Reward engineering applied to a optimization of Univector Field Parameters Using CMA-ES

José Ronaldo S. H. F.[1] and Nean Segura[2]

***Abstract*— An effective path planning algorithm for the IEEE Very Small Size Soccer league, a mobile robotics competition featuring cubic differential robots playing autonomous soccer matches, is the univector field navigation. In this paper, we propose utilizing the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to optimize the univector parameters involved in implementing this path planning method. Our research is centered around the theme of Reward Engineering applied to the optimization of Univector Field Parameters Using CMA-ES.**

I. Introduction

The Very Small Size Soccer League, VSS for short, is a mobile robotics competition featuring cubic differential robots playing autonomous soccer matches. Each robot has a shape of a cube of edges of size up to 7.5 cm, and each team has 3 robots. In the match, the same way as football, the main goal is to get the ball into the opponent's goal, while don't letting the opponent bring the ball to your own. The VSS league can be used as a great way to teach all sorts of engineering undergraduate students, since it posses challenges from the low level mechanical and electrical basics up to the high level of artificial intelligence and multi-agent system. Although the project is not made only by the AI, it's definitely its core, since it is the brain that commands the whole system.

In the ITAndroids team it is used the univector field navigation as a path planning algorithms in the agent. The parameters of this algorithms will be optimized using a CMA-ES algorithm and using reward engineering for modelling our reward function that maximizes the optimization rate of the CMA-ES.

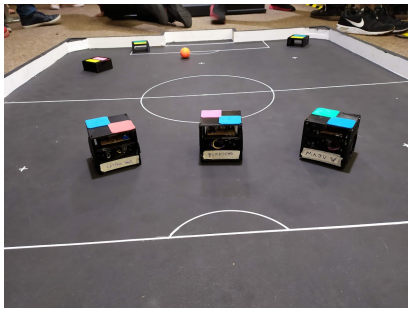

Fig. 1. Two VSS teams facing each other

[1]Aeronautical Institute of Technology, Electronic Engineering Graduate jrs.henrique9@gmail.com

[2]Aeronautical Institute of Technology, Computer Engineering Graduate neansegura@hotmail.coml

II. Theoretical Background

A. Univector Field Navigation

Univector field navigation is a method used for guiding autonomous agents, such as robots or drones, through complex environments. This navigation approach involves encoding the desired direction of motion as a vector field, referred to as the univector field. The univector field assigns a specific direction to each point in the environment, guiding the agent to move along the vector direction towards its goal. Unlike traditional waypoint-based navigation, univector field navigation allows for smooth and continuous motion, enabling agents to maneuver through intricate and dynamic environments. This technique is particularly useful in scenarios where the agent needs to adapt its path to avoid obstacles or find efficient routes in real-time, making it a valuable tool for autonomous navigation in challenging and changing surroundings.

The univector field navigation used in the ITAndroids team is a composition of two fields. One of those, the Move-to-goal univector field ϕ_{TUF}, is a composition of two hyperbolic spiral univector fields that leads the robot to move to its destination with the desired angle:

$$\phi_{TUF} = \begin{cases} \phi_{h(CW)}(p_r, d_e) & y \leq -d_e \\ \phi_{h(CCW)}(p_l, d_e) & y \geq d_e \\ \frac{y_l N_{h(CCW)}(p_l,d_e) + y_r N_{h(CW)}(p_r,d_e)}{2d_e} & else \end{cases}$$

$$\phi_h(p, d_e) = \begin{cases} \theta + \frac{\pi}{2}\left(2 - \frac{d_e + K_r}{\rho + K_r}\right) & \rho \geq d_e \\ \theta \pm \frac{\pi}{2}\sqrt{\frac{\rho}{d_e}} & 0 \leq \rho \leq d_e \end{cases}$$

$$p_l = [x, y - d_e]^T, p_r = [x, y + d_e]^T, \tag{3}$$

$$y_l = y + d_e, y_r = d_e - y, \tag{4}$$

$$N_h(p, r_e) = [\cos\phi_h, \sin\phi_h]^T \tag{5}$$

θ angle from x-axis at the position p
K_r adjustable parameter
ρ distance between the origin and the position p
d_e predefined radius that sets the size of the spiral,
CW clock wise

CCW counter clock wise

The other univector field is the Avoid-obstacle ϕ_{AUF} it makes the robot avoid moving or standing still obstacles. It is based in the repulsive univector field and uses a virtual obstacle to predict its movements:

$$\phi_{AUF}(p) = \phi_r(p - P^{'}_{obstace}), \quad (6)$$

$$\phi_r(p) = \theta, \quad (7)$$

$$P^{'}_{obstacle} \begin{cases} P_{obstacle} + \boldsymbol{s} & d \geq s \\ P_{obstacle} + \frac{d}{s}\boldsymbol{s} & d \geq s \end{cases}$$

$$\boldsymbol{s} = K_0(V_{obstacle} - V_{robot}) \quad (9)$$

K_0 angle from x-axis at the position p
$P'_{obstacle}$ virtual obstacle's position,
$P_{obstacle}$ real obstacle's position
P_{robot} robot's position
d distance between $P_{obstacle}$ and P_{robot}
$\overrightarrow{V}_{obstacle}$ obstacle's velocity vector
$\overrightarrow{V}_{robot}$ robot's velocity vector

In possession of the two univector fields, we can obtain its composition, that we wish to optimize as it follows:

$$\phi_c = \begin{cases} \phi_{AUF}G(R', \sigma) + \phi_{TUF}(1 - G(R', \sigma))R > d_{min} \\ \phi_{AUF}R \leq d_{min} \end{cases}$$

$$G(r, \sigma) = e^{-\frac{r^2}{2\sigma^2}} \quad (11)$$

$$R' = R - d_{min} \quad (12)$$

B. CMA-ES

The Covariance Matrix Adaptation Evolution Strategy is an evolutionary optimization algorithm used for solving complex and non-linear optimization problems. It is particularly effective in scenarios where the objective function is continuous and may have a high-dimensional search space.

At its core, CMA-ES maintains a population of candidate solutions represented by a multivariate normal distribution. The distribution is characterized by two main components: the mean vector, which represents the center of the distribution and acts as the current generation's solution vector, and the covariance matrix, which controls the spread and orientation of the distribution in the search space.

During each iteration, CMA-ES generates a set of offspring solutions by sampling from the current distribution. The mean vector of these offspring represents the mean of the next generation. The offspring solutions are then evaluated using the objective function to determine their fitness or quality. From this offspring population, the algorithm selects the best solutions, which contribute to updating the mean vector and covariance matrix for the next iteration.

The mean vector is updated to move towards the better solutions in the search space, thereby focusing the search on promising regions. Meanwhile, the covariance matrix is updated to reflect the correlations between the search variables and control the distribution's spread and orientation in the search space. The covariance matrix adaptation allows CMA-ES to strike a balance between exploration and exploitation, efficiently navigating complex landscapes to converge towards the optimal solution.

C. Reward Engineering

Reward engineering is a crucial aspect of designing and fine-tuning reinforcement learning systems. In the context of artificial intelligence and machine learning, reward engineering involves defining and shaping the rewards given to an agent based on its actions in an environment. The rewards serve as feedback to the agent, guiding it towards learning optimal behaviors and achieving desired goals. By carefully designing the reward function, developers can encourage the agent to exhibit desired behaviors and discourage unwanted actions.

Diffent choices of the reward function can greatly impact the behaviour of the agent, giving one and only reward at the end of a successful task can be harmful to the system since it does not reward positives action immediately and it masks which action contributed for the success of a task. In the other hand, giving more rewards for every single correct action taken while being better in the sense of rewarding good actions it is mostly certain harder to model than a single reward at the end of a correct task.

III. Implementation

This work aims to optimize the following univector parameters: d_e, K_0, K_r, σ, and d_{min}, using not only a reward for goals score but also using a control metric for evaluating the parameter vector. It was used a reward to incentives the agent to follow the line of the predicted trajectory.

The idea of the reward engineering it is very simple. We used in this work the difference between the angles that are predict by Univector Field Navigation(Here we got some bias, because we have to start with initial values) and the actual angle that the agent it is following as the control parameter. We used the class of the agent: class UnivectorExtremeAttackersOptimization_AgentTest implemented by [1].

First we started by creating the privates functions calculateAngleDifference and calculateDeviation. The calculateAngleDifference calculates the smaller absolute difference between two angles(it could be an angle θ or $2\pi - \theta$) and the calculateDeviation function calculates the differences between all angles of a trajectory(sequence of angles).

In the UnivectorExtremeAttackersOptimization_AgentTest::mainThread() of [1] implementation, we initialized the variables angle1, angle2, angle3 and angle4 to storage the mentioned angles of the two attackers robots. We also initialize the accumulateDeviation variable,

to accumulate the deviations of the trajectory in every loop. The implementation done in that form, takes into account that deviations in the beginning of the trajectory are added up more times, so that the cma-es would avoid deviations in the beginning, which could propagate the error to the rest of the trajectory.

In every match we create the trajectories 1, 2, 3 e 4 to the storage the respective angles. To get the angles of univector and actual it's the difficult part of the implementation, due to the big complexity and extension of the VSS project. We start by taking information of the simulator using: modeling::WorldModel& worldModel = modeling::WorldModel::getInstance(); After this we initialize the variables previousUnivectorControlAngle and secondPreviousUnivectorControlAngle.

To get the information of Univector Field Navigation, we go to the ally.h file and create a variable double angleControlledByUnivector; which we implement in UnivectorControl.cpp as it follows: $univectorData- > player- > angleControlledByUnivector = univectorData- > referenceAngle$;

To use the information of the simulator we create $shp < representations :: Ally > robot = worldModel.getPlayerById(robotId)$ pointer and get the angle using the get.rotation() method. The information of Univector is collected by the angleControlledByUnivector variable implemented, we always use the info that univector sent one instruction before, so if we are in iteration number n of the loop we are using the actual angle and the angle sent by Univector on iteration n-1.

Then, the accumulated deviation is given by using the calculateDeviation and the trajectories from simulator and from Univector command after we push the angles that we obtained. The output line is given by accumulatedTimeToScore[1]+accumulatedDeviation, which the CMA-ES will try to minimize.

To understand the relevance of doing this reward engineering, we divided the implementation in three forms: (i) No use of reward-engineering, (ii) Implementing for just one robot of the attackers and (iii) Implementing for both attackers, as described previously.

We built the agents op_agent and def_agent, for the team attacker for optimization and for the defensive team respectively. Finally, we ran the run_extreme_optimization.py file and got the results described in the next section.

IV. Results and Discussion

For the case that didn't use any reward engineering [1] the results are as it follows:

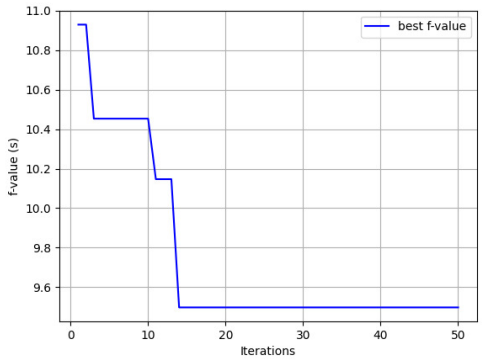

Fig. 2. No reward engineering f-value

TABLE I
Optimized Parameters withour reward engineering

Parameter	Value
d_e	0.10733474544355358
K_r	0.3349282527953902
σ	0.07389630554845375
d_{min}	0.1140344239715725
K_e	2.8845181562706768
K_0	0.05132130481795742
σ'	0.04217055608945829
d'_{min}	0.0052953689423864605
$\sigma_{Opponent}$	0.0009237019912197586
σ_{Ball}	0.3987059958531254

Implementing the reward engineering previously explained the results obtained are the following:

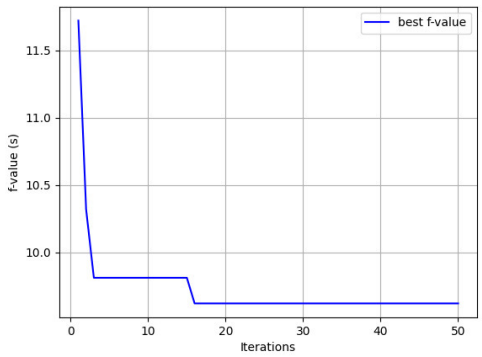

Fig. 3. One agent best f-value x iteration

Parameter	Value
d_e	0.15450297269947078
K_r	0.27977824594981754
σ	0.033526072089658875
d_{min}	0.02664196142232531
K_e	2.81987881771746
K_0	0.03690349301567551
σ'	0.03855778288166193
d'_{min}	0.09380390041530917
$\sigma_{Opponent}$	0.03500235662108152
σ_{Ball}	0.5639684897810793

TABLE II

OPTIMIZED PARAMETERS USING CONTROL REWARD WITH ONE OF THE ATTACKING ROBOTS

The last case, in which both attacking robots used the modified reward function generated the following results:

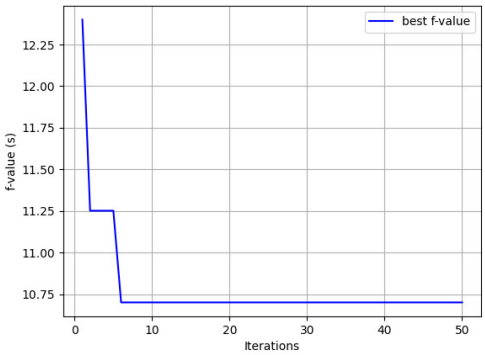

Fig. 4. Two agent best f-value x iteration

Parameter	Value
d_e	0.12483970911267323
K_r	0.44883881076398785
σ	0.00010395140625529879
d_{min}	0.03166878593856747
K_e	2.916688004553248
K_0	0.14985682728497965
σ'	0.0003746697804858056
d'_{min}	0.04436891168410736
$\sigma_{Opponent}$	0.006255839757039329
σ_{Ball}	0.5755576356734309

TABLE III

OPTIMIZED PARAMETERS USING CONTROL REWARD IN BOTH OF THE ATTACKING ROBOTS

It can be seen that the addition of a reward engineering process cost more time for the best-value of the function to converge, since after 50 iterations, the minimum cost increases as we put the reward system in one and two agents.

It can also be seen that no task could approach 0, which can be explained by the number of iterations. To run all the simulations on CLion, the time of the matches accumulated gets too big, to run 50 iteration we spent on average 2h, this in the accelerated mode. Better results could be obtained change the max number of iterations to 200, it's left this option in the code too.

We can see by the table of parameters that they change significantly, which means that control it's a good and more intermediate metric to determine the Univector parameters.

V. CONCLUSIONS

By implementing a model for the reward function that prioritizes immediately right actions through reward engineering concepts, significant improvements in the VSS team's performance were observed. The emphasis on rewarding timely and accurate actions enabled the agents to learn more effectively and make better decisions during the matches. Additionally, the introduction of more frequent rewards, offering feedback at intermediate time steps or states during the VSS football match, proved crucial in tackling the challenges posed by the complex environment.

This approach facilitated faster learning and aided the agents in navigating the intricate dynamics of the game. Ultimately, the combination of reward engineering principles and the incorporation of more frequent rewards showcased promising results, enhancing the overall performance of the VSS team and elevating their competitive capabilities in the football domain.

VI. REFERENCES

[1] T. Baldão, M. Maximo, and T. Yoneyama, "Optimizing Univector Field Navigation Parameters Using CMA-ES," in Anais do XIII Simpósio Brasileiro de Robótica e XVIII Simpósio Latino Americano de Robótica, Online, 2021, pp. 318-323.

[2] "The IEEE Very Small Size Soccer League," Available: `https://vsssleague.github.io/vss/index.html`. [Accessed: July 20, 2023].