# Implementation of Hierarchical Route Planning Framework for Off-Road Vehicles

Shubham Gupta
gupta9@g.clemson.edu

Shubhankar Kulkarni
sskulka@g.clemson.edu

Priyanshu Rawat
prawat@g.clemson.edu

***Abstract*— Detection of the near-field environment, path planning, and tracking are an area of immense interest in autonomous driving. Most path-planning algorithms, in both on-road and off-road settings, structure the task into global and local plan components. To achieve these tasks, a hierarchical approach using a coarse grid Dynamic Program for a global plan and a receding horizon Model Predictive Control for the local plan is demonstrated. An application for off-road vehicles is presented by utilizing various layers of terrain knowledge.**

## I. INTRODUCTION

This work is a reconstruction of a similarly titled paper, "A Hierarchical Route Guidance Framework for Off-Road Connected Vehicles"[1]. The original paper has been supplemented here by providing certain missing implementation details, corrections, improvements, and interesting computation highlights.

The project explores a hierarchical path-planning strategy for off-road vehicles. Path planning is usually achieved as a two-step process - generating a global and a local path. A global path focuses on the bigger picture with the intent of reaching the intended destination. As an example, a global path is what our phone navigation services provide to us. Global path planning relies on static route information, like elevation change, and can be calculated prior to trip initiation. On the other hand, a local path accounts for the dynamic environment changes and supplements the global path. Analogous to a local path would be a driver's understanding on the visible road ahead and driving lane choices based on traffic. The driver still follows the global path suggested by the navigation service, though he augments it with real-time surrounding conditions.

The complexity of path planning differs significantly between on-road and off-road vehicles. On-road path planning considers limited sets of defined roads or, mathematically speaking, a graph with a limited number of vertices and edges. This results in a relatively computationally easier problem. Off-road path planning, however, needs to deal with infinite path possibilities and the computational complexity depends on the resolution choice of the designer.

Ideally, we would want to employ an optimal routine with an infinite horizon and the highest resolution terrain graph to achieve the most optimal path. It is easy to see how that would be a poor and computationally infeasible choice. Most optimal routines like dynamic programming and model predictive control suffer from the curse of dimensionality. As we increase the graph resolution, whose vertices will act as waypoints, the dimensionality increases and renders the optimal routine calculation infeasible. Moreover, an infinite horizon for local path planning would not make sense because the dynamic surrounding knowledge is obtained from the limited horizon capability onboard perception sensors of the autonomous vehicle.

Therefore, a hierarchical path planning approach has been explored that leverages the strengths of two optimal routines while keeping the dimensionality cost in mind. Furthermore, this approach is feasible due to the availability of various terrain information maps from invaluable sources such as the U.S. Geological Survey[1].

The report is structured as follows: section II presents the explored hierarchical approach. Section III presents the Djikstra-based dynamic program used for global path planning. Section IV explains the model predictive controller-based local path planning. Both section III and section IV highlight various implementation details and choices made throughout the project. Section V discusses the simulation results.

## II. HIERARCHICAL PATH PLANNING ARCHITECTURE

In this work, a hierarchical path planning architecture has been implemented by using a Djikstra-based Dynamic Program for the global path and a receding horizon Model Predictive Control approach for the local path.

### A. Architecture

The optimal routine choices for the two components of path planning are explained below:

1) **Dynamic programming**: Dynamic programming is a powerful tool to compute the optimal cost-to-go between a start point and the destination. Based on the *Principle of Optimality*, dynamic programming provides a computationally efficient way of exploring all possible paths. The optimal cost-to-go map can be calculated once before the journey begins and stored on local memory. Thereafter, we know the optimal cost-to-go between any two points. As a result, we could de1viate from the optimal path and still correct ourselves without having to recompute the cost map. The biggest disadvantage of Dynamic Programming arises from the curse of dimensionality, making the approach infeasible for large grids. Therefore, dynamic programming presents a suitable optimal routine for global path planning where coarse grid static terrain information maps can be used. A global cost-to-go for the entire region is generated using the dynamic programming routine and serves as

a starting point. This map can be thought of as a sparser cost map that does not have a thoroughly detailed representation of the features leading to a less rampant changing cost from point to point.

2) **Model Predictive Planner**: The local path planning in our architecture uses a receding horizon Model Predictive Controller (MPC) based approach. An MPC provides the optimal trajectory by taking into account the resultant costs in a defined horizon. However, as the length of the prediction horizon increases, the MPC takes longer to converge and becomes infeasible. Therefore, a Model Predictive Controller provides the most suitable optimal routine that can leverage the perception capabilities of the limited-range onboard vehicle sensors to design the local path.
The MPC uses high-resolution terrain maps similar to the ones used by the Dynamic Program. In addition, the real-time surrounding perception of obstacles is appended to these maps. This enables the vehicle to identify lesser prominent terrain features which may not be captured within the low-resolution global maps or some factors which have changed since the last instance the maps were created. It also gives a good idea about the dynamic factors which the vehicle might encounter over the length of its horizon such as a moving obstacle. The MPC is a computationally expensive algorithm that mandates that a global heuristic should be provided to it in the form of costs-to-go on the horizon of its operation. The local path planner then gives a sequence of admissible control inputs over the stages of the horizon which drive the vehicle

### B. Terrain Maps

A conventional on-road vehicle path planner is constrained in the sense that the road network is present in finite locations between the start and the goal. Conversely, it can be safely presumed that the areas which constitute the road surfaces would be having a gentler slope variation with a good enough surface conducive to higher traction forces. This simplifies the path planning for on-road vehicles to just extrinsic conditions like blockades, lengths, and duration. Off-road vehicles on the other hand need to have an imperative understanding of the environments which they are planning to traverse beforehand. This is a rather subjective matter since each vehicle has a distinct set of kinematics, and the resulting dynamics vary over the same surfaces. A vehicle needs to accurately know the grade change on the path that it's taking as well as the soil conditions since they can pose problems of not being traversable. Simultaneously it would also benefit from taking the trajectory which has the least gradient increase and/or more of a downward slope from the point of energy conservation. This, coupled with unpredictable conditions like small obstacles which might not be picked up when the global map was last updated, would create an issue in traversing that point. Hence a two-pronged approach of combining a global path estimation coupled with a local real-time shorter distance planner seems to address all those problems.

The hierarchical approach presented in section II-A is feasible only due to the availability of terrain information maps from the U.S. Geological Survey and the U.S. Department of Agriculture. For our study, a geographical area of 10,000 meters by 10,000 meters is considered. Various terrain information maps with a resolution of 50 meters are considered as follows:

a *Elevation profile*: Figure 1 shows the elevation profile of the terrain with the altitude ranging from 1000 meters to just about 3200 meters. Elevation changes affect energy consumption and are subject to the vehicle's traversability capabilities.

b *Soil profile*: Figure 2 depicts the soil trafficability ratings of the terrain. When performing off-road route planning, soil trafficability is the most pertinent information. An untraversable soil profile could lead to loss of vehicle. The soil trafficability ratings are based on soil slipperiness, soil strength, stickiness, stoniness and slope. The ratings class describes the expectation to complete the task for the given military vehicle category, season of the year, and number of passes to complete the task. For computational ease, numerical values were used instead of the probabilistic trafficability ratings of the soil. The probabilistic ratings and the corresponding numerical values have been replicated from the original work in Table I.

c *Visibility*: In the scope of this project, an army off-road vehicle application is also kept in mind, and hence visibility in foreign territories is a necessity. The elevation map has been used with the function *voxelviewshed* in MATLAB to generate a visibility map. A value of 0 indicates an invisible or safe area, while a value of 1 indicates a visible or unsafe area. Figure 3 presents the visibility map.

All of the terrain maps in figures 1, 2, and 3 have been plotted using the *surf* function in MATLAB. It is important to note that *surf* is usually computationally feasible only for smaller square grids of size up to 2000 x 2000 vertices. Since our maps are 200 x 200, *surf* can be easily used.

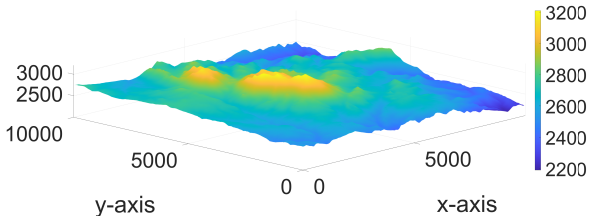


Fig. 1. Elevation profile

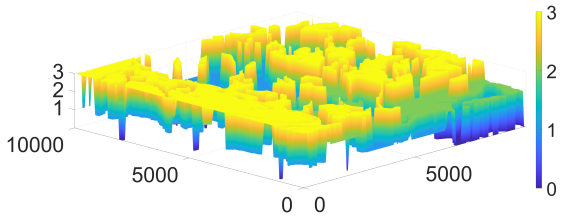


Fig. 2. Soil trafficability profile

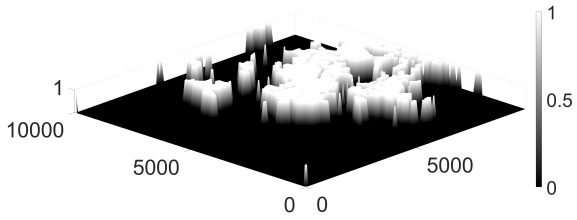


Fig. 3. Visibility map

TABLE I
SOIL TRAFFICABILITY RATINGS

| Rating | Probabilistic value | Numerical value |
|---|---|---|
| Excellent | 0.90-1.00 | 4 |
| Good | 0.75-0.89 | 3 |
| Fair | 0.50-74 | 2 |
| Poor | 0.00-49 | 1 |

Figure 4 presents a visual representation of the hierarchical architecture.

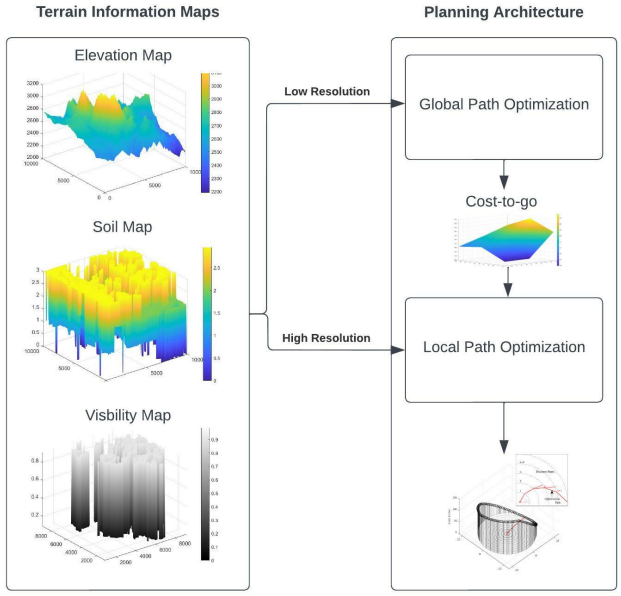


Fig. 4. Overall architecture

## III. GLOBAL PATH PLANNING USING DYNAMIC PROGRAMMING

In this work, the dynamic program for global path planning has been implemented using Dijkstra's Algorithm (DA). DA is one of the most important algorithms in the fields of Operations Research, Statistics, and Computer Science for solving shortest path problems. It can be thought of as a specialized implementation of dynamic programming based upon successive calculations of positive inter-node distances. Understanding the similarities between its "greedy" approach characteristics with the reliance on conditions necessary for optimality helps us portray DA as a subset of DP.

Initially, only the end node is considered, and the adjacent nodes of it are added to the queue after their cost-to-go values are computed. A DA deploys a greedy rule to determine the next node for processing by looking up the minimum of costs of the nodes in the to-be-processed queue. Another important feature is the basis of cost updation in DA occurs in only the nodes that haven't been processed yet. This ensures computational efficiency, unlike the traditional DP, which processes all the adjacent nodes and assigns costs if they are lesser than the existing ones on that specific node. The general algorithm flow can be given as:

Initialize: F(1)=0; F(j)=inf, j$\epsilon$1; $Q = \{1\} \cup A(1)$

While(|Q|>1) Do:
  j = argmin{F(i) : i $\epsilon Q$}
  Q = Q $\setminus \{j\}$
  for i $\epsilon A(j) \cap Q$ *Do* :
    G = min{F(i),F(j)+D{(j,i)}
    If(G<F(i)) Do
      F(i) = G
      Q = Q $\cup A\{i\}$
    End Do
  End Do
End Do

Here $F(i)$ represents the cost-to-go at point $i$. The $Q$ represents the queue of nodes to be processed. $G$ represents the computed cost of the node $i$ from the adjacent nodes of node $j$ selected based on the least cost.$A(j)$ denotes the adjacent nodes of node $j$ which can be typically thought of as successors i.e. the nodes which aren't processed yet.

The cost function on each node in the low-resolution global cost map is a cumulation of sub-types of costs pertaining to euclidean distance, the soil properties, the elevation difference, and the boolean elevation cost coupled with a soft constraint regarding to the max allowable slope. A set of tuned weights is attached to these scalar cost values, which is found only through experimenting around with the cost values being generated and having an idea of how much penalty to attach to each of the entities being violated, rendering the costs higher when they get violated. The cost computation is shown as follows:

Distance Cost :

$$J_1(i,j) = ||(x(i), y(i), z(i)), (x(j), y(j), z(j)||_2 \quad (2)$$

Soil Cost :

$$J_2(i,j) = 1/soil(i) + 1/soil(j) \quad (3)$$

Elevation Cost :

$$J_3(i,j) = |z(i) - z(j)| \quad (4)$$

Visibility Cost :

$$J_4(i,j) = visibility(j) \quad (5)$$

Their results are added up to give a cost-to-go for the low-resolution map, which gives us a heuristic of what trajectories would render the least cost of movement from the start point to the endpoint.

$$J(i,j) = \sum_{n=1}^{4} w_n * J_n(i,j) \quad (6)$$

Furthermore, to take the physical limits of the vehicle into account, maximum allowable slope constraints were implemented as follows:

$$slope(i,j) > slope_{max} \quad (7)$$

where $slope_{max} = 0.3$ representing a maximum allowable slope of 30% for off-road vehicles.

For the purpose of our simulation, we chose the cost weights as $w_1 = 2$, $w_2 = 7.5$, $w_3 = 10$, and $w_4 = 50$. Since distance only affects energy consumption, it has been penalized the least. Soil profile and elevation present a small possibility of a loss of vehicle and have been penalized more than distance. Finally, visibility presents imminent danger and has been assigned a huge cost.

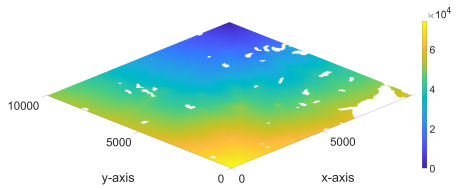


Fig. 5. Global Cost-to-go map

The global cost-to-go map that gets generated can be seen above. The computation starts from the endpoint, and hence it has an assigned cost of zero, and the other nodes get an incrementally high cost as they proceed toward the starting point. This has the advantage that due to high initial node costs-to-go, provided a suitable weight to scale it; the local planner gives a higher weightage towards the trajectories trending in the direction of the endpoint. It can also be seen that the shading is not uniform, and there are some local minima and maxima, as seen in the figure. This serves as a good input to the local planner for it to route its planner accordingly. Some of the areas on the map appear to be white, representing the zones that exceed the maximum gradient constraint and hence get assigned an infinity cost causing them not to be further evaluated. In this way, the vehicle can have a pre-determined sense of what trajectories to take depending upon the cost-to-go values, which can be interpolated to a high-resolution map and mapped to the points on the horizon of the MPC algorithm.

## IV. Local Path Planning Using Model Predictive Path Programming

The local plan is generated using a receding horizon Model Predictive approach in real-time as the vehicle traverses. The MPC uses a cost function similar to the one used by the dynamic program in section III. However, high-resolution terrain maps and obstacle perception are used to incorporate finer details. Finally, the MPC includes the global optimal cost-to-go generated by the DP as its terminal cost to ensure the global mission is met.

### A. Algorithm

To reduce the degree of freedom of the existing system, the speed of the vehicle was assumed to be constant. A no-slip kinematic vehicle model was considered to get a discretized system model as follows:

$$\begin{aligned} x_{k+1} &= x_k + \Delta s_k \cos(\psi_k) \\ y_{k+1} &= y_k + \Delta s_k \sin(\psi_k) \\ \psi_{k+1} &= \psi_k + \delta_k \end{aligned} \quad (8)$$

where $x_k$ and $y_k$ denote the global position of the vehicle's center of gravity at step $k$ of the MPC, and $\psi_k$ denotes the heading angle of the vehicle with respect to the global x-axis. The vehicle's displacement in the x-y plane at step $k$ is denoted by $\Delta s_k$. Note that the MPC uses radial steps as shown in figure 6 as they resemble the horizon shape of most onboard perception sensors.

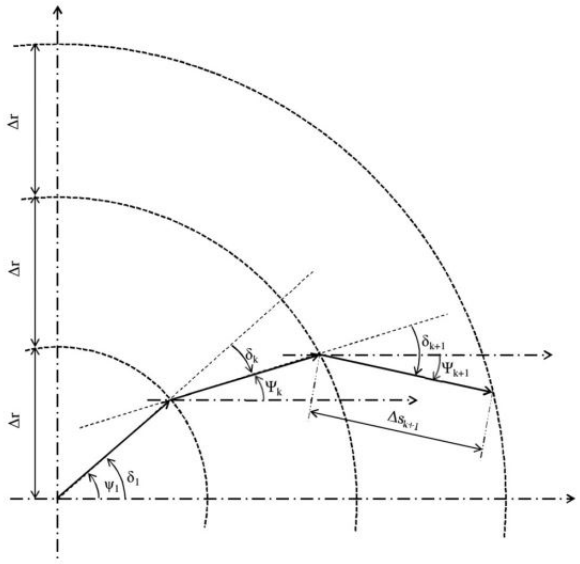


Fig. 6. Radial Steps in MPC

The term $\delta_k$ in equation 8 represents the steering input and should ideally be the control variable. However, we see that the states $x$ and $y$ depend on the heading angle $\psi$ and not on $\delta$. Therefore, we treat $\psi$ as the control input for our MPC while defining $x$ and $y$ as the states.

$$States : x, y \qquad Input : \psi \tag{9}$$

Nevertheless, it is important to keep the physical limits of the vehicle into account. Therefore, constraints are applied on $\delta_k$ and its rate of change to replicate steering rate limitations:

$$\begin{aligned} |\delta_k| &\leq \delta_{max} \\ |\delta_{k+1} - \delta_k| &\leq \delta_{max}^{rate} \end{aligned} \tag{10}$$

Finally, the MPC uses the following objective function:

$$\begin{aligned} \underset{min\ \psi_1..\psi_n}{J} &= w_1 \sum_{k=1}^{n} \Delta l_k + w_2 \sum_{k=1}^{n} \frac{1}{soil_k} \\ &+ w_3 \sum_{k=1}^{n} |z_k - z_{k-1}| + w_4 \sum_{k=1}^{n} visibility_k \\ &+ w_5 \sum_{k=1}^{n} obstacle_k + J_{DP}^{*}(x_n, y_n) \end{aligned} \tag{11}$$

In the cost function above, *n* represents the length of the prediction horizon in radial steps, $\psi_i$ are the optimization variables, and $w_i$ are the penalty weights for respective terms in the cost function. The terms in the cost function are:

1) $\Delta l_k$ is the 3D distance covered in the $k^{th}$ step, computed as:

$$\Delta l_k = ||\Delta s_k, (z_k - z_{k-1})|| \tag{12}$$

   here $z$ is the elevation and $\Delta S_k$ is the topographic 2D distance.
2) Inverse of the soil computed value is used to ensure the prescribed path has the best possible soil properties.
3) $|z_k - z_{k-1}|$ penalizes the change in elevation to prevent steep paths in one radial step.
4) Penalizing the areas which can be detected by adversarial towers; 0 for invisible, and 1 for visible.
5) To ensure local path generated avoids collision with obstacles they are penalized heavily. For any obstacle *O*, let *A(O)* denote the circular area surrounding the obstacle centered at the geometric center of the obstacle whose radius is equal to the defined safe distance. The cost is assigned 0 if $(x_k, y_k)$ position of the vehicle does not lie in *A(O)*, else a cost of 1 is assigned.
6) The last term $J_{DP}^{*}(x_n, y_n)$ denotes the optimal cost-to-go provided by the global path planning dynamic program.

### *B. Implementation Details*

The local path planner MPC has been implemented using the *fmincon* function in MATLAB. The various challenges, implementation choices, and improvements over the original work[1] can be summarized as follows:

1) **High-resolution maps:** For this work, only coarse 50 m step low-resolution terrain maps were available, necessitating interpolation to obtain high-resolution data. The *interp2* function in MATLAB acts as a handy tool but comes with a high computational cost. The *tic/toc* functions indicated *interp2* take up to a second. When used every time the cost function is called, with multiple calls by *fmincon*, and more than 15,000 iterations of *fmincon*, this approach becomes infeasible. Therefore, a high-resolution map of 10,000 x 10,000 grid size should be interpolated once from the 200 x 200 map and indexed every time information is needed. Furthermore, for soil and visibility data, the *round* Matlab function should be used on the interpolated matrices to adhere to the rating system.
2) **Initial control input:** *fmincon* convergence time can be improved by initializing the optimization variable carefully. To do so, we determined the set of possible terminal coordinates ($x_N$,$y_N$) at the last radial step of the prediction horizon, removed out-of-bound coordinates, and determined the ($x_N$,$y_N$) with minimum cost-to-go. Finally, we set:

$$\psi_1 ... \psi_n = \frac{y_N - y_0}{x_N - x_0} \tag{13}$$

3) **Scaling local and global costs:** The objective function in equation 11, proposed by the original work[1], cannot balance between the local and global planners if the weights $w_k$ are constant throughout the journey. It is important to note that the local horizon cost terms evaluate graph edge costs, while the terminal cost term $J_{DP}^{*}$ is an accumulated measure. Consequently, at the beginning of the journey, $J_{DP}^{*}$ (around 50,000 numerically for our choices of $w_k$) far outweighs the local horizon costs (around 1,000), thereby rendering the MPC useless. The contrary scenario happens near the destination where the edge costs start outweighing the global cost-to-go, risking the overall mission to reach the destination. Therefore, we propose and implemented an improvement where the sum of local costs is scaled to a factor (like 40%) of the global cost. This would ensure that both local and global costs are considered at all times.
4) **Out-of-bound coordinate cost:** Accessing information map matrices for out-of-bound coordinates results in Matlab run-time error. To prevent this, we initially returned infinite cost to *fmincon* if the intermediate control trajectory lead the vehicle out-of-bounds. However, this is not a good approach and can result in *fmincon* getting stuck at a local minima. Therefore, we assigned a scaled cost of order $10^2$ to $10^4$ higher than other costs to out-of-bound coordinates.
5) **Topographic distance:** The expression presented in the original work[1] for the 2D topographic distance $\Delta s_k$ in equation 12 was found to be incorrect. The required $\Delta s_k$ can be correctly computed as follows:

$$\Delta s_k = \sqrt{(2k+1)(\Delta r)^2 + \Gamma_k^2} - \Gamma_k \tag{14}$$

where,

$$\Gamma_k = (x_k - x_0)cos\psi_k + (y_k - y_0)sin\psi_k \quad (15)$$

Note that only the expression for $\Gamma_k$ differs from the original paper. The Appendix presents a proof for the reader for equation 15. Furthermore, it is important to note that the index $k = 0$ and not 1 for the first step of the MPC. The radial step radius $\Delta r = 1$ meter.

6) **Inequality Constraints:** The input constraints on $\delta$ and $\delta^{rate}$ in equation 10 can be provided to *fmincon* in terms of the inequality constraint matrices $A$ and $b$ as follows:

$$A1_{N_p-1*N_p} = \begin{bmatrix} -1 & 1 & 0 & ... & 0 & 0 \\ 0 & -1 & 1 & ... & 0 & 0 \\ : & : & : & ... & 0 & 0 \\ : & : & : & ... & 0 & 0 \\ 0 & 0 & 0 & ... & -1 & 1 \end{bmatrix}$$

$$A2_{N_p-1*N_p} = \begin{bmatrix} 1 & -1 & 0 & ... & 0 & 0 \\ 0 & 1 & -1 & ... & 0 & 0 \\ : & : & : & ... & 0 & 0 \\ : & : & : & ... & 0 & 0 \\ 0 & 0 & 0 & ... & 1 & -1 \end{bmatrix}$$

$$A3_{N_p-2*N_p} = \begin{bmatrix} 1 & -2 & 1 & 0 & ... & 0 & 0 \\ 0 & 1 & -2 & 1 & ... & 0 & 0 \\ : & : & : & ... & 0 & 0 & 0 \\ : & : & : & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & ... & 1 & -2 & 1 \end{bmatrix}$$

$$A4_{N_p-2*N_p} = \begin{bmatrix} -1 & 2 & -1 & 0 & ... & 0 & 0 \\ 0 & -1 & 2 & -1 & ... & 0 & 0 \\ : & : & : & ... & 0 & 0 & 0 \\ : & : & : & ... & 0 & 0 & 0 \\ 0 & 0 & 0 & ... & -1 & 2 & -1 \end{bmatrix}$$

$$A_{((2*(N_p-1)+2*(N_p-2))*1)} = \begin{bmatrix} A1 \\ A2 \\ A3 \\ A4 \end{bmatrix}$$

$$b_{((2*(N_p-1)+2*(N_p-2))*1)} = \begin{bmatrix} [\delta_{max}]_{2(N_p-1)*1} \\ [\delta^{rate}_{max}]_{2(N_p-2)*1} \end{bmatrix}$$

7) **MPC Parameters:** The radial prediction horizon $Np = 25$ for our experiments. This results in a diametric coverage of 50 meters, matching the distance between consecutive global grid points. The constraints in equation 10 use $\delta_{max} = \pi/3$ radians. Assuming a constant velocity of 30 mph and 1-meter step size, we calculated $\delta^{rate}_{max} = 3\delta_{max}/20$.

The cost weights in equation 11 are set to $w_1 = 2$, $w_2 = 7.5$, $w_3 = 10$, $w_4 = 1000$, and $w_5 = 50$, utilizing the same ratios as the DP while heavily penalizing obstacles through $w_4$.

## V. Results and Discussions

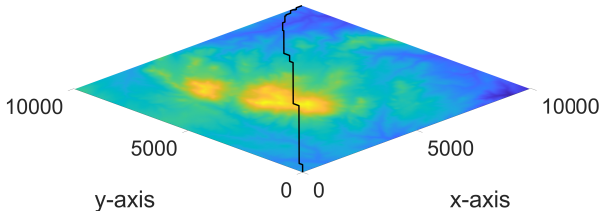


Fig. 7. Global path based on distance costs

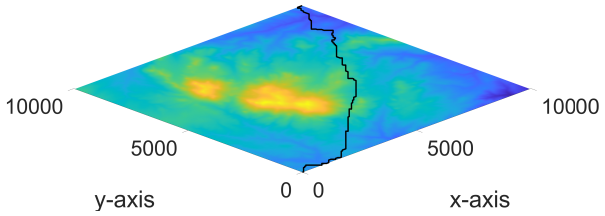


Fig. 8. Global path based on soil profile costs

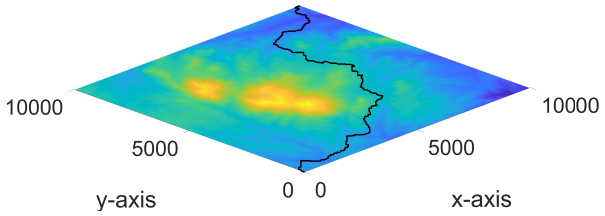


Fig. 9. Global path based on elevation change costs

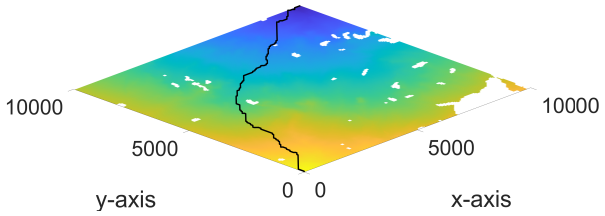


Fig. 10. Global path based on distance, soil, elevation change, and visibility profiles

The Dijkstra-based dynamic program generates an optimal cost-to-go map and global trajectory based on various terrain information layers. Plots of generated global paths pertaining to different cost maps have been presented to understand the impact of adding information sources.

Figure 7 demonstrates Dynamic Programming based global path when only distance costs are considered. We can see that the planner recommends a nearly linear path between the start and end points, resulting in the shortest

3-dimensional distance. Figure 8 shows the global path corresponding to only soil profile costs. It can be seen that this global path is very different from the distance-penalizing global path shown in Figure 7. Figure 9 shows the global path when dynamic programming penalizes only elevation changes. We can see that the planner is avoiding high-elevation areas.

Finally, figure 10 shows the global path obtained when the dynamic programming penalizes distance, soil trafficability, elevation change, and visibility maps as discussed in section III with the provided cost weights. The plotted global paths were obtained by doing a forward pass on the cost-to-go maps.

Furthermore, our computationally efficient approach leveraging matrix operations in Matlab computed the Dijkstra-based optimal cost-to-go map for a grid size of 200 x 200 (or 40,000 vertices) in under 7 seconds. To visualize the impact of the MPC, we artificially

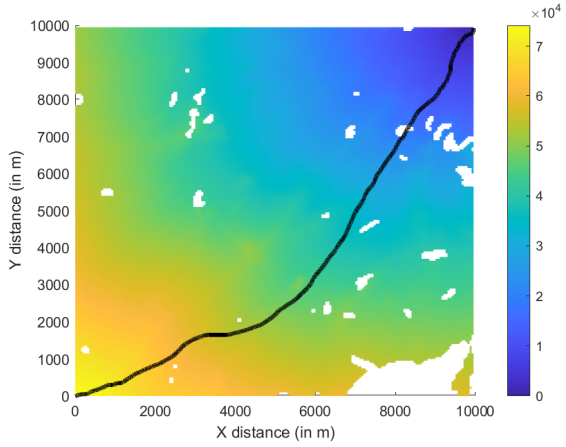


Fig. 11. Vehicle trajectory based on the hierarchical framework

added high costs (representing dynamic local obstacles, for example) in the high-resolution interpolated cost map to differentiate it from the available data. Note that the coarse data maps were untouched, thereby not affecting the global path recommendations. The trajectory followed by the vehicle by using the hierarchical approach is shown in figure 11. Even though the final trajectory did not exactly follow the global path, the result was guided by the Dynamic Programming routine to satisfy the mission scenario of reaching the destination while addressing the dynamic surrounding changes. From a computational efficiency point-of-view, our *fmincon* based MPC took less than 0.05 seconds to converge for each vehicle step, thereby proving the real-time applicability of the hierarchical approach.

## VI. Appendix

a) The proof for the expression for $\Gamma_k$ in equation 15 has been provided here. The coordinates at step $(k+1)$ lie on a circle centered at the initial coordinates, thereby giving us:

$$(x_{k+1} - x_0)^2 + (y_{k+1} - y_0)^2 = (k+1)^2 \Delta r^2 \quad \text{(A1)}$$

Rewriting equation A1 by substituting model states:

$$\begin{aligned} &((x_k - x_0) + \Delta s_k \cos\psi_k)^2 \\ &+ ((y_k - y_0) + \Delta s_k \sin\psi_k)^2 \\ &= (k+1)^2 \Delta r^2 \end{aligned} \quad \text{(A2)}$$

which results in

$$k^2 \Delta r^2 + (\Delta s_k)^2 + 2\Delta s_k \Gamma_k = (k+1)^2 \Delta r^2 \quad \text{(A3)}$$

where $\Gamma_k$ = $(x_k$ - $x_0)\cos\psi_k$ + $(y_k$ - $y_0)\sin\psi_k$. Solving above quadratic equation for $\Delta s_k$ we get

$$\Delta s_k = \sqrt{(2k+1)(\Delta r)^2 + \Gamma_k^2} - \Gamma_k \quad \text{(A4)}$$

b) The code for this project can be accessed on GitHub[7].

## VII. Contributions

Shubham contributed to the implementation of the MPC and the DP, their integration, and wrote the implementation details and results in the report. Shubhankar contributed to implementing the DP algorithm and experimenting with the MPC tuning. Priyanshu contributed towards problem formulation, researching into solving the same problem through linearizing the system and experimenting with implementing MPC and DP. All three authors contributed to generating the data required for the experiment, tuning the MPC to achieve the desired output, and formulating this report.

## References


[1] Roy, J., Wan, N., Goswami, A., Vahidi, A., Jayakumar, P., and Zhang, C. (February 13, 2018). "A Hierarchical Route Guidance Framework for Off-Road Connected Vehicles." ASME. J. Dyn. Sys., Meas., Control. July 2018; 140(7): 071011. https://doi.org/10.1115/1.4038905

[2] Sniedovich, M., 2006, Dijkstra's Algorithm Revisited: The Dynamic Programming Connexion, Control Cybern., 35(3), p. 599.http://matwbn.icm.edu.pl/ksiazki/cc/cc35/cc3536.pdf

[3] Z. Dong, X. Xu, X. Zhang, X. Zhou, X. Li and X. Liu, "Real-time Motion Planning Based on MPC With Obstacle Constraint Convexification For Autonomous Ground Vehicles," 2020 3rd International Conference on Unmanned Systems (ICUS), 2020, pp. 1035-1041, doi: 10.1109/ICUS50048.2020.9274881.

[4] D. Savitski, P. Nedoma, J. Machan, J. Plihal, V. Ivanov and K. Augsburg, "Cost functions for assessment of vehicle dynamics," 2013 IEEE Symposium on Computational Intelligence for Engineering Solutions (CIES), 2013, pp. 48-55, doi: 10.1109/CIES.2013.6611728.

[5] Ahmad, Mohiuddin Polotski, Vladimir Hurteau, Richard. (2000). Path tracking control of tracked vehicles. Proceedings - IEEE International Conference on Robotics and Automation. 3. 2938 - 2943 vol.3. 10.1109/ROBOT.2000.846474.

[6] Kronqvist, J., Bernal, D.E., Lundell, A. et al. A review and comparison of solvers for convex MINLP. Optim Eng 20, 397455 (2019). https://doi.org/10.1007/s11081-018-9411-8

[7] https://github.com/shubham6531/Hierarchical_Off-Road_Path_Planner.git