Collision Avoidance for an Autonomous Surface Vehicles using Deep Reinforcement Learning

Ivana Collado González
Tecnologico de Monterrey
School of Science and Engineering
Av. Eugenio Garza Sada Sur 2501
Monterrey, Mexico
a00569475@itesm.mx
Intelligent Systems Project I - IMT

Victor Sebastian Martinez Perez
Tecnologico de Monterrey
School of Science and Engineering
Av. Eugenio Garza Sada Sur 2501
Monterrey, Mexico
a01232474@itesm.mx
Intelligent Systems Project I - ISD

Leonardo Garrido
Tecnologico de Monterrey
School of Science and Engineering
Av. Eugenio Garza Sada Sur 2501
Monterrey, Mexico
leonardo.garrido@tec.mx
Professor

***Abstract*—Fully-autonomous Unmanned Surface Vehicles (USVs) must be capable of dealing with high levels of uncertainty, while completing complex maritime tasks. In order to cope with the increasing demand in maneuverable flexibility in unknown environments, robust collision avoidance methodologies are required. This project presents a deep reinforcement learning neural network that acts as a collision avoidance guidance law, while using a low-lever controller. The low-lever adaptive sliding mode controller makes this DRL collision avoidance strategy more robust in comparison to other collision avoidance DRL strategies.**

***Index Terms*—collision avoidance, USV, velocity obstacle, deep reinforcement learning, deep deterministic policy gradient**

I. Introduction

More than 75% of incidents in the shipping industry are caused by human error [2]. Unmanned Surface Vehicles (USVs) can substitute human operations that are time consuming or dangerous, such as environmental monitoring or mine searching [3]. Development of reliable collision avoidance is essential to the creating of fully-autonomous USVs, immune to human error, reducing human accidents.

Collision avoidance methods can be divided into global, local and hybrid approaches. Global methods include graph search methods like A* [12] and Voronoi Graphs, and randomizing methods like rapidly-exploring random trees [5] and probabilistic road maps [6]. Global methods require a complete global map of the environment to be processed in order to find an optimal trajectory to a goal in advance. Moreover, an optimal trajectory must be recalculated in case of environment uncertainty and disturbances, making them computationally expensive and infeasible for real-world applications with limited processing power [8]. Local methods, also known as reactive methods, like, artificial potential fields [14], dynamic window approaches, Velocity Obstacle (VO) methods [7] and control-based methods, make their immediate guidance decisions only on environmental sensing [8]. Local approaches are therefore computationally less expensive than global ones, which in turn makes them better suited for real-time applications and applications where the global environment is unknown. However, they are susceptible to local minimum or "local death ends". Hybrid approaches, as the name implies, are combinations between global and local methods . First a global trajectory is created and then local approaches are applied in case of environment disturbances.

Reinforcement Learning (RL) has been recently applied successfully to solve USV control and guidance problems. A Deep Reinforcement Learning Neural Network (DRL NN) used in [8] was trained end to end to perform path following control of an autonomous surface vehicle while performing collision avoidance. A deep reinforcement learning (DRL) technique was presented in [9] to act as a path following guidance law for an autonomous surface vehicle using an adaptive sliding mode controller.

This project proposes a novel DRL NN to act as a guidance law for collision avoidance, similar to [8], while using a low-level adaptive sliding mode controller for path following, similar to [9]. This work only deals with static, round-shaped obstacles.

Section II presents a simplified version of the original Velocity Obstacle method [7] to perform collision avoidance of static round-shaped obstacles. Section III presents the DRL problem definition. Section IV then presents the mathematical model of the USV used in this project. Afterwards, the Methodology section resents the specific algorithms used for the DRL environment, followed by the specific implementation parameters used in the MDP and training of the NN. Finally, preliminary results of the project are resented in section VII, followed by conclusions.

II. Velocity Obstacle

Consider an USV (*A*) with velocity V_A, and a static obstacle (*B*), with known position. In order to compute the *Velocity Obstacle* (VO), it is necessary to map the *Configuration Space* of *B* (the space of all the possible positions of *B*) to *A* by reducing the latter to a point and by enlarging *B* by the radius of *A*. A *Collision Cone* ($CC_{A,B}$) is defined as the set of relative velocities between *A* and *B* that will eventually result in a collision:

$$CC_{A,B} = \{V_A | \lambda_A \cap B \neq 0\} \tag{1}$$

where λ_A is the line of V_A.

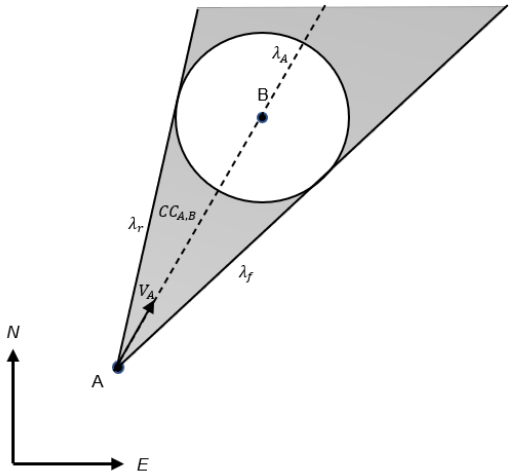


Fig. 1: V_A and the Collision Cone $CC_{A,B}$

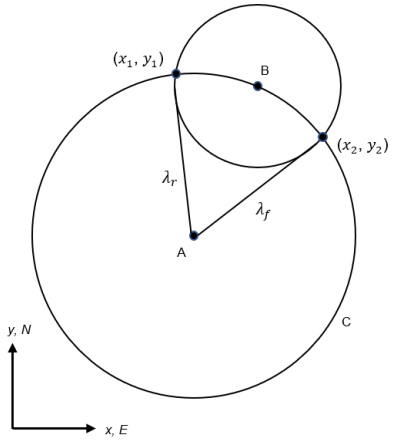


Fig. 2: Intersections of C and B

The $CC_{A,B}$ (Fig. 1) is a sector with apex in *A* and bounded by two tangent lines λ_f and λ_r that go from *A* to *B*.

In order to define the points where λ_f and λ_r are tangent to *B*, a circle *C* is defined with center in *A* and with a radius *D*, which is the distance from *A* to the center of *B*. Considering NED as the global reference frame, the equations that define circles *C* and *B* are:

$$r_B{}^2 = (x-a)^2 + (y-b)^2 \tag{2}$$

$$r_C{}^2 = (x-c)^2 + (y-d)^2 \tag{3}$$

$$D = \sqrt{(c-a)^2 + (d-b)^2}, \quad \text{and } r_B + r_C > D > |r_B - r_C| \tag{4}$$

where (2) represents *B* with origin at (a, b) and (3) is *C* with origin at (c, d), as shown in figure 2. The intersection of *C* and *B* is described by the following equations:

$$x_{1,2} = \frac{a+c}{2} + \frac{(c-a)(r_B{}^2 - r_C{}^2)}{2D^2} \pm 2\frac{b-d}{D^2}\partial \tag{5}$$

$$y_{1,2} = \frac{b+d}{2} + \frac{(d-b)(r_B{}^2 - r_C{}^2)}{2D^2} \mp 2\frac{a-c}{D^2}\partial \tag{6}$$

$$\partial = \frac{1}{4}\sqrt{(D+r_B+r_C)(D+r_B-r_C)} * \sqrt{(D-r_B+r_C)(-D+r_B+r_C)} \tag{7}$$

with points (x_1, y_1) and (x_2, y_2), the $CC_{A,B}$ is fully defined.

To avoid multiple obstacles, we consider the union of multiple Velocity Obstacles:

$$VO = \cup_{i=1}^{m} VO_{B_i} \tag{8}$$

where *m* is the number of obstacles. The avoidance velocities, then, consist of those that lie outside of the multiple *VO*'s.

When considering the case of many obstacles, it is useful to prioritize those obstacles that present an *imminent collision*. This is done by considering those collisions that occur at some time $t < T_h$, where T_h is a suitable time horizon selected based on system dynamics, obstacle trajectories and computation time.

$$VO_h = \{V_A | V_A \in VO, ||V_A|| \leq \frac{d_m}{T_h}\} \tag{9}$$

Considering d_m as the shortest relative distance between the boat and the obstacle, the set VO_h is built. This set represents the velocities that would result in a collision, occurring beyond the time horizon. To account for *imminent collisions* the set VO_h is subtracted from VO.

A. Avoidance maneuver

Any velocity outside of *VO* would avoid collisions. However, the USV presents actuator constraints such as maximum linear acceleration a_{long} and maximum angular acceleration a_{rot} that must be considered. With this information, the admissible longitudinal and rotational accelerations must be set and integrated over a certain time step to obtain a set of *Reachable Velocities* (RV).

$$RV = \{V \in [V_A - a_{long} * \Delta t, V_A + a_{long} * \Delta t], W \in [V_A - a_{rot} * \Delta t, V_A + a_{rot} * \Delta t]\} \tag{10}$$

By subtracting VO from RV a set of *Reachable Avoidance Velocities* (RAV) can be obtained. With this, a maneuver to avoid *B* can be performed by selecting any velocity within RAV.

At most two RAV subsections are created from each VO, denominated the right subsection S_r and the left subsection S_l. Figure 3 shows the set of *Reachable Avoidance Velocities* and the *Velocity Obstacle* for a single obstacle.

B. Computing of avoidance trajectories

Two categories for computing trajectories for avoidance of static and dynamic obstacles are proposed in [7]. A trajectory consists of a sequence of avoidance maneuvers, selected by searching over a tree of feasible maneuvers computed at discrete time intervals. A global search is proposed for off-line applications, and a heuristic search for on-line applications. In

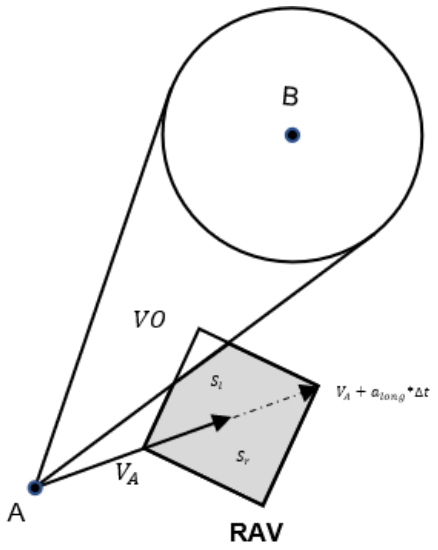


Fig. 3: *RAV*, *VO* and left and right sections

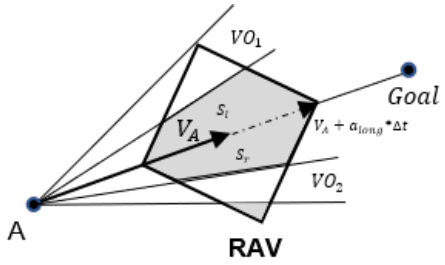


Fig. 4: Vertex closest to goal considering multiple obstacles

this project, a simplified heuristic approach is used to compute the desired velocity at each time step.

1) Simplified Heuristic: After successfully computing the RAV subsections, each geometric vertex from the S_l and S_r is saved and processed. Afterwards, the vertex that has the minimum euclidean distance to the goal is chosen, as shown in figure 4.

III. Deep Reinforcement Learning

A. Reinforcement Learning

Reinforcement learning is a branch of ML that focuses on goal-directed learning from interaction between an agent and its environment [13]. It uses the formal framework of Markov Decision Processes (MDP) to define the interaction between a learning agent and its environment in terms of a state space S, an action space A, an initial state distribution $p(s1)$, a transition model $p(s_{t+1}|s_t, a_t)$, and a reward function $r(s_t, a_t)$ [13]. At each time step t, the agent receives an observation s_t, selects an action at based on that and obtains a reward r_{t+1} a time step later, finding itself in a new state s_{t+1}. A policy π describes the agent's behavior, by selecting actions based on previous states. The agent's objective is to maximize the amount of reward it receives over time by finding an optimal policy π^*.

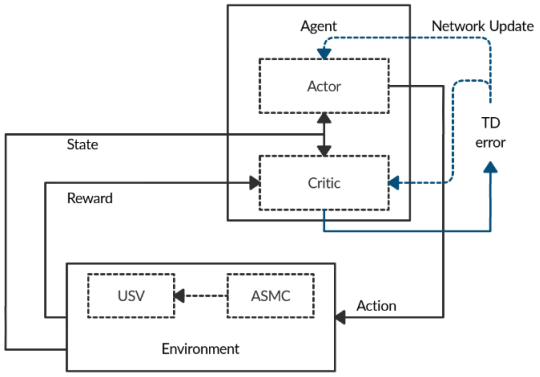


Fig. 5: DDPG architecture

In a MDP, the probabilities given by p completely characterize the environment's dynamics [13]. That is, the probability of each possible value for s_t and r_t depends only on the immediately preceding state and action, S_{t-1} and A_{t-1}, and, given them, not at all on earlier states and actions. This is best viewed a restriction not on the decision process, but on the state. The state must include information about all aspects of the past agent–environment interaction that make a difference for the future.

Action-value functions are widely used in reinforcement learning algorithms [10]. These functions express the expected discounted reward after taking an action a_t in state s_t, and follows the policy π, as:

$$q_\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \quad (11)$$

The combination of RL with DL establishes the field of DRL [4]. One class of DRL algorithm is the actor-critic algorithm, which approximates both the policy and action-value functions with a function approximator such as a deep neural network (DNN).

B. Deep Deterministic Policy Gradient

The DDPG algorithm is an actor-critic DRL method developed by [10], and its capability of "robustly solving challenging problems across a variety of domains with continuous action space" [10] makes it suitable for the guidance problem. The policy function $\pi(s|\theta_a)$ and the action-value function $Q(s, a|\theta_c)$ are DNNs, where θ_a and θ_c are the DNN parameters. To update both functions, stochastic gradient descent is executed on batches $\mathcal{B}$ of transitions (s_i, a_i, r_i, s_{i+1}) following the rules proposed in [9]:

$$\theta_c \leftarrow \theta_c - \alpha_c \frac{1}{N} \sum_{i \in \mathcal{B}} \nabla_{\theta_c} (y_i - Q(s_i, a_i|\theta_c))^2 \quad (12)$$

$$\theta_a \leftarrow \theta_a - \alpha_a \frac{1}{N} \sum_{i \in \mathcal{B}} \nabla_{a_i} Q(s_i, a_i|\theta_c) \nabla_{\theta_a} \pi(s_i|\theta_a) \quad (13)$$

where $\alpha_{c'}$ and $\alpha_{a'}$ are learning rates, and y_i is the action-value estimate, or Temporal Difference (TD) target, obtained from:

$$y_i = r_i + \gamma Q'(s_{i+1}, \pi(s_{i+1}|\theta_a)|\theta_{c'}) \tag{14}$$

Here, θ_c and θ_a are parameters from two target networks. The target networks are introduced in DDPG to stabilize training [10], by slowly learning the parameters using the following equations:

$$\theta_{c'} = (1-\tau)\theta_{c'} + \tau\theta_c \tag{15}$$

$$\theta_{a'} = (1-\tau)\theta_{a'} + \tau\theta_a \tag{16}$$

where τ is the target network update rate. Fig. 5 depicts an overview of the DDPG architecture and the MDP as proposed in [9], whereas Algorithm 1 describes the DDPG algorithm [10].

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta_c)$ and actor network $\pi(s|\theta_a)$ with weights θ_c and θ_a
Initialize target networks $Q^{'}$ and π with weights $\theta_{c'} \leftarrow \theta_c$ and $\theta_{a'} \leftarrow \theta_a$
Initialize replay buffer R
for $episode = 1, M$ **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for $t = 1, T$ **do**
    Select action $a_t = \pi(s_t|\theta_a) + \mathcal{N}_t$ according to the current policy and exploration noise
    Take action a_t, observe reward r_t and observe new state s_{t+1}
    Save transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions from R
    Set y_i with Equation (14)
    Update the critic by minimizing the loss using Equation (12)
    Update the actor policy using Equation (13)
    Update the target networks using Equations (15) and (16)
  end for
end for

IV. Mathematical Model

The mathematical model of the USV platform used in this work was developed following the methodology proposed by Fossen [16]. The USV is an underactuated vehicle with a differential thruster configuration. Fig. 6 depicts the north-east-down (NED) and body-fixed reference frames, as well as the forces created by the port and starboard thrusters of the boat. The overall equation of motion is equation (17), and equation (18) expresses the kinematics of the USV model. $\boldsymbol{\eta} = [x, y, \psi]^T$ denotes the xy coordinates and angle about z, and $\boldsymbol{\upsilon} = [u, v, r]^T$ corresponds to the surge and sway velocities, and the yaw rate.

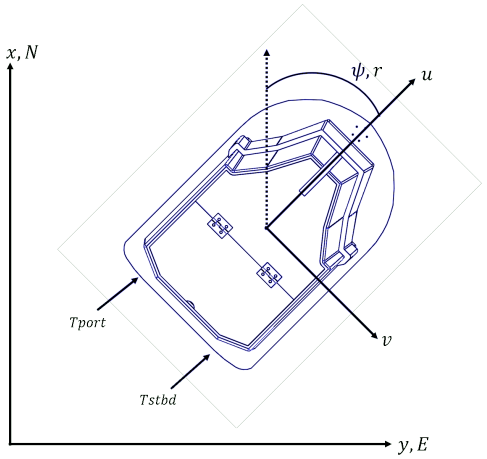


Fig. 6: NED and body-fixed reference frames.

$$\boldsymbol{\tau} = \boldsymbol{M}\dot{\boldsymbol{\upsilon}} + \boldsymbol{C}(\boldsymbol{\upsilon})\boldsymbol{\upsilon} + \boldsymbol{D}(\boldsymbol{\upsilon})\boldsymbol{\upsilon} \tag{17}$$

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\upsilon} \tag{18}$$

Equation (19) is a rotation matrix of velocity vectors in body-fixed and NED reference frames.

$$\boldsymbol{J}(\boldsymbol{\eta}) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{19}$$

$\boldsymbol{M}$ is a matrix sum of a rigid body mass matrix and an added mass matrix defined by:

$$\boldsymbol{M} = \begin{bmatrix} m - X_{\dot{u}} & 0 & -my_G \\ 0 & m - Y_{\dot{v}} & mx_G - Y_{\dot{r}} \\ -my_G & mx_G - N_{\dot{v}} & I_z - N_{\dot{r}}, \end{bmatrix} \tag{20}$$

$\boldsymbol{C}(\boldsymbol{\upsilon})$ is a Coriolis matrix which includes the sum of a rigid body matrix and an added mass matrix as the $\boldsymbol{M}$ matrix. It is represented by:

$$\boldsymbol{C}(\boldsymbol{\upsilon}) = \begin{bmatrix} 0 & 0 & -m(x_G r + v) \\ 0 & 0 & -m(y_G r - u) \\ m(x_G r + v) & m(y_G r - u) & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 2(Y_{\dot{v}} v + (\frac{Y_{\dot{r}} + N_{\dot{v}}}{2})r) \\ 0 & 0 & -X_{\dot{u}} u \\ 2(-Y_{\dot{v}} v - (\frac{Y_{\dot{r}} + N_{\dot{v}}}{2})r) & X_{\dot{u}} u & 0 \end{bmatrix} \tag{21}$$

In Equation (21), a factor of 2 was implemented to fit the twin-hull configuration of the USV [15]. Moreover, Equation (22) is the addition of linear and nonlinear drag matrices:

TABLE I: VTec S-III Physical Parameters

Parameter	Value
Length overall	1.01 [m]
Draft	0.09 [m]
Beam overall	0.75 [m]
Centerline-to-centerline side hull separation	0.41 [m]
Individual hull beam	0.27 [m]
Mass	30 [kg]
Longitudinal center of gravity	0.45 [m]
Moment of inertia	4.1 $[kgm^2]$

TABLE II: Hydrodynamic Coefficients

Coefficient	Non-dimensional Factor	Dimensional Term
X_u		25, $(u > 1.2)$ 64.55
Y_v	0.5	$f(Y, v)$
Y_r	3	$-\pi\rho\sqrt{(u^2+v^2)}T^2L$
N_v	0.06	$-\pi\rho\sqrt{(u^2+v^2)}T^2L$
N_r	0.02	$-\pi\rho\sqrt{(u^2+v^2)}T^2L^2$
$X_{\dot{u}}$		-2.25
$Y_{\dot{v}}$		-23.13
$Y_{\dot{r}}$		-1.31
$N_{\dot{v}}$		-16.41
$N_{\dot{r}}$		-2.79
$X_{u\|u\|}$		0, $(u > 1.2)$ -70.92
$Y_{v\|v\|}$		-99.99
$Y_{v\|r\|}$		-5.49
$Y_{r\|v\|}$		-5.49
$Y_{r\|r\|}$		-8.8
$N_{v\|v\|}$		-5.49
$N_{v\|r\|}$		-8.8
$N_{r\|v\|}$		-8.8
$N_{r\|r\|}$		-3.49

$$f(Y,v) = -40\rho|v|\left[1.1 + 0.0045\frac{L}{T} - 0.1\frac{B_{hull}}{T} + 0.016\left(\frac{B_{hull}}{T}\right)^2\right]\left(\frac{\pi TL}{2}\right)$$

$$\boldsymbol{D}(\boldsymbol{v}) = -\begin{bmatrix} X_u & 0 & 0 \\ 0 & Y_v & Y_r \\ 0 & N_v & N_r \end{bmatrix} - \begin{bmatrix} X_{u|u|}|u| & 0 & 0 \\ 0 & Y_{v|v|}|v| + Y_{v|r|}|r| & Y_{r|v|}|v| + Y_{r|r|}|r| \\ 0 & N_{v|v|}|v| + N_{v|r|}|r| & N_{r|v|}|v| + N_{r|r|}|r| \end{bmatrix} \tag{22}$$

Equation (23) is a vector of forces and moments created by the thrusters:

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} T_{port} + 0.78T_{stbd} \\ 0 \\ (T_{port} - 0.78T_{stbd})B/2 \end{bmatrix} \tag{23}$$

Finally, Table I [15] contains the physical parameters of the USV, and Table II contains the constant and variable hydrodynamic coefficients and equations, as shown in [15]. Further explanation of the model parameters can be found in [17].

TABLE III: Sensor configuration

Parameter	Description	Value
N	Number of sensor readings	225
d	Number of sectors	25
n	Number sensor readings per sector	N/d
S_s	Total visual sensor span	$\frac{4}{3}\pi$ [rad]
S_r	Maximum rangefinder distance	100 [m]
L_r	LiDAR resolution	1.066°
W_b	Boat radius	0.5 [m]
W_s	Safety radius	0.3 [m]
S_d	Safety distance	0.1 [m]

V. Methodology

In this section the DRL environment setup and MDP definition is specified in accordance to the problem at hand. The simulated sensor for obstacle detection and the RL NN training process are also explained.

A. Environment

The environment the agent interacts with in this project is a rectangular area 20 x 40 m^2 filled with round static obstacles. The number of obstacles s was randomly set to be between 0 and 20. The position of the obstacles was randomly calculated in y and normally distributed in x within the NED reference frame. The sizes of the obstacles was also randomly set with radius ranging from 0.1 meters to 1.5 meters. The boat is set to start with a random position and orientation inside a rectangular surface of 10 x 5 m^2 in the lower region of the environment surface.

The path is defined by two points. The initial point (x_0, y_0), is set in the same region the boat starts in, and a final point (x_d, y_d) in the superior region of the environment surface. The path is defined to be completely vertical, so $y_d = y_0$. Then, the angle $a_k = atan2(y_d - y_0, x_d - x_0)$ of the path with respect to NED is calculated.

The desired velocity is a random number between 0.3 and 1.4 m/s^2, just above and below of the minimum and maximum velocity the boat can achieve, this is done so the boat can move and the actuators are not extensively stressed.

B. Simulating a 2D LiDAR sensor

For this project, a simulation of a 2D LiDAR sensor was created. Table III contains the sensor configuration used for the simulation. From it, the sensor span S_s, was chosen to be similar to the one proposed in [8]. S_r was established to correspond to the max range of the VLP-16 Velodyne LiDAR. Also, the LiDAR resolution L_r was set to 1.066 degrees, though its real resolution is 0.3 degrees, it was changed to reduce computation costs.

Algorithm 2 demonstrates how the distance values for each laser was obtained. This algorithm calculates the intersection of the laser line of sight with the closest obstacle.

C. Obstacle detection

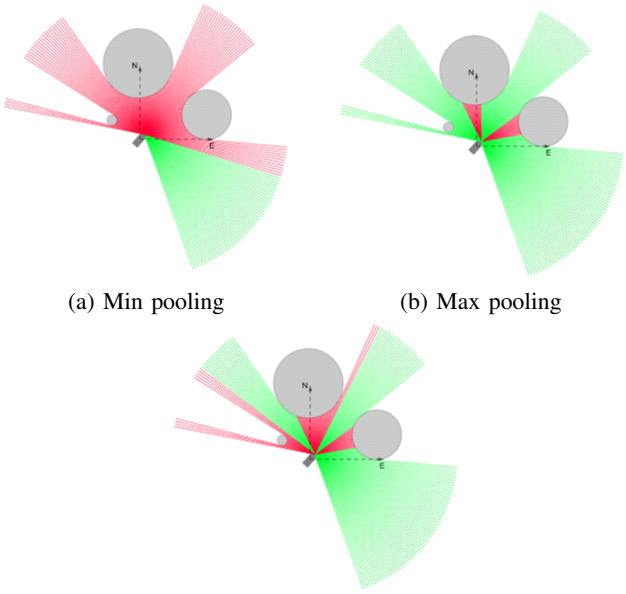
Obstacle avoidance methods require the use of a sensor suite to identify possible obstacles in the environment. Including

Algorithm 2 Laser scan simulation

o number of obstacles
O_o obstacle array ordered from closest to farthest from the boat
N number sensor readings
S_s sensor span
S_d sensor measured distance
s_α sensor angle
L_r LiDAR resolution
x=$\{x_1, ..., x_N\}$ Sensor rangefinder measurements
for i in N **do**
  $S_d = S_r$
  $s_\alpha = -S_s/2 + L_r \cdot i$
  $s_\alpha = s_\alpha$ with respect to NED $\in [-\pi, \pi]$
  for j in o **do**
    Calculate coordinates (obs_x, obs_y) of $O_o i$ with respect to the sensor reference frame $obs_r = O_o i$ radius
    if $obs_x >= 0$ **then**
      $g = obs_r^2 - osb_y^2$
      if $g >= 0$ **then**
        Obtain intersections:
        $x_1 = obs_x + \sqrt{g}$
        $x_2 = obs_x - \sqrt{g}$
        $y_1 = y_2 = 0$
        Obtain distances d_1 and d_2 to the intersections
        if $d_1 > d_2$ **then**
          $x_i = d_2$
        else
          $x_i = d_1$
        end for
      end if
    end if
  else
    $x_i = S_r$
  end if
end for

all the sensor readings in the training of a neural network is not a viable approach, given the complexity required for the satisfactory mapping between the full sensor suite to the steering signal. To cope with this issue, three approaches for transforming the sensor readings into a reduced observation space was proposed [8]. This involves partitioning the sensor suite into d disjoint sensor sets, hereafter referred to as *sectors*.

Within each sector, a single scalar that represents the local sensor readings is meant to be found. Now, instead of having N sensor measurements, only d features are preserved. Always returning the minimum sensor reading within the sector is known as *min pooling* and returning the maximum reading *max pooling*, each of them presenting certain constraints as the former could overlook feasible passing between obstacles and the latter ignore small nearby objects [8] as shown in Fig. 7. In order to cope with this problem, a *feasible pooling* approach was developed in [8], in which the maximum feasible travel distance within a sector is computed. The feasibility pooling



(a) Min pooling (b) Max pooling

(c) Feasible pooling

Fig. 7: Pooling techniques for sensor dimensional reduction

algorithm is explained in detail in Algorithm 3.

D. Collision detection

In order to validate a collision, a series of simple steps can be followed. First, it is necessary to compute the distance to each obstacle. Then, subtract from the distance the radius of the obstacle and the radius of the boat, and an extra 'safety' radius. Finally, if the resulting distance is less than a user defined safety distance, a collision occurs.

VI. Implementation

A. Markov Decision Process

The Markov Decision Process for this specific application is defined by the state 25, action 24 and reward 33.

$$a = [u_d, e_\psi] \tag{24}$$

$$S = \{u, v, r, y_e, \dot{y_e}, \chi_{\alpha k}, u_{ref}, sectors, a[0], a[1]\} \tag{25}$$

The state is conformed by u the surge speed, v the sway speed, r the yaw rate, y_e the cross-track error, $\dot{y_e}$ the change in the cross-track error, $\chi_{\alpha k}$ the angle between the boat and the path considering slip, u_{ref} the reference surge, $sectors$ are the normalized sector readings, and the actions u_d and e_ψ, representing the desired surge and error on the boat angle respectively.

The reward 33 is a composed equation, as it considers both path following 31 and obstacle avoidance 32 rewards, given a certain weight λ. The path following reward 31, includes the cross track error reward 26, the speed reward 27, the angle regard 28, and the change in velocity and angle rewards 29 and 30.

$$C_{a_0} = \frac{1}{(\frac{a[0]max - a[0]min}{\Delta t})^2}$$

Algorithm 3 Feasibility Pooling algorithm

d number of sectors
Assign S_r to all $sectors$
n number of lasers in a sector
a_l arc length
L_r LiDAR resolution
O_w opening width
O_f opening found
W_b boat radius
W_s safety radius
for s in d **do**
  x = $\{x_1, ..., x_n\}$ Sensor rangefinder measurements for each individual sector
  $x_o = x$ sorted in ascending order.
  for $i \in x_o$ **do**
    $a_l = l_r * x_{o_i}$
    $O_w = a_l/2$
    $O_f = false$
    for $j \in x_o$ **do**
      if $x_j > x_{o_i}$ **then**
        $O_w = O_w + a_l$
        if $O_w > (2 * (W_b + W_s))$ **then**
          $O_f = true$
          break
        end if
      else
        $O_w = O_w + \frac{a_l}{2}$
        if $O_w > (W_b + W_s)$ **then**
          $O_f = true$
          break
        end if
        $O_w = 0$
      end if
    end for
    if O_f is false **then**
      $sectors_i = x_{o_i}$
    end if
  end for
end for
Normalize sector readings

$$C_{a_1} = \frac{1}{(\frac{a[1]max - a[1]min}{\Delta t})^2}$$

$$\chi_{\alpha k} = -\chi_{\alpha k}$$

$$r_{y_e} = max \begin{cases} e^{-k_{ye}|y_e|} \\ e^{-k_{ye}(y_e)^2} \end{cases} \quad (26)$$

$$r_u = e^{-k_{u_u}(u_{ref} - \sqrt{u^2+v^2})^2} \quad (27)$$

$$r_\chi = cos(\chi_{\alpha k}) \quad (28)$$

$$r_{a_0} = \tanh -C_{a_0}\dot{a}[0]^2 \quad (29)$$

$$r_{a_1} = \tanh -C_{a_1}\dot{a}[1]^2 \quad (30)$$

$$r_{pf} = w_{y_e} \cdot r_{y_e} + w_\chi \cdot r_\chi + w_u \cdot r_u + w_{a_0} \cdot r_{a_0} + w_{a_1} \cdot r_{a_1} \quad (31)$$

TABLE IV: Reward parameters

Parameter	Value
w_{y_e}	0.4
w_χ	0.4
w_u	0.2
w_{a_0}	0.2
w_{a_1}	0.2
λ	0.9
γ_θ	4.0
γ_χ	1.0
k_y	0.5
k_{u_u}	-15.0

$$r_{oa} = -\frac{\sum_{i=1}^{N}(1 + |\gamma_\theta \theta_i|)^{-1}(\gamma_x max(x_i, \epsilon_x)^2)^{-1}}{\sum_{i=1}^{N}(1 + |\gamma_\theta \theta_i|)^{-1}} \quad (32)$$

where $\epsilon_x > 0$ is a small constant removing the singularity at $x_i = 0$.

$$R = \begin{cases} -1000 & \text{if there is a collision} \\ \lambda \cdot r_{pf} + (1 - \lambda) \cdot r_{oa} & \text{otherwise} \end{cases} \quad (33)$$

B. Neural Network Architecture and Training

The DNN architecture and hyper-parameters were selected to be similar to those expressed in [9]. The actor network inputs the state, has two hidden layers of 400 and 300 neurons respectively, and outputs the action. The activation functions are Rectified Linear Units (ReLU) for the hidden layers, a Sigmoid activation function for velocity output, and a hyperbolic tangent activation function for the angle output. The critic network inputs the state, then has a hidden layer of 400 neurons, next adds a hidden layer of 300 neurons which also inputs the action taken, then has a third hidden layer, composed of 300 neurons, and outputs the action-value. All of the hidden layers have ReLU activation functions, except the output, which has a Linear activation function for the output.

To train the DNNs, the gradient descent-based Adam optimizer was used to learn the DNN parameters, all the parameters of the optimizer where chosen in accordance with [9]. The training data consisted of simulating different straight-line paths, while starting the USV in random positions and orientations, and v_o= [0,0,0]. Each episode consisted of 600 training steps, with a time step of 0.05 seconds, which simulates 20 seconds running the guidance law at 20 Hz. However, the ASMC was running in the simulation at a higher frequency of 100 Hz, which is used in practical implementation. Thus, for every training step, the ASMC runs 5 times before the next observation. This decision was made because, in practice, several guidance systems need to run at lower frequencies, particularly when perception systems, such as computer vision with cameras or LiDAR, are involved. Finally, training was stopped after 1500 episodes.

VII. Preliminary Results

In order to successfully program the VO method, programming 2D Boolean set operation was necessary in order to obtain the RV and RAV sets. For these special operations,

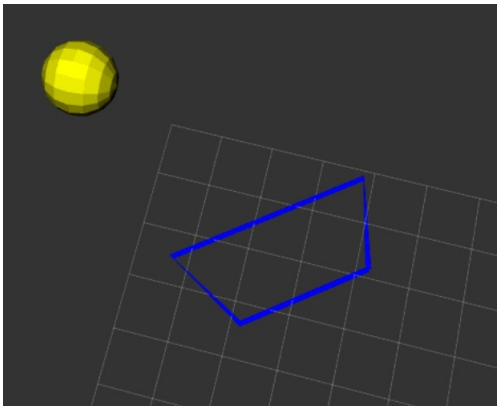


Fig. 8: Velocity obstacle visualisation in RVIZ.

the C++ the Computational Geometry Algorithms Library (CGAL) [11] was used. A graphic interface using ROS and RVIZ was created for easy visualization and program debugging, as seen in Fig. 8.

The code implementation of the DRL presented method makes use of the RL framework provided by the Python toolkit **OpenAi Gym** [1], which was created as a mean to standardize environment creation, so that research on AI becomes easily reproducible. Keras is a deep learning API also written in Python. It was designed to be user friendly, so DL models can be easily designed and fast to produce, its use is further simplified using TensoFlow notation. The visualization of the environment is presented in Fig. 9.

Preliminary results of the DRL NN demonstrates that at this stage the USV is able to avoid collision 53 out of 100 episodes, which indicates an efficiency of 53%. A few visualization trials of USV behavior in simulation can be found by clicking this link. It is evident that the USV avoids obstacles while following a path (light blue), but still presents unstable behaviour.

VIII. Conclusion

In the first stage of the project, the simplified Velocity Obstacle [7] method was chosen and implemented in order to obtain deep understanding of collision avoidance basic technical aspects. This technique can potentially be used as a comparison strategy for the final DRL collision avoidance method.

In this second stage of the project, Reinforcement Learning was defined, along with DDPG, the DRL method chosen for this project. USV DRL collision avoidance methods where researched and compared in order to chose a simple and effective obstacle input strategy, from which feasibility pooling was chosen for its effective representation of the environment and low computation cost.

In the last stage of this project, the strategies for creating the DRL NN environment and training strategy are presented, along with state vector and reward function definitions. Simple preliminary testing of the DRL NN is also be presented, showing promising but still unreliable behaviour from the

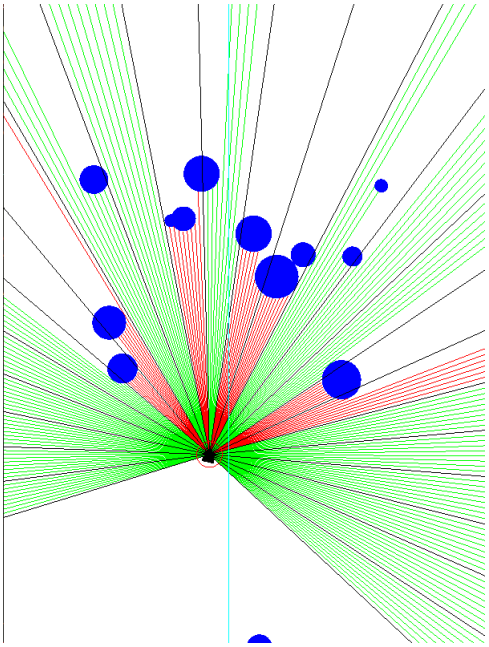


Fig. 9: OpenAI Gym visualization of environment for simulation and training.

USV. Training of the NN was interrupted and renewed a few times because of updates and improvements necessary to NN architecture, state and rewards, in order to make the USV behaviour more efficient. The DRL NN still requires more training and overall reward tuning.

This project will continue to be improved and modified until a stable path following and avoidance behaviour is reached. A potential training strategy being considered to improve current results is to use transfer learning to first show the NN how to follow a path and afterward to avoid obstacles. Another improvement strategy is to remove the velocity control from the problem formulation, this way the USV can focus on avoidance and path following, controlling only angle.

Acknowledgment

This work was supported by the student group VantTec, as well as their sponsors: Hacsys, VectorNav, Google, ifm efector, Uber ATG, RoboNation, Velodyne Lidar, NVIDIA, Skysset, Akky, Greenzie, and Tecnologico de Monterrey

References

[1] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J.,Tang, J., Zaremba, W., 2016. Openai gym.arXiv:arXiv:1606.0154

[2] D. Campos, et al. "An Adaptive Velocity Avoidance Algorithm for Autonomous Surface Vehicles", IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),2019.

[3] J. Woo and N. Kim. "Collision avoidance for an unmanned surface vehicle using deep reinforcement learning", Ocean Engineering, 2020.

[4] K. Arulkumaran et al., "Deep Reinforcement Learning: A Brief Survey,"IEEE Signal Processing Magazine, 2017.

[5] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.[19]

[6] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," IEEE Trans. Robot. Autom., vol. 12, no. 4, pp. 566–580,Sep. 1996.

[7] P, Fiorini and Z. Shiller, "Motion Planning in Dynamic Environments Using Velocity Obstacles", The International Journal of Robotics Research, 1998.
[8] E. Meyer, H, Robinson, A. Rasheed, and O. San, ¨Taming an Autonomous Surface Vehicle for Path Following and Collision Avoidance Using Deep Reinforcement Learning¨, IEEE Access, 2020.
[9] A. Gonzalez,-Garía, H. Castañeda, and L. Garrido, ¨USV Path-Following Control Based On Deep Reinforcement Learning and Adaptive Control¨, unpublished.
[10] T.P. Lillicrap et al., "Continuous control with deep reinforcement learning," https://arxiv.org/pdf/1509.02971.pdf, 2016.
[11] E. Fogel, et al. "2D Regularized Boolean Set-Operations. In CGAL User and Reference Manual", CGAL Editorial Board, 5.1 edition, 2020.
[12] P. Hart, N. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum CostPaths",IEEE Transactions on Systems Science and Cybernetics,4, 100–107, 1968.
[13] R.S. Sutton and A. Barto. "Reinforcement Learning: An Introduction", MIT Press, 2018.
[14] O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots",IEEE International Conference on Robotics and Automation, 2, 500–505, 1985.
[15] A. Gonzalez,-Garía, H. Castañeda, and L. Garrido, ¨USV Path-Following Control Based On Deep Reinforcement Learning and Adaptive Control¨, unpublished.
[16] Fossen, T.I. 2011. *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd.
[17] Gonzalez-Garcia, A. and Castañeda, H. 2019. Modeling, Identification and Control of an Unmanned Surface Vehicle. In *AUVSI XPONENTIAL 2019: All Things Unmanned*.

APPENDIX A: GANTT

Figure 10 shows the Gantt chart for the project. The Gantt diagram can also be found here.

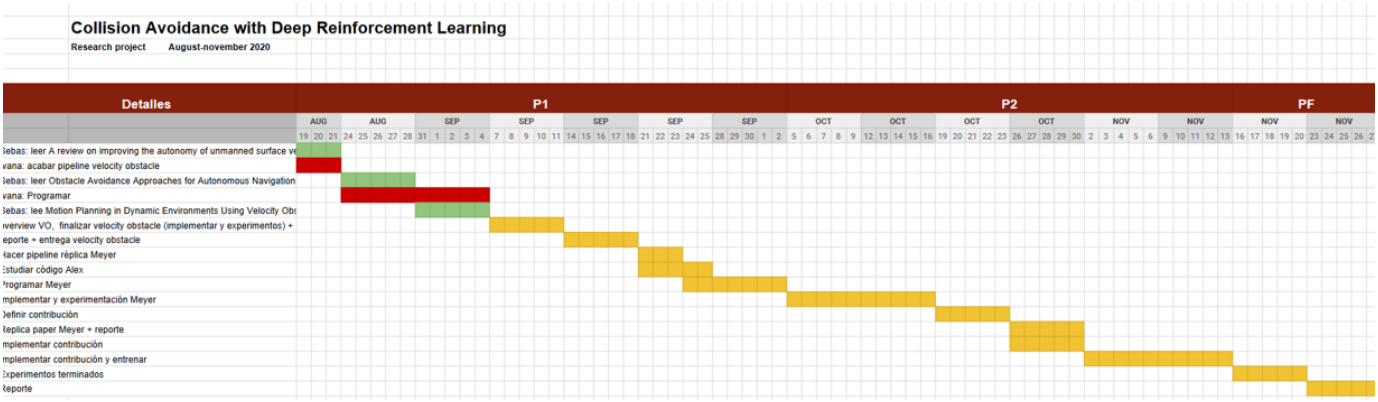


Fig. 10: Gantt for the project