Abstraction, Sensory-Motor Coordination, and the Reality Gap in Evolutionary Robotics

Scheper, Kirk; de Croon, Guido

DOI
10.1162/ARTL_a_00227

Publication date
2017

Document Version
Final published version

Published in
Artificial Life

Abstraction, Sensory-Motor Coordination, and the Reality Gap in Evolutionary Robotics

Kirk Y. W. Scheper*,**
Guido C. H. E. de Croon**
Delft Institute of Technology

Keywords
Sensory-motor control, evolutionary robotics, reality gap, micro air vehicle

Abstract One of the major challenges of evolutionary robotics is to transfer robot controllers evolved in simulation to robots in the real world. In this article, we investigate abstraction of the sensory inputs and motor actions as a tool to tackle this problem. Abstraction in robots is simply the use of preprocessed sensory inputs and low-level closed-loop control systems that execute higher-level motor commands. To demonstrate the impact abstraction could have, we evolved two controllers with different levels of abstraction to solve a task of forming an asymmetric triangle with a homogeneous swarm of micro air vehicles. The results show that although both controllers can effectively complete the task in simulation, the controller with the lower level of abstraction is not effective on the real vehicle, due to the reality gap. The controller with the higher level of abstraction is, however, effective both in simulation and in reality, suggesting that abstraction can be a useful tool in making evolved behavior robust to the reality gap. Additionally, abstraction aided in reducing the computational complexity of the simulation environment, speeding up the optimization process. Preeminently, we show that the optimized behavior exploits the environment (in this case the identical behavior of the other robots) and performs input shaping to allow the vehicles to fly into and maintain the required formation, demonstrating clear sensory-motor coordination. This shows that the power of the genetic optimization to find complex correlations is not necessarily lost through abstraction as some have suggested.

1 Introduction

Evolutionary robotics (ER) is a robotic design methodology centered around the concept of evolving a robot's body and mind. Since its origins in the 1990s, it has been thought that the best way to facilitate the evolution of intelligent behavior was to provide the robot with raw sensor readings and control the motor output so that the resultant behavior would perform the given task [20]. We refer to individual inputs and outputs accessible to the evolutionary process as *primitives*. Having evolution utilize all available primitives gives the optimization the greatest freedom while removing user bias from the final solution, as primitive preselection is not necessary.

* Corresponding author.
** Faculty of Aerospace Engineering, Delft University of Technology, 2629HS Delft, The Netherlands. E-mail: k.y.w.scheper@tudelft.nl (K.Y.W.S.); g.c.h.e.decroon@tudelft.nl (G.C.H.E.dC.)

This design method initially met with a series of quick successes with reactive agents solving type-2 problems:[1] obstacle avoidance [9], object characterization [2, 21], and mapless spatial localization [11], just to name a few. These tasks were however simple compared to other autonomous robotic applications. Additionally, the robotic platforms used were rudimentary, with relatively few sensors and actuators as well as slow dynamics. Extending ER to more complex systems has proven difficult.

One of the major problems limiting the study of ER on more complex robotic systems and tasks is the *reality gap*. This gap is defined as the difference between the relatively low-fidelity simulated environments where the evolution takes place and the real world. Evolutionary optimization often tends to exploit the intrinsic characteristics of the sensory inputs, motor actions, and feedback between these two. Due to differences between simulation and reality, these solutions do not transfer well to the real robot.

The tight coupling between the actions taken by the robot and the sensory inputs received is referred to as *sensory-motor coordination* (SMC). Utilizing SMC, an agent can partially determine the sensory input patterns it receives by acting in a particular way [21]. It has been suggested that this is the main reason that the simple agents evolved have been able to solve many nontrivial type-2 problems.

With this in mind, it may not be so beneficial to use low-level, unprocessed sensory inputs and motor outputs, as these tend to be environment- and robot-specific. The combination of this reliance on a tight coupling between sensing and action and the level of sensor and action modeling fidelity tends to exacerbate the reality gap, limiting progress in ER research.

Additionally, as we move to more complex tasks, the use of low-level primitives makes it more difficult for evolution to find good solutions early on in the evolutionary process. This so-called *bootstrap problem* is caused by the fact that the fitness function rewards performance on the complex, high-level task, while the robot has to master low-level sensor processing and control. It is very difficult to design fitness functions that guide evolution in the correct direction while not introducing too much designer bias into the final solution.

The main contribution of this article is to show that abstracting away from the conventionally accepted use of low-level primitives can make the optimized behavior more robust to the reality gap.[2] To do this, we optimized two controllers to solve a decentralized formation task with a homogeneous swarm of three quadrotor micro air vehicles (MAVs). One controller, which we call the *low-level controller*, controls the rotor rpm of the quadrotor, while the second controller commands a velocity, which is sent to a closed-loop control system that controls the vehicle. Additionally, we investigate the effect that using a closed-loop control system and high-level primitives has on the ability of evolution to develop SMC.

The following sections will describe work done to achieve our goal, beginning with Section 2, where we discuss SMC in more detail. Section 3 discusses the task we optimized, followed in Section 4 with a description of the real vehicle used in our tests as well as the kinematic model implemented for the evolutionary optimization. Section 5 presents the control schemes used in this article. Section 6 details the implementation of the evolutionary optimizer used in this article and the results of the optimization. The real-world flight test results are presented in Section 7, and a discussion of the reality gap is presented in Section 8.

2 Sensory-Motor Coordination

The main difference between autonomous robotic systems and many other forms of artificial intelligence (AI) is the core principle that robots must physically interact with a real-world environment. Robots are embodied agents that must perceive and understand the world around them in order to

1 Type-1 problems involve data sets where a direct input-output mapping can be found through statistical means. Type-2 problems do not have a direct mapping [5].

2 This work is a significant extension of an article published in the Proceedings of the 14th International Conference on the Simulation of Adaptive Behavior (SAB2016) held in Aberystwyth, UK, in August 2016 [28]. The extension includes evolution of a low-level controller, allowing the comparison of high-level with low-level control and extra analysis of the effect of control system reliability on the reality gap.

take rational actions to achieve their goals. This fact makes it imperative to understand embodied cognition and how it relates to rational behavior. Early work in this field had trouble understanding the cognition process until John Dewey first proposed the concept of a *sense-think-act* cycle [7]. He theorized that instead of an open-loop cognitive process where a perceived state prompts an action, humans in fact exhibited a closed-loop cognitive process by which the actions we take alter our perceived state, thus creating a feedback loop with no clear initial point. He termed this tight coupling between perception and action *sensory-motor coordination*. These ideas about human cognition were most notably further developed by Jean Piaget's investigation of the development of SMC in infants [26] and James Gibson's work in perception by aircraft pilots [10]. This concept can also be generalized to embodied agents within an AI perspective [3, 4, 24].

Pfeifer et al. state that SMC serves five main purposes: (1) physical control over objects (i.e., interaction with the world); (2) perception (i.e., understanding the world); (3) induced correlations of the world, which reduce the dimensionality of the sensor-motor space (i.e., simplifying the world); (4) integration of several sensory modalities (i.e., sensor fusion); and (5) accounting for sensor-motor coordination itself [24]. They go on to say that not only does SMC make correlations; it helps to filter out sensory inputs that are not correlated with the action required to achieve a task [23]. SMC can be seen as a structuring of the sensor inputs of our complex world to facilitate the development of intelligent behavior [17].

If we consider our agent from a dynamical systems perspective, SMC can potentially be seen as a lower-dimensional projection of the higher-dimensional interaction with the world. If so, then optimizing behavior actually entails placing behavioral attractors in this system that will perform the task at hand [18].

In ER, SMC has also been used in connection with self-organization and emergence. Nolfi suggests that in embodied cognition, behavior cannot be considered to be based on internal mechanisms only, but in fact emerges from a strong coupling with the agent's interaction with the environment [20]. This emergence can be explained in three ways: (1) an agent's behavior that is surprising and not fully understood; (2) a property that is not constrained to any of its parts (i.e., self-organization); and (3) behavior resulting from the agent-environment interaction whenever that behavior is not preprogrammed [25].

The general view in ER is to give evolution a bag of low-level primitives, which it has to sift through using SMC to find relevant correlations needed to achieve the task. This adds little restriction or user-biased direction to the task [17, 25]. However, it has the unwanted side effect of introducing a large bootstrapping problem for the optimization, making the optimization of complex behavior difficult. Additionally, the low-level primitives are typically raw sensor inputs and raw motor outputs, which do not generalize well from one robot to another, nor from a simulated to a real robot. The use of low-level primitives is not productive of the development of complex robotic behavior within the ER framework.

Given these difficulties, it seems beneficial to use higher-level primitives when evolving behavior for a complex task, but an unanswered question is: Can evolution still exploit SMC when using high-level primitives with an underlying closed-loop controller? The remainder of this article will try to answer this.

3 Task

In this article, the evolutionary optimization is tasked with developing the behavior of a homogeneous swarm of quadrotor MAVs to form a given asymmetric formation. This task is based on that presented in [14], where a swarm of three SHERES spacecraft was optimized in simulation only. Before [14], methods had been developed to autonomously form symmetric formations, the asymmetric case proving difficult [13]. The design of asymmetric formations using a distributed control system without explicit roles in the formation is a nontrivial task for most human designers, making it an ideal task for automatic optimization.

Unlike [14], we use a simplified two-dimensional formation task to make analysis of the resultant behavior more straightforward. The goal of the swarm is to achieve an asymmetric triangular formation with sides of length: 0.7, 0.9, and 1.3 m. The MAVs can observe the position relative to the other members in the formation as well as relevant ego-motion primitives.

An artificial neural network (ANN) with one hidden layer was selected to perform the robotic control. A tanh activation function was used in the neurons, and network weights were constrained to the range [−1, 1]. Additionally, bias nodes were added to the input and hidden layers, and the exiting network weights were constrained to the range [−5, 5].

To sense the relative position to the other members of the swarm, the main input to the ANN is the sum of the Cartesian components of the relative positions of the other members of the formation ($\mathbf{r}$) defined in

$$\mathbf{r} = \sum_{n=2}^{k} (\mathbf{p}_n - \mathbf{p}_1) \tag{1}$$

where $\mathbf{p}$ is the position vector of a vehicle and k is the total number of vehicles in the swarm. Summing the components of all aircraft relative positions instead of inputting individual vehicle distances ensures that the ANN cannot associate a specific input with a unique vehicle. Note that $\mathbf{r}$ is mathematically equivalent to triple the distance from the ownship to the centroid of the formation ($\mathbf{c}$). An additional and redundant input is the sum of absolute distances (d) as given by

$$d = \sum_{n=2}^{k} |(\mathbf{p}_n - \mathbf{p}_1)|. \tag{2}$$

This was added to help the ANN achieve accurate positioning.

These inputs are computed for each vehicle, where $\mathbf{p_1}$ is the ownship location. This formulation avoids the possibility of inadvertently assigning vehicles a role that might occur if the position relative to each other vehicle were given a dedicated input to the ANN. Figure 1 illustrates a possible solution to the formation problem and a sample computation of the inputs to one of the vehicles.

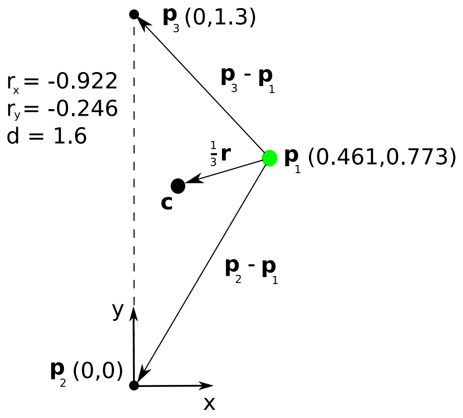


Figure 1. Illustration of a possible formation and a sample computation of the spatial parameters for the highlighted vehicle position (green). Notice that the combination of the three spatial inputs (**r** and *d*) represents a rotationally unique formation.

4 Quadrotor Kinematic Model

The Parrot ARDrone 2 quadrotor MAV was used for all real-world experiments described in this article. This 420-g vehicle is equipped with a 1-GHz 32-bit ARM Cortex A8 processor running a Linux operating system [22]. The default flight software provided by Parrot was overwritten by custom flight software implemented using Paparazzi UAS, an open-source flight control program [12, 27].

The ARDrone 2 has a typical X configuration widely used with quadrotors. These vehicles have two clockwise and two counterclockwise rotors. Controlling pairs of these rotors can create moments around the center of mass, causing the vehicle to rotate with six degrees of freedom. Controlling all motors collectively creates a net linear acceleration in the direction of the rotors aligned with the vehicle's z axis. Figure 2 shows the rotor directions and axis system for the ARDrone 2 quadrotor.

In this article we will only investigate rotations around the x and y directions, or *roll* and *pitch* respectively. Rotations around the z axis, or *yaw*, will be fixed for all flight tests. Additionally, all flights are limited to the xy plane, and a closed loop controller is used to maintain a fixed altitude of the vehicle. To ensure that the vehicle can maintain constant altitude, the maximum allowable pitch and roll attitude of the vehicle will be conservatively constrained to $\pm 20°$.

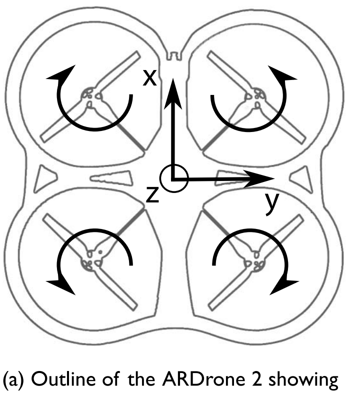
In this article the rotations around the x and y axes are generated by commanding a change in the rotational velocity of a pair of rotors. This will result in a moment around the appropriate axis, causing an eventual angular acceleration of the vehicle. Given an actuator pair setting ($\boldsymbol{\omega}$), the angular acceleration ($\dot{\Omega}$) can be determined using

$$\dot{\boldsymbol{\Omega}} = \mathbf{G}\boldsymbol{\omega} \tag{3}$$

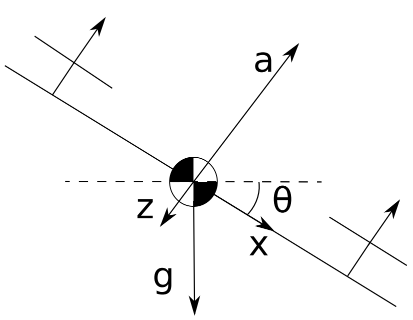
where $\boldsymbol{\Omega}$ is the angular rate of the vehicle and $\mathbf{G}$ is the actuator effectiveness matrix. The values of the parameters in $\mathbf{G}$ have been identified through experimentation with a real ARDrone 2 [29]. Additionally, we model the rotors of the vehicle with a first-order response with the time constant $\tau_\omega = 0.0306$.

Ignoring aerodynamic damping on the vehicle rotation, we can determine the vehicle attitude angles around the x axis (roll, or ϕ) and y axis (pitch, or θ) by simply integrating the angular rates. As all rotors have their thrust vector along in the z axis of the vehicle, when the vehicle is rotated w.r.t. Earth, the rotors will have to produce more thrust to counteract gravity and maintain altitude. The acceleration due to the force created by the rotors can be computed as $a = g/(\cos\theta \cos\phi)$, where g is acceleration due to gravity.

Now, the tilted thrust vector of the rotors will also induce a linear acceleration in the direction of the vector. The vehicle is not operating in a vacuum, so as it picks up speed it will encounter atmospheric drag. In practice this means that the vehicle will not have a constant linear acceleration



(a) Outline of the ARDrone 2 showing rotor directions and axis system.



(b) Resultant linear acceleration when assuming constant altitude and that the vehicle is at an angle w.r.t. to ground.

Figure 2. ARDrone 2 quadrotor.

as suggested above, but rather will achieve a fixed velocity given a particular attitude. To model this, we have added two drag terms: profile or rotor drag ($D_r \propto v$) caused by the rotors moving through the air, and parasitic drag ($D_p \propto v^2$), which is mainly caused by the non-lifting structure moving through the air [1]. The velocity kinematics are described in

$$\dot{\mathbf{v}} = \begin{bmatrix} -\sin\theta\cos\phi \\ \sin\phi \end{bmatrix} a - D_r|\mathbf{v}|\mathbf{v} - D_p\mathbf{v}. \tag{4}$$

This velocity can then be directly integrated to update the vehicle position. The kinematics are summarized in

$$\dot{\mathbf{x}} = \begin{bmatrix} -\frac{1}{\tau_\omega + h} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{\tau_\omega + h} & 0 & 0 & 0 & 0 & 0 & 0 \\ G_{\omega_x} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & G_{\omega_x} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -g & 0 & D_{x_1}|v_x| + D_{x_2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{g}{\cos\theta} & 0 & D_{y_1}|v_y| + D_{y_2} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \frac{u_x}{\tau_\omega + h} \\ \frac{u_y}{\tau_\omega + h} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \omega_{\max} \tag{5}$$

where $\mathbf{x} = [\omega_x, \omega_y, p, q, \phi, \theta, v_x, v_y]^T$, $\mathbf{u}$ is the output of the neural network, h is the time step of the simulation, and $\omega_{\max}$ is the maximum rotor setting. Now, the ARDrone, like many other MAVs, is equipped with a sensor suite to provide feedback on the real-world performance of the vehicle. It is important to accurately model the sensor response of the vehicle along with the eventual sensor noise [15]. For this article we use the observation matrix $\boldsymbol{y}$ as identified from real-world flight testing:

$$\mathbf{y} = \begin{bmatrix} R_p \sim \mathcal{N}(p, \sigma_p) \\ R_q \sim \mathcal{N}(q, \sigma_q) \\ \phi + \phi_0 \\ \theta + \theta_0 \\ v_x \\ v_y \end{bmatrix} \tag{6}$$

where σ_p and σ_q are the variances of p and q, respectively, and are set to 0.002 rad/s for this work; and ϕ_0 and θ_0 are the roll and pitch biases of the inertial navigation system (INS), which are randomly generated for each simulation run in the range $\pm 3°$. We performed all experiments in a motion capture arena, which is used to give position and velocity feedback to the vehicle. When combined with the onboard INS, the resultant velocity estimate is quite accurate. As a result, we will not add significant noise to this sensor reading, as there is unlikely to be much reality gap here. In addition to the sensor inputs, the simulation constants $\mathbf{G}$, τ_ω, $\mathbf{D_1}$, and $\mathbf{D_2}$ were all randomly perturbed $\pm 5\%$ from their ideal values at the start of each simulation and for each vehicle.

5 Control Schemes

To evaluate the effect of abstraction on SMC we will have evolution optimize two different solutions to the same problem, applying control to different levels in the vehicle dynamics scheme described in Section 4. These two control schemes will be referred to as the *low-level controller* and *high-level controller*.

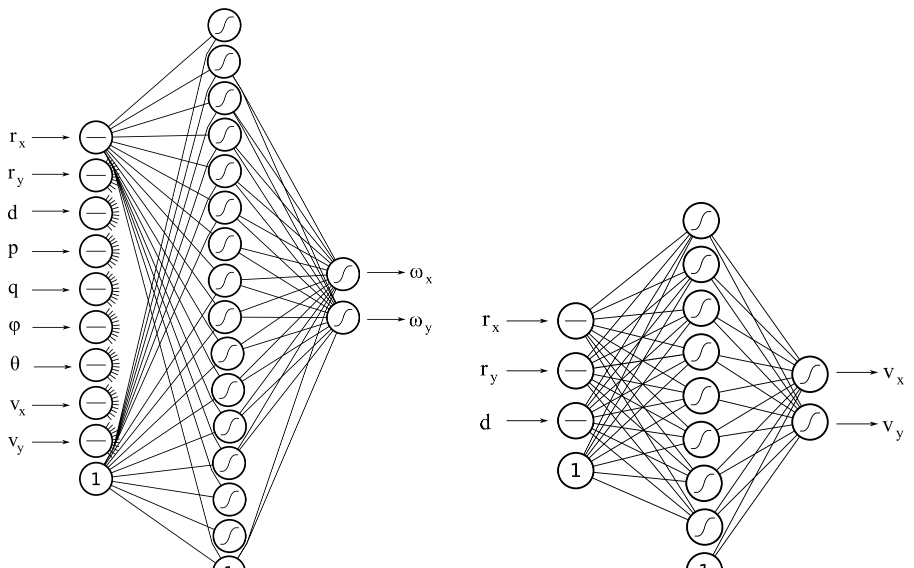
5.1 Low-Level Controller

For the low-level controller, the ANN is tasked with commanding the moments generated by the rotors. The output of the ANN is scaled to 15% of the total range of the rotor speeds, to limit the maximum rotational rate of the vehicle.

As the controller must achieve a particular formation by controlling the moments of the quadcopter, the controller must have a feedback mechanism to achieve stable flight. For this, the inputs to the ANN controller are the roll rate (p), pitch rate (q), roll angle (ϕ), pitch angle (θ), velocity along the x axis (v_x), and velocity along the y axis (v_y), as well as the formation inputs p_x, p_y, and d. The structure of the ANN for this controller, showing the inputs and outputs, can be seen in Figure 3a. We chose for the fixed structure a single hidden layer network with 15 hidden neurons.

5.2 High-Level Controller

To demonstrate a high level of abstraction, the evolved behavior will be tasked with controlling the velocity setpoint of the quadrotor. The output of the ANN is linearly scaled to the velocity limits of



(a) *Low level*: Tasked with controlling vehicle rotor pairs.

(b) *High level*: Tasked with controlling velocity.

Figure 3. Inputs, outputs, and structure of the neural controllers used in this article. A single-hidden-layer, fully connected ANN is used with a tanh activation function and an additional bias node at each layer. Shared inputs are the summed Cartesian components of the positions relative to the other vehicles in the formation (**r**) along with the summed absolute distances (d). Additional inputs are roll rate (p), pitch rate (q), roll angle (ϕ), pitch angle (θ), and velocity (**v**).

the vehicle, which in the case of this article is set as $v_{\max} = 0.5$ m/s. This setpoint is used as the input to the closed loop control architecture, which achieves the given velocity by controlling the individual motors for the quadrotor. This abstraction separates the optimization of the control system of the quadrotor from the optimization of the high-level robotic behavior. Designing control systems is a long-investigated area of research with many available methods to effectively control a wide range of systems. In its simplest form, a controller simply tries to reduce the error between a required state and the current state of a system. Much work has been done to make the control performance robust to unmodeled dynamics and environmental disturbances [16]. By definition, this makes the system more robust to the eventual reality gap between the dynamics of the simulated system and that of the real vehicle.

As a closed loop controller will ensure performance of the vehicle dynamics, we can simplify the model described in Section 4 to a first-order system with time constant $\tau_v = 0.3636$ as shown in

$$\dot{\mathrm{v}} = \begin{bmatrix} -\dfrac{1}{\tau_v + h} & 0 \\ 0 & -\dfrac{1}{\tau_v + h} \end{bmatrix} \mathbf{v} + \begin{bmatrix} \dfrac{u_x}{\tau_v + h} \\ \dfrac{u_y}{\tau_v + h} \end{bmatrix} v_{\max}. \tag{7}$$

As before, the value of τ_v used in the simulation is randomly perturbed by ±5% at the start of each simulation and is different for each vehicle.

This assumption greatly reduces the computational requirements of the simulator. As the time constant of the velocity system is large, we can use a large time step in the simulation as compared to that for the low-level controller, which requires a very small time step. Additionally, the number of inputs to the ANN is also reduced, to the formation inputs p_x, p_y, and d as shown in Figure 3b, as the closed loop controller will account for all closed loop control.

6 Evolutionary Algorithm

All evolutionary algorithms (EAs) share a common set of features, namely, *initialization*, *evaluation*, *selection*, and *recombination* [8]. As the ANNs used in this article have a fixed structure, the EA is tasked with optimizing the neural weights interconnecting the neurons of the network. A population of 100 individuals was generated with neural weights initialized, with a random number in the applicable range. Once initialized, all individuals were simulated using a kinematic simulator with model structure as described in Section 4. The three vehicles were initialized at hover in a 4 × 4-m area with a minimum intervehicular separation of 1.5 m. The centroid of this initial formation was set at the origin of the test area. All simulations were constrained to a maximum simulation time of 50 s. Simulations were cut short if any two vehicles came within 30 cm, which would constitute a real-world collision, or any vehicle exceeded the maximum allowable bank angle of ±20°. The resultant behavior was evaluated using the set of fitness functions to be minimized as described in

$$\begin{aligned} f_1 &= 1 - \frac{t_{\text{sim}}}{t_{\max}} \\ f_2 &= \sum_{n=1}^{k} |L_n - l_n| \\ f_3 &= \sum_{n=1}^{k} |\mathbf{v}_n|^2 \\ f_4 &= \begin{cases} 0, & |\mathbf{c}| < 3 \\ |\mathbf{c}|, & \text{else} \end{cases} \end{aligned} \tag{8}$$

where L is the required distance, l is the distance between the vehicles at the end of the simulation, both L and l are sorted in ascending order, $\mathbf{v}$ is the velocity vector of the MAV at the end of the simulation, and $\mathbf{c}$ is the location of the centroid of the triangle. t_{sim} is the simulation time at the end of the simulation, and $t_{\max}$ is the maximum allowable simulation time. All fitness functions are limited to the range [0, 100].

These equations attempt to cause the EA to optimize individuals that complete the desired formation (f_2) with zero resultant speed (f_3); stable behavior is indicated by long flights (f_1). Here f_4 tries to impose a restriction on the behavior for the formation to form within the 3×3-m box around the center of the simulation environment. This is to facilitate the real-world test, which must occur in a flight arena that is 10×10 m. The final fitness function provides a negative reinforcement when the simulation is prematurely terminated either due to a collision or due to exceeding the vehicle's maximum bank angle. f_2 and f_3 are similar to the function used in [14].

Once evaluated, the population is ranked using the widely used multi-objective Nondominated Sorting Genetic Algorithm II (NSGA-II) with crowding sorting applied in the fitness domain [6]. Selection is achieved using a tournament of eight randomly selected individuals from the sorted population, returning the highest-ranked individual.

Mutation was the only evolutionary operator used in this article, as some works have shown mutation-only evolution to be effective [30]. Each weight in the ANN was considered for mutation with a probability of 10%. Mutation consisted of a random perturbation of the previous value using

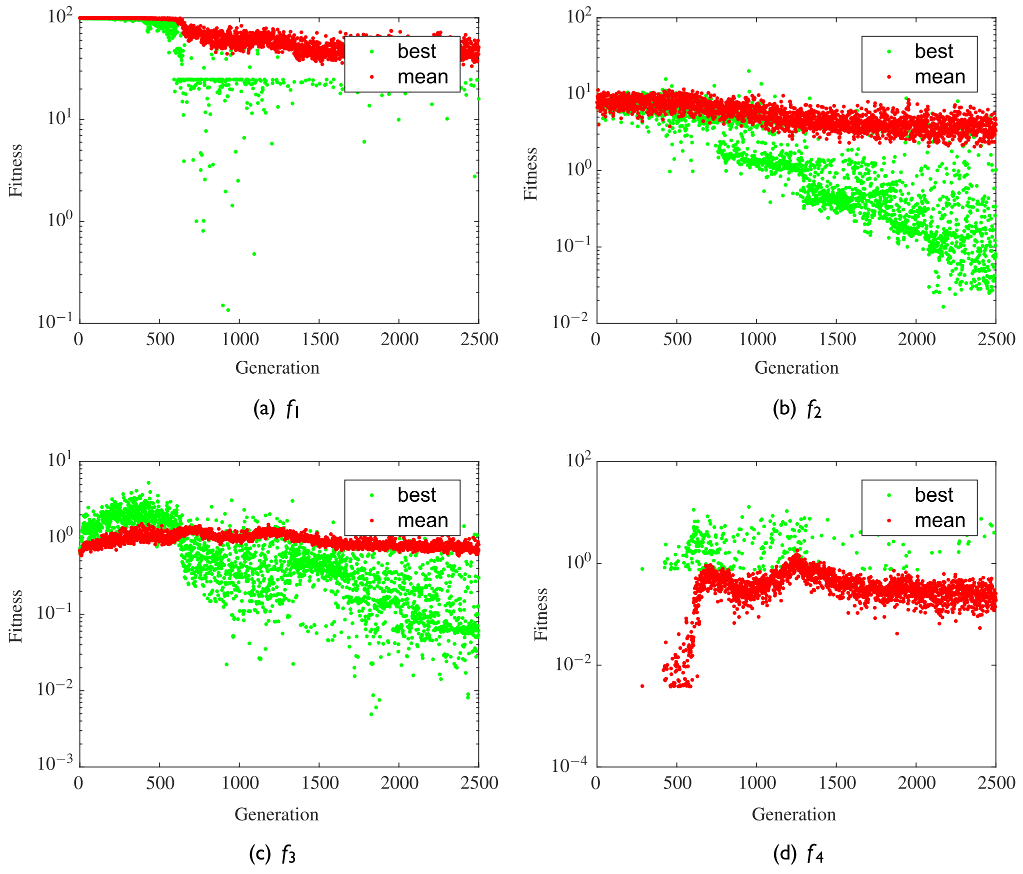


Figure 4. Progression of the performance of the best individual and the mean of the population during evolution for the low-level controller.

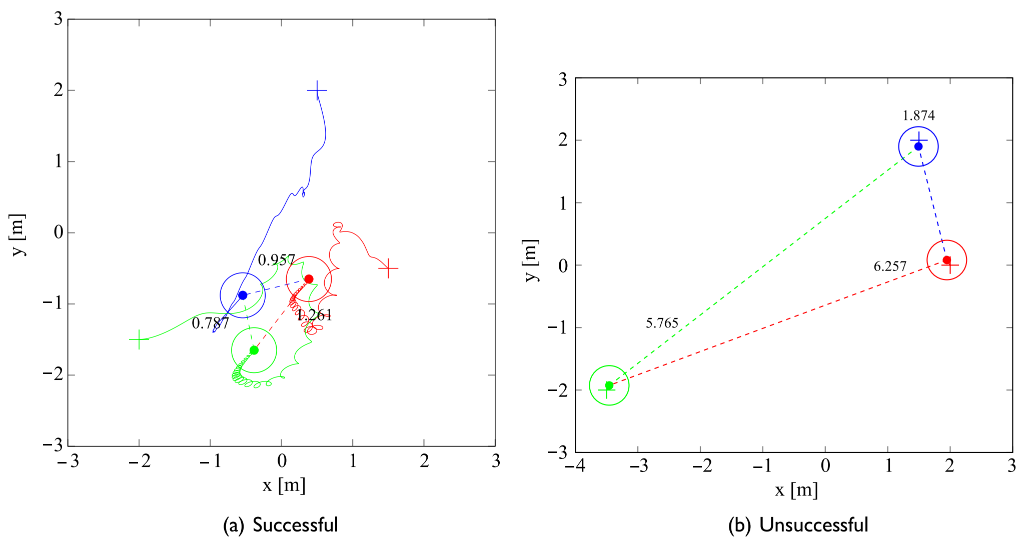


Figure 5. Ground track of one successful and one unsuccessful flight of the low-level controller. The length of each side is shown in a numerical label, + marks the start location, and the circle with a dot at the center marks the end location with the diameter of the vehicle to scale.

roulette selection; small perturbations have a higher probability than larger ones. All weights were kept in the range $[-1, 1]$ except for the weights from the bias nodes, which were kept in the range $[-5, 5]$.

6.1 Low-Level Controller Optimization

Figure 4 shows the progression of the evolution for the low-level controller. Here the best individual is the individual with the largest Euclidean distance from the origin in the four-dimensional fitness space. Notice that around 500 generations were needed before basic flight capability was evolved, after which there was slow but steady improvement in the performance, with the total formation error around 3 cm after 2500 generations. The total simulation time was about 18 h on a standard desktop PC with a simulation time step of 0.01 s, which is necessary to simulate the fast dynamics of the actuators.

The individual with the lowest average fitness was evaluated in more detail by a validation run of 250 different initial conditions. During the validation run, the simulation was not cut short if a collision occurred. A formation is considered accurate when the summed errors of the lengths is below 0.15 m, or 5-cm average error over the final 2 s of the flight. The results show that 79% of runs resulted in a successful triangle formation within 50 s; if we ignore crashes, this increases to 90%. Of these successful runs, the mean error was 0.0189 m with a standard deviation of 0.01. Of the runs that would not have completed the formation even when not considering crashes, the main failure case was that the vehicles exceeded the maximum attitude limits. Figure 5 shows one of the successful runs of the formation behavior and one case where a collision would have occurred.

Interestingly, it was observed that the formation formed was always oriented with the same rotation to the axis system. This suggests that the behavior is trying to change the output of the network, which would result in a unique set of formation inputs ($\mathbf{r}$ and d) rather than a rotationally invariant formation.

6.2 High-level Controller Optimization

Figure 6 shows the performance of the best individual from each generation of the evolutionary optimization for this problem. This figure shows that evolution gradually reduces the error in the

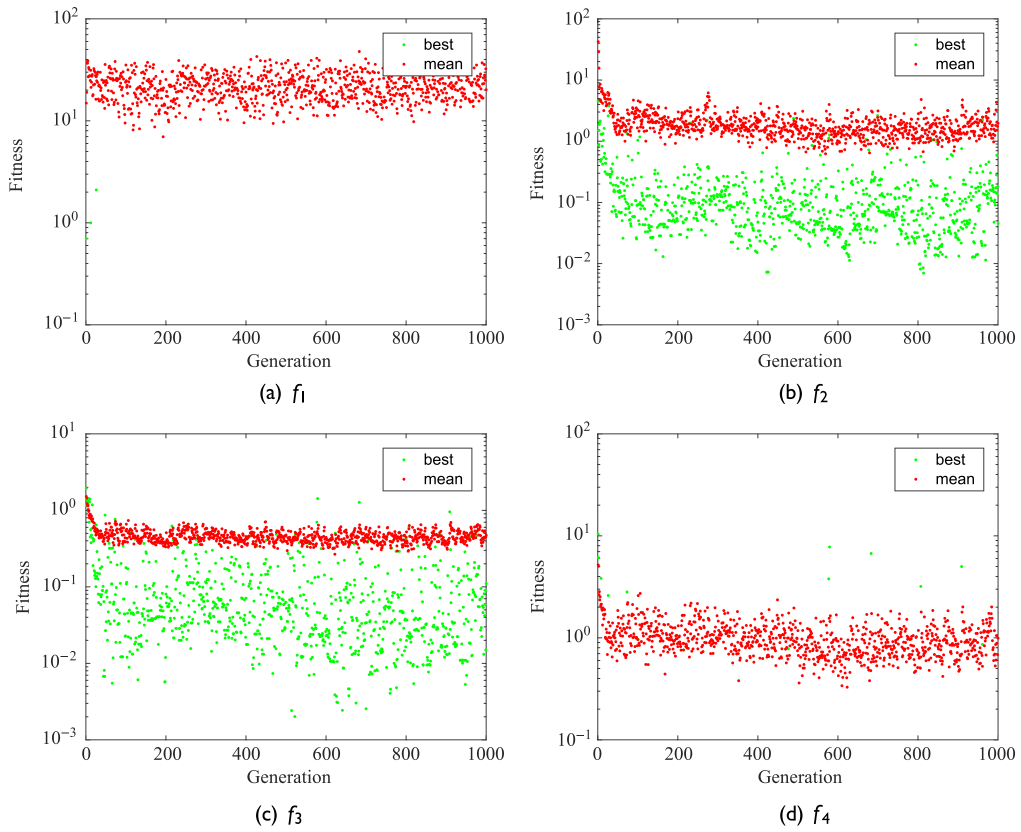


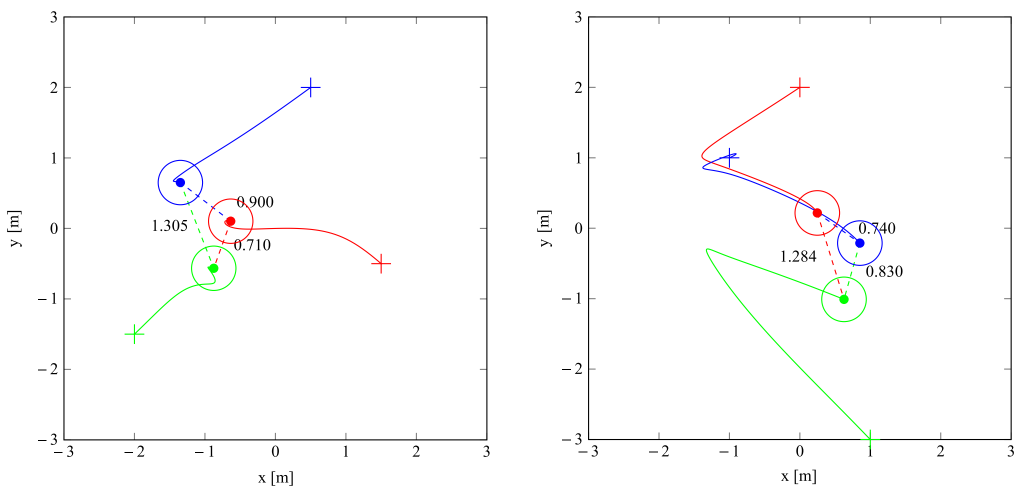
Figure 6. Progression of the performance of the best individual and the mean of the population during evolution for the high-level controller.

final vehicle distances and the final velocity. It also shows that the behavior does not guarantee a collision-free flight for all initial conditions. After 1000 generations the average error of each length of the formation is about 2 cm. This was significantly faster than the low-level controller. Additionally, the total optimization time was 30 min on a standard desktop PC with a simulation time step of 0.2 s facilitated by the slower dynamics of the velocity controller's response.

As before, the behavior for the best controller was evaluated by a validation run of 250 different initial conditions. The results show that 84% of runs resulted in a successful triangle formation within 50 s; again, if we ignore crashes, this value increases to 98%. Of these successful runs, the mean

Table 1. Success rate of the formation task.

	Low level	High level
With all termination conditions	79%	84%
Ignore collision	90%	98%
Ignore collision and t_{max} = 100 s	91%	100%
Average formation error	0.0189 m	0.0222 m



(a) Collision free (b) Collision at (-1.25, 1)

Figure 7. Ground tracks of a collision-free flight and a flight that would have ended in a collision, of three ARDrones performing the asymmetric formation task. The length of each side is shown in a numerical label, + marks the start location, and the circle with the dot at the center marks the end location with the diameter of the vehicle to scale.

error was 0.0222 m with a standard deviation of 0.0262 m. In the remaining 2% of the runs, the triangle was not formed after the given 50 s but was in fact formed later; if we extend the 50-s simulation to 100 s and ignore collisions, we find a 100% success rate. These results are summarized in Table 1. Figure 7 shows one of the successful runs of the formation behavior and one case where a collision would have occurred.

An interesting observation was that the formation created was observed to be in the same orientation to the axis system, a similar solution strategy to that seen in the low-level optimization. The orientation of the formation for the two controllers is different, and in fact, each time the behavior was re-evolved, a different orientation was optimized, suggesting that there is no special geometric optimum, but rather the output is being regulated to maintain a fixed combination of the inputs p_x, p_y, and d.

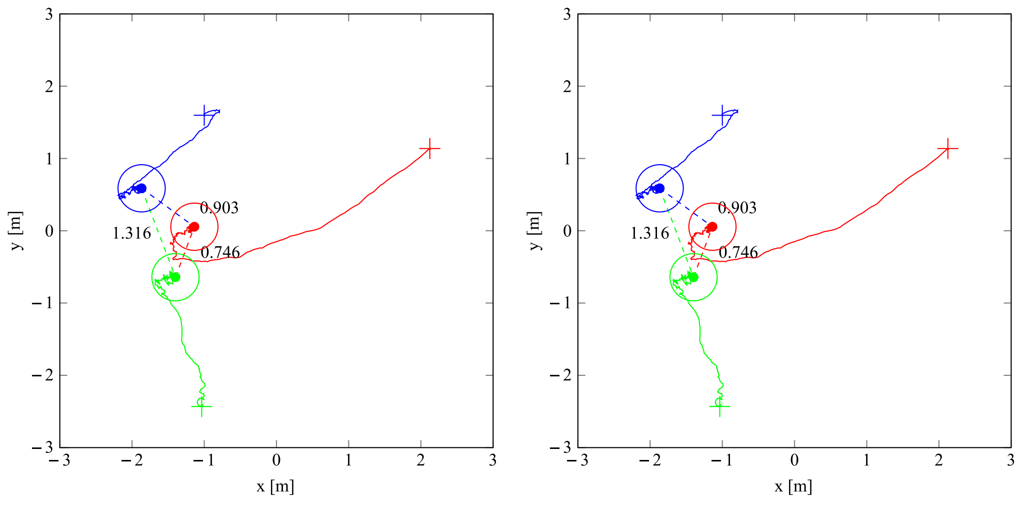
7 Flight Test Results

Moving from the simulated world to the real, the behaviors shown above were implemented on a swarm of three ARDrones. Flights were performed in a 10 × 10-m flight arena, and the flight path of the vehicles was captured using an Optitrack motion camera system which was used to determine the relative location inputs to the neural network [19].[3] For the first set of tests, as in simulation, the three vehicles were initialized at random in a 4 × 4-m area in the flight arena with the centroid of the initial formation at the origin of the arena, just as in simulation.

When testing the low-level behavior it quickly became apparent that there was a significant disparity map. When switching from hover into autonomous mode, the vehicles would quickly lose stability and exceed their 20° attitude limits, ending the trial. The root cause of the gap was not immediately clear and was quite difficult to pinpoint, given the multiple inputs and the fast dynamics of the vehicle.

The high-level controller did however successfully cross the reality gap; the flight path of one flight is shown in Figure 8 compared with a simulated flight with the same initial position.

3 A video of some of the flight tests can be found at the following URL: https://www.youtube.com/playlist?list=PL_KSX9GOn2P8ru70sZZj0H6K2R7lWCoDx.



(a) Real world (b) Simulation

Figure 8. Ground track of the real-world flight test and the simulated flight for the same initial positions. The length of each side is shown in a numerical label, + marks the start location, and the circle with a dot at the center marks the end location with the diameter of the vehicle to scale.

The flight path observed in reality was very similar to that predicted in simulation; however, deviations can be seen when the vehicles come into close proximity with each other. This was due to aerodynamic interactions of the rotor downwash, something that was not modeled. The closed-loop controller was however able to cope, keeping the vehicles close to the desired formation. Figure 9 shows the commanded velocity of the ANN and the true vehicle velocity along with the result of the simulation.

In contrast to the simulation, the real-world quadrotors have clear velocity tracking errors and oscillations, in part due to the aerodynamic interactions. These tracking errors represent a significant reality gap. Despite this, the observed high-level behavior is very similar to that seen in simulation,

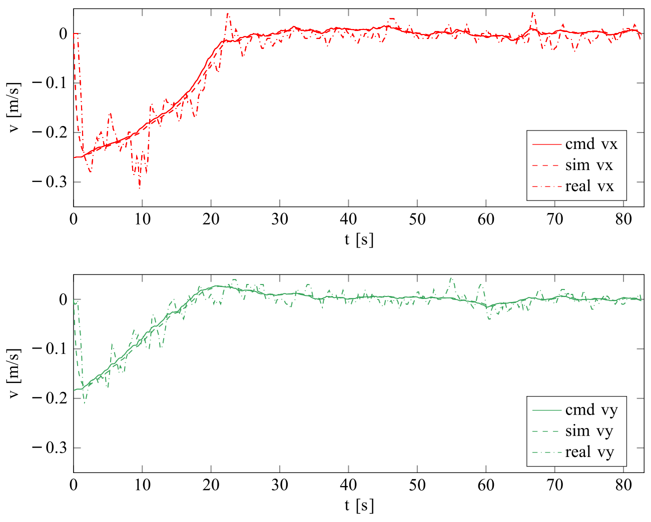


Figure 9. Tracking performance of the velocity controller on the simulated and real ARDrone 2 given the same velocity command, highlighting the reality gap.

Table 2. Success rate of high-level controller when simulated with varying second-order vehicle dynamics.

	Success rate (%)		
ω_0	$\zeta = 0.9$	0.6	0.3
3.854	96.0	95.6	97.2
2.854	95.6	94.0	97.2
1.854	90.4	94.4	96.8
0.854	87.2	85.2	0

the correct formation being achieved with an average formation error of 0.034 m, only a 10-cm deviation from that in simulation. The resultant stable formation in the face of these unmodeled external disturbances highlights the robustness of the optimized solution.

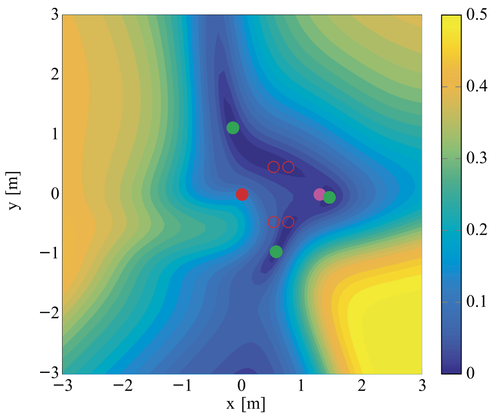
Looking more closely at Figure 9, we can see some overshoot and oscillation in the real-world velocity controller that was not seen in simulation. Unlike the first-order system used to model the velocity response of a vehicle in simulation, the real-world controller resembles more a second-order system with a short rise time. This difference in simulated and real dynamics is one clear source of the reality gap. To investigate the robustness of the optimized behavior of this controller, we implemented a second-order system in our simulator with the following transfer function: $\omega_0^2/(s^2 + 2\zeta\omega_0 + \omega_0^2)$. The natural frequency (ω_0) and damping ratio (ζ) were varied and tested with the previously optimized behavior. Table 2 summarizes the results of this investigation. The parameters identified to best fit the observed velocity response were $\omega_0 = 3.854$ and $\zeta = 0.9$. Interestingly, the behavior was robust to a wide range of velocity dynamics, breaking down when the natural frequency dropped below 0.85. The damping ratio had little effect on the final formation performance, but did affect the number of collisions as the behavior became more oscillatory.

8 Analysis of the Sensory-Motor Coordination

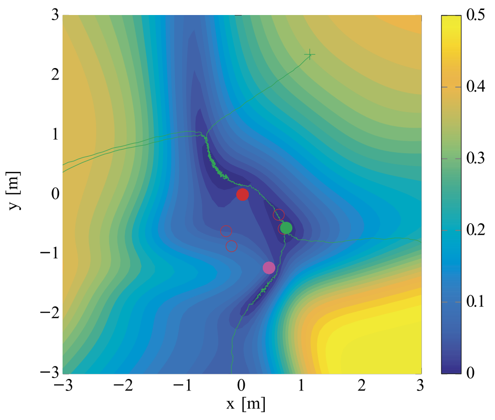
To analyze the effect of abstraction on the extent to which evolved robots exploit their environment and make use of SMC, we must first dive deeper into the optimized behavior. Firstly we will investigate an observation made in [14]. There, although the task was solved by the genetic optimization during analysis, researchers fixed two of the robot satellites so that one leg of the formation was correct. When the third satellite was left free to move, it did not settle into the correct location to complete the formation. This led the authors to an interesting hypothesis: Perhaps the asymmetric formation could only be reached if all three satellites were free to move. Here we will investigate if we can observe a similar phenomenon for our specific evolved solution, and evaluate whether SMC plays a role in successful formation flight.

In the flight tests described in this article, it was observed that all successful formations resulted in a triangular formation with the same rotational orientation to the Cartesian axis system. The orientation can be seen in Figure 7 and Figure 8. Given the location of any two vehicles separated by any of the three lengths of the formation, there are four possible locations for the third vehicle that would satisfy the task given. The optimized solution seems to only utilize one of these. Additionally, there is a unique set of inputs to describe every possible rotational orientation of the triangle formation. The fact that the formation always converges to the same orientation suggests that the solution optimized in evolution is to define the output of the ANN so that the commanded velocities force the vehicle into a specific relative location where the values for the inputs p_x, p_y, and d produce a minimum-velocity command that is stable to disturbances.

If we fix two of the vehicles as done in [14] and evaluate the velocity output of the ANN of the third vehicle when in various locations around the arena, we can get an idea of the solution strategy,



(a) Velocity map when two fixed vehicles are fixed in the wrong rotational orientation to facilitate the formation.



(b) Velocity map when two fixed vehicles are fixed in the correct rotational orientation to facilitate the formation. Overlaid in green are ground tracks of all real-world flights. This shows that the real-world performance mirrors what is expected from simulation quite well, despite a clear reality gap.

Figure 10. Basin of attraction for one vehicle given that the other two vehicles (shown in red and blue) are fixed in space. The hollow red circle highlights the solution to the formation problem, and the green dots show the stable attractor points. The color bar represents the magnitude of the commanded velocity, $|\mathbf{v}_{\mathbf{cmd}}|$.

as shown in Figure 10. Figure 10a shows the velocity map of the third vehicle when the other members of the swarm are aligned with the x axis, an orientation not seen when the vehicles were all free. It can be seen that there are three stable minima in this basin of attraction and none of them satisfy the given formation. Notably, although the highlighted spots are stable points when the two other vehicles are free, the commanded velocities of the other two vehicles are nonzero, showing that this formation is not stable. In Figure 10b, the two fixed vehicles are situated in a similar orientation to that seen when the vehicles were all free. This basin of attraction shows only one stable minimum, which does satisfy the formation. We also performed real flight tests with two vehicles fixed in the correct orientation, with Figure 10b showing that the ground tracks of the real flights overlap almost exactly with this velocity field. This confirms the hypothesis that output shaping converges to a single relative position and thus a single set of inputs p_x, p_y, and d, and is a strong indication that the high-level controller is in fact using the SMC to complete the task.

When considered from the point of view of the decentralized swarm, as mentioned, when two vehicles are fixed, it is possible to enter a local minimum that is not observed when all are free to move. This would suggest that the solution strategy is based on the fact that the swarm is homogeneous and therefore there is an inherent modeling of the motion of the other members of the swarm. This implicit modeling can be also expressed as a form of environmental exploitation, which is a hallmark of SMC. Given Nolfi's suggestion that the emergence of behavioral attractors is indicative of SMC [21], the high-level controller would appear to effectively exhibit SMC, albeit on a more abstract level. Although the evolution has no access to the low-level control or sensory inputs, the resultant behavior was still able to exploit the implicit knowledge that the other members of the swarm would implicitly cooperate to solve the task.

9 Discussion

It was not the goal of this article to suggest that low-level primitives should not be used, or that it is not possible to evolve complex behavior using these primitives. Although we were unable to bridge the reality gap for our low-level controller, that does not mean that it is not possible. Likely, revising the modeled dynamics and the use of a recurrent neural network or incremental evolution may provide the tools necessary to cross the gap, but that is in itself one argument for using abstraction. We investigated what effect abstracting away from low-level primitives has on the effectiveness of the optimized behavior and the optimization process itself. We have shown that the abstraction of actions can reduce the effective dynamics of the robotic platform, which, in combination with the reduced simulation fidelity required, can reduce the computational load of simulations, speeding up optimization times. Shifting the action space to a higher level also helps to reduce the search space of the optimization, reducing the bootstrapping problem. Importantly, we have demonstrated that abstraction does not necessarily prevent the emergence of SMC as some have suggested.

This article has used abstraction on the actions, but the authors expect similar success when it is applied to the sensory inputs, where large gains can be seen in the use of preprocessed visual inputs. This however comes with a few caveats: by preselecting a subset of abstracted inputs for evolution, we are effectively reducing the search space; that is beneficial for the optimization, but also removes degrees of freedom, which can be detrimental. Open questions are: how do we select the level of abstraction; how do we determine the inputs necessary to complete the task at hand; how do we abstract away from raw inputs without injecting unnecessary amounts of designer bias? We hope to answer some of these questions in future work.

Reflecting on the evolution of the behavior itself, it is interesting to note that neither behavior found solutions that achieved the formation while guaranteeing collision-free flight. Collisions were in fact the largest source of failed flights. This may be due to the fact that collision was used to preemptively end the simulation without an explicit negative reward. It might have been more effective to impose some explicit fitness change when a collision occurred.

The fact that the optimization selected a rotationally fixed solution to the formation problem may be a result of the selection of the fitness functions. The fitness functions used in this article tried to promote the formation of a triangle with three fixed-length sides, but gave no specification of the required orientation. Given that freedom, the optimization simply found the simplest solution to the problem, which in this case is a formation resulting in a unique combination of the relative position inputs.

10 Conclusion

In this article we have investigated the application of abstraction on the inputs and outputs of a neural network controller within the evolutionary robotics paradigm. Two flight controllers using differing levels of abstraction were evolved to solve an asymmetric triangle formation task with a swarm of three MAVs. Both controllers were able to control the vehicles to complete the swarming task; however, only the controller using the higher level of abstraction successfully bridged the reality gap and worked on the real robot. We have shown that abstraction can be a useful tool in making evolved behavior robust to the reality gap. Additionally, abstraction has been shown to reduce the level of fidelity required of the simulator, as low-level interactions of the sensors and actuators are modeled by a higher-level system, thereby reducing the optimization time. Most importantly, we have demonstrated that sensory-motor coordination, a typical emergent phenomenon of reactive agents, is not necessarily lost when abstracting away from the raw inputs and outputs, as some have suggested, but is rather is simply shifted to a higher level of abstraction.

The use of abstraction has the potential to accelerate the development of more complex robotic behaviors than have been previously attempted. There are however some open questions that must be answered before it can be widely used. The most significant of these is how the abstraction is to be selected. By only providing the evolutionary optimization with a subset of possible sensory inputs or actions, the user may inadvertently limit the process or guide it away from a potential optimum. Future work will investigate how the optimal input and output set can be automatically determined or optimized within the evolutionary process.

References

1. Bangura, M., & Mahony, R. (2012). Nonlinear dynamic modeling for high performance control of a quadrotor. In *Australasian Conference on Robotics and Automation (ACRA 2012)* (pp. 1–10). Wellington, NZ: Australian Robotic and Automation Association.
2. Beer, R. D. (1996). Toward the evolution of dynamical neural networks for minimally cognitive behavior. In P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, & S. W. Wilson (Eds.), *From animals to animats 4: Proceedings of the 4th International Conference of Simulation of Adaptive Behavior (SAB1996)* (pp. 421–429). Cambridge, MA: MIT Press.
3. Chiel, H. J., & Beer, R. D. (1997). The brain has a body: Adaptive behavior emerges from interactions of nervous system, body and environment. *Trends in Neurosciences*, *20*(12), 553–557.
4. Clark, A. (1998). *Being there: Putting brain, body, and world together again*. Cambridge, MA: MIT Press.
5. Clarke, A., & Thornton, C. (1997). Trading spaces: Computation, representation, and the limits of uninformed learning. *The Behavioral and Brain Sciences*, *20*(1), 57–66.
6. Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197.
7. Dewey, J. (1896). The reflex arc concept in psychology. *The Psychological Review*, *3*(4), 357–370.
8. Eiben, A. E., & Smith, J. E. (2015). *Introduction to evolutionary computing* (2nd ed.). Berlin, Heidelberg: Springer-Verlag.
9. Floreano, D., & Mondada, F. (1994). Automatic Creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In D. Cliff, P. Husbands, J.-A. Meyer, & S. Wilson (Eds.), *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From animals to animats 3* (pp. 421–430). Cambridge, MA: MIT Press.
10. Gibson, J. J. (1950). *The perception of the visual world*. Boston, Houghton Mifflin.
11. Gigliotta, O., & Nolfi, S. (2007). Formation of spatial representations in evolving autonomous robots. In *Proceedings of the 2007 IEEE Symposium on Artificial Life, CI-ALife 2007* (pp. 171–178).
12. Hattenberger, G., Bronz, M., & Gorraz, M. (2014). Using the Paparazzi UAV system for scientific research. In G. C. H. E. de Croon, E. van Kampan, & C. de Wagler (Eds.), *International Micro Air Vehicle Conference and Competition 2014* (pp. 247–252). Delft, Netherlands: IMAV.
13. Izzo, D., & Pettazzi, L. (2007). Autonomous and distributed motion planning for satellite swarm. *Journal of Guidance, Control, and Dynamics*, *30*(2), 449–459.
14. Izzo, D., Simões, L. F., & de Croon, G. C. H. E. (2014). An evolutionary robotics approach for the distributed control of satellite formations. *Evolutionary Intelligence*, *7*(2), 107–118.
15. Jakobi, N. (1997). Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behaviour*, *6*(2), 325–368.
16. Love, J. (2007). *Process automation handbook* (1st ed.). No. 800: Production & process engineering. London: Springer.
17. Lungarella, M., & Pfeifer, R. (2001). Robots as cognitive tools: An information theoretic analysis of sensory-motor data. In *Proceedings of the 2001 IEEE RAS International Conference on Humanoid Robots* (pp. 245–252). New York: IEEE.
18. Montebelli, A., Herrera, C., & Ziemke, T. (2008). On cognition as dynamical coupling: An analysis of behavioral attractor dynamics. *Adaptive Behavior*, *16*(2–3), 182–195.
19. Natural Point Inc (2014). Optitrack. Available at www.naturalpoint.com/optitrack/ (accessed on 2015-07-23).
20. Nolfi, S. (1998). Evolutionary robotics: Exploiting the full power of self-organization. *Connection Science*, *10*(3–4), 167–184.
21. Nolfi, S. (2002). Power and the limits of reactive agents. *Neurocomputing*, *42*(1–4), 119–145.
22. Parrot. ARDrone 2. Available at www.ardrone2.parrot.com/ (accessed on 2016-04-07).
23. Pfeifer, R., & Bongard, J. (2007). *How the body shapes the way we think: A new view of intelligence*. Cambridge, MA: MIT Press.
24. Pfeifer, R., & Scheier, C. (1997). Sensory-motor coordination: The metaphor and beyond. *Robotics and Autonomous Systems*, *20*, 157–178.

25. Pfeifer, R., & Scheier, C. (1999). *Understanding intelligence*. Cambridge, MA: MIT Press.

26. Piaget, J. (1952). When thinking begins. In *The origins of intelligence in children* (pp. 25–36). New York: International University Press.

27. Remes, B., Hensen, D., van Tienen, F., de Wagter, C., van der Horst, E., & de Croon, G. (2013). Paparazzi: How to make a swarm of Parrot AR drones fly autonomously based on GPS. In *Proceedings of the International Micro Air Vehicle Conference and Flight Competition* (pp. 17–20). Toulouse, France: IMAV.

28. Scheper, K. Y. W., & de Croon, G. C. H. E. (2016). Abstraction as a mechanism to cross the reality gap in evolutionary robotics. In E. Tuci, A. Giagkos, M. Wilson, & J. Hallam (Eds.), *From animals to animats 14: Proceedings of the 14th International Conference on Simulation of Adaptive Behavior (SAB2016)* (pp. 280–292). Cham, Switzerland: Springer International Publishing.

29. Smeur, E. J., Chu, Q. P., & de Croon, G. C. (2016). Adaptive incremental nonlinear dynamic inversion for attitude control of micro aerial vehicles. In *AIAA Guidance, Navigation, and Control Conference* (pp. 1–16). Reston, VA: AIAA.

30. Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, *87*(9), 1423–1447.