University of Amsterdam

MSc Artificial Intelligence
Master Thesis

The predator-prey evolutionary robots system: from simulation to real world

by
JiunHan Chen
11649712

July 10, 2019

36 Credits
February 2018 - September 2019

Supervisor:
Mr. Gongjin Lan
Prof. Dr. Guszti Eiben

Assessor:
Prof. Dr. Arnoud Visser

VU University Amsterdam

Graduate School of Informatics, Faculty of Science,
Universiteit van Amsterdam
Department of Computer Science, Faculty of Informatics,
Vrije Universiteit Amsterdam

Acknowledgements

Thank Dr. Guszti Eiben for inspiring guidance and providing us a favorable research environment. I would like to thank Gongjin Lan for guiding me to achieve the whole project and always sharing his rich experience with me. Of course, their help are far beyond a few words. Thank Dr. Arnoud Visser is willing to be my assessor, and he always provides me helpful and critical comments. Thank Dr. Diederik M. Roijers for the valuable suggestions. Thank anonymous experimental participants. Thank my family, especially my sister Yu-Chen Chen and brother-in-law Lutz Wiegart, for both the practical support and sharing life experience with me, I cannot finish my master program without them. Finally, I would like to thank Ya-Chu Yang for her endless support and warm company.

Abstract

In this thesis, we propose an evolutionary predator-prey robot system which can be generally implemented from simulation to the real world. The thesis can be separated into two main stages. In the first stage, the predators are driven by a concise neural network, and the prey is driven by a Gaussian model-based evasion strategy. We evolve the robot controllers in simulation, followed by evolution on real robots. The evolutionary search process is driven by a specific fitness function that balances between minimizing the distance to the prey, while avoiding collisions among predators. The best evolved controllers are further analyzed for their sensitivity to the initial positions of the robots before choosing the winning pursuit strategy. In the second stage, robots rely on the camera and the infrared sensors as inputs of controller. Both the predators and prey are co-evolved with NEAT to develop complicated behaviour. We integrate Gym of OpenAI, ROS(Robot Operating System), Gazebo to provide a framework such that users only need to focus on algorithms without being worried about the detail of manipulating robots in both simulation and the real world. Combining simulations, real-world evolution, and robustness analysis, it can be applied to develop good solutions for the predator-prey problem. For the convenience of users, the source code and videos of the simulated and real world are published on Github[1].

[1]`https://github.com/chenjiunhan/Predators_and_Prey/`

Contents

Chapter 1

Introduction

Predator-prey is a classical pursuit-evasion problem. A typical scenario is that there are a predator and a prey in a square arena, the predator must catch the prey within a certain time. The game ends when the predator catches the prey or time is beyond the limitation. The application of the solution of predator-prey problem might be used for searching, rescuing, exploration...etc. If we extend the predator-prey problem as multiple predators to catch a prey. The collaboration among multiple agents strengthen the reliability and scalability of completing a task.

The training for robots in the real world can be a time-consuming and money expensive task. An alternative way can be that we train the robots in the simulation world, and then we transfer the well-trained robots to the real world. Also, in the real world, we can even train the trained robots for a shorter time. The simulation environment can be created by Gazebo[1], which is a widely used 3D real-time simulation physics engine. Figure 1.1 shows the user inferface of Gazebo.

There are different methods to solve the predator and prey problem. An early work from Bryson and Baron considered the pursuit-evasion problem as a differential game, it means the solution of the pursuit-evasion problem can be solved analytically when the map and the pose of the predator and the prey are known[11]. Another the work was from Raboin et al., they invented an algorithm to solve partially observable pursuit-evasion problem with optimizing the uncertainty of the prey[21]. In the meantime, some researchers tried to solve this problem with **Evolutionary Algorithm** for various scenarios. We followed their steps and tried to explore interesting things from that point.

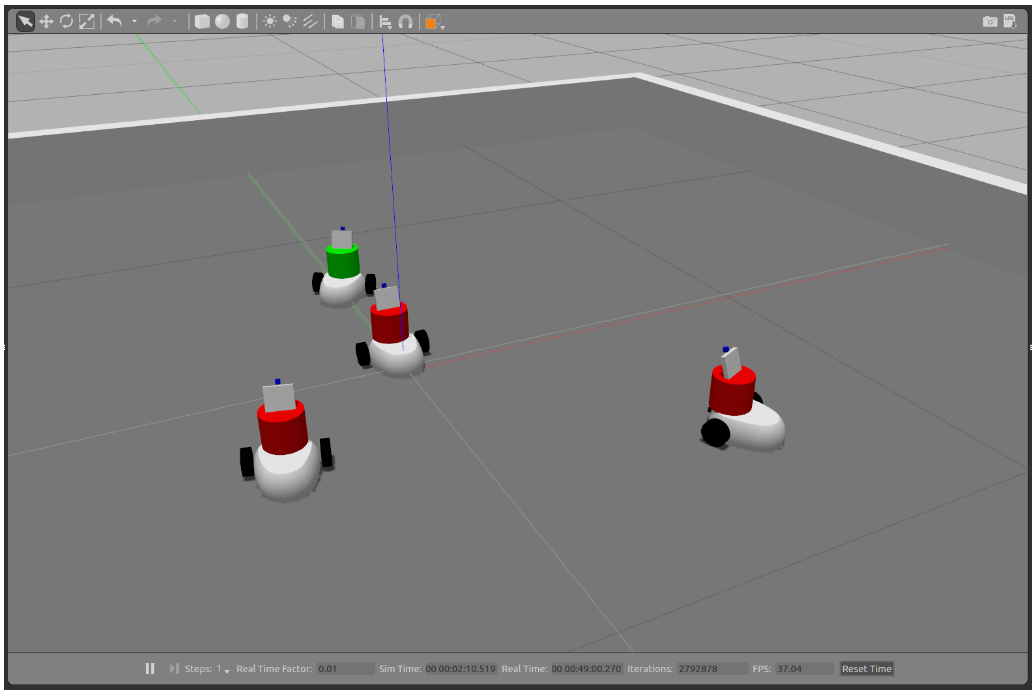
Our research topics includes that we implement different experimental settings to evolve the predators and show their advantages and disadvantages. Also, we aim to provide an simulated experimental environment for the predator-prey task. From an experimental point of view, many aspects should be considered in this task. We can ask ourselves: "What's the number of predators? Can we change the number dynamically? Predators are controlled by a single controller or multiple controllers? Do we evolve the predators and the prey at the same time or we use a fixed strategy for one party? Is the environment fully observable or partially observable?" These questions help us to develop our research.

The research is separated into two stages. In the first stage, we consider the pose of the predator and the prey can be extracted by a camera on the top of the field, which means that the environment is fully observable. Thymio II robots are used as the predators and the prey. We coordinate all the predators with a controller instead of allocating one controller for each predator, because we want to make the number of predators have scalability. In our result, the predators show collective behaviour like flocking. The prey is controlled by fixed strategy instead of coevolution for both predators and prey, because coevolution may suffer from "Red Queen Effect" which leads to the predators and the prey only try to beat each other with simple behaviour.

After finishing the first stage, we try different somethings in the second stage. we expand our work to a partially observable environment. Instead of using an overhead camera, the sensors from the robots are used. For the more powerful sensors, Robobo is introduced in the second stage. The robot has short-range(20cm) IR sensors and a camera by carrying a mobile phone on itself. In order to reduce the reality gap, we must simulate the camera and IR sensors in Gazebo. After finishing the evolution in simulation, we can transfer the evolved agents into the real world to further evolve or examine the performance. Both the predators and the prey co-evolved, even though the "arms race" might be not easy to be triggered, which is a key point for

[1]http://gazebosim.org/

competitive coevolution. Fortunately, ”Hall of Fame” is a technique that can stabilize the competitive coevolution process. We choose to coevolve the sensor-based predators with an all-knowing prey in the simulation world, and the sensor-based predators can be transferred to the real world immediately. In the simulation world, we can access the data which is hard to get from the real world to help the evolution process. An all-knowing prey might encourage predators to develop better performance. Also, the fitness function for the predators is computed by the distance between agents, which is hard to get in the real world.



Steps: 1 Real Time Factor: 0.01 Sim Time: 00 00:02:10.519 Real Time: 00 00:49:00.270 Iterations: 2792878 FPS: 37.04

Figure 1.1: Graphical user interface of Gazebo. It shows a part of our experiment. The green robot is prey and the red robots are the predators.

Chapter 2

Preliminaries

2.1 Fixed Strategy or Competitive Coevolution

To create the predator with higher complexity, should we use a fixed strategy prey or competitive coevolution between the predators and the prey? The work from Nolfi and Floreano has already answered this question[18]. They compared the performance of both the predator and the prey from coevolution and simple evolution, the agents from coevolution outperformed the agents from simple evolution. The reason that simple evolution fails to beat the coevolution is because of an invalid fitness value, which means the agent with simple evolution usually get zero for fitness value(1 for win, 0 for loss). We might be able to design an easier optimized fitness function to help simple evolution, but we can do the same thing for coevolution. Another challenge for the simple evolution is to design a competitive opponent, in contrast, the coevolution has the ability to finish the design automatically. This fact shows that the coevolution has the potential to evolve the agents with higher complexity. However, the competitive coevolution works only if the "arms race" is triggered. The arms race here means the predators must become better to catch a better prey, also the prey must improve to escape from the better predators, and the whole process goes on and on.

2.2 Cooperative and Competitive Coevolution

The cooperative coevolution happens among the predators, the multiple predators must collaborate to catch the prey, the work from Yong and Miikkulainen[27] evolved the predators with Enforced SubPopulations(ESP) such that the predators can collaborate without direct communication. Another work from Gomes et al. also evolved the predators with two approaches, a standard fitness-driven cooperative coevolutionary algorithm(CCEA) using NEAT and novelty-driven coevolution, and both approaches were successfully transferred to the real robots. The competitive coevolution happens between the predator and the prey. In the previously mentioned work from Nolfi and Floreano[18], both a predator and a prey evolved. Miikkulainen et al. performed competitive and cooperative coevolution at the same time, in other words, there are multiple the predators to catch one or two prey(s), and all of the agents were evolved[22].

2.2.1 Red Queen Effect

Red Queen effect means that a species must persistently adapt and evolve to survive and compete against also persistently evolving opposing spices. But it has no too much different change of the relative advantage between two spices. If the Red Queen effect happens in the coevolution of predators and prey, it means that the predators and the prey fail to evolve the complex behaviour, but they only change a little to beat the opponent and were trapped into a cycle to beat each other alternatively. Van Valen named this effect because of a quote from Through the Looking-Glass: "Now, here, you see, it takes all the running you can do, to keep in the same place."[26] Therefore, to evolve the complex behaviour, the key point is to make the "arms race" happen. In practice, the arms race is hard to be triggered. But, there are some techniques that may lead to the arms race, for example, a method "Hall of Fame" was proposed from Rosin and Belew[23]. The new individuals must be competitive against previously saved individuals. Another method is to increase the complexity of models for encouraging the arms race. An evolutionary algorithm is called NeuroEvolution of Augmenting Topologies(NEAT), and it can evolve the topologies of a neural network. When one side can outperform the other side because of the complexity of the model. The only way to beat the opponent is to increase the complexity too, which causes the arms race.

2.2.2 Hall of Fame

An evolving predator or prey should be able to beat the best opponents from earlier generations. Also, we want to avoid from that the evolution process is trapped into a cycle, so we break the cycle by using the evolving predator or prey to compete against the best opponents from earlier generations. This technique called ”Hall of Fame” was proposed by Rosin and Belew to increase the probability of triggering arms race. The opponents of the evolved target are 10 randomly selected best controllers from previous generations.

2.2.3 Master Tournament

For the coevolution, one question that we must answer is ”How do we know the behaviour of agents becomes more complicated to beat opponents?” S. Nolfi and D. Florean mentioned ”Master Tournament” in their paper to answer this question [6], which predator or prey compete against each best competitor of all generations. A theoretically ideal result should look like in the Figure 2.1. The evolved agents should be able to beat the opponents from early generations in theoretically. This graph implies that the difficulty for both sides is the same. However, in our task, one prey must compete with three predators, the difficulty is higher for the prey. So the graph is different for our experiment. After finishing ”Master Tournament”, we are able to accumulate the fitness as accumulated score, and the later generations should have higher accumulated scores.

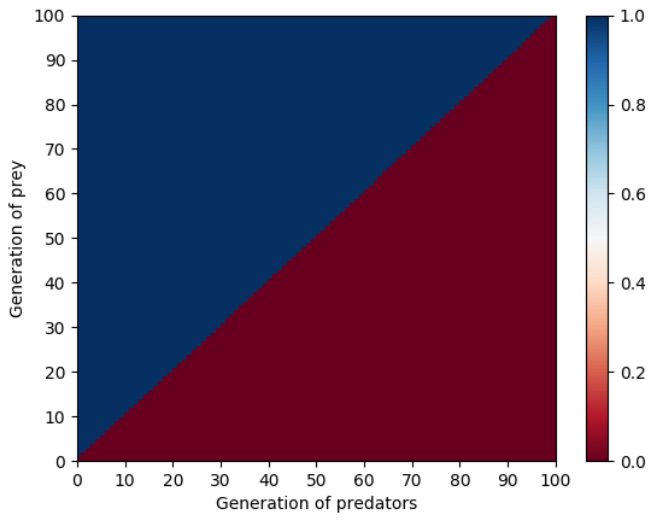


Figure 2.1: Theoretically ideal co-evolution result for one predator versus one prey. The agent from the later generation should beat the agent from the early generation.

2.3 Homogeneous or Heterogeneous Controller

The multiple predators can be controlled either by a homogeneous controller or multiple heterogeneous controllers. The former means there is only one controller for all the predators and the latter means that we allocate the different controller to each predator. When we consider the success rate for the predators to catch the prey, according to two reports from Haynes and Sen[10], Yong and Miikkulainen[27], the heterogeneous controllers outperformed the homogeneous controller at least in the prey-capture domain. But the homogeneous controller still shows interesting behaviour like team formation movement and flocking[1].

2.4 Fully or Partially Observable World to Robots

In a simulation world, we are able to directly access the coordinates of the predators and the prey in the world, this kind of worlds are fully observable. A fully observable environment makes training and evaluation easier. In the real world, robots have a limited range of sensor, which makes this world partially observable. One problem of a partially observable environment is that we must design a new fitness function. Take the popular fitness in the predator and prey problem for example:

$$F_{predator}(\tau, d_i, d_f) = \begin{cases} 2 - \frac{\tau}{T} & \text{if prey caught,} \\ max(0, \frac{(d_i - d_f)}{\ell}) & \text{otherwise} \end{cases}$$

τ is the time to catch the prey, and T is the constant duration of one episode. d_i and d_f are initial and final average distance from the predators to the prey. ℓ is the fixed edge length of the square arena. This fitness implies that we can measure the distance between the predators and the prey anytime, but for a partially observable environment, the predators even cannot be sure that an object in front of themselves is prey. Fortunately, we can still use a similar fitness function as above in the simulation world to evolved the agents as if they can only observe part of the world, because most of the data can be easily accessed from the simulation world. During the evaluation in the real world, we only need the data from robot sensors to make a controller work.

2.5 Evolutionary Algorithm

Evolutionary algorithm is able to solve various optimization problems. The algorithm is inspired by nature, including many concepts from the theory of evolution like crossover, mutation, selection...etc. At the beginning of an evolution process, the algorithm generates a group of candidate solutions for our problem. Next step, we evaluate the performance of these candidate solutions and assign a score according to a fitness function. The genes from candidate solutions with higher scores are usually passed to create the next generation of candidate solutions, at the same time, the genetic mutation may happen. The genes here may represent as weights and biases of a neural network. Figure 2.2 shows the standard process of evolutionary algorithm. There are some examples of classic Covariance Matrix Adaptation Evolution Strategy(CMA-ES), Evolution Strategies(ES), and Bayesian Optimization(BO). In a robotic task, our objective function can be interfered by many noise sources. According to the work from Hansen and Nikolaus, CMA-ES is especially suitable to deal with such noisy scenario[8]. Evolution Strategies from OpenAI is also competitive to modern reinforcement learning technique Trust Region Policy Optimization(TRPO)[24]. In our case, Bayesian Optimization is able to find an acceptable solution at the early stage, and also dominate the other algorithms. However, the search process time of Bayesian Optimization largely increases when the number of evaluation grows. This fact makes us choose CMA-ES as the algorithm that we apply in the real world.

2.5.1 NeuroEvolution of Augmenting Topologies

In our second stage, we use another evolutionary algorithm, which is called NeuroEvolution of Augmenting Topologies(NEAT). NEAT is able to evolve not only the weights and the biases of a neural network, but also the topologies of a neural network, which can incrementally grow from the minimal structure of a neural network[25]. Gomes et al. applied NEAT to cooperative coevolution for a real multi-robot system[7]. They also applied novelty search[12], and the score of individuals depends on both performance and behaviour novelty. Both NEAT and novelty search can be transferred to real robots and successfully catch the fixed strategy prey.

2.6 From Simulation to Real World

Gazebo is a 3D real-time simulation physics engine, robots or objects can be described by URDF or SDF file. The full name of URDF is called Unified Robot Description Format, URDF is used for describing kinematic and dynamic properties of a robot, also the relative position between joints or links, in other words, URDF focuses on the robot level. On the other hand, the SDF can describe not only the robot itself, but also describe the properties of the world, including the pose of robots, the source of light, the friction of the floor...etc. There are many works show that evolved robots can be transferred to the real world[15]. The work from Stefano Nolfi Domenico Parisi evolved the Khepera robot such that it can locate recognize and grasp an object in simulation, and then transfer the controller into the real world[19]. Although the difference between the simulation world and the real world causes the difficulty of directly transferring, which is called ”reality gap”, but the robots are still able to keep a certain degree of functionality. So we trained the controllers of the robots in the simulation world first, after that, we transfer the better performance controllers into the real world to evaluate their performance or further evolve the controllers again.

2.7 Summary of Preliminaries

After the review, we know that from the environment perspective, it can be divided into fully or partially observable to robots, from the type of controller, we can use a homogeneous controller for all predators or

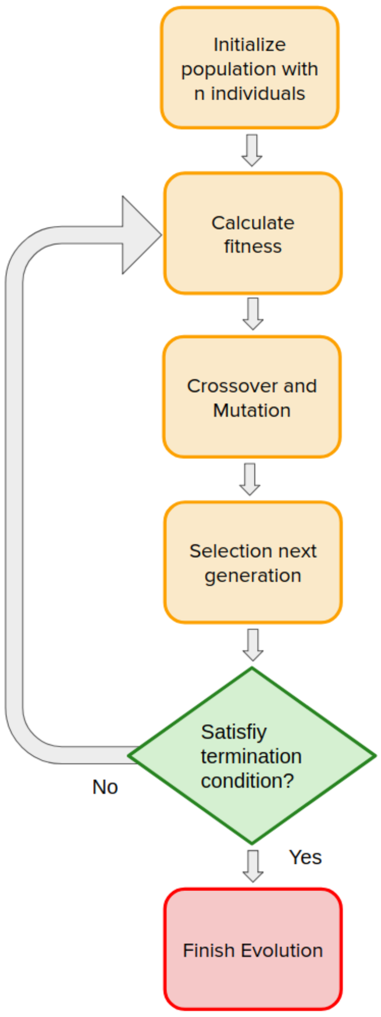

Initialize population with n individuals
Calculate fitness
Crossover and Mutation
Selection next generation
Satisfiy termination condition?
No
Yes
Finish Evolution

Figure 2.2: Standard process of evolutionary algorithm

multiple heterogeneous controllers for each predator. For evolving predators, we can use a fixed strategy for the prey or competitive coevolution. In the first stage, we started from the options with lower complexity, which includes using a homogeneous controller, a fully observable environment, and a fixed strategy for prey. In the second stage, we used the opposite setting, including heterogeneous controllers, a partially observable environment, and competitive coevolution between the predators and the prey. The experimental settings for two different stages are listed in Table 2.1.

	First Stage	Second Stage
Sensor	An camera above to get the position and the direction of each agent	For predators, using sensors of a robot itself.
Predator controller	Using a homogeneous controller for all predators	Using heterogeneous controllers for each predator.
Prey controller	Simple potential function(Fixed Strategy)	In simulation, using a co-evolved all-knowing prey to help the evolution of the predators. In the real world, replacing the prey controller with a human player.
Evolution process	Evolving one controller for all predators.	Co-evolving the predators and the prey alternatively.
Algorithm	ES, CMA-ES, Bayesian Optimization	NEAT

Table 2.1: Experimental settings for the two stages

Chapter 3

First Stage

3.1 Method

3.1.1 Robot

In our experiment, we use Thymio-II as both the predators and the prey. A Thymio has two wheels, and the speed and the direction of a Thymio can be controlled by setting the speed of the left wheel and the right wheel independently. The size of Thymio is around 11cm$\times$11cm. In the simulation world, we need a 3D model for Thymio, so that we can simulate the evolution process in the Gazebo. Blender is free and open 3D creation software, it was used to draw the 3D model that can be imported into Gazebo. Figure 3.1a shows Thymio in the real world and Figure 3.1b in Gazebo.

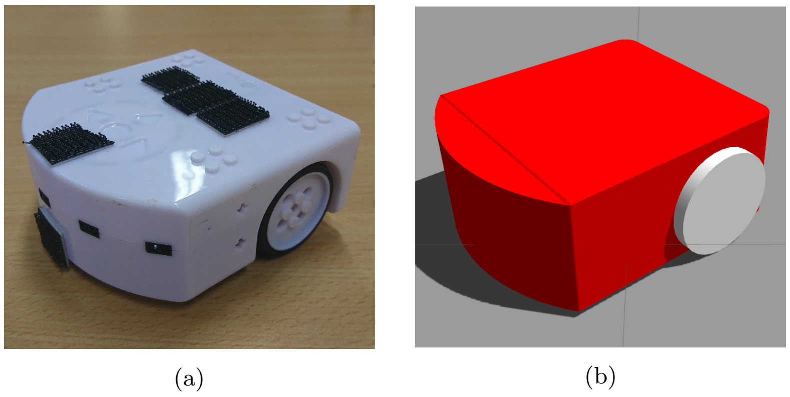


Figure 3.1: Appearance of Thymio in the real world and the simulation world.

3.1.2 The Simulation Environment

Our simulation environment is built on Gazebo, which is a well-designed 3D real-time robot simulator. The robots are able to interact with each other. The arena for pursuit and evasion is a 2m by 2m square world with walls, and there is no obstacle in the square world as shown in Figure 3.2a, where the green robot represents the prey and the three red robots are the predators. In the simulated world, we design a similar robot to the physical Thymio robots in the real world. To reduce the reality gap, appropriate physics parameters of the simulation are important. In Gazebo, the physics parameters can be defined by SDF files for robot and world. We executed the preliminary experiments and found that the upper bound of velocity is one of the important physics quantities in our task. The upper bound of velocity must be coherent in the simulated world and the real world. This can be achieved by measuring the top speed of the robot in the real world, and limit the wheel speed in the simulated world. There are other parameters that can be tuned, for example, the moment of inertia, but we found that appropriate values for the high-level physics quantities velocity and angular speed are enough for good simulations of our predator-prey task.

3.1.3 The Real World Environment

In the real world, we use the same robots as the simulation, a set of Thymios. To improve the computing power and the communication between robots we add a Raspberry Pi (that can handle wifi) and an extra

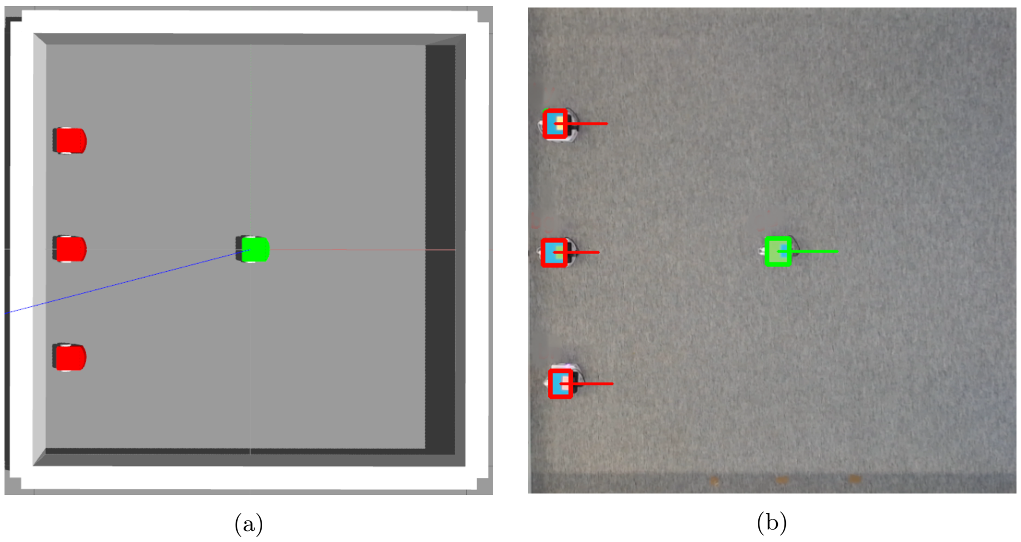


Figure 3.2: Three predators and a prey in the simulated (a) and the real (b) world. In (a), the red robots are predators, the green robot is prey. (b) is the overhead photo of the real world. The bounding boxes with texts are the predators and prey that recognized by the overhead camera.

battery. The field is a 2m×2m square arena as in the simulated world. Instead of using the cameras on the robots, we choose to use an overhead camera above the arena to provide information for the robot controllers. For localization in the real world, we develop a color recognition system where the robots are designated by different colors, such that it is easy to recognize the position and direction of robots. We achieve this by placing colored papers on the top of the robots and a smaller colored square that indicates the heads (i.e., directions). We also put 4 pieces orange square paper at the four the corners as calibration to locate the field and origin point. The real world setting can be seen in Figure 1b. The bounding boxes indicate robots and the lines indicate the direction of robots.

3.1.4 Fixed Strategy for Prey

As for the prey, we deliberately use a fixed evasion strategy. To develop this we observed that when predators are close to the prey, the situation becomes more dangerous for the prey. Also, when the prey stays near the wall it can be trapped easily. Thus, the main idea behind our prey controller is to model danger zones with multiple Gaussian functions. Specifically, we use a 2D Gaussian function to model the danger zone around the predators and a 1D Gaussian function to model the danger zone close to the walls. The predator-induced danger zone distribution can be expressed as:

$$\mathcal{P}_k(x,y) = \frac{1}{2\pi\sigma_p^2} exp\left\{ - \frac{(x-x_k)^2}{2\sigma_p^2} - \frac{(y-y_k)^2}{2\sigma_p^2} \right\} \tag{3.1}$$

where x_P, y_P are the coordinates of the predator, and σ_P is a hyperparameter. $P_k(x, y)$ returns higher values for those coordinates that are close to predators, in other words, those coordinates are more dangerous. When σ_P is low, the danger zone induced by predators is relatively narrow. This makes the prey “feel in danger” only when it is relatively close to the predators.
Our experimental arena is a square world with two vertical walls and two horizontal walls. The danger zone distribution belonging to a vertical wall can be expressed as:

$$\mathcal{W}_i(x) = \frac{1}{\sigma_w\sqrt{2\pi}} exp\left\{ - \frac{(x-x_i)^2}{2\sigma_w^2} \right\}, \quad i \in \{1,2\} \tag{3.2}$$

where i is the index of the two vertical walls. x_i is the coordinate of the two vertical walls in horizontal axis (x). The danger zone distribution belonging to a horizontal wall is:

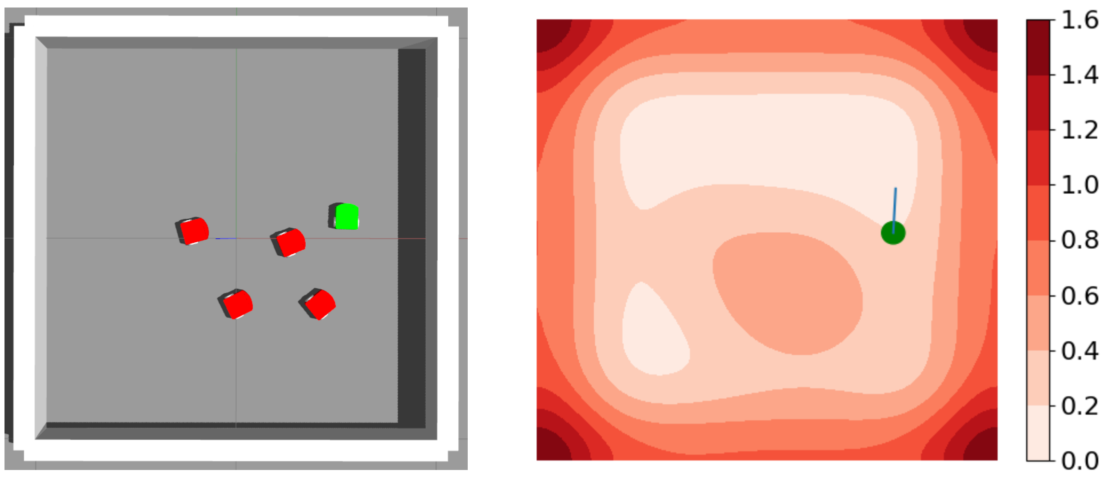
$$\mathcal{W}_j(y) = \frac{1}{\sigma_w\sqrt{2\pi}} exp\left\{ - \frac{(y-y_j)^2}{2\sigma_w^2} \right\}, \quad j \in \{1,2\} \tag{3.3}$$

where j is the index of the two horizontal walls, y_j is the coordinates of the two horizontal walls in vertical axis (y). This model returns higher values besides the wall, because it is easier to be trapped there. When σ_W

is lower, the danger zone caused by walls is relatively narrow. This makes the prey “feel in danger” only when it is relatively close to the wall.
Combining all the danger zone distributions above, the final danger zone distribution is:

$$d(x,y) = \sum_{i=1}^{2} w_i(x) + \sum_{j=1}^{2} w_j(y) + \alpha \sum_{k=1}^{\mathcal{N}_p} \mathcal{P}_k(x,y) \tag{3.4}$$

where the coefficient α determines the relative importance of avoiding predators and avoiding being trapped nearby walls, $\mathcal{N}_p$ is the number of predators. For instance, the final danger zone distribution for a random situation with four predators is shown in Figure 3.3. The value of the heat map represents the level of danger, higher values means the more dangerous locations for the prey on the map.



(a) (b)

Figure 3.3: The robots distribution and corresponding heat map of the danger level. (a) represents the positions of the robots in the simulated world. (b) is the corresponding danger level that displayed in heat map. The green point and line represent the prey and its direction towards the relative safe location.

With the danger zone map, the prey has the navigation towards less dangerous places to avoid the predators. This can be achieved by moving towards the direction of the descending gradient at the coordinates of the prey. The direction of the descending gradient can be calculated analytically. The navigation can be applied in a real-time physics simulation and also in the real world. The escaping direction vector $\mathcal{D}$ can be derived as the following:

$$\begin{aligned} g(x) &= \frac{\partial d(x,y)}{\partial x} \\ &= \sum_{i=1}^{2} \frac{d\mathcal{W}_i(x)}{dx} + \alpha \sum_{k=1}^{\mathcal{N}_p} \frac{\partial \mathcal{P}_k(x,y)}{\partial x} \\ &= \sum_{i=1}^{2} \frac{1}{\sigma_w \sqrt{2\pi}} exp\left\{ -\frac{(x-x_i)^2}{2\sigma_w^2} \right\} \cdot (-\frac{x-x_i}{\sigma_w^2}) + \\ &\sum_{k=1}^{\mathcal{N}_p} \frac{1}{2\pi\sigma_p^2} exp\left\{ -\frac{(x-x_k)^2}{2\sigma_p^2} - \frac{(y-y_k)^2}{2\sigma_p^2} \right\} \cdot (-\frac{x-x_p}{\sigma_p^2}) \end{aligned} \tag{3.5}$$

$$\begin{aligned} g(y) &= \frac{\partial d(x,y)}{\partial y} \\ &= \sum_{j=1}^{2} \frac{dw_j(y)}{dy} + \alpha \sum_{k=1}^{\mathcal{N}_p} \frac{\partial \mathcal{P}_k(x,y)}{\partial y} \\ &= \sum_{j=1}^{2} \frac{1}{\sigma_w \sqrt{2\pi}} exp\left\{ -\frac{(y-y_{W_j})^2}{2\sigma_w^2} \right\} \cdot (-\frac{y-y_{W_j}}{\sigma_w^2}) + \\ &\sum_{k=1}^{\mathcal{N}_p} \frac{1}{2\pi\sigma_p^2} exp\left\{ -\frac{(x-x_{\mathcal{P}_k})^2}{2\sigma_p^2} - \frac{(y-y_{\mathcal{P}_k})^2}{2\sigma_p^2} \right\} \cdot (-\frac{y-y_p}{\sigma_p^2}) \end{aligned} \tag{3.6}$$

$$\mathcal{D} = -\left(\frac{g(x)}{\sqrt{g(x)^2 + g(y)^2}}, \frac{g(x)}{\sqrt{g(x)^2 + g(y)^2}}\right) \tag{3.7}$$

The escaping direction $\mathcal{D}$ towards the less dangerous locations can be calculated real-time in the simulated world and real world. The prey strategy is to move forward when it matches the escape direction (with some tolerance), otherwise the prey turns left or right until the escape direction is matched.

3.1.5 Homogeneous controller for Predators

To evolve good predator strategies we use neural networks as controllers in the predators, and optimize the parameters of the neural network with an evolutionary algorithm. We choose to use a homogeneous team of predators that work with identical controllers. The reason is scalability; with such a system we can vary the number of predators easily and the optimal number of predators can be determined by the difficulty of task or cost of hardware.

To keep things simple we use neural networks with one hidden layer and two outputs nodes that drive the two wheels. As for the inputs we identified three features to be used, one for avoiding collisions with other predators and two for chasing the prey. The first input is the inverse of the distance from the predator to the nearest predator, and when the nearest predator is on the left, the first input will be change as a negative value. The second input is the angle between the orientation of predator and the direction of prey relative to the predator and the third input is the distance between the predator itself and the prey. Let us note that these features can be used for any number of predators easily, because the inputs are independent of the number of predators. We have only 4 hidden neurons and 2 outputs as left and right wheel angular velocity. Both the hidden and the output layers use hyperbolic tangent as the activation function. The following list and Figure 3.4 show the details of the predator controllers more clearly.

Input Layer:

- r: The inverse of the distance between predator itself and nearest predator. If the nearest is on its left, then change it as a negative value.
- $\Delta\theta$: The angle difference between the orientation of predator and the direction of prey with relative to the predator.
- d: The distance between predator itself and prey.

Hidden Layer: 4 neurons.
Hyperbolic tangent as the activation function.
Output Layer:

- ω_L: Angular velocity of the left wheel
- ω_R: Angular velocity of the right wheel

Hyperbolic tangent as the activation function.

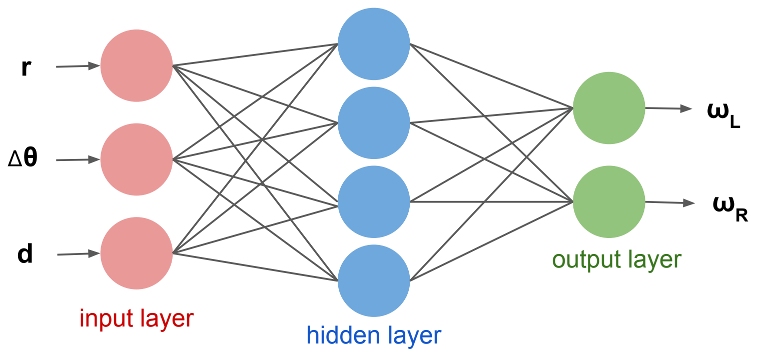


Figure 3.4: The structure of the neural network for predator controller. $r, \Delta\Theta, d$ are the inputs. ω_L, ω_R are the outputs.

3.1.6 Evolutionary algorithm

The Fitness Function

The predators are homogeneous agents, i.e., the predators have the same controller during an evaluation. The controllers in different evaluations are evaluated by fitness function. In our experiments, the predators and the prey are reset to the same starting positions for each evaluation. At the time t of the end of an evaluation time T, we can calculate the following measurements:

- r_{it}: is the distance between i^{th} predator itself and the nearest predator.
- d_{it}: is the distance between i^{th} predator and the prey.

where i is the index of predators. Considering a situation for two different controllers, the first one captured the prey at the end of evaluation time. The second one captured the prey at half of the evaluation time and followed the prey closely at the rest half of the evaluation time. Obviously, we expect the second predator controller has higher fitness than the first one. We expect the predators to capture the prey as soon as possible, and follow the prey to keep being close during the evaluation time. Therefore, we evaluate the predator controller for the whole performance during the evaluation time. We can calculate the average distance from the predator to the prey as $\frac{1}{T}\sum_{t=0.2}^{T} d_{it}$, and the average distance from the predator itself to the nearest predator as $\frac{1}{T}\sum_{t=0.2}^{T} r_{it}$, where T is the evaluation time (60 seconds in this work) and the time interval between different time t is 0.2 second. Furthermore, we expect the predator controller can drive the predators not one of them to capture the prey during an evaluation. we therefore evaluate the predator controller by considering the performances of all the predators. The fitness function can be expressed as:

$$fitness = \frac{1}{\mathcal{N}_p}\sum_{i=1}^{\mathcal{N}_p}\left[\frac{1}{\frac{1}{T}\sum_{t=0.2}^{T} d_{it}}\right] + \frac{1}{\mathcal{N}_p}\sum_{i=1}^{\mathcal{N}_p}\frac{1}{T}\sum_{t=0.2}^{T} r_{it} \tag{3.8}$$

where $\mathcal{N}_p$ is the number of the predators.

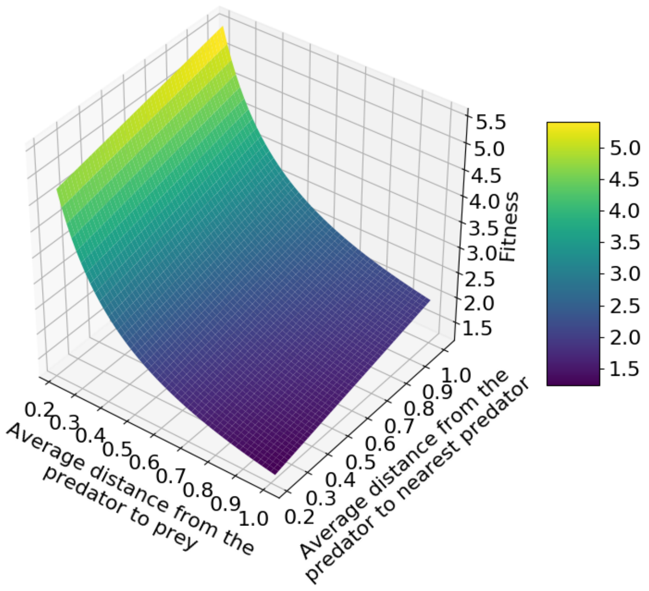


Figure 3.5: Visualization of the fitness function in Equation 3.8. We slightly encourage the controller to increase the distance between predators for collision avoidance. Minimizing the distance between the prey and the predator is more important.

Evolution Framework

To select a good optimizer we compare three algorithms, an evolution strategy (OpenAI), the CMA-ES and Bayesian optimization using the libraries from Github [16, 17, 5] and the recommended parameter values, e.g., from [9]. We evolve 100 generations with population size 13 with all algorithms and repeat this 10 times with different random seeds. The outcomes of these preliminary comparative experiments are shown in Figure 3.7. The Bayesian Optimization is able to get high fitness at an early stage, but CMA-ES surpasses Bayesian Optimization after around 30th generation. Comparing the CPU time, CMA-ES also outperforms other

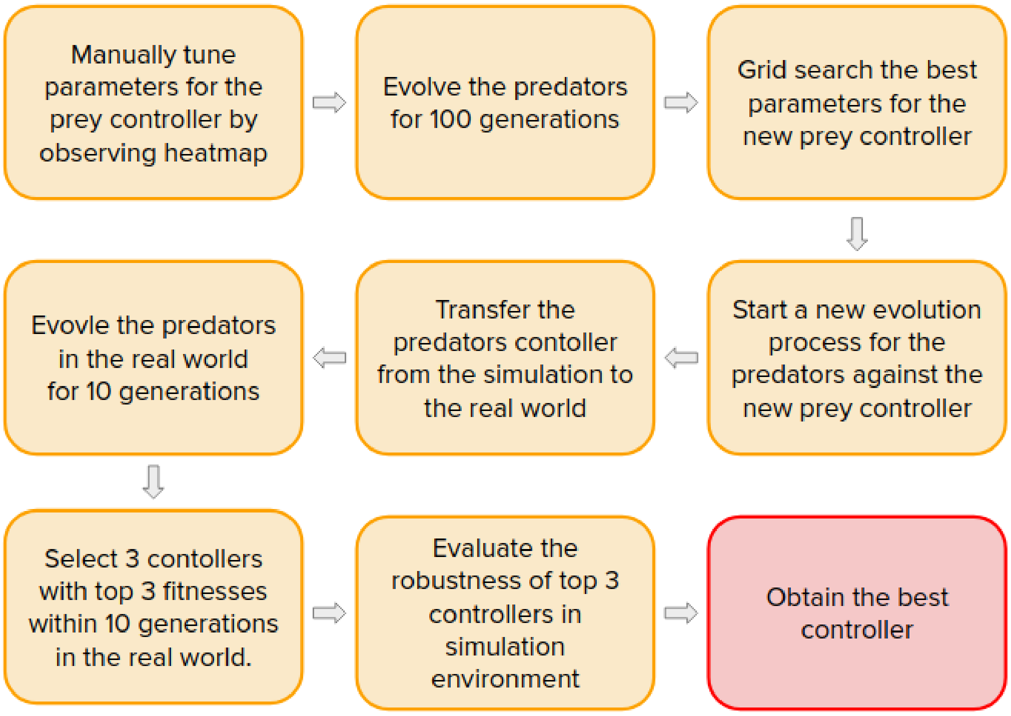

Manually tune parameters for the prey controller by observing heatmap
Evolve the predators for 100 generations
Grid search the best parameters for the new prey controller
Start a new evolution process for the predators against the new prey controller
Transfer the predators contoller from the simulation to the real world
Evovle the predators in the real world for 10 generations
Select 3 contollers with top 3 fitnesses within 10 generations in the real world.
Evaluate the robustness of top 3 controllers in simulation environment
Obtain the best controller

Figure 3.6: Flow chart for evolving and selecting the best controller

algorithms as shown in Figure 3.8. Although we have not performed an extensive parameter optimization and a rigid statistical comparison, the results are sufficiently different to select the CMA-ES.

In the fixed strategy for prey, we introduced three parameters σ_W, σ_P and α that determine the danger zone and the escape direction. Thus, these parameters have a crucial impact on the behaviour of the prey and to make the prey really hard to catch, their values should be optimized. In order to do this, we carried out an evolutionary process to obtain good weights for the neural network controller for the predators with 100 generation, and then we did a grid search for the prey strategy on possible values for σ_W, σ_P and α against the evolved predators. The best parameters from this comparison are shown in Table 3.1. The small alpha implies that the prey must keep the distance from the walls and the corners, in case of being surrounded by predators. If necessary, it is possible to iterate the whole process, that is, train the predators with a prey and optimize the prey parameters with the best predators repeatedly, until σ_w, σ_P and α converge or the performance gains diminish. We chose to do the grid search once, and started a new evolution process for the predators.

parameter	σ_w	σ_p	α
value	0.2	0.25	0.1

Table 3.1: The value of parameters we used in the Gaussian model-based prey escaping strategy

parameter	value
Number of generation	100
Population size	13
Sigma	0.4
Learning rate	0.2
Decay	0.98

Table 3.2: Main parameters of ES

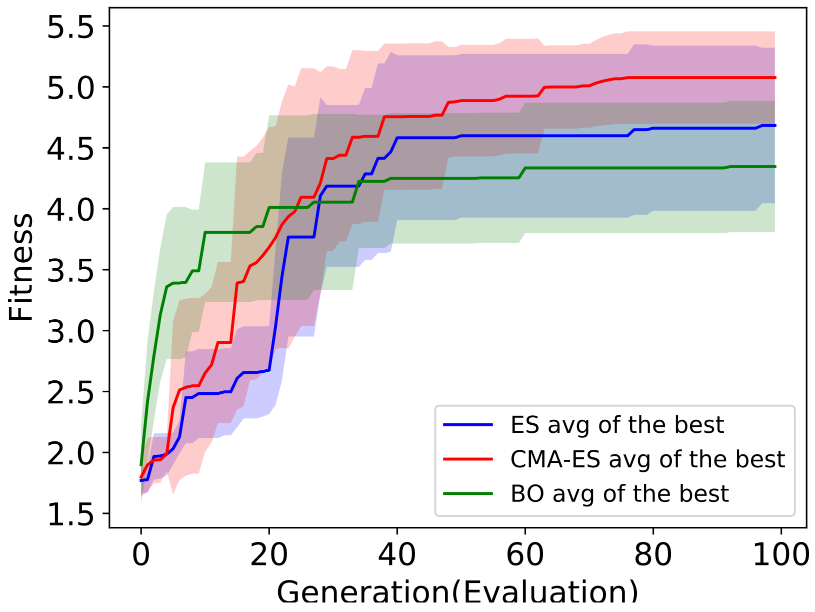


Figure 3.7: The average of the best fitness of the three algorithms over generations in 10 runs. The blue, red, green represent the average of Evolutionary Strategy (ES), CMA-ES, Bayesian Optimization (BO) respectively.

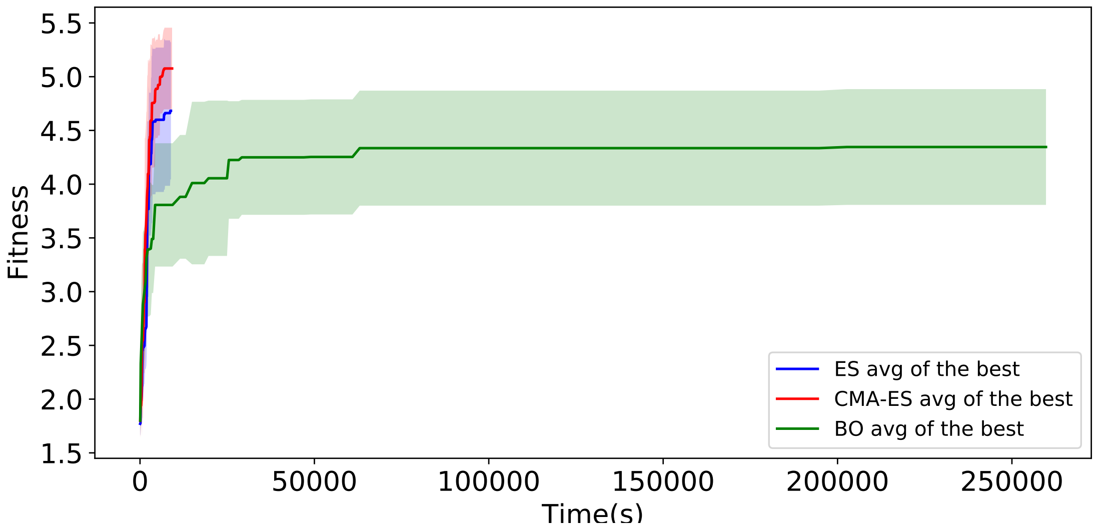


Figure 3.8: The blue, red, green represent the average fitness to evolution time of the best controllers from Evolutionary Strategy (ES), CMA-ES, Bayesian Optimization (BO) respectively. The evolution time for ES is 8865.05s, CMA-ES is 9196.11s, and BO is 259770.85s.

parameter	value
Number of generation	100
Population size	13
Sigma	0.5

Table 3.3: Main parameters of CMA-ES
Population size is computed by default value $4 + int(3 * np.log(N))$ from python CMA-ES package. N is the dimension of problem.

parameter	value
Number of evaluation	1300
ξ	0.01
κ	1.0
Kernel	Matern
μ of kernel	2.5
α	10^{-5}
Length scale	0.05
Acquisition function	ucb

Table 3.4: Main parameters of BO

3.2 Experiments

3.2.1 Evolution in simulation

In the beginning of an episode, the prey is placed at the center of the square. 3 predators are placed parallel to each other and perpendicularly to one of the edges, and then set off to chase the prey as shown in Figure 3.2a. One episode is 60 seconds. The episode is not stopped before 60 seconds even if the prey is trapped, because it is possible that the prey will escape again and the predators must turn around to chase the prey. Clearly, we cannot say that the predators are good if the prey has the chance to slip away after being caught.

Fitness and behaviour analysis

As outlined in [4] the fitness functions in ER are more complex than in other areas of evolutionary optimization. In particular, a good fitness function must reflect desirable *behaviour*, rather than “just” a given objective function. This makes an analysis of fitness and behaviour advisable to verify that higher fitness indeed belongs to better behaviour. In our study, the question is: How do the fitness values given by Equation 3.8 represent the performance of predators?

To answer this question we collect all 1300 controllers generated during an evolutionary run, rank them by fitness and inspect the behaviour of controllers with fitness around 2.0, 3.0, 4.0 and the overall best fitness of 5.38 by plotting the average distance of the predators to the prey during the evaluation period, the 60 seconds episode. The results are shown in Figure 3.9. The fitness of the red controller is 5.38, the average distance from predators to prey is around 0.13m at the end of the episode. Considering that the width of a Thymio itself is 0.11m and we have multiple predators, 0.13m is small enough to indicate that the predators indeed caught the prey. For the orange line with fitness 3.03, it was a classic battle between predators and prey that was almost captured, but could get away twice.

The trajectories of these four controllers can be seen in Figure 3.10. The predators can only follow the trail of prey with fitness 2.04. When fitness is 3.03, predators use different skill to chase the prey more than just follow, they even can change team formation alternatively, in other words, predators are able to alternatively become the center of the team formation. This fact is also shown when fitness = 4.02. In the beginning, the center predator moved to the left side of the pack and the predator on the top moved to the center. This is an interesting behaviour that we did not see during our preparatory experiments with just two input nodes $\Delta\theta$ and d. It only occurred after we added the third input r related to the distance between predators.

In some circumstances, there might be an optimal number of predators to catch the prey. Predators intend to move to prey, but at the same time, they must avoid hitting each other. A skillful prey might lure predators into a position which is easy to get stuck. Therefore we can examine if there is an optimal number of predators to catch the prey. Of course, we cannot put too many predators into the field. Considering the real world setup, we assume that only a limited number of predators can be used. To see if our system is workable when we need to increase the number of predators we compare 3, 4 and 5 predators in a new set of experiments. We select the best controllers from 3, 4 and 5 predators from these experiments plot their average distance to the prey in Figure 3.11. These curves show that 5 predators are a bit too many. The teams of 3 or 4 predators are equally successful, but 4 predators catch the prey faster and with lower variance in the beginning. Yet, we choose to use 3 predators in the real world, because 3 robots are easier to work with and the performance after approximately 20 seconds is the same as for a team of 4.

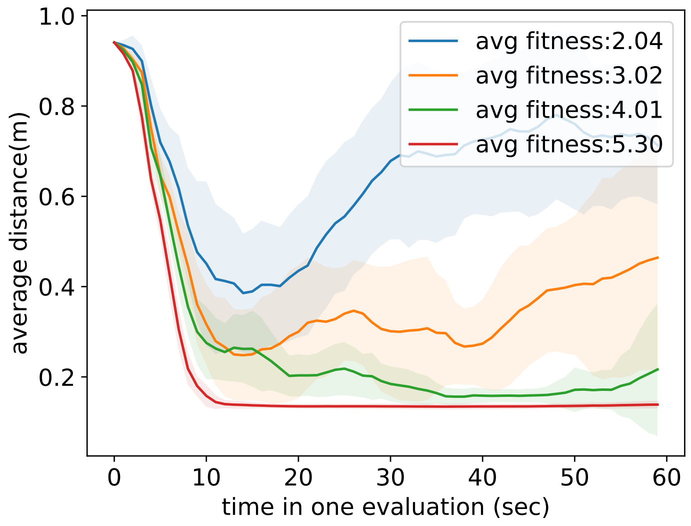


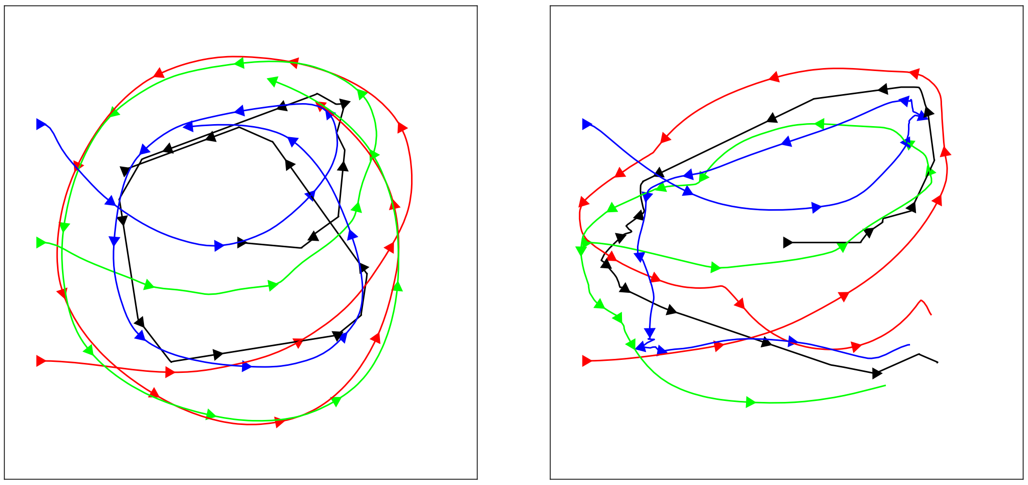
Figure 3.9: The average distance (from predator to prey) in runs of 4 controllers with different finesses from high to low.

3.2.2 Evolution in the real world

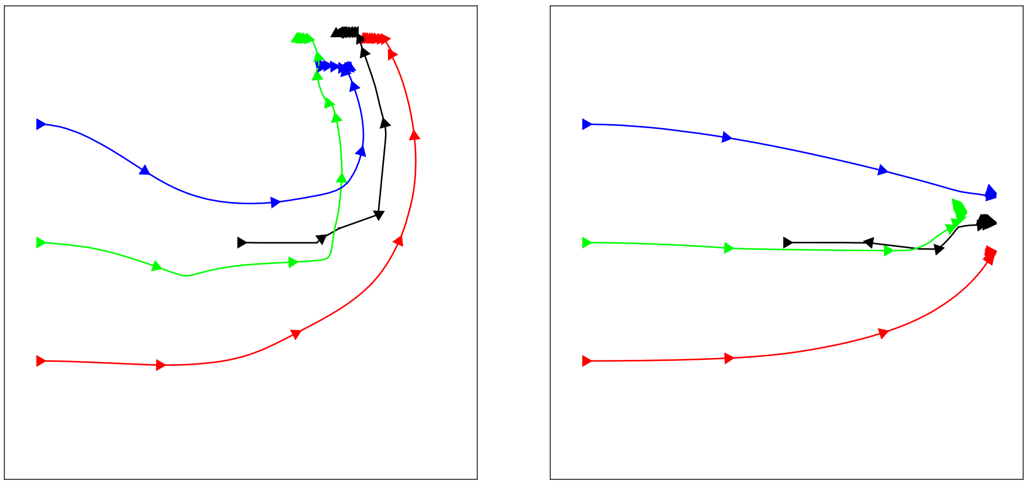
The real world setting has the same 2m by 2m square field as the simulated world, and we use three Thymios as the predators and one as the prey. To extract the features as the inputs of controllers of predators and prey, an overhead camera above the field collects the coordinates, orientation and unique identification of robots. To seed the initial population in the real world setting we use the last generation of the simulated evolutionary process. Figure 3.12 shows a clear reality gap between generation 100 and 101. The average fitness of the same population drops from 2.94 to 2.12, while the maximum fitness drops from 3.77 to 2.48. After that, we further evolved our predators 10 generations in the real world. The average of fitness slowly grows and the average fitness in the last generation reaches 2.87, with 4.55 as maximum fitness. Although, it might come from the noise of the experiment, for example, the delay of packet transmission, or the different physics parameters between the simulated world and the real world. Also, the generation 100 may not include the controller with the best fitness of the simulated world. In the real world, the robots get stuck more easily because of the wires used to connect the extra battery. This can be an advantage for the prey because predators can get stuck with each other, but in other cases the predators can immobilize the prey in this way. Figure 3.13 shows the trajectories of the best three controllers evolved in the real world.

3.2.3 Robustness tests in simulation

To verify that the trained model is able to catch the prey in various situations, we can set random initial positions and play 1000 episodes to examine the robustness by plotting the fitness distribution and keeping the hit scores, i.e., the number of times (out of 1000) the prey was really caught. Obviously, this is much easier and cheaper in the simulated world. In the end, we can select the controller with the best robustness from the best 3 controllers evolved in the real world. Figure 3.14 shows the fact that the controller with higher fitness in the real world is not necessary with strong robustness. The red histogram represents the controller with the highest fitness, the green histogram is for the second highest fitness and the blue histogram is for the third highest fitness. (We will call them the red controller, the green controller, and the blue controller.) The median and 25th percentile of the red histogram is much smaller than the blue histogram. The red histogram has more polarization distribution and a large number of episodes ending with fitness values between 2.6 and 2.8. The reason why it concentrates between 2.6 and 2.8 is that the red controller is relatively slow, once the prey escapes from predators, the predators can only follow behind the prey, but the red controller is good at keeping a spread team formation, which makes the controller able to surround prey in some circumstances. The green controller is quite the opposite of the red controller, it has the high speed but more casual team formation, which makes predators can re-catch prey when a caught prey escapes. The blue controller combines advantages from the two other controllers and therefore we pick the blue controller as our champion with both high fitness and better robustness.



(a) Fitness = 2.04 (b) Fitness = 3.03



(c) Fitness = 4.02 (d) Best fitness = 5.38

Figure 3.10: Trajectories of 4 controllers with different fitness values that presented in Figure 3.9. The black dashed line represents the trajectory of the prey. The red, green, blue solid lines represent the trajectories of three predators.

3.3 Discussion and Limitations

3.3.1 Prey Behaviour

During the evolution, the prey sometimes show the skillful behaviour to escape from the predators. For example, when the prey was surrounded by three predators at the corner, the prey had the chance to slip between the gap among the predators. The heat map in Figure 3.15 shows the process of how the prey escaped from a corner.
The upper bound of the predators' performance depends on the ability of the prey. Therefore, a fixed strategy for prey means there is a fixed upper bound of the predators' performance. In other words, if we want to improve the performance of the predators, we must design a delicate or complicated prey controller as the opponent to stimulate the potential of the predators. Another option can be applying coevolution to develop strategies for both predators and prey as what we are going to do in the second stage.

3.3.2 Predators Behaviour

Collective Behavior

In the swarm intelligence, people can create simple rules to achieve collective behaviours. During the evolution, we observed that the predators showed collective behaviors like moving in formation and collision avoidance. It happened after we added the distance to the closest team member as the input of the controller.

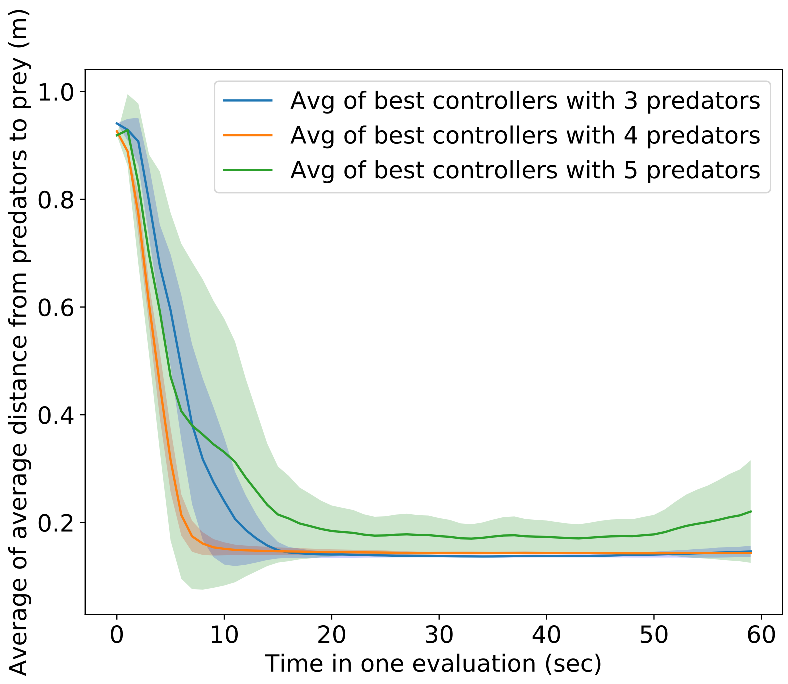


Figure 3.11: Average distance from predators to prey during an evaluation in the situations of 3, 4 and 5 predators in 10 runs.

Because when the collision happens between predators, it interfered themselves to catch the prey. On the contrary, when the predators keep far away distance from each other, it's hard to surround the prey. Therefore, the optimal distance should lay in somewhere between these two extreme conditions. Consequently, the predators move in a formation to catch the prey together. This result shows the potential of homogeneous controller to be evolved as a collective behaviour controller.

We perform another new experiment. The wall is removed and the prey is controlled by human to guide the moving direction of the predators. The prey was accelerated to avoid being caught. Figure 3.16 shows the steps of the prey was guiding the predators, the predators followed the prey and moved in formation without colliding to each other.

3.3.3 Limitation of the experimental setting

In the first stage, an overhead camera is used to get the pose of the predators and the prey. It's a strict limitation, especially for an outdoor environment, however, it shows that the information about the pose of robots is enough for the predators to catch the prey. The functionality of the camera can be replaced with GPS, compass or SLAM(Simultaneous localization and mapping) to locate the pose of the robots. In the work from Gomes et al. [7], they used GPS and compass in a noisy outdoor environment for a prey-capture task. Except for selecting the pose of robots as inputs of controller, we can also choose the data from the sensors of the robots, without using any external equipment to evolve the robots, which is exactly what we do in the second stage.

3.3.4 Homogeneous Controller and Heterogeneous Controllers

In the first stage, we evolve one controller for all the predators. The predators showed some collective behavior like having team formation and collision avoidance as the work from Quinn et al. [14]. Baldassarre et al. also studied the collective behaviour of evolving mobile robots[2]. The relatively small population size is 13 and 100 generations can be evolved within only 30 minutes with the acceleration in the simulation world. Another advantage of homogeneous controller is the scalability of the number of the predators, when we increase the number of the predators, the structure of controller and training time can remain the same. These advantages make the homogeneous controller especially suitable for the low-cost robots, we can controller and coordinate numerous robots with a controller. However, the homogeneous controller sometimes merely follow the tail of the prey and fail to catch the prey. It shows the homogeneous controller has the problem of lacking diversity. In the second stage, we will try to evolve three predators with a heterogeneous controller.

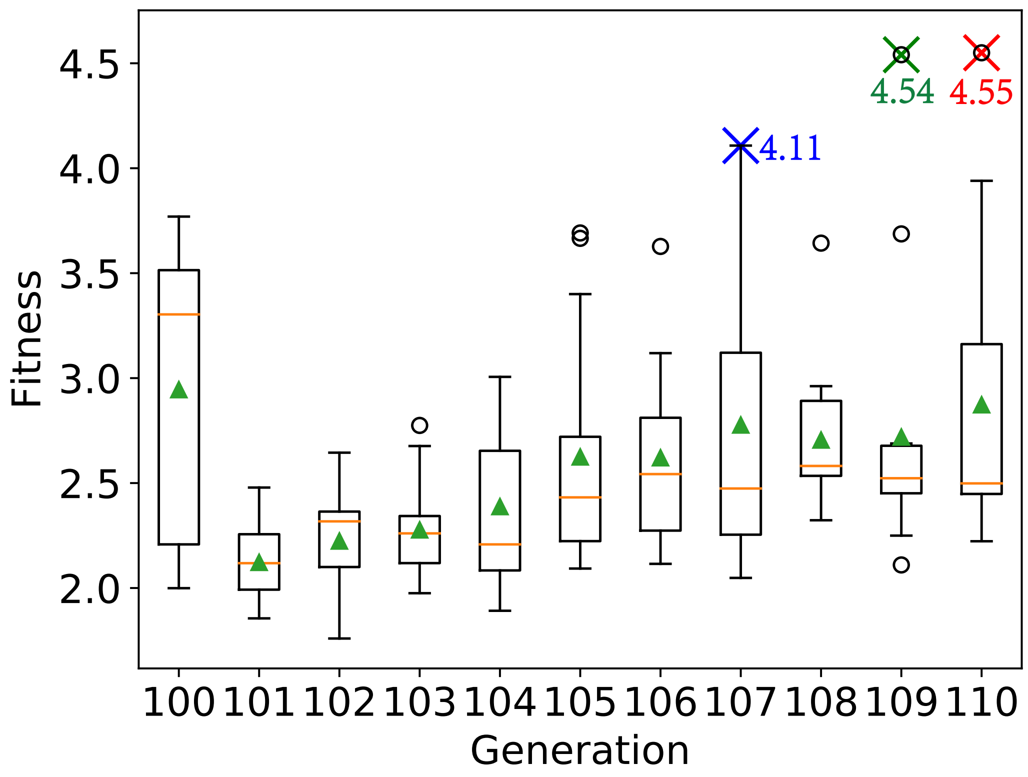
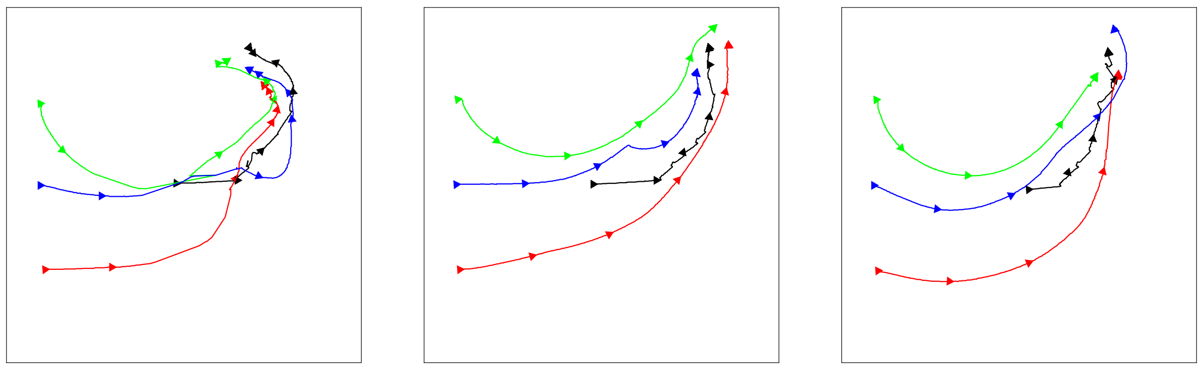


Figure 3.12: The fitness of post transfer learning in the real world. The fitness in generation 100 is obtained from the simulation. The blue, green, and red crosses are the top three controllers with fitness 4.55, 4.54, 4.11 during the post transfer learning in the real world. The green triangles, the orange lines, and the black circles represent average fitness, median fitness, and outliers in a generation respectively.



(a) Fitness = 4.55 (b) Fitness = 4.54 (c) Fitness = 4.11

Figure 3.13: Trajectories of the top three controllers that evolved during further learning in the real world. The black dashed line represents the trajectory of the prey. The red, green, blue solid lines represent the trajectories of three predators.

3.3.5 Evolution Time

The difference between evolution in simulation and evolution in the real world can be significant considering solution quality and execution time. As exhibited in Figure 3.12, the fitness drops by approximately 30% when we switch from simulation to real robots (average fitness: 28%, maximum fitness: 34%).

The differences in execution times are even greater. In stage 1 of our system, evolution in the simulated world, it took about 2.5 hours to run 100 generations (with population size 13) on an Intel Core i5-5200U CPU @ 2.20GHz 4. The evolution in the real world took about 3 hours to execute 10 generations with the same population size. Thus, evolution in the real world for 100 generations would take about 30 hours to complete – approximately 12 times slower. Execution times are of course subject to many practical details, such as the computers clock speed, the number of cores, simulation accuracy, reset time of the physical robots, etc., but these numbers give an indication of the time vs. quality trade-off inherent to evolutionary robotics

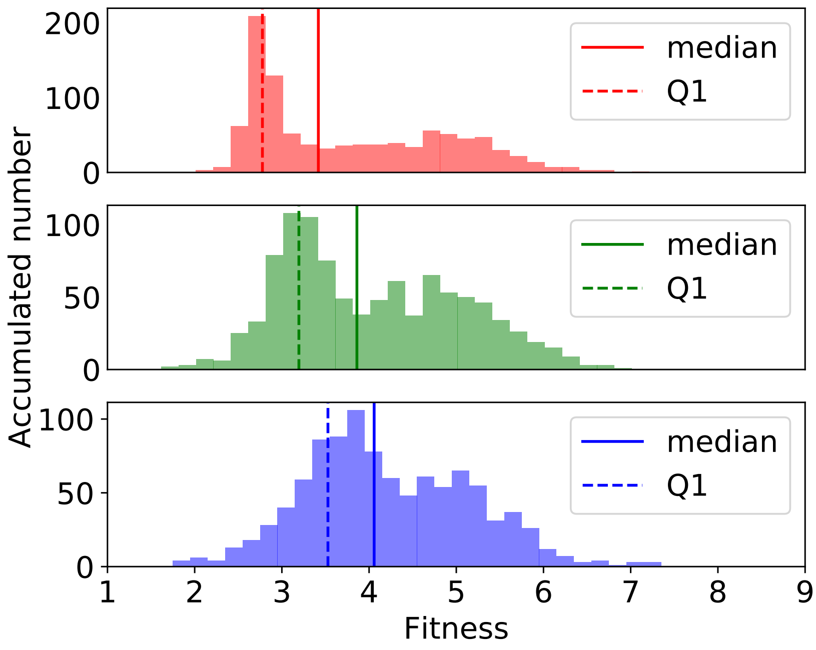


Figure 3.14: Robustness histograms of the best three controllers evolved on real robots. The colors red, green, blue denote the controllers with fitness 4.55, 4.54, 4.11 respectively as identified in Figure 3.12.

experiments.

3.3.6 Comparison of the Different Algorithms

Another influential factor is the presence of noise. As shown in Figure 3.7, the CMA-ES performs better than Bayesian Optimization and ES. This fact might be (partly) attributed to the capability of CMA-ES to handle the noisy fitness function[8]. Recent work from Boeing and Bräunl [3] lists various error sources that one should be aware of when using complex real-time physics simulation systems. The amount of simulation noise is not constant, for example, it will depend on the acceleration factor (x times faster than the real world time clock) chosen by the user. Nevertheless, the ability of an EA to cope with noisy fitness functions is an important aspect.

3.3.7 Selecting Features

With selecting the appropriate feature about the distance between predator and predator, we can achieve collision avoidance with only one input. Increasing information entropy in each input makes us able to decrease the dimensions of neural network such that the training can be faster, even standard Bayesian Optimization can be applied. Although compared to CMA-ES and ES, Bayesian Optimization needs quite long time(more than 10 hours with core i5) to finish 100 generations. In our experience, select inputs as "the distance from predator to prey" and "the angle between predator's orientation and direction from predator to prey" have higher training efficiency than using the coordinate offset between predator and prey. Also, the inputs of the controller are independent of the number of predators, which makes our controller has the potential to manipulate numerous predators.

3.3.8 Number of predators

To determine an effective number of predators, we must consider the balance between catching and collision avoidance. At least among 3, 4 and 5 predators, selecting 4 predators has the most elegant balance. We also try to compete our predators with human players, and most of the time the human players are hard to beat the predators in the simulation world, although it's another challenge to quantify the performance of human players.

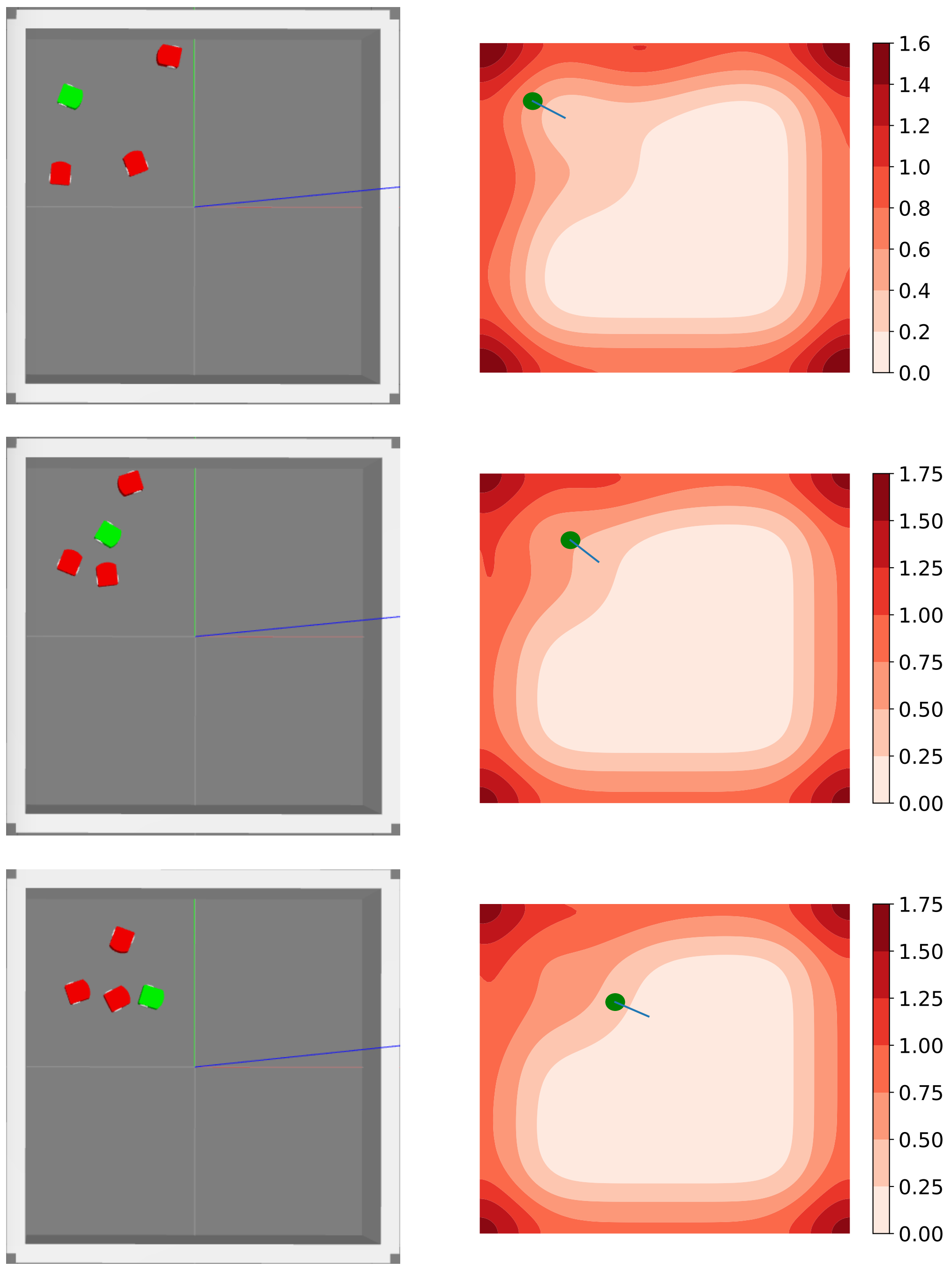


Figure 3.15: The prey escapes from the corner

3.4 Conclusion

we presented an evolutionary predator-prey system to develop pursuit strategies for a group of predator robots to capture a prey. This system is generic, i.e., applicable to various types of robots and it reaches from simulation to the real world. Specifically, the evolution of predator strategies takes place in two stages, the first stage in computer simulations followed by a second stage where evolution is executed on real robots.

To validate this system design we experimented with a group Thymio II robots. To obtain a good evolutionary system we chose to use the CMA-ES and defined a specific fitness function that considers the distance between a predator and the prey as well as the distance between this predator and its nearest fellow predator. By design, the controller of the prey was fixed and to provide a challenge for the predators.

The experiments provided feedback about the working of the system and illuminated its advantages and disadvantages. In line with our expectations, we found that the system was able to produce predators that

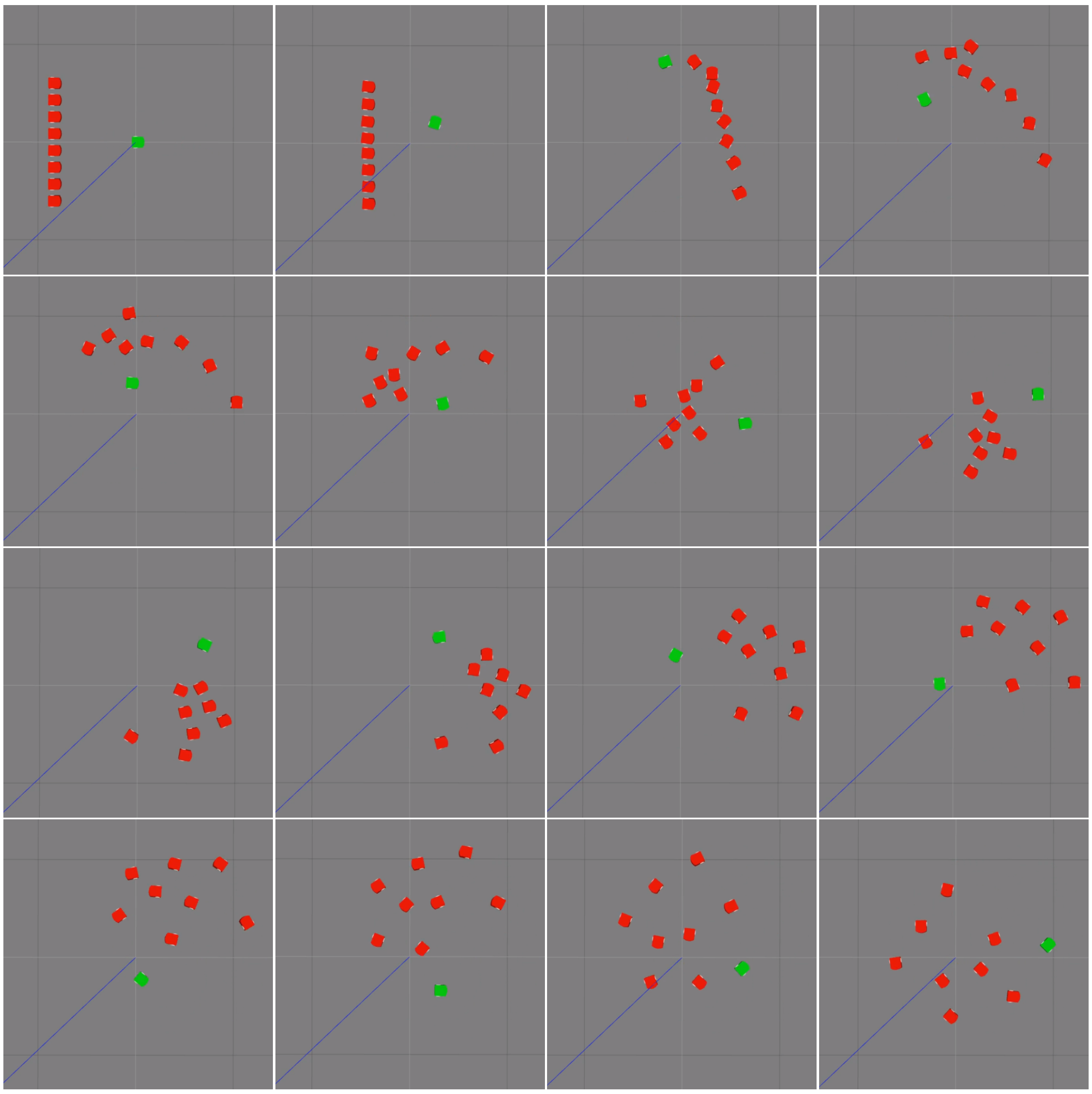


Figure 3.16: These images are ordered in time. They must be read from left to right and top to bottom. The prey is controlled by human, which can be used to guide the predators to make them move like a flock of birds without collision.

successfully captured the prey in the real world. Another advantage is the good integration between the simulated and the real world. Hereby we have the best controller of both worlds. Testing different algorithmic options is more practical in simulation, e.g., it is easy to vary the number of predators, try more or less sophisticated prey strategies, or compare several EA variants. In the meanwhile, running evolution on the real robots can mitigate the reality gap problem.

The main disadvantage we encountered was the time needed to carry out the hardware experiments. As noted above, these were 70 times slower than the simulations. Using bigger populations we expect that this ratio becomes even worse. On the positive side, 10 real world generations turned out to be enough to reach the fitness level achieved in 100 simulated generations. Thus, we can consider the simulations as a manner to kick-start evolution in hardware and to reduce the total time needed to evolve a solution that works in the real world. All in all, with our combined software-hardware evolutionary system, the know-how, and the algorithmic components in place it seems feasible to evolve predator strategies for another type of robots in a few months, although this will heavily depend on the availability of a good simulator and the perception capabilities of the robots.

Considering perception, let us recall that in this study we use an overhead camera to provide exact location information to predators and the prey. To get a ”real” real-world system that works outside the lab, we are extending the Thymios with cameras and re-evolve the controllers such that (pre-processed) camera images are part of the inputs to the neural network.

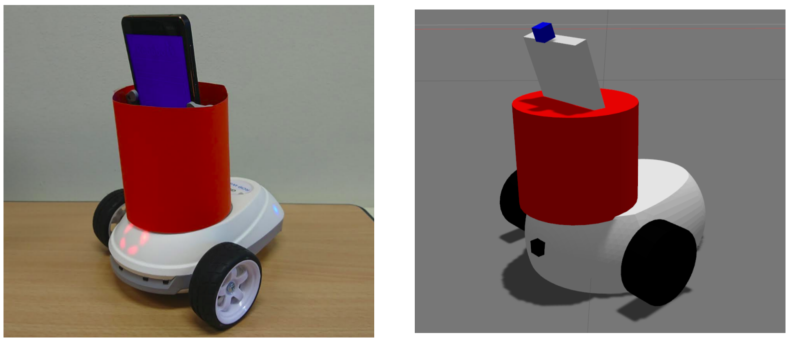
Chapter 4

Second Stage

4.1 Method

4.1.1 Robot

In the second stage, Robobo is our new robot to perform our experiments. Robobo is capable of carrying a mobile phone as a camera. We can use the camera to locate the relative position of the predators and the prey. We choose to use the front camera of the mobile phone to reduce the delay of data processing, because the resolution of the front camera is relatively low. Robobo has five front IR(infrared) sensors and three back IR sensors. The maximal range of IR sensors are only 20 cm. so we merely use one front IR sensor in the middle for collision detection. The overhead camera in the second stage is not necessary because we want to make our system only relevant to the sensors of the robots.Robobo has five front IR(infrared) sensors and three back IR sensors, but we merely use one front IR sensor in the middle to avoid from collision. The blue cube in Figure 4.1b represents the camera of Robobo. The red cylinder is for the camera to detect the robot, and the prey can be distinguished by using green cylinder.



(a) The real robot (b) The simulated robot

Figure 4.1: Appearance of Robobo in the real world and the simulation world.

4.1.2 The Simulation Environment

Integrating Gym of OpenAI, ROS, and Gazebo

Two of the most popular simulation environments are Gazebo and VRep. But it spends time to build an environment for research requirement. That's why there are platforms such as Gym[1] from OpenAI to encapsulate the environment with an interface to perform action such that users can only focus on algorithms. So we integrate Gym, ROS, and Gazebo together to process all the details including controlling robots, resetting the experiment and getting world information. Figure 4.2 shows the architecture of the integration.

In the beginning of one evaluation, the predators are placed parallel to one side of the wall, the prey is placed at the center. The arena remains a square field, but the size is double as 4m×4m to allow that the robots move more freely. Figure 4.3 shows the environment in the simulation world and the viewpoints from the different agents. When the robot detects the other robots, a bounding box is created to select the region of

[1]https://gym.openai.com/

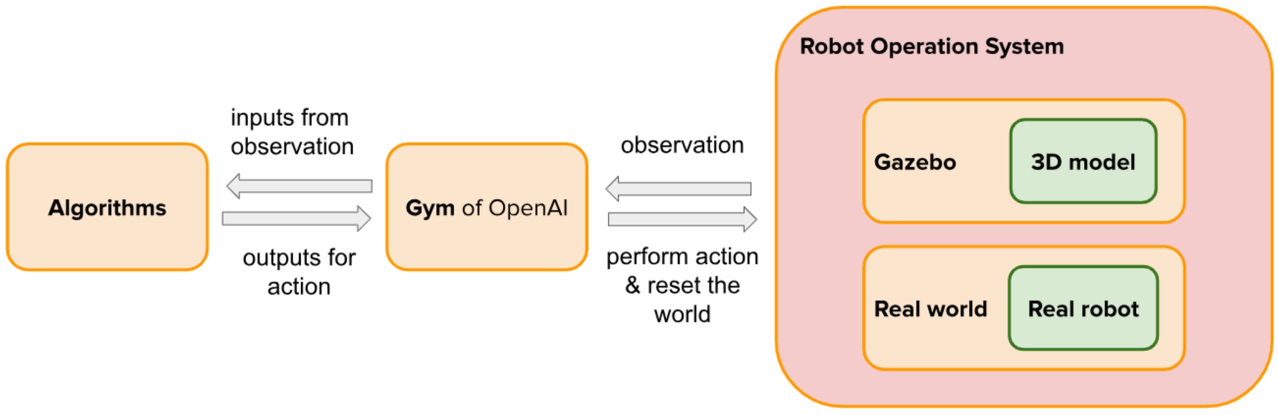


Figure 4.2: Integrating Gym, ROS, and Gazebo. The users only need to focus on algorithms by receiving observed world data and generating output for action. The rest of things will be processed by Gym and ROS. RROS encapsulate Gazebo and the real world to provide the control interface for Gym.

detected agent by color detection. The gazebo allows user to simulate IR sensor and camera, so we try to make the difference between the simulation and the real world as small as possible. The 3D model of the robots were created by FreeCAD, which is an open source technical drawing software to create 3D models. Figure 4.4 shows the sketch of the robot and its 3D model in FreeCAD.

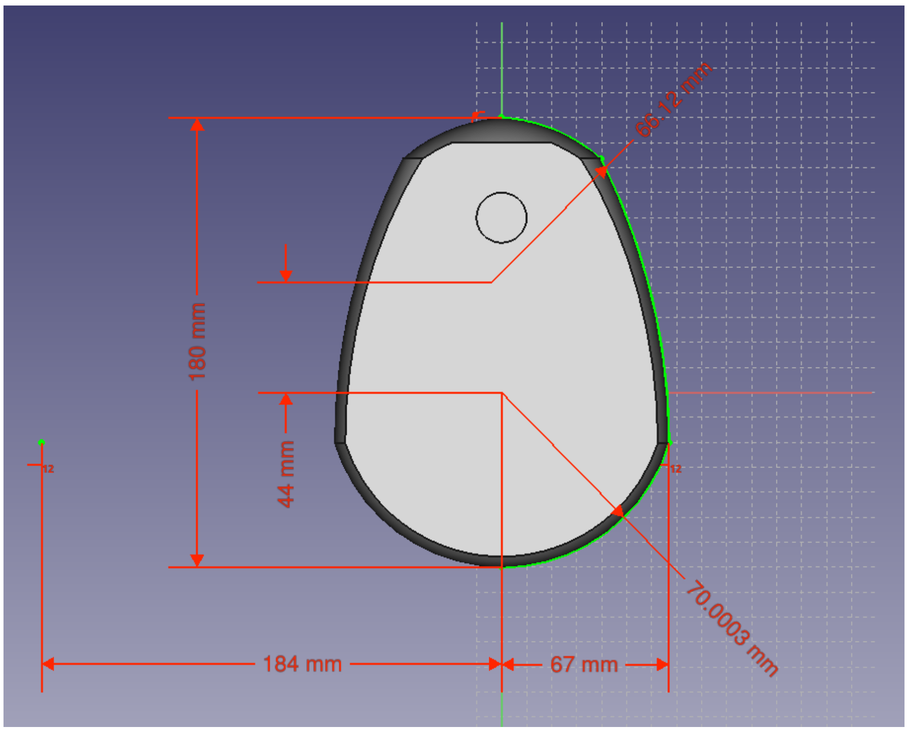
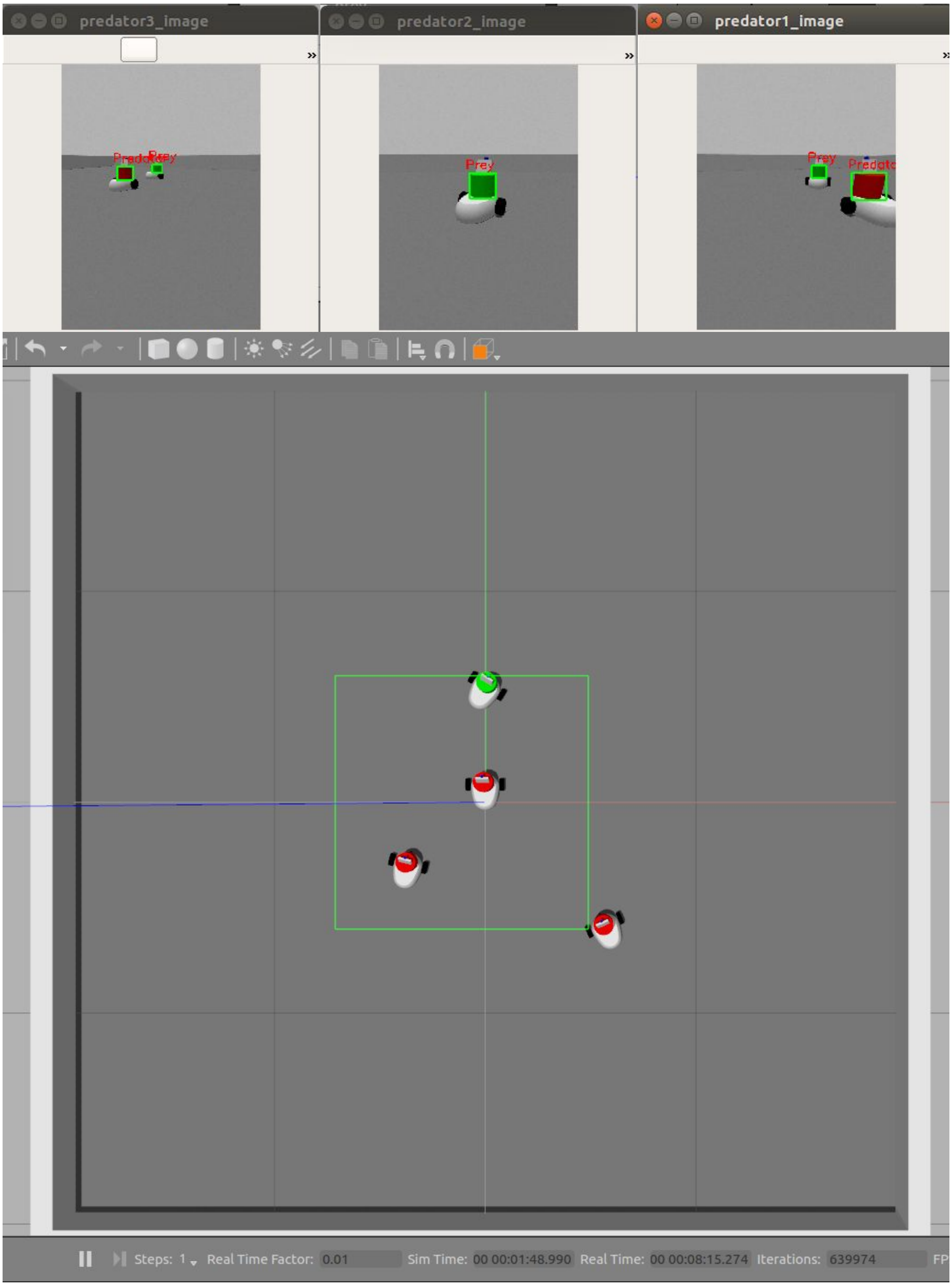


Figure 4.4: Sketch of the robot in FreeCAD

4.1.3 The Real World Environment

The real world environment is a 4m×4m arena. In this time, we removed the camera above. The robots must rely on their own sensors. Three predators and one prey are connected with a computer via Wifi. A ROS master server is running on the computer. So robots pass their information from sensors to the computer via ROS, and then the controllers on the computer take data from sensors as inputs to generate outputs. In the final, the computer send control command back to the robots. Figure 4.5 shows the viewpoints from the agents and the experimental setting in the real world.



Steps: 1 Real Time Factor: 0.01 Sim Time: 00 00:01:48.990 Real Time: 00 00:08:15.274 Iterations: 639974

Figure 4.3: The simulation environment of the second stage in Gazebo. Three windows on the top are the views from different agents. The detected objects are selected by the bounding boxes.

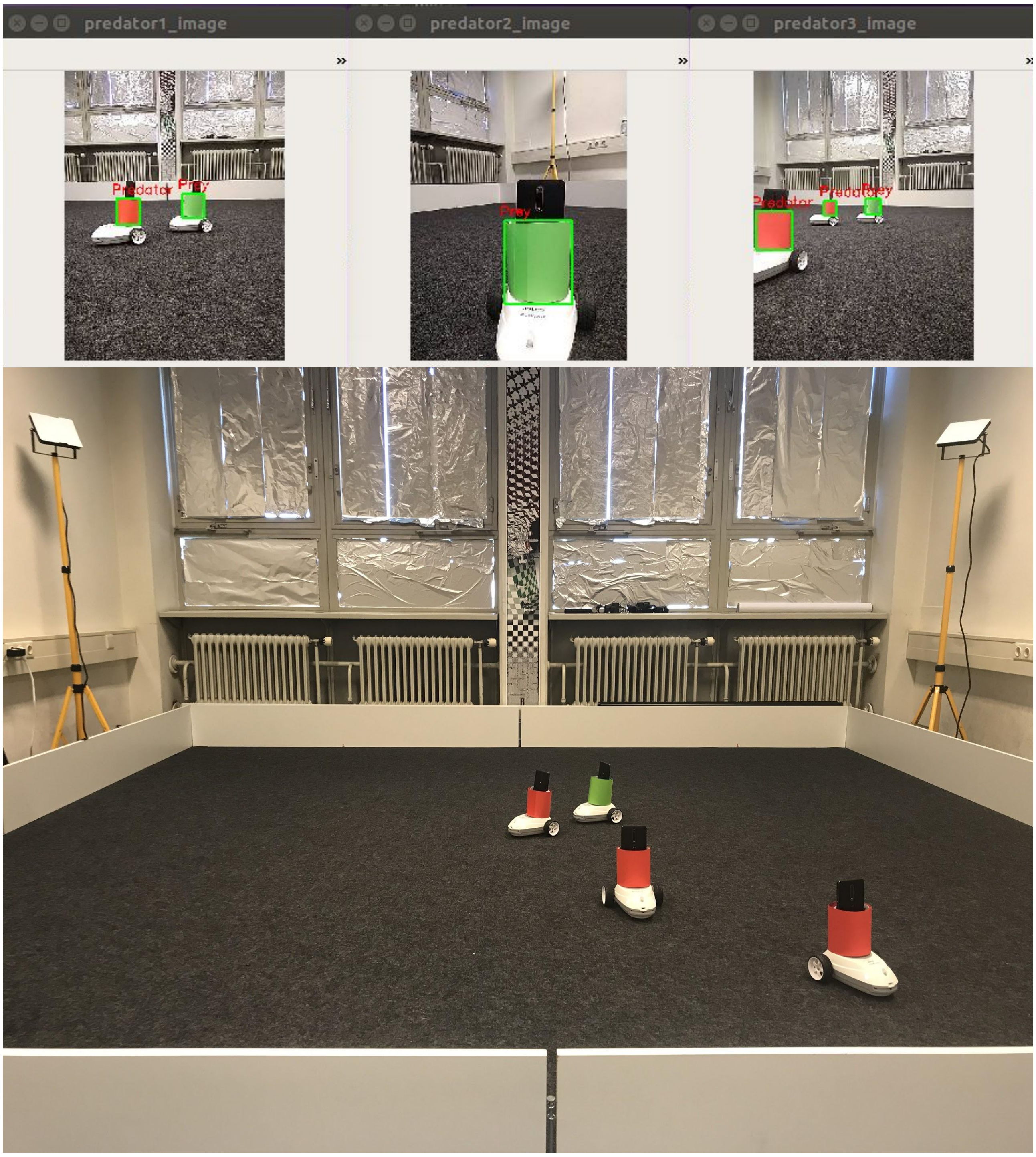


Figure 4.5: The real world experiment of the second stage. Three windows on the top are the views from different agents. The detected objects are selected by the bounding boxes.

4.1.4 Partially Observable

Unlike the first stage using an overhead camera, in the second stage, we merely rely on the camera and the IR sensors on the robots themselves. This choice can extend the application of the predator-prey evolution system. However, we must face another challenge when we merely rely on the sensors of robots, which is that the world becomes partially observable for robots. When a robot is placed at a new place, it can't be sure where it is in this world and how it should move until the robot has detected the environment nearby itself. Take self-driving vehicles as an example, a self-driving system contains multiple deep neural networks to detect the environment, including the model to detect pedestrians, the model to detect traffic light...etc. Based on that environment information, the computer of the vehicle can make the decision of the next action.

4.1.5 Controller for prey

The controller of the prey and the controllers for the predators are both evolved by NEAT. Our purpose is to evolve the predators with the sensors to catch the prey. One of the advantage of the simulation environment is that we can access specific data easily, compared to the real world. For example, we are able to directly get the pose of robots from Gazebo, however, in the real world, we must rely on some techniques like SLAM or GPS to locate the pose of robot. This fact allows us to create a all-knowing prey to help the evolution of predators. Here we list the inputs and the outputs of the prey:

Input Layer:

- $\Delta\theta_1, \Delta\theta_2, \Delta\theta_3$: The angle difference between the orientation of the prey and the direction of the predators with relative to the prey.
- d_1, d_2, d_3: The distance between prey and predators.
- x, y: The coordinates of the prey.

Output Layer:

- ω_L: Angular velocity of the left wheel
- ω_R: Angular velocity of the right wheel

Using hyperbolic tangent as the activation function.

To be aware of that the inputs are much more than the predators controller, which we can see in next subsection. This fact implies the mission is much harder for the prey, because the difference in the number of agents. Also, in most of situations, the predators only need to focus on moving toward the prey, however, the prey can't just simply run away from the predators, instead the prey has to turn 90 degrees to avoid from hitting the wall or being trapped from multiple predators. To evolve a better prey, we need a robot with wide-ranging sensors, otherwise, we have no choice but to use an all-knowing prey to make predators better.

4.1.6 Heterogeneous controllers for predators

Instead of the all-knowing prey, the predators can merely rely on their cameras and IR sensors. Using the heterogeneous controllers for each predator can increase the diversity of the predators and prevent from evolving into monotonous behaviour like only trailing the prey. We tried various inputs, but the simple inputs have a better effect. Here we list our inputs and outputs for the controllers of the predators:
Input Layer:

- x_{image}: The horizontal position of prey shown in the image coordinate, the center is defined as zero.
- A: The area of the prey in the image, and -1 if there is no prey in the image.
- c: +1 if middle front IR sensor detects an object, otherwise -1.

Output Layer:

- ω_L: Angular velocity of the left wheel
- ω_R: Angular velocity of the right wheel

Using hyperbolic tangent as the activation function.

4.1.7 Standard Coevolution Framework

In the second stage, we want to try heterogeneous controllers for the predators, therefore, there are four targets that need to be evolved, three predators and one prey. Assuming one of the predators is evolving, what are the teammates and the opponent that we should pick for an evolving target? For the teammates and the opponent we both pick the individuals with the best fitness from the previous generation, and randomly pick for initialization. It's more meaningful to cooperate with better teammates, because if we randomly pick two teammates for the evolved target, which leads to the performance of the team more depends on the teammates but not evolved target. The noise of fitness could be huge and more randomly. Also, it's more meaningful to compete with the best opponent, if the evolving predator can raise the performance even against the best opponent and the teammates are fixed comparing to the individuals in the same generation, which means the performance should be credited to the evolving predator. Otherwise, we don't know a good or bad

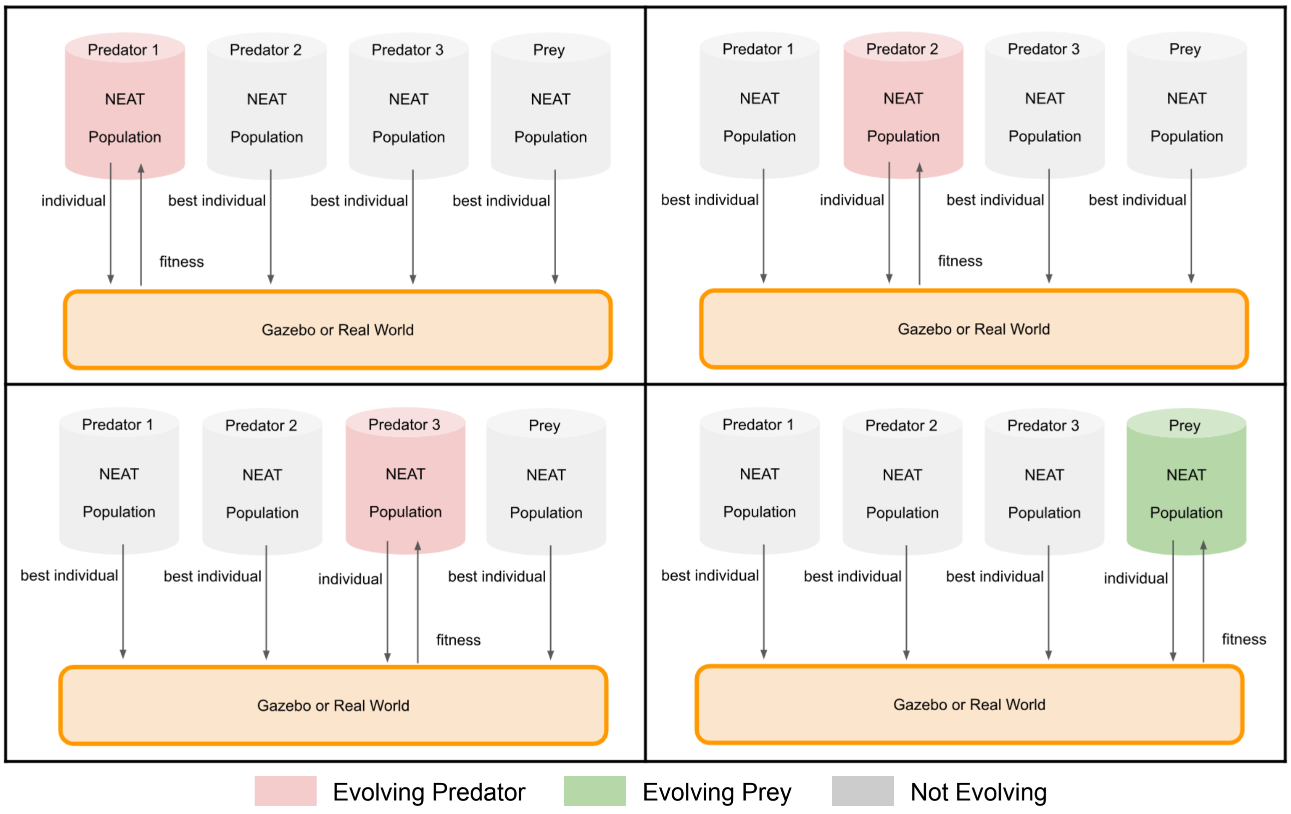


Figure 4.6: The standard co-evolution framework. Each agent evolves alternatively. The evolving agent(pink or green) tests its individual and get fitness as feedback. The other agents(grey) send the best controller from the previous generation to form the team or be the opponent(s).

performance should be credited to the evolving predator or a randomly chosen prey. Potter and De Jong talked about the architecture for the standard coevolution framework[20]. We apply the standard coevolution framework to our predators and prey task, and the architecture of coevolution can be drawn as Figure 4.6 we evolve all the agents alternatively. Except for the evolving target, the other agents are usually controlled by the best controllers from the previous generation. But we also implemented the technique ”Hall of fame” for competitive coevolution, which means that the agent must play against the best opponents from each earlier generation. To reduce the evolution time, we make the evolving target to play against the best opponents from the previous 5 generations. The parameters of NEAT is given in Table 4.1

parameter	value
Number of generation	100
Population size	20
Weight mutate rate	0.8
Bias mutate rate	0.7
Probability of adding(deleting) connection	0.1
Probability of adding(deleting) node	0.1
Number of elites	4

Table 4.1: The main parameters of NEAT

4.1.8 Fitness Function

We take the fitness function from [22] as a reference. For the prey, the fitness is computed by the survival time, the longer the prey survives, the higher the fitness can be. So the upper bound of fitness is when the prey never get caught during the evaluation time 30 seconds. For the predators, we choose to use selfish fitness function for each predator, in other words, only the final distance between evolved predator and the prey is considered. We want to emphasize that one evaluation is ended when one of the predators catches the prey. If there is an outstanding predator, it may lower the fitness of the other predators, which can motivate the other predators to become the first one who can catch the prey. But if all the predators are so selfish that they merely chase behind the prey, which may fail to catch the prey. We also tried to share the same fitness between all the predators, however, it seems that an outstanding predator usually makes the other predators lazy. The outstanding predator confuses others so that the other predators don't know if they are good. Here we display the mathematical representation of the fitness functions:

Fitness function for prey:

$$f_{prey}(t) = \frac{t}{T} \tag{4.1}$$

T is a constant for evaluation time 30. t is caught time, t = T if the prey is not caught.

Fitness function for a predator:

$$f_{predator}(d) = \frac{1}{d} \tag{4.2}$$

d is the distance between the evolving predator and the prey. The reason that we use the inverse of distance instead of that a constant minus d is because there is a huge difference between "close" and "extremely close". Otherwise, the predators will be inclined to get easy points by staying somewhere close to the prey but never catch it.

4.2 Experiments

The experiments include the evolution process and the evaluation in the simulation world, and evaluation in the real world.

4.2.1 Evolution and Evaluation in simulation

As we introduced the standard coevolution framework in subsection 4.1.7, we evolved both the predators and the prey alternatively with 100 generations and relatively small population size 20. According to the experience from stage 1, the small population size, even 13 is enough to evolve the agents. After the evolution process is finished, we tried to evaluate the controllers with Master Tournament, which means that we select the controllers with the highest fitness in every generation, and then we make them play against each other. So there will be 100 $\times$ 100 evaluation times for all the 100 generations of both the predators and the prey. Figure 4.7 shows the caught time of the master tournament. The later generation inclines to perform better when it competes against early generation. However, we can see that the task is more difficult to the prey from that most of values are below 10 seconds. Also the instability happened nearby 90th generation of predators and 13th generation of prey.

Figure 4.8 is drawn by accumulated scores of agents. The score of prey is defined by caught time, and it accumulates a prey from a generation to play against all generations of predators. The score of predators is defined by that a constant minus the caught time, and the calculation of the accumulated score is similar. The graph shows that the coevolution can be stable in this evolution framework, although it seems that the upper limit of performance is hit around the 50th generation of predators.

To see that the behaviour was getting more complicated, we cherry-picked the predators from the 41st generation play against the preys from all the generations, and draw their trajectories as shown in Figure 4.9. During the generations between 0 to 19, the best strategy for the prey is to move to the corner for prolonging the survival time. From the 20th to the 39th generation, the prey developed the strategy to slip away from the gap between the predator and the wall. The prey successfully went through the gap at the 24th generation. The generations from 40 to 100, Although, the performance for prey didn't change too much, the prey still tried to run away from the predators along different paths.

4.2.2 Evaluation in the real world

In the real world, the prey cannot be an all-knowing player, but we are still interested in evaluating the performance of the evolved predators when the controllers are transferred from the simulation world to the real world. So we use multiple human prey players to evaluate the performance of evolved predators in both simulation and the real world.

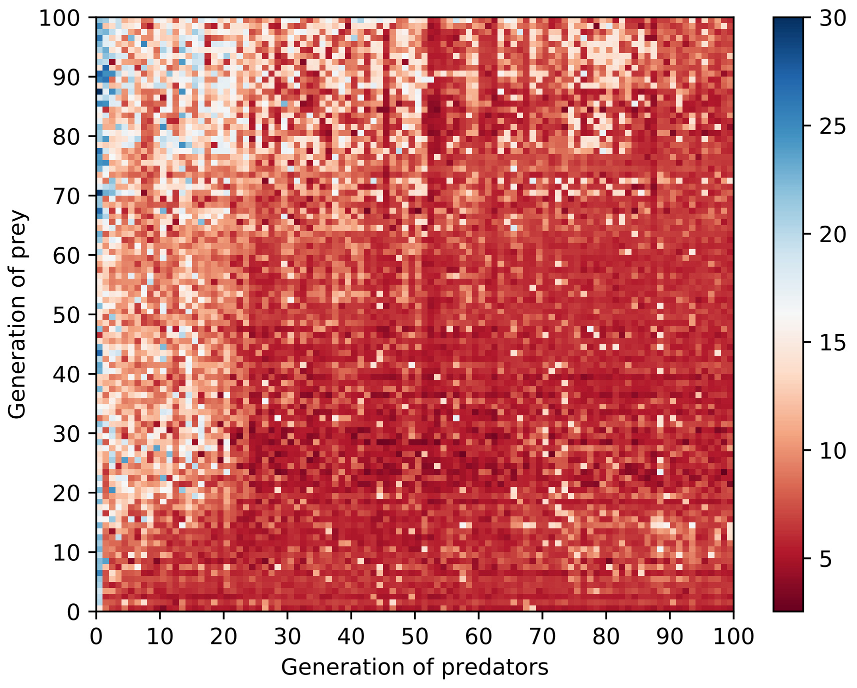


Figure 4.7: The best prey from 100 generations versus the best predators from 100 generations. It shows a trend that the agent from the later generation performs better when it plays against the agent from the early generation.

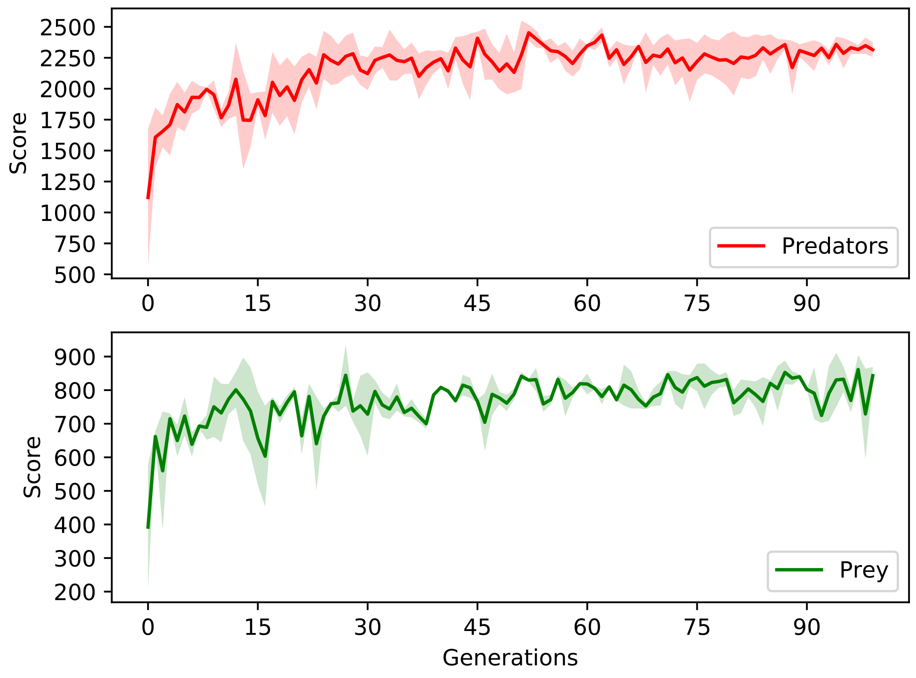
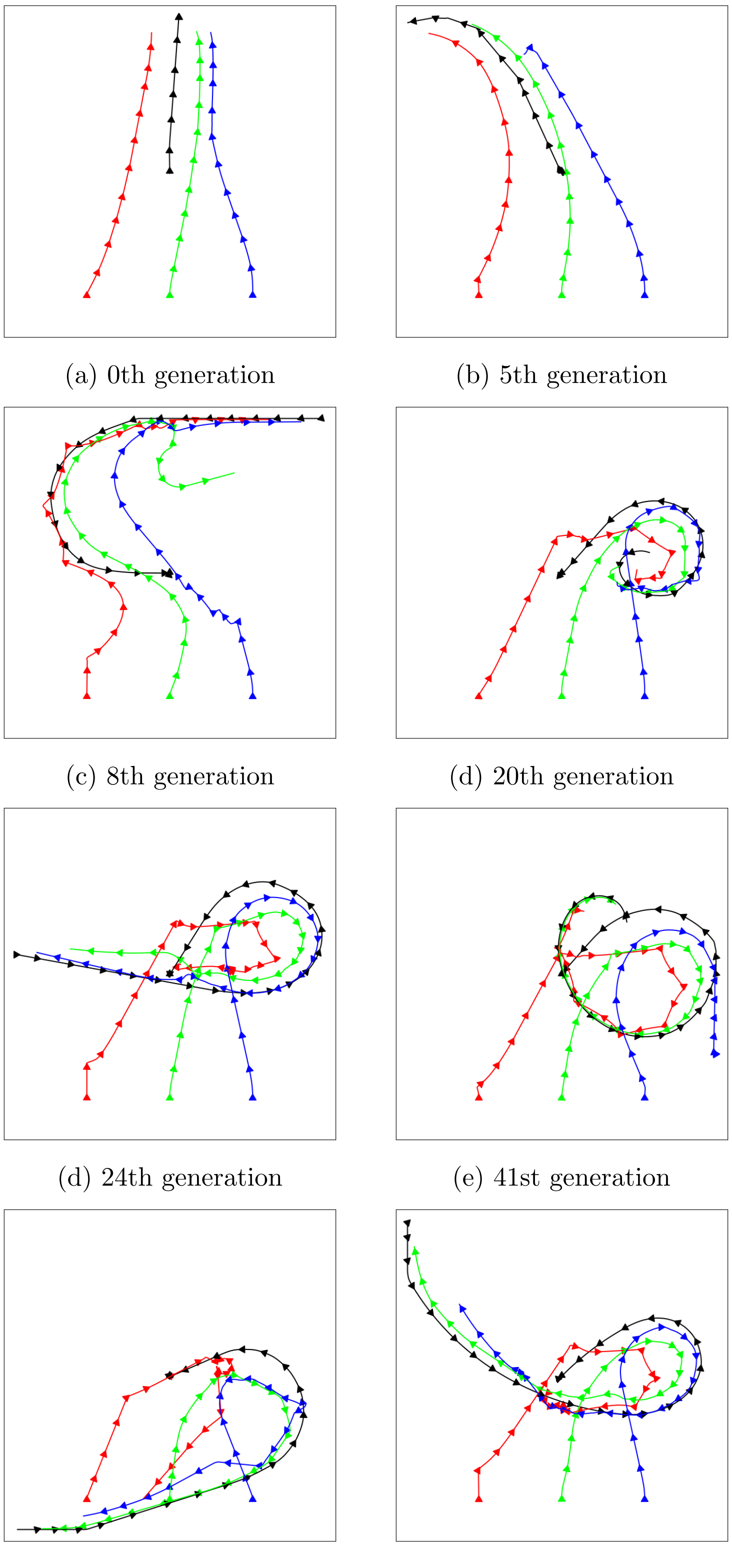


Figure 4.8: The best prey from 100 generations versus the best predators from 100 generations. The graph shows the accumulated score for each generation and the final performance is stabilized.

We created an interface for the human to control the predator with keyboard for both the simulation world and the real world. However the human players may introduce the bias, taking the average from multiple



(a) 0th generation
(b) 5th generation
(c) 8th generation
(d) 20th generation
(d) 24th generation
(e) 41st generation
(f) 53rd generation
(g) 75th generation

Figure 4.9: The trajectories of the predators from 41th generation versus the prey from the other generations

human players is necessary. The time spent to catch the human prey player is the criterion. The result can be shown in Figure 4.10. The average survival time in the simulation world is 8.43 seconds, however in the real world the average survival time is 21.59 seconds. Even though the predators in the real world show the behaviour of pursuit, they still need further evolution to be applied in the real world. The reality gap will be further discussed in the section 4.3.

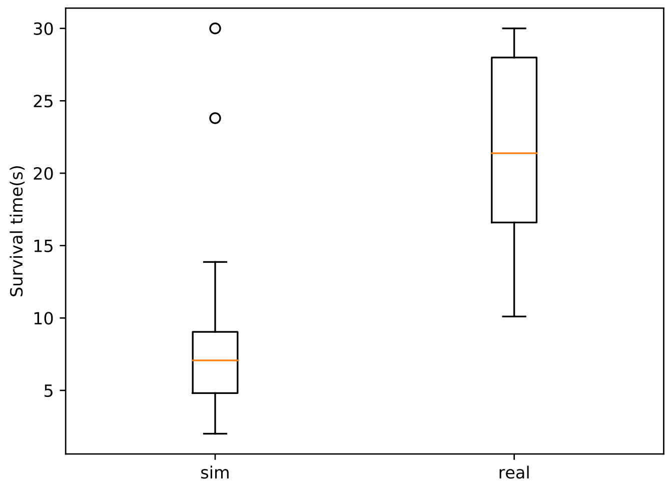


Figure 4.10: The performance of the human controlled prey versus the evolved predators in both the simulation world and the real world.

4.3 Discussion and Limitations

4.3.1 The source of the reality gap

The number of frames per second(FPS) could be a source which leads to increase the reality gap. FPS in the simulation world is around 10(because of the computation of simulation, image processing, and running algorithm), however due to the transmission of high quality of images via local WiFi, ROS and a router, the FPS in the real world can only be 5-7. The lower FPS has an impact on the object detection, some frames can be missed which leads to over rotate, even though during the rotation the object can be detected as shown in Figure 4.11.

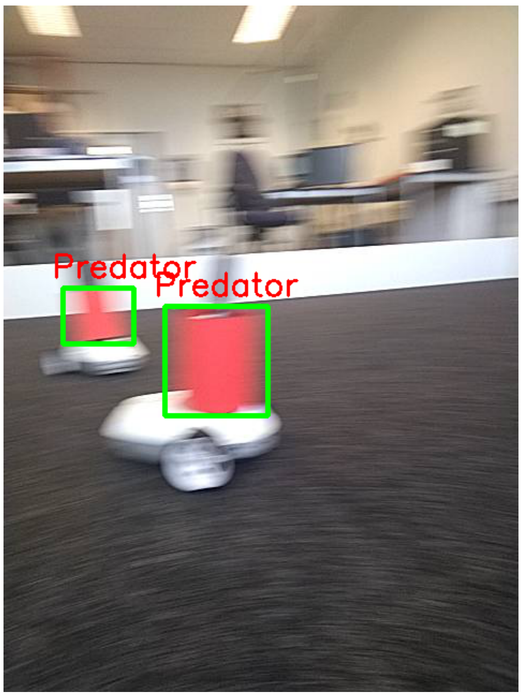


Figure 4.11: The viewpoint from a rotating robot. Two predators are detected.

4.3.2 Triggering the arms race

The arms race is not easy to be triggered. Even the observed behaviour becomes more complicate, it's not necessary that the performance is always better. However, the advantage of coevolution is to evolve various opponents without delicate human design. With the help of "Hall of Fame", the predators are able to catch various opponents and get higher average fitness. The variety of the evolution process makes the predators complete their evolution target in a certain degree. "Hall of Fame" can be implemented by choosing the opponent by random sampling. We also tried to select opponents from 10 previous generations, but it cannot stabilize the coevolution process. A sampling method for selecting opponents is necessary to stabilize coevolution. However, if we select opponent random sampling, it means that when the generation increases, it's possible that the sampling method, unfortunately, selects only bad performance controllers, which may lead to misestimating fitnesses, and it may further lead to hitting the upper limit of performance. In our experiment, the predators hit the upper limit of performance about 50th generation, so we may improve the performance with performance-based methods as mentioned in the work from Rosin and Belew [23].

4.3.3 Fitness function

In a few preliminary experiments, different fitness functions are tested. For the predator, fitness function $f(d) = constant - d$ and $f(d) = \frac{1}{d}$, d is the distance between the prey and the evolving predator, have a huge difference, because it distinguishes the difference between "close" and "extremely close". The fitness function also can be defined by team performance, instead of only individual performance. In our other preliminary experiments, the fitness which defined by full team performance usually generates a lazy individual. It may be caused by that the difficulty of this task for multiple predators is relatively lower to the prey. The capitalism styled fitness function $f(d) = \frac{1}{d}$ works well at least in the simulation world, in other words, every predator works for itself to maximize the resource on its hands, but the whole team can be benefited from the result too. The predators are also not that selfish to just follow behind the prey, otherwise the severe collision should be observed, which means that they must collaborate in a certain degree.

4.3.4 The Heterogeneous Controller

The heterogeneous controller indeed shows the diversity compared to the homogeneous controller. According to our observation, some of the predators can rotate in a fixed direction(e.g. clockwise) to search the prey, and move toward prey directly when they find the prey, another observed controller can follow the prey smoothly and rotate in different directions which depends on where the prey disappeared in the views of predators. The various types of predators make they wouldn't just follow behind the prey as single homogeneous controller.

4.3.5 Evolution Time

The evolution time of the second stage is much longer compared to the first stage. There are two main reasons, the first one is because we need "Hall of Fame" to stabilize the process of coevolution, which makes our evolution n times longer than simple evolution, and n depends on how many previous generations to be competed with. Another reason is the time for simulating the camera and IR sensors to reduce the reality gap, it largely slows down the simulation speed of Gazebo. The whole evolution process needs 100 hours for 100 generations and population size as 20, which is 40 times slower than using a homogeneous controller and a fully observable environment.

4.4 Conclusion of the Second Stage

We presented a better evolutionary framework for both in the simulation world and the real world by integrating Gym, Gazebo and ROS. The interface provided by Gym makes users and research focus on the algorithms, instead of spending time on the communication with Gazebo in the simulation world or hardware in the real world. The framework allows us to perform the experiments about both cooperative coevolution and competitive evolution, furthermore, reinforcement learning algorithm can be also applied because Gym is part of the integration.

Our experiment shows that the standard coevolution framework can be also applied for sensor-based 3-versus-1 predator(s) and prey scenario. However, the transferability from the simulation to the real world can be limited by the hardware resource. The evolved robots still need to further evolve in the real world. The sampling for selecting opponents is the key step to stabilize the coevolution. At least the random sampling must be used.

The simulation of camera and IR sensors consumes large computational resource such that the whole evolution process becomes much slower. It weakens the advantage in the simulation world, the cost of time is not cheap anymore compared to the real world. Also, the requirement of computation resources enlarges the reality gap, because usage of computation resource and FPS cannot be consistent.

Chapter 5

Conclusion

5.1 Conclusion Combining Two Stages

5.1.1 Pure Gazebo V.S Integrating Gym, Gazebo, and ROS

The evolutionary system is improved in the second stage. Gym is the bridge between the algorithm and the world, users only need to take care of the algorithm part after integrating the world into Gym. ROS can be used as the inferface for controlling robots in both the simulation world and the real world to increase the code reusability. Choosing Gym brings us another advantage. It is a popular toolkit for reinforcement learning and it also can be used for evolutionary algorithm, so the users are able to get started quickly.

5.1.2 Homogeneous controllers v.s Heterogeneous Controllers for Predators

In a classical predator-prey task, it's hard to doubt that if the performance of heterogeneous controllers is better than homogeneous controllers. Many studies have shown the fact that heterogeneous controllers have higher variety, which leads to better performance. However, the heterogeneous can be still applied to a non-classical predator(s)-prey task. In an imaginary scenario, if the cost of predators robot is cheap. So it's allowed us to use numerous predators to catch a prey. It's impractical to evolve the numerous predators one by one when we use heterogeneous controllers, and we have no choice but to apply homogeneous controllers for the predators. In our experiment, the obvious collective behaviour expends the potential application of homogeneous controller. And the development of collective behaviour can be encouraged by selecting features and the design of fitness function.

5.1.3 Fixed Strategy v.s Coevolution for Prey

Both the fixed strategy and the coevolution process for prey are able to train the predators to catch the target. The coevolution needs much longer evolution process. Not only because the prey needs to evolve, but also the coevolution needs some techniques like "hall of fame" to stabilize the evolution process. It requires almost 13 times slower than using the fixed strategy. However, the coevolution increase the variety of prey, the predators get higher fitness only when they can catch various prey. We don't need an elaborately designed strategy when we apply coevolution.

5.1.4 Fully Observable v.s Partially Observable Environment

In the first stage, the fully observable environment needs the shorter time to evolve compared to the partially observable environment. However, it requires the above camera to get the coordinates of the robots in the real world. But the functionality of the camera should be replaced by GPS or SLAM to make it more practical in the different scenarios.

In a partially observable environment, the robots only rely on the sensors of themselves. To reduce the reality gap, we needed to simulate the camera and IR sensors. This requires more computational resources, which leads to the simulation process of the second stage become around 7 times lower than the first stage. In the second stage, we combine the fully observable environment for prey and the partially observable environment for predators. Some features are easy to get from the simulation world, for example, the coordinates of the agents. It allows us to create an all-knowing prey to co-evolve with the predators. Also there are 3 predators but only one prey, the difficulty of this task for prey and predators are different. So in the simulation world, we need an all-knowing prey which can evolve into a wiser prey compared to a sensor-based prey. And then the evolved predators can be used in the real world to catch the prey. We also tried to develop sensor-based co-evolved prey, however, it seems that the range of sensors is not wide enough for this task. Here is a quote[13] from the work of Cliff and Miller that we see in the work from Nolfi and Dario Floreano: "..pursuers usually evolved eyes on the front of their bodies (like cheetahs), while evaders usually evolved eyes pointing sideways or even backwards (like gazelles)."

Appendix A

Full parameters of NEAT

A.1 For Predator

The following are the parameters that we used in the second stage for python module NEAT-Python:

```
[NEAT]
fitness_criterion = max
fitness_threshold = N/A
no_fitness_termination = True
pop_size = 20
reset_on_extinction = False
DefaultGenome
node activation options
activation_default = tanh
activation_mutate_rate = 0.0
activation_options = tanh
node aggregation options
aggregation_default = sum
aggregation_mutate_rate = 0.0
aggregation_options = sum
node bias options
bias_init_mean = 0.0
bias_init_stdev = 1.0
bias_max_value = 30.0
bias_min_value = -30.0
bias_mutate_power = 0.5
bias_mutate_rate = 0.7
bias_replace_rate = 0.1
genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient = 0.5
connection add/remove rates conn_add_prob = 0.1
conn_delete_prob = 0.1
connection enable options
enabled_default = True
enabled_mutate_rate = 0.01
feed_forward = True
initial_connection = full
node add/remove rates
node_add_prob = 0.1
node_delete_prob = 0.1
network parameters
num_hidden = 0
num_inputs = 3
num_outputs = 2
node response options
response_init_mean = 1.0
```

response_init_stdev = 0.0
response_max_value = 30.0
response_min_value = -30.0
response_mutate_power = 0.0
response_mutate_rate = 0.0
response_replace_rate = 0.0
connection weight options
weight_init_mean = 0.0
weight_init_stdev = 1.0
weight_max_value = 30
weight_min_value = -30
weight_mutate_power = 0.5
weight_mutate_rate = 0.8
weight_replace_rate = 0.1
DefaultSpeciesSet
compatibility_threshold = 3.0
DefaultStagnation
species_fitness_func = max
max_stagnation = 20
species_elitism = 2
DefaultReproduction
elitism = 4
survival_threshold = 0.2

A.2 For prey

[NEAT]
fitness_criterion = max
fitness_threshold = N/A
no_fitness_termination = True
pop_size = 20
reset_on_extinction = False
DefaultGenome
node activation options
activation_default = tanh
activation_mutate_rate = 0.0
activation_options = tanh
node aggregation options
aggregation_default = sum
aggregation_mutate_rate = 0.0
aggregation_options = sum
node bias options
bias_init_mean = 0.0
bias_init_stdev = 1.0
bias_max_value = 30.0
bias_min_value = -30.0
bias_mutate_power = 0.5
bias_mutate_rate = 0.7
bias_replace_rate = 0.1
genome compatibility options
compatibility_disjoint_coefficient = 1.0
compatibility_weight_coefficient = 0.5
connection add/remove rates conn_add_prob = 0.1
conn_delete_prob = 0.1
connection enable options
enabled_default = True
enabled_mutate_rate = 0.01
feed_forward = True
initial_connection = full
node add/remove rates

```
node_add_prob = 0.1
node_delete_prob = 0.1
network parameters
num_hidden = 0
num_inputs = 8
num_outputs = 2
node response options
response_init_mean = 1.0
response_init_stdev = 0.0
response_max_value = 30.0
response_min_value = -30.0
response_mutate_power = 0.0
response_mutate_rate = 0.0
response_replace_rate = 0.0
connection weight options
weight_init_mean = 0.0
weight_init_stdev = 1.0
weight_max_value = 30
weight_min_value = -30
weight_mutate_power = 0.5
weight_mutate_rate = 0.8
weight_replace_rate = 0.1
DefaultSpeciesSet
compatibility_threshold = 3.0
DefaultStagnation
species_fitness_func = max
max_stagnation = 20
species_elitism = 2
DefaultReproduction
elitism = 4
survival_threshold = 0.2
```

Bibliography

[1] Gianluca Baldassarre, Stefano Nolfi, and Domenico Parisi. Evolving mobile robots able to display collective behaviors. *Artificial life*, 9(3):255–267, 2003.

[2] Gianluca Baldassarre, Stefano Nolfi, and Domenico Parisi. Evolving mobile robots able to display collective behaviors. *Artificial life*, 9(3):255–267, 2003.

[3] Adrian Boeing and Thomas Bräunl. Evaluation of real-time physics simulation systems. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 281–288. ACM, 2007.

[4] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, 2nd edition, 2015.

[5] fernando(fmfn). Github - fmfn/bayesianoptimization: A python implementation of global optimization with gaussian processes. `https://github.com/fmfn/BayesianOptimization`. (Accessed on 01/25/2019).

[6] Dario Floreano and Stefano Nolfi. God save the red queen! competition in co-evolutionary robotics. In *2nd Conference on Genetic Programming*, number CONF, 1997.

[7] Jorge Gomes, Miguel Duarte, Pedro Mariano, and Anders Lyhne Christensen. Cooperative coevolution of control for a real multirobot system. In *International Conference on Parallel Problem Solving from Nature*, pages 591–601. Springer, 2016.

[8] Nikolaus Hansen. Benchmarking a bi-population cma-es on the bbob-2009 noisy testbed. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2397–2402. ACM, 2009.

[9] Nikolaus Hansen and Stefan Kern. Evaluating the cma evolution strategy on multimodal test functions. In *International Conference on Parallel Problem Solving from Nature*, pages 282–291. Springer, 2004.

[10] T Haynes and Sandip Sen. Co-adaptation in a team. *International Journal of Computational Intelligence and Organizations*, 1(4):1–20, 1996.

[11] Y Ho, A Bryson, and Sheldon Baron. Differential games and optimal pursuit-evasion strategies. *IEEE Transactions on Automatic Control*, 10(4):385–389, 1965.

[12] Joel Lehman and Kenneth O Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[13] Pattie Maes, Maja J Mataric, Jean-Arcady Meyer, Jordan Pollack, and Stewart W Wilson. Co-evolution of pursuit and evasion ii: Simulation methods and results. 1996.

[14] Lincoln Smith Matt Quinn, Giles Mayley, and Phil Husbands. Evolving teamwork and role-allocation with real robots. *Artificial Life 8*, 8:302, 2003.

[15] Orazio Miglino, Henrik Hautop Lund, and Stefano Nolfi. Evolving mobile robots in simulated and real environments. *Artificial life*, 2(4):417–434, 1995.

[16] Alireza Mika(alirezamika). Github - alirezamika/evostra: A fast evolution strategy implementation in python. `https://github.com/alirezamika/evostra`. (Accessed on 01/25/2019).

[17] nikohansen. Github - cma-es/pycma: Python implementation of cma-es. `https://github.com/CMA-ES/pycma`. (Accessed on 01/25/2019).

[18] Stefano Nolfi and Dario Floreano. Coevolving predator and prey robots: Do arms races arise in artificial evolution? *Artificial life*, 4(4):311–335, 1998.

[19] Stefano Nolfi and Domenico Parisi. Evolving non-trivial behaviors on real robots: an autonomous robot that picks up objects. In *Congress of the Italian Association for Artificial Intelligence*, pages 243–254. Springer, 1995.

[20] Mitchell A Potter and Kenneth A De Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation*, 8(1):1–29, 2000.

[21] Eric Raboin, Ugur Kuter, and Dana Nau. Generating strategies for multi-agent pursuit-evasion games in partially observable euclidean space. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*, pages 1201–1202. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[22] Aditya Rawal, Padmini Rajagopalan, and Risto Miikkulainen. Constructing competitive and cooperative agent behavior using coevolution. In *Proceedings of the 2010 IEEE conference on computational intelligence and games*, pages 107–114. IEEE, 2010.

[23] Christopher D Rosin and Richard K Belew. New methods for competitive coevolution. *Evolutionary computation*, 5(1):1–29, 1997.

[24] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[25] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[26] Leigh Van Valen. The red queen. *The American Naturalist*, 111(980):809–810, 1977.

[27] Chern Han Yong and Risto Miikkulainen. Coevolution of role-based cooperation in multiagent systems. *IEEE Transactions on Autonomous Mental Development*, 1(3):170–186, 2009.