac{dy}{dt} &= v \cdot \sin(\theta) \\
\frac{d\theta}{dt} &= \left(\frac{v}{L}\right) \cdot \tan(\delta) \\
\frac{dv}{dt} &= a
\end{aligned}$$

where:

- $\frac{dx}{dt}$ represents the rate of change in horizontal position (x) over time (t), determined by v and $\cos(\theta)$.
- $\frac{dy}{dt}$ represents the rate of change in vertical position (y) over time (t), determined by v and $\sin(\theta)$.
- $\frac{d\theta}{dt}$ represents the rate of change in orientation angle (θ) over time (t), influenced by v, wheelbase L, and $\tan(\delta)$, where δ is the steering angle.
- $\frac{dv}{dt}$ represents the rate of change in linear velocity (v) over time (t), determined by linear acceleration a.

VII. Next Steps

Design a simulation environment that represents the robot's operating environment, including terrain and obstacles. Develop the simulation to be used for the user experiment for the robot controller including autonomous path planning with local obstacle avoidance Design a user-friendly interface for human operators to interact with the unmanned ground vehicle (UGV). Implement features to display both local and global views of the UGV's surroundings, providing real-time feedback on its environment. Incorporate visualization tools to highlight detected obstacles, planned paths, and the UGV's current position, aiding the operator in monitoring its progress. Include functionality for the operator to add or adjust waypoints on the UGV's planned path, enabling manual intervention when necessary or specifying custom routes. Conduct usability testing to gather feedback from operators and iterate on the interface design to optimize user experience and effectiveness.

References

[1] R. Raj and A. Kos, "A comprehensive study of mobile robot: History, developments, applications, and future research perspectives," *Applied Sciences*, vol. 12, no. 14, p. 6951, 2022.

[2] O. Hachour, "Path planning of autonomous mobile robot," *International journal of systems applications, engineering & development*, vol. 2, no. 4, pp. 178–190, 2008.

[3] L. Chang, L. Shan, C. Jiang, and Y. Dai, "Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment," *Autonomous robots*, vol. 45, pp. 51–76, 2021.

[4] J. R. Sánchez-Ibáñez, C. J. Pérez-del Pulgar, and A. García-Cerezo, "Path planning for autonomous mobile robots: A review," *Sensors*, vol. 21, no. 23, p. 7898, 2021.

[5] Z. Lin, M. Yue, G. Chen, and J. Sun, "Path planning of mobile robot with PSO-based APF and fuzzy-based DWA subject to moving obstacles," *Transactions of the Institute of Measurement and Control*, vol. 44, no. 1, pp. 121–132, 2022.

[6] M. Li and C. Yang, "Navigation simulation of autonomous mobile robot based on TEB path planner," in *Proceedings of the 2021 1st International Conference on Control and Intelligent Robotics*, 2021, pp. 687–691.

[7] J. Wu, X. Ma, T. Peng, and H. Wang, "An improved timed elastic band (TEB) algorithm of autonomous ground vehicle (AGV) in complex environment," *Sensors*, vol. 21, no. 24, p. 8312, 2021.

[8] A. Boeing, S. Pangeni, T. Bräunl, and C. S. Lee, "Real-time tactical motion planning and obstacle avoidance for multi-robot cooperative reconnaissance," in *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2012, pp. 3117–3122.

[9] T. M. Howard, C. J. Green, and A. Kelly, *Receding horizon model-predictive control for mobile robot navigation of intricate paths*. Springer, 2010.

[10] W. H. Kwon and S. H. Han, *Receding horizon control: model predictive control for state models*. Springer Science & Business Media, 2005.

[11] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou, "Aggressive driving with model predictive path integral control," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1433–1440.

[12] R. Pepy, A. Lambert, and H. Mounier, "Path planning using a dynamic vehicle model," in *2006 2nd International Conference on Information & Communication Technologies*, vol. 1. IEEE, 2006, pp. 781–786.