Enhanced Subsumption Architecture for Multi-Agent System Planning and Merging

Fabio Sancinetti

fabio.sancinetti@pm.me
https://github.com/fsan

***Abstract*—Create a plan to solve tasks and merge the results is a common research problem studied by robotics and logistics. The work proposed by [2] and [3] suggests that many tasks can be solved by intelligence arisen from simple algorithms that not rely on symbolic AI processing. This work intends to present an enhancement to Subsumption Architecture of [2] and detailed at [11] by adding an layer of non-symbolic deliberation level on the top of Subsumption Architecture. In this model, the Subsumption Architecture controls the solution to stay in solvable condition and the non-symbolic deliberation layer acts to create and merge sub-plans. Both parts rely on prediction systems based on MLP. For testing the idea the OpenAI gym Lunar Lander challenge was chosen. The proposed architecture demonstrated capability of creating a reasonable plan during the simulation but faced problems related to the curse of dimensionality.**

***Index Terms*—Multi-Agent Systems, Subsumption Architecture, Distributed Problem Solving, Machine Learning**

I. Introduction

One approach of dealing with complex tasks is to split the main task and work on sub-plans for solving easier tasks and merge the results to achieve the solution. In many situations, planning much ahead is not an option because it is not possible to predict the future state with certainty. Also, although desirable, it is not true in dynamic systems that merging sub-plans will always cause an improvement in the direction of solution. This work presents a situation such as described and propose a solution based on the ideas of Subsumption Architecture [2] and Hybrid Architecture (Reactive Agents + Situated Automata) [11].

II. The problem

The main objective of this work is to tackle the planning in dynamic environment. The Open-AI Lunar Lander challenge was chosen because it has a built-in structure for results evaluation and presents the qualities that this work proposes to deal with.

A. OpenAI Gym

OpenAI is a non-profit organization that among several projects provides challenges that can be tackled by researchers and students. The Gym is a toolkit for development developing and comparing reinforcement learning (RL) [9]. The Gym scenarios are detailed by the authors in [1]. In this paper, the Gym is not used for RL, but works as a interesting challenge to test the technique proposed.

B. Lunar Lander

Among the scenarios from OpenAI Gym there is the Lunar Lander. The Lunar Lander consists on a landing module approaching to the surface of a moon displayed on figure 1. The ship has three engines: main, left and right. The main engine is the strongest and is kept in the center of the ship. The main engine pushes the ship in the normal direction of the angle of the ship. The left and right engines pushes (consequently rotates) the ship in the opposite direction, i.e. the left engine pushes the ship the right. Only one engine can be active per simulation step. Once active, the engine keeps throttling during all the period of the simulation step. The ship has infinite fuel, but each time an engine is used a discount on the total score is applied proportional to the power of the engine. The challenge also applies penalties for bad landing considering the force and the angle which the main module is pushed against the ground. The objective is to touch the ground gently using the left and right legs of the ship. Hitting the surface with strong force will cause the legs to spread and make the main module base to collide then accumulating score penalties. The challenge does also penalize the score if both legs of the ship touches the ground but then loose contact due the action of any of the engines. The target location is always in the middle over a plain surface, but the surroundings may change between runs. The beginning of each run gives a random initial movement vector (horizontal, vertical and rotational speeds).

The Lunar Lander has continuous and discrete variations. In the former variation, the engines can be activated partially. This means that the engines can be activated with a minimum force of 50%. As stated in [8], according to Pontryagin's principle, the optimal scenario to fire an engine is full throttle or turn it off, so it was defined at [8] the discrete scenario where the engines can be only activated this way. Only one engine can be active in each simulation step.

C. Additions and differences to the problem

The Lunar Lander physics is based in Box2D engine [4]. No prior knowledge of the Physics Engine is used in this research. No information about the ship capabilities or layout is inputted to the system. No information about the functioning of the world is defined except the target location, so no general specification about the challenge is defined. Even with these limitations, approaches such as in the heuristic approach

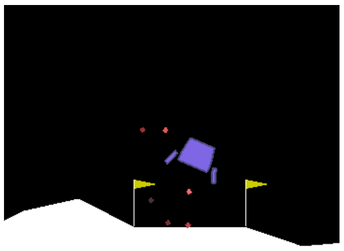


Fig. 1. Lunar Lander from OpenAI gym approaching ground

provided by the example source code [8] is possible. For example, the heuristic approach takes in account the position of the legs in relation to the main body of the ship. The lack of this information, as seen in the VI makes the challenge much harder.

As mentioned in II-A, the core of the design for the challenge is to serve as a challenge for RL, but this work does not adopt this approach. As it is explained on IV, the runs are independent from each other, and except from the training phase, no data or side effect is kept from each run.

III. Material and methods

The OpenAI Gym is offered in Python and although integration with other tools and languages is possible the solution proposed in this work is written only in the same language. The support for Machine Learning, such as for Multi Layer Perceptrons, is provided by Scikit-Learn at [10] within BSD License.

IV. Approach

This work proposes to consider each engine as an different agent for this system and add a coordinator agent. The existence of a central actor was chosen for convenience to avoid passing multiple messages among the agents until they agree in a solution, but removing the coordination is possible and should not affect the results.

The execution is divided in two parts: *Learning* and *Simulation*.

A. Learning phase

The Lunar Lander challenge have three observable outputs for each simulation step: instantaneous reward, state observation and the done flag.

As stated in item II-C, in the beginning of the execution the agents will not have information regarding the environment, except by its initial state. No information regarding the throttle power of the engines, the layout of the lander or the terrain is input.

The *instantaneous reward* is determined by OpenAI Gym and should be considered as an evaluation of how well the last action performed, so the sum of this value in each step by the end of the run determines the total score of the run. The OpenAI Gym system does also compute the penalties mentioned in item II by giving negative rewards. The *done flag* determines whether the run has stopped. This flag may indicate that the lander has successfully landed, left the observation area or crashed into the ground. The *state observation* is composed by:

- Horizontal and vertical position of the center of main module
- Horizontal and vertical speed
- Angle of the main module
- Angular speed
- Two values indicating if the left and right legs of the lander have touched the surface

In the former phase, the lander will be submitted to the forces of the environment and will randomly choose its next action. The observations of the current state, current action and next state are registered and used for training a system capable of predicting the next state and reward.

The prediction subsystem are then divided in two parts: *state prediction* and *instantaneous reward prediction*. Both subsystems are executed separately from the simulation for convenience. Both subsystems were implemented as MLP Regressors provided by Scikit-Learn.

The *state prediction* subsystem has one single hidden-layer with 100 neurons using RELU activation, Adam optimizer and constant learning rate. It receives as input one *state observation* and the identification of the active engine and output the prediction for the next *state*. These parameters are enough to obtain a score with average of 0.91 when submitting approximately 300.000 observations.

It is much harder to obtain an MLP Regressor for *instantaneous reward prediction* subsystem as the system has several rules for penalties. The end MLP Regressor used for this subsystem has three hidden layers with layout 27-54-27. It uses RELU activation, and Adam optimization as well. It was used Grid Search optimization algorithm, also provided by Scikit-Learn, to try to improve the parameters for this MLP Regressor but the results vaguely differences from the defaults for MLP Regressor by Scikit-learn. It receives one *state observation* and the identification of a engine and outputs a reward value. With these parameters and approximately 300.000 observations the average score of this regressor was 0.52.

The considerably bad performance for *instantaneous reward prediction* subsystem is actually not a problem for the execution of this work because this value is only used in one of the strategies during the simulation phase.

B. Simulation Phase

The simulation phase is the execution of Lunar Lander scenario. Each engine is considered as an agent. The coordinator agent is optional, but for the proof of concept it was modeled save more messages across the agents and extra work over in the negotiation step.

At each step, the current state is captured and passed to the coordinator agent. The coordinator agent is the responsible for accessing the Enhanced Subsumption Architecture (ESA). To access the top level of sub-plan deliberation the coordinator has to pass through the checking of all other levels. These other levels are intended to act in situations where the solution could become unsolvable. For the Lunar Lander, this would be a crash, loose proximity with the target point or the end of the landing process.

If any of the lower levels of the ESA is achieved, the coordinator agent selects the appropriate strategy and passes the task to the agents. Then each agent will evaluate which action is the most appropriate for accomplishing the task by using a minimization task that is packed with the strategy provided by the coordinator and submit their result. When all agents finish posting their results, the coordinator will select the best result through the English Auction using the fitness of the pair $(strategy, action)$ provided by the engine agents.

If the coordinator does not meet any obstacle, it will send to the agents a task for choosing the next strategy, the current state and the meta-heuristic function (MHF) 1. The agents will check for every possible choice of actions in their capabilities the best result for the MHF. In such way, the tasks of finding the next action is distributed among the agents. The agents will emulate their choices for N simulation steps and the best fitting sub-plan for the MHF will be shared with the coordinator who will use the English Auction to pick the best plan. It is possible in this step to have the merging of sub-plans into a new better sub-plan, but this would need more computational power by requesting the re-planning for every agent to provide new predictions and fitting for every selected step of the sub-plan.

The general algorithm flow considering the both top and lower layers of the ESA is shown in the figure 2.

The sub-plan size is a shared parameter for all agents of the system. Having a long sub-plan have two shortcomings: (1) The *state predictor* subsystem is reliable but it has small errors that will accumulate after each step of the sub-plan and will generate a considerable difference from the real state during execution, and (2) long sub-plans would demand more computational power as re-planning would be a necessary step during the execution of each sub-plan. To overcome these negative points, a sub-plan size should be though in order to be effective in the prediction and not too long to cause re-planning.

$$MHF(s) = \min\left(\Delta height + \frac{1}{distance_{target}}\right)$$

where $\Delta height$ is the variation in height between the current state s and $distance_{target}$ is the euclidean distance to the center of the target

(1)

C. Strategies

During *Task distribution* the coordinator will try to reach the top level of ESA. If it met a condition in a lower level of ESA, it will select a pre-defined strategy to solve the situation before letting the engine agents deliberate about the rest of the plan. If the top level of ESA is reached, the coordinator will provide the MHF 1 and let the engines process in parallel the best strategy each one can provide.

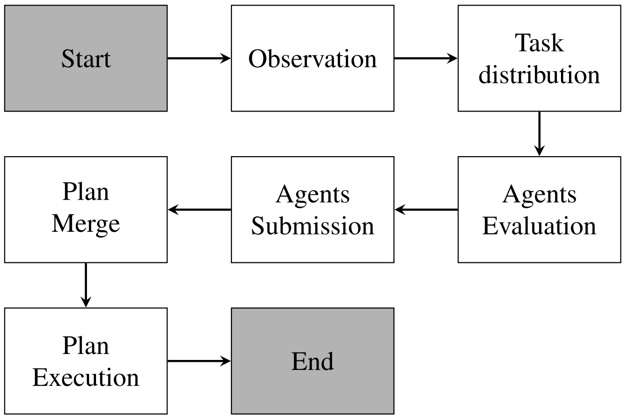


Fig. 2. The flowchart represents every actions taking during the selection of next sub-plan.

The strategy in both lower and top levels of ESA is posed as minimization or maximization problem. i.e.: (1) If the lander is too far from the proximity point and will leave the area destined to the simulation, before continue descending the lander must return to a safer zone, or (2) if the lander is starting to rotate aggressively the priority will be to reduce the angular rotation.

The ESA architecture considering top and lower levels are shown in figure 3.

D. Top ESA layer

The strategies the engine agents can select by deliberating are:

- **Slow-Fall (SF)**: minimizes the directional and angular speeds but incentives the descending. Eq. 2
- **Reduce Acceleration Angle (RAC)**: minimizes the angle of the vertical descending. Eq. 4
- **Reduce Angular Velocity (RAV)**: similar to RAC, but includes an argument to approximate the target. Eq. 3
- **Zero Movement (ZM)**: similar to SF, but includes an argument to approximate the target in horizontal direction. Eq. 5

Each strategy has an associate minimization function that the each engine agent uses to evaluate the result of its possible actions.

$$Strategy_{\mathrm{SF}} = \min\left(|angle_{speed}| + |angle| + |pos_y| + \alpha * \left\|speed_{x,y}\right\|\right) \quad (2)$$

$$Strategy_{\mathrm{RAV}} = \min\left(\alpha * |pos_{x,y}| + |angle_{speed}| + \sqrt{(speed_y)^2}\right) \quad (3)$$

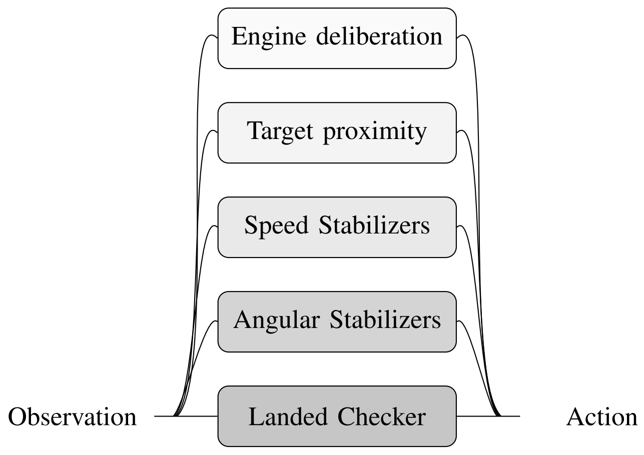


Fig. 3. The Enhanced Subsumption Architecture (ESA) for the Lunar Lander case.

$$Strategy_{\text{RAC}} = \min\Big(1 - \big(\frac{speed_{x,y}}{\|speed_{x,y}\|} \cdot \begin{bmatrix}0\\1\end{bmatrix}\big)\Big) \quad (4)$$

$$Strategy_{\text{ZM}} = \min\left(\sqrt{(pos_x)^2 + (pos_y - 1)^2} + \left\| \begin{bmatrix} speed_x \\ speed_y \\ angle_{speed} \end{bmatrix} \right\| \right) \quad (5)$$

E. Lower ESA Layers

Layers in lower levels of ESA are placed before requesting the predictions of the engines to make the engines have to comply to a selected strategy instead of choosing one. The subsumption architecture is displayed on figure.

The ESA lower structure have four levels for ensuring the next step of the problem being solved does not become unsolvable. For the Lunar Lander, these layers represent situations of danger, disconnection with target proximity or the end of the process. These levels are:

- **Landed checker (LC)** verifies if both legs have touched the surface and if the horizontal distance to the target position is correct. If both conditions match, this layer posts an strategy to the agents that does not allow the engines to be turned on again. This is useful for finishing the simulation and avoiding the penalty for departure after touching the ground.
- **Angular Stabilizers (AS)** observes the angle and angular velocity of main module. If the angle passes 0.4π radians to the left or the right, the engines receive an stabilizing strategy to minimize angle speed. Eq 6.
- **Speed Stabilizers (SS)**: intends to minimize the angular and directional speeds while keep tracking of the horizontal coordinate of the target. Eq. 7.
- **Minimize Horizontal Distance (MHD)**: intends to reduce the horizontal distance to the target without loosing track of the vertical speed. Eq. 8.

$$Strategy_{\text{AS}} = \min\Big(\alpha * pos_x + \beta * pos_y + |speed_y| + |angle| + |angle_{speed}|\Big) \quad \text{where } \alpha < \beta \quad (6)$$

$$Strategy_{\text{SS}} = \min\Big(|pos_x| + |vel_y| + \alpha * |vel_{x,y}| + |angle| + |angle_{speed}|\Big) \quad (7)$$

$$Strategy_{\text{MHD}} = \min\Big((pos_x + \frac{1}{pos_y} + |speed_y| + (-angle_{speed} * \|\begin{bmatrix}speed_x\\speedy\end{bmatrix}\|) + speed_x + angle_{speed}\Big) \quad (8)$$

F. Sub-Plan Selection and Merging

When the coordinator receives the results from the engine agents it has to merge the results to form a final sub-plan. The possible options for realizing this task are:

- Select the best-fitting received plan according the MHF 1.
- Create a payoff matrix and select the best results that does not create a catastrophic situation
- Instead of receiving the N_{th} receive only the first prediction for each agent, select the best according to MHF 1 and cause the other agents to re-plan the sub-plan.

Select the best-fitting result is easier because it can be reduce to a maximization problem such as the English Auction and was chosen because its simplicity and clarity represented in the equation 9.

The payoff matrix [11] is a concept from Game Theory where each agent chooses its action based on the fact of having a payoff by cooperating or defecting. This approach would be the selected if there was no coordinator agent. Yet, with the coordinator being present the payoff could be used to find potential catastrophic simulation. This would be done by creating a $E+1$ dimensional matrix considering all agents and adding a dimension for each step iterative after verifying the best result for the previous step.

The re-planning strategy is similar to the English Auction, but consider only the result for the N_i step each time and making the other agents to re-plan the next N_{i-n} and disputing again for the control of the step instead of the plan.

$$\text{Auction Result} = \max\Bigg(\big(Strategy(state_{current}, Engine_j)_i\big)\Bigg)_{\substack{\forall Engine_j \in E \\ \forall Strategy_i \in S}} \quad \text{where } E \text{ is the set all engine agents and } S \text{ is the set of all strategies in which the } Engine_j \text{ can take.} \quad (9)$$

Another advantage of not letting each agent own a step is that this could cause the system to be unstable if two agents selected contradictory strategies.

V. Related work

This work applies concepts of Multi-Agent Systems and Machine Learning developed by many researchers over the years.

A. Perceptrons and Multi-Layer Perceptrons

The concept of artificial neurons, named Rosenblatt's Perceptrons were constructed around the McCulloch-Pitts model. The model consists in weighted linear combination of inputs followed by a hard limiter [7]. This concept is used on supervised training because it is possible to use backpropagation to change the weights thus making the perceptron a good candidate for linear regressor and classifier. The Universal Approximation Theorem determines that it is possible to approximate any continuous function on compact subsets from combinations of linear equations [5]. Together both concepts allowed scientists gathering and stacking perceptrons to create systems that approximate other functions. One possible combination of these structures are known as Multi Layer Perceptrons (MLP).

MLP are formed by one layer as being the input, one or more layers as the hidden layers and one final layer as the output as exemplified in figure V-A. The components of the MLP are linked in such way that each element of each layer is linked to all elements of the next layer. The backpropagation mechanism is an essential tool for setting weights during the training process.

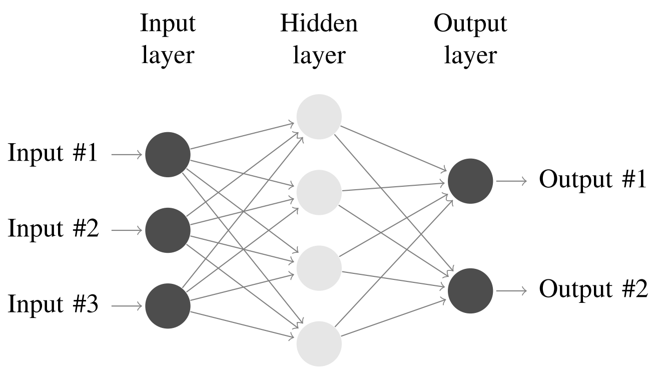


Fig. 4. Example Multi Layer Perceptron with 3 inputs, 4 neurons in the hidden layer and 2 outputs.

B. Multi-Agent Systems

The Multi-Agent System (MAS) commonly refers is generally applied to systems that can be composed by multiple interacting agents [6]. According to [11], an agent is a computer system situated in some environment and that is capable of autonomous action in order to meet its design objectives.

Besides the concept of MAS itself, this work applies ideas of some common concepts in MAS research:

- Reactive Agents are the simplest form of agents which only reacts to conditions of the environment. [2] demonstrates that it is possible to obtain a collective intelligence to realize a task using this approach.
- Hybrid Architecture does reasoning through symbolic AI and uses reactive systems as support for realizing the tasks [11]
- Subsumption Architecture by [2] and [3] is a control structure is not based on symbolic representation. It decompose behaviors into sub-behaviors into a hierarchy of layers [11].
- Auctions are a common way of competing by the ownership of a property or item. There are several types of auctions, but this work uses the English Auction System. This model uses the first-price to determine the winner, has open cry for buyers and ascending pricing [11].
- Task sharing takes place in the problem decomposition and allocated to several agents. Result Sharing is used to gather the information and changes in the environment made by the agents into the synthesis 'of the solution [11].

VI. Results

The Lunar Lander challenge presents to the world a measure of success given by the sum of all rewards during the system execution. According to [9] and [9] the descending process accumulates between 100 and 140 points and landing touching both legs and not having the core of the ship touch the surface adds another 100 points.

The first step was validated that it was highly improbable that the ship could realize the task without any interaction. As expected the positive outcomes are as low as 0.001%.

The second step was to validate that if there was any possibility of random actions could result in positive results in the total reward system. As expected, the positive outcome are as low as 0.001%.

Two variations of the ESA were set for the validation of the proposed solution:

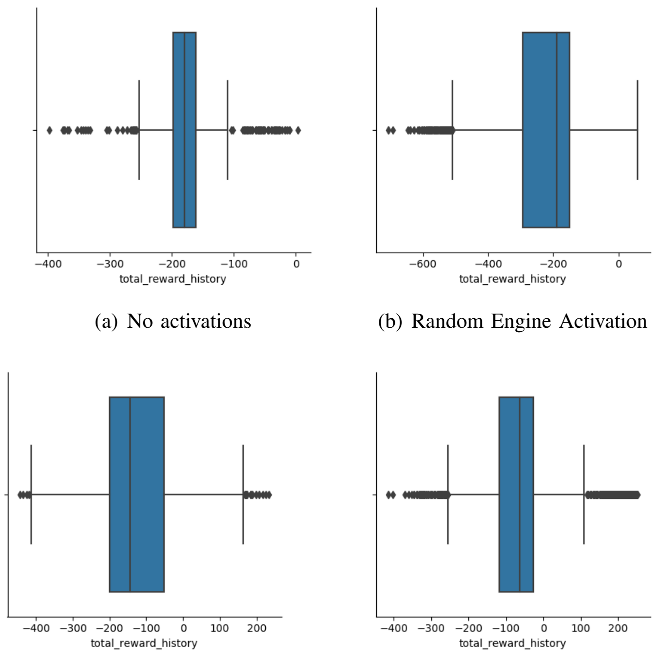
1) **ESA with Automatic Strategy Selection** selects the best sub-plan through the meta-heuristic function except when a dangerous situations (detected by lower layers of the subsumption architecture) are detected and use the sets of heuristics mentioned in item IV-C.
2) **ESA with Strict Rules** selects the based best strategy automatically based on observations of height of the ship. The set of strategies used by this solution is bigger but all strategies are simpler than the ones mentioned in IV-C. This variation after checking for potential harmful situations posts the tasks to the agents based on the height and distance to the target variables.

The results display a better performance with automatic strategy selection. The distribution of the scores are shown in the figure 5.

The success rate in both cases are considerably low. Checking the executions it is possibly to observe the ESA and the

# Tests	Condition	$Score >= 0$	Avg score
1000	No action	1	-178.37
1000	Random Action	1	-227.56
1000	ESA-Strict	166	-51.15
1000	ESA-Auto	246	-111.26

TABLE I
SCORE RESULTS ON DIFFERENT SCENARIOS



(a) No activations (b) Random Engine Activation (c) ESA-Auto (d) ESA-Strict

Fig. 5. The distribution of scores for the tests of table I

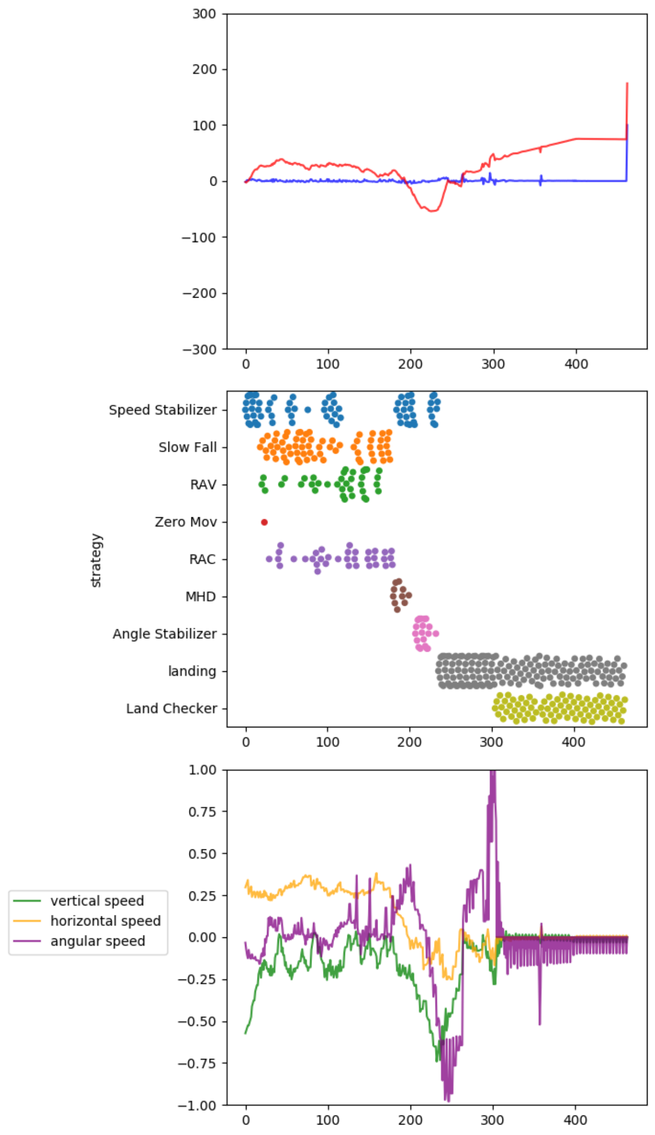


Fig. 6. One execution from ESA-Auto

automatic strategy selection mechanism are working together well as in the figure 6.

When verifying the locations where the simulation ended, shown at figure 7 it is possible to notice that ESA-Auto has a problem maintaining the proximity to the center and ESA-Strict correctly selects the target.

ESA-Strict has better performance at select landing but is not capable of smooth landing so it looses lots of points due to the collision.

The behavioral components do work as expected. The ESA lower layers has their priorities over the top layer system. And the ESA Top Layer is capable of selecting the best fitting strategy to approximate the target. One example of the interaction of among the agents, strategies is shown at figure 8.

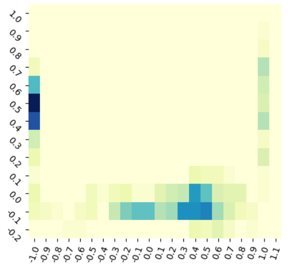
Investigating several situations such as shown at 8, it is possible to conclude that the MAS mechanism and the subsystems based on MLP works as expected. The coordinator and engine agents are capable of planning a solution based on a problem of maximization problem of a heuristic function that has its peak on the target location. The difficulties encountered are majorly caused by strategy functions boundaries.

The major flaws found during the analysis of the results were caused by interaction of two strategies from lower layers defined for the ESA of the Lunar Lander

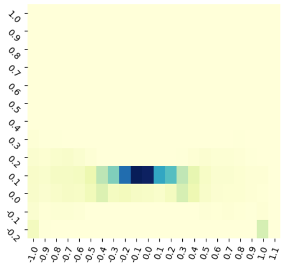
- *Angle Stabilizer Strategy* works well in the domain of $-0.5 < angle_speed < +0.5$ but completely fails out of these limits.
- *MHD Strategy* controls the approximation correctly in the domain where $0.5 < pos_x < 1.0$ but fails in $-1.0 < pos_x < -0.5$ because it cannot turn $angle$ enough to return to the center

VII. CONCLUSION

The solution proposed by this article propose an automated system that can plan the solution automatically. The core of this works is based on Multi-Agent System ideas presented by [2] and [3]



(a) ESA-Auto



(b) ESA-Strict

Fig. 7. Final location for ESA-Auto and ESA-Strict

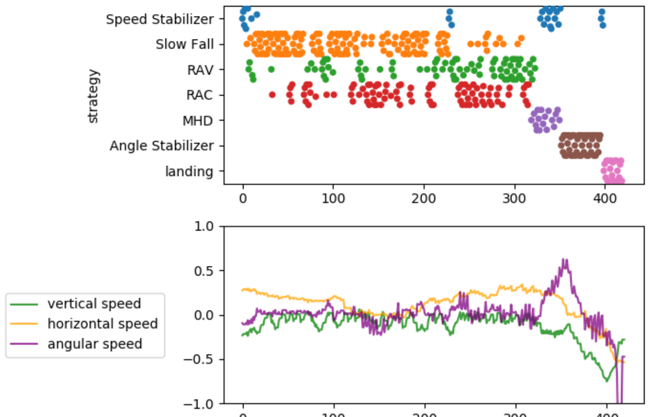


Fig. 8. Example of result of interaction among the lander subsystems and engines

The proof of concept developed during this research uses an enhanced subsumption architecture which turns the global system into a reactive agent that intended to avoid major problems, and in its top level uses a deliberation-like system that is lighter than symbolic AI processing.

This deliberation system has as input a meta-heuristic function that evaluates strategies chosen by each agent. Two subsystems are then used for evaluating, the strategies: *reward prediction* and *state prediction*. At each simulation step, each agent posts a tuple $(action, strategy, nextpredictedobservation)$ which is evaluated by the coordinator agent. The best-fit received strategy according to the meta-heuristic function is selected and a sub-plan is determined on the fly.

If the system is under a dangerous situation, the subsumption architecture enter in action before the deliberation system starts. The subsumption architecture does not let the agents selecting a preferred strategy. Instead it imposes the strategy to the agents and requests the actions of the agent to fulfill the current task.

During the result analysis, the bad scores were not caused by the proposed architecture, plan selection or merging. Flaws were encountered in two important subsumption strategy functions. Improving these strategies is a hard task due to the number of dimensions of the problem and the interaction between the tasks. A possible solution is to simplify the main strategies and have more layers at the subsumption architecture as demonstrated in ESA-Strict version.

The ESA-Strict final results show at 7 achieves the region of the target easily but fails to decelerate completely after the legs of the ship have touched the ground. Solving the landing problem is an easier task because during this phase there is no other strategies which can be active at same time.

VIII. Further Research and Work

To improve the results for this research, a machine learning system can be built to adjust the parameters for strategies or even develop new strategies based on observations. Further research in this direction could include the results of stacking many layers composed of the architecture proposed in this work and the machine learning for adjusting or creating strategies in order to solve complex tasks.

Acknowledgment

This project is a study for Multi-Agents System lecture at PPGEE program for *Escola Politécnica da Universidade de São Paulo*.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
[2] R. A. Brooks, "Intelligence without reason," in *International Joint Conference on Artificial Intelligence*, 1991.
[3] ——, "Intelligence without representation," *Artificial Intelligence 47*, 1991.
[4] E. Catto, "Box2d - a 2d physics engine for games." [Online]. Available: http://box2d.org/
[5] B. C. Csáji, "Approximation with artificial neural networks," *Faculty of Sciences; Eötvös Loránd University, Hungary*, 2001.
[6] E. H. Durfee and J. S. Rosenschein, "Distributed problem solving and multi-agent systems: Comparisons and examples," 1994, pp. 94–104.
[7] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.
[8] O. Klimov, "Lunar lander source code," GitHub, 01 2018.
[9] OpenAI, "Gym." [Online]. Available: https://gym.openai.com/docs/
[10] scikit-learn developers, "scikit-learn: machine learning in python." [Online]. Available: http://scikit-learn.org/
[11] M. Wooldridge, *An introduction to MultiAgent Systems*. John Wiley & Sons Ltd, 2002.