DESIGN AND IMPLEMENTATION OF A MOTION PLANNING ALGORITHM FOR OFF-ROAD VEHICLES ON ROUGH TERRAIN

AU5712 PROJECT l REPORT

Submitted by

JAIKISHAN S (2020502018)

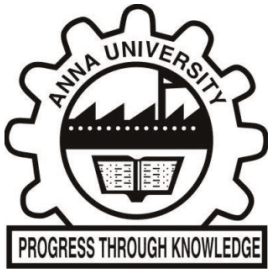
LALITHA M (2020502021)

SUGANRAJ P (2020502544)

BACHELOR OF ENGINEERING

in

AUTOMOBILE ENGINEERING



DEPARTMENT OF AUTOMOBILE ENGINEERING

MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY-MIT CAMPUS

CHENNAI-600 044

DECEMBER 2023

ABSTRACT

Autonomous vehicles, also known as self-driving cars, are vehicles that can operate without human input. They use a variety of sensors, including cameras, radar, and lidar, to perceive their surroundings and make decisions about how to navigate. The main objective of them is to be able to navigate and drive to the desired location without any driver input. This paper presents the implementation of a motion planning algorithm for an autonomous navigation system. The algorithm is designed to enable a vehicle to autonomously navigate through a known or unknown environment while avoiding obstacles and reaching a specified goal.

This project on the design and implementation of a novel motion planning algorithm tailored for off-road vehicles traversing rough and unpredictable terrains. The proposed algorithm integrates perception, decision-making, and control strategies to enable robust navigation in complex environments. Moreover, the A* star algorithm employs advanced path planning methodologies, considering the terrain's irregularities and obstacles to compute optimal or nearoptimal paths while ensuring vehicle stability and safety. The integration of localization and mapping algorithms facilitates precise positioning and mapping of the vehicle's surroundings, aiding in accurate decision-making and finding the shortest path.

ABBREVATION

ADC	Analog Digital Converter
ADAS	Advanced Driver Assistance System
ANS	Autonomous Navigation System
AV	Autonomous Vehicle
FMCW	Frequency Modulated Continuous Wave
GPS	Global Positioning System
INS	Inertial Navigation System
MMP	Maximum Margin Planning
OHV	Off-Highway Vehicle
ORV	Off-Road Vehicle
POI	Point of Interest
RADAR	Radio Detection and Ranging
ROS	Robot Operating System
TEB	Timed Elastic Band
UAV	Unmanned Ground Vehicle
VIL	Vehicle In Loop

INTRODUCTION

In the realm of autonomous systems, the ability to navigate seamlessly through unknown environments stands as a pillar of true independence. To achieve this remarkable feat, we turn to the art of motion planning, a computational dance that orchestrates the delicate balance between efficient pathfinding and meticulous obstacle avoidance. In this paper, we delve into the implementation of a motion planning algorithm that merges the elegance of the A* algorithm with the precision of RADAR sensing, forging a path towards autonomous navigation that is both intelligent and perceptive. A* algorithm stands as a cornerstone of motion planning. Its brilliance lies in its ability to chart the most promising path from a starting point to a desired destination, navigating through a maze of obstacles with both efficiency and foresight. By meticulously evaluating the cost of each potential step, A* relentlessly pursues the optimal route, promising a journey that is both swift and secure.

RADAR emerges as the vigilant eye, scanning the surroundings with unwavering precision. Emitting radio waves that ricochet off objects, RADAR paints a detailed portrait of the environment, revealing the location, distance, and velocity of obstacles that dare to obstruct the path. This invaluable information empowers the motion planning algorithm to make informed decisions, ensuring a safe and obstacle-free journey. RADAR diligently sketches a map of the environment, pinpointing obstacles and defining the boundaries of navigable terrain.

A* meticulously analyzes this map, identifying the shortest path that steers clear of hazards, prioritizing both efficiency and safety. As the autonomous system ventures forth, RADAR continuously scans for unexpected obstacles or changes in the environment. In response, A* dynamically adapts the path, ensuring the system gracefully navigates unforeseen challenges. Through this harmonious blend of algorithmic intelligence and sensor-driven perception, we empower autonomous systems to traverse their surroundings with unprecedented autonomy. This work offers a tangible pathway towards a future where vehicles, robots, and other intelligent machines navigate confidently through complex environments, unburdened by the need for human intervention.

Motion planning for off-road vehicles presents a significant challenge due to the unstructured and unpredictable nature a tough challenge for vehicles. To tackle this, we've developed a unique motion planning algorithm for off-road vehicles, blending real-time sensor data, terrain analysis, and path planning techniques. Our algorithm relies on a range of sensors, like radar and cameras, to understand the surroundings and spot potential obstacles. It then analyzes the terrain to figure out safe paths and avoid risky areas. Ultimately, it crafts a collision-free and efficient path for the vehicle, considering both its dynamics and environmental constraints. Our project focuses on creating a motion planning algorithm for autonomous off-road vehicles using radar sensors, cameras, and an A* approach to find the optimal path. We've tested the proposed algorithm through simulations and real-world experiments, showcasing its ability to navigate challenging off-road terrains effectively while prioritizing safety and performance.

Our aim is to design and implement a motion planning algorithm for off-road vehicles on rough terrain using MATLAB. By finding the shortest path to a given set of start and end points namely waypoints using A* Path planning algorithm in a rough terrain. The primary main goal of our project is to design and implement a motion planning algorithm for autonomous off-road vehicles in rough terrain. This can be done by finding the shortest path in unpredictable environment using A* algorithm which also includes obstacle avoidance using RADAR sensors and cameras. And also to utilize the full potential, that are required to operate in areas with full of obstacles such as in rough terrain.

Further we also aimed to increase the planning time for quicker response. This will result in finding the more accurate shortest path in the unstructured such as hill slope environment. For obstacle avoidance using RADAR sensor and camera that resulted in tracking the object in all the sides of the ego vehicle by mounting four RADAR sensor and two cameras in the ego vehicle.

IMPLEMENTATION OF PATH PLANNING

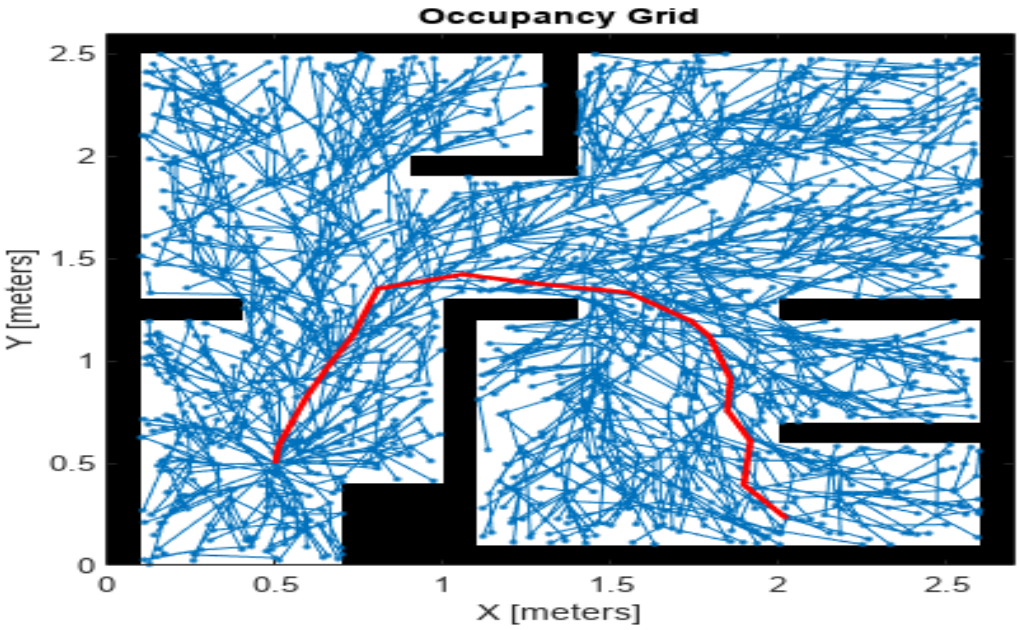
The motion planning is to plan a path through an environment by using common sampling-based planners like RRT, RRT*, and Hybrid A*, or specify your own customizable path-planning interfaces. Use path metrics, state space sampling, and state validation to ensure the path is valid and has proper obstacle clearance or smoothness. Follow the path and avoid obstacles using pure pursuit, vector field histogram (VFH), and timed elastic band (TEB) algorithms.

RRT* PLANNER:

The planner RRT* object creates an asymptotically-optimal RRT planner. The RRT* algorithm converges to an optimal solution in terms of the state space distance. Also, its runtime is a constant factor of the runtime of the RRT algorithm. RRT* is used to solve geometric planning problems. A geometric planning problem requires that any two random states drawn from the state space can be connected. The planner sets properties using one or more name-value arguments in addition to the input arguments in the syntax. It can specify the StateSampler, ContinueAfterGoalReached,MaxConnectionDistance, *GoalReachedFcn*, and *GoalBias* properties as name-value arguments. State space for the planner, specified as a state space object. We can use state space objects such as *stateSpaceSE2*, and *stateSpaceSE3*. customize a state space object using the *nav.StateSpace* object.

Create a state space. Create an occupancyMap-based state validator using the created state space. Create an occupancy map from an example map and set map resolution as 10 cells/meter. Set validation distance for the validator. Update state space bounds to be the same as map limits. Create RRT* path planner and allow

further optimization after goal is reached. Reduce the maximum iterations and increase the maximum connection distance. Set the start and goal states. Plan a path with default settings. The visualize of the result is given below,



RRT* planner

If the environment is well-known and static, A* is often a good choice. If it's unknown or dynamic, RRT* is more suitable. In path optimality, if finding the shortest or most efficient path is critical, A* is preferable. If quick path generation and exploration are more important, RRT* is better. In computational resources, A* is generally more efficient for simple environments, while RRT* can be more computationally intensive. Our primary main aim is to find the shortest path and also more efficient in the unstructured environment so we move to the A* algorithm to find the shortest path in the unstructured environment.

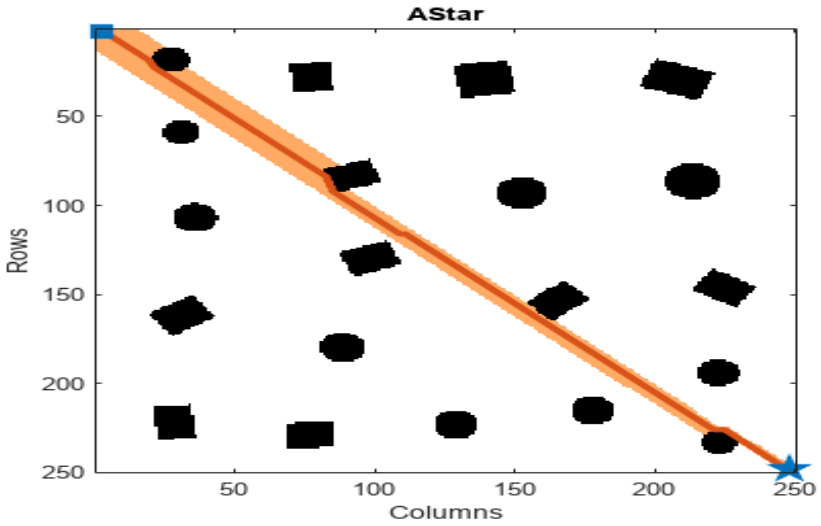
PLANNER

A* Search algorithm is one of the best and popular technique used in pathfinding and graph traversals.It is a searching algorithm that is used to find the shortest path between an initial and a final point. It is a handy algorithm that is often used for map traversal to find the shortest path to be taken. A* was initially designed as a graph traversal problem, to help build a robot that can find its own course. It still remains a widely popular algorithm for graph traversal. It searches for shorter paths first, thus making it an optimal and complete algorithm. An optimal algorithm will find the least cost outcome for a problem, while a complete algorithm finds all the possible outcomes of a problem. Another aspect that makes A* so powerful is the use of weighted graphs in its implementation. A weighted graph uses numbers to represent the cost of taking each path or course of action. This means that the algorithms can take the path with the least cost, and find the best route in terms of distance and time

The *plannerAStarGrid* object creates an A* path planner. The planner performs an A* search on an occupancy map and finds shortest obstacle-free path between the specified start and goal grid locations as determined by heuristic cost. The planner creates a *plannerAStarGrid* object with a *binaryOccupancyMap* object using a width and height of 10 meters and grid resolution of 1 cell per meter. the planner creates a *plannerAStarGrid* object using the specified map object map. Specify map as either a *binaryOccupancyMap* or occupancyMap object. The map input sets the value of the Map property.

The planner sets properties using one or more name-value pairs. Unspecified properties have default values. For example, *plannerAStarGrid* creates an A* path planner object using the Manhattan cost function. Plan the shortest collision-free path through an obstacle grid map using the A* path planning algorithm. Generate

a *binaryOccupancyMap* object with randomly scattered obstacles using the *mapClutter* function. Use the map to create a *plannerAStarGrid* object. Define the start and goal points. Plan a path from the start point to the goal point. Visualize the path and the explored nodes in the given blow figure.



A* Planner

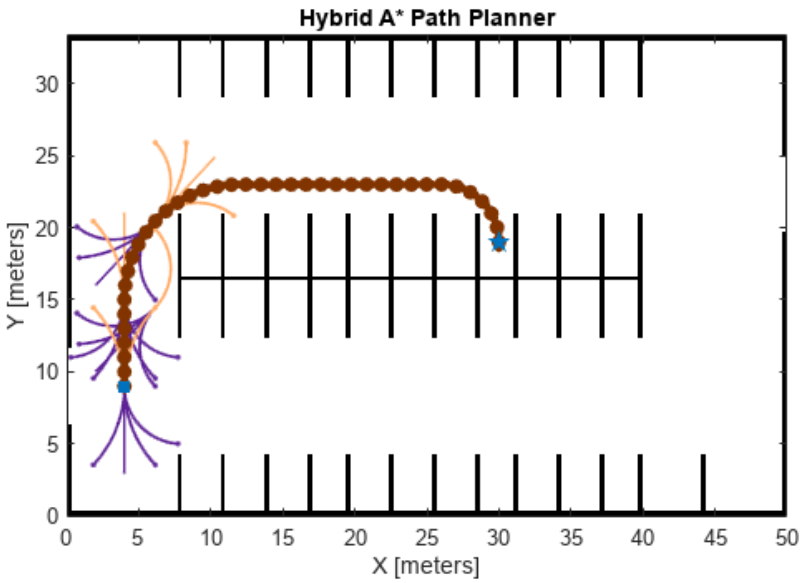
HYBRID A* PLANNER

The Hybrid A* algorithm is a powerful pathfinding technique that combines the strengths of both A search* in continuous space and the efficiency of discretized search. This makes it ideal for finding paths for nonholonomic vehicles like cars and robots that can't make sharp turns or move sideways.

The Hybrid A* path planner object generates a smooth path in a given 2-D space for vehicles with nonholonomic constraints. The plannerHybridAStar object uses the Reeds-Shepp connection to find an obstacle-free path. This can modify the behavior of the connection by tuning properties like MinTurningRadius, ForwardCost, and ReverseCost. And can use the Analytic Expansion Interval property to set the cycle to check for the Reeds-Shepp connection. the

planner creates a path planner object using the Hybrid A* algorithm. Specify the validator input as a validator Occupancy Map or validator Vehicle Costmap object. The validator input sets the value of the State Validator property.

The planner sets Properties of the path planner by using one or more name-value pair arguments. Plan a collision-free path for a vehicle through a parking lot by using the Hybrid A* algorithm. Load the cost values of cells in the vehicle costmap of a parking lot. Create a binaryOccupancyMap with cost values. Create a state space. Update state space bounds to be the same as map limits. Create a state validator object for collision checking. Assign the map to the state validator object.I nitialize the plannerHybridAStar object with the state validator object. Specify the MinTurningRadius and MotionPrimitiveLength properties of the planner. Define start and goal poses for the vehicle as [*x*, *y*, *theta*] vectors. *X* and *y* specify the position in meters, and *theta* specifies the orientation angle in radians. Plan a path from the start pose to the goal pose.



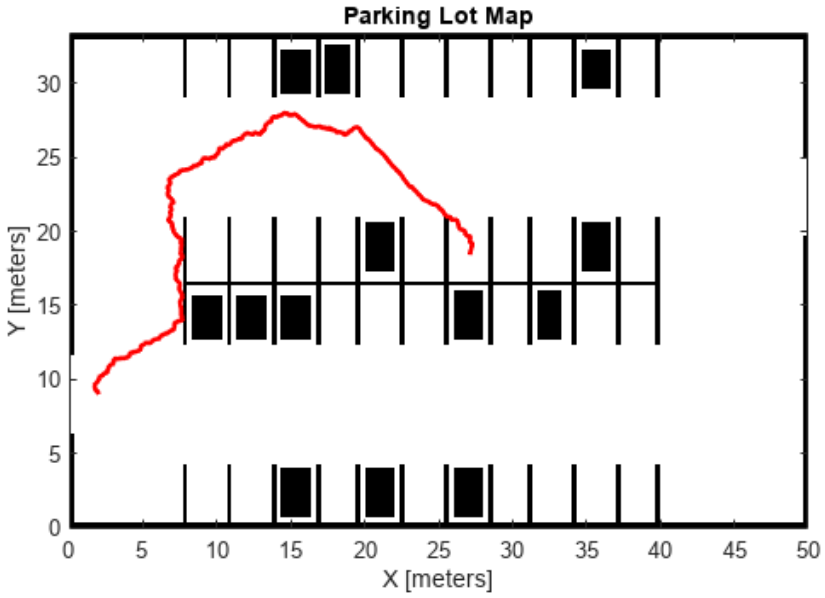
Hybrid A* planner

CONTROLER TEB

The controllerTEB object creates a controller (local planner) using the Timed Elastic Band (TEB) algorithm. The controller enables a robot to follow a reference path typically generated by a global planner, such as RRT or Hybrid A*. Additionally, the planner avoids obstacles and smooths the path while optimizing travel time, and maintains a safe distance from obstacles known or unknown to the global planner. The object also computes velocity commands and an optimal trajectory using the current pose of the robot and its current linear and angular velocities. The controller creates a TEB controller object, controller, that computes the linear and angular velocity commands for a differential-drive robot to follow the reference path and travel for 5 seconds in an obstacle-free environment. The reference path input sets the value of the Reference Path property. the controller attempts to avoid obstacles in the specified occupancy map map. The controller assumes the space outside the map boundary is free. The map input sets the value of the Map property. The controller specifies properties using one or more name-value arguments in addition to any combination of input arguments from the previous syntaxes.

Create an occupancyMap object from a parking lot map and set the map resolution to 3 cells per meter. Visualize the map. The map contains the floor plan of a parking lot with some parking slots already occupied. Create a validatorOccupancyMap state validator using the stateSpaceSE2 definition. Specify the map and the distance for interpolating and validating path segments. Set the start and goal states. Plan a path with default settings. RRT* uses a random orientation, which can cause unnecessary turns. Align the orientation to the path, except for at the start and goal states. Create a local occupancyMap object with a width and height of 15 meters and the same resolution as the global map. Create

a controllerTEB object by using the reference path generated by the global planner and the local map. Specify the properties of the controllerTEB object. Create a deep clone of the controllerTEB object. Initialize parameters. Compute velocity commands and optimal trajectory.



Sample parking lot map using Controller TEB

3.5 GRAPH-BASED ROUTE PLANNER:

The terrain is represented as a point cloud. Sites like this are interesting because they offer a mix of structured and unstructured terrain, meaning may want to apply different planning techniques depending on the characteristics of the terrain or problem want to solve. This uses a 38 MB pointcloud, generated via aerial RADAR. The next is to convert the raw point cloud to a digital elevation model using the helper. This function to sample the initial elevation over the XY limits of the pointcloud. Then, to fill missing regions of the elevation matrix and lastly compute the gradient.

With the slope information identified, separate the terrain into traversable and impassable zones by setting a maximum allowable slope and comparing this against the gradients computed earlier. This gradient mask can be used to restrict our navigable space and does a decent job of highlighting the switchbacks. Our end goal is to generate a navigation graph that can be used to plan high-level routes, and while the raw gradient mask could be used, the noisy image can lead to disconnected edges, or a large number of paths weaving through tight spaces. To produce a more sensible road-network, apply several morphological operations to the mask:

1) To fill in small gaps between max-slope regions, producing more consistent off-limit boundaries.
2) To eliminate small pockets that would be disconnected from the large free regions in of the map, and therefore useless to our route-planner.
3) To identify and remove small pockets of above-average slope. This will make the roads follow the center of major free regions, rather than splitting into multiple paths around small bumps in the terrain.
4) The remaining mask, solidifying any remaining boundaries

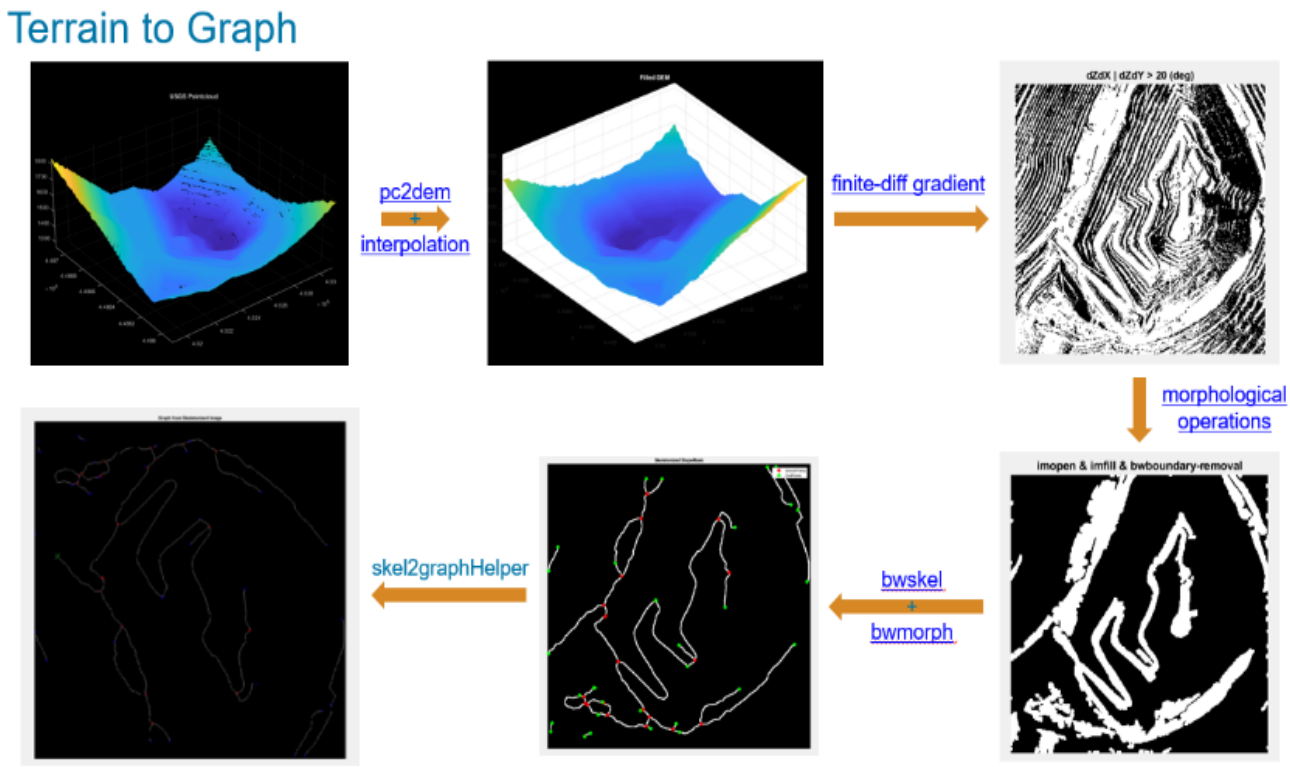
Having reduced our terrain to a skeleton, extract the individual branches from the image. This done by starting at each end-point and visiting the next free neighbor until another end-point or branch-point is found. The same process is applied to lines originating from the branch-points, with the cells visited along each edge recorded in a corresponding element of a struct-array.

Convert Edges to navigation graph

The last is to convert our set of edges to a navigation graph, which can be used to return shortest paths between nodes in our road network. The helper, takes in the

list of edges generated by skeleton edges and converts it to a set of connected nodes, as well as the edges which connect them. The navGraph also allows for custom edge-costs, so below you will compute the length along the dense path, and store these as Weight in the sparse edges of the graph. Custom cost functions could also consider things like elevation, slope, or even dynamic congestion in cases where the graph is being used to route multiple agents throughout the mine.

This allows you to fully reconstruct the path between nodes after a route is found. For this particular route planner, you will use this to store Edge2PathIdx - a mapping between the edges in our graph and the set of dense paths in cached Path. This allows you to fully reconstruct the path between nodes after a route is found



Graph based route planner- Terrain to Graph

A TERRAIN-AWARE PLANNER

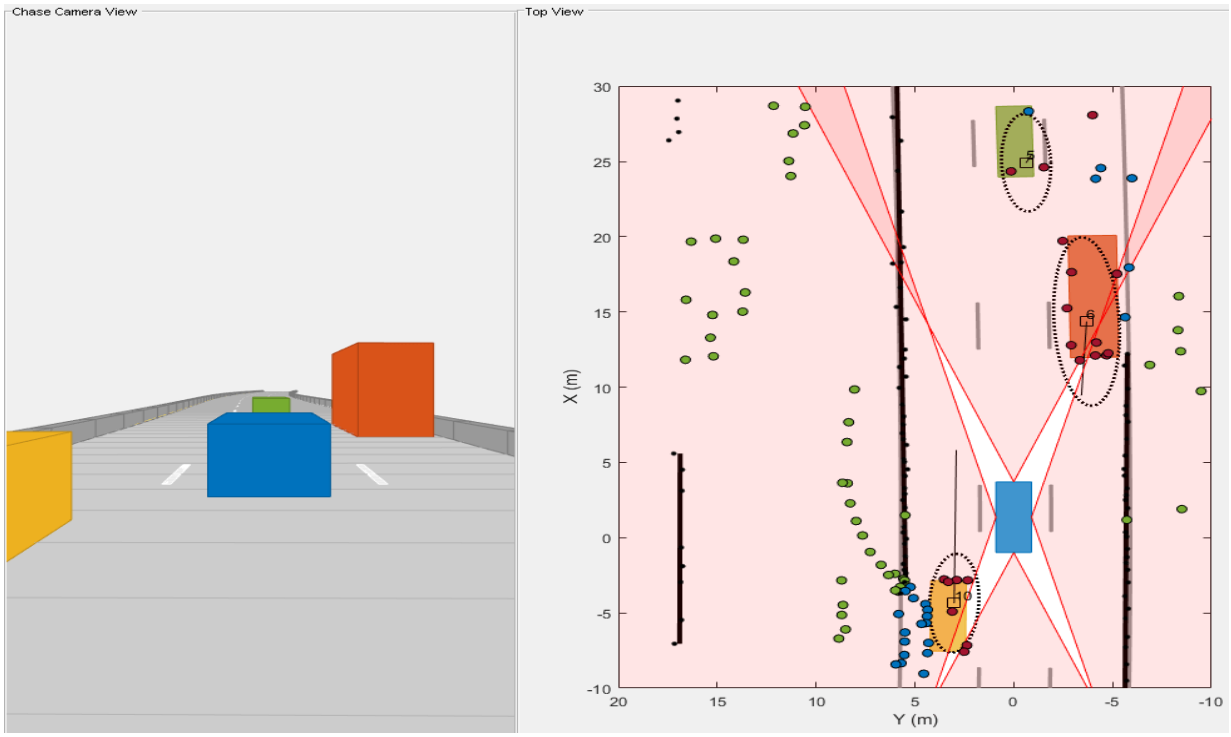
In the previous, a graph-based planner was developed to plan routes through a road network derived from the offroad elevation data. In this section, develop a simple onramp planner to help connect the vehicle to the shortest path through the road-network. Later, provide hybrid A* with a custom cost function that allows for terrain-aware planning. This planner can be used in regions of the map that lack structure, or as a fallback planner in the event that the path-following controller fails to find a local solution. Convert the list of XY paths to a navigation graph and place this inside A* star. This will serve as a high-level planner that generates pre-computed routes through the mine.

This should improve the likelihood that any start and goal SE2 poses will have a collision-free path to the nearest node's edges, while keeping the graph relatively sparse. A path through the graph, with start and goal poses lying away from the graph. Note that these poses contain orientation, which will become relevant in the later sections, but the graph only considers XY. Once a path is found, use the metadata stored on the edges of navigation graph to reconstruct the full path.

SENSOR FUSING USING SYNTHETIC RADAR AND VISION DATA FOR OBSTACLE AVOIDANCE (ON-ROAD)

Automotive radar sensors are robust against adverse environment conditions encountered during driving, such as fog, snow, rain, and strong sunlight. Automotive radar sensors have this advantage because they operate at substantially large wavelengths compared to visible-wavelength sensors, such as lidar and camera. As a side effect of using large wavelengths, surfaces around the radar

sensor act like mirrors and produce undesired detections due to multipath propagation. These detections are often referred to as ghost detections because they seem to originate from regions where no target exists. This shows the impact of 25 these multipath reflections on designing and configuring an object tracking strategy using radar detections.



Simulation of RADAR sensor for obstacle avoidance

The animation that follows shows the result of the radar data processing chain. The black ellipses around vehicles represent estimated tracks. The radar detections are visualized with four different colors depending on their predicted classification from the algorithm. The black dots in the visualization represent static radar target detections. Notice that these detections are overlapped by black lines, which represent the static reflector found using the DBSCAN algorithm. The maroon

markers represent the detections processed by the extended object tracker, while the green and blue markers represent radar detections classified as reflections via static and dynamic objects, respectively. Notice that the tracker is able to maintain a track on all four vehicles during the scenario.

Next, analyze the performance of the algorithm using different snapshots captured during the simulation. The snapshot below is captured at 3 seconds and shows the situation in front of the ego vehicle. At this time, the ego vehicle is approaching the slow-moving truck, and the left radar sensor observes reflections of these objects via the left barrier. These detections appear as mirrored detections of these objects in the barrier. Notice that the black line estimated as a 2-D reflector is in the line of sight of these detections. Therefore, the algorithm is able to correctly classify these detections as ghost targets reflected off static objects. Next, analyze the performance of the algorithm using the snapshot captured at 4.3 seconds. At this time, the ego vehicle is even closer to the truck and the truck is approximately halfway between the green vehicle and the ego vehicle. During these situations, the left side of the truck acts as a strong reflector and generates ghost detections. The detections on the right half of the green vehicle are from two-bounce detections off of the green vehicle as the signal travels back to the sensor after reflecting off the truck. The algorithm is able to classify these detections as ghost detections generated from dynamic object reflections because the estimated extent of the truck is in the direct line of sight of these detections.

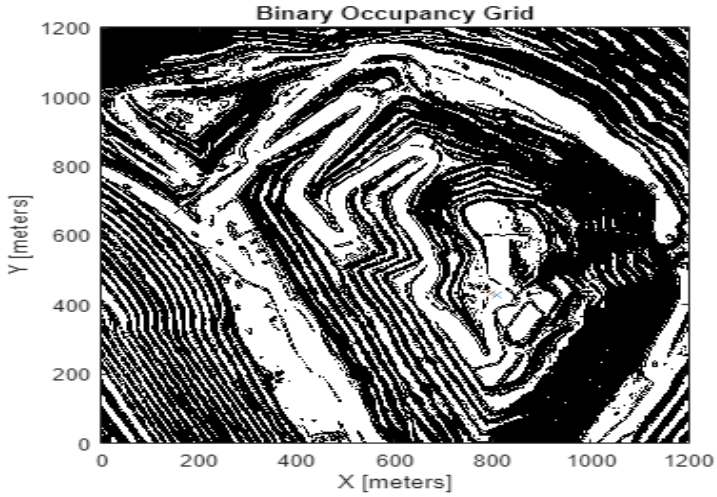
Also notice the passing vehicle denoted by the yellow car on the left of the ego vehicle. The detections, which seem to originate from the nonvisible surface of the yellow vehicle, are two-bounce detections of the barriers, reflected via the front face of the passing vehicle. These ghost detections are misclassified as target detections because they seem to originate from inside the estimated extent of the vehicle. At the same location, the detections that lie beyond the barrier are also two-bounce detections of the front face when the signal is reflected from the barrier and returns to the sensor. Since these detections lie beyond the extent of the track and the track is in the direct line of sight, they are classified as ghost detections from reflections off dynamic objects.

RESULTS AND DISCUSSION

TERRAIN AWARE PLANNER WITH A* ALGORITHM

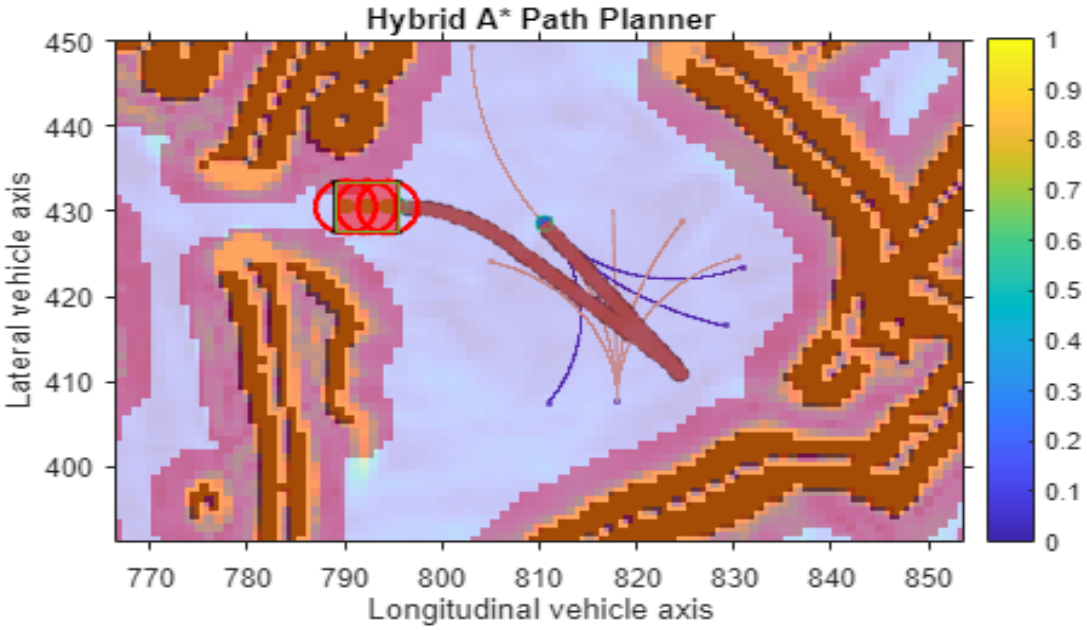
Begin by specifying some parameters needed to represent the geometry and kinematic constraints of the vehicle. These have been roughly separated into a set of parameters that might change throughout the vehicle's operation, like costs, and those that will likely remain fixed, like the length and width of the vehicle. Ensure there is enough room around the previously-computed road network to allow the global planners to operate. To do this, convert the road-network to an occupancy map, inflate it by a desired radius, and then remove those inflated cells from the slope mask.

In this sample terrain, it is ensured that there is enough room around the previously-computed road network to allow the global planners to operate. To do this, convert the road-network to an occupancy map, inflate it by a desired radius, and then remove those inflated cells from the slope mask. Note that in the real world, we may instead want to remove roads that are too narrow to traverse.



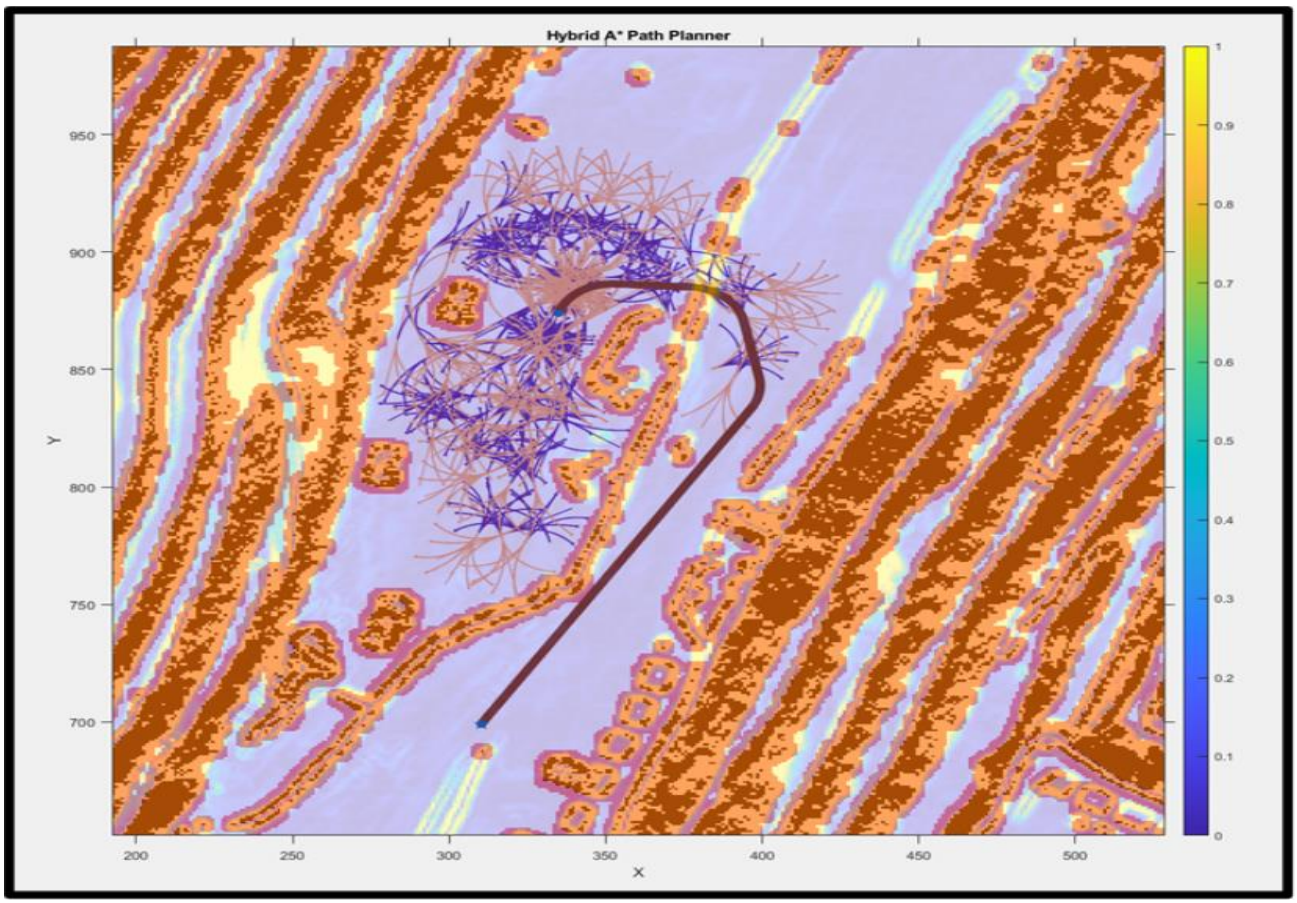
Terrain-Aware Planner with A* algorithm

TERRAIN AWARE PLANNER WITH HYBRID A* ALGORITHM



Terrain-Aware Planner with Hybrid A* algorithm

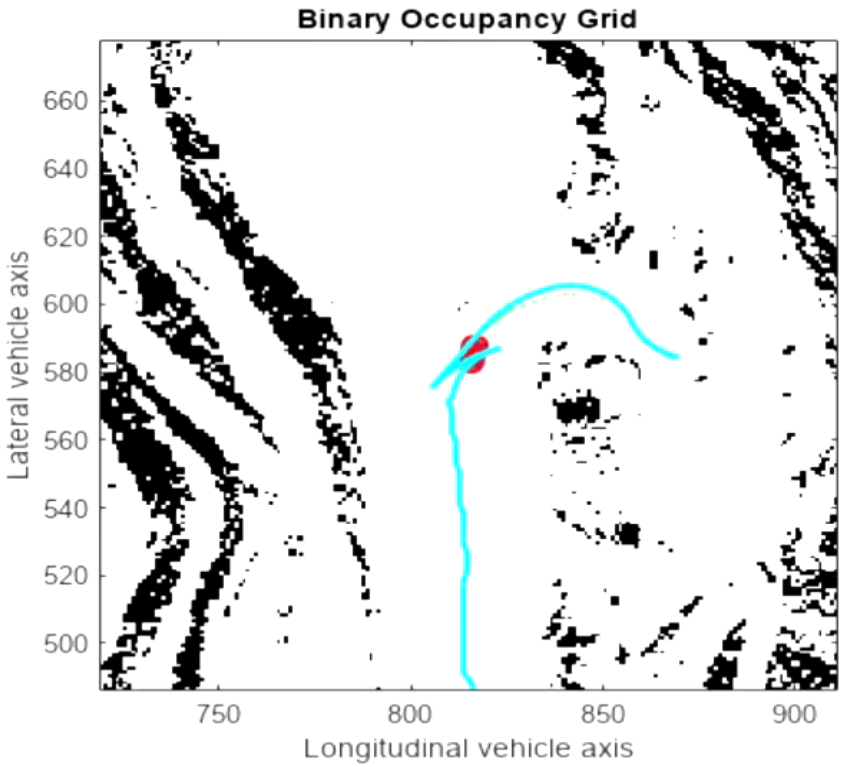
The developed two planners which simply operates in the SE2-space and can be used to form short connections through free space while adhering to your vehicle's non-holonomic constraints. The first of which was the "Onramp Planner", which attempts to connect an SE2 start location to any edges attached to the closest node in the route-planner. The second planner developed using planner Hybrid A*, which utilizes a custom Transition Cost function to penalize motions that move along steep gradients. This planner can always be used, but due to the comparatively large computation costs, it will be used in a fallback capacity when the Onramp Planner is unable to produce paths to/from the road-network, or when the vehicle needs to replan while path-following.



Top view of Terrain-Aware Planner with Hybrid A* algorithm

SIMULATING AUTONOMOUS VEHICLE ON A SAMPLE TERRAIN

In this grid, we developed a controller which attempts to follow a reference path in the sample terrain which is hilly slope region (unstructured environment) while avoiding obstacles using controllerTEB. This example separated the controller properties into two categories - those that one might want to tune during runtime, and those that should be fixed during simulation, either because they are physical properties of the vehicle.

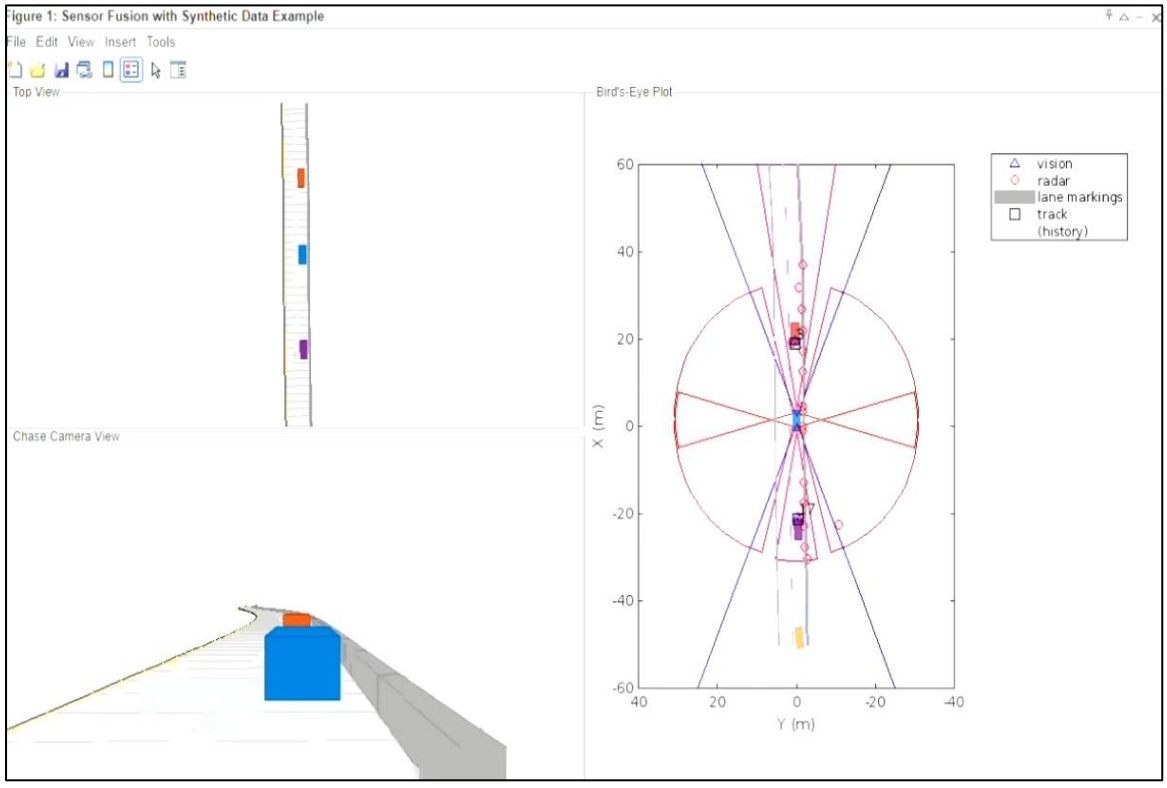


Autonomous vehicle in the hilly slope terrain simulation

Similar to the terrain-aware planner in the previous, should consider separating these parameters into those than might change throughout the vehicle's operation, and those that will likely remain fixed. It is also worth noting here that the parameters of the local planner should be chosen with those of the global planner in mind. As an example, this could either mean keeping shared parameters like the MinTurningRadius identical in both planners, or possibly using a larger radius in the global planner, thereby ensuring that the local controller is able to follow the reference path effectively. In this simulation, the shortest path has been found in the hilly slope unstructured region with the efficient obstacle avoidance using the RADAR sensor, Cameras and Hybrid A* algorithm for finding the shortest path in the ustructured region

4.4 SENSOR FUSING USING SYNTHETIC RADAR AND CAMERA FOR OBSTACLE AVOIDANCE (ON-ROAD)

In this, simulate the multipath detections from radar sensors in an urban highway driving scenario. The highway is simulated with a barrier on both sides of the road. The scenario consists of an ego vehicle and four other vehicles driving on the highway. The ego vehicle is equipped with four radar sensors providing 360-degree coverage and the two cameras placed in the front and rear of the ego vehicle to track the objects for obstacle avoidance. This image shows the configuration of the radar sensors and detections from one scan of the sensors. The tracked regions represent the field of view of the radar sensors.



Simulate sensor detections, and use sensor fusion to track simulated vehicles.

This Program shows a scenario to simulate sensor detections, and use sensor fusion to track simulated vehicles. The main benefit of using scenario generation and sensor simulation over sensor recording is the ability to create rare and potentially dangerous events (on-road) and test the vehicle algorithms with them. This Scenario generation comprises generating a road network, defining vehicles that move on the roads, and moving the vehicles. By this ability of the sensor fusion to track a vehicle that is passing on the left of the ego vehicle. The scenario simulates a highway setting, and additional vehicles are in front of and behind the ego vehicle.

CONCLUSION

We have created and algorithm using A* algorithm to find out the shortest path in the unstructured environment for the autonomous navigation system. This resulted that the algorithm finds the shortest path in the given unstructured environment which is the hill slope environment we have chose. We also created an algorithm using Hybrid A* algorithm which is the extension of A* algorithm which resulted in finding the more accurate shortest path in the unstructured hill slope environment. We also created an algorithm for obstacle avoidance using RADAR sensor and camera that resulted in tracking the object in all the sides of the ego vehicle by mounting four RADAR sensor and two cameras in the ego vehicle. To analyze the efficient function of the created algorithms has to be implemented in the working model that must be run on the unstructured environment to find out the shortest path and also to track the object for obstacle avoidance.

The A* algorithm has proven itself as a powerful and efficient tool for finding the shortest path in autonomous navigation systems. Its ability to combine a heuristic function with the systematic search process allows it to quickly identify the optimal route, even in complex and dynamic environments. This makes it particularly suitable for autonomous vehicles, robots, and other systems that require real-time path planning. This project was limited to small rough terrain environment and can be further extended with additional optimizations such as testing dynamic obstacle avoidance cases and other worst-case scenarios. The outcome of this analysis was an introductory study of autonomous navigation of robots in a rough terrain concept.

METHODOLOGY FOR WORKING MODEL:

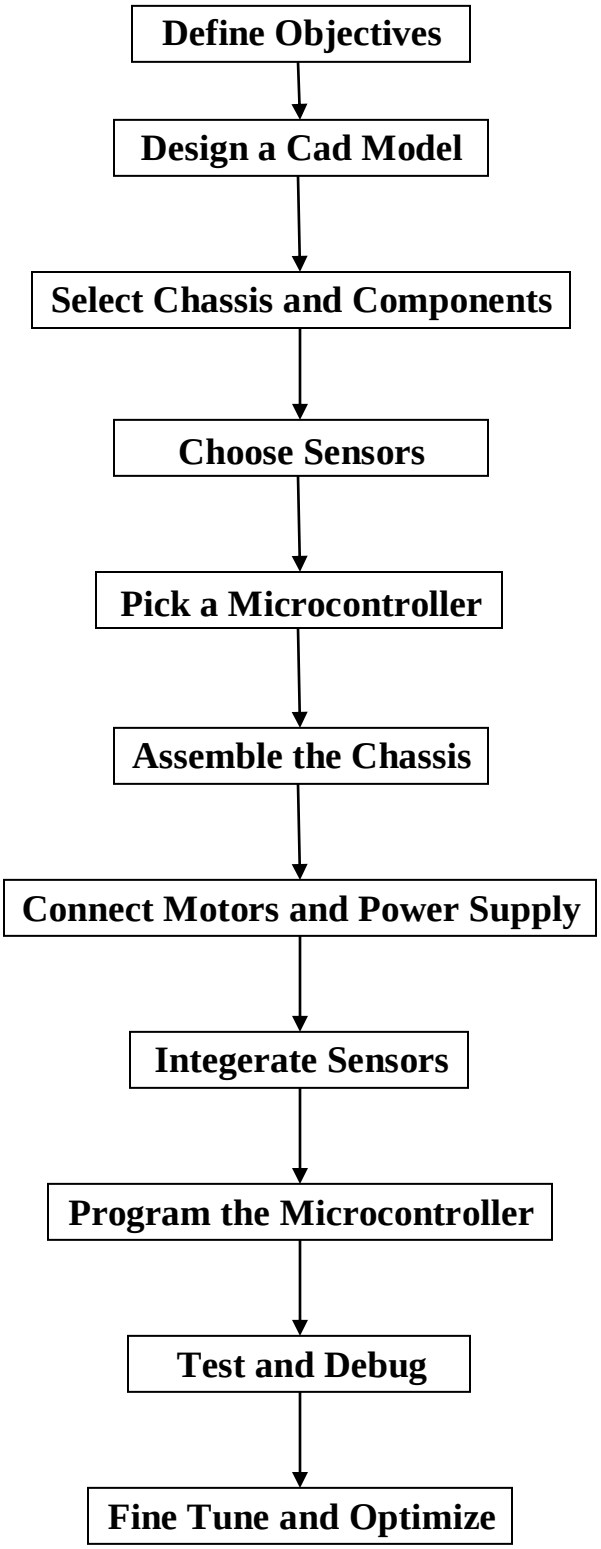


Figure no.6.1:Methodology for working model

In order to implement the above path planning algorithm to the system or vehicle. We have to create a working model to implement the path planning algorithm for the off-road navigation system to find the shortest path. First of all, clearly outline the purpose and requirements of the autonomous vehicle. Consider the factors like size, terrain, and desired functionalities. And then, design the vehicle using designing software like CATIA, AutoCAD, Solidworks, Creo, etc. Then choose a small off-road chassis that accommodates the selected components. Include DC motors, wheels, a power source (battery), and a motor driver to control the movement. Select sensors for environment perception. Common choices include RADAR sensor, ultrasonic sensors for obstacle detection, infrared sensors for line following, or a camera for vision-based navigation.

We have selected RADAR sensor and Camera for this navigation system to find out the shortest path and obstacle avoidance in the unstructured environments. Choose a microcontroller platform such as Arduino or Raspberry Pi to process sensor data and control the vehicle in the unstructured environments. Ensure it has sufficient processing power and I/O capabilities. Build the physical structure of the vehicle by assembling the chassis, motors, wheels, and any other mechanical components. Ensure proper weight distribution for stability to enhance the safety in the unstructured environment. Then wire the DC motors to the motor driver, and connect the motor driver to the microcontroller. Also, connect the power supply (battery) to provide power to the motors and the microcontroller. Connect the selected sensors such as RADAR sensor and the Camera to the microcontroller.

Implement the necessary circuitry for sensor interfacing, and ensure proper alignment for accurate data collection. The written code in the MATLAB software code to program microcontroller. Implement algorithms for sensor data processing, decision-making and control of the motors. This code will guide autonomous behavior of the vehicle. Test each component individually and then the integrated system. Debug the code and address any issues that arise during testing. Ensure the vehicle responds appropriately to sensor inputs. If your vehicle is for specific tasks, implement features such as obstacle avoidance, line following, or GPS-based navigation, depending on your objectives.

REFERENCES

1. *Pranav Ramachandran, Saulius Japertas* [1] (2023), Investigation of Unmanned Ground Vehicle Obstacles Avoidance Model.
2. *JiajiaChen , Rui Zhang , Wei Han , Wuhua Jiang , Jinfang Hu , Xiaoshan Lu , Xingtao Liu and Pan Zhao* [2] (2020), Path Planning for Autonomous Vehicle Based on a Two-Layered Planning Model in Complex Environment.
3. *Zandra B. Rivera , Marco C. De Simone and Domenico Guida* [3] (2019), Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments.
4. *Maradona Rodrigues , Andrew McGordon, Graham Gest, and James Marco* [4] (2018), Autonomous Navigation in Interaction-Based Environments—A Case of Non-Signalized Roundabouts.
5. *Bijo Sebastian, Pinhas Ben-Tzvi* [5] (2018), Physics Based Path Planning for Autonomous Tracked Vehicle in Challenging Terrain.
6. *Khalil , S. Hegazy , A.Roheim and Y. H. Hossam El-Din* [6] (2016), Design and implementation of navigation control system for unmanned ground vehicles.
7. *Kuralamudhan Arutselvana, Vijayakumari* [8] (2015), Assistive Autonomous Ground Vehicles in Smart Grid.
8. *Diogo Amorim, Rodrigo Ventura* [7] , (2014), Towards efficient path planning of a mobile robot on rough terrain
9. *David Silver, J. Andrew Bagnell, Anthony Stentz* [9] (2010), Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain.

10. *Steven Scheding, Gamini Dissanayake, Eduardo Mario Nebot, and Hugh Durrant-Whyte* [10] (1999), An Experiment in Autonomous Navigation of an Underground Mining Vehicle.
11. *A. Lacaze, Y. Moscovitz, N. Declaris, K.Murphy* [11] (1998), Path Planning for autonomous vehicle driving over rough terrain.
12. *Martin Adams, Wijerupage Sardha Wijesoma, and Andrew Shacklock* [12] (2007), Autonomous Navigation: Achievements in Complex Environments