Evolving State Machines and Neural Networks for an Autonomous UGV

Anonymous Author(s)

ABSTRACT

Unmanned ground vehicles (UGVs) are well suited to tasks that are either too dangerous or too monotonous for people. For example, UGVs can traverse arduous terrain in search of disaster victims. However, it is difficult to design these system so that they perform well in a variety of different environments. In this study, we evolve controllers and physical characteristics of a UGV with transformable wheels to improve its mobility in a simulated environment. The UGV's mission is to visit a sequence of coordinates while automatically handling obstacles of varying sizes by extending wheel struts radially outward from the center of each wheel. Evolved finite state machines (FSMs) and artificial neural networks (ANNs) are compared, and a set of controller design principles are gathered from analyzing these experiments. Results show comparable performance between FSM and ANN controllers, but the evolved strategies differ. Furthermore, we show that a UGV's controller and physical characteristics can be effectively chosen by examining results from evolutionary optimization.

CCS CONCEPTS

- **Computer systems organization** → **Evolutionary robotics**;
- **Computing methodologies** → *Evolutionary robotics*;

KEYWORDS

evolutionary robotics, artificial neural network, finite state machine, adaptive, unmanned ground vehicle

ACM Reference Format:
Anonymous Author(s). 2018. Evolving State Machines and Neural Networks for an Autonomous UGV. In *Proceedings of the Genetic and Evolutionary Computation Conference 2018 (GECCO '18).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Autonomous unmanned ground vehicles (UGV) provide an excellent solution to tasks that requires searching or monitoring in environments deemed too remote or dangerous for humans. Consider *search and rescue*: after a natural disaster a UGV can be used by first responders to help locate victims in unstable and hazardous locations. Compared with an unmanned aerial vehicle (UAV), a UGV has a longer operating duration, can carry heavier sensors, and can search in narrow and covered places. Ideally, a heterogeneous swarm of UAVs and UGVs would coordinate to gain the advantages of both search modes [11]. In this paper, however, we limit our investigation to UGVs.

Ensuring that a UGV can handle many different types of terrain is an ongoing challenge. Researches have invented several different methods for addressing the issue of mobility in varied terrain. Specifically, robots have been designed with treaded wheels, tracks, legged-wheels (wheels are rimless, wheel spokes make contact with the ground), wheeled-legs (wheels are on the end of legs and the suspension is potentially actively actuated), or transformable wheels [4, 6, 9, 15, 17]. Although these systems provide an advantage over traditional wheeled robots, it is not clear that they are optimally taking advantage of their unique features.

The device in this study, the *Adabot* (see Figure 1), includes transformable wheels that can smoothly be converted from a round wheel, to a wheel with tire *studs*, to a legged-wheel. Our device transitions to a legged-wheel by extending wheel struts radially outward from the center of the wheel (see Section 3 for details). Adabot has been optimized using an evolutionary algorithm such that its physical characteristics and its controller are better able to handle terrain that includes obstacles of varying sizes. In previous work, a similar system was optimized to maximum forward velocity. The present study differs in two main ways: (1) here we evolve controllers for a more difficult task: *way-point following*, and (2) we analyze results from evolving two controller types.

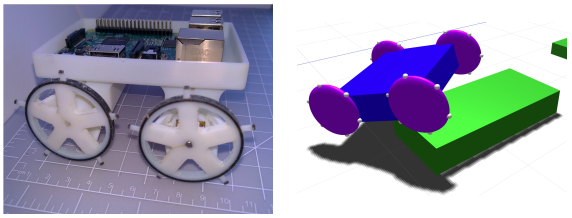
In this study, we evolve the robot's chassis dimensions, its wheel radius, the number of wheel struts (described in Section 3), along with either a finite state machine (FSM) controller or an artificial neural network (ANN) controller. The best evolved FSMs and ANNs are analyzed and compared. For this initial work, to ensure that we are able to effectively analyze the ANN, the network only has three input nodes, zero hidden nodes, and three output nodes. The inputs are fully connected to the outputs. In fact, the network is only slightly more complex than a Type 2 Braitenberg vehicle [2]. Conclusions drawn from our analysis are used to create a set of design principles for a new controller that takes advantage of both techniques. In particular, it is attractive to design a controller that is not a black-box like an ANN but less rigidly defined than an FSM.

The results presented in this study demonstrate how a UGV can be optimized to improve robustness. Specifically, evolving Adabot has made it more robust in the presence of varied terrain. The techniques and concepts presented in

GECCO '18, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM. . . $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

this study can be applied to the optimization of any physical system. The source code has been made available on GitHub[1].



(a) 3D Printed Prototype (b) Simulated Device

Figure 1: (a) An image of the 3D printed Adabot prototype, and (b) a visualization of the Adabot in its simulated environment.

2 RELATED WORK

Robots are well suited to tasks that are too mundane, precise, or dangerous for humans. Increasingly, autonomous robots are being developed to operate in rugged and dynamic environments. Such terrain is particularly difficult for wheeled UGVs, which has prompted the design of many different suspensions and wheel configurations [18]. The challenge for these devices is to successfully navigate an area without prior knowledge of the obstacles and ground consistency. Moreover, robotics engineers must choose values for all free parameters (e.g., the chassis and wheel dimensions).

In the field of evolutionary robotics (ER), an evolutionary algorithm (EA) optimizes free variables of a given system [19]. ER methods have been successfully applied to many different types of robotic systems (aerial, aquatic, walking, etc.). For example, Clark et al. [3] used differential evolution to evolve an adaptive neural network for a robotic fish, and Moore et al. [12] evolved hierarchical controllers for segmented worm-like animats. Although evolution has been regularly utilized at an abstract level to optimize wheeled-robot navigation processes (for example, see [5]), it has not often been used to directly evolve UGVs. However, as identified by Bongard [1], and in the work presented here, ER methods can be effectively applied to wheeled robots.

A large number of ER studies utilize ANNs to control mobile robots. This includes Sims [20] *Evolving Virtual Creatures*, which is considered one the the first ER works. ANNs provide several benefits when using an evolutionary method. First, since ANNs are so-called universal approximators [8], evolution often produces *novel* and sometimes unintuitive results that may not have been found when creating a controller by hand [1]. And second, ANNs require a minimal amount of user design. Specifically, an evolutionary algorithm can automatically decide the importance of each input (sensor values) in the calculation of each output (actuation mechanisms) [21]. The primary disadvantage of using an ANN is that it is considered a black-box system. That is, *how* an ANN achieves its results is not often clear or analyzed. Recently, however, some researchers have attempted to extract state machines from evolved neural networks [22]. In the aforementioned study, Yaqoob et al. [22] automatically created a state machine with the same properties of an evolved spiking neural network.

[1] <link removed to preserve anonymity>

3 ADABOT

In this section, we describe the prototype Adabot hardware, including the transformable wheel mechanism, along with the simulation environment utilized during evolutionary optimization.

Hardware The Adabot, pictured in Figure 1, is a prototype device that includes a Raspberry Pi 3 Model B (RPi) as its main control board. The RPi was chosen for its ability to run the Robot Operating System (ROS) [14], which Adabot uses to manage its software systems. The size of an RPi constrains the minimum dimensions of the Adabot's chassis. Specifically, the chassis must be at minimum 8 cm by 8 cm. Table 1 lists all configurable parameters for Adabot's physical characteristics, where *WheelBase* and *TrackWidth* denote the distance between the front and rear axles and the lateral distance between wheels, respectively, and *StrutCount* parameter indicates the number of struts per wheel.

Table 1: Adabot Physical Parameters

Name	Range
WheelBase	8 to 16 cm
TrackWidth	8 to 16 cm
WheelRadius	2 to 3 cm
StrutCount	0 to 7

Each of the four wheels is driven by its own DC gear-motor with magnetic encoders. Likewise, each wheel includes a set of struts that can be extended and retracted by a linear servo. For sensing, Adabot includes three forward facing distance sensors and an IMU (3-axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer). Finally, it uses a 2.4 GHz wireless communication module and is powered by a 2200 mAh battery pack, which provides roughly two hours of operating time.

Strut Extension Figure 2 depicts the strut extension process. This mechanism enables the wheel to exhibit a range of characteristics. With the struts fully retracted, the wheels operate conventionally; when extended a small amount the struts act as tire studs; and with the struts fully extended each wheel resembles a legged-wheel. The radius of the wheel and the number of struts are both free parameters that can be evolved during optimization. For the following discussion, the most important aspect of this mechanism is that the maximum extension of the struts is equal to 1 cm minus the wheel's radius ($MAXEXT = WheelRadius - 1$ cm). For a more detailed discussion of Adabot's software and wheel extension mechanism, and an example of evolving Adabot with ROS and Gazebo (a simulation environment tightly coupled with ROS) [10] see our preliminary study <citation removed to preserve anonymity>.

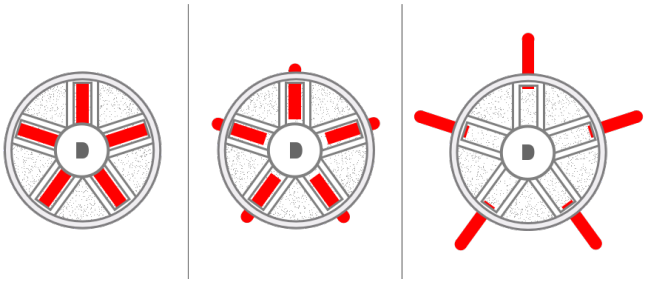


Figure 2: An illustration of the Adabot's wheel extension mechanism, where the struts are fully retracted (left), partially extended (middle), and fully extended (right).

Simulation An image of the simulation environment is shown in Figure 3. The environment is populated by generating 40 boxes with random dimensions, positions, and densities. If a newly generated box collides with another box it is removed from the simulation. We see on average 31 boxes in the environment. Box heights are in the range 2 to 5 cm, which is high enough compared to the potential *WheelRadius* values that most traditional wheeled robots will have difficulty climbing them [15]. Moreover, rather than each box being in a fixed position, it is possible for the Adabot to push a box (depending on its size and density).

For this study, we are using the Dynamic Animation and Robotics Toolkit (DART)[2]. DART is specifically designed for robotics applications (unlike the Open Dynamics Engine[3], which we've used in the past), and is comparable in speed (if not faster) than common alternatives [13]. In this study, we do not simulate noise, and we assume that the robot has direct access to its global position and rotation. Essentially, we are mimicking a lab setup with an overhead camera that communicates directly with the robot.

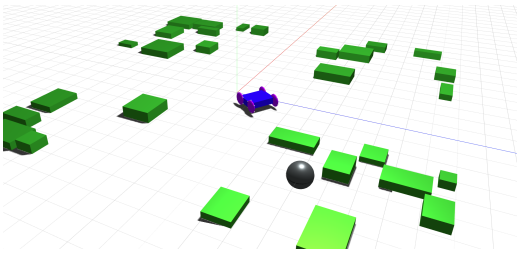


Figure 3: An example environment including randomly generated obstacles. The current way-point is shown in dark gray, and the robot starts at the origin facing in the positive x-axis (away from the way-point, along the red axis line).

Way-Point Navigation Control Adabot is a skid-steer style robot–it turns by rotating its left and right wheels at different rates. Although each wheel (and wheel strut set) can be controlled independently, in this study we only have three control outputs: (1) an angular rate for the left wheels, (2) and angular rate for the right wheels, and (3) a single extension amount for all four sets of struts. For Adabot to aid during a search and rescue operation, it must be able to successfully *cover* (completely search) its designated area. A simplified version of this task, called *way-point navigation*, is considered during evolutionary optimization. For this task, a UGV must visit a set of way-points in sequence.

FSM Control A hand-designed FSM for this task is depicted in Figure 4(a). This FSM includes two independent actions: (1) directing the robot towards the next way-point by controlling the left and right wheels, and (2) extending the struts when the robot is experiencing reduced mobility due to an obstacle.

The first action is handled by the three states shown in the top-half of the figure. Essentially, the robot remains in the *Forward* state as long as the angle between the heading of the UGV and the direction to the target (α_{target}) is within some threshold. Once the threshold is surpassed, the FSM transitions to either the *Left* or *Right* state. In the *Left* and *Right* states, the robot will rotate in place until α_{target} is less than another threshold, after which point the FSM transitions back to *Forward*. Threshold angles are shown in Figure 4(b).

The second action, handling the wheel struts, is accounted for by the state shown in the lower part of Figure 4(a). This state calculates the expected linear (v_{UGV}) and angular (ω_{UGV}) speed using a simple differential drive model:

$$v_{UGV} = \frac{V_r + V_l}{2}, \tag{1}$$

$$\omega_{UGV} = \frac{V_r - V_l}{TrackWidth} \tag{2}$$

where V_r and V_l are the left and right wheel linear velocities (depending on the left and right wheel rates, described below, as well as *WheelRadius*), respectively, and *TrackWidth* represents the distance between wheels on the same axle line (front or rear axles). As we are dealing with a noise-free, idealized simulation, this differential drive model has shown to be very accurate. It compares the expected speed to the actual speeds of the device, which are known exactly in simulation and can be measured accurately using an overhead camera setup in physical experiments, to calculate the scaled (between 0 and 1) absolute error values v_{UGV_e} and ω_{UGV_e}. These errors values are filtered using exponential smoothing and then used in the following equations to calculate the extension amount of all struts:

$$ext_v = m_{ext} \cdot v_{UGV_e} + b_{ext}, \tag{3}$$

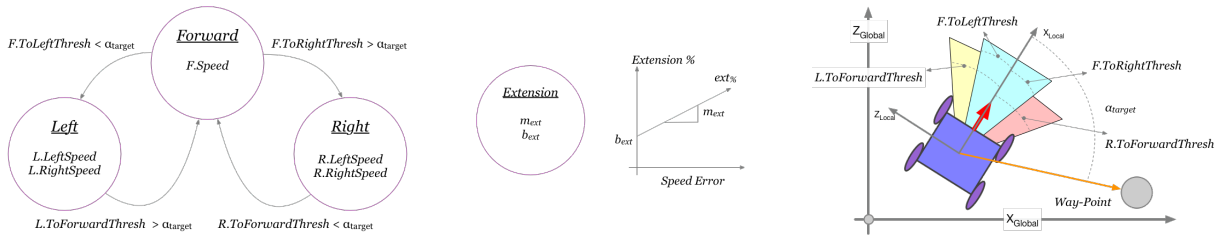
$$ext_\omega = m_{ext} \cdot \omega_{UGV_e} + b_{ext}, \tag{4}$$

$$ext_\% = \max[ext_v, ext_\omega], \tag{5}$$

$$ext = MAXEXT \cdot ext_\% \tag{6}$$

where ext_v and ext_ω denote the extension amount calculated due to the linear and angular speed values, respectively. These two values are calculated using a linear equation with a configurable slope (m_{ext}) and intercept (b_{ext}). The final extension amount is based on the maximum of these two values, and is calculated as a percentage of the maximum possible extension ($MAXEXT$). In essence, the struts will be extended an amount that is linearly proportional to the

[2]https://dartsim.github.io/index.html
[3]https://bitbucket.org/odedevs/ode



(a) Direction Control (b) Strut Extension Control (c) Environment Diagram

Figure 4: (a) The FSM controlling the direction of the robot, (b) the single state controlling wheel struts, and (b) a diagram depicting the angles used to trigger transitions between states.

current error in speed. Thus, when Adabot encounters an obstacle that blocks it from moving according to the differential drive model it will extend the struts in an effort to climb over any obstacles. Table 2 shows all configurable parameters for the FSM. Aside from the first two rows, each name in the table takes the following form: a capital letter representing a state in Figure 4(a) (**F***orward*, **L***eft*, or **R***ight*), followed by a period, followed by either an angular wheel rate or a angle threshold value also found in Figure 4(a).

Table 2: Adabot FSM Parameters

Name	Range
m_{ext}	0 to 1
b_{ext}	0 to 1
$F.Speed$	0 to 20 rad s^{-1}
$F.ToLeftThresh$	0 to 90°
$F.ToRightThresh$	−90 to 0°
$L.LeftSpeed$	−20 to 20 rad s^{-1}
$L.RightSpeed$	−20 to 20 rad s^{-1}
$L.ToForwardThresh$	0 to 90°
$R.LeftSpeed$	−20 to 20 rad s^{-1}
$R.RightSpeed$	−20 to 20 rad s^{-1}
$R.ToForwardThresh$	−90 to 0°

For this study we decided to use this single, simple state (based on just to configurable values) to reduce the number of factor that must be considered while comparing the FSM to the ANN. In future work we plan to extend this FSM to handle more complex tasks and exhibit more complex dynamics. One final, important note regarding Adabot's control system. To reduce vibration and potential damage to the wheel struts, the maximum angular rate of the wheels is linearly scaled down from 20 rad s^{-1} to 4 rad s^{-1} when the struts are fully extended.

ANN Control As an alternative to the FSM controller, we evolve an ANN for the same task. The neural network receives three inputs: (1) α_{target} scaled between 0 and 1, (2) v_{UGV_e}, and (3) ω_{UGV_e}. Essentially, the ANN is given the same information as the FSM, and produces the same three output values (left and right wheel rates and an extension amount). For the sake of analyzing the evolved networks, we chose to make the ANN as simple as possible. Thus, the network is feed-forward and it does not include any hidden nodes. The genome for our ANN includes 13 values: one integer value representing the activation function (logistic, hyperbolic tangent, or the rectified linear unit) and 12 values for the neural network weights (three inputs plus one bias times three outputs).

Evolution For this study, we employ the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [7]. In particular, we use the Python implementation, pycma[4], developed by the original author of CMA-ES. CMA-ES was chosen as it works well on real-valued problems and has support for handling integer values in the genome (as is the case for *ExtCount*). Although we experimented with several of the configurable parameters, we did not notice any positive results when changes values from their defaults.

4 DISCUSSION AND RESULTS

In this section we provide our results from evolving the Adabot. Specifically, we evolved the Adabot in two environments (with and without obstacles) and with two different controllers (the FSM defined in Section 3 and an ANN). Each of these four experiments is repeated 20 times. Finally, we discuss principles that can be learned from these experiments.

4.1 Fitness Calculations

Here, Adabot's goal is to visit a set of coordinates (way-points) in sequence. During a single simulation, the device has 30 seconds to visit **four** predefined way-points, but the simulation will terminate as soon as the fourth way-point is reached. Fitness is calculated as follows:

$$f = 2w + (1 - \frac{\min[d, d_{max}]}{d_{max}}) + \frac{(30 - t)}{30} \quad (7)$$

where w represents the number of way-points reached, d and d_{max} denote the distance to the next way-point and a scaling factor for distances, respectively, and t denotes the time at the end of the evaluation. This function is meant to provide a smooth gradient for generating controllers that quickly

[4]https://github.com/CMA-ES/pycma

navigate to all way-points in order. The first part of the equation ensures that the CMA-ES algorithm heavily favors any controller that reaches even a single way-point; values for this component range from 0 to 8. Next, a distance component is added to reward any solutions that drive near its next way-point. This is particularly useful at the beginning when solutions are randomly generated. The distance component results in a value scaled between 0 and 1. Since the simulation ends once all four way-points have been reached, the time component will be a value between 0 (zero time remaining) and 1 (all four way-points are reached in an instant). The time component is meant to favor any controllers that solve the task quickly. Thus, the maximum possible fitness is 10.

4.2 Evolution Without Obstacles

In our first experiment, *FSM-0-1*, we evolve the fifteen parameters found in Tables 1 and 2 in an environment without obstacles. The naming scheme for our experiments indicates the controller type (*FSM* or *ANN*), the maximum number of potential obstacles (0 or 40), and the number of trials per fitness evaluation (1 or 2). Plots of fitness vs iteration are show in Figure 5 (this figure shows the fitness values for all four experiments). In this first experiment, there are zero obstacles and therefore the environment will always be the same. In later experiments, each fitness evaluation includes two trials with randomly generated obstacles. As shown in the figure, in all replicate trials the Adabot reaches all four way-points in approximately 10 seconds, which corresponds to a fitness value of 9.7. The population quickly converges on a final value, likely because this experiment was seeded with a hand-designed set of parameters known to achieve good results; the evolved results, however, quickly outperform the hand-chosen values. This experiment serves as a convenient baseline with which the others can be compared.

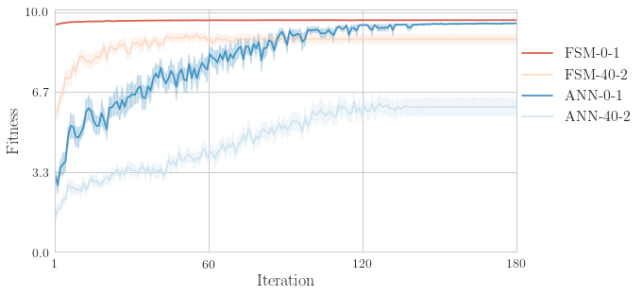


Figure 5: Fitness plots for the maximum fitness found in each of the four experiments. The shaded regions indicate confidence intervals of one standard deviation from the mean for the 20 replicated trials of each experiment. The maximum possible fitness is 10 and any fitness value above 2 means that UGV was able to reach at least one way-point.

The second experiment, denoted *ANN-0-1.* also reaches a fitness value of 9.7, which shows that the an ANN can effectively perform the task of navigating the robot to a sequence of points. For this experiment, 17 total parameters were evolved: the four physical characteristics listed in Table 1 and the 13 ANN parameters discussed in the previous section. Although both of these experiments reach the same final fitness value, an examination of Figure 5 shows that the ANN result takes longer to evolve–roughly 120 iterations compared with less than 10 iterations for the FSM. This can be explained by the lack of a seed controller and the fact that, unlike an FSM, an ANN must learn the entire solution from scratch.

Figure 6 depicts the trajectories taken by the best performing controllers from these two experiments. Although these trajectories look similar, there is one key difference: the ANN actively controls only one wheel. The FSM, on the other hand, occasionally rotates in both directions by turning its left and right side wheels in opposite directions.

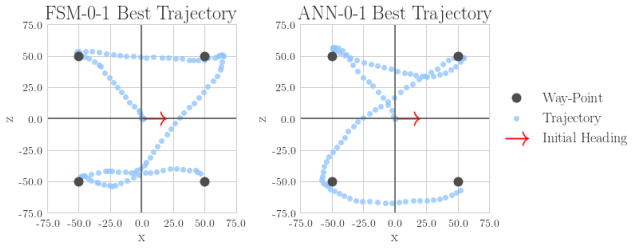
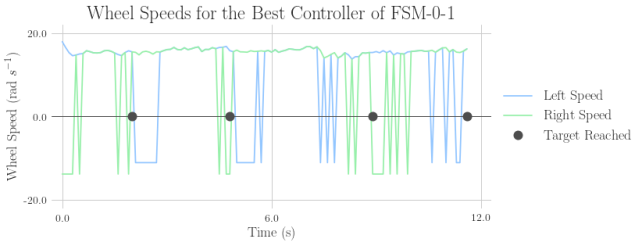


Figure 6: Trajectories of the best evolved individual for the first two experiments: FSM-0-1 and ANN-0-1. No obstacles were present for these trajectories.

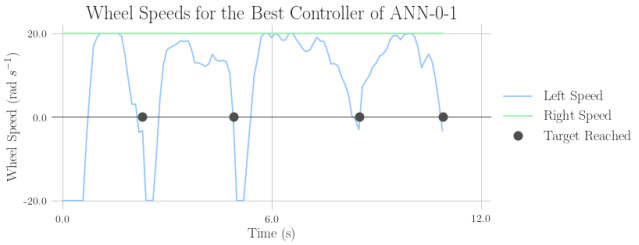
Figure 7 shows the wheel speeds for the best FSM and ANN controllers. The evolved ANN perpetually sets the right wheel to its maximum speed. The ANN moves forward by setting its left wheel to the same value, and turns by making the left wheel rotate in the opposite direction. Effectively, the ANN can only turn left, however, this is not a problem for the relatively simple task at hand.

Figure 8 provides a comparison of the fitness values and evolved physical characteristics for these two experiments. This figure only shows results for the combined final populations of all replicate experiments. From this figure, we can establish what will be good physical characteristics for the Adabot when it does not face any obstacles. Specifically, *WheelBase* and *TrackWidth* should be 8.5 cm and 11.5 cm, respectively, *WheelRadius* should be 3 cm, and each wheel should have at least 4 wheel struts (*StrutCount*).

Finally, Figure 9(a) plots the distributions for all evolved FSM parameters. It is worth noting that neither the evolved FSMs or ANNs extend the struts. This results is not unexpected as any extension would result in a reduced speed (due to the scaling mentioned in Section 3), and the struts are not needed when obstacles are not present. Also of interest is the evolved symmetry of the FSM. Specifically, the threshold values and speeds evolved for the *Left* and *Right* states are nearly perfect mirror images of each other.



(a)



(b)

Figure 7: Left and right wheel speeds for the best evolved solutions for *FSM-0-1* (a) and *ANN-0-1* (b).

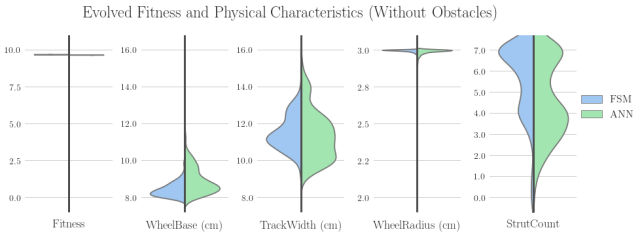


Figure 8: Distributions for the evolved fitness values and physical characteristics for the combined final populations of the *FSM-0-1* (left side) and *ANN-0-1* (right side) experiments. The y-axis limits are the parameter limits allowed during evolution.

4.3 Evolution With Obstacles

The final two evolutionary experiments are referred to as *FSM-40-2* and *ANN-40-2*. These experiments differ from the previous two in two respects. First, each fitness value is calculated as the average of two trials (where each trial lasts at most 30 seconds), and second, each fitness trial occurs in an environment with at most 40 randomly generated obstacles. Utilizing multiple trials during fitness evaluations improves the robustness of the evolved results [16]. The fitness plots for these experiments appear in Figure 5. Of note is that the ANNs evolved with obstacles have a greatly reduced maximum fitness. A few individuals achieve a fitness above 9, however, we found that this was only when the randomly generated environment did not pose much difficulty. We have provided videos for the highest fitness individuals from both experiments[5][6].

[5] *FSM-40-2* example behavior: https://youtu.be/VXnrwwpE598
[6] *BNN-40-2* example behavior: https://youtu.be/q8PFqQps5e4

Similar to Figure 8, Figure 10 shows the distributions for the evolved physical characteristics. The values found in these distributions indicates that the presence of obstacles does not have a drastic affect on the evolution of physical characteristics. At first this was unexpected, however, analyzing these values (and visualizing their resulting behaviors) reveals a few basic principles: (1) for a skid steer robot it is important for the *WheelBase* to be less than the *TrackWidth* (this will reduce the amount of *skidding* and improve controllability), (2) to maximize velocity *WheelRadius* should be maximized (since we are evolving wheel angular rate a larger wheel will result in a higher velocity), and (3) as long as the number of struts is greater than 4 the system will be able to navigate the generated environments. The first and second principles match results that we have seen on the physical prototype, and we intend to investigate the third principle in the near future. This leaves us with control being the most important factor in navigating obstacles.

While the physical characteristics are similar between the two sets of experiments, control strategies have been changed to handle the obstacles. Figure 11 shows the control patterns for two solutions randomly selected from the best performing individuals of the *FSM-40-2* and *ANN-40-2* experiments. Note that since environments are randomly generated, even though the evolved ANN does not reach all four way-points for this test, it does not mean that it did not do so during fitness evaluation. The most important feature of the plots in Figure 11 are that the evolved controllers are operating at a reduced speed. Examining the evolved FSM values , we see that nearly identical values are discovered for all parameters except b_{ext} (a set of distributions similar to Figure 9 has been omitted to save space). In the experiment with no obstacles, b_{ext} was consistently set to a value near zero, however, for this experiment b_{ext} was set to 0.45. A higher value for b_{ext} results in the struts always being extended (even when no obstacle has been encountered). Thus, the cause of the slower behavior is that the controllers are using the struts to improve mobility in the face of obstacles.

The final set of evolved parameters to examine is those of the neural network. Directly examining the evolved weights provides only a limited view of the resulting behavior. Likewise, comparing each input's effect on each output in isolation also obscures analysis. For example, some output values are only active when some combination of multiple input values is provided. Thus, in Figure 12 we provide all pairwise heat-maps for all inputs and outputs. These heat-maps were generated using a parameter sweep over all possible input combinations. Each square represents the average output value given the two input values on the x- and y-axes and all possible values for the remaining input. Looking at the second and third rows in this figure we can see that for any combination of input values the right wheel is always rotating at a constant forward speed and the struts are always fully extended, respectively. As was the case for the *ANN-0-1* experiment, all navigation is handled by driving the left wheel at different speeds. Examining the first row shows that the left wheel's speed has a linear relationship with all three

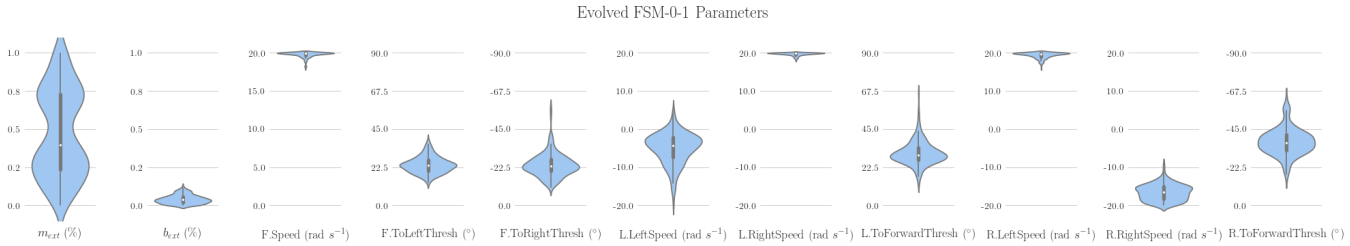


Figure 9: Distributions for all evolved FSM parameters for the *FSM-0-1* experiment.

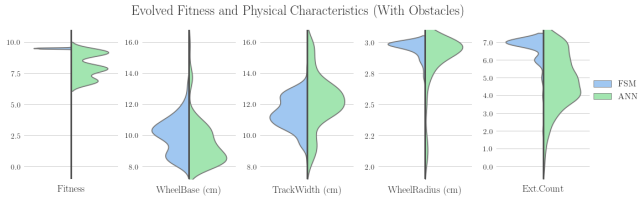


Figure 10: Distributions for the evolved fitness values and physical characteristics for the combined final populations of the *FSM-40-2* (left side) and *ANN-40-2* (right side) experiments. The y-axis limits are the parameter limits allowed during evolution.

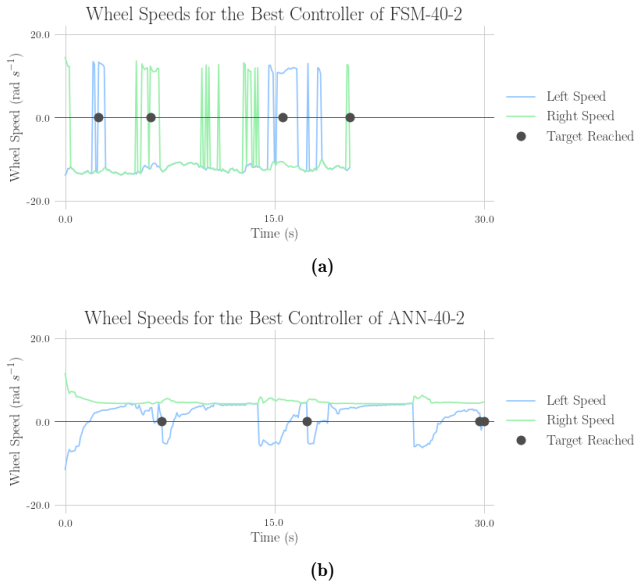


Figure 11: Left and right wheel speeds for the best evolved FSM (a) and ANN (b) in a randomly generated environment that includes obstacles.

neural network inputs. As any of the inputs are increased, so to does the left wheel speed.

In both experiments including obstacles, the evolved controllers extended the struts and never fully retracted them. However, there is a clear advantage to retracting the struts: the robot has a higher maximum allowed speed. Thus, it is likely an issue with using the differential drive model to calculate the error. We have identified two sources of error with simple model: (1) it does not take into account that when the struts are extended the wheel has a larger effective radius, and (2) the model does not take into account the noisy nature of a skid steering and extended struts.

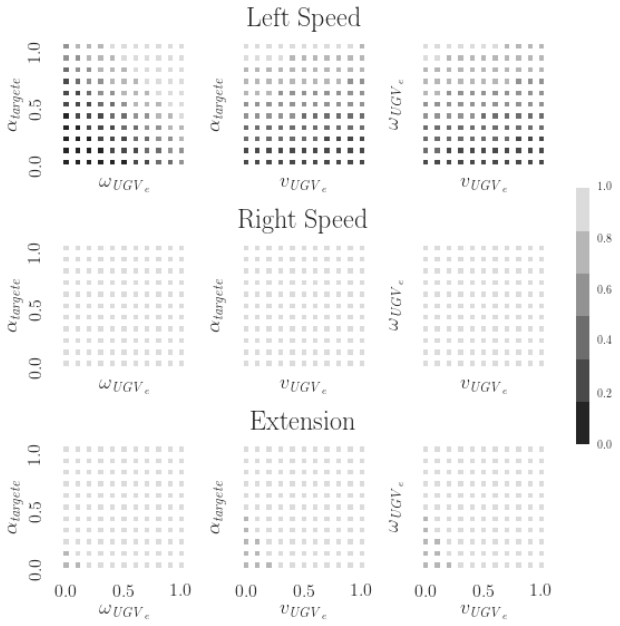


Figure 12: Heat-maps showing the relationship between input and output for the best evolved ANN. For example, the first row represents the output wheel rate for the left wheels, a light shade indicates that the wheel is at its maximum forward rate, and a dark color indicates that it is at its maximal reverse rate. All input and output values are scaled between 0 and 1.

To summarize our findings regarding the optimization of the Adabot system:

(1) Similar physical characteristics are optimal with and without obstacles in the environment.
(2) The speed of the left and right wheels should have a linear relationship with α_{target} (rather than a discrete relationship as is the case with the current FSM).
(3) The task can be solved by controlling only a single wheel, though, this is likely not a desirable trait. In future work, we plan to add an evolutionary pressure so that the evolved ANNs turn in both directions.

(4) Controlling the strut will require a more complex model of the robots dynamics. Once the struts are extended, it is difficult to discern when they can be retracted.

Taking these observations into account, we developed a hybrid two-state controller. The controller is in *LeftState* when α_{target} is greater than zero and in *RightState* otherwise. Equations for these states are as follows:

$$\alpha_{scale} = 2 \cdot (1 - \frac{\alpha_{target}}{\pi}) - 1 \quad (8)$$

$$LeftState_{left} = MAXRAD \cdot \alpha_{scale} \quad (9)$$

$$LeftState_{right} = MAXRAD \quad (10)$$

$$RightState_{left} = MAXRAD \quad (11)$$

$$RightState_{right} = -MAXRAD \cdot \alpha_{scale} \quad (12)$$

where α_{scale} is α_{target} scaled between 1 and -1 (the sign is flipped). This simple hybrid controller is able to visit all waypoints in 9.9 seconds, which is one tenth of a second faster than the evolved controllers reported above. The controller also works well in the presence of obstacles when the struts are extended 10%. Overall, this hybrid controller provides a smoother motion and good performance. For future work, we intend to evolve this hybrid controller along with a more sophisticated approach to handling strut extension.

5 CONCLUSION

UGVs are becoming more prevalent. Likewise, their envisioned environments are becoming more dynamic and varied. We have evolved a UGV so that it is better able to handle obstacles of varying sizes. Specifically, we compared and analyzed FSM and ANN controllers with and without obstacles in the environment. Comparing these two techniques enabled us to propose a new hybrid controller that incorporates the advantages of both. Although a direction controller was straightforward to optimize, the complex dynamics associated with climbing over obstacles makes it more difficult to design a controller for extending the Adabot' struts. Our future work will focus on optimizing the hybrid controller and investigating different techniques for extending the struts.

REFERENCES

[1] Josh C Bongard. 2013. Evolutionary robotics. *Communications of the ACM*, 56, 8, 74–83.

[2] Valentino Braitenberg. 1986. *Vehicles: experiments in synthetic psychology*. MIT press.

[3] Anthony J. Clark, Philip K. McKinley, and Xiaobo Tan. 2015. Enhancing a model-free adaptive controller through evolutionary computation. In *Proceedings of the 2015 ACM Genetic and Evolutionary Computation Conference (GECCO)*. Madrid, Spain, (July 2015), 137–144.

[4] M. Eich, F. Grimminger, and F. Kirchner. 2008. A versatile stair-climbing robot for search and rescue applications. In *Proceedings of the 2008 IEEE International Workshop on Safety, Security and Rescue Robotics*. (Oct. 2008), 35–40. DOI: 10.1109/SSRR.2008.4745874.

[5] Jorge Gomes, Pedro Mariano, and Anders Lyhne Christensen. 2015. Devising effective novelty search algorithms: a comprehensive empirical study. In *Proceedings of the 2015 annual conference on genetic and evolutionary computation* (GECCO '15). ACM, Madrid, Spain, 943–950. ISBN: 978-1-4503-3472-3. DOI: 10.1145/2739480.2754736. http://doi.acm.org/10.1145/2739480.2754736.

[6] D. W. Haldane, K. C. Peterson, F. L. Garcia Bermudez, and R. S. Fearing. 2013. Animal-inspired design and aerodynamic stabilization of a hexapedal millirobot. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*. (May 2013), 3279–3286. DOI: 10.1109/ICRA.2013.6631034.

[7] Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary computation*, 11, 1, 1–18. DOI: 10.1162/106365603321828970.

[8] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2, 5, 359–366.

[9] G. Kenneally, A. De, and D. E. Koditschek. 2016. Design principles for a family of direct-drive legged robots. *IEEE Robotics and Automation Letters*, 1, 2, (July 2016), 900–907. ISSN: 2377-3766. DOI: 10.1109/LRA.2016.2528294.

[10] N. Koenig and A. Howard. 2004. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. (Sept. 2004), 2149–2154. DOI: 10.1109/IROS.2004.1389727.

[11] Kazuya Yoshida and Satoshi Tadokoro, (Eds.) 2014. *Experience in system design for human-robot teaming in urban search and rescue. Proceedings of 8th International Conference on Field and Service Robotics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 111–125. ISBN: 978-3-642-40686-7. DOI: 10.1007/978-3-642-40686-7_8. https://doi.org/10.1007/978-3-642-40686-7_8.

[12] Jared M. Moore, Anthony J. Clark, and Philip K. McKinley. 2017. Effect of animat complexity on the evolution of hierarchical control. In *Proceedings of the 2017 acm genetic and evolutionary computation conference*. Berlin, Germany, (July 2017). DOI: https://doi.org/10.1145/3071178.3071246.

[13] Jean-Baptiste Mouret and Konstantinos Chatzilygeroudis. 2017. 20 years of reality gap: a few thoughts about simulators in evolutionary robotics. In *Proceedings of the genetic and evolutionary computation conference companion* (GECCO '17). ACM, Berlin, Germany, 1121–1124. ISBN: 978-1-4503-4939-0. DOI: 10.1145/3067695.3082052. http://doi.acm.org/10.1145/3067695.3082052.

[14] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. 2009. ROS: an open-source robot operating system - Willow Garage. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Robotics (OSS)*. (May 2009).

[15] R. D. Quinn, J. T. Offi, D. A. Kingsley, and R. E. Ritzmann. 2002. Improved mobility through abstracted biological principles. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3, 2652–2657. DOI: 10.1109/IRDS.2002.1041670.

[16] E. L. Ruud, E. Samuelsen, and K. Glette. 2016. Memetic robot control evolution and adaption to reality. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. (Dec. 2016), 1–7. DOI: 10.1109/SSCI.2016.7850169.

[17] Uluc Saranli, Martin Buehler, and Daniel E. Koditschek. 2001. RHex: a simple and highly mobile hexapod robot. *The International Journal of Robotics Research*, 20, 7, 616–631. DOI: 10.1177/02783640122067570.

[18] Aravind Seeni, Bernd Schäfer, and Gerd Hirzinger. 2010. Robot mobility systems for planetary surface exploration–state-of-the-art and future outlook: a literature survey. In *Aerospace technologies advancements*. InTech, 189–208. DOI: 10.5772/6930.

[19] Fernando Silva, Miguel Duarte, Luis Correia, Sancho Moura Oliveira, and Anders Lyhne Christensen. 2016. Open issues in evolutionary robotics. *Evolutionary Computation*, 24, 2, (June 2016), 205–236. ISSN: 1063-6560. DOI: 10.1162/EVCO_a_00172.

[20] Karl Sims. 1994. Evolving virtual creatures. In *Proceedings of the 21st annual conference on computer graphics and interactive techniques*. ACM, 15–22.

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

[22] Muhammad Yaqoob and Borys Wróbel. 2017. Very small spiking neural networks evolved to recognize a pattern in a continuous input stream. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. Honolulu, Hawaii, USA, (Dec. 2017).