Tartu University

Faculty of Science and Technology

Institute of Technology

Anish Shrestha

# Physical A*: Graph-Based Search Algorithm for Robot Navigation On-the-Go

Master's thesis (30 ECTS)
Robotics and Computer Engineering

Supervisor:

MSc. Tambet Matiisen

Tartu 2024

# Resümee/Abstract

**Physical A*: Graph-Based Search Algorithm for Robot Navigation On-the-Go**

Robot navigation is commonly viewed as a trajectory planning problem, relying on a pre-existing map. However, the availability of a prior map can be problematic, especially in military or rescue scenarios. This thesis elaborates on the concept of a two-level planning and navigation algorithm called physical A* to address this problem, focusing typically in use cases where a prior map is not known. Physical A* is an A* graph traversal where the robot physically drives along the nodes of the graph. The graph is constructed on-the-go. A lower level planning component proposes multiple waypoints stored as graph nodes. A higher level planner, with a broader understanding of the geographical or spatial context computes the goal heuristic for these nodes. Based on the goal heuristic, the waypoint with least cost is selected to explore towards the goal. Physical A* mainly concentrates on exploring the waypoints that would lead the robot towards the goal in the most optimal form.

**CERCS:** T125 Automation, robotics, control engineering, T120 Systems engineering, computer technology

**Keywords:** robot navigation, trajectory planning, physical A*, graph traversal, goal heuristic

**Füüsiline A*: graafil põhinev otsingualgoritm ilma kaardita navigeerimiseks**

Robotnavigeerimist käsitletakse tavaliselt trajektoori planeerimise ülesandena, mis toetub varem loodud kaardile. Varasem kaardistamine võib aga olla probleemne näiteks militaar ja pääste kasutusjuhtude puhul. Antud töö pakub nende olukordade jaoks välja lahenduse, mis põhineb kahetasemelisel planeerimis- ja navigeerimisalgoritmil nimega Füüsiline A*. Füüsiline A* on sarnane klassikalisele A* graafi läbimise algoritmile, ainult et graafi laiendamiseks läbib robot graafi tippe reaalselt, füüsilises maailmas. Madalama taseme planeerija pakub välja potentsiaalseid järgmiseid teekonnapunkte, milleni viib takistustevaba tee. Kõrgema taseme planeerija hindab tõenäosust, millise teekonnapunkti kaudu jõuaks kõige kiiremini sihtkohani. Kombineerides omavahel teekonnapunkti sõitmise teepikkust ja teekonnapunktist sihtkohta jõudmise hinnangut valib Füüsiline A* välja kõige optimaalsema teekonnapunkti järgmiseks külastuseks. Järk-järguline teekonnapunktide külastamine algoritmiliselt optimaalses järjekorras viib roboti lõpuks sihtkohta.

**CERCS:** T125 Automatiseerimine, robootika, juhtimistehnika, T120 Süsteemitehnologia, arvutitehnoloogia

**Märksõnad:** robotnavigeerimine, trajektoori planeerimine, füüsiline A*, graafi läbimine, sihtkoha heuristika

# Contents

# List of Figures

# List of Acronyms

**ROS** - Robot Operating System

**SLAM** - Simultaneous Localization and Mapping

**LiDAR** - Light Detection and Ranging

**GNSS** - Global Navigation Satellite System

**GPS** - Global Positioning System

**UTM** - Universal Transverse Mercator

**MPC** - Motion Predictive Control

**ONNX** - Open Neural Network Exchange

**RECON** - Rapid Exploration for Open-World Navigation with Latent Goals

**ViKiNG** - Vision-Based Kilometer-Scale NAvigation with Geographic Hints

**CBF** - Control Barrier Function

**VIB** - Variational Information Bottleneck

**3D** - 3 Dimensional

**2D** - 2 Dimensional

**FOV** - Field of View

**AIRE** - Artificial Intelligence and Robotics Estonia

# 1 Introduction

Robot navigation has been a topic of research for quite a while, and has been approached across various disciplines. Traditionally, it has been tackled through mapping the world around the robot, followed by path planning on top of this geometrically precise map. This integration of mapping and path planning has been pivotal to numerous state-of-the-art navigation systems in robot navigation [1].

Similar strategy is also used by Simultaneous Localization and Mapping (SLAM) which relies on sensor data, such as cameras or Light Detection and Ranging(LiDAR), and interprets these data to identify the landmarks or obstacles to compute the most optimal trajectory for precise navigation. SLAM constructs a map continuously localizing the robot in the map and fuses the map updates along with the position estimates to iteratively update and refine both the map and the robot's position in the map.

However, map-based solutions rely heavily on some restrictive assumptions, such as access to structured sensor data like point clouds generated by LiDAR and precise localization which limits the applicability in unstructured environments and requires precise and expensive sensor suite. Also, relying on pre-existing maps can be detrimental in situations where the environment around the robot undergoes substantial changes over time, as seen in off-road landscapes.

Deep learning based solutions provide an alternative approach. Taking inputs as images from on-board cameras, the learned neural networks would compute the necessary driving actions and yield out some driving commands necessary to navigate the robot [2]. The robot learns these vision based navigational intelligence by training neural network models on a large diverse range of data. This approach can be beneficial in many instances as common environmental patterns can be learned, for example recognizing that tall grass can be driven through while a wall should be avoided.

However, completely ignoring the geometric layout of the environment might not be a smart choice either, as the spatial arrangement of the world contains some basic regularities for robot navigation, such as the need to steer clear of narrow passages, not to drive towards the wall or dead-ends, etc. Instead of learning these details from scratch, this thesis utilizes existing information of the robot environment using some sensors.

Recent research has focused on robot navigation in unfamiliar environments, exemplified by Vision-Bsaed Kilometer-Scale Navigation with Geographic Hints(ViKiNG) [3] which dynamically constructs a graph as the robot explores new terrain. ViKiNG emphasizes the concept of creating a mental map by combining visual perception with geographical cues to form a navigational graph. This approach draws inspiration from human navigation, where individuals integrate visual inputs with a topological map like a rough roadmap to navigate between two

points without relying on precise and highly detailed maps.

A very close approach to ViKiNG is the Rapid Exploration for Open-World Navigation with Latent Goals(RECON) [4]. RECON performs a search rapidly exploring around the environment to reach a visual goal. Similar to RECON, ViKiNG also leverages over exploration of a visual goal. But along with a goal image, ViKiNG also takes in a topological map to navigate to the goal, making it more directed and efficient over a range of a few kilometers.

Prior methods to ViKiNG such as RECON or ViNG [5] rely solely on camera images which restricts the navigation to be limited for a short range (tens of meters). ViKiNG is able to break this deadlock by incorporating the geographic context into the picture. The foundational idea that makes ViKiNG efficient in reaching far away goals is the iterative graph construction with geographic hints. This geographic context is analogous to having a geometrically imprecise topological roadmap providing a rough direction leading the robot towards the goal. It can happen that the nodes added previously have a better likelihood of reaching the goal. In such a case, the robot will backtrack through the graph to reach the goal. The search process for the best node in the graph is an A* graph search. The robot physically explores the space and expands the graph towards the goal. This showcases an innovative method of trajectory planning and navigation without the need for geometrically detailed maps or sophisticated sensor suite. With the aid of geographic supervision and graph expansion on the go, ViKiNG demonstrates a robust solution for long range outdoor robot navigation.

The distinction between a conventional A* and physical A* is in the fact that conventional A* already has the nodes and a goal heuristic set on a predefined map. The robot would drive along the best path based on this heuristic. While with physical A*, expansion of the graph happens iteratively. As the robot drives towards the best possible node along the graph, the bounds of the drivable area is expanded. The discovery of the best path through the graph happens on the go.

Building upon this concept of graph construction and traversal, this thesis touches upon building a similar navigation algorithm as ViKiNG, retaining some elements from the VikiNG approach while adapting others for the sake of practicality. The thesis serves as a feasibility study to assess the effectiveness of the physical A* algorithm for robot navigation in unfamiliar terrains. The work in the thesis includes testing the algorithm in two different simulation modalities.

The works in this thesis makes use of the laser scans produced by a 2D planar LiDAR in a simulation environment to propose collision free waypoints for the robot to traverse which forms the graph. The results in simulations show that graph traversal and backtracking helps the robot to get out of some navigationally difficult situations which the robot would be unable to demonstrate without the graph. The primary characteristic of the physical A* algorithm is the exploration of the traversable boundaries utilizing the goal-oriented heuristic.

Despite having the same sensing capabilities, navigation without the graph is less efficient in reaching the desired goal. For straight forward traversal there is no significant difference between the two, but in situations like avoiding the dead ends, the naive approach without using the graph gets stuck.

## 1.1 Problem Statement

The main problem statement that this thesis attempts to address is the effective robot navigation in unseen environments, creating a graph on-the-go and querying over the graph to reach the end goal. The nodes in the graph can be represented in any form, for instance: images, GPS coordinates, or some local coordinates. The only requirement would be that the robot should be able to confirm that it has reached that specific node while driving along the graph.

# 2 Background

## 2.1 Graph Representation

Graph theory is a branch of mathematics which specifically deals with the study of mathematical structures which represent the relationships between different objects or entities.

A graph consists of a set of vertices known as nodes, and the relationship between those nodes are defined by the lines joining those nodes called edges.

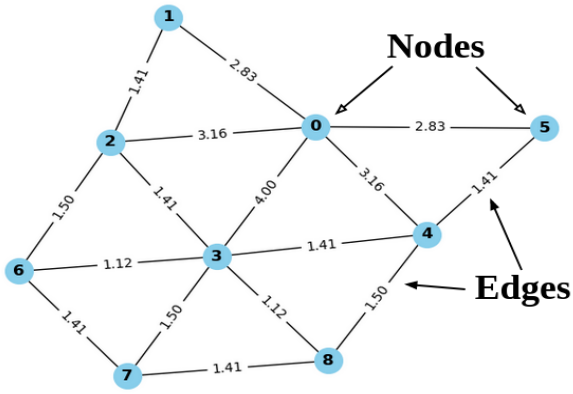



Figure 2.1: Graph representation with nodes as some abritrary cartesian coordinates and the edges representing distances between the nodes

Graphs have been used and have been very successful in designing search-based algorithms in computer science. For instance, while finding the shortest path between two cities, the problem can be solved by using graph representation where the nodes in the graphs would be the cities, and edges would be the distance between these coordinates. There might be multiple pathways going from city A to city B, but mapping all this information in a graph form simplifies the entire problem statement. As in Figure 2.1, while travelling from node 0 to node 7, there are multiple trajectories, but the graph would give the shortest path through node 3.

In robotics as well, graphs have been used in many algorithms to model the relationship between robot poses and sensor observations for mapping and localization purposes for robot navigation, for example getting a global path to a goal point with an occupancy grid map. In this thesis, a graph is used to represent all collision free drivable area. The nodes in the graph would be some specific GPS coordinates, while edges between the nodes would be the distance between the nodes that it is connecting.

## 2.2 A* Search Algorithm

A* is a popular graph search method, one of the most efficient path finding algorithms. The algorithm is initialized by setting the start and goal node, and it starts exploring the nodes finding the shortest path to the goal node. In A* graph search, it traverses to the lowest cost node from the current node. To calculate this cost, it employs a goal heuristic to select the subsequent node that offers a greater likelihood of progressing towards the goal.

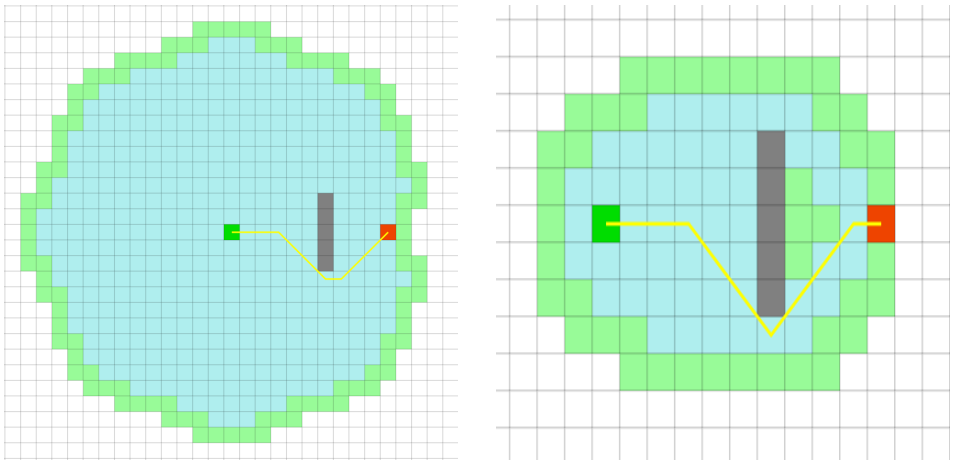
Goal heuristic is the main difference between Dijsktra and A* graph search. The Dijkstra algorithm expands the search in all directions and aims to find the shortest path between a node and all other nodes. This computation is done iteratively for all nodes and the distances are updated. On the other hand, A* uses the goal heuristic to find the shortest distance between a source node and a target node.

A* expands towards the nodes that are less expensive, as illustrated by this formula:

$$f(n) = g(n) + h(n)$$

$f(n)$ : Total cost associated with going from start to goal node through node **n**
$g(n)$ : cost to reach node **n** from start node
$h(n)$ : goal heuristic which represents the cost to reach from node **n** to the goal

On each iteration, A* chooses the node with minimal cost which eventually leads to the goal node traversing the shortest distance.



(a) (b)

Figure 2.2: Visualization of path exploration [6] performed by Dijkstra and A* to reach the same goal starting from same node: Dijkstra (a) and A* (b).

## 2.3 LiDAR

LiDAR is one of the most common sensors used in robotics, especially in mapping and localization applications. LiDARs provide distance information using laser emission and retrieval,

calculating the time of flight. LiDARs come in both 2D and 3D variations, where 2D LiDARs provide the spatial information about the world in the form of laser scans projecting laser beams in a plane, while 3D LiDARs produce a point cloud which represent the 3D structure of the world.

In this thesis, a 2D LiDAR is used to propose collision free waypoints. Raw laser scans are downsampled and manipulated to propose waypoints avoiding any form of collision.

## 2.4 Global Navigation Satellite System (GNSS)

GNSS is a satellite-based navigation system that provides location and time relevant information anywhere on the Earth's surface (global position). GNSS comprises a network of satellites orbiting the Earth which have their own ground stations and receivers.

GNSS is a general term referring to international Multi-Constellation Satellite System which includes the satellite network of different regions. Global Positioning System (GPS) is one of the most common satellite-based navigation system maintained by the Unites States. Others include GLONASS (Russia), Galileo (European Union), Beidou (China), etc.

Based on the position and time information from multiple satellites, a precise location of the GPS receiver can be estimated. To have the position estimate of the receiver device, it requires at least 4 satellites for initial position fix to calculate the time difference between local time clock and the GPS time. Once this time difference is established, 3 satellites are enough for getting the global position information. GPS receivers determine the position by triangulation of the signals from multiple satellites. The global position of the receiver refers to the latitude, longitude and the altitude of the reveiver device.

In this thesis, GPS position of the robot is used to locate the robot in a specific position. This GPS position of the robot is also used for initial GPS heading calibration and computation of the GPS coordinates of the proposed waypoints.

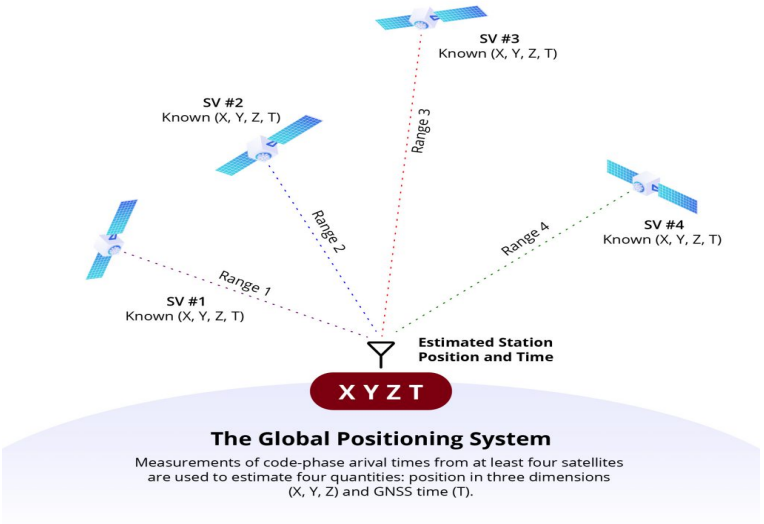



Figure 2.3: Global position and time estimation for GPS receivers using satellites [9].

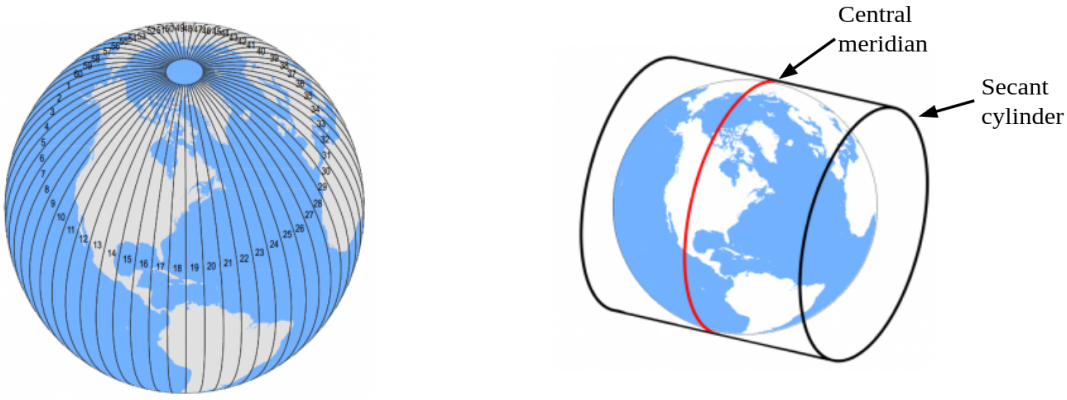
## 2.5 Universal Transverse Mercator (UTM) Projection

UTM projection divides the Earth's surface into several zones offering a more convenient coordinate system, where each zone has its own coordinate system resembling cartesian coordinate system making the computations easier, unlike GPS coordinates.

UTM fits a secant cylinder of the same width as the central meridian which intersects the ellipsoid along 2 circles parallel to the central meridian. Unfolding this secant cylinder will provide the flat coordinate system that UTM offers as shown in Figure 2.4b.

Individual UTM zone is a 6° segment of the Earth. Each UTM zone has its own coordinate system which uses a simple Cartesian coordinate system with easting and northing values measured in meters.

Also, there is an easy conversion technique between GPS to UTM coordinate system and vice-versa. In this thesis, UTM projections are used to set initial position of the robot with repsect to the goal in some simulation mode. It is easier to perform the local positioning computations in the UTM coordinate system.




(a) (b)

Figure 2.4: UTM projection: UTM zones (a) and the secant cylinder required for UTM projections (b) [10].

## 2.6 Control Barrier Function

For safety critical systems, it is essential to ensure that the current state is in a safe region. Therefore, there needs to exist a safe set such that it is a collection of all state vectors (vectors representing the position and/or orientation) that lie in a safe region. In relevance to robot navigation, this safe set can be the region in the robot's configuration space where it does not collide with any obstacles.

In order to confirm that the robot's current state is in the safe set, this safety is defined in terms of a function h(x), where x represents the robot's state vector. This function h(x) is called the control barrier function.

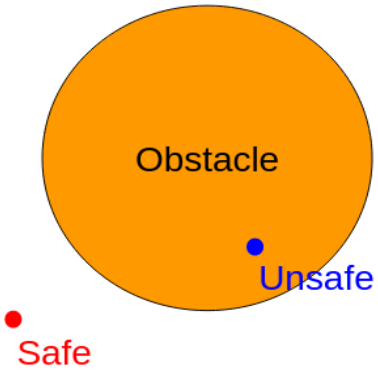



Figure 2.5: Safe region for robot navigation.

$$system = \begin{cases} \text{safe} & \text{if } h(x) > 0, \\ \text{boundary condition} & \text{if } h(x) = 0, \\ \text{unsafe} & \text{if } h(x) < 0. \end{cases}$$

For the most simplistic case, the function h(x) can be defined as:

$$h(x) = (x - x_o)^2 + (y - y_o)^2 - (r_o + r_r)^2$$

$(x, y)$ : Robot's position
$(x_o, y_o)$ : Obstacle's position
$r_o$ : Obstacle's radius
$r_r$ : Robot's radius

However, the safety of the system does not depend on h(x) only but also on the higher order derivatives of h(x). Synthesis of the appropriate velocities with the CBF approach along with a Motion Predictive Control (MPC) has been previously used in robot navigation in safety critical systems [7]

Moreover, CBF based controllers [8] have demonstrated some impressive obstacle avoidance behavior ensuring safety and smoothness over other similar controllers.

# 3 Literature Review

## 3.1 RECON

RECON attempts to tackle the problem of robot navigation in an unknown environment using visual inputs. The novelty lies in the fact that RECON can navigate in environments which the robot has never seen which is changing quite often just using camera images.

Given a goal image and the current camera image, RECON samples a potential latent goal which encodes relative information of the goal with reference to the observation image. RECON achieves this latent goal using a deep neural network called latent goal model as shown in 3.1. The model encodes both the current image as well as goal image into latent space and determines how cloes they are in this space. This model predicts two outputs: temporal distance to the latent goal which is the number of robot time-steps required to reach the latent goal, and the first action which is the waypoint or the trajectory that the robot needs to follow.

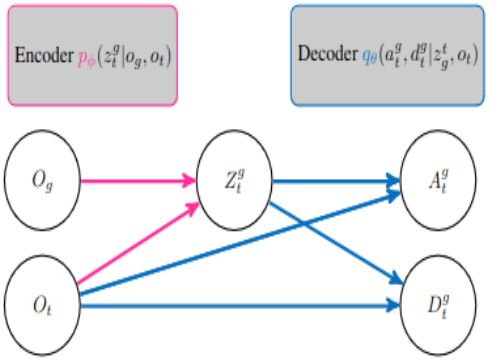



Figure 3.1: RECON latent goal model which takes in the goal image $O_g$ and the current camera observation image $O_t$ and samples a latent goal $Z_t^g$. The final predictions of the model are the temporal distance $D_t^g$ to the latent goal and the first action $A_t^g$ which is the trajectory to reach $Z_t^g$ [4].

Based on current camera observation, the latent goal model samples a latent goal, and as per the model predictions, the robot start to drive towards this sampled subgoal for a fixed number of time steps. During this navigation, the robot collects data from the surrounding. After this fixed number of timesteps, the graph is updated with the robot's current observation. At this point, RECON uses the newly collected data to fine-tune the robot's model giving the robot a detailed information about the environment.

Once the model is fine-tuned, RECON samples another latent goal and computes the cost for all waypoints in the graph. The cost for the waypoints are computed taking into account the edges

in the graph which represents the distance predictions from the latent goal model. The nearby unexplored nodes will have relatively lower cost. Once the least cost node is selected, the robot starts navigating to this waypoint along the graph.

If the robot can reach the goal directly, then it sets the final goal as subgoal and starts navigating using the same latent goal model and expanding the graph. Else, it explores new areas and expands the graph as it drives. Eventually, a graph is created all over the free space and the robot navigates to the intermediate subgoals while driving towards the final goal.

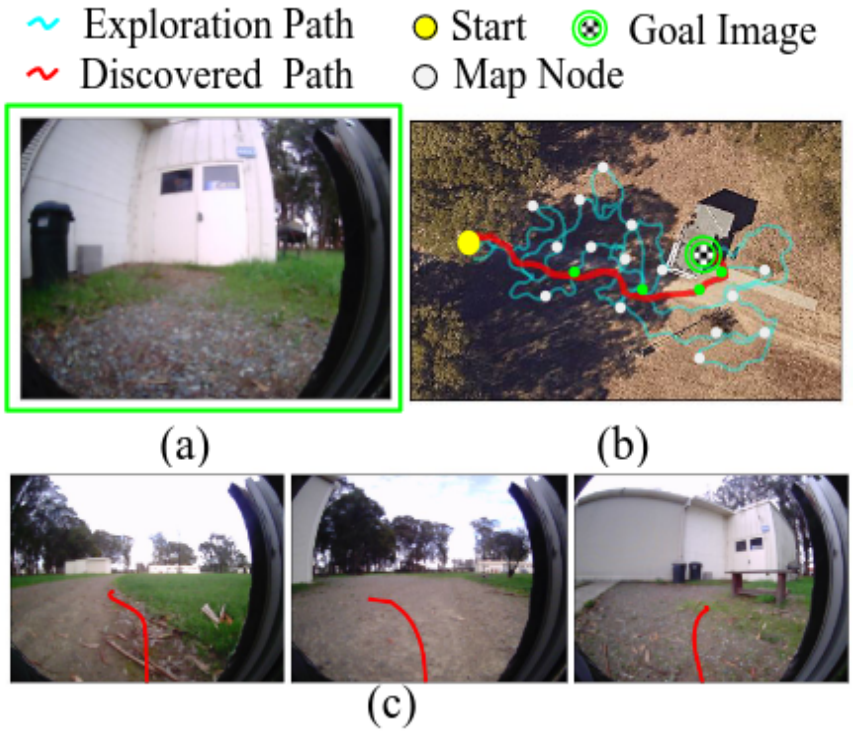



Figure 3.2: RECON system overview: Given a goal image (a), RECON samples latent goals and constructs a topological graph (b) helping it to explore the environment based on visual observations. The first actions to follow (c) - the red line based on current camera observation and the sampled latent goal [4].

## 3.2 ViKiNG

This thesis draws motivation from the ViKiNG paper which demonstrates efficient robot navigation in an environment which the robot has never seen before. This approach is a two-level robot navigation algorithm. The robot is provided with a goal image and goal GPS coordinates. The task in hand is to find a collision free path on-the-go that will eventually lead to the desired goal. The robot is equipped with cameras and a semi-precise GPS.

The first step is the local planner, which based upon the current camera image proposes multiple collision free traversable waypoints. This is similar to RECON in the sense that it uses only visual observations, but the local planner samples multiple potential waypoints for a single input image. ViKiNG maintains a graph with traversed waypoints and a priority queue with unexplored waypoints which were proposed previously. All these predicted waypoints that have

not been visited yet are added to the priority queue.

The main difference between RECON and ViKiNG is that ViKiNG uses a global planner to select the best waypoint from the priority queue. Global planner predicts the probabilities of these unvisited waypoints. The global planner predictions is the likelihood of a waypoint leading to the goal. Based on these probabilities, goal heuristic for all unvisited waypoints is computed. Using these probabilities, the goal heuristic for all unvisited waypoints is calculated. Based on this heuristic cost, the best waypoint is selected in A* fashion. Once selected, the path to this waypoint is forwarded to the robot to follow along the graph.

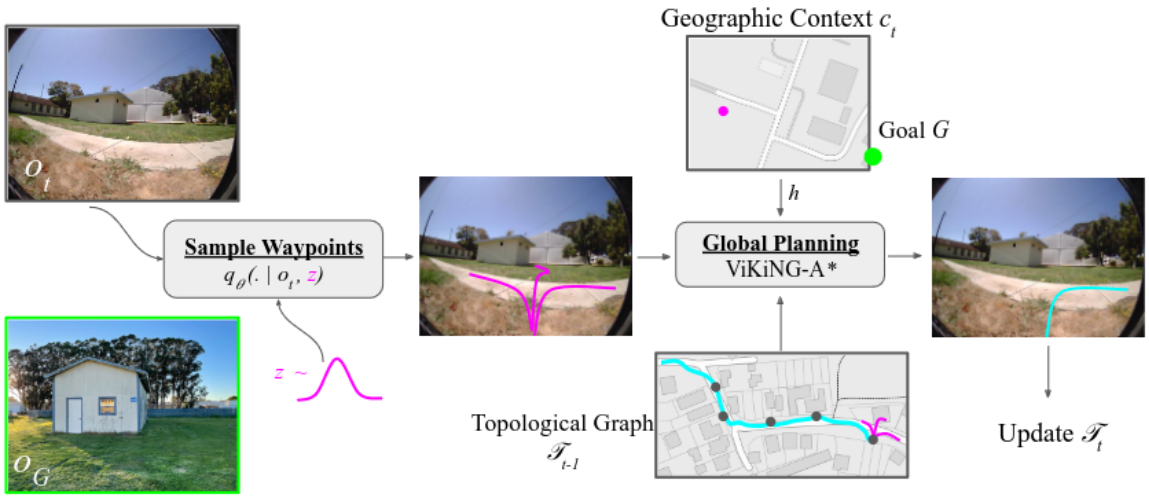



Figure 3.3: ViKiNG system architecture [3].

### 3.2.1 Local Planner

The local planner is a Variational Information Bottleneck (VIB) model which samples the waypoints based on current camera observation. It samples multiple waypoints from potential subgoals from the learned latent space. For an input image, there will be multiple latent goals, producing different temporal distances and first actions. Each of these temporal distances and first actions would represent a proposed waypoint.

The local planner model is trained by sampling pairs of time steps in trajectories from the dataset, where the image from the earlier time step is the current image and the later image is the waypoint image. The number of time steps between the images supervise the temporal distance, and the robot maneuver at the earlier time step provides the supervising signal for the first action. The temporal distances and first actions should drive the robot from current image to waypoint image.

During training, the waypoint image provides a prior distribution (mean and standard deviation from latent space) to sample waypoints to feasible collision free locations based on current camera observation. While integrating the local planner on-policy, the waypoints are sampled from a normal distribution N(0, 1).

The graph will have the images from the visited waypoints as nodes, and the edges will contain

the temporal distances between the node images. The cost for driving from one node to another is given by this edge. All unexplored waypoints are added in the priority queue and are annotated with the parent node from where these waypoints are proposed. Maintaining the graph and priority queue simultaneously provides the cost of travelling from the current position to all waypoints. The cost of driving to an unexplored waypoint is the cost of travelling from the current node to the parent node of the waypoint along the graph in addition to the cost of driving from this parent node to respective waypoint (temporal distance for the waypoint from it's parent node).

### 3.2.2 Global Planner

The global planner model is a goal heuristic model trained to predict the score for favorability of candidate waypoints to reach the goal, conditioned over an overhead topological map.

The model estimates the probability for a waypoint, based on whether the waypoint lies on a valid path from the robot's current GPS position to the goal in the overhead map. This creates a bias towards the waypoints that lie in the direction of the goal.

As the probabilities for more likely waypoints will be high, the heuristic cost for these waypoints should be less than others. This scaling of probability estimates to a heuristic cost is done by following computation:

$$h = \lambda * (1 - p)$$

$p$ : probability estimates coming from the global planner model
$\lambda$ : scaling constant for the heuristic (a hyperparameter)
$h$ : goal heurisitc for the waypoint with probability **p**

### 3.2.3 ViKiNG A* for Physical Search


(a) (b) (c)

Figure 3.4: ViKiNG A* for physical search: Local planner sampled waypoints in (a), global planner selected waypoint from the waypoints proposed by local planner in (b) and the robot trajectory in one of the test locations of ViKiNG where global planner is conditioned over a road map in (c) [3].

The cost of travelling along the graph to all the waypoints in priority queue is added with the goal heuristic cost and the node's visitation count to get the total cost. The cost of travelling

to the least cost waypoint along the graph is the sum of the cost of travelling to the waypoint's parent node plus the cost of travelling to the waypoint from that waypoint's parent node. The heuristic cost comes from the global planner estimates. And finally, the node's visitation count is basically the scaled version of the number of times that a node has been visited. This toal cost is computed as:

$$f(w) = g(t, w) + dw^{w}_{Pr[w]} + h(w) + v(\Pr[w])$$
$$v(\Pr[w]) = C * (NPr[w])$$

$g(t, w)$ : cost of travelling to the parent node of waypoint **w**
$dw^{w}_{Pr[w]}$ : cost of travelling to the waypoint **w** itself from it's parent node
$h(w)$ : global planner based goal heurisitc for the waypoint **w**
$v(\Pr[w])$ : visitation count of waypoint **w**
$NPr[w]$ : Number of visitations of the waypoint **w**
$C$ : Scaling constant (hyperparameter)

Once the cost for all waypoints have been computed, the least cost waypoint from prioirity queue is selected as the target waypoint. Since the priority queue has the parent node of this waypoint, a path along the graph from the current node to this waypoint is retrieved and sent to the robot. This will the desired path for the robot to follow to reach the least cost waypoint. Once the robot traverses along the graph and reaches the desired waypoint, it adds this waypoint in the graph. At this point, it will run the local planner model to propose the waypoints and the same continues.

This is the ViKiNG A* where the robot expands the boundaries of the drivable areas which eventually leads the robot to navigate towards the goal. With the combination of local and global planner, and iterative graph construction, the robot carries out the graph traversal and backtracking to drive towards the goal.

The combination of local and global planner can be achieved without maintaining and updating the graph and the priority queue. This approach is likely to fail, getting stuck in dead ends or around the corners, because it has no memory of previously visited waypoints. It just processes from the current batch of candidate waypoints.

The graph and priority queue provides the robot a memory of which waypoints have been already traversed, and offers a set of unexplored waypoints as probable options.

# 4 Methodology

The core idea of this method is based on graph traversal, motivated from ViKiNG. The General workflow architecture is illustrated in figure 4.1.

## 4.1 General Workflow

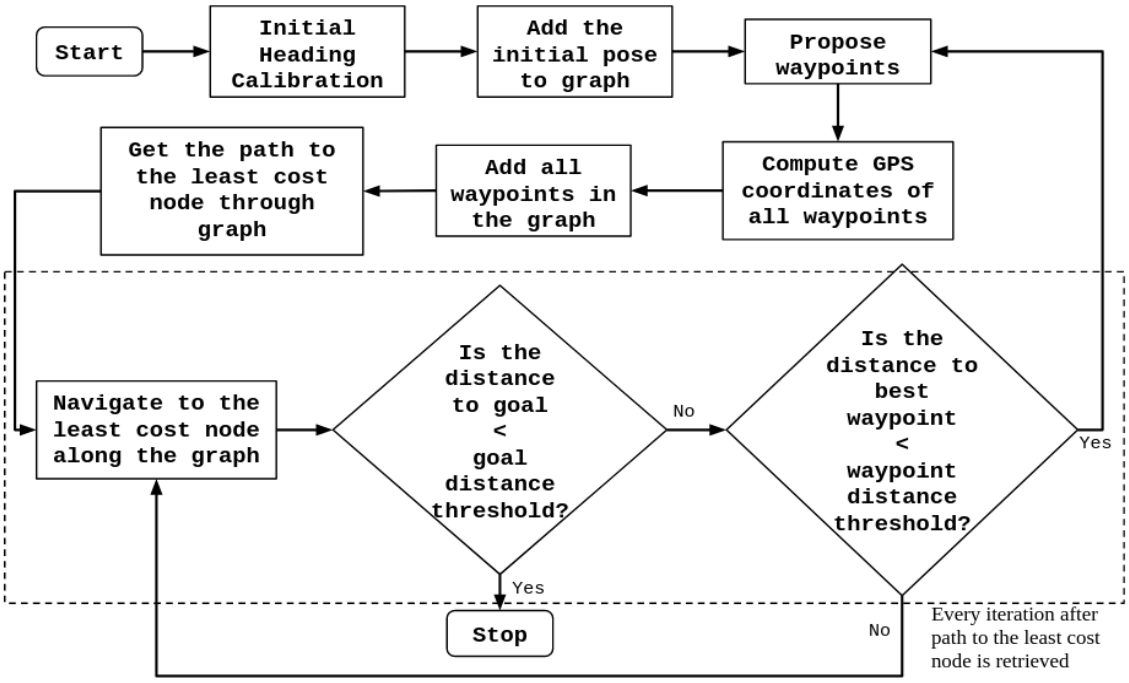



Figure 4.1: General workflow architecture.

## 4.2 Sensors Used

### 4.2.1 LiDAR

For gazebo simulation (section 4.5.1) with Jackal [21], a front facing 2D lidar is used for proposing collision free waypoints to makr up the graph nodes. A SICK LMS1xx 2D lidar [22] is used for this purpose while simulating Jackal.

### 4.2.2 GPS Device

For GPS, Jackal employs UBlox-NEO-M8N [23] GPS module, where the GPS receiver is placed at rear left corner of the Jackal's chasis. This GPS module is well-supported in ROS and comes built-in while installing Jackal software.

## 4.3 Softwares and Libraries Used

The physical A* algorithm as illustrated in Figure 4.1 is implemented using the following versions of softwares and libraries:

- Python 3.9.16
- NumPy 1.26.2
- queue/PriorityQueue
- NetworkX 3.2.1 [11]
- shapely 2.0.3 [13]
- Rasterio 1.3.9 [12]
- OpenCV 4.2.0 [19]
- onnxruntime-gpu 1.16.0 [14]
- ROS Noetic [16]
- CVXPY 1.4.3 [17]
- Conda 4.12.0 [18]

## 4.4 Modules of General Workflow

### 4.4.1 Robot Initialization and Heading Calibration

First and foremost, when the robot starts the navigation towards the goal, it is essential for the robot to know its heading angle with respect to the geographic North. So the first thing that the robot needs to carry out is the initial position initialization and heading calibration. The position information comes from the GNSS receiver attached to the robot which provides the GPS coordinates of the receiver.

In order to calibrate the initial heading of the robot, the robot is driven 1m in a straight path and the initial and final GPS coordinates are recorded. Based on these latitude and longitude values, the heading of the robot is computed with respect to the geographic North in clockwise direction when it reaches the 1m mark, using the following formulas [20]:

$$\Delta\lambda = \lambda_2 - \lambda_1$$

$$\Theta = \text{atan2}\left(\sin(\Delta\lambda)\cdot\cos(\Phi_2),\ \ \cos(\Phi_1)\cdot\sin(\Phi_2) - \sin(\Phi_1)\cdot\cos(\Phi_2)\cdot\cos(\Delta\lambda)\right)$$

$(\Phi_1, \lambda_1)$ : (latitude, longitude) of the starting point

$(\Phi_2, \lambda_2)$ : (latitude, longitude) of the final point

$\Delta\lambda$ : Difference between longitudes

$\Theta$ : Heading of the robot with respect to Geograpahic north in clockwise direction

Once this calibration is done, then the robot's built-in IMU or motor encoders can be used to compute the robot's latest heading.

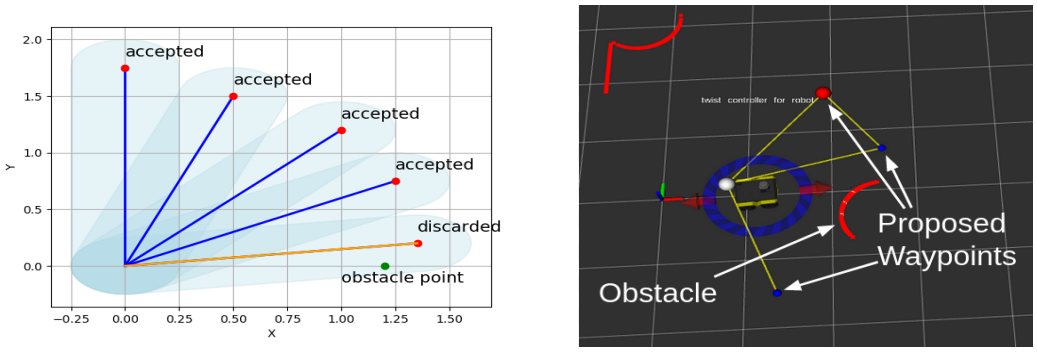
### 4.4.2 Waypoints Proposition

The waypoint proposition module in this workflow architecture is the substitution for the local planner in the ViKiNG paper. In ViKiNG, based on the current camera observation, multiple collision free waypoints are predicted by the local planner model.

For the sake of simplicity and getting it to work in simulations, the waypoint proposal using camera images and the local planner model is replaced by appropriate waypoint proposition methods. The way in which the waypoints are proposed differ based upon how the physical A* algorithm is implemented, i.e., either laser scan based waypoints proposal in Gazebo simulation as explained in section 4.4.2.1 or the random waypoints proposal in raster map simulation as explained in section 4.4.2.2.

#### 4.4.2.1 Laser Scan Based Waypoints Proposal

If the Physical A* method is being validated using Gazebo Simulation as explained in section 4.5.1 for Jackal simualation, then the waypoints are proposed using a front-facing SICK LM1xx LiDAR.




(a) (b)

Figure 4.2: Waypoint proposal: Visualization of accepted waypoints using shapely buffers (a) with accepted and discarded waypoints, and the integration of same technique to sample waypoints from current LiDAR observation in Jackal Simulation in RViz (b).

The raw laser scans are downsampled within a field of view (FOV) of 150 degrees facing front, i.e., from -75 degrees to +75 degrees, with a resolution of 10 degrees. Among these laser scans, only those scans are taken into account which have a range more than 3 meters. These laser scans come in the form of range and angles with respect to the lidar. For all valid scans with range more than 3 meters, a waypoint at a distance of 2m is set in the corresponding angle. These waypoints are converted to cartesian coordinates (x,y coordinates from range and angle), and transformed to the base_link frame.

As a final check to validate collision avoidance, the robot's width is taken into consideration. Using Shapely, a linestring from the robot's base_link frame to individual waypoints is drawn. A buffer (from Shapely) of the same size as the robot's width is placed on top of the linestring. Only those waypoints are proposed for which this buffer does not contain any points from the raw laser scans.

Similarly to figure 4.2a, in case there are any raw laser scan points within a buffer for a linestring joining the base_link frame and the waypoint, then that waypoint is discarded. In the worst case scenario when all the buffers contain points from raw laser scans, then the robot rotates continuously in clockwise direction until it finds a valid waypoint which is collision free and proposes that waypoint.

#### 4.4.2.2 Random Waypoints Proposal

An alternative to laser scan based waypoint proposition is sampling the waypoints randomly. Here, random waypoints are proposed within a FOV of 120 degress and within a distance range of 5-12 meters with respect to the robot's base_link. This method of proposing the waypoints is used in raster map simulation as explained in section 4.5.2.

### 4.4.3 Computation of Waypoints GPS Coordinates

The waypoints proposed as illustrated in section 4.4.2 are with respect to the base_link of the robot. To get the GPS coordinates for these waypoints, the waypoints need to be converted from the relative coordinates to GPS coordinates. This can be achieved using the following formulas [20]:

$$\delta = \frac{d}{R}$$

$$\Phi_2 = \text{asin}\left(\sin(\Phi_1) \cdot \cos(\delta) + \cos(\Phi_1) \cdot \sin(\delta) \cdot \cos(\Theta)\right)$$

$$\lambda_2 = \lambda_1 + \text{atan2}\left(\sin(\Theta) \cdot \sin(\delta) \cdot \cos(\Phi_1), \cos(\delta) - \sin(\Phi_1) \cdot \sin(\Phi_2)\right)$$

$(\Phi_1, \lambda_1)$ : (latitude, longitude) of robot's current position

$d$ : Distance of the waypoint from the robot's base_link

$R$ : Earth's radius (6370 m)

$\Theta$ : Heading of the robot with respect to Geograpahic north in clockwise direction

$(\Phi_2, \lambda_2)$ : (latitude, longitude) of the waypoint

### 4.4.4 Graph Construction and Physical A*

The initial GPS coordinates of the robot is added to the graph. This is followed by the waypoint proposal (section 4.4.2).

When the robot proposes waypoints, these waypoints are converted from base_link coordinates to the GPS coordinates as discussed in section 4.4.3 Once converted to GPS coordinates, the waypoints are added as nodes in the graph. However, to exclude the case where two nodes are very close to each other, a restriction is applied where a radius of one meter can have only one node. In case there are more than one nodes within a radius of one meter, the node that already exists in that radius is retained. All the latter nodes are discarded.

The distances from the robot's current position (current node in the graph) to these nodes are computed and added as the cost of the edges joining the nodes. In addition, if there are any other nodes in near proximity (roughly 2m), then edges between those nodes are also added by computing the distance between those nodes. This will construct a densely connected graph.

Alongside the graph, a priority queue is also maintained as in ViKiNG. The PriorityQueue class from standard queue library(Python) is used for this implementation. The nodes that are there in the graph are added to priority queue as well. This addition of nodes in both graph and the priority queue is a deviation from ViKiNG's approach of A* for physical search. Once the priority queue is appended with these nodes, the goal heuristic for all nodes in the priority queue is computed. The computation of goal heuristic is explained in section 4.4.5.

The total cost associated with a waypoint in priority queue is computed as the sum of the cost of travelling to the node along the graph plus the goal heuristic for that node. It is slightly different than ViKiNG in regards that it takes into consideration the visitation cost for the node as well which is not accounted in this thesis. The waypoint with minimum cost is selected as a target waypoint, and the robot traverses to this waypoint using the robot controller as discussed in section 4.4.6. The use of priority queue acts as a virtual memory which implicitly marks these nodes as "unvisited nodes". In later iterations, only these unvisited nodes are considered to be potential waypoints leading to the goal, while the graph provides a global path to these unvisited nodes.

### 4.4.5 Goal Heuristic

Depending on how the physical A* method is being validated and visualized, there exist mainly two flavors of goal heuristics for the computation of cost from the goal for individual waypoints.

#### 4.4.5.1 Distance Based Goal Heuristic

This goal heuristic is used when the Physical A* is being exhibited in the Jackal Simulation in a Gazebo world and visualized using RViz. For all waypoints in the priority queue, the goal heuristic is computed as the Euclidean distance between the GPS coordinates of the waypoint and the goal point. The formula to calculate the distance beween two GPS points given that the current heading with respect to geographic north (in clockwise direction) is known is as follows:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$\Delta\phi = \phi_2 - \phi_1$$

$$\Delta\lambda = \lambda_2 - \lambda_1$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right)$$

$$d = R \cdot c$$

$(\Phi_1, \lambda_1)$ : (latitude, longitude) of inital GPS point

$(\Phi_2, \lambda_2)$ : (latitude, longitude) of final GPS point

$R$ : Earth's radius (6370 m)

$d$ : Distance of the waypoint from the robot's base_link

#### 4.4.5.2 Learned Goal Heuristic

The motivation for using a neural network based goal heuristic comes from ViKiNG's global planner. Similar idea was used in the vision based navigation Milrem AIRE project [27], where the global planner model takes in an overhead map, the pixel coordinates of the robot's position and the goal position, and in turn predicts the probability map as shown in 4.3. The gps coordinates are converted to their corresponding pixel coordinates using Rasterio Python library.

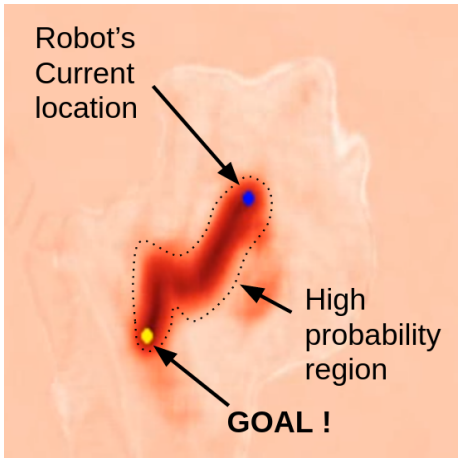



Figure 4.3: Probability map: Probability map showing the region with high probabilites in darker shades. Global planner will predict high probabilites for the waypoints in this region, taken from video[1]

This global planner model is trained with the georeferenced maps and recorded GPS trajectories, where all points in between the start and end points on a trajectory are marked as high probability points. This creates a probability map in the direction of the goal. Based on this probability map, the probabilities for individual waypoints is computed. Any waypoint falling on the region will have higher probabilities, and implies having lower cost. The pixels in darker shade in figure 4.3 are the high probability pixels. This model is an ONNX [15] model, and is run using the onnxruntime-gpu python package.

This retrofitted global planner is not a part of this thesis, and is an open-source resource. This global planner was plugged in while simulating the Physical A* implementation using the raster map simulation explained in section 4.5.2. Training this model was not part of the thesis, but it was integrated for the Physical A* implementation.

To scale these probabilities into metric cost, the Euclidean distance between the respective waypoint and the goal is divided by the probability associated with that waypoint. This would result in the heuristic cost for all waypoints in the priority queue. For waypoints with higher proba-

[1]Simulation video for testing global planner.

bilities, for instance 1.0, it will be equivalent to the Euclidean distance to the goal. Conversely, for a waypoint with lower probability, like 0.1, it will be mathematically equal to ten times the Euclidean distance to the goal.

### 4.4.6 Robot Controller

Robot navigation in this approach is attained in a two-level fashion. Firstly, when the target waypoint is chosen from the queue, a path through the graph is obtained. This is a higher level component of the navigation algorithm which passes the path for the robot to navigate through. Taking in this path (like a global path) and the downsampled laser scans as obstacles, the CBF controller generates the velocities and publishes them for the robot to reach the desired waypoint avoiding all obstacles in the path.

CBF based controller is implemented using the cvxpy library [17] which takes in the robot's position, target waypoint's position and the downsampled laser scans (point obstacles). The velocities that the cvxpy generates are holonomic drive velocities, i.e., linear velocities in x and y direction in the robot's base_link frame.

These velocities are then converted to differential drive velocities which will be linear velocity in x direction and angular velocity about the z axis (yaw) using the following formulas:

$$v = \sqrt{v_x^2 + v_y^2}$$

$$\omega = \frac{v_y}{b}$$

$v_x, v_y$ : velocities in x and y direction generated by cvxpy

$v$ : linear velocity in x direction (differential drive)

$\omega$ : angular velocity in z direction (differential drive)

$b$ : wheelbase: distance between wheels of the differential drive robot

## 4.5 Simulations

There are two different simulation modalities where the Physical A* algorithm is validated for robustness and performance. The workflow architecture remains more or less consistent, but some modules alter as per the requirements and implementation.

### 4.5.1 Gazebo Simulation

When simulating the Physical A* implementation over Gazebo, a world is designed such that the pathway from the robot's initial position to the goal might not be straight forward. This depends on the obstacle configuration. Some configuration have the pathway pretty straight forwards, and some comsists of different kind of obstacles in between.

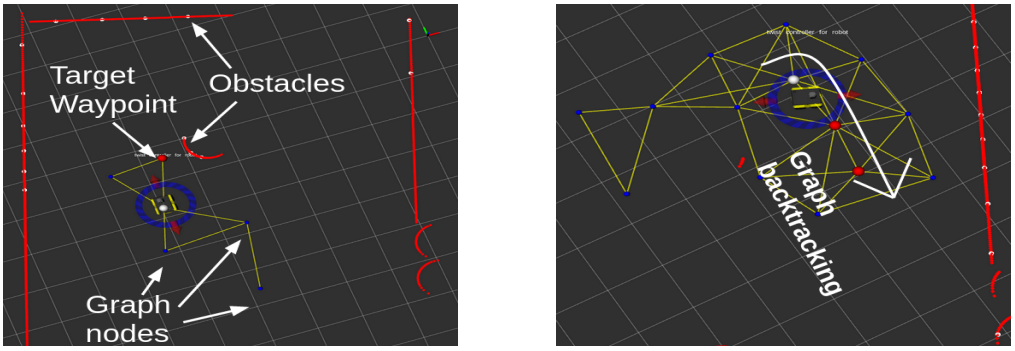
Firstly, the robot is spawned at a specific location from where it would need to calibrate the initial heading with respect to geographic north as mentioned in section 4.4.1. When the initial robot heading is calibrated, it starts proposing waypoints and adding those to both the graph and priority queue as discussed in section 4.4.4. In this simulation mode, a simple distance-based

goal heuristic is used as discussed in section 4.4.5.1.

The waypoint with least cost is selected as the target waypoint, and the path along the graph to this waypoint is sent to the robot's controller to reach that waypoint. Using the CBF controller (section 4.4.6), the robot drives to this waypoint, checking the proximity to the final goal in all timesteps. If it finds itself within goal distance threshold, then it confirms that it reached the final goal. If not, then it keeps navigating to the selected waypoint keeping in check if it is within the waypoint distance threshold. If the robot is within this waypoint distance threshold, then it confirms that the target waypoint has been reached. Else, it continues the waypoint navigation.

When the robot reaches this waypoint but is not in near vicinity to the goal, it proposes another set of waypoint and again the graph expands in that direction. All these waypoints are again added to the graph and priority queue, and the same continues until it reaches the final goal.

This is the Physical A* implementation, where the graph expansion happens on the go, unlike the conventional A* where the nodes are already existent, and the problem in hand is only to find the shortest path from start to finish.




(a) (b)

Figure 4.4: Physical A* in gazebo simulation: Graph expansion in free space and graph traversal (single red sphere) towards the target waypoint in (a), and graph backtracking (two red spheres) to get to the best waypoint along the graph in (b).

Eventually, it creates a mesh like grid with numerous nodes and edges expanding towards the end goal. This grid is a depiction of the drivable area in the robot's configuration space. This graph representation of the traversable regions is similar to occupancy grid map, but a lot sparser and light-weight.

### 4.5.2 Raster Map Simulation

In the raster map simulation, some modules slightly alter in terms of implementation. Firstly, there is no initial heading calibration as a point at a specific global position is just spawned representing the robot. And secondly, the goal heuristic used in this mode of simulation is a learned neural network model as explained in section 4.4.5.2.

In order to implement the Physical A* algorithm with this global planner model, a simulation using OpenCV is created where the robot is spawned over appropriate locations on top of a top-down raster map image. The pixel coordinates of the respective GPS position are retrieved

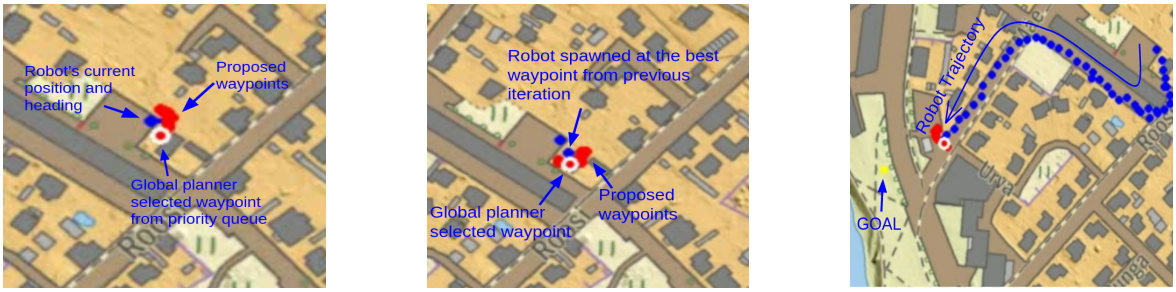
using rasterio APIs. The robot location, waypoint locations and the nodes in the graph that the robot traverses are places and visualized on each iteration. Some modules from the general system architecture deviate slightly in this simulation model.

Robot's initial position is set with respect to the goal GPS. GPS coordinates of the goal are converted to UTM coordinates, and the starting position of the robot is set at some offset distances in East and North directions from that point. The initial heading is set randomly.

Once the initial position and heading is set, the initial robot position is added to the graph and the robot proposes waypoints randomly as discussed in section 4.4.2.2. Just like in ViKiNG, these waypoints are added to the priority queue but not the graph.

Once the cost for all waypoints are computed, the one with least cost will be taken out from the priority queue as the target waypoint. The path along the graph to this waypoint is retrieved and sent for the robot to traverse through the path. In this simulation model, the robot is just spawned over these nodes for visualization purposes. Corresponding pixel coordinates from the GPS coordinates are visualized over the map using the Rasterio APIs.

Once it reaches the target waypoint, a new set of waypoints are proposed and the same continues until the robot reaches its final goal.




(a) (b) (c)

Figure 4.5: Physical A* in raster map simulation: Robot spawned at initial position at a random heading with proposed waypoints and global planner selected waypoint in (a), second iteration of robot navigation where robot is spawned over the best waypoint location and proposes another batch of waypoints in (b), and the successive robot navigation executing Physical A* leading towards the goal in (c).

# 5 Results

For results and analytics, the performance and robustness of the physical A* algorithm was examined and quantified comparing the result for the same scenario against a naive approach.

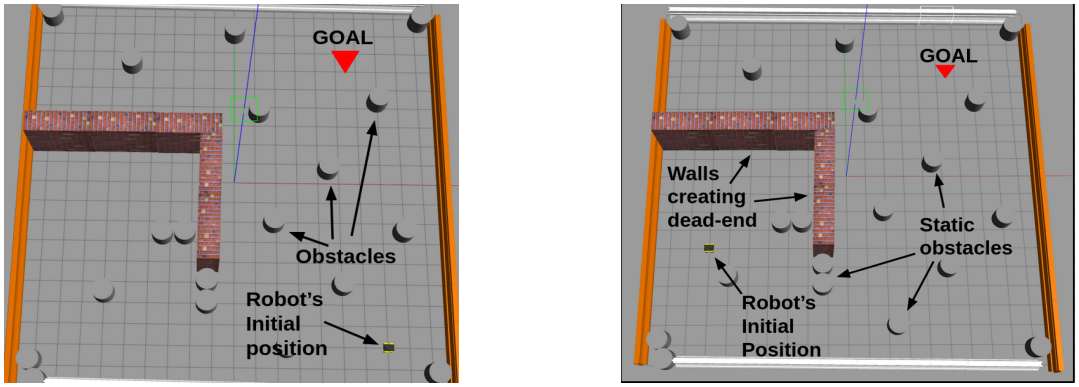
Naive approach refers to the the algorithm which does not make use of the graph and priority queue. The robot would consider only the current batch proposed waypoints. Everytime when there is a set of collision free waypoints sampled, the goal heuristic for these waypoints are only computed. And the waypoint with least cost is selected as the target waypoint. The navigation to reach the target waypoint is the same for both naive and physical A* approaches.

## 5.1 Results for Gazebo Simulation

The physical A* implementation in Gazebo simulation follows the exact software architecture as illustrated in section 4.1 and the graph construction is carried out as explained in section 4.5.1.

### 5.1.1 Qualitative Results for Gazebo Simulation

The robot was initialized in the same position for the two approaches and let free to explore and reach the goal. Goal here is a point farther away in space whose GPS coordinates are feeded to the system while initializing the robot. The qualitative performance was assessed in two disticnt obstacle configurations, namely easy (scenario 1) and difficult (scenario 2).




(a) (b)

Figure 5.1: Scenarios for qualitative analysis of gazebo simulation: scenario 1 - easy (a) and scenario 2 - difficult (b).

#### 5.1.1.1 Scenario 1 (Easy)

Scenario 1 (easy) consists only few static obstacles in the path originating from robot's initial position to the goal position as shown in figure 5.1a.

#### 5.1.1.2 Scenario 2 (Difficult)

Scenario 2 (dificult) is a relatively complicated obstacle configuration, which consists of a dead-end corner in the path between the robot's initial position and the goal position along with some static obstacles as shown in figure 5.1b.

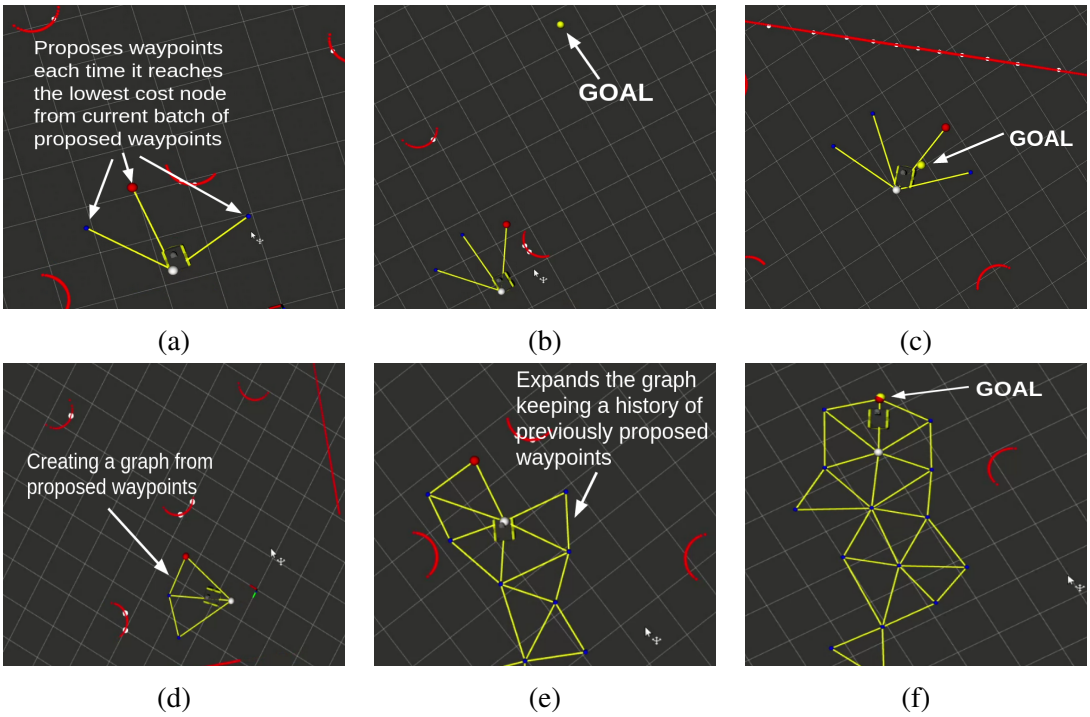



Figure 5.2: Qualitative results for easy scenario in gazebo simulation: Naive approach drives to the lowest cost node from a batch of waypoints proposed each time (a), (b) and (c), and Physical A* creates a graph considering all previously sampled waypoints as well and drives to the unvisited waypoint with least cost (d), (e) and (f) taken from robot simulation videos: video[2] and video[3]

For the easy scenario, both naive and physical A* algorithm is able to drive the robot to the goal as shown in figure 5.2.

Figure 5.3 shows that the naive approach fails when the obstacle configuration becomes more complicated. since the naive approach does not have the memory of previously visited nodes and the unexplored waypoints, taking into account the current batch for waypoints is not efficient in all use cases.

In contrast, physical A* method which has this memory of visited and unvisited nodes in the drivable space demonstrates that it can travel towards the goal even in such difficult circumstances. In order to achieve the successful navigation towards the goal, the robot performs

[2]Simulation video for Gazebo scenario 1 naive implementation video
[3]Simulation video for Gazebo scenario 1 physical A* implementation video

backtracking through the graph back and forth to get to the most optimal node leading towards the goal.

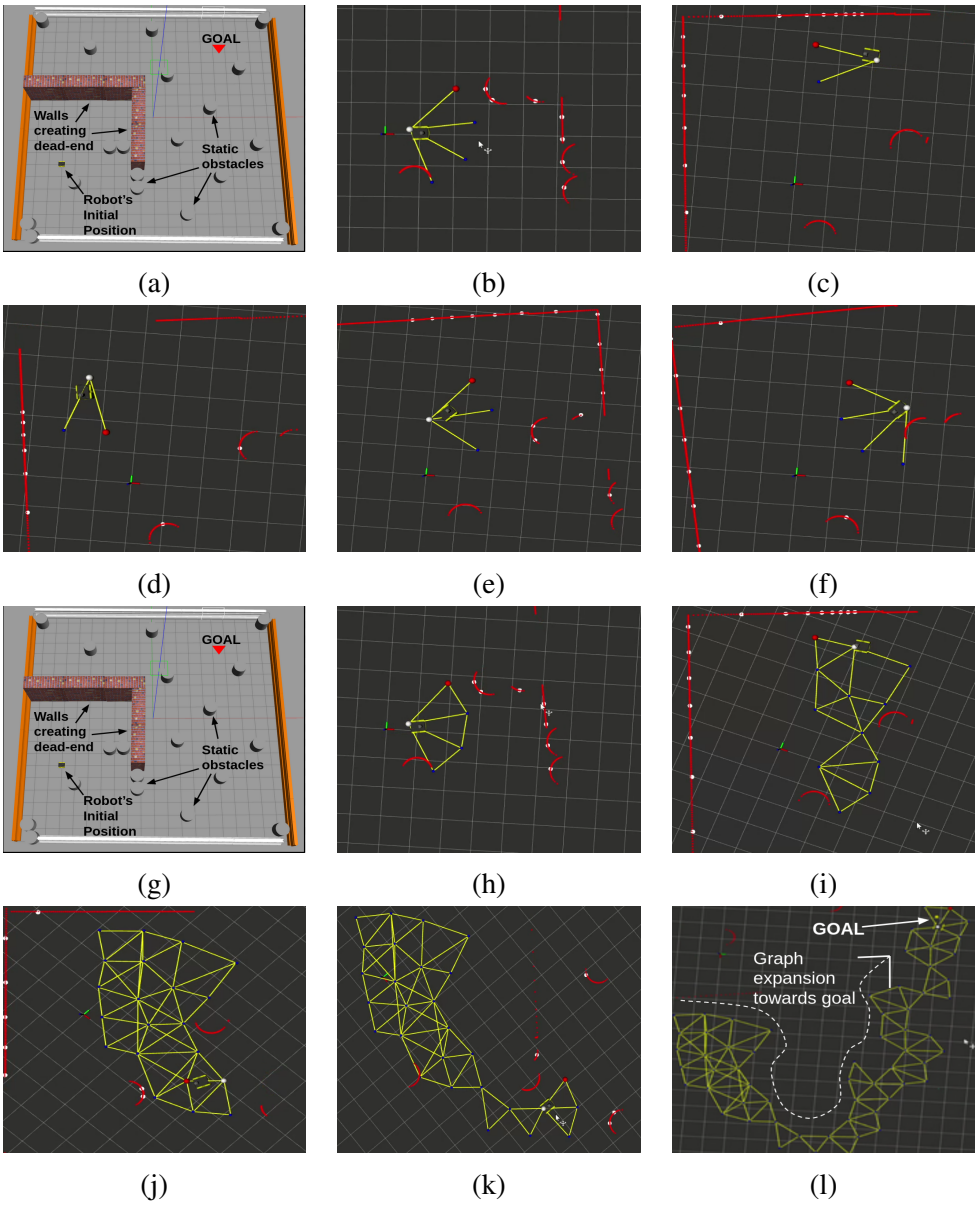



Figure 5.3: Qualitative analysis: Naive approach- (a) to (f) gets stuck in the dead end, physical A*- (g) to (l) expands the graph towards the goal eventually and gets out of the dead end taken from simulation videos: video[4] and video[5]

#### 5.1.1.3 Remarks on the Qualitative Tests for Gazebo Simulation

For simpler tasks both the approaches work equally good. But, when the obstacle configurations gets complicated, then the algorithms are tested to their limits and shows different behaviors.

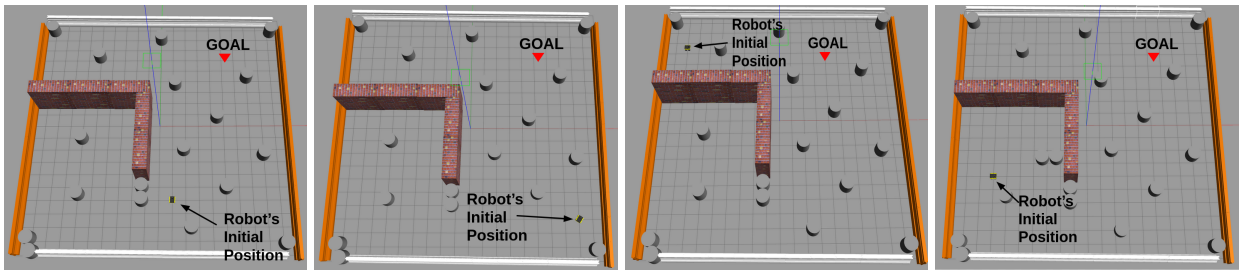
As figure 5.3 illustrates, for a difficult obstacle configuration like a dead end in the path would make it impossible for the robot to get out from there in naive implementation. As the goal is

[4]Simulation video for Gazebo scenario 2 naive implementation video
[5]Simulation video for Gazebo scenario 2 physical A* implementation video

on the other side of the wall, the goal heuristic for waypoints near the dead ends will always be lower. Therefore, the naive approach fails to reach the goal.

However, the physical A* algorithm making use of the graph and the information in the priority queue about the undiscovered nodes in the graph can make its way out of the dead end and eventually drive towards the goal.

### 5.1.2 Quantitative Results for Gazebo Simulation




(a) (b) (c) (d)

Figure 5.4: Obstacle configuration for quantitative analysis in gazebo simulation: Four different obstacle configuration where the robot is initialized in different positions in simulation: simulation videos[6]

For quantifying the results, both the naive approach and physical A* algorithm were implemented in four different scenarios. For each scenario, the initial position of the robot and the goal point is the same. The results are quantified in terms of the number of robot time steps taken by the naive approach against the physical A* algorithm. The number of time steps are recorded only after the initial GPS heading is calibrated.

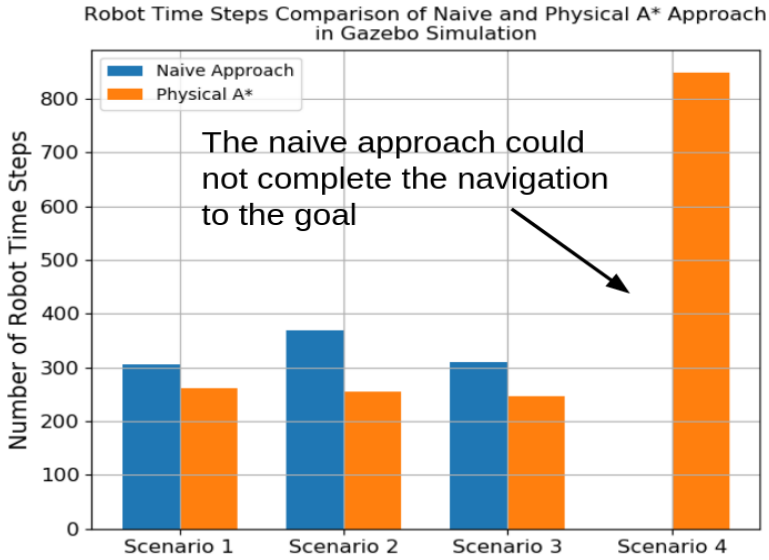



Figure 5.5: Quantitative results for gazebo simulation: the number of time steps robot needed to reach the goal in four distinct scenarios for naive and physcial A* algorithm

[6]Simulation videos for quantitative results in gazebo videos.

The final quantified results are presented as bar graphs, where the bars for naive and physical A* method for the same scenario are adjoined together as shown in figure 5.5.

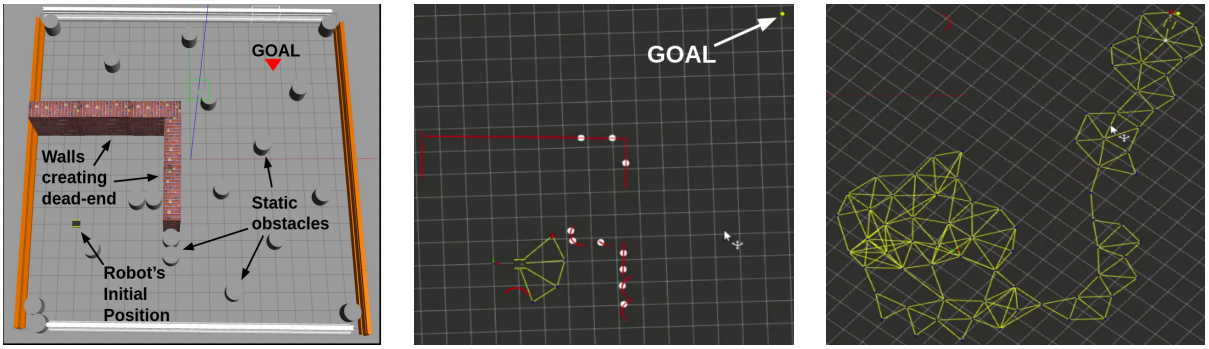
#### 5.1.2.1 Remarks on the Quantitative Tests for Gazebo Simulation

From the results as shown in figure 5.5, the graph-based physical A* algorithm is apparently more time-efficient as it recorded lower number of robot time steps in each scenario (also completed all scenarios) and can drive along difficult obstacle configuration as well.

### 5.1.3 Physical A* with Odometry Coordinates

As expalined in section 4.4.4, the physcial A* algorithm is implemented with GPS coordinates of the nodes. Any positioning system should work, as long as there exists a metric to calculate the cost for individual waypoints.

To test the performance of the physical A* with a noisy(unstable) positioning sytem, the implemntation of physical A* was examined using odometry coordinates in place of GPS coorid-inates.



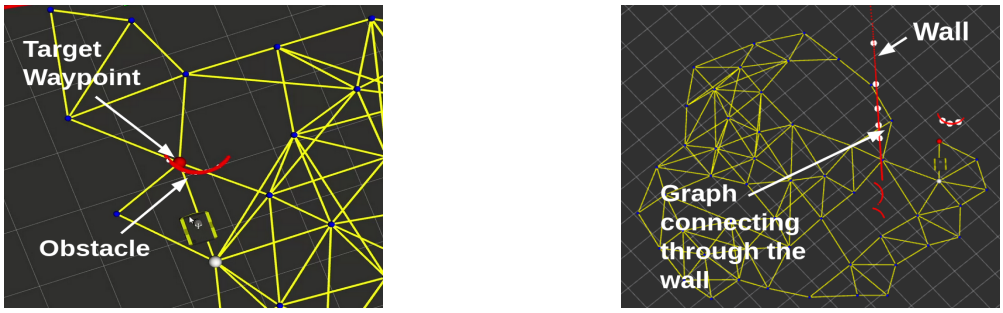

(a) (b) (c)

Figure 5.6: Physical A* with odometry coordinates: the same gazebo environment for scenario 2 (a), first set of proposed waypoints added to the graph with goal in top right corner (b), and the final graph that the robot expands towards while searching for the goal (c) taken from Gazebo simulation video[7].

#### 5.1.3.1 Remarks on Comparative Analysis between GPS vs Odometry based Physcial A*

Physical A* with odometry coordinates in general works good enough, but is comparatively less reliable. This mainly comes from the odometry drift as a result the noise in IMU and/or wheel encoders. When the graph is constructed using odometry coordinates, then the nodes in the graph will have the odometry coordinates from the timestamp when these waypoints were proposed. When new nodes are added, these newly added nodes will have the odometry coordinates from a later timestamp which incorporates the odometry drift between these timestamps that the robot has experienced. This drift will have some synchronization problems as depicted in figure 5.7.

[7]Gazebo simulation video for physical A* with odom coordinates video.



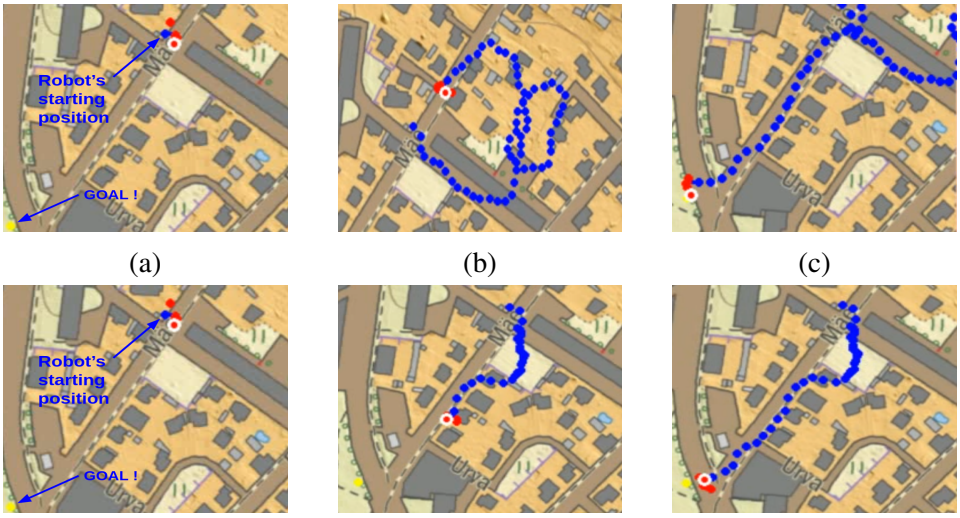

(a) (b)

Figure 5.7: Problems with physical A* using odometry coordinates: odometry drift causing the target waypoint to be inaccessible around the obstacle (a), and the graph connecting through the wall (b).

## 5.2 Results for Raster Map Simulation

The physical A* implementation in Raster Map simulation follows the procedure as discussed in section 4.5.2.

### 5.2.1 Qualitative Results for Raster Map Simulation




(a) (b) (c)

(d) (e) (f)

Figure 5.8: Robot trajectories for raster map simulation scenario 1: Initializing the robot in the same place, behavior shown by naive approach- (a), (b) and (c), trajectory followed by physical A*- (d), (e) and (f), taken from simulation videos: video[8], video[9]

The robot was spawned at a specific location initally over a top-down raster map image, and then the physical A* method was implemented. For qualitative anlaysis, the naive and physical A* methods were tested in two different scenarios, initializing the robot in a different location. For Scenario one, the robot is initialized 120 meters east and 150 meters north from the goal. While for scenario two, the starting point of the robot is 100 meters east and 300 meters north from the goal.

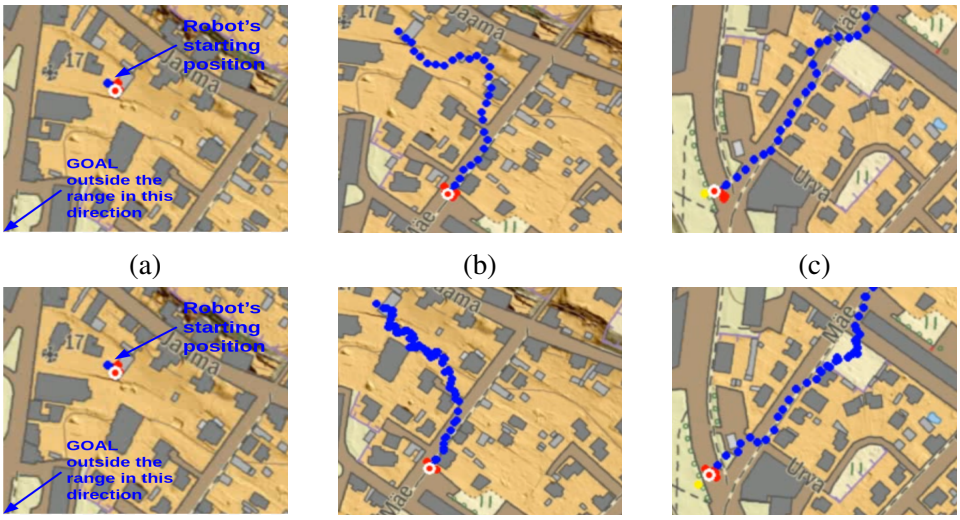
[8]Raster map simulation video naive approach scenario 1 video.
[9]Raster map simulation video physical A* approach scenario 1 video.

#### 5.2.1.1 Remarks on Qualitative Tests for Raster Map Simulation

As seen in figures 5.8 and 5.9, the naive approach seems to explore the space more randomly. Since it can only choose from a set of waypoints that it proposes in each iteration, when the waypoints are facing some other direction than the goal, this makes the robot to navigate in those directions. Hence, to find a waypoint optimal for leading towards the goal, the robot demonstrates some exploration in random directions.

As briefly explained in section 6.2.2, one of the limitation of the learned goal heuristic model is that the probability map extends in both directions (occasionally), and when the robot is facing the opposite direction to the goal, then it will choose the waypoints away from the goal (for naive approach). Once it moves farther away, the probability map will predict higher probability regions towards the goal and the goal heuristic model directs the robot towards the goal (as seen in video.)

But in case of physcial A* algorithm, it prevents this random exploration by backtracking through the graph as there will be waypoints closer to the goal making the robot face towards the goal (video).




(a) (b) (c)

(d) (e) (f)

Figure 5.9: Robot trajectories for raster map simulation scenario 2: Initializing the robot relatively farther than in scenario 1 and recording the behavior shown by naive approach- (a), (b) and (c), trajectory followed by physical A*- (d), (e) and (f) taken from simulation videos: video[10] and video[11]

While with the physical A* navigation, the graph search makes it more directed forcing the robot to backtrack along the graph to drive in the direction of the goal as the probabilities for pixels in probability map will be higher in that region implying lower costs. In some section of the robot trajectory in figures 5.8e and 5.9e, the blue dots are densely populated. This is because the robot backtracked around that section more often.

[10]Raster map simulation video naive approach scenario 2 video.

[11]Raster map simulation video physical A* approach scenario 2 video.

### 5.2.2 Quantitative Results for Raster Map Simulation

For quantifying the results for the raster map simulation, the number of robot time steps were chosen as the qauntifying figure. This figure provides the information about how quickly could the robot navigate over the real world maps, given that the waypoints proposed are random.

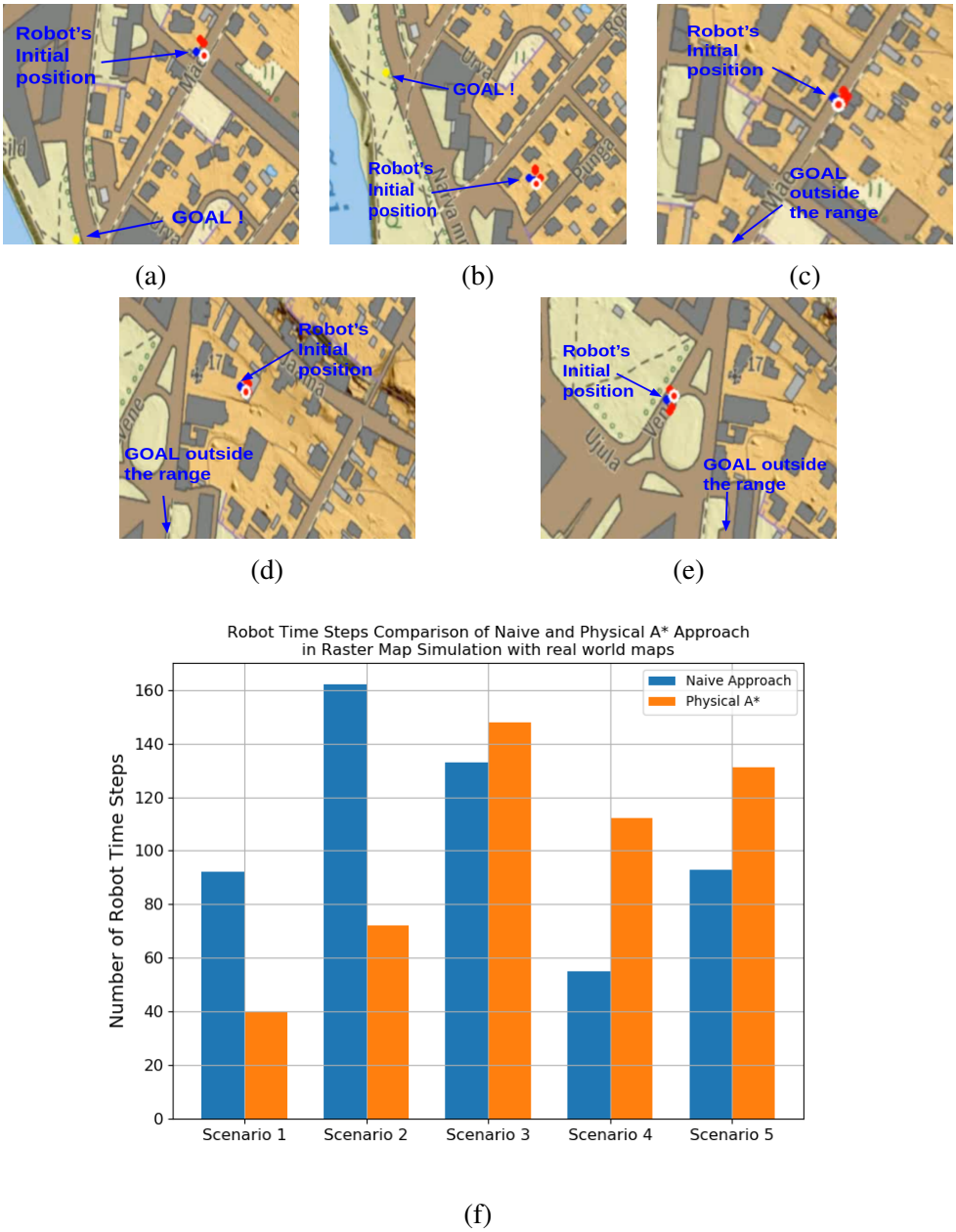



Figure 5.10: Quantitative analysis of raster map simulation: Starting location of robot in five different scenarios for quantitative analysis (a) to (e) taken from raster map simulation videos videos[12], and the number of robot timesteps taken by the naive and physcial A* implementations in corresponding scenarios (f)

To get to the quantitative analysis of results, the robot was deployed in five different scenarios (figures 5.10a, 5.10b, 5.10c, 5.10d, 5.10e ). For each scenario, the robot is initialized in a different start location. For both the approaches, the number of robot time steps were recorded and the quantitative results are presented as bar-graphs showing the number of time steps the

[12]Raster map simulation videos for naive and physical A* approach in 5 different scenarios videos.

robot needed to ge to the goal by naive exploration and physical A* exploration.

#### 5.2.2.1 Remarks on Quantitative Tests for Raster Map Simulation

Taking a look at the number of robot time steps that the individual approaches need to reach the goal point illustrated in figure 5.10f, it is difficult to explicitly announce which method is superior in terms of number. At times, the naive algorithm reaches the goal in less number of time steps than the physical A*.

This most probably comes down to the scaling of probabilities to cost. The probabilities for all sampled waypoints in the high probability region in the probability map is expected to be somewhat similar. When the global planner computed probabilities are scaled to the corresponding cost as explained in section 4.4.5.2, the waypoints which are closer in distance to the goal will always be chosen. One thing to acknowledge here is that when the distance is larger, the heuristic cost can be in an order of hundreds (for example, if the distance between the waypoint and goal is 50 meters and the probability is 0.4, then the heuristic cost will be 125 meters), while the cost of travelling through the graph will be in a few tens of meters. So, the heuristic cost overrides this graph traversal cost. Therefore, as long as there are such waypoints in the priority queue, the robot will keep backtracking again and again. This repetitive backtracking through the graph accounts for more number of timesteps required for the robot to drive towards the goal.

Once the robot faces towards the goal and the proposed waypoints are in the direction of the goal, then it drives towards the waypoints closer to the as these waypoint would have a lower heuristic cost. This justifies the behavior shown by the raster map simulation (can be visualized in videos.) upto a certain extent on why the naive approach at times is able to drive to the goal in less number of time steps.

# 6 Discussion and Analysis

Looking at the qualitative and quantitative results of the tests conducted, physical A* in general seems to perform better than the naive approach. Being more specific, in case of Gazebo simulation, physical A* outperformed the naive implementation both in terms of success in navigating through a difficult terrain including dead-ends and static obstacles, as well as in regards to the number of robot time steps required to reach the goal.

However, the tests carried out in raster map simulation differed from the ones in Gazebo slightly. For some scenarios, the naive approach was able to reach the goal quicker (lower number of robot time steps). This does not necessarily imply that the naive approach is better. Since the physical A* takes in account the entire history of waypoints proposed, as long as there is an unvisited waypoint available which has a lower cost, the algorithm tries to navigate to the goal through that waypoint. Therefore, the number of robot time steps for physical A* is likely to be more than the naive approach in this regard, for some scenarios.

The main characteristic of physical A* is to scan over the drivable collision free regions and create a graph to provide the supervision signals on where to drive. As the raster map simulation does not have the collision free waypoints and the entire map is a drivable area, drawing strong conclusions from this test is not a fair assessment of the system.

Overall, the physical A* algorithm seems to overcome issues that the naive approach is unable to handle (especially the dead ends case). This opens the door for the validation of the algorithm in some real life use cases and gives the research a direction with some future works that can be experimented. Along side the positive aspects, the tests conducted also hints towards some limitations of the system (or parts of the system).

## 6.1 Potential Use Cases

This thesis leverages undirected graphs as a sparse map for extracting drivable areas in any unseen environment. This lightweight map is constructed as the robot drives. With some geographical context around the robot, it is demonstrated that the robot can drive back and forth along this graph to reach the desired goal.

This approach is best suited for application domains where a geometric map of the robot environment is unavailable or difficult to map, for instance the off-road navigation for military and rescue applications.

This thesis is mainly concerned with expanding the graph in all free areas, where the main attribute for the graph nodes is the GPS coordinates. This is not a hard requirement. The graph can be constructed over any data as long as the edges between the nodes can be expressed as

cost (distance or any other metric) and the positioning system is robust against drifts.

## 6.2 Drawbacks

The main limitation of this approach is that the trajectory planning is non-optimal as compared with map based navigation. When there is a prior map, then the robot already has the privilege of planning the path on top of the map, computing all the costs beforehand. As the robot expands the graph as the robot traverses while implementing the physical A*, the robot needs to drive along the graph too much back and forth, for example avoiding dead ends (as shown in Gazebo simulation) or when the robot is directed in some other directions (as shown in raster map simulation while using the global planner model).

To be more specific of the the parts of the entire system which can act as a point of failure are as follows:

### 6.2.1 Local Planner Limitations

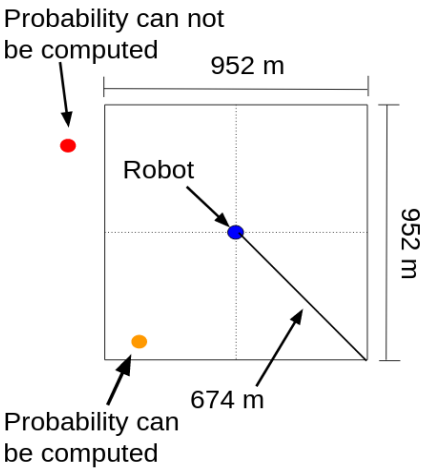
In this thesis, the waypoint proposal is solely based on laser scans. Although it provides a good spatial information of the environment, it misses out on some aspects of intellgence which is expected to have if this algorithm is to be intergrated in an autonomous (or semi-autonomous) unmanned ground vehicle.

A laser scan representation would treat both tall grass and trees in the same way. However, a camera based system with some aid with deep learning techniques can distinguish between these two objects. For an outdoor navigation robot, especially designed for off-road landscapes tall grass should be drivable whereas a tree must be avoided.

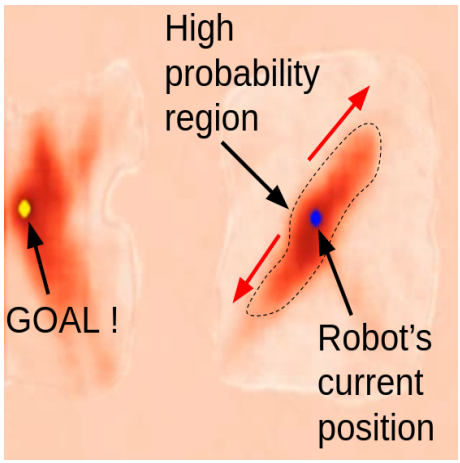
### 6.2.2 Global Planner Limitations

For the learned global planner used in this thesis, the probability map that the model predicts is of size 400 pixels X 400 pixels with a resolution of 0.42 X 0.42, corresponding to 954 meters X 954 meters centered around the robot. For a waypoint outside the map range as shown in figure 6.1a, the probability for that waypoint cannot be computed. Even if there is a revised model which has higher range of proability map prediction, there will be a limitation of how far should the goal be. This can of course be tackled using intermediate subgoals which are within this range.

But a bigger limitation of the learned goal heuristic model is the probability map prediction which elongates in either direction (occasionally) of the robot's current position as shown in Figure 6.1b. Since the high probability region is in both sides of the robot, when there are waypoints only in the direction away from the goal, these waypoints would have lower heuristic cost. This would eventually drive the robot far away from the goal. An expected behavior for the global planner would be to have these high probability region only in the direction towards goal so that these waypoints would have a relatively lower costs and the robot would choose driving towards these nodes through the graph.




(a)




(b)

Figure 6.1: Probability map limits: Probability map range shown as a square, and examples when the probability of a waypoint can be computed and when it cannnot be computed.

## 6.3 Future Works

### 6.3.1 Image Based Local Planner

There are a couple of deep learning models as listed in the General Navigation Models [24], such as NoMaD [25] which samples multiple waypoints for a single input (current image plus some prior context images) and a goal image. NoMaD was attempted to be integrated, but did not show very promising results.

A camera based waypoint prediction is much preferred as mentioned in section 6.2.1. A potential future work for this thesis would be coming up with image-based waypoint proposals, something very close to NoMaD, making sure the waypoints are collision free. This version of local planner will consist of images at specific locations as nodes, and does not incorporate GPS coordinates at all while creating the graph, just like in ViKiNG. GPS coordinates would be used just for retrieving the pixel coordinates in top-down raster map and compute the distance necessary for computation of goal heuristic.

But, for this image-based local planner to work efficiently, it would require an array of cameras

(at least two: front and rear). The robot would see in forward direction while driving forward, but the perspective would change completely while backtracking as it needs to drive in an opposite direction.

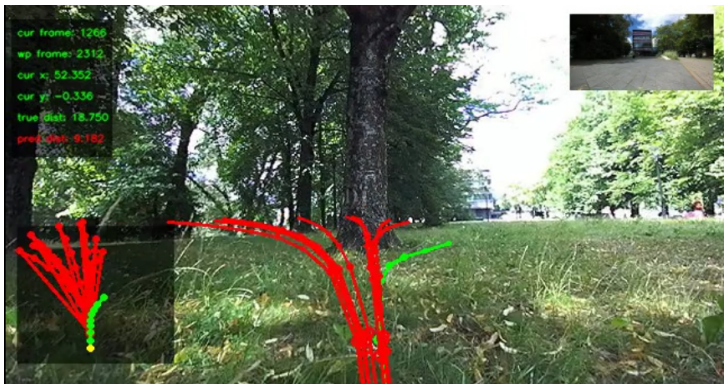



Figure 6.2: NoMaD failing to propose collision free waypoints: Waypoints proposed by NoMaD (in red) in front of trees do not exhibit complete avoidance of collision. Few waypoints seem to avoid the tree, but not all. Image taken from video[13] [27].

### 6.3.2 Global Planner Remedy

As discussed in section 6.2.2, at times the global planner prediction for probability map elongates in both directions. A potential extension for this thesis can also be training deep learning models that would predict the proabability map where the high probability regions will be directed only towards the goal from the robot's current position. This would minimize the issue of robot traversal in other directions rather than the goal.

[13]NoMaD waypoint proposal video.

# 7 Conclusion

This thesis touches up on an approach for robot navigation, which gets rid of the usage of geometric maps which can be an overhead while navigating outdoors, especially in situations where the world around the robot is changing more often. Creating a sparse and light weight map in drivable areas in terms of a densely connected graph with some geographic cues with a goal heuristic makes this method a strong alternative to map-based navigation.

Based on the results obtained from Gazebo simulation, the physical A* outperforms the naive approach. It demonstrates that the physical A* algorithm can navigate around in a difficult obstacle configuration and does that in less number of time steps. Also, physical A* is able to navigate in difficult terrains that a naive algorithm fails to complete.

While drawing conclusions from the raster map simulation, the physical A* algorithm makes the robot follow a more directed trajectory towards the goal starting from the robot's initial location. The naive approach shows exploratoty behavior with limited number of waypoints, driving towards the least cost waypoint from that pool. Depending on where the robot is facing and where the latest batch of waypoints are directed, the number of time steps might vary for the two approaches. Comparing the performace of the two approaches, physical A* apparently prevents the random exploration and makes the robot point towards the goal.

# Acknowledgement

I would like to express a sincere gratitude towards all individuals who played an important role in completion of my Master's dissertation at the University of Tartu.

Firstly, I would like to extend my deepest thanks to my thesis supervisor, Tambet Matiisen for his unwavering mentorship and guidance throughout the academic journey. His timely feedbacks, suggestions and opinions have proven to be invaluable in navigating the research in a proper direction and uplifting the quality of the thesis overall. Also, I would like to thank him for helping me translating the abstract to Estonian.

This dissertation is a continuation of the Milrem AIRE project [27], an industrial collaboration project between Milrem Robotics and Autonomous Driving Lab (ADL), Institute of Computer Science, University of Tartu. I would like to thank Milrem Robotics and AIRE for funding the research which led me to my thesis topic. Also, I would like to express huge appreciation and gratitude to ADL for setting up a place for me to work, helping me sharpen my robotics skillsets and maintaining a positive atmosphere around at all times.

I referred to some works from the Milrem AIRE project in this dissertation. So, I would also like to thank Romet Aidla for getting the global planner models working alongside easy plug-and-run rasterio APIs which was straightforward to use.

For the oppotunity offered to me to get myself enrolled in the Masters program in Robotics and Computer Engineering, I would like to thank the University of Tartu. I would also like to express a deep gratitude towards my friends and family for their unconditional support and understanding through out my entire journey.

Last but not the least, I would like to thank OpenAI for creating ChatGPT [26] which played a crucial role in creation of the thesis, especially rephrasing the texts and providing appropriate vocabulary to express proper writing thoughts into the dissertation.

# References


[1] Thrun, Sebastian. "Probabilistic robotics." Communications of the ACM 45.3 (2002): 52-57.

[2] Zhu, Yuke, et al. "Target-driven visual navigation in indoor scenes using deep reinforcement learning." 2017 IEEE international conference on robotics and automation (ICRA). IEEE, 2017.

[3] Shah, Dhruv, and Sergey Levine. "Viking: Vision-based kilometer-scale navigation with geographic hints." arXiv preprint arXiv:2202.11271 (2022).

[4] Shah, Dhruv, et al. "Rapid exploration for open-world navigation with latent goal models." arXiv preprint arXiv:2104.05859 (2021).

[5] Shah, Dhruv, et al. "Ving: Learning open-world navigation with visual goals." 2021 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2021.

[6] Xiaochen Qiao. PathFinding.js: Online graph visualization tool. `https://qiao.github.io/PathFinding.js/visual/`. Accessed: 2024-05-18.

[7] Zeng, Jun, Bike Zhang, and Koushil Sreenath. "Safety-critical model predictive control with discrete-time control barrier function." 2021 American Control Conference (ACC). IEEE, 2021.

[8] Singletary, Andrew, et al. "Comparative analysis of control barrier functions and artificial potential fields for obstacle avoidance." 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2021.

[9] Tallysman Wireless Inc. GNSS Positioning Techniques. `https://www.tallysman.com/gnss-positioning-techniques/`. Accessed: 2024-05-18.

[10] GIS Geography. UTM - Universal Transverse Mercator Projection. `https://www.tallysman.com/gnss-positioning-techniques/`. Accessed: 2024-05-18.

[11] NetworkX. NetworkX Reference Documentation. `https://www.stereolabs.com/docs/ros/depth-sensing`. Accessed: 2024-05-18.

[12] Rasterio. Rasterio Reference Documentation. `https://rasterio.readthedocs.io/en/stable/`. Accessed: 2024-05-18.

[13] Shapely. Shapely Reference Documentation. `https://shapely.readthedocs.io/en/stable/`. Accessed: 2024-05-18.

[14] ONNX Runtime. ONNX Runtime Reference Documentation. `https://onnxruntime.ai/docs/`. Accessed: 2024-05-18.

[15] ONNX. ONNX Homepage. https://onnx.ai/onnx/index.html. Accessed: 2024-05-18.

[16] ROS Wiki, ROS Noetic Ninjemys Documentation. http://wiki.ros.org/noetic. Accessed: 2024-05-18.

[17] CVXPY. CVXPY Reference Documentation. https://www.cvxpy.org/. Accessed: 2024-05-18.

[18] Anaconda, Inc., Conda Reference Documentation. https://docs.conda.io/en/latest/. Accessed: 2024-05-18.

[19] OpenCV Contributors. OpenCV Reference Documentation. https://docs.opencv.org/4.x/. Accessed: 2024-05-18.

[20] Chris Veness. Calculate distance and bearing between two Latitude/Longitude points using Haversine formula. https://www.movable-type.co.uk/scripts/latlong.html. Accessed: 2024-05-18.

[21] Clearpath Robotics. Clearpath Robotics Documentation: Simulating Jackal. https://docs.clearpathrobotics.com/docs/ros1noetic/robots/outdoor_robots/jackal/tutorials_jackal#simulating-jackal. Accessed: 2024-05-18.

[22] SICK. LMS1xx. https://www.sick.com/tw/en/catalog/products/lidar-and-radar-sensors/lidar-sensors/lms1xx/c/g91901. Accessed: 2024-05-20.

[23] u-blox, NEO-M8 Series https://www.u-blox.com/en/product/neo-m8-series Accessed: 2024-05-20.

[24] Shah, Dhruv, et al. "Gnm: A general navigation model to drive any robot." 2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023.

[25] Sridhar, Ajay, et al. "Nomad: Goal masked diffusion policies for navigation and exploration." arXiv preprint arXiv:2310.07896 (2023).

[26] OpenAI. OpenAI: ChatGPT: Language model for dialogue. https://openai.com/chatgpt. Accessed: 2024-05-18.

[27] Autonomous Driving Lab. Milrem Visual Offroad Navigation GitHub Repository. https://github.com/UT-ADL/milrem_visual_offroad_navigation/#off-policy-results. Accessed: 2024-05-18.

# Appendices

## I External Web Sources Used

**Online graph visualization**: Online tool for visualizing graphs [6]

**Global Navigation Satellite Systems Positioning Concepts**: Information and image source for GNSS positioning techniques [9]

**UTM projections**: Information and image source for how UTM works [10]

## II Tools

**Github**: Code source for some programs working out of the box some some applications

**Gitlab**: Project repository for contiuos integration continuous development

**Python wiki**: Information source for python programming language

**Geeks for Geeks**: Information source for general code implementations

**W3Schools**: Information source for general code implementations

**Stack Overflow**: Information forums for debugging commonly appearing code issues

**Ask Ubuntu**: Debugging tool and information source for ubuntu software packages, installation and unmet dependedncy errors

**Grammarly**: Writing assitance tool

**Arxiv**: Information source for reasearch papers

**IEEE**: Information source for research papers

**google scholar**: Source for creating research citations

**Wikipedia**: General information source

**ChatGPT3.5**: Writing and debugging tool, used in various layers of thesis- writing, rephrasing, creating boilerplate codes, etc. [26]

# III. Code Repository

The codebase to reproduce all works expalined in this thesis can be found at:
https://github.com/AnishShr/physical_astar

The GitHub readme file will guide all readers on setting up the environment and running the files necessary for reproducing the same(or similar) results.

# IV. Licence

## Non-exclusive licence to reproduce thesis and make thesis public

I, **Anish Shrestha**

1. grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, my thesis

   **"Physical A*: Graph-Based Search Algorithm for Robot Navigation On-the-Go "**

   supervised by **Tambet Matiisen**

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

*Anish Shrestha*
**20.05.2024**