Trajectory planning for fish-inspired robots

A Report

Presented to

The Academic Faculty

by

Darekar Akshay Yuvraj

1911MT05

Under guidance of

Dr Atul Thakur

DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY PATNA

June 2021

Dedicated To My Family, Teachers and the Almighty

Acknowledgements

I would like to express my deepest appreciation to my M.Tech thesis supervisor Dr. Atul Thakur for giving me the opportunity to do this wonderful project. I learnt lot of things both technical and non-technical and this dissertation would not have been possible without him.

I would also like to thank my seniors Aditya Ratnaparkhi, Sahil Sharma and Nischal Hoschal for suggestions and solving my queries. I would like to extend my thanks to PhD fellows of MICL Mr. Dharmveer Singh and Mr. Mukesh Singh for supporting me.

I would also like to thank all my fellow batch mates for making this thesis complete.

Finally, I would like to thank my parents, teachers and friends in supporting me through the Covid crisis and giving me hope for the brighter future.

Darekar Akshay Yuvraj

Certificate

This is to certify that the thesis entitled **Trajectory planning for fish-inspired robots**, submitted by **Darekar Akshay Yuvraj** to Indian Institute of Technology Patna, is a record of bonafide research work under my (our) supervision and I (we) consider it worthy of consideration for the degree of Master of Technology of this Institute. This work or a part has not been submitted to any university/institution for the award of degree/diploma. The thesis is free from plagiarized material.

Date:

Dr. Atul Thakur
Supervisor

Certificate of Approval

Date: .

Certified that the thesis entitled **Trajectory planning for fish-inspired robotss** submitted by **Darekar Akshay Yuvraj** to Indian Institute of Technology Patna, for the award of the degree of M.Tech has been accepted by the examination committee and that the student has successfully defended the thesis in the viva-voce examination held today.

(Supervisor) (External Examiner) (Internal Examiner)

Declarations

I certify that

a. The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.

b. The work has not been submitted to any other Institute for degree or diploma.

c. I have followed the Institute norms and guidelines and abide by the regulation as given in the Ethical Code of Conduct of the Institute.

d. Whenever I have used materials (data, theory and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the reference section.

e. The thesis document has been thoroughly checked to exclude plagiarism.

Date:

Darekar Akshay Yuvraj
Roll No. 1911MT05

Contents

List of Figures

List of Abbreviations

USV . . . . . Unmanned Surface Vehicles

UAV . . . . . Autonomous Underwater Vehicle

LOS . . . . . Line of Sight

ENU . . . . . Earth North Up Co-ordinate System

ODE . . . . . Open dynamics Engine

STL . . . . . Stereo Lithography

VO . . . . . Velocity Obstacles

List of Symbols

$\{O\}$	ENU fixed frame
$\{R\}$	Body-fixed frame of fish robot
η_o	Pose of fish robot
${}^{O}T_R$	Transformation matrix of robot w.r.t to $\{O\}$
v_o	Velocity of robot w.r.t to $\{O\}$
v_r	Velocity of robot w.r.t to $\{R\}$
η_g	Goal location
η_s	Start pose of the robot
Γ	Obstacle boundaries
T	Trajectory : Set of points from start to goal
F_o	Force in $\{O\}$
F_b	Force in body frame $\{B\}$
C	Drag coefficient matrix
τ_o	Torque in frame $\{O\}$
ψ	Attitude of robot w.r.t to $\{O\}$
ε_v	General:Vertical bearing.
ε_h	Horizontal bearing
kp_h	Proportional gain for LOS error ε_h

kv_h	Proportional gain for LOS error ε_v
Λ	Prismatic joint actuated distance
θ	Caudal fin joint angle
δ	Joint offset for caudal fin.
ω	Undulation frequency in rad/s
ϕ_v	Attitude of way_point from $\{R\}$
ϕ_h	Azimuth of way_point from $\{R\}$
r	Action distance
D	Flow direction
γ	LOS error
M	Motion primitive
$\bar{M}$	Inverse motion primitive
U	Continuous action set
U_d	Discrete action set
G	Discrete state space or the Grid
R	Scanning window or rectangle
O	Priority Queue
W	Target Waypoint

Abstract

Over the past few years, interest in fish inspired robots has increased significantly. Fish inspired robots can perform complex maneuvers and can work in cluttered environments. For applications like exploration, navigating through difficult environments and periodic operations, trajectory planning for fish robots need to be implemented taking external environment conditions into account. This report presents an optimal trajectory replanning approach for fish inspired robots particularly anguilliform-inspired robots in a dynamic environment based on the model predictive planning framework. The dynamic constraints were captured via simulations in CoppeliaSim and were expressed as motion primitives. The developed approach utilizes the motion primitives for generating search trees and a replanning algorithm based on D-star lite. Along with the position, the velocity of obstacles was also used to predict collision. After the way-point generation, a locomotion controller and line-of-sight guidance were used for trajectory tracking. The trajectory generated by the above approach was found to be dynamically feasible, collision free and optimal given every obstacle motion remains constant. This approach can help not only anguilliform-inspired robots but other bio-inspired robots to plan in a dynamic environment where they can interact with other creatures and complete tasks without colliding with them. Further, this planning approach can also be extended to 3D and fluid flow environment. It is envisaged that the developed approach can help in imparting autonomy to fish-inspired robots which can find applications in the applications like underwater rig maintenance, surveillance, and stealth operations for the military.

Chapter 1

Introduction

All living things have evolved to best suit the habitat they are living in. Four-legged animals have legs to run faster in any terrain which, a wheeled robot cannot do. Humans and monkeys have two legs which enable them to walk and climb things. In an underwater environment, various fishes have evolved with different anatomy. The fins of fishes have different shapes and sizes allowing them to swim in different styles. For example eels, sharks, ray fish and manta ray, jellyfish, etc have different locomotion. These different features are made by millions of years of evolution by adapting to the environment. Therefore why not use the design by the nature itself. This is where biologically inspired robots come in. Given their advantages, locomotion and planning for bio-inspired robots is complex. That is why working on the trajectory planning of fish-inspired robots is crucial.

1.1 Background

Bio-inspired fish robots can make complex manoeuvres in cluttered environments as a result, they have huge applications. Particularly in exploration, a fish robot can explore the underwater environment without disturbing aquatic life. This also comes with the added advantage of stealth, particularly for military applications.

In fish robots, particularly anguilliform-inspired robots have been reported [1, 2] to be applicable for inspection of underwater man-made structures where humans cannot stay for a longer time due to pressure and lack of oxygen. Robots inspired by fishes can have different anatomy as well as different applications. Fig 1.1 shows an anguilliform-inspired robot operating around a subsea structure. The ability of an anguilliform robot to move through small and

complex structures helps in reaching places inaccessible to human divers. A specific application would be regular inspection of subsea structure in the presence of other fishes. Other application would be exploring an unknown area where the robot only has a partial map around it and there are moving obstacles.

1.2 Motivation

For any application, the fish robot needs to reach its commanded destination without colliding with the surrounding dynamic obstacles, a critical consideration for anguilliform locomotion. Adding to the problem for planning considering the dynamic obstacles, the uncertainty of the environment also needs to be considered and this is where trajectory replanning is required. In this report an optimal trajectory replanning approach based on D* lite algorithm is used [3, 4] combined with velocity obstacles [5] for anguilliform-inspired robots operating in a dynamic environment based on the model predictive planning framework [6, 1, 7]

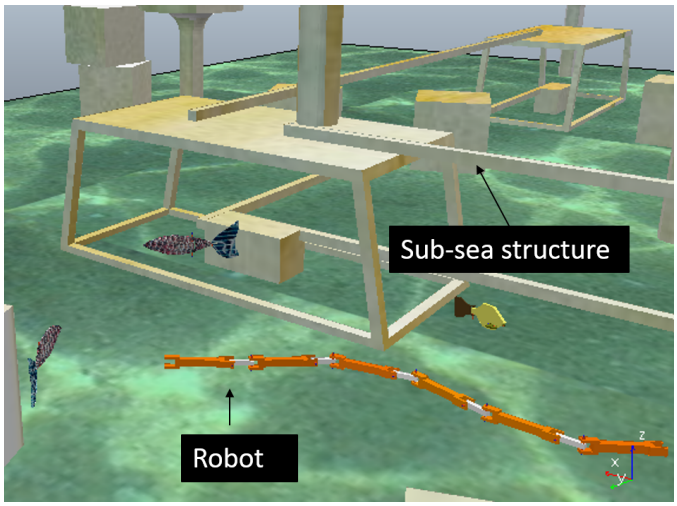
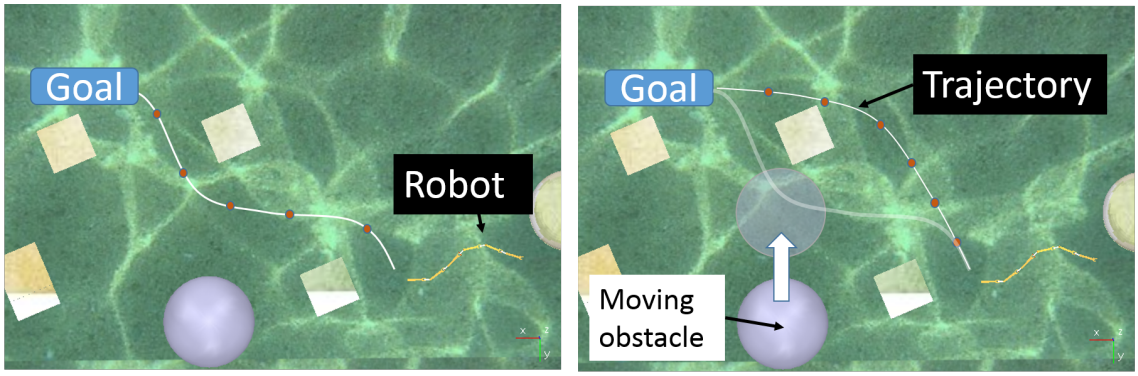


Figure 1.1: Anguilliform-inspired robot operating around a sub-sea structure

Anguilliform-inspired robots have a large number of serially connected joints actuated by coupled oscillators and thus several parameters are needed to define the locomotion controller. Therefore, generating a dynamically feasible path is one of the key challenges for such hyper-redundant robots. To handle the dynamics constraints, the model predictive framework can be used [6, 7, 1] The solutions for planning in a static environment cannot be applied to a dynamic one. The planner needs to have additional information like obstacle behaviour and robot-environment interaction to plan a trajectory. A successful planner would be able to plan

a dynamically feasible optimal path through moving obstacles. Although, the planned path is only optimal till the time the environment was predicted. Increased uncertainty demands just a feasible path around the moving obstacles where only obstacle avoidance is needed. But any environment can be considered predictable for a short period, and this information can be fed to the optimal path planner. The result is an optimal dynamically feasible path for a short period.



(a) Without moving obstacle (b) Considering moving obstacle

Figure 1.2: Anguilliform navigating towards the goal in presence of obstacles

Fig 1.2 describes two situations for an anguilliform-inspired robot. In case (a), a path is planned through the static environment. But when an obstacle (depicted as a circle) starts moving, there is a chance of collision. Hence the motion of the obstacle needs to be taken into account. Case(b) shows a path planned considering the moving obstacle which is a feasible path.

1.3 Thesis objectives

The objective of the thesis is as follows:

1. To model and simulate anguilliform-inspired robot in CoppeliaSim
2. Generation of motion primitives for anguilliform-inspired robot
3. To do real-time efficient trajectory replanning in the presence of dynamic obstacles.

1.4 Scope of thesis

The scope of the thesis is as follows:

1. Anguilliform inspired robots are considered in this thesis. Since they comprise of multiple links, the solution to their problems are also applicable for carangiform, sub-carangiform and ostraciform inspired robots.

2. Trajectory planning is currently limited to 2D. Although the methodology for extending to 3D has been only discussed in chapter 9.

3. Fluid flow is considered zero.

1.5 Thesis organisation

The thesis is organised into 10 chapters. Chapter 2 discusses the literature review on trajectory planning for fish inspired robots. The problem statement for the thesis is presented in chapter 3. In chapter 4, the methodology for modelling and simulation of anguilliform-inspired robots is shown. In the developed approach, forces acting on the robot are computed in CoppeliaSim [8, 9] environment to simulate the dynamic behaviour of the anguilliform-inspired robot. Chapter 5 discusses the generation of motion primitives and state-space discretisation for better performance. Motion primitives are generated and used to build the search tree using D^*Lite. Methods for collision detection for static as well as dynamic obstacles are presented in chapter 6. In chapter 7, the replanning approach using D^*Lite is discussed using collision detection. D^*Lite if used alone would not be able to plan a feasible path if an obstacle map is used, since the generated path might be obsolete for the future. Thus a collision function is made to account for the motion of obstacles and finding out a collision at any desired point on the map. The collision function provides a prediction of collision at any given point discussed in chapter 6. This allows D^*Lite to plan for the future as well as to update the search tree. Chapter 8 presents the results of the presented approach using certain metrics like number of expanded node, replanning attempts and success-rate. Challenges in the presented approach and their solutions are discussed in chapter 9. The methodology of extending the mentioned approach to 3D is also discussed. Chapter 10 concludes the thesis and discusses the future scope.

Chapter 2

Literature review

This chapter presents literature on trajectory planning for anguilliform robots. It also discusses the literature on planning for dynamic obstacles which do not include anguilliform robot but can be used for trajectory planning.

2.1 Trajectory tracking for fish robots

Morgansen et al.[10] used non-linear control to produce propulsion and turning gaits. Low et al.[11] created different actuation methods for different types of fish. Kelasidi et al.[12] produced a model of kinematics and dynamics of snake robot in water in 2D. Integral Line of site (ILOS) is used for guidance in 2D plane.[2]. ILOS with Artificial potential fields are used for trajectory planning for underwater snake robots [13]. Kohl et al.[14] introduced a control system for guiding an Underwater snake robot (USR) in a straight line. Makrodimitris et al.[15] presented a novel method for guiding fish using inverse dynamics.

2.2 Trajectory planning for AUVs and USVs

A large literature exists for trajectory planning for AUVs (autonomous underwater vehicle) and USVs(unmanned surface vehicle). These include approaches using fast marching methods FMM its modifications FMM^* [16], rapidly exploring random trees RRT, RRT^*, swarm-intelligence like particle swarm optimisation and firefly optimisation [17]. Arumbula et al.[18] used artificial potential fields and genetic algorithm (GA) for navigating through high-density obstacles. Niu et al.[19] used Dijkstra's and graph search methods for USVs.

2.3 Trajectory planning for fish inspired robots

In contrast to USVs and AUVs, there are only a few papers reported in the area of trajectory planning for fish-inspired robots. For underwater snake robots, Kelasidi et al. used artificial potential fields for generating way-points[13]. Although this method suffers from local minima. Raj et al. used a model predictive framework to generate motion primitives which were then used to build a search tree using A^* [1] around static obstacles. The fluid flow was also taken into account while generating the motion primitives. The aforementioned methods are only applicable for static obstacles and known environments.

2.4 Trajectory planning considering dynamic obstacles

Vector Field Histogram VFH is also popularly used for obstacle avoidance [20]. VFH^+ [21] uses a polar histogram and finds primary candidate directions and selects the direction with the lowest cost considering the kinematic constraints of the robot. VFH^* [22] combines VDH^+ and A^* to evaluate the heading direction using cost-to-come and heuristics.

Most trajectory planning approaches for dynamic obstacles involve two steps; a global path is first generated ignoring the moving obstacles and then obstacle avoidance is attempted when needed and a local plan is generated. Zongh et al. proposed a hybrid approach for trajectory planning with global plan computed using safe A* with risk assessment while 2D lidar-based replanning [23]. Chunhui et al. break down the problem of path planning as multi-modal constraints and introduces the algorithms in three stages as 1) Route Planning 2) Trajectory Planning 3) Motion Planning [24]. In the case of USVs, COLREGS (International Regulations for Preventing Collisions at Sea) is widely used while designing the obstacle avoidance system [25, 26]. Shamsuddin et al. discussed the current state of path planning for USV in terms of path planning and collision avoidance architecture incorporating (COLREGs) [27]. The methods using COLREGS will allow quick obstacle evasion but does not guarantee an optimal path. Singh et al. presented a constrained A* approach for optimal path planning in the presence of ocean currents and local planning for moving obstacles considering them as ellipses in the binary map. This approach is robust and optimal only for far encounters of obstacle and for known environments [28]. Yaghmaie et al. used the potential field approach while using the Kalman filter for predicting the obstacle movement and using an escaping algorithm for evasion [29]. Chongyang et al. for AUV used an Extended Kalman filter (EKF) for predicting the

velocities using sonar data and determine the probability of collision. Then magnitude and direction of the velocity of the vehicle are adjusted to evade the collision. This approach is similar to Velocity Obstacle.

To capture the dynamics to determine collision avoidance, the Velocity obstacles VO method is often used. It finds out the collision between two objects where if object A has a constant bearing with respect to object B or if the velocity of object B with respect to object A is heading towards B is considered as colliding. Fulgenzi et al. used a probabilistic VO approach which generated risk of collision and then used a navigation algorithm [30]. W.Zhang et al. used improved VO for dynamic obstacle where key obstacles were found with motion uncertainty [5]. Jaradat et al. used Q-learning for navigating towards the goal with dynamic obstacles given that obstacle state is known every time [31].

2.5 Trajectory planning for dynamic environments

The aforementioned methods are local planning methods and may not guarantee optimality, completeness, and dynamic feasibility while global planners may offer optimality but are often infeasible for replanning task. Andersson et al. used the Lattice-Based Motion planning approach by using A^* in receding horizon considering dynamic obstacles with wait actions [32]. Although for changes in the map, the trajectory needs to be re-planned. The replanning is computationally costly as all the information is lost thus, an incremental planner like D^* can be used for changing environments [33]. A lighter version built on LPA^* which is D^*Lite [3] is also an incremental planner. These algorithms repair only the affected part of the graph rather than building a new one. D^*Lite can be used where frequent re-routing is needed such as in logistic tasks, Cechinel et al. presents one such method for hospitals that uses D^*Lite, local planners and task scheduling where reaching in optimal time is an essential requirement [4]. Hyowon et al. used D^*Lite for quadcopter UAV for 3D path planning. Bing et al. also used D^*Lite for AUV for path planning in 3D for unknown environments. Although both of the mentioned approaches do not consider dynamic obstacles.

2.6 Research gap

To the best of our knowledge, no papers have been reported in the area of trajectory planning in the context of dynamic obstacles for fish-inspired robots. D^*Lite is suitable for unknown environments and slow-moving obstacles but cannot handle fast-moving obstacles. On the other hand Velocity Obstacles approach VO does not provide optimal path planning. The thesis aims to incorporate dynamically feasible trajectory planning in the presence of moving obstacles which none of the above-mentioned approaches provides.

2.7 Summary and key idea towards approach

Literature on various trajectory planning methods for anguilliform were discussed. Trajectory planning with dynamic obstacles were discussed for USV, AUV and other vehicles. Approaches for faster replanning like D^* and D^*Lite were also discussed. To have autonomy in traversing through a dynamic environment, the trajectory planner should be able to plan a dynamically feasible path considering moving obstacles. The trajectory planned would be only optimal until the environment is steady. Once the obstacles change their translation velocity, a new trajectory needs to be planned if the previous one is infeasible.

Velocity of obstacles can be used to predict their next states given their velocity is constant for some time thus, if velocity information of obstacles can be fed to these planners, a hybrid approach can be developed. Such a hybrid planner would be accounting for map changes, optimality and also dynamic obstacles. Therefore, D^*Lite which provides an optimal solution as well as accounts for map changes and the VO approach which considers moving obstacles can be combined and this forms the basis of the approach presented here.

Chapter 3

Problem statement

This chapter discusses the problem statement for the thesis with few assumptions. Finally, the approach for solving the problem is also presented.

Let,

1. the East North Up (ENU) whose XY plane is tangent to earth surface be a reference frame for the underwater fish robot denoted by {O} and body fixed frame {R} for robot. For Anguilliform-inspired robot {R} would be located at center of mass (p_{CM}) of its main body where, $p_{CM} = \sum_{i=1}^{N} p_i$, and p_i is centroid of i^{th} link expressed in $\{O\}$ frame. Similarly let v_{CM} be the velocity of robot. The heading angle of robot $\bar{\theta} = \sum_{i=1}^{N} \theta_i$. N is the number of links of the robot. Fig 3.1 shows the centroids, orientation and joint angles of links of robot.

2. The initial state of robot with respect to {O} is $\eta_s = [p_{CM.s}, v_{CM.s}, \bar{\theta}_s]$ and goal state be $\eta_g = [p_{CM.g}, v_{CM.g}, \bar{\theta}_g]$.

3. Γ be the set of 2D boundaries of obstacles and $\Gamma(i)$ be the boundaries of i^{th} obstacle given by an obstacle detection system, where $\Gamma \subset \mathbb{R}^2$. Any point inside Γ is considered colliding. It is assumed that the obstacle location is available at all time. These locations can be obtained using camera [34] and sonar [35]. Also Γ_v be the set of the average translation velocities of obstacle and $\Gamma(i)_v$ be the velocity of i^{th} obstacle i. The rotational velocity of the obstacle is assumed to be zero.

4. A collision function $C(\Gamma, \Gamma_v, p_c, p_{CM}, v_{CM})$ is used to determine whether point p_c is collision-free $C =$ False or not $C =$ True. This function provides a prediction whether a

given point would be collision-free or not when the robot reaches that point. The function would be discussed in detail in chapter 6, section 6.2.

5. $\alpha_i(t) = A_j \sin(\omega t + \beta_j) + \gamma$ be the desired joint angles that can drive the robot towards way-point using a Eel like gait generator[2], where $A_j = A\frac{N-j}{N+1}$ is the amplitude of oscillation of j^{th} joint from tail, ω is the joint frequency, $\beta_j = (j-1)\beta$ where β is the phase-shift at joint j, A and β are parametric constants describing the amplitudes and phase-shifts, $\gamma = k_\theta(\theta_{ref} - \bar{\theta})$ is the offset determined using a proportional controller, k_θ is the proportional gain and θ_{ref} is the desired orientation.

6. $g_p = [A, \omega, \beta, k_\theta]$ is the gait parameter vector.

Refer fig 3.2, given η_s, η_g and Γ, determine a collision free dynamically feasible optimal trajectory T such that $\forall p_t \in T$, the function $C(\Gamma, \Gamma_v, p_t, p_{CM}, v_{CM}) =$ False

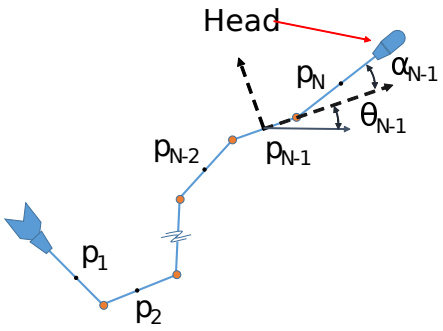


Figure 3.1: Robot model

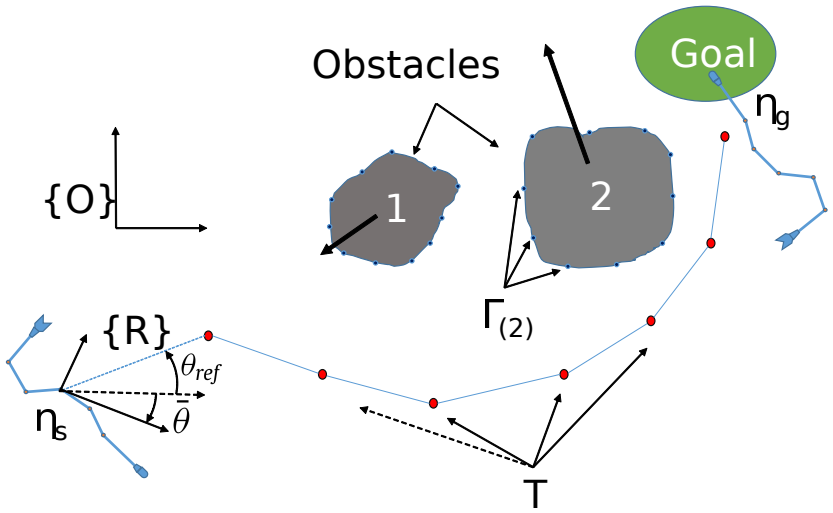


Figure 3.2: Problem statement

3.1 Overview of approach

To solve the problem of trajectory planning in presence of dynamic obstacles, the following steps are employed.

1. Forces acting on a moving submerged link are calculated and sent to the Open Dynamics Engine [36] to simulate the underwater robot. This is done in CoppeliaSim [8, 9] which is discussed in chapter 4.

2. To use lattice-based tree search, motion primitives are first generated. The state-space is discretised for better performance.

3. When the initial plan fails, the plan needs to be regenerated which is computationally costly using A^* or any other graph search methods thus D^*Lite is used for replanning. Motion primitives are modified for using it in D^*Lite since D^*Lite builds the search tree from the goal to the robot.

4. An obstacle map is only valid for static obstacles therefore a collision function is designed such that it accounts for the velocity of the obstacles to predict future collisions. This function evaluates a node whether it would be collision-free or not.

5. The edge costs are updated at the beginning of the plan and D^*Lite is used to plan the trajectory. Feasible way-points are generated and Integrated line-of-sight guidance[2] is used to pass through the way-points. The collision function is used to update the edge costs of nodes as well as continuously check for collision-free trajectory.

The approach would be discussed in detail in the next chapters.

3.2 Summary

The problem statement was discussed. The pose, goal and obstacle data was defined. The locomotion controller was also introduced. The approach for the solution is also discussed in detail in chapters 4-7.

Chapter 4

Modelling and Simulation of fish-inspired robots

This chapter presents 1) Dynamics of submerged link 2) Simulation of robot 3) Controller

4.1 Dynamics of submerged body

Consider an object or link with body fixed frame {b} moving with velocity v_o inside water. Figure 4.1 shows a submerged link on which external forces are acting on it. The equations describing these external forces are mentioned below. The major forces on body are buoyancy, viscous, added mass, drag forces and inertia given by Eqn 4.1. Let ${}^{O}R_b$ be the rotation matrix of body with respect to {O}. Eqn 4.2 is used to get the velocity in {b}.

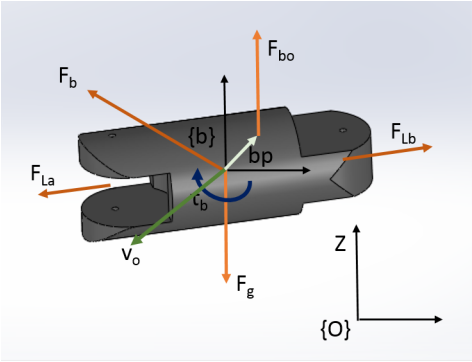


Figure 4.1: Forces on the submerged link

$$F_o = [{}^oR_b][F_b] + F_g + F_{bo} + Fl = -m[\ddot{X}] \tag{4.1}$$

$$[v_b] = [{}^oR_b]^{-1}[v_o] \tag{4.2}$$

where, F_b and F_o are applied drag forces in body frame {b} and inertial frame {O} respectively shown by eqn 4.3. v_b and m are velocity of body in frame {b} and mass of link respectively. $F_g = \begin{bmatrix} 0 & 0 & -mg \end{bmatrix}$ and $F_{bo} = \begin{bmatrix} 0 & 0 & \rho g V \end{bmatrix}$ where g and V are gravity and volume of robot respectively. Fl is link force.

$$F_b = f_D^I + f_D^{II} + f_A \tag{4.3}$$

f_D^I is linear drag, f_D^{II} is non-linear drag and f_A is added mass effect given by eqn 4.4, 4.5 and 4.6 respectively. Added mass effect is the inertia force due to acceleration of accumulated fluid on surface.

$$f_D^I = [C_I][v_b] \tag{4.4}$$

$$f_D^{II} = [C_{II}][v_b]^T[I][v_b].sign([v_b]) \tag{4.5}$$

$$f_A = [C_A][\dot{v}_b] \tag{4.6}$$

$$C = \begin{bmatrix} c_x & 0 & 0 \\ 0 & c_y & 0 \\ 0 & 0 & c_z \end{bmatrix} C_A = \begin{bmatrix} ca_x & 0 & 0 \\ 0 & ca_y & 0 \\ 0 & 0 & ca_z \end{bmatrix} \tag{4.7}$$

Referring to eqn 4.7 C is drag coefficient matrix. c_x,c_y and c_z depend on the shape of the object. In C_A matrix ca is the added mass constant.

Torque equation 4.8, 4.9 is similar to eqn 4.1 where translation velocities are replaced by rotational velocities. $\tau_{bouyancy}$ is the balancing torque which stabilises the body and is given by $\tau_{bouyancy} = [bp] \times [F_{bo}]$ where bp is the distance between C.G and B.P. τ_l is the applied link

torques, τ_b is external applied torque by fluid forces which is the sum of linear τ^I, non linear τ^{II} drag and added mass effect τ_A. J is the polar moment of inertia.

$$[\tau_b] = [\tau^I] + [\tau^{II}] + [\tau_A] \tag{4.8}$$

$$[\tau_o] = [{}^oR_b].[\tau_b] + [\tau_{bouyancy}] + [\tau_l] = -J[\ddot{\theta}] \tag{4.9}$$

The values of drag coefficients can be calculated using Computational fluid dynamics (CFD) or can be found by performing experimental procedures. Solving 4.1, 4.2, 4.8 and 4.9 for every link yields the dynamic solution for the whole robot.

4.2 Simulation of fish-inspired robot

External forces 4.1 can be added to the physics engine excluding link forces Fl such that the object behaves as a submerged body in a fluid. In CoppeliaSim the dynamic behaviour of the object in an underwater environment is achieved by writing a script in Regular API which adds external forces. In the script sim.AddForceandTorque method is used to add all drag forces and torques while sim.AddForce method is used to add buoyancy force which is applied at offset bp from C.G of the link which creates balancing torque as mentioned in equation 4.8. This offset keeps the robot in the same orientation which allows the robot to move only in a plane unless an external torque is used to roll or pitch the robot. The rest of the dynamics is handled by the Open Dynamics Engine(ODE) physics engine of CoppeliaSim which employs a Linear Complementarity Problem (LCP) solver. Figure 4.2 shows the simulation of different types of fishes.

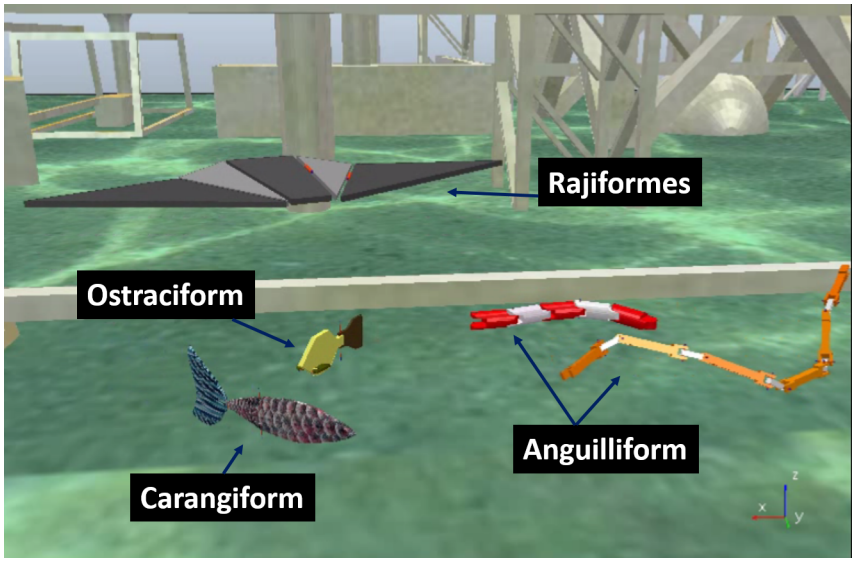


Figure 4.2: Simulation of different types of fish

The design parameters of the robot are the same as the robot developed by our research group in the past [37]. Figure 4.3 shows the CAD model of the eel robot. While the CAD model has the same dimensions and properties as the model developed in IIT Patna MICL lab[1] it is only for reference and visualisation. Although, it receives dynamic properties from the script. After adding external forces to every link of the anguilliform robot, the robot locomotion would be similar to that of the eel or underwater snake robot.

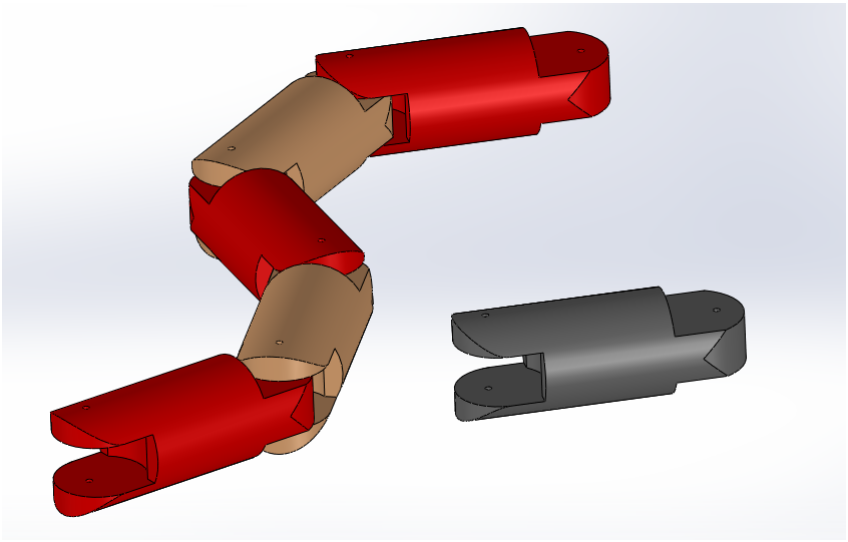


Figure 4.3: CAD model of eel robot for simulation

4.3 Controller

4.3.1 For no fluid flow condition

The controller uses integrated line-of-sight guidance for navigating towards the way-points [2] which controller employs high level and low level controller. The high-level controller determines the required joint angles using equation $\alpha_i(t) = A_j \sin(\omega t + \beta_j) + \gamma$, where γ is some proportion of bearing given by $\gamma = k_\theta(\theta_{ref} - \bar{\theta})$. Referring to fig 3.2, the reference angle is given by following equation 4.10.

$$\theta_{ref} = atan2((y_{CM,g} - y_{CM}), (x_{CM,g} - x_{CM})) \tag{4.10}$$

Refering to figure 4.4, the robot pose is start pose is η_s whose co-ordinates are x_{CM}, y_{CM} and goal is η_g whose co-ordinates are $x_{CM,g}$, $y_{CM,g}$. The low level controller is a *PD* controller which generates appropriate torques for getting the desired joint angle(α) given by eqn 4.11

$$\tau = k_p e_\alpha + k_d \dot{e}_\alpha \tag{4.11}$$

The robot links receive natural derivative feedback due to fluid drag therefore, k_d value can be taken low or even zero while the k_p value was taken 0.5. The robot uses servo motors whose high-level controller is also written in regular API for better performance while the planner code is written in Python using the Pyrep library.

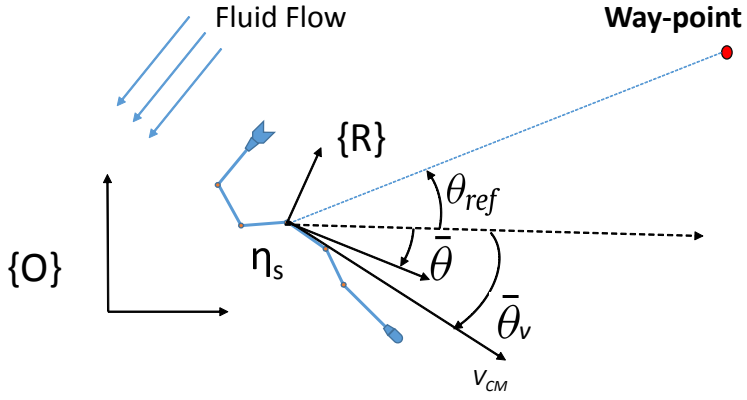


Figure 4.4: Pure pursuit: LOS guidance

4.3.2 In presence of fluid flow

In presence of flow, the high level controller has a poor performance. Due to fluid flow, the robot goes in a tail chase situation in order to reach the way-point or goal. This can be solved by using direction of velocity vector v_{CM} instead of the robot heading $\bar{\theta}$. Let this direction be $\bar{\theta}_v = atan2(v_{CM,y}, v_{CM,x})$. Therefore the error γ is given by the following equation,

$$\gamma = k_\theta(\theta_{ref} - \bar{\theta}_v) \tag{4.12}$$

where, $v_{CM,x}$ and $v_{CM,y}$ are x and y components of velocity of centroid of robot in {O}. In case of no flow, the v_{CM} points in same direction as $\bar{\theta}$ and the controller functions as before. Figure 4.5 shows comparison between using velocity vector and orientation as heading for two fluid flow 0.05m/s and 0.1 m/s. It was seen that not only it takes a shorter path but also the robot orientation is well aligned with respect to previous way-point. This allows better tree branching.

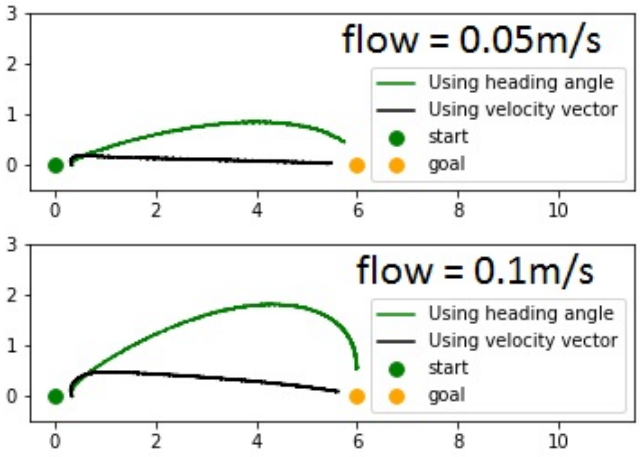


Figure 4.5: Using velocity vector instead of orientation heading

The above mentioned controllers perform pure pursuit as seen in figure 4.4. Measuring $\bar{\theta}$ is easier compared to measuring v_{CM} therefore, it would benefit to use proportional guidance where θ can be used by treating the way-point as a moving target whose velocity is same as the fluid flow but in opposite direction as seen in figure 4.6. Proportional guidance is widely used for pursuits in missile systems although, this remains a subject for future scope of this thesis.

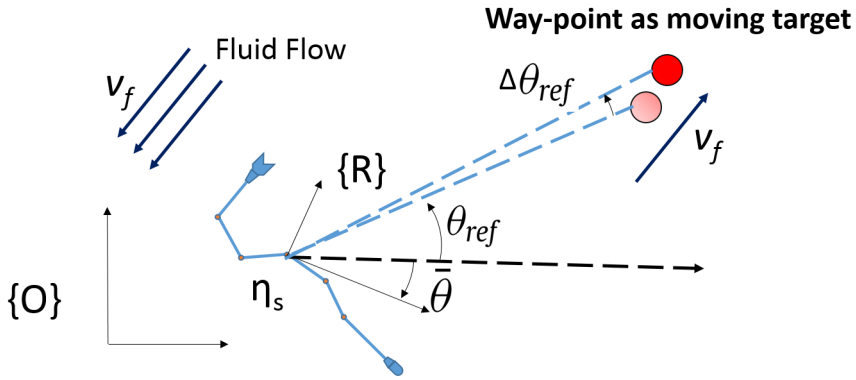


Figure 4.6: Considering way-point as moving target

4.4 Summary

This chapter discussed the modelling and simulation of the anguilliform inspired robot. External drag forces, buoyancy forces and drag torques were calculated and applied on the robot links. The remaining dynamic equation was solved in CoppeliaSim ODE. The result was a dynamic simulation of the anguilliform robot. The controller for trajectory tracking was also discussed. The controller used for two step controller integral line of site controller. The modifications in controller were also discussed for better control in fluid flow. The above to steps would now allow to generate motion primitives for tree search which would be discussed in next chapter.

Chapter 5

Generation of motion primitives and state-space discretisation

In this chapter, motion primitives are generated for D^*Lite. The state-space is discretised for better performance.

5.1 Generating motion primitives

A discretised action set $U_d \in U$ is generated wherein each control action $u_{d,k,p} \in U_d$ commands the robot to reach three way-poses in the presence of flow p. The three way-points are selected to be: $2b_L \begin{bmatrix} cos(-30) & sin(-30) \end{bmatrix}$, $2b_L \begin{bmatrix} 1 & 0 \end{bmatrix}$, $2b_L \begin{bmatrix} cos(30) & sin(30) \end{bmatrix}$. For a particular flow p, after the robot reaches the way-point its pose is recorded. The value $2b_L$ is the way-point distance or step-cost and b_L is the overall length of robot which is 0.65m in this case. $\eta_k = [x_{CM,k}, y_{CM,k}, v_{CM,k}, \bar{\theta}_k]$, where $k = 1,2,3$. The flow direction p can be represented as $[x_f, y_f]$ where, $x_f = (-1,0,1)$ and $y_f = (-1,0,1)$. Thus there are 3x3 = 9 directions and therefore there are 3x9 = 27 actions. For values $x_f = 0$ and $y_f = 0$, the flow direction $p = [0,0]$ is null and represents no fluid flow. Since robot reaches the desired location only orientation $\bar{\theta}_k$ needs to be recorded.

Figure 5.1 shows the motion primitives. Motion primitives map the parent node to the child node. Thus, using these motion primitives, a search tree can be built using graph search techniques from the robot to the goal. Figure 5.2 shows an example of a tree generated from robot to goal in presence of an obstacle. The red path indicates the dynamically feasible optimal path.

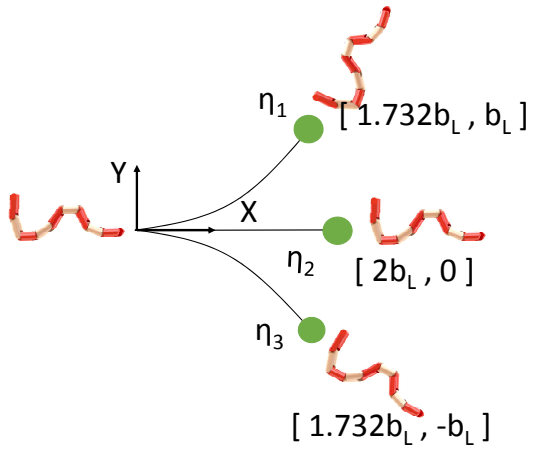


Figure 5.1: Motion primitives

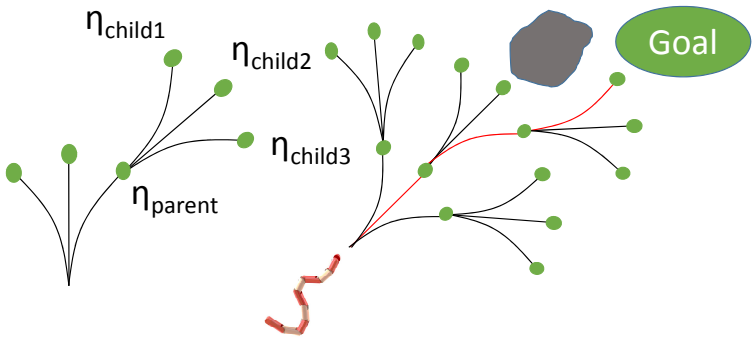


Figure 5.2: Example: Tree search from robot to goal

5.1.1 Motion primitives for D^*Lite

Motion primitive maps $\eta_{old} \rightarrow \eta_{new}$, but since D^*Lite searches the space from goal to robot, we need a mapper which maps from $\eta_{new} \rightarrow \eta_{old}$. Thus, the original motion primitives cannot be used.

Let $M_k, k = [1,2,3]$ be a transformation matrix which maps $\eta_{old} \rightarrow \eta_{new} \mid [\eta_{new,k}] = [M_k][\eta_{old}]$. M_k is formed using the co-ordinates and orientation of way-point poses and combining it into a transformation matrix. For mapping from $\eta_{new} \rightarrow \eta_{old}$ for D^*Lite/D^*, we define an inverse motion primitive $\bar{M_k} \mid [\eta_{old,k}] = [\bar{M_k}][\eta_{new}]$, where $[\bar{M_k}] = [{M_k}^{-1}]$.

Just as M_k predicts the next possible states as seen in fig 5.1, $\bar{M_k}$ predicts the previous possible states as seen in fig 5.3.

Figure 5.4 shows a expansive tree search using action set generated by inverse motion

primitives.

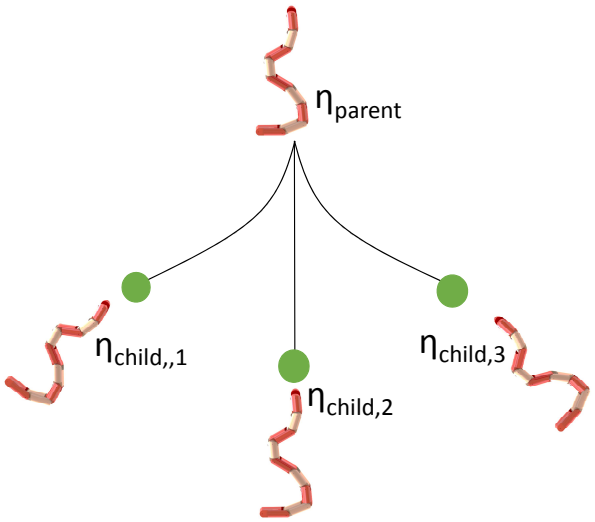


Figure 5.3: Inverse motion primitives

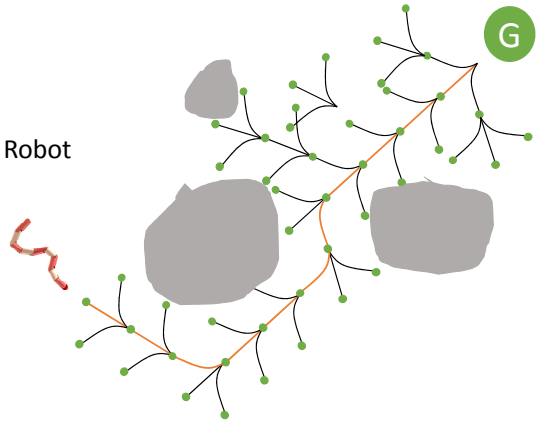


Figure 5.4: Example: Tree search from goal to robot

5.2 Generating discrete planning space

The 2D space of 14m $\times$ 14m is discretised in 75 levels. The orientation is discretised in 16 levels referring to figure 5.5. In total there are $75 \times 75 \times 16 = 90000$ states ignoring the velocity of the robot. The anguilliform-inspired robot being stretched in a wider area provides high damping which makes the robot's terminal velocity reach the flow velocity quickly. So if the flow of water is zero, the robot reaches to rest very quickly. Therefore velocity of the robot

is nearly the same as the fluid velocity and hence need not be considered in the state of the Anguilliform-inspired robot. Let the 2D discretised space, grid $G = [x][y][\theta]$ such that $x, y \in \{0, 1, ..., 69\}$ and $\theta \in \{0, 1, .., 15\}$. The grid is represented as a 3D list wherein each element point towards some discrete point in a 2D space of size 14m x 14m with some orientation 0-360 degree.

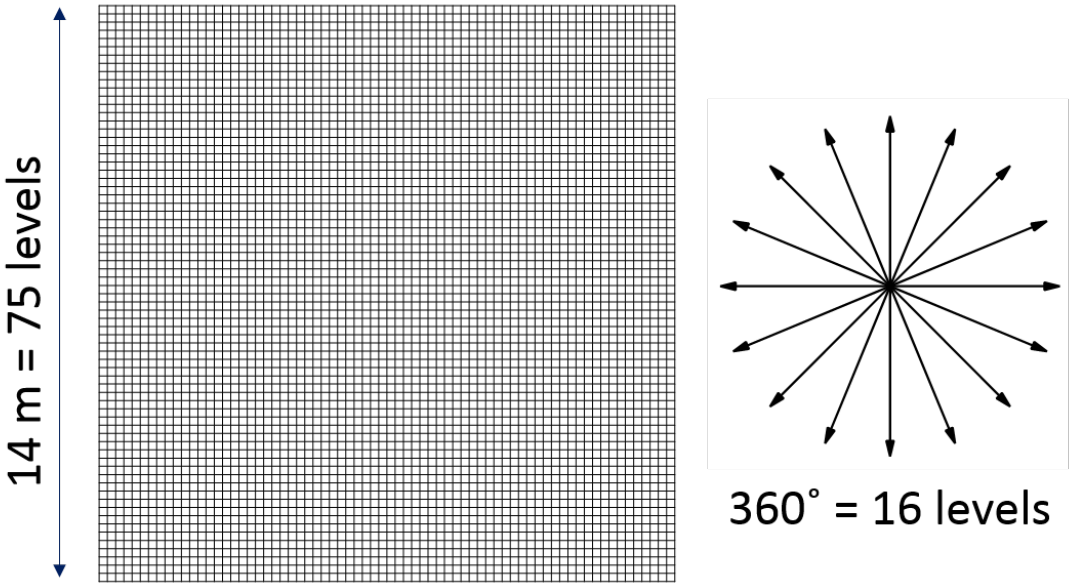


Figure 5.5: State space discretisation

5.3 Summary

Motion primitives were generated using the controller mentioned before. Since D^*Lite builds tree from goal to robot the motion primitives were modified for D^*Lite. In order to find next state, the motion primitives are represented as transformation matrix $[\bar{M}]$. The state of the robot is also represented as transformation matrix $[\eta]$. The next or previous state can be given by simple matrix multiplication $[\bar{M}][\eta]$. State-space was discretised because searching continuous space is difficult. Using this discretisation a 3D grid $x \times y \times \theta$ was made which would be used to create a search tree using D^*Lite. Further more it would also allow faster collision detection. This would be discussed in next chapter.

Chapter 6

Collision detection

This chapter discusses methods for faster collision detection for static as well as dynamic obstacles.

6.1 Collision check for static obstacles

6.1.1 Using in-built collision detection

CoppeliaSim has an in-built collision detection algorithm that uses bounding boxes for collision check. To perform a collision check for the undulating eel robot, a three-dimensional hull for each motion primitive is created. The hull shape is equal to the shape of the path traced by the robot and robots size. Therefore, such three hulls are created and then collision check can be performed using CoppeliaSim as shown in figure 6.1. The procedure for creating hulls is as follows:

1. The x,y coordinates of the robot are recorded when the robot moves for all three way-poses.
2. For each path a 3D CAD structure is created along the path with some thickness.
3. This file is converted to STL format and imported as a mesh in CoppeliaSim. This shape is not pure and therefore it has poor performance for collision detection.
4. The pure shapes are extracted from the complex shape in triangle edit mode. Pure shapes link cuboid, cylinder and sphere have good performance for detecting a collision.
5. This is done for all three hulls.

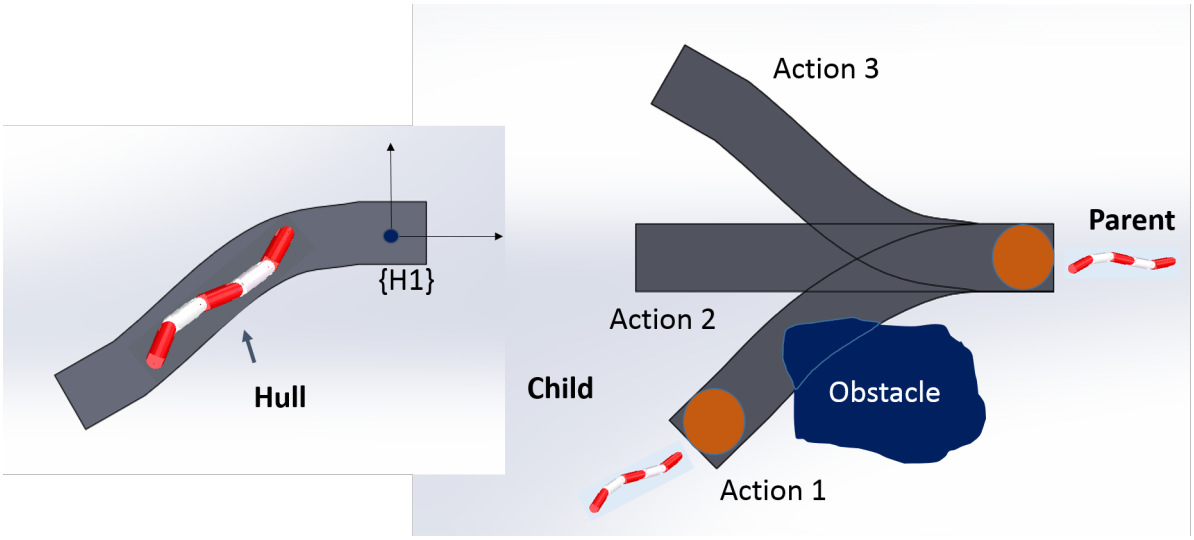


Figure 6.1: Collision checking using hulls

The hulls need re-positioning with the robot and this can be done using the following algorithm.

Algorithm 1: Collision check

1 nod **Given**: Γ, node, hulls, action_no
Result: Collision C
2 parent = node.parent : Get the parent of concerned node
3 hull = hulls[action_no] : Get the respective hull
4 Set hull {H} = parent pose η_{parent} : Position the respective hull
5 C = Do collision check for hull with all obstacles Γ

Referring to figure 6.1, the next state of the robot is denoted as a parent while the current state is denoted as a child. This is because D^*Lite is used for tree search and that it performs a search from goal to the robot. Therefore in line no {2}, we get the next state node which is the parent node. In line no {3}, we get the concerned hull from action no and in line no {4} we position the hull frame {H} to the parent pose or parent frame of the node. line no {5} uses the inbuilt collision detection system of CoppeliaSim to check the collision of the respective hull with all obstacles.

This method of collision detection is applicable for all size and shapes of obstacles and for all orientations of robot. This method can also be extended to 3D by incorporating the motion primitives in 3D space. Figure 9.2 shows the hulls for checking collision for a fish robot in 3D.

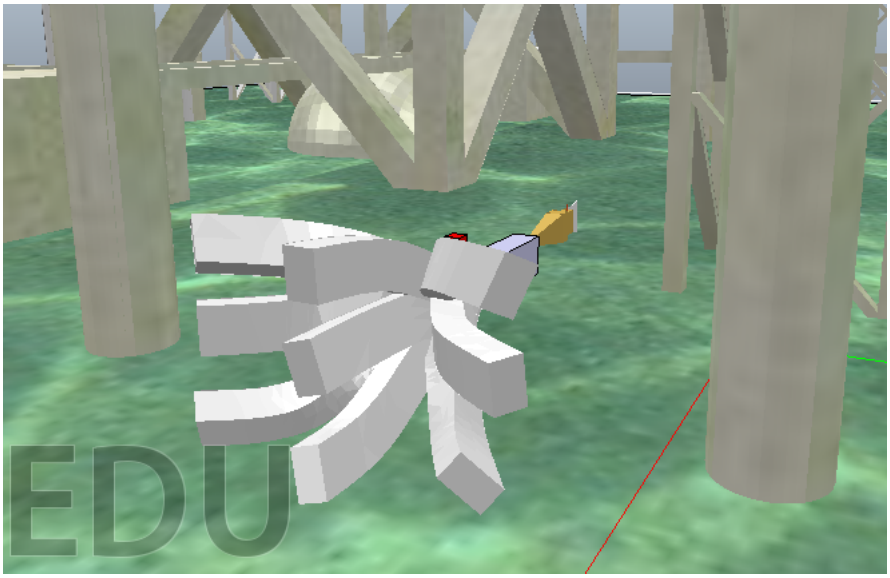


Figure 6.2: Hull used to check collision for 3D

6.1.2 Collision detection in planar space with circular hull

It is assumed that a collision detection system provides the boundary points and velocities of obstacles. Given that $\Gamma(i)$ and $\Gamma(i)_v$ be the 2D boundaries and average velocity of i^{th} of obstacle respectively. Any point inside Γ is considered colliding. Finding the point, say p_c inside a boundary Γ can be considered as a *Point in Polygon* problem, which can be solved using *Ray casting algorithm* or *Winding number algorithm*. Let a function, say *inside_boundary(* $\Gamma(i)$*,* p_c*)* uses any of the algorithm mentioned above and returns *True* if inside or else *False*. To account for the shape of the robot, a circular hull can be used and the boundary can be offset by some radius for checking collision. Although this method is not precise, it is safer and computationally faster. Fig 6.3 shows colliding points in the tan colour and a feasible path is illustrated in red colour.

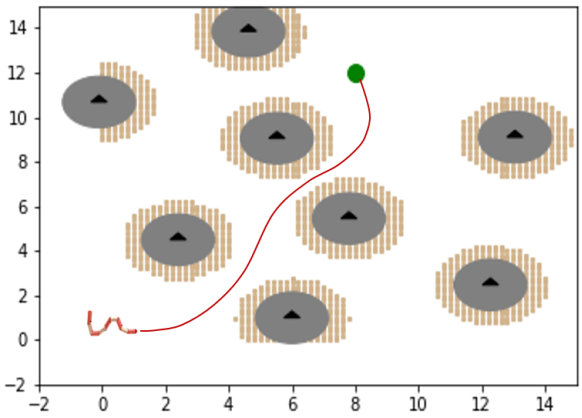


Figure 6.3: Colliding points in tan colour for static obstacles

6.2 Collision check for dynamic obstacles

A planner can use the above collision detection method to generate and plan a path, but this would not work for dynamic obstacles. Let T_τ be the trajectory generated at time $t = \tau$ considering obstacles Γ_τ. The trajectory T_τ is invalid for $\Gamma_{\tau+1}$ obstacles because there might be a collision hence a new trajectory needs to be re-planned. replanning would generate a feasible trajectory but it is only suitable for very slow-moving obstacles. Even a fast re-planner like D^*lite would require frequent replannings for moving obstacles. The reason is that the robot takes time to reach far-away trajectory points, until that time the obstacles have changed their location significantly and thus these points may be colliding in future but not in present. Thus there is a need to consider the velocity of the obstacles also.

Velocity obstacle provides collision-free velocities, however, it works in velocity space. Therefore to work in the x, y, θ space the velocities need to be integrated and is done as follows. A collision function is defined $C(\Gamma(i), \Gamma(i)_v, p_c, p_{CM}, v_{CM})$ such that $C = True$ for collision and vice-versa. The average velocity $\Gamma_v(i)$ is found by taking the mean of velocities of boundaries of each obstacle. The velocities of both robot and obstacles are integrated to find their future positions as shown in figure 6.4. In the algorithm 2, Collision line no $\{2\}$ calculates the time taken by the robot from its position to reach the point that needs to be evaluated. Given the speed of the robot, this time can be found using either a heuristic or the difference in the depth of the nodes in distance. In line no $\{4\}$, instead of translating each obstacle to its future location which is computationally intensive if a lot of boundaries are present, the point is translated with respect to the obstacle and then *point in polygon* problem or in-built collision detection in line

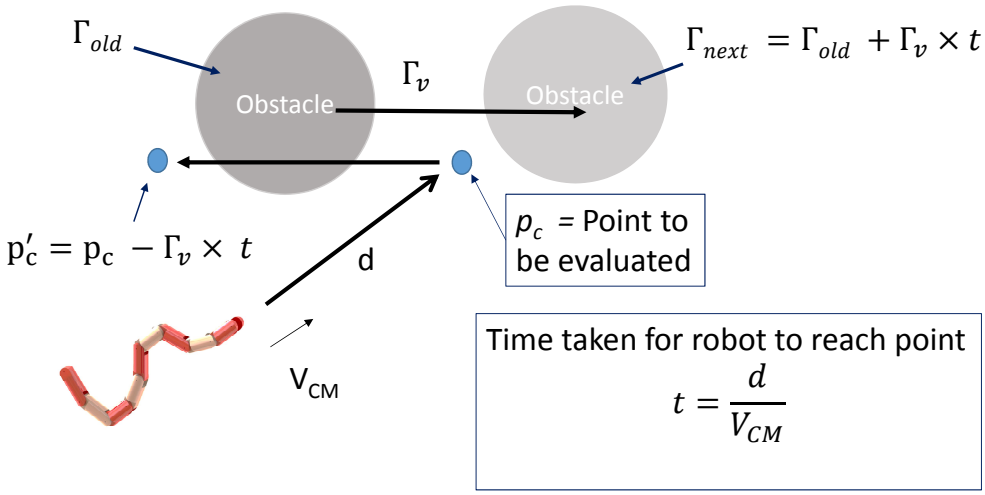


Figure 6.4: Collision prediction

no {5} can be used for finding a collision. The future scope would be to include the rotational velocity of obstacles also. In this case, an inverse transformation matrix can be applied to the point.

Algorithm 2: Collision function

nod **Given**: $\Gamma, \Gamma_v, p_c, p_{CM}, v_{CM}, d_{CM}$(Node depth in distance),d_c,h (heuristic)
Result: Collision C
$\tau = (d_{CM} - d_c)/v_{CM}$ or $h(p_c)/v_{CM}$: Get the time taken for robot to reach that point
for *Each obstacle* $\Gamma(i) \in \Gamma$ **do**
    $[p_c]' = [p_c] - \tau[\Gamma(i)_v]$: Get the translated point
    if *inside_boundary*($\Gamma(i)$, p_c') **then**
        Collision detected
        return **C = True**
    end
end
return **C = False**

Figure 6.5 shows the colliding points when we consider velocity of obstacles also. The path may seem to pass through the obstacles but they are actually collision free when robot reaches that point.

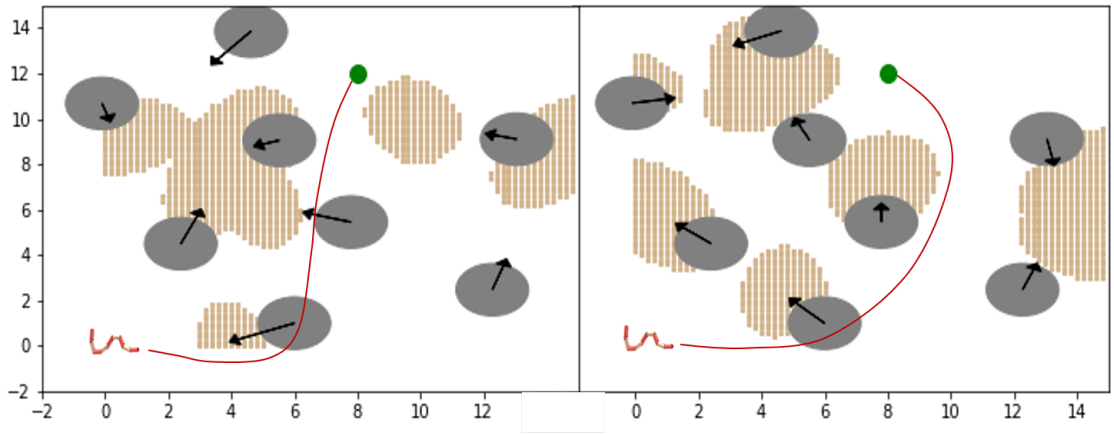


Figure 6.5: Colliding points in tan colour for moving obstacles with different velocities

6.3 Speeding up collision detection

In the case of a dynamic environment, the motion of objects away from the robot becomes more uncertain. It is also computationally costly to scan the whole map for collisions. Therefore it would be better to scan and update the edge costs of nearby nodes. Since the grid is a rectangle, a rectangular scan would be preferable, since it would be easier to navigate to all nodes. The centre of the rectangle can be on the centroid of the robot p_{CM} or can be shifted towards the direction in which the robot is headed. Figure 6.6 shows rectangle $R \in G$, whose centre R_c is shifted in front of the robot by vector r and whose width and breadth are R_x and R_y respectively.

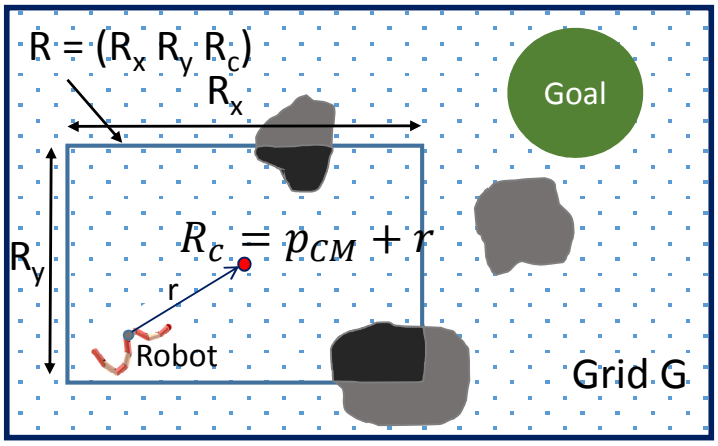


Figure 6.6: Collision check inside the surrounding rectangle R

6.4 Summary

In this chapter, a suitable method for detecting collision by static obstacles is discussed. This involved inbuilt collision detection as well using conventional 2D point in polygon solution methods. This method is expanded to collision detection for dynamic obstacles using collision function which uses collision prediction. Further, the method to improve this collision detection was also shown. Finally, this collision detection can be used to update the map when there are changes detected and this can be used in D^*lite which is presented in the next chapter.

Chapter 7

D-star Lite for replanning

When the initial plan fails, the trajectory needs to be re-planned from scratch for algorithms like A^*, RRT, DFS, BFS which are computationally costly. The advantage of planning from a goal to the robot is that only the part of the tree which is damaged needs to be repaired since the search tree remains the same from goal to robot. Thus combining incremental search and planning from goal to robot makes the D^* Lite algorithm to re-plan faster than the aforementioned methods. This chapter discusses all the algorithms to implement dynamically feasible replanning in the context of dynamic obstacles.

7.1 Data structure for node and D^*Lite

The method employed here uses an expansive tree search as in [1] and using D^*Lite first version[3]. Each state is represented as a node in the tree. The data structure is shown in figure 7.1. For a given point or a node, it stores pointers to its parent node and child nodes. It also stores edge cost to reach itself from the parent node. Infinite edge cost is assigned if the node is colliding else a step cost b_L is assigned. This edge cost may not necessarily be the same for all nodes and may even change with time depending on the situations like changing flow environments. The pose η_{CM} or state is stored which is needed for calculating the heuristic and costs of child nodes.

D^*Lite [3] uses rhs, gcost and key values which are also stored. The gcost and rhs refer to the actual distance from the node to the goal. The rhs refers to the immediate local update of this distance while the gcost refers to the actual distance. When rhs = gcost, that means the node is consistent and that the ancestors of this node are also consistent and that part of the

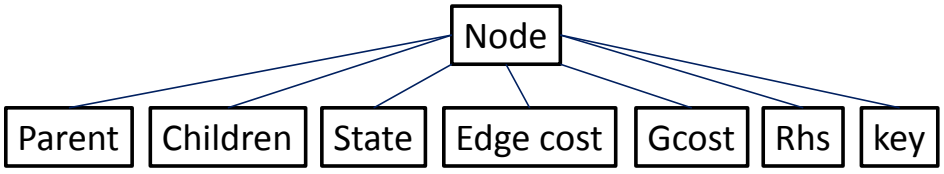


Figure 7.1: Node structure

map is properly updated. The fcost of that node is believed to be the distance from the goal to the robot passing through that node therefore, the node with the lowest fcost should be the one that is expanded in A^*. The $fcost = h + gcost$, where the heuristic h is the least cost from that node to that robot. The heuristic is needed to be admissible in all cases for D^*Lite.

If the tree structure is maintained, when a node is colliding, its child nodes and subsequently their child nodes are non-traversable. This can be related to chopping a real tree branch, whose sub-branches also fall with it when it is cut. When there are changes in the edgecosts of the nodes in the tree such as some nodes are colliding we can say that the tree is corrupted and the tree needs to be repaired. D^*Lite, repairs the tree similarly by updating rhs values using the heuristic itself. This update begins from the lowest point of the corrupted tree and passes down the hierarchy of child nodes. To achieve this the fcost is replaced by key value. The key value is a collection given by $key = [min(gcost, rhs) + h, min(gcost, rhs)]$. The first part of the key is similar to the fcost, while the second part serves as a tie-breaker when the corrupted and non-corrupted nodes have the same fcost or first part. This allows the changes to be passed from the root corrupted node.

7.2 Trajectory planning for dynamic obstacles

This section discusses the algorithms used to navigate the robot towards reaching the goal by avoiding dynamic obstacles as well as reaching the goal in the least amount of time.

7.2.1 Main algorithm

The main algorithm, Run robot() first initialises similar to Initialise() in D^*Lite in lines {2-4}. Obstacle boundaries and velocities are obtained from obstacle detection system {5} and then an initial plan is made {6}. Lines $\{7-17\}$ checks continuously for collision and navigates through way-points. In {8-12} checks for collision on the planned path T inside a rectangle R

which encompasses the robot. Line {11} Re-plans and gets a new set of way-points T once a collision is detected. Lines {13-16} navigates the robot towards the target way-point W using ILOS guidance and updates the target once the current one is completed.

Algorithm 3: Run robot

1 **Given**: $p_{CM,s}, v_{CM,s}$,Grid(G),O = empty
Result: Goal reached
2 $\forall$ Nodes $\in$ G set parent = 0, gcost=rhs= ∞
3 $s_{goal}.rhs = 0$
4 $O \leftarrow$ insert(s_{goal},CalcKey(s_{goal})
5 Capture Γ, Γ_v //Obtain obstacle data
6 T = Compute_Shortest_Path($\Gamma, \Gamma_v, p_{CM}, v_{CM}$,O) // Return list of way-points
7 Define Rectangle R encompassing the robot
8 **while** $p_{CM} \neq p_{CM,g}$ **do**
9 Capture Γ, Γ_v
10 **if** $C(\Gamma, \Gamma_v, p_w, p_{CM}, v_{CM})$== *True* $\forall p_w \in T \cup R$ **then**
11 T = *Re-plan*($\Gamma, \Gamma_v, p_{CM}, v_{CM}$,O,R))
12 W = $T \leftarrow pop()$
13 **end**
14 *Navigate towards W*
15 *Update and centre the rectangle R with respect to robot*
16 **if** *Reached W* **then**
17 W = $T \leftarrow pop()$
18 **end**
19 **end**

7.2.2 Replanning

The Re-plan function is a part of the main function of D^*Lite. Only the nodes inside the rectangle which encompasses the robot are considered because the nodes away from the robot are uncertain This also decreases the computation time. It updates the values (edge cost, rhs, gcost) of changed nodes and computes the path and returns the new plan T.

The Compute_Shortest_Path() is almost similar as in D*Lite except line {4-6} and {13-15}. Goal condition {4} is used to determine whether the current node is near the robot. Compared to the conventional method of waiting until the start node is consistent, this method allows tolerating minor orientation errors otherwise the tree needs to expand until it tree reaches the robot with the exact desired position and orientation which is computationally costly. Line no {5} returns the trajectory using a hierarchy of nodes. The predecessors(in terms of movement direction) of a node are referred to as children. Only feasible children i.e which do not collide

Algorithm 4: Re-plan

```
1  Given: Γ, Γ_v, p_CM, v_CM, O, R
   Result: T
2  Scan for graph changes inside a rectangle R
3  if edge cost changed then
4      for Each node with changed edge cost do
5          update the edge cost c(u)
6          UpdateVertex(u)
7      end
8  end
9  T = Compute_Shortest_Path(Γ, Γ_v, p_CM, v_CM, O))
```

are obtained and for every child, only one parent is allowed similar to A^* in line {14}. This allows the graph changes to be passed down faster and also reduces the length of the priority queue.

Algorithm 5: Compute_Shortest_Path

```
1  Given: Γ, Γ_v, p_CM, v_CM, O
   Result: T
2  while O ≠ empty do
3      u = O←pop()
4      if Goal_Condition(u)==True then
5          return GetTrajectory(u)
6      end
7      if u.gcost > u.rhs then
8          u.gcost=u.rhs
9      else
10         u.gcost= ∞
11         UpdateVertex(u)
12     end
13     children = GetChildren(u)
14     ∀ child ∈ children:
15     UpdateVertex(child)
16 end
17 return Null //Failure
```

The approach used for searching is tree search where the previous states of the current node are referred as children. In tree search each node has a single parent but, there is always a possibility that two different nodes may share at least one children. The following algorithm GetChildren() returns the feasible children as well as maintains the tree structure. This reduces computation time by reducing expansions in D^*Lite. In line {5}, each child is obtained for every motion primitive $[\bar{M}]$. Line {6} and line {7}, sets the current node as the parent of that

child if the child gcost is higher than the sum of the gcost of the parent node and the step-cost. This is similar to the method in A^*.

Algorithm 6: GetChildren

```
Given: u, M̄, F
Result: children
children = {empty}
η= u.state // Get robot pose as matrix
for k ← 0 to 3 do
    p = Get the flow index for the given flow field F
    child = [M̄_{k,p}][η]
    if child.gcost>u.gcost+step-cost then
        child.parent = u
        children ← insert(child)
end
return children
```

After getting the final node, the trajectory (T) which is the list of way-points can be obtained by backtracking till the goal node. This is shown in the algorithm GetTrajectory which is the same as in [1].

Algorithm 7: GetTrajectory

```
Given: u
Result: Trajectory T
T = u.state
u = u.parent
while u ≠ empty do
    T ←insert(u.state)
    u = u.parent
end
return T
```

7.3 Summary

In this chapter, D^*Lite was implemented using the motion primitives. This allows dynamically feasible replanning. The map or tree was updated using collision function which allowed to plan for dynamic obstacles. The algorithm 3 run robot() does real time replanning as well as collision avoidance. It consisted of Re-plan algorithm 4, which updates the map and replans using algorithm 5 compute shortest path. Finally it returns a trajectory using algorithm 7

GetTrajectoy which is set of ordered way-points from robot to goal and these way-points are intermediate goals for the LOS guidance for the robot. The algorithms shown in this chapter can now be implemented in real-time in the CoppeliaSim environment for validation. The results are discussed in the next chapter.

Chapter 8

Results and discussion

This chapter presents numerical simulations performed to validate the approach. Figure 8.1 show the environment created in CoppeliaSim. The length of the links for the robot was taken as 0.13m and the number of the links was five. In the simulation, the moving obstacles were taken as spheres and their velocities were changed from -0.4m/s to 0.4m/s after certain fixed intervals, we refer to this time-span as steady-time. The robot velocity was 0.2m/s for 1.4Hz undulation frequency, $k_p = 0.5$ for high level controller, $\beta = 1.03^c$ and $\alpha = 40°$. The heuristic for D^*Lite is Euclidean distance.

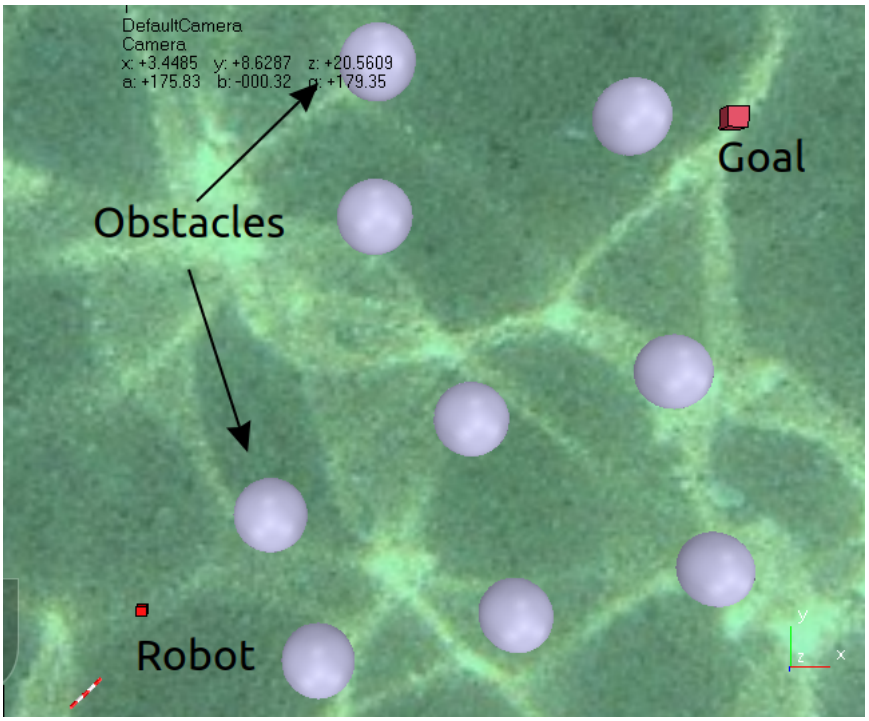


Figure 8.1: Simulation environment in CoppeliaSim

Fig 8.2 and fig 8.3 shows scenarios where an initial plan is made. Then when a collision is detected along the path, a new path is planned. In the figure 8.2 it can be seen that the planned path goes through the obstacle, although it is a feasible path since the obstacle position would have changed when the robot will arrive there.

The simulation video resources can be found by clicking **here**.

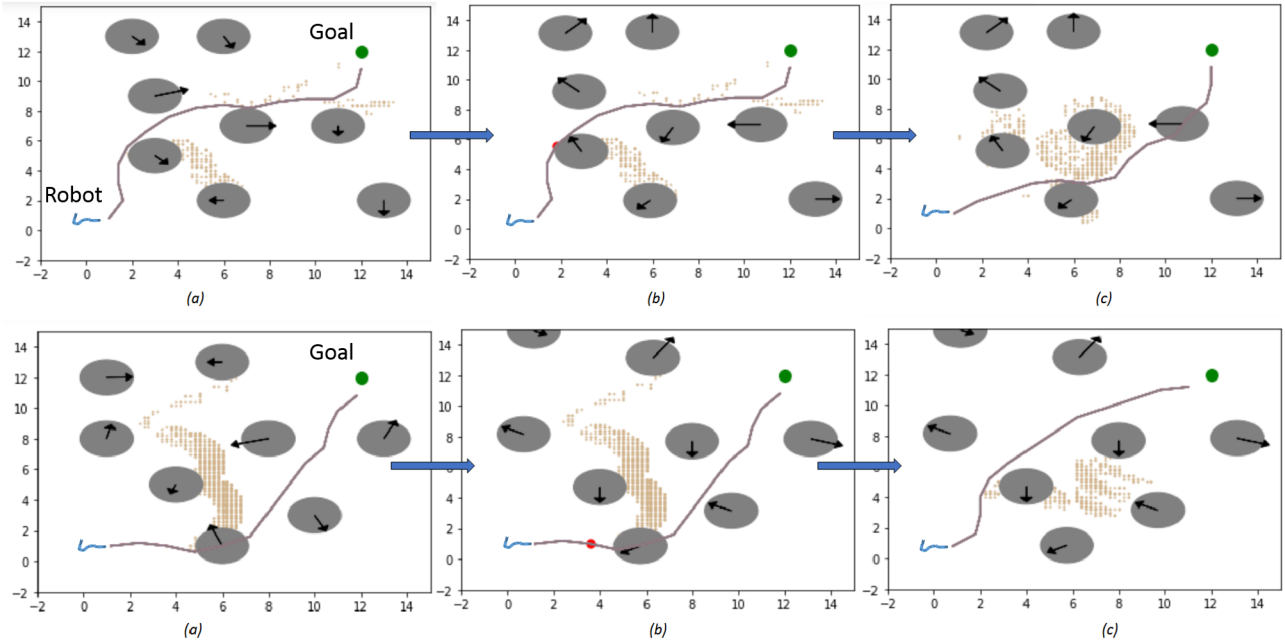


Figure 8.2: a) Planned path b) Collision detected c) Re-planned path for rectangle = full grid

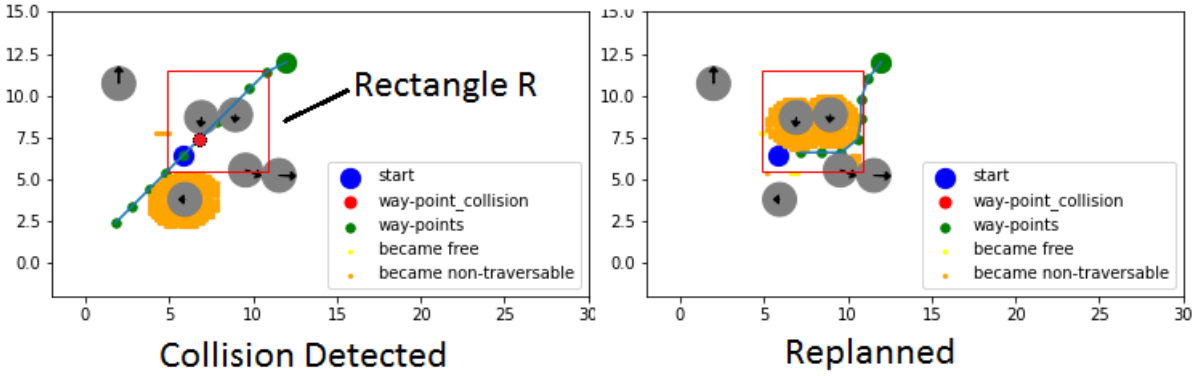


Figure 8.3: Replanning with scanning window(rectangle)

To find out the performance, the number of replanning attempts, graph/map updating time, computation time and expanded nodes while replanning for different: 1) goal located at varying distances, 2) period after which the obstacle velocities are randomly changed was determined, 3) scanning window size, 4) a safe distance from obstacles, and 5) the number of obstacles.

The hit rate which is the ratio of the number of successful attempts to total attempts was also found. The simulation was done on a 1.7GHz Intel i3, 3rd Gen processor with Nvidia 950M graphic processor and 8GB RAM.

8.1 Goal located at varying distances

The robot was located at (1, 1) at the start position at zero orientation and steady-time was taken as 15 sec. For different goal locations (4, 4), (5, 5) $\rightarrow$ (12, 12), the mentioned parameters were measured for several simulations and their average was taken. The average replanning attempts increase with the in goal distance as seen in fig 8.4, since the probability for an obstacle to collide with the trajectory increases as the path length is large. The expanded nodes while replanning by D^*Lite in general showed an increasing trend because the search-tree spreads as goal distance increases. A similar trend was shown for a UAV by A^* based planning in receding horizon using motion primitives for dynamic obstacles and the path was dynamically feasible [32]. Although due to the large state-space of UAV, expanded nodes were found to be the order of 10^4.

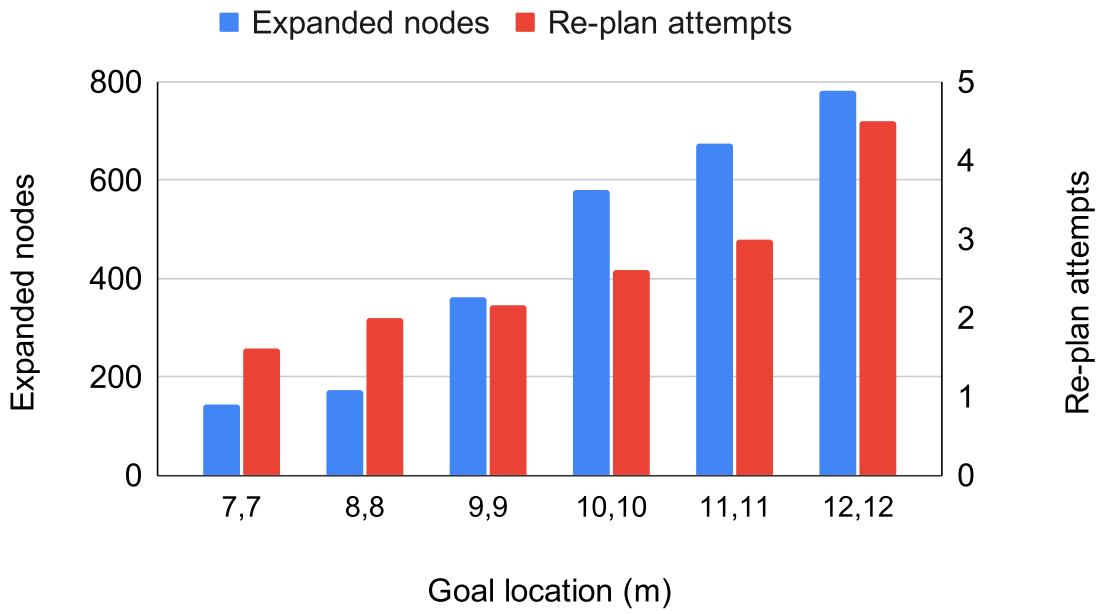


Figure 8.4: Re-plan attempts and expanded nodes for different goal locations

8.2 Different times after which the obstacle velocities are randomly changed.

The obstacles were created randomly with uniform distribution and also given random velocities. To mimic uncertainty in the environment, after certain intervals of time the velocities of all obstacles were randomly changed. For various time intervals (steady time), the replanning attempts were measured. The goal is placed at (12,12).

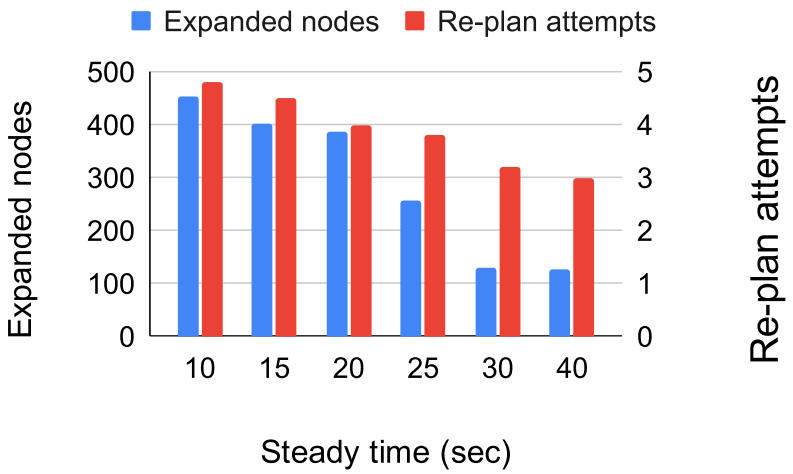


Figure 8.5: Re-plan attempts and expanded nodes for different steady-time-intervals

As the time intervals increase, the uncertainty decreases. From fig 8.5 the replanning attempts decrease with an increase with time-interval although this decrease is significant. This may be due to changes in velocities of obstacles because of collision among themselves. The expanded nodes showed decreasing trend as uncertainty decreases this may be due to lesser changes in edge costs in nodes. The replanning attempts are also determined by the density of the obstacles. A path through a dense region of obstacles though optimal will be more likely to be re-planned, thus obstacle density can also be factored in while planning the path.

8.3 Hit rate for different steady times

The hit rate was measured for different steady times for goal kept at (12,12) for different obstacle speeds. The hit rate relies on real-time measurement of obstacle data as well as simulation with the same speed as the real world. This way the hit rate can be accurately

measured. To properly measure hit rate, other programs using resources must be shut down. For high performance, it is recommended to split the controller hardware and planning hardware.

Referring to fig 8.6 for low obstacle speeds and less uncertainty or more steady time, the hit rate was high as 95% which decreased to less than 30% for 20sec steady time. Also, for high obstacle speeds that were faster than the robot, the hit rate was below 30%. The factors behind the low hit rate must be due to the high speed of obstacles, nearby collision, sudden change in speed and inaccurate time estimation to reach the nodes. Given that computation time is t_c and d_i is the distance to target/next way-point from i^{th} obstacle let $t_h = d_i/\Gamma_{v,i}$ be the time for the obstacle to reach that way-point, then there would be a collision if $t_c > t_h$. Due to collision or sudden change in speed in nearby obstacles may result in $t_c > t_h$. The problem of uncertainty can be addressed via the Markov decision process and stochastic dynamic programming can be used to solve it for obtaining a safe trajectory [7]. The collision function uses a euclidean heuristic to find out the time taken to reach the node thus if the path is curved, the time estimate would be wrong and thus the collision prediction.

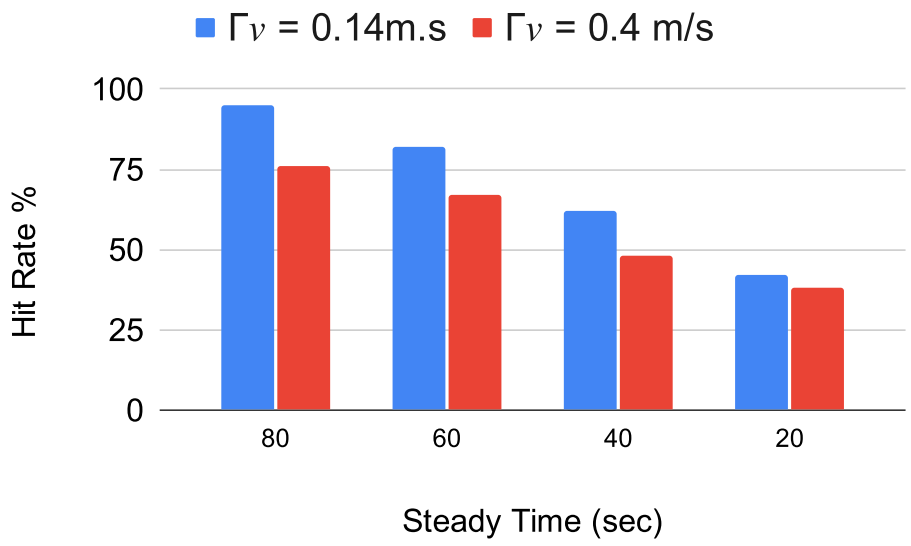


Figure 8.6: Steady time versus hitrate

8.4 Measurements for different window size

The graph update time, computation time, expanded nodes and the number of replanning attempts were measured for different window size that is the rectangle R for updating the map.

Window size(m)	Update (ms) time (ms)	Replanning attempts	Replanning time(ms)	Number of Expanded nodes
3	211	2.16	11.4	133
4	353	2	35.7	132
5	391	3.25	43.37	125
6	662	3	81.58	168
7	799	3.2	80	196

Table 8.1: Measurements for different window size

Here, size refers to both dimensions of breadth and width of the rectangle. The number of obstacles was 6, steady time was 20sec and the diameter of obstacles were 2.3mts. Table 8.1, primarily shows how the graph update time changes with changes in the scanning window size. Scanning and updating the whole map is computationally costly also, the motion of the obstacles away from the robot cannot be measured accurately. Referring to figure 6.6 the part of the trajectory away from the robot is more likely to be non-traversable and uncertain therefore instead of updating the whole map, only the map around the robot can be updated. It is seen that the update time decreases as the window size decreases. While the other measures like replan attempts, replanning time and expanded nodes also decrease with window size but the decrease isn't significant. It can be noted that as the window size decreases the planner shifts towards the local planning side and as window size increases the planner becomes more of a global planner.

8.5 Safe distance with obstacles versus hit rate and replanning attempts

To account for the uncertainty, the obstacle size can be offset and increased by some distance. This way the planner treats them as large obstacles and plans a safe path. The parameters that are affected are the replanning attempts and the hit rate. The window size is 3m, the size of the obstacle is 2.3m and the steady time is 20sec.

As seen in figure 8.7, the replanning attempts decrease as the safe distance increases. This indicates that the planner is accounting for uncertainty but the hit rate first increases and then decreases. The hit rate first increases because of the safe plans but, it starts decreasing afterwards because the apparent size of the obstacles(obstacle size + safe distance) increases

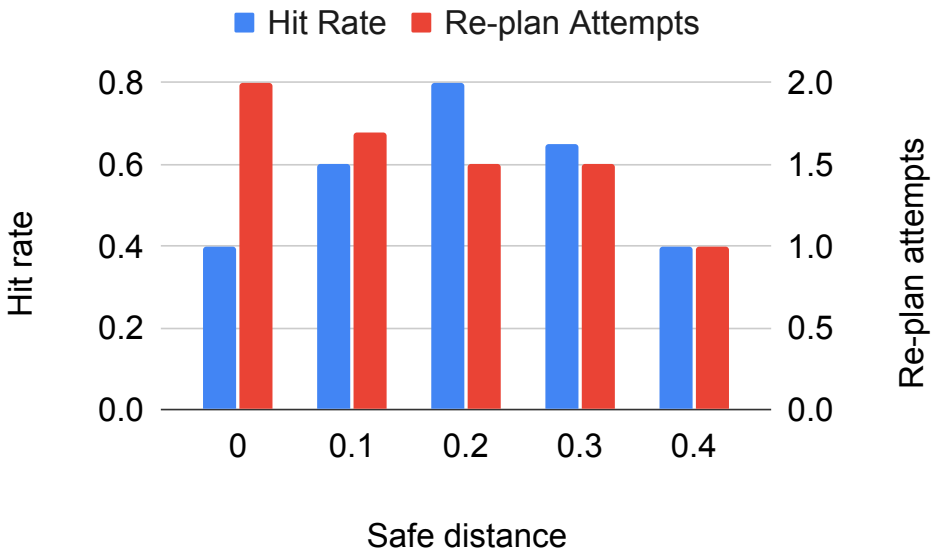


Figure 8.7: Safe distance versus hitrate and re-plan attempts

which either envelop the area around the goal or the area around the robot. This does not allow the search tree to expand to reach the robot.

8.6 Measurements for different number of obstacles

The number of obstacles was changed and measurements were made. Referring to table 8.2, it is seen that the hit rate decreases as the number of obstacles increase while, the update time, computation time and expanded nodes. Update time increases because of the increased time in collision detection while expanded nodes increases due to high branching due to more obstacles.

Number of obstacles	**Hit Rate**	**Update time(ms)**	**Replanning time(ms)**	**Expanded Nodes**
8	0.85	243	23.6	159
10	0.65	350	187	471
12	0.5	359	217	767
14	0.5	479	281	653

Table 8.2: Measurements for different number of obstacles

8.7 Using the above analysis to simulate for best performance

From the measurements, it is seen that the scanning window size of 3m and safe distance of 0.2m perform well subjected to the same simulation environment. But for planning with a low safe distance like 0.2m is highly susceptible to velocity changes in nearby obstacle and collision may occur. These parameters may change in real-world scenario depending on better processor and obstacle detection system.

8.8 Summary

This chapter discussed the performance of the proposed approach. Different measures were used to judge the performance of the approach. For dynamic environments, it was found that there is a need to standardise tests so that comparisons can be made with other approaches. Various limitations were also found. With proper improvements and using better hardware, this approach can be meliorated.

This work was published at AIR conference of July 2021 at IIT Kanpur as *Trajectory planning for Anguilliform-inspired robots in presence of dynamic obstacles*.

Chapter 9

Challenges and extensions

This chapter discusses the causes of failure and how to remove them. It also presents an approach using sampling and using bi-directional search for improving the performance. It also shows the methodology to extend the proposed approach to 3D.

9.1 Causes of failed plan

Following were the observed failure causes that resulted in a low hit rate. Finding the solution to these problems would allow better trajectory planning with a high success rate or hit rate.

1. When the velocity of obstacles near the obstacles changes significantly, the robot is unable to plan a trajectory in time and collision happens. This can be solved by switching to a local planner like velocity obstacles in such a scenario.

2. When a collision is predicted at the start node, no amount of tree expansion would be able to reach the start node from the goal node. Therefore it would be always a failure. Similar is the case if a collision is predicted at the goal node, the tree is unable to expand since the root itself is colliding. This can be solved using a wait action which waits until the target nodes are free from the collision.

3. When the obstacles cover or envelope the goal or the robot, the tree cannot expand from goal to robot. Therefore at that moment, there are no feasible paths available. This case is similar to the before-mentioned cause. Although, as mentioned before, an action for waiting would allow the robot to wait until the scenario has changed.

4. If the obstacle size is less than step size, then there won't be collision detected using the solution of point in polygon problem. This is because the obstacle is lying between two nodes but not on them.

5. Final cause is the performance of the hardware itself. A faster system would allow using large scanning window size as well as provide faster replanning. This way a robust system can be made.

9.2 Need for reducing the state space and action space by sampling

Let the discretised state of 2D static environment treated as a binary image be given as $S = \{x, y\}$ where $S \in \{0, 1\}$. So the total states are $n = 2^{xy}$ which are exponential as space increases. Now for dynamic environment for discretised time t, the state is $S_t = \{x, y, t\}$. Therefore there are a total t number of environment states until time t. For such a high number of states, it is still difficult for trajectory planner to plan a faster path. In the proposed approach the state space was reduced by only considering the map around the robot which was a rectangular window therefore the number of states for new state-space has $2^{x_r y_r t}$ compared to the original 2^{xyt}. Since $x_r < x$, the state space decreased exponentially. But this method doesn't allow to consider them part of the map away from the robot therefore it would not allow the robot to plan for situations like blockade or obstacles surrounding the robot.

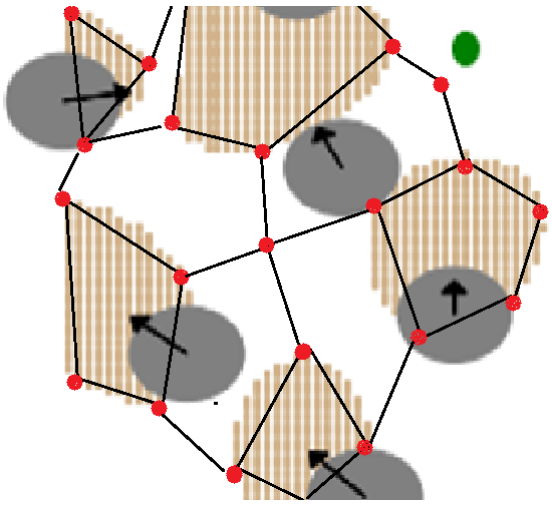


Figure 9.1: Sampling around predicted collision region

To solve this, sampling the state space around the predicted parts of the obstacles will reduce the state space significantly as seen in figure 9.1. This method may not guarantee optimality as well as dynamic feasibility and thus would require additional processing like converting the coarse sampled graph to finer tree/graph structure. This would also require re-sampling once the predicted states of obstacles change.

9.3 Using unequal bi-directional search

D^*Lite is expansion is slow when collision happens near robot. The search tree needs to expand more and connect the robot until the nodes position and orientation matches that of the robot. This is computationally costly. This can be solved by performing an unequal bidirectional search where A^*, breadth first search BFS is used to create a small tree with small limited depth. The final nodes or the open nodes of this tree are the multiple destination or goals for the D^*Lite. The heuristic needs to be modified because there are multiple goals here. Let $h(node, goal)$ be the heuristic function which returns heuristic value for the given node. Referring to figure 9.2, there is the final goal, while the red colour nodes are the multiple goals for the D^*Lite created by BFS. For such case the modified heuristic would be $h_m = max(h(node, g1), h(node, g2), h(node, g2), h(node, g3), h(node, g4))$. In the figure9.2 it seen that there is an expansion prevented which reduced the computation time. This decreases computation time significantly. Although the validation for this approach is out of scope of this thesis.

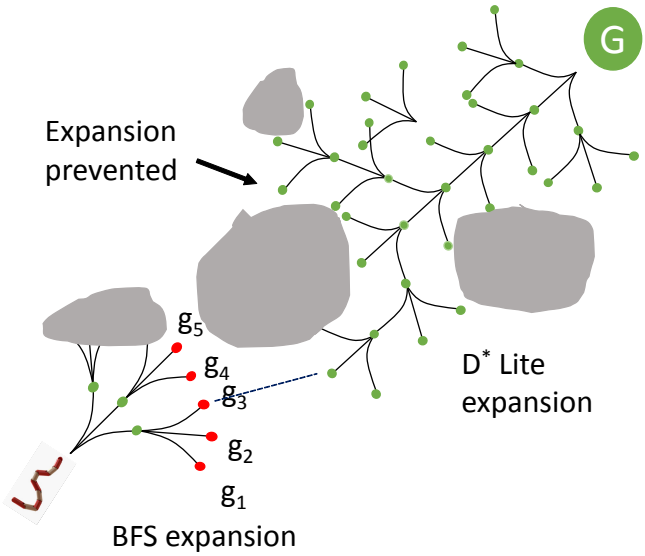


Figure 9.2: Unequal bidirectional search

9.4 Extending to 3D

This section deals with extending the trajectory planning to 3D for a carangiform robot in presence of flow.

9.4.1 Action Space Discretisation

The flow in 3D is discretised in 27 levels. Directions are indicated by direction ratios $D_p = (a,b,c)$. The values a,b,c are discretised in 3 levels each as $a = (-1,0,1)$, $b = (-1,0,1)$, $c = (-1,0,1)$. Thus there are $3 \times 3 \times 3$ directions of flow which are further encoded to integer values from 0 to 26 and p = 1,2,3...26.

The action length r is 1.5 $\times$ (Body Length = 0.4) = 0.6m. Considering the spherical co-ordinate system, we discretise the attitude $\phi_v = (-15°, 0°, 15°)$ and azimuthal angle $\phi_h = (-30°, 0°, 30°)$. The way point generated as per the spherical co-ordinate system are:

$$\begin{aligned} x_k &= r.cos(\phi_v).cos(\phi_h) \\ y_k &= r.cos(\phi_v).sin(\phi_h) \\ y_k &= r.sin(\phi_v) \end{aligned} \tag{9.1}$$

Therefore there are 3×3 waypoint poses $\eta_k = (x_k, y_k, z_k)$ and $k = (0,1,2,...,8)$. Then for each flow direction D_p p=0 to 26 and each way-point pose η_k k=0 to 8, action set $\mu_{k,p}$ of $9 \times 27 = 243$ actions is generated. Further action set is encoded to $\mu_q = \mu_{k,p}$ and $q = 0,1,2,..,242$.

The q value can be extracted as $q = (k + 9 \times p)$

Fig 9.3 shows the 9 way-point poses in no flow condition.

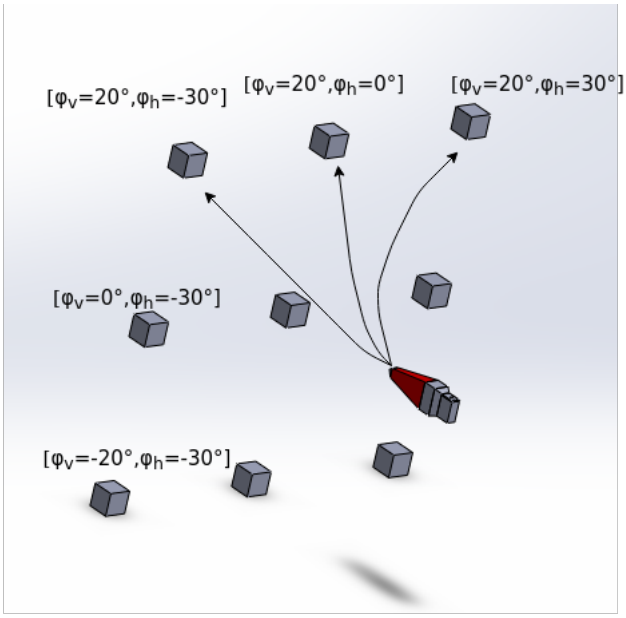


Figure 9.3: Nine 3D Waypoints

In order to use the actions, the final pose of robot η_q is stored as transformation matrix $[T_q]$ w.r.t to $\{O\}$. Thus there are 243 transformation matrices $[T_q]$ and $q = (0, 1, 2, ..., 242)$ for 242 action set μ_q. The advantage of using a transformation matrix instead of storing the pose as position and orientation are that it facilitates the generation of child nodes.

9.4.2 Generating child nodes

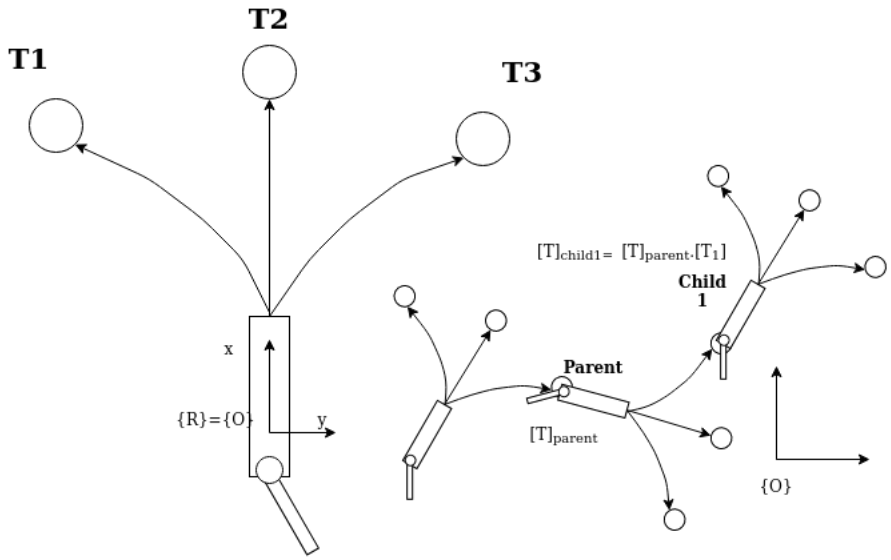


Figure 9.4: Generating child nodes in 2D

The pose of the robot is stored as a transformation matrix $[T]$. The child node can be easily generated as
Given flow direction D_p where $p = 0-26$ and for each action $k = 0-8$

$$\begin{aligned} q &= (k + 9 \times p) \\ [T_{child,k}] &= [T_{parent}].[T_q] \end{aligned} \tag{9.2}$$

Thus by simple matrix multiplication child nodes for the tree can be generated in 3D as seen in figure 9.2.

9.5 Summary

Various causes of failure for the proposed approach were discussed in the proposed approach. Method for sampling and unequal bidirectional search for improving performance were also discussed. Solving these problems would allow a better success rate. This approach can also be extended to 3D which would enable trajectory planning in 3D in real-world considering fluid flow also.

Chapter 10

Conclusion

In this report, we have presented a dynamically feasible trajectory planning technique under the presence of dynamic obstacles for Anguilliform-inspired robots using model predictive framework [6] and D^*Lite with a collision function. Motion primitives were used in generating the action set by performing dynamic simulations in CoppeliaSim and then D^*Lite was used for replanning. Since the environment was dynamic D^*Lite was able to update and repair the path efficiently. A collision function that provides the prediction for collision with moving obstacles was used with D^*Lite which reduces the number of replanning attempts. Since D^*Lite is used as a planner which is a modification of incremental A*, the path planned would be always optimal provided the heuristic is admissible. Euclidean distance was used in the proposed approach hence the solution would be optimal, given there is no external fluid flow assisting or resisting the robot. However, this optimality holds until the velocity of the obstacles remain same. Further, this approach can also be extended to 3D using transformation matrices which are motion primitives for generating tree.

The approach was tested with multiple cases. It was found that the replanning attempts increase with the number of obstacles, goal distance and uncertainty or steady time. While lower replanning attempts means good environment sensing and prediction but as the environment uncertainty increases the need for faster replanning also increases. To solve the problem of uncertainty and reduce map update time, the map/tree was updated only around the robot in a rectangular window.

10.1 Future scope

The scope of the thesis can be expanded as follows:

1. Obstacle density can be taken into account for safer and better prediction of the environment.
2. For the rapid movement of obstacles near the robot, velocity obstacles [30] approach and a specific action for waiting can be used for obstacle avoidance for the possibility where the planner could not generate a path in time for quick evasion. This way an emergency local evasive planner can be combined with the proposed approach.
3. Improving the heuristics provided, they are admissible would help to reduce the replanning time, such as heuristics that accounts for orientation and fluid flow [1].
4. Extending the planning approach in the presence of fluid flow.
5. An improved controller for trajectory tracking in fluid flow for Anguilliform-inspired robot can be used, which may reduce the travel time as well as the size of the search tree.
6. Developing a locomotion controller for an anguilliform inspired robot in 3D. Extending the planning to 3D has been discussed in chapter 9.
7. For handling large state-action space, Markov decision process and value iteration or reinforcement learning can be used for generating actions for given states [7].
8. Unequal bidirectional search can be used for improving the performance which is discussed in previous chapter.

This approach can be used in a dynamic environment where there are other fishes or moving elements for inspection of subsea structures, stealth operations, maintenance and repair.

Bibliography

[1] A. Raj and A. Thakur, "Dynamically feasible trajectory planning for Anguilliform-inspired robots in the presence of steady ambient flow," *Robotics and Autonomous Systems*, vol. 118, pp. 144–158, 2019, ISSN: 09218890. DOI: 10.1016/j.robot.2019.05.001. [Online]. Available: https://doi.org/10.1016/j.robot.2019.05.001.

[2] E. Kelasidi, K. Y. Pettersen, P. Liljeback, and J. T. Gravdahl, "Integral line-of-sight for path following of underwater snake robots," *2014 IEEE Conference on Control Applications, CCA 2014*, pp. 1078–1085, 2014. DOI: 10.1109/CCA.2014.6981478.

[3] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005, ISSN: 15523098. DOI: 10.1109/TRO.2004.838026.

[4] A. K. Cechinel, A. Luiz Fernandes Perez, P. D. Plentz, and E. R. De Pieri, "Autonomous mobile robot using distance and priority as logistics task cost," *IECON Proceedings (Industrial Electronics Conference)*, vol. 2020-October, pp. 569–574, 2020. DOI: 10.1109/IECON43393.2020.9255008.

[5] W. Zhang, S. Wei, Y. Teng, J. Zhang, X. Wang, and Z. Yan, "Dynamic obstacle avoidance for unmanned underwater vehicles based on an improved velocity obstacle method," *Sensors (Switzerland)*, vol. 17, no. 12, 2017, ISSN: 14248220. DOI: 10.3390/s17122742.

[6] T. Howard, M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Model-predictive motion planning: Several key developments for autonomous mobile robots," *IEEE Robotics and Automation Magazine*, vol. 21, no. 1, pp. 64–73, 2014, ISSN: 10709932. DOI: 10.1109/MRA.2013.2294914.

[7] A. Thakur, P. Svec, and S. K. Gupta, "GPU based generation of state transition models using simulations for unmanned surface vehicle trajectory planning," *Robotics and Autonomous Systems*, vol. 60, no. 12, pp. 1457–1471, 2012, ISSN: 09218890. DOI: 10.1016/j.

robot.2012.07.009. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2012.07.009.

[8] M. Freese, S. P. N. Singh, and E. Rohmer, *Robot simulator coppeliasim: Create, compose, simulate, any robot - coppelia robotics*, 2013. [Online]. Available: https://www.coppeliarobotics.com/.

[9] E. Rohmer, S. P. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," *IEEE International Conference on Intelligent Robots and Systems*, pp. 1321–1326, 2013, ISSN: 21530858. DOI: 10.1109/IROS.2013.6696520.

[10] K. A. Morgansen, V. Duindam, R. J. Mason, J. W. Burdick, and R. M. Murray, "Nonlinear control methods for planar carangiform robot fish locomotion," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 1, pp. 427–434, 2001, ISSN: 10504729. DOI: 10.1109/ROBOT.2001.932588.

[11] K. H. Low and A. Willy, "Biomimetic motion planning of an undulating robotic fish fin," *JVC/Journal of Vibration and Control*, vol. 12, no. 12, pp. 1337–1359, 2006, ISSN: 10775463. DOI: 10.1177/1077546306070597.

[12] E. Kelasidi, K. Y. Pettersen, J. T. Gravdahl, and P. Liljeback, "Modeling of underwater snake robots," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4540–4547, 2014, ISSN: 10504729. DOI: 10.1109/ICRA.2014.6907522.

[13] E. Kelasidi, K. Y. Pettersen, and J. T. Gravdahl, "A waypoint guidance strategy for underwater snake robots," *2014 22nd Mediterranean Conference on Control and Automation, MED 2014*, pp. 1512–1519, 2014. DOI: 10.1109/MED.2014.6961590.

[14] A. M. Kohl, K. Y. Pettersen, E. Kelasidi, and J. T. Gravdahl, "Planar Path Following of Underwater Snake Robots in the Presence of Ocean Currents," *IEEE Robotics and Automation Letters*, vol. 1, no. 1, pp. 383–390, 2016, ISSN: 23773766. DOI: 10.1109/LRA.2016.2517827.

[15] M. Makrodimitris, K. Nanos, and E. Papadopoulos, "A novel trajectory planning method for a robotic fish," *2017 25th Mediterranean Conference on Control and Automation, MED 2017*, pp. 1119–1124, 2017. DOI: 10.1109/MED.2017.7984268.

[16] C. Pêtrès, Y. Pailhas, P. Patrón, Y. Petillot, J. Evans, and D. Lane, "Path planning for autonomous underwater vehicles," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 331–341, 2007, ISSN: 15523098. DOI: 10.1109/TRO.2007.895057.

[17] D. Li, P. Wang, and L. Du, "Path planning technologies for autonomous underwater vehicles-a review," *IEEE Access*, vol. 7, pp. 9745–9768, 2019. DOI: 10.1109/ACCESS.2018.2888617.

[18] F. Arambula Cosío and M. A. Padilla Castañeda, "Autonomous robot navigation using adaptive potential fields," *Mathematical and Computer Modelling*, vol. 40, no. 9-10, pp. 1141–1156, 2004, ISSN: 08957177. DOI: 10.1016/j.mcm.2004.05.001.

[19] H. Niu, Y. Lu, A. Savvaris, and A. Tsourdos, "An energy-efficient path planning algorithm for unmanned surface vehicles," *Ocean Engineering*, vol. 161, no. May, pp. 308–321, 2018, ISSN: 00298018. DOI: 10.1016/j.oceaneng.2018.01.025.

[20] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots - Robotics and Automation, IEEE Transactions on," *IEEE Transactions on Robotics*, vol. 7, no. 3, pp. 278–288, 1991.

[21] I. Ulrich and J. Borenstein, "VFH+: Reliable obstacle avoidance for fast mobile robots," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2, no. May, pp. 1572–1577, 1998, ISSN: 10504729. DOI: 10.1109/ROBOT.1998.677362.

[22] I. Ulrich and J. Borenstein, "VFH*: Local obstacle avoidance with look-ahead verification," *Proceedings-IEEE International Conference on Robotics and Automation*, vol. 3, no. April, pp. 2505–2511, 2000, ISSN: 10504729. DOI: 10.1109/ROBOT.2000.846405.

[23] X. Zhong, J. Tian, H. Hu, and X. Peng, "Hybrid Path Planning Based on Safe A* Algorithm and Adaptive Window Approach for Mobile Robot in Large-Scale Dynamic Environment," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 99, no. 1, pp. 65–77, 2020, ISSN: 15730409. DOI: 10.1007/s10846-019-01112-z.

[24] C. Zhou, S. Gu, Y. Wen, Z. Du, C. Xiao, L. Huang, and M. Zhu, "The review unmanned surface vehicle path planning: Based on multi-modality constraint," *Ocean Engineering*, vol. 200, no. January, p. 107 043, 2020, ISSN: 00298018. DOI: 10.1016/j.oceaneng.2020.107043. [Online]. Available: https://doi.org/10.1016/j.oceaneng.2020.107043.

[25] A. Vagale, R. Oucheikh, R. T. Bye, O. L. Osen, and T. I. Fossen, "Path planning and collision avoidance for autonomous surface vehicles I: a review," *Journal of Marine Science and Technology (Japan)*, vol. i, no. 0123456789, pp. 2018–2028, 2021, ISSN:

09484280. DOI: 10.1007/s00773-020-00787-6. [Online]. Available: https://doi.org/10.1007/s00773-020-00787-6.

[26] A. Vagale, R. T. Bye, R. Oucheikh, O. L. Osen, and T. I. Fossen, "Path planning and collision avoidance for autonomous surface vehicles II: a comparative study of algorithms," *Journal of Marine Science and Technology*, no. 0123456789, 2021, ISSN: 0948-4280. DOI: 10.1007/s00773-020-00790-x. [Online]. Available: https://doi.org/10.1007/s00773-020-00790-x.

[27] P. N. F. Bt Mohd Shamsuddin and M. A. Bin Mansor, "Motion Control Algorithm for Path following and Trajectory Tracking for Unmanned Surface Vehicle: A Review Paper," *Proceedings - 2018 3rd International Conference on Control, Robotics and Cybernetics, CRC 2018*, pp. 73–77, 2018. DOI: 10.1109/CRC.2018.00023.

[28] Y. Singh, S. Sharma, R. Sutton, D. Hatton, and A. Khan, "A constrained A* approach towards optimal path planning for an unmanned surface vehicle in a maritime environment containing dynamic obstacles and ocean currents," *Ocean Engineering*, vol. 169, no. June, pp. 187–201, 2018, ISSN: 00298018. DOI: 10.1016/j.oceaneng.2018.09.016. [Online]. Available: https://doi.org/10.1016/j.oceaneng.2018.09.016.

[29] F. Adib Yaghmaie, A. Mobarhani, and H. D. Taghirad, "A new method for mobile robot navigation in dynamic environment: Escaping algorithm," *International Conference on Robotics and Mechatronics, ICRoM 2013*, no. February, pp. 212–217, 2013. DOI: 10.1109/ICRoM.2013.6510107.

[30] C. Fulgenzi, A. Spalanzani, and C. Laugier, "Dynamic obstacle avoidance in uncertain environment combining PVOs and occupancy grid," *Proceedings - IEEE International Conference on Robotics and Automation*, no. April, pp. 1610–1616, 2007, ISSN: 10504729. DOI: 10.1109/ROBOT.2007.363554.

[31] M. A. Kareem Jaradat, M. Al-Rousan, and L. Quadan, "Reinforcement based mobile robot navigation in dynamic environment," *Robotics and Computer-Integrated Manufacturing*, vol. 27, no. 1, pp. 135–149, 2011, ISSN: 07365845. DOI: 10.1016/j.rcim.2010.06.019. [Online]. Available: http://dx.doi.org/10.1016/j.rcim.2010.06.019.

[32] O. Andersson, O. Ljungqvist, M. Tiger, D. Axehill, and F. Heintz, "Receding-horizon lattice-based motion planning with dynamic obstacle avoidance," in *2018 IEEE Conference on Decision and Control (CDC)*, IEEE, 2018, pp. 4467–4474.

[33] A. Stentz, "Optimal and efficient path planning for unknown and dynamic environments," *International Journal of Robotics and Automation*, vol. 10, no. 3, pp. 89–100, 1995, ISSN: 08268185.

[34] E. Kelasidi, S. Moe, K. Y. Pettersen, A. M. Kohl, P. Liljebäck, and J. T. Gravdahl, "Path following, obstacle detection and obstacle avoidance for thrusted underwater snake robots," *Frontiers in Robotics and AI*, vol. 6, p. 57, 2019, ISSN: 2296-9144. DOI: 10.3389/frobt.2019.00057. [Online]. Available: https://www.frontiersin.org/article/10.3389/frobt.2019.00057.

[35] L. Jia, Z. Qian, X. Feng, and J. Lijun, "Obstacle detection method for underwater mine emergency rescue auv," in *2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST)*, IEEE, 2017, pp. 458–461.

[36] R. L. Smith, *Open dynamics engine*, 2001. [Online]. Available: https://www.ode.org/.

[37] A. Raj and A. Thakur, "Hydrodynamic parameter estimation for an anguilliform-inspired robot," *Journal of Intelligent & Robotic Systems*, vol. 99, no. 3, pp. 837–857, 2020.