Evolving Controllers for a Transformable Wheel UGV

Anthony J. Clark[1], Keith A. Cissell[1] and Jared M. Moore[2]

[1]Missouri State University, Springfield, Missouri, USA
[2]Grand Valley State University, Allendale, Michigan USA
AnthonyClark@MissouriState.edu

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 systems 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 differing strategies. Finally, we show that a UGV's controller and physical characteristics can be effectively chosen by examining results from evolutionary optimization.

Introduction

Autonomous unmanned ground vehicles (UGV) provide an excellent solution to tasks that require 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. UGVs have long operating durations, can carry heavy payloads (e.g., sensors), and can search in narrow and covered places such as forests and caves.

Ensuring that a UGV can handle many different types of terrain is an ongoing challenge. Researchers 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), and transformable wheels (Saranli et al., 2001; Quinn et al., 2002; Eich et al., 2008; Haldane et al., 2013; Kenneally et al., 2016). Although these systems provide an advantage over traditional wheeled robots, optimization is not performed in the vast majority of these studies.

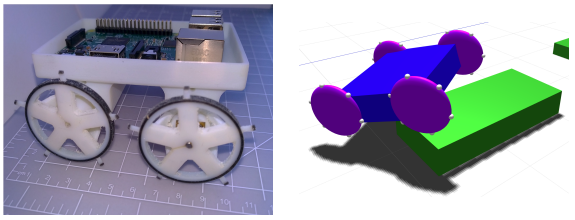
The device in this study, the *Adabot* (see Figure 1), includes transformable wheels that can smoothly be converted from a round wheel, to a wheel with tire *studs*, to a legged-wheel. Wheel transformations are performed by extending wheel struts radially outward from the center of the wheel (see Figure 2). 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 (Clark, 2017), a similar system was optimized to maximum forward velocity over uneven terrain. 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 of feedback controllers (rather than feed-forward).

In this study, we evolve the robot's chassis dimensions, its wheel radius, the number of wheel struts, 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. The network is only slightly more complex than a Type 2 Braitenberg vehicle (Braitenberg, 1986). 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. Source code has been made available on GitHub[1].

Related Work

In the field of evolutionary robotics (ER), an evolutionary algorithm (EA) optimizes free variables of a given

[1]`https://github.com/anthonyjclark/adabot02-ann`



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

Figure 1: Adabot, a UGV with transformable wheels.

system (Silva et al., 2016). ER methods have been successfully applied to many different types of robotic systems (aerial, aquatic, walking, etc.). For example, we have previously used differential evolution to evolve an adaptive neural network for a robotic fish (Clark et al., 2015), and Moore et al. (2017) 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 Gomes et al. (2015)), it has not often been used to directly evolve UGVs, and to the best of our knowledge this is the first study in which the characteristics of a transformable wheel are evolved.

A large number of ER studies utilize ANNs to control mobile robots, including *Evolving Virtual Creatures* (Sims, 1994), 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 (Hornik et al., 1989), evolution often produces *novel* and sometimes unintuitive results that may not have been found when creating a controller by hand (Bongard, 2013). 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) (Stanley and Miikkulainen, 2002). 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. For example, Yaqoob and Wróbel (2017) automatically generated a state machine with the same properties of an evolved spiking neural network.

Adabot

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) (Quigley et al., 2009), which Adabot uses to deploy 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 gearmotor 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. Due to limitations of the design, the maximum extension of the struts is equal to the wheel's radius minus 1 cm ($MAXEXT = WheelRadius - 1$ 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) see our preliminary study (Clark, 2017).

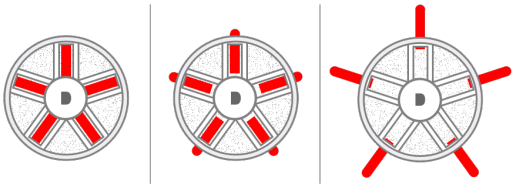


Figure 2: Illustration of Adabot's wheel extension mechanism, where the struts are fully retracted (left), partially extended (middle), and fully extended (right).

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 an existing box it is removed from the simulation. We

see on average 31 boxes placed in the environment. Box heights range from 2 to 5 cm, which is high enough (compared to *WheelRadius* values) to drastically reduce mobility for a wheeled robot (Quinn et al., 2002). 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, and is comparable in speed (if not faster) than common alternatives (Mouret and Chatzilygeroudis, 2017).

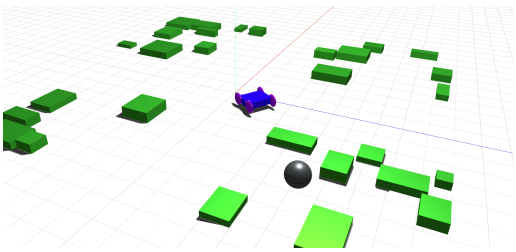


Figure 3: An example environment including randomly generated obstacles. The current way-point is shown as a dark gray sphere, and the robot starts at the origin facing in the positive x-axis (away from the way-point, along the red axis line).

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) an 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: The hand-designed FSMs for this task are depicted in Figure 4. This FSM includes two independent actions: (a) directing the robot towards the next way-point by controlling the left and right wheels, and (b) extending the struts when the robot is experiencing reduced mobility due to an obstacle. 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 greater than $L.ToForwardThresh$ or less than $R.ToForwardThresh$, respectively, after which point the FSM transitions back to *Forward*. Threshold angles are shown in Figure 4(c).

To determine when, and by how much, wheel struts should be extended, we use a simple model based on the UGV's current error. Specifically, we calculate an expected linear (v) and angular (ω) velocity (based on the wheel rates) using the following model:

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

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

where V_r and V_l are the left and right wheel linear velocities, respectively, and $TrackWidth$ represents the distance between wheels on the same axle line (front or rear axles). These calculated values are then compared to the measured values, which we have direct access to in simulation. Error values are scaled between 0 and 1 and then filtered using exponential smoothing. Finally, they are used in the following equations to calculate the extension amount of all struts:

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

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

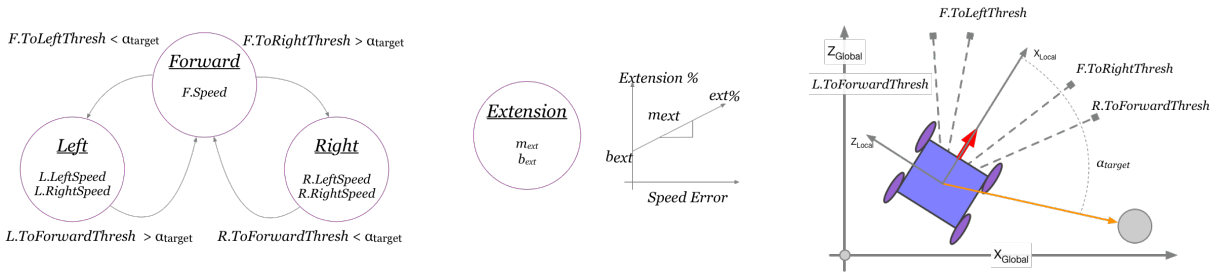
$$ext_\% = \max[ext_v, ext_\omega], \tag{5}$$

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

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 (ext) 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 current error in speed (maximum between linear and angular error). 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 (hand-chosen values are shown in parentheses). Aside from the first m_{ext} and b_{ext}, 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 described in Figure 4(a). Finally, to reduce vibration and potential damage to the wheel struts, the maximum angular rate of the wheels is linearly scaled down from 20 rads^{-1} to 4 rads^{-1} when the struts are fully extended.

[2] `https://dartsim.github.io/index.html`



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

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

Table 2: Adabot FSM Parameters (Hand-Chosen)

Name	Range
m_{ext}	(0.5) 0 to 1
b_{ext}	(0.5) 0 to 1
$F.Speed$	(20) 0 to 20 rads^{-1}
$F.ToLeftThresh$	(10) 0 to 90°
$F.ToRightThresh$	(-10) −90 to 0°
$L.LeftSpeed$	(-20) −20 to 20 rads^{-1}
$L.RightSpeed$	(20) −20 to 20 rads^{-1}
$L.ToForwardThresh$	(5) 0 to 90°
$R.LeftSpeed$	(20) −20 to 20 rads^{-1}
$R.RightSpeed$	(-20) −20 to 20 rads^{-1}
$R.ToForwardThresh$	(-5) −90 to 0°

ANN Control: As an alternative to the FSM controller, we evolve an ANN for the same task. The neural network receives three inputs (each scaled between 0 and 1): (1) α_{target}, (2) v_e, and (3) ω_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). In our preliminary work, we found hidden nodes were unnecessary for this task (the same strategies and fitness values were attained with and without 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 for each of the three outputs).

Evolution: For this study, we employ the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen et al., 2003). In particular, we use pycma[3] (developed by Hansen et al.), which works well on real-valued problems and has support for handling integer values such as *ExtCount*.

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

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. Each of these four experiments is repeated 20 times. Finally, we discuss principles that can be learned from these experiments.

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:

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

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 transpired. This function is meant to provide a smooth gradient for generating controllers that quickly 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 solutions that drive near the next way-point in sequence, but do not reach all four. This is particularly useful at the beginning when solutions are at an early stage of evolution. 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.

Evolution Without Obstacles

In our first experiment, *FSM-0-1*, we evolve the fifteen parameters found in Tables 1 (physical) and 2 (control) 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 experiments 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 (see Table 2); the evolved results, however, quickly outperform the hand-chosen values. This experiment serves as a convenient baseline with which the others can be compared.

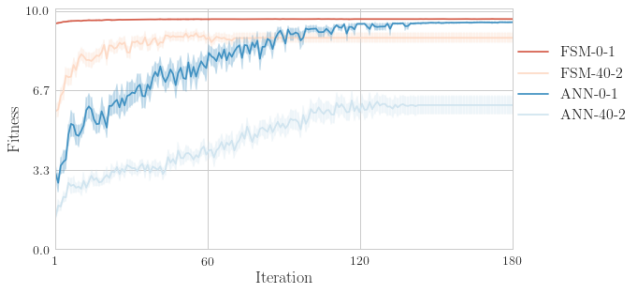


Figure 5: Plots for the maximum fitness found in each of the four experiments. Shaded regions indicate confidence intervals of one standard deviation from the mean for the 20 replicates of each experiment. The maximum possible fitness is 10, and fitness values above 2 indicate that Adabot was able to 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. FSMs, on the other hand, rotate both directions in-place, which is why there are sharper turns in the left plot.

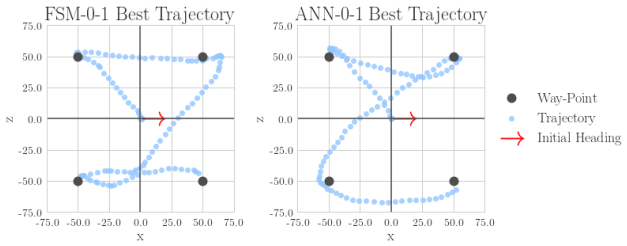


Figure 6: Trajectories of the best evolved individual for the first two experiments: FSM-0-1 and ANN-0-1. No obstacles were present for these trajectories.

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.

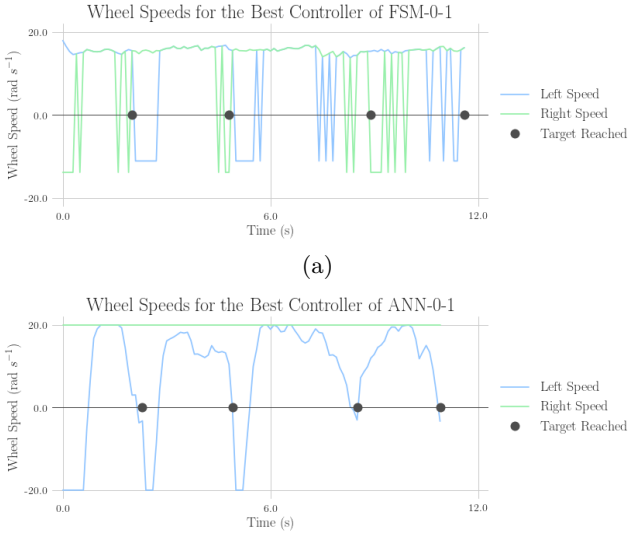


Figure 7: Left and right wheel speeds for the best evolved solutions for *FSM-0-1* (a) and *ANN-0-1* (b).

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 *StrutCount* does not matter since the struts are not extended.

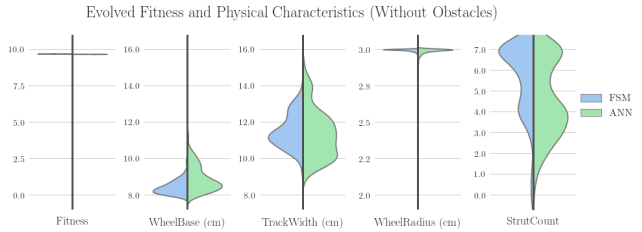


Figure 8: Distributions for the evolved fitness values and physical characteristics for the combined final populations of the *FSM-0-1* (left side) and *ANN-0-1* (right side) experiments. The y-axis limits are the parameter limits allowed during evolution.

Finally, Figure 9(a) plots the distributions for all evolved FSM parameters. It is worth noting that in the absence of obstacles, neither the evolved FSMs or ANNs extend the struts. This result is not unexpected as any extension would result in a reduced speed (due to the scaling mentioned previously), 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.

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 around 31 randomly generated obstacles. Utilizing multiple trials during fitness evaluations improves the robustness of the evolved results (Ruud et al., 2016). 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. Videos (and interactive animations) for high fitness individuals can be found here: *FSM-40-2*: `https://youtu.be/VXnrwwpE598` (`https://goo.gl/NtoVYe`); and *ANN-40-2*: `https://youtu.be/q8PFqQps5e4` (`https://goo.gl/2xjh6X`).

Similar to Figure 8, Figure 10 shows the distributions for the evolved physical characteristics. These distributions have a larger spread due to the randomly generated environments. The values found in these distributions indicate 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.

While the physical characteristics are similar between the two sets of experiments, control strategies have been adjusted 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 reduced speeds. 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} converged to zero, however, for this experiment b_{ext} converged to 0.45. A higher value for b_{ext} results in the struts always being extended (even when no obstacle has been encountered). Thus, these behaviors are slower because the struts are required to climb obstacles.

Directly examining the evolved weights of a neural network provides only a limited view of the resulting behavior. Likewise, comparing each input's effect on each output in isolation obscures the resulting behaviors. For example, some output values are only active when some combination of multiple input values are provided. Thus, in Figure 12 we provide all pairwise input relationships on the output for the speed of the left wheels in the form of heat-maps. These heat-maps were generated using a parameter sweep over all possible input combinations. Each square represents the output value given the two input values on the x- and y-axes averaged over all possible values for the remaining input. As was the case for the *ANN-0-1* experiment, all navigation is handled by driving the left wheel at different speeds, and so we have not provided heat-maps for the wheel strut and right speed outputs. Examining the

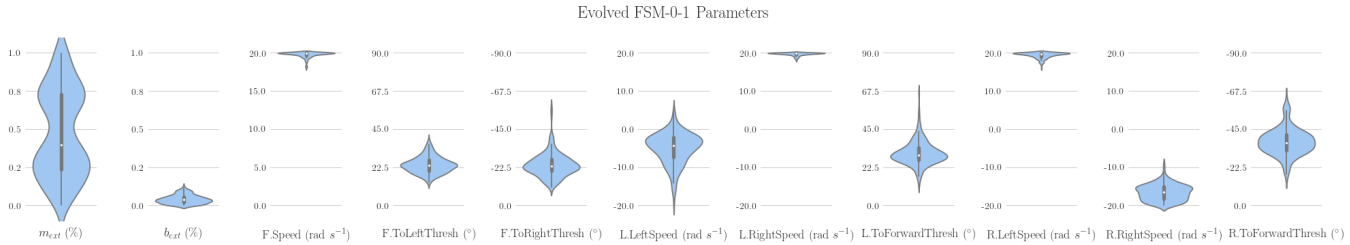


Figure 9: Distributions for all evolved FSM parameters for the *FSM-0-1* experiment.

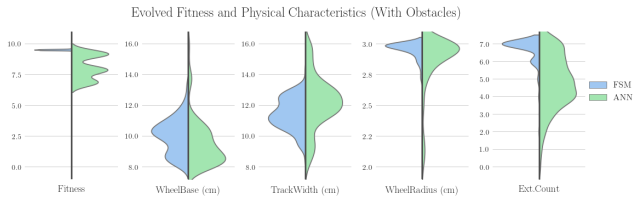


Figure 10: Distributions for the evolved fitness values and physical characteristics for the combined final populations of the *FSM-40-2* (left side) and *ANN-40-2* (right side) experiments. The y-axis limits are the parameter limits allowed during evolution.

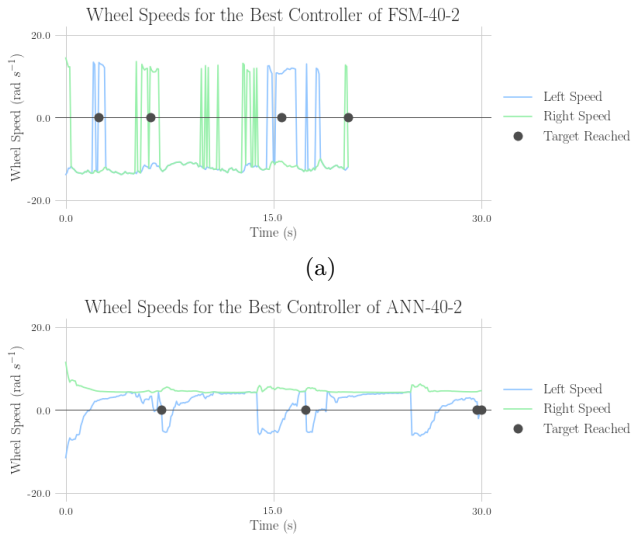


Figure 11: Left and right wheel speeds for the best evolved FSM (a) and ANN (b) in a randomly generated environment that includes obstacles.

figure shows that the left wheel's speed has a positive linear relationship with both ω_e and α_{target}, and that α_{target} has the greatest effect on control (since it is used to turn the robot towards the target).

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 the 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.

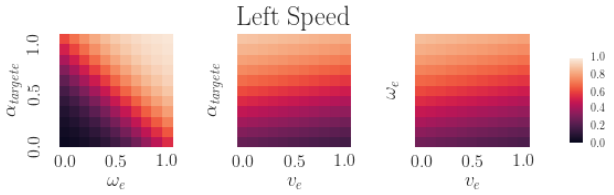


Figure 12: Heat-maps showing the relationship between input and output for the best evolved ANN. For example, the first row represents the output wheel rate for the left wheels, a light shade indicates that the wheel is at its maximum forward rate, and a dark color indicates that it is at its maximal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

In summary, regarding the optimization of the Adabot system we found that:

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 should be retracted.

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

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

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

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

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

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

where α_{scale} is α_{target} scaled between -1 and 1. This simple hybrid controller is able to visit all way-points 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.

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. In comparing these two techniques we were able to find design principles that incorporate the advantages of both. Although a direction controller is 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

Bongard, J. C. (2013). Evolutionary robotics. *Communications of the ACM*, 56(8):74–83.

Braitenberg, V. (1986). *Vehicles: Experiments in synthetic psychology*. MIT press.

Clark, A. J. (2017). Evolving adabot: A mobile robot with adjustable wheel extensions. In *Proceedings of the IEEE Symposium on Robotic Intelligence in Informationally Structured Space (IEEE RiiSS'17)*, pages 1–8, Honolulu, Hawaii, USA.

Clark, A. J., McKinley, P. K., and Tan, X. (2015). Enhancing a model-free adaptive controller through evolutionary computation. In *Proceedings of the 2015 ACM Genetic and Evolutionary Computation Conference (GECCO)*, pages 137–144, Madrid, Spain.

Eich, M., Grimminger, F., and Kirchner, F. (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*, pages 35–40.

Gomes, J., Mariano, P., and Christensen, A. L. (2015). Devising effective novelty search algorithms: A comprehensive empirical study. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 943–950, New York, NY, USA. ACM.

Haldane, D. W., Peterson, K. C., Bermudez, F. L. G., and Fearing, R. S. (2013). Animal-inspired design and aerodynamic stabilization of a hexapedal millirobot. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3279–3286.

Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Kenneally, G., De, A., and Koditschek, D. E. (2016). Design principles for a family of direct-drive legged robots. 1(2):900–907.

Moore, J. M., Clark, A. J., and McKinley, P. K. (2017). Effect of animat complexity on the evolution of hierarchical control. In *Proceedings of the 2017 ACM Genetic and Evolutionary Computation Conference*, Berlin, Germany.

Mouret, J.-B. and Chatzilygeroudis, K. (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, pages 1121–1124, New York, NY, USA. ACM.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (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)*.

Quinn, R. D., Offi, J. T., Kingsley, D. A., and Ritzmann, R. E. (2002). Improved mobility through abstracted biological principles. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2652–2657.

Ruud, E. L., Samuelsen, E., and Glette, K. (2016). Memetic robot control evolution and adaption to reality. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7.

Saranli, U., Buehler, M., and Koditschek, D. E. (2001). RHex: A simple and highly mobile hexapod robot. 20(7):616–631.

Silva, F., Duarte, M., Correia, L., Oliveira, S. M., and Christensen, A. L. (2016). Open issues in evolutionary robotics. 24(2):205–236.

Sims, K. (1994). Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 15–22. ACM.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

Yaqoob, M. and Wróbel, B. (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)*.