Solving the Shepherding Problem with Neural Networks

Captain Jake York (z5109674)
ZEIT3902 Engineering Research 3B
Australian Defence Force Academy

***Abstract*—For this report, four neural networks were trained and implemented to solve the shepherding problem modelled by Strömbom et al. Coupled to a shepherding agent, the neural networks performed comparably to the heuristic programmatic design when tested under ideal conditions. Non-ideal conditions such as arena size difference, variation of the number of sheep and applying error to input values were also analysed to test and then quantify the limits of capability of the networks. These non-ideal conditions have not been previously tested on the heuristic model developed by Strömbom et al. and so the quantified limitations of the networks are considered in isolation.**

I. Introduction

The shepherding problem presents a centuries-old concept based on the interactions between a herding sheepdog and a herded flock of sheep (or similar). In their paper, Strömbom et al. [1] describe the modelling of the problem as a key to understanding the behaviours of flocking animals and how robots can be used to influence these behaviours (as well as those of other, artificial agents). Related areas of research are fish school behaviour, the deterrence of flocking animals from particular areas (such as birds near airports), crowd control methods and domestic farming technology[2][3][4][5].

As background for this research project, literature and Matlab code written by Baxter was consulted [6][7]. Baxter builds upon the work by Strömbom et al. by introducing multi-shepherd dynamics and a theoretically scalable working area to the same problem set. In the early stages of this project, his Matlab code allowed for low level understanding of the dynamics of the problem. However, in all of the following solutions and results, all code has been originally generated and multi-shepherd dynamics were not considered. This was due to anomalous conditions arising from Baxter's work, for which debugging and further investigation was not conducted.

Neural network implementation was selected to solve the shepherding problem due to the ability of neural networks to easily identify patterns in datasets when provided with the appropriate amount of data. As neural networks apply a pattern-matching algorithm to a given problem they have been trained on, it makes them robust to non-ideal input data. This becomes very important when analysing the limits of neural network decision making when compared to the heuristic shepherding algorithm. Matlab features a neural network Deep Learning Toolbox (DLT) that allows users to train networks from datasets they provide. This report discusses the scope of the shepherding problem in detail, methodology behind selection of particular networks for decision making, development of training scenarios to teach the networks, testing of the networks under ideal and non-ideal conditions, and analysis of the limits of the networks' performance under these non-ideal conditions.

II. Original Problem Dynamics

In the shepherding problem, there are two types of agent: the shepherding agent (considered to be a sheepdog in reality) and the flocking agents (sheep in reality). Hereafter, in this report, these will be referred to as shepherd and sheep respectively.

In the original model by Strömbom et al., a two-dimensional arena of a predetermined size is used as the effective 'paddock' in which the shepherd and sheep interact. This is a square area with side lengths equal to 150 m. At the beginning of a simulation, all of the sheep are randomly generated in the upper-right quadrant of the paddock, while the shepherd is placed at a similarly random point in any of the other three quadrants, as shown in the example setup of Figure 1. It is the goal of the shepherd to herd the sheep

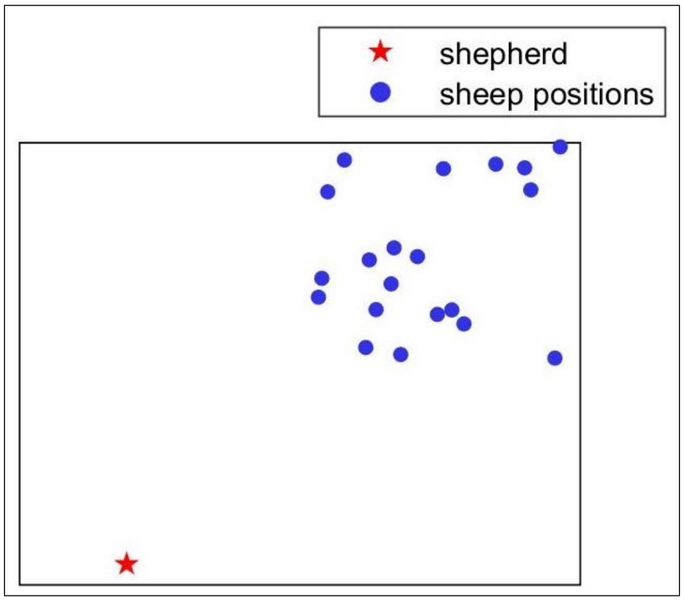

Fig. 1. An example initial setup of the shepherd and sheep.

flock's global centre of mass (GCM) to a predetermined short distance from the lower-left corner of the paddock. Doing so ends the simulation successfully. If the shepherd does not achieve its goal inside a predetermined period of time, the simulation ends as a failure. In the heuristic models by both Strömbom et al. and Baxter, although the paddock boundaries are marked, they are only used as fixed limits for the generation of the agents in the simulations. That is to say, none of the agents are actually contained by the paddock boundaries once a simulation has started - they may pass through the boundaries unhindered. Although this is disregarded as a trivial concern in the work by Strömbom et al., it can have significant implications on how simulations are completed. This is discussed further in section VI.

Once a simulation has started, the shepherd and sheep will interact according to predetermined behaviours until simulation success or failure is determined. These behaviours are described below and an example of such interactions is shown in Figure 2.

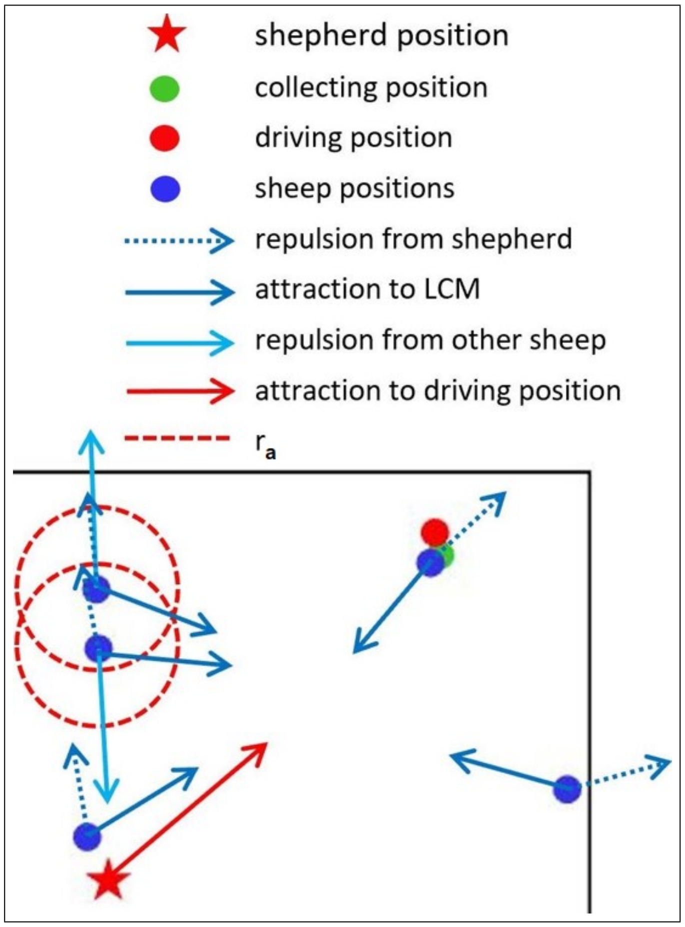

Fig. 2. An example of the interactions between the shepherd and several sheep.

A. Original Sheep Dynamics

In the paper by Strömbom et al., the sheep have behaviours that can be triggered. For the execution of these behaviours, relevant variables are distance, direction and velocity. These are technically in dimensionless units, although metres and time steps are referred to in the work by Strömbom et al. Details of the sheep dynamics are below:

- number: the simulated numbers of sheep was between 20 and 201;
- speed: all sheep are restricted to moving a distance of 1 m per time step. This means that any sheep that experiences velocity vectors must have the sum of those vectors normalised to unity before calculation of its position in the next time step;
- repulsion from other sheep: if any sheep moves inside a specified radius (r_a, set to 2 m) of the active sheep, then the active sheep experiences a velocity vector of 2 units in the opposite direction from the first sheep;
- repulsion from the shepherd: if the shepherd moves within a specified radius (r_s, set to 65 m) of the active sheep, then the active sheep experiences a velocity vector of 1 unit in the opposite direction from the shepherd;
- attraction to local centre of mass (LCM): if the shepherd is inside r_s of the active sheep, then the sheep also experiences a velocity vector of 1.05 units in the direction of the LCM of its n closest neighbours;
- attraction to previous direction of movement: in order to smooth out the turning of a moving sheep, it experiences a velocity vector of 0.5 units towards its previous time step's direction of movement;
- random movement: the sheep have a 5% probability of experiencing a random velocity vector of 1 unit based on no interactions with the shepherd or any other sheep (to simulate realistic grazing behaviour);
- noise: to simulate reality, each sheep has angular noise of 0.3 units configured to prevent deadlock scenarios and simulate imperfect agents.

The goal of the original model was to simulate the sheep as realistically as possible and to reduce the impact of particular behaviours (for example, a sheep conducting a 180° turn in one time step and having the same velocity magnitude but now in the opposite direction).

B. Original Shepherd Dynamics

Much like the sheep, the shepherd has its own attributes and experiences competing behaviours. Again, the relevant variables are distance, direction and velocity. The shepherd dynamics are quantified below:

- number: in the heuristic model by Strömbom et al., there is only ever one shepherd;
- speed: the shepherd has a fixed speed of 2 m per time step;
- movement prevention: if the shepherd is within $3r_a$ of any sheep, it receives no velocity vectors for the current time step;
- field of view: the shepherd has a blind spot past ±90° from its direction of movement. This was introduced to prevent it being caught between smaller groups of sheep and not being able to resolve the situation;
- collecting positioning: if any sheep are not sufficiently aggregated within a given radius ($r_a N^{2/3}$) of the flock

GCM[1], then the shepherd receives a velocity vector in the direction of a location that is r_a distance (2 m) from the furthest sheep from the GCM and on the opposite side of that sheep to the GCM - this is the collecting position;
- driving positioning: if the collective body of sheep is aggregated sufficiently, then the shepherd receives a velocity vector in the direction of a location that is a distance $r_a\sqrt{N}$ from the flock and on the opposite side of the flock from the goal position - this is the shepherd driving position;
- collecting behaviour: once the shepherd has moved to the collecting position, it then drives the non-aggregated sheep towards the GCM to force flock aggregation (it is then common to see the shepherd immediately move to the driving position and drive the herd);
- driving behaviour: if the sheep are aggregated and the shepherd is correctly positioned, it then receives a fixed velocity vector in the direction of the goal location, forcing the herd to also move in that direction;
- noise: as per the sheep behaviour, the shepherd experiences angular noise in all movements.

Strömbom et al. also experimented with the shepherd being tasked with bringing in smaller groups of sheep at a given time, with the greatest success rate found to be for $n = 20$.

III. Revised Problem Dynamics

The goal of this project was the implementation of neural networks to solve a given problem. The networks were not required to solve the problem faster or more efficiently than the original heuristic; it was only necessary to solve a structured problem involving pattern-matching tasks. In fact, it was identified that some complexities of the original model were non-essential to the overall problem construct and that their removal or amendment would still allow the neural networks to achieve a shepherding task. That being said, it was also logical to include one additional dynamic to the problem: the shepherd and the sheep were limited by the boundaries of the arena. The inclusion of this dynamic and its impact is discussed further in Section VI.

In solving the shepherding problem, many thousands of simulations have been run to train and then evaluate the effectiveness of the neural networks under a range of conditions, with each simulation running for up to 1,000 time steps. Therefore, reduction of complexity without any great change to system dynamics was determined to be highly desirable to reduce overall computation, provided the fundamental interactions between sheep and shepherd were still present. As a result, several problem parameters were adjusted or removed so that the behaviours of the shepherd and sheep were less variable and more straightforward. Details of these changes are below.

[1]N is the total number of sheep being simulated.

A. Revised Sheep Dynamics

For the conduct of the neural network design, training and ideal testing, the following changes were made to the sheep dynamics:
- number: the amount of sheep simulated was fixed at 20 sheep;
- attraction to LCM: given the fixed number of sheep, this was revised to attraction to GCM only;
- not simulated:
  - attraction to previous direction of movement;
  - angular noise;
  - probability of random velocity vectors.

The number of sheep simulated became a testing variable for non-ideal conditions later during the analysis of network performance.

B. Revised Shepherd Dynamics

The following changes were made to the shepherd dynamics:
- collecting and driving: it was found that merely by positioning the shepherd at either the collecting or driving position, the respective tasks of collecting and driving the flock were achieved through the reactive behaviours of the sheep. Therefore, the behaviours of the shepherd were limited to adopting the collecting or driving positions but never actually collecting or driving the flock. This has the same effect regardless, as the collecting and driving positions are updated every time step based on the current distribution of the flock;
- not simulated:
  - the shepherd blind spot;
  - angular noise.
- bringing in smaller groups of sheep: given the fixed flock size of 20 sheep, this was not required.

Although the shepherd was never given the behaviour of splitting the flock into sub-flocks in order to bring in smaller groups, as the number of sheep was used as one of the non-ideal testing parameters, it became apparent that this behaviour was actually redundant. This is discussed further in Section VII.

IV. Selection of Neural Networks

Neural networks are pattern-matching devices. Given one or more inputs and one or more target outputs, a neural network can be trained to identify a pattern (if there is one) in the inputs that corresponds to the associated output(s). Examples of this are:
- predicting the fuel consumption rate of a car based on velocity and engine revolutions per minute;
- estimating the body mass index of a person based on body measurements.

Accurate results for these kinds of problems can require many, many training scenarios. Although neural networks are designed (loosely) after the operation of the human brain, they do not operate this way to find patterns in reality.

The simplest neural networks (either the multilayer perceptron or sigmoid neuron) are a combination of three layers: an input node layer, a hidden (secondary decision) node layer and an output node layer. Theoretically, a network can contain multiple hidden layers of multiple hidden nodes and the determination of both number of layers and number of nodes is contextualised by the problem and how it is being solved. In general, a neural network can be seen to apply the following matrix multiplication for a single layer of hidden nodes:

$$\begin{bmatrix} a_0^{(1)} \\ a_1^{(1)} \\ \vdots \\ a_N^{(1)} \end{bmatrix} = \Omega\left(\begin{bmatrix} w_{0,0}^{(0)} & w_{0,1}^{(0)} & \dots & w_{0,N}^{(0)} \\ w_{1,0}^{(0)} & w_{1,1}^{(0)} & \dots & w_{1,N}^{(0)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K,0}^{(0)} & w_{K,1}^{(0)} & \dots & w_{K,N}^{(0)} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_N^{(0)} \end{bmatrix} - \begin{bmatrix} b_0^{(0)} \\ b_1^{(0)} \\ \vdots \\ b_N^{(0)} \end{bmatrix} \right)$$

Here, $a_i^{(1)}$ is the ith hidden node of layer one, $a_i^{(0)}$ is the ith input node of layer zero, $w_{j,k}^{(0)}$ represents a scalar weighting coefficient, $b_i^{(0)}$ is the ith bias element of layer zero and Ω is a forcing function to force the right side equation outputs to be within specified limits. Effectively, the inputs to the hidden node layer are represented as the weighted sum of the input layer (or previous layer in a multilayer network) minus a bias vector. This will then lead to inputs to the output layer for network decision-making. After each training scenario, weight and bias element values are updated to reduce the difference between the network output and the target output. Commonly, in multilayer perceptron training networks, this is done through a gradient-descent method using error back-propagation.

The primary method of training a neural network using Matlab's DLT is by using the Levenberg-Marquardt (LM) second-order algorithm. This algorithm uses more memory but takes far less time than the well-known error back-propagation method [8]. Although the LM algorithm can solve complex pattern-matching tasks with fewer hidden nodes, the memory requirement was deemed to be insignificant in determining neural network parameters for the shepherding problem. Detailed differences in individual computation are discussed in [8] and so will not be repeated here. For the networks used, a single hidden node layer of 16 nodes was found to achieve the training tasks sufficiently.

To solve the problem of shepherding a flock of sheep, a decision needed to be made on what the neural network or networks would need to achieve. One possible option was to provide a neural network with a distribution of sheep and the location of the shepherd and have it estimate a target output vector. However, given the possible distributions of sheep and different behaviours that the shepherd would need to adopt, it was deemed that the nuance of different situations was beyond the scope of one network. Rather, it was determined that it would be more appropriate to hard-code the behaviours of the shepherd based on a set of inputs that were decided by multiple neural networks. Specifically, it was decided that four neural networks would be required to provide the shepherd with sufficient information to achieve its task. These networks were:

- GCMNetwork:
  - input: the x and y coordinates of all 20 sheep;
  - target output: the x and y coordinates of the GCM[2].
- isClusteredNetwork:
  - input: the x and y coordinates of the GCM and those of the furthest sheep from the GCM;
  - output: a 0 or 1 representing a decision of whether the flock is clustered enough for the shepherd to move to the driving position.
- collectingPosNetwork:
  - input: the x and y coordinates of the GCM and the furthest sheep from the GCM;
  - output: the x and y coordinates of the collecting position.
- drivingPosNetwork:
  - input: the x and y coordinates of the GCM, as well as x and y coordinates of the 'back of the flock'[3];
  - output: the x and y coordinates of the driving position.

Originally, all networks were provided with only the x and y coordinates of the sheep as inputs. However, it was found that in the case of the collecting and driving position networks, they were not able to find a pattern between the distribution in the arena and a particular point (despite this being possible for the GCMNetwork). Thus, extra computation for the inputs was required.

These neural networks were trained with static scenarios, where the inputs and target outputs were programmatically derived. This allowed the networks to estimate their outputs based on perfect heuristic target outputs and minimise possible error.

V. Development of Training Scenarios and Provision of Data

In order to train the four selected networks, the shepherd position was not required as part of any simulation. With ideal testing conditions as a goal, the only simulation requirements were fixed arena dimensions and random distributions of sheep. For arena dimensions, these were held at the height and width of the arena being 150 m. To ensure random sheep distributions, the following limits were applied in order to generate the sheep inside a randomised area within the arena:

- x coordinate upper limit: $outerModx = rand(1) * width$;
- y coordinate upper limit: $outerMody = rand(1) * height$;
- x coordinate lower limit: $innerModx = rand(1) * outerModx$;

[2]This GCM output was then used as the GCM input for the other neural networks.

[3]The 'back of the flock' was programmatically determined by finding the distance from the GCM to the furthest sheep and then rotating around to the point behind the GCM that was in line with the flock orientation to the goal.

- y coordinate lower limit: $innerMody = rand(1) * outerMody$.

In Matlab, $rand(1)$ will generate a (pseudo)random number on a scale between 0 and 1. Once this step was completed, each sheep was generated with randomised x and y coordinates that were forced inside the above limits. The detailed script for the provision of data to the networks is provided at Appendix A.

To ensure the networks were provided with enough data that featured sufficiently random distributions over the entire arena (as the flock could theoretically be forced towards any part of the arena), 20,000 static[4] training scenarios were generated for the networks to be trained on. The neural networks were trained on a subset (typically 70%) of the training data, with the network outputs being adjusted after every training sample according to their error (difference from target). The remainder of the data is split between validation (used to measure network generalisation and stop training if generalisation improvement ceases - typically 15% of the data) and testing (used to provide an independent measure of the network performance of its task - typically 15% of the data). For the four neural networks that were trained, the mean squared error (MSE) was recorded as per Table I. With MSE as large as 4.43×10^{-2}, it was found

Network	MSE
GCMNetwork	4.79×10^{-8}
isClusteredNetwork	2.26×10^{-2}
collectingPosNetwork	4.43×10^{-2}
drivingPosNetwork	8.37×10^{-3}

TABLE I
MEAN SQUARED ERROR FOR NETWORK OUTPUTS COMPARED TO TARGET OUTPUTS OVER 3,000 TESTING SCENARIOS

that the network performance was still very satisfactory in terms of absolute differences[5]. In order to assess network performance more meaningfully, however, the measure of success for each network was individually determined as per Table II. Here, μ corresponds to the mean absolute difference

Network	Measure of success
GCMNetwork	$\mu < 1$
isClusteredNetwork	$\mu + 2\sigma < 0.5$
collectingPosNetwork	$\mu < 1$
drivingPosNetwork	$\mu < 1$

TABLE II
MEASURE OF WHETHER A NETWORK WAS TRAINED SUCCESSFULLY FOR EACH NETWORK

between the network outputs and the heuristic outputs, and σ corresponds to the standard deviation. For the three networks that result in positions as outputs, the measure of success was determined as such due to the ideal arena dimensions. With arena width and height equal to 150 m, a mean difference of output less than 1 means less than a 1% difference if the flock was spread over the entire arena and less than a 2% difference for standard initial flock distributions (where the entire flock begins contained within the upper-right quadrant of the overall arena). Nevertheless, the networks achieved mean differences approximately five times less than their required measures, satisfying the criteria comfortably.

The isClustered network has previously been established as needing to match a target output of 0 or 1. In order to ensure that the Matlab simulations used for Section VI operate correctly, the output of this network was subsequently rounded to the nearest appropriate output as per the following:

$$\begin{cases} 1, & 0.5 \leq output \\ 0, & output < 0.5 \end{cases}$$

Therefore, the network output needs to be less than 0.5 different from the target output in order to provide a correct solution. Otherwise, it will be rounded to the wrong solution. This is an example of a binomial distribution, which can be approximated by a normal distribution with the same mean and standard deviation if:

$$n * p > 5$$

where n is the number of data samples and p is the probability of success. Given a large set of data and any reasonable expectation of success, then the isClustered network data can thus be assumed to be normally distributed. For a normal distribution, 95% of the outputs fall within two standard deviations of the mean. Therefore, the condition:

$$\mu + 2\sigma < 0.5$$

ensures that approximately 95% of results have an output less than 0.5 different from the target, which will be rounded to the correct solution (0 or 1). As is discussed in Section VI, simulations had a runtime up to 1,000 time steps, meaning that a 5% probability of an incorrect result from the isClustered network in a single time step corresponds to a 0.25% probability of an incorrect result twice in a row, and a 0.0125% probability of an incorrect result three times in a row. Therefore, even if the network was incorrect in its determination of the clustering of the flock for one time step, this was unlikely to have a significant effect on the overall outcome of a simulation if the network achieved its defined measure of success. To see if the four networks achieved their respective measures of success, they were tested with 10,000 independent static cases for absolute differences between target outputs and network-determined outputs. The mean differences and standard deviations are recorded in Table III and a single testing scenario with the neural network outputs is presented in Figure 3 (both overpage).

[4]The term static is used to refer to a single time step of a simulation, in which there are no dynamic interactions between agents.

[5]It was assumed that the high MSE of the isClusteredNetwork was due to the fact that the target output was always a 0 or 1 and the network output did not have this restriction - it could take any value.

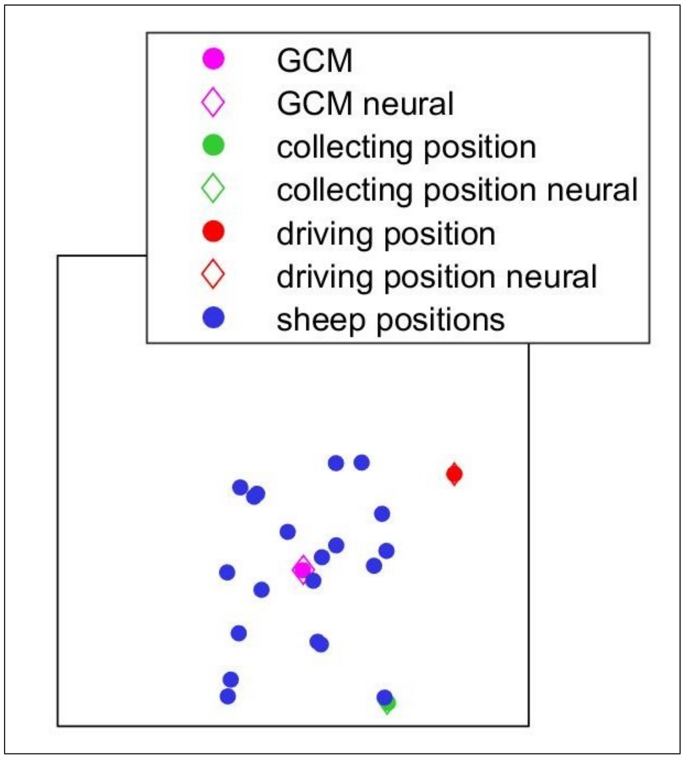

Fig. 3. An example testing scenario for the neural networks.

Network	Mean difference (μ)	Standard deviation (σ)
GCMNetwork	7.510×10^{-4}	5.393×10^{-4}
isClusteredNetwork	0.014	0.118
collectingPosNetwork	0.218	0.215
drivingPosNetwork	0.059	0.106

TABLE III
MEANS AND STANDARD DEVIATIONS OF THE DIFFERENCES BETWEEN THE NEURAL NETWORK OUTPUTS AND THE TARGET OUTPUTS

The most extreme differences recorded from this testing were:

- GCM: 0.01;
- collecting position: 2.12;
- driving position: 3.42.

However, it can be observed that the networks all achieved their measures of success. With this satisfied, the networks were able to be tested under full dynamic simulation conditions.

VI. Full Simulation Testing Under Ideal Conditions

A full dynamic simulation under ideal conditions had the following parameters:

- arena dimensions of 150 m $\times$ 150 m;
- arena boundaries prevented all agents from exiting of passing through them;
- goal position at the bottom-left corner of the arena;
- maximum number of time steps equal to 1,000;
- 20 sheep, generated randomly within the upper-right quadrant of the arena;
- one shepherd, generated randomly in any other quadrant of the arena;
- revised sheep and shepherd behaviours as per Section III;
- success condition: both shepherd and driving position within 20 m of the goal position[6].

A typical simulation first time step is shown in Figure 4. As

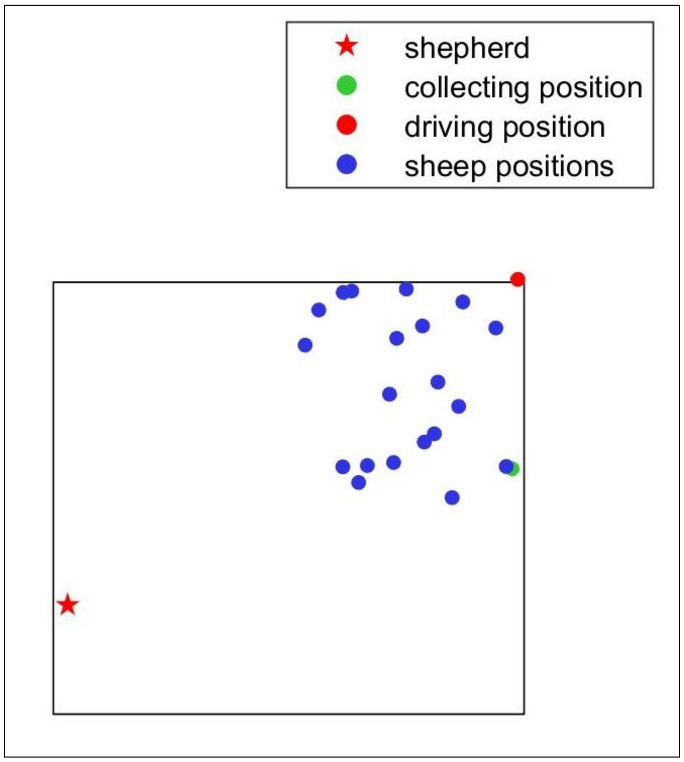

Fig. 4. First time step from a random simulation test.

mentioned in Section IV, the shepherd and sheep had their revised behaviours hard-coded and the neural networks were responsible for providing inputs to the shepherd that could then be turned into actions. It was decided that the most accurate way to quantify the performance of the neural networks was to run two identical input simulations. In the first, the shepherd received inputs from heuristic calculations. In the second, the shepherd received inputs from the neural networks instead. Importantly, in both simulations, the initial conditions were the same[7] but once the simulations had started, the outcome was variable depending on the inputs given to the shepherd.

The simulations were given a maximum number of time steps in order to break out of any deadlock situations where the shepherd was functionally prevented from reaching its goal. This occured when, for example, the shepherd was not able to move to the appropriate position because it was on the other

[6]If this has occurred, the shepherd has successfully herded the flock of sheep.
[7]The only independent variables for these simulations were the starting positions of the sheep and shepherd.

side of the flock, which was pinned in a corner of the arena[8]. This can be seen in Figure 5, where the shepherd has forced just such a deadlock situation. If the maximum number of time steps was reached in a simulation, it ended as a failure. Although it is likely that not including functional boundaries would have resulted in less (or no) deadlocks, it is asserted that the inclusion was a more necessarily realistic dynamic to the problem than any of the behaviours that were removed.

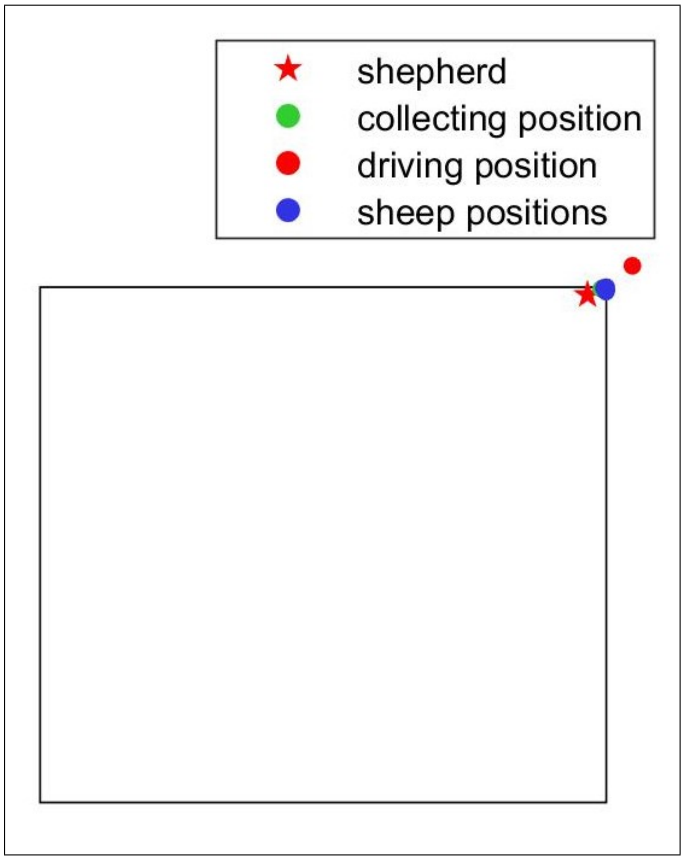

Fig. 5. Shepherd causing a deadlock scenario with the pinned flock of sheep.

Given the measures of success determined in Section V for the training of the networks, it was deemed appropriate that a similarly high expectation would be reasonable for the networks' dynamic performance. To that end, a requirement was set for the neural networks to achieve a quantified performance of less than 1% worse than the heuristic solution. The following was used to measure the performance of the neural networks as compared to the heuristic:

- mean number of time steps taken in a simulation;
- standard deviation of time steps taken;
- total percentage of failures.

If any simulation reached the maximum number of time steps, it was recorded as a failure but disregarded from the mean and standard deviation calculations for that solution method. 100,000 pairs of simulations were run to determine how well the neural network inputs performed against the ideal heuristic. The details of these are shown in Table IV. As can be seen in the table, the networks achieved more favourable mean and standard deviation results than the heuristic solution. However, it is important to note that these calculations are not conducted on failed simulations, for which the networks performed worse than the heuristic. On the total simulations conducted, the failure rate of the neural networks was 0.925% higher. Given the above performance measure only applies to this metric (as the mean and standard deviation are better), this was deemed acceptable and thus, the neural networks were deemed to have comparable performance to the heuristic solution. This meant that the networks did not require a revised regime of training and allowed for non-ideal conditions testing to be conducted.

Solution	Mean time steps taken	Standard deviation of time steps taken	Failure %
Heuristic	341.691	67.613	3.576
Neural network	337.842	62.551	4.501

TABLE IV
MEASURED PERFORMANCE OF THE NEURAL NETWORKS AGAINST THE HEURISTIC SOLUTION UNDER FULL SIMULATION CONDITIONS

VII. Full Simulation Testing Under Non-Ideal Conditions

Full dynamic simulations under non-ideal conditions contained all of the same parameters as those under ideal conditions, except where specific parameters were varied in order to test the performance of the neural networks. These specific parameters were:

- arena size;
- number of sheep simulated;
- the addition of error to the inputs of the neural networks.

Measurement of performance of the neural networks under the non-ideal conditions was not compared to an ideal heuristic performance, but to that of the networks under ideal conditions[9]. This was because the purpose of non-ideal conditions testing was to test how the networks performed when they were given situations that were different to those they had been trained on. A secondary objective was to identify the limits of how inputs could be varied so that a clear statement could be made about what the neural networks could reliably achieve and what they could not. Performance for each parameter change was measured independently. Therefore, each parameter will be discussed separately.

A. Arena Size

The neural networks were trained on random distributions of sheep within a static area of 150 m × 150 m. In terms of Matlab vector coordinates, each sheep had a possible x and y range of 0 to 150; the neural networks were never trained on numbers greater than 150. Therefore, testing

[8]This scenario would never have occurred in the original model by Strömbom et al. as the arena boundaries had no impact on either type of agent. The shepherd may have pushed the sheep outside of the ideal area but would eventually be able to get around them to the desired position.

[9]That being said, heuristic performance was required to provide an objective measure of non-ideal condition performance in some instances.

of arena size variation was conducted. The arena shape was always held fixed as a square; that is to say, that variation in height also corresponded to variation in width.

Two tests were conducted. First, mean and standard deviation calculations were conducted on the difference between ideal network performance and the performance of the networks under different arena sizes for single, static cases without agent interaction (as per the training instances). Second, failure rate calculations were conducted. The arena size was varied from 160 m $\times$ 160 m to 300 m $\times$ 300 m, in increments of 10 m. Sizes less than the original arena were deemed not necessary as theoretically, the networks were trained on flock distributions that fell within sub-areas of the total arena. Figure 3, for example, shows a flock distribution that falls inside a 120 m $\times$ 120 m area, which would make this increment redundant.

For single, static cases, 10,000 measurements were taken per increment. The mean absolute difference in performance of each network compared to the ideal case is shown in Figure 6. Although the networks' inaccuracy increases as the arena

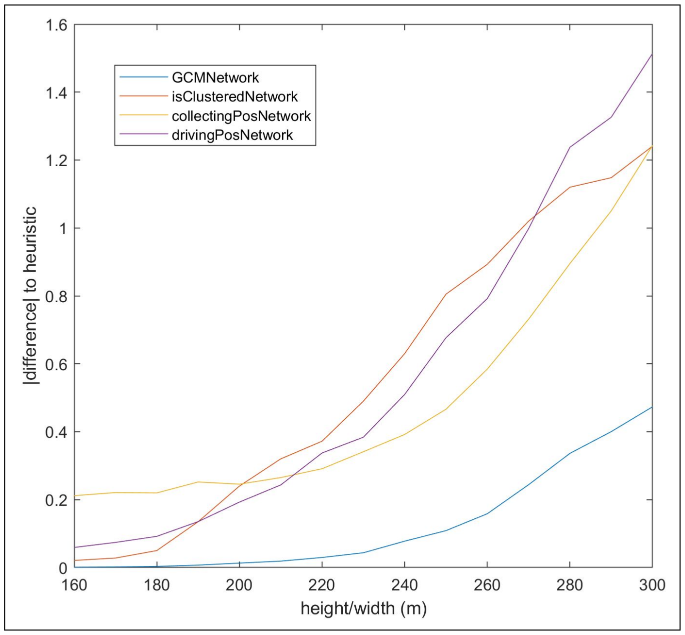

Fig. 6. Mean absolute differences in performance of each network as arena size is varied (for 10,000 samples per increment).

size increases, a doubled arena size still only corresponds to a maximum positional inaccuracy of 1.514 m (for the drivingPosNetwork). For arena sizes of 270 m $\times$ 270 m or less, this is also still inside the original specified measures of success for all except the isClusteredNetwork. Unfortunately, the isClusteredNetwork fails its measure of success ($\mu + 2\sigma < 0.5$) at arena sizes of 180 m $\times$ 180 m or greater (at this size, $\mu + 2\sigma = 0.759$, meaning the probability of an incorrect answer is greater than 5%). Therefore, although the other three networks maintained their accuracy to a high degree, this critical network output was potentially a breaking point for the shepherd in a full simulation, as it is the output that determines if collecting or driving behaviour is necessary.

For the full-length simulation, 10,000 simulations were conducted per arena size increment to determine the network failure rate relative to the ideal conditions. These failures rates are plotted in Figure 7. As can be seen from the plot,

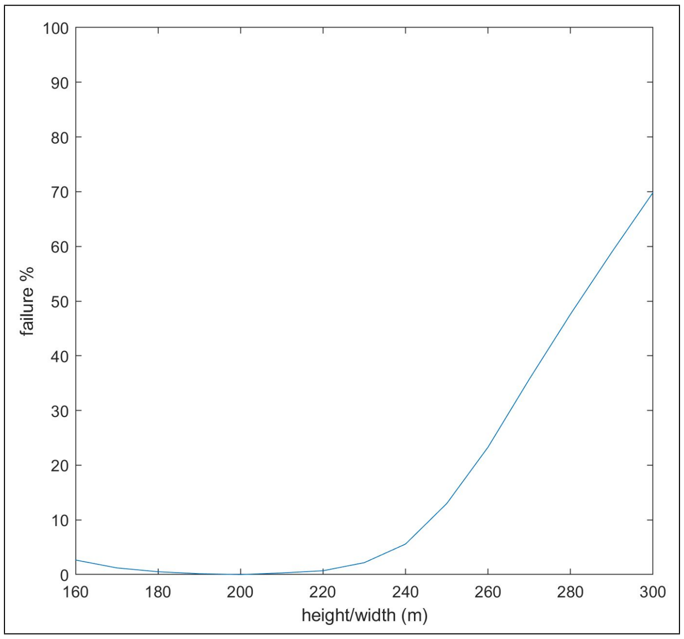

Fig. 7. Failure rates for the shepherd in full-length simulations as arena size is varied (for 10,000 samples per increment).

despite the isClusteredNetwork having a significantly higher error rate at lower arena sizes, it isn't until the arena is 240 m $\times$ 240 m that this begins to correspond to significant failure rates for full simulations. At the upper limit of 300 m $\times$ 300 m, the failure rate for the network shepherd is 69.86%. This, as well as the zero error rate at arena sizes of 200 m $\times$ 200 m was investigated further and it was found to definitely be related to the isClusteredNetwork accuracy.

For the upper limits of arena size, the isClusteredNetwork was deciding that the flock of sheep was clustered enough for the shepherd to move to the driving position when it truly wasn't. This would lead to the situation present in Figure 8. Here, the shepherd has adopted the driving position but isn't physically within range of any sheep to influence them. Thus, the simulation will continue in this state until failure is determined.

For arena sizes of 200 m $\times$ 200 m, this situation was often true as well. However, simply due to the fact that all sheep are likely to be generated inside a total diameter less than the shepherd influence[10], the isClusteredNetwork misidentifying whether the flock is clustered still allows the shepherd to complete its task by driving straight away.

[10]The shepherd influence is 65 m, and at this arena size, the maximum distance between two sheep is 70.7 m in the first simulation time step.

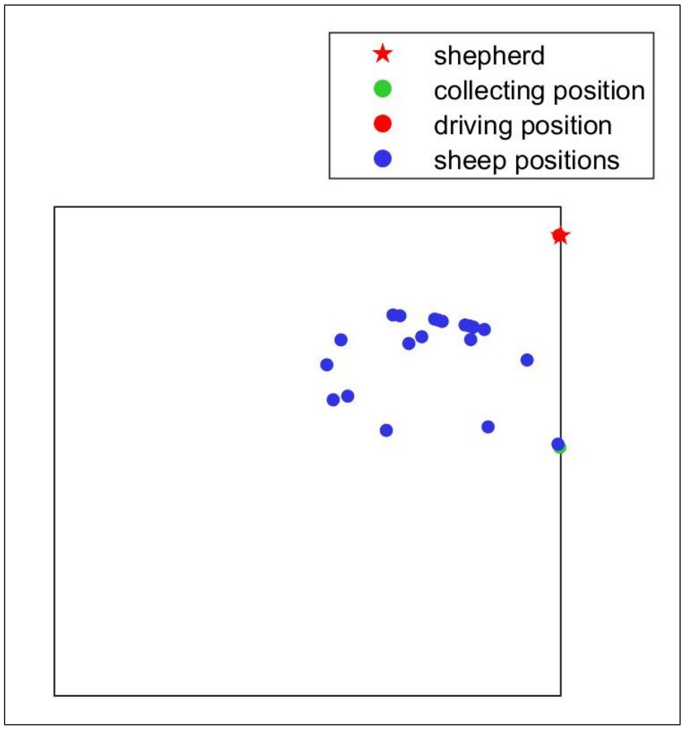

Fig. 8. Shepherd attempting to drive a non-clustered flock (arena size 300 m $\times$ 300 m.

B. Number of Sheep Simulated

In Matlab's DLT, the size of a vector input is fixed and one-dimensional. Therefore, it is not possible to train a network on an input or output vector that varies in size[11]. This presented issues for testing of the four trained neural networks as the size of the tested input must be the same size as the input used in training.

To overcome this issue, particularly with respect to changing the number of sheep, some expansion or compression of the input data needed to be conducted before it could be given to the networks. The code provided at Appendix B was applied to the matrix of sheep so that the end result was 20 meaningfully averaged x and y coordinate sets. This could then be passed to the networks, which then governed the behaviours of the shepherd.

If less than 20 sheep were provided, then the code at Appendix B would fill up the sheep matrix to 20 by adding new elements at the GCM of the flock plus some small noise characteristic. These positions would then be updated as per the other, 'real' sheep. Although not a perfect representation of the true distribution of sheep, this workaround still allowed testing to be conducted.

If more than 20 sheep were provided[12], then the following logic was applied:

- determine the lowest multiple of 20 that is higher than the number of sheep, $sheepUpperLimit$;
- find the total amount of sheep between $(sheepUpperLimit - 1)$ and the total number of sheep - this is $sheepCollectMultiple$;
- for 1 up to $sheepCollectMultiple$, collect sets of $sheepUpperLimit$ sheep together and take their mean;
- the mean sheep locations will go into the fixed-limit matrix, along with any leftover sheep locations that weren't added to a multiple.

The above logic is shown more clearly in the following example. The number of sheep is 23. The lowest multiple of twenty greater than 23, $sheepUpperLimit$, is 2 $(2*20 = 40)$. $sheepCollectMultiple$ is the sum of sheep greater than $(sheepUpperLimit - 1)$, which is 3. Therefore, from 1 up to 3, sheep are collected and their means are taken. This means sheep 1 and sheep 21, sheep 2 and sheep 22, sheep 3 and sheep 23. All other sheep positions are unchanged. The determined mean positions and the unchanged sheep positions go into the final sheep matrix. Again, although this method is a workaround, it provides the shepherd with a fixed overall distribution of sheep to work with.

To test the neural network performance, only full simulation testing was conducted, for which mean and standard deviations of total time steps were taken. This was conducted from 5 sheep up to 200, at increments of 5 sheep, with 1,000 samples taken per increment. The means and standard deviations for total time taken are presented in Figure 9. From

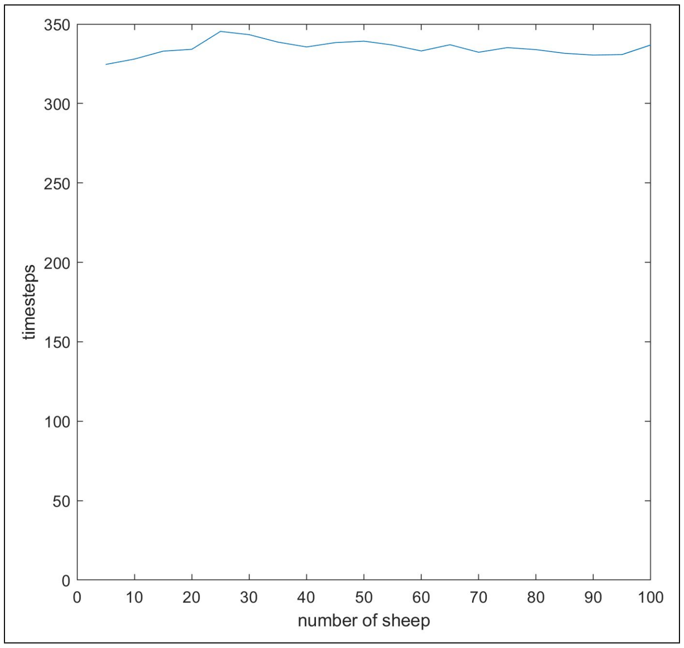

Fig. 9. Means and standard deviation of shepherd completion time as the number of sheep is varied from 5 to 200 (in increments of 5).

[11]This is not strictly true. A network input vector can vary in size. However, the DLT requires all inputs to have the same length as the largest input, which would mean that for a majority of inputs, there would be spurious zeros present. This could lead the network to learn to ignore zeros, which could have no impact on its performance of its task or could cause anomalous conditions if a real zero-valued input was present (either in training or testing). To that end, the neural network vector lengths were considered to be fixed and requiring real inputs for all elements.

[12]This was exhaustive and limited to a maximum upper limit of 400 sheep.

this, it can be seen that the number of time steps taken for the differing numbers of sheep remains relatively constant. Even for 100 sheep, the mean time steps taken is 336.86. This was investigated further by inspecting several full simulations. As can be seen in Figure 10, there is a 'wave' effect when the shepherd influencing range, r_s, reaches the edge of the flock. The first layer of sheep move away from the shepherd,

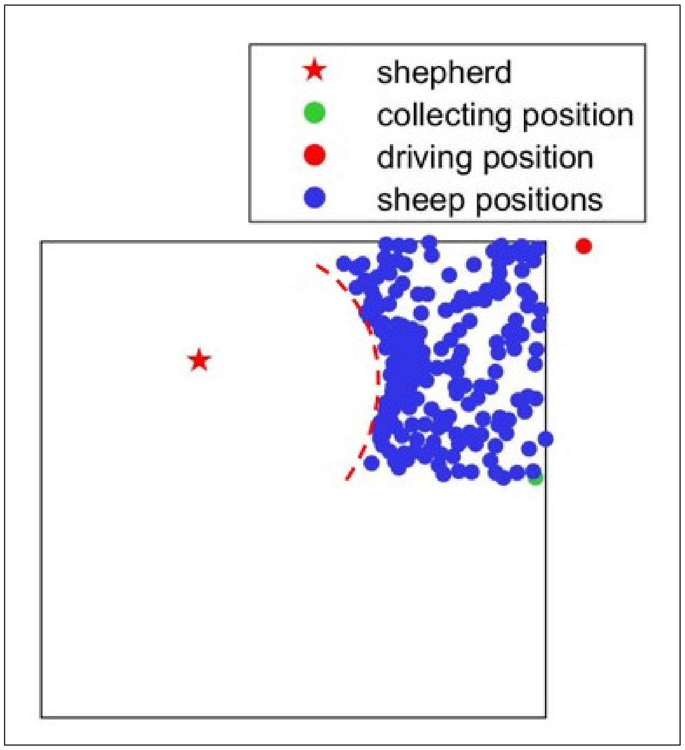

Fig. 10. Shepherd 'wave' effect when herding large flocks (the dashed line indicates r_s).

which causes any sheep inside r_a of another sheep to be repelled by it. This has a cascading effect which becomes far more prominent as the number of sheep is increased. Due to this effect, as well as the size of the arena and the initial distribution of sheep being fixed in the upper-right quadrant of the arena, it was observed that the behaviour of the shepherd that brought in smaller groups of sheep (as per the original model by Strömbom et al., mentioned at the end of Section II) was redundant. Having said that, it may not have been the case if particular problem parameters were changed, such as the arena size and numbers of sheep being scaled up together.

C. Error Added to Inputs

For this testing, error was applied to simulate uncalibrated instruments for the shepherd's neural networks. Only full-length simulation performance of the shepherd was analysed. The error that was added to the inputs of the networks was based on the positions of the sheep and the shepherd at the start of a simulation. Specifically, a randomised scalar was added to or subtracted from the true position of each sheep in both the x and y direction at the start of a simulation, giving it a possible maximum radius of error, as shown

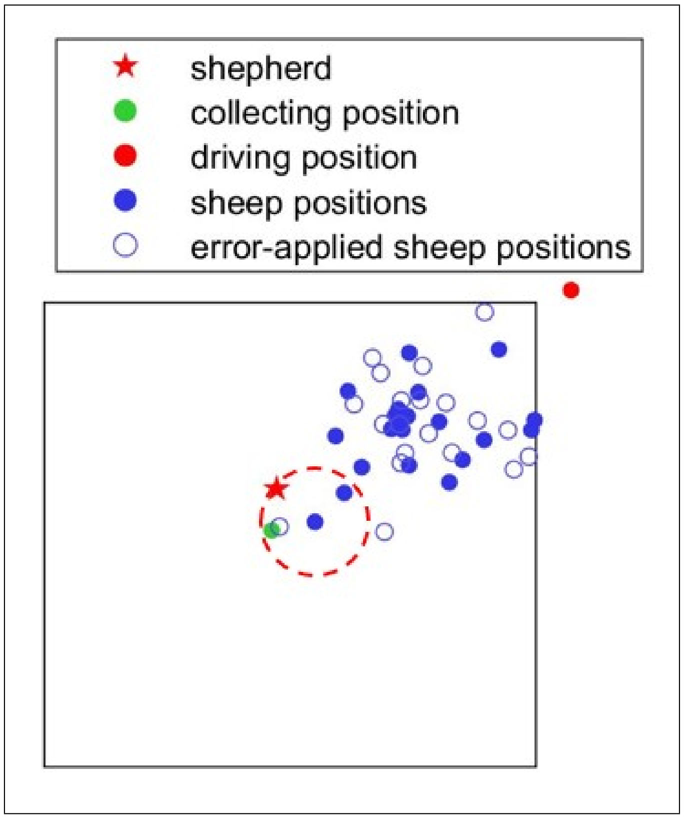

Fig. 11. The maximum error radius shown on the furthest sheep from the flock GCM.

on the furthest sheep in Figure 11. In this example, the shepherd has a clearly wrong determination of where the collecting position is, due to the error. The maximum error radius is scaled by how far the shepherd was from the GCM of the flock at the beginning of simulation, imitating a starting level of clarity in the 'vision' of the shepherd.

The error-applied inputs were provided to the neural networks to conduct calculation on, which then governed the behaviour of the shepherd during simulation. As the shepherd approached the flock, the level of error in the inputs was linearly reduced up until a 'clarity' threshold, set to be 40 m from the GCM of the flock[13]. Once the shepherd reached this threshold, the error in the inputs was more significantly reduced every time step, such that it is functionally zero after a very short period, resulting in the shepherd vision becoming clear[14].

The neural networks were not directly the subjects for the error testing. If error-applied inputs were provided to the networks, they would have no way to understand the 'real' sheep locations and thus, measurement of the individual network accuracy would be meaningless. Rather, the intent of the error testing was to determine if the overall performance of

[13]This distance was selected as from observation, the shepherd was typically 40 m from the flock GCM when switching from collecting to driving behaviour. This indicates that the shepherd can 'see' clearly where all of the sheep are at that point in the simulation.

[14]This short period is approximately 3 or 4 time steps depending on the initial level of error that was generated.

the shepherd varied significantly when error was present in the sheep inputs, or if the shepherd could still function properly.

To measure the effect of the error in the inputs, means of total time steps were taken for error scaling factors between 0.05 and 1, incremented by 0.05, with 1,000 samples taken per increment. Effectively, the error scaling factor can be considered to provide maximum and minimum error radii for each sheep. If we consider the average starting flock GCM being in the centre of the upper-right quadrant (112.5, 112.5), then the error can be considered in the extreme cases as per Table V. As can be seen from the table values, at

Error scaling factor	Minimum error radius	Maximum error radius
0.05	2.7	11.3
0.50	26.5	79.5
1.00	53.0	224.9

TABLE V
MINIMUM AND MAXIMUM ERROR RADII EACH SHEEP CAN BE GIVEN FOR MINIMUM AND MAXIMUM ERROR SCALING FACTORS

the upper limit of error scaling factor, the maximum error is greater than the diagonal length of the arena ($\approx$ 212 m). Therefore, it can be safely assumed that the shepherd will not move in the direction of an accurate or useful collecting position[15] and this will lead to longer shepherding times.

As can be seen in Figure 12, the above assertion is not true. For all scaling factors tested, the mean time steps taken

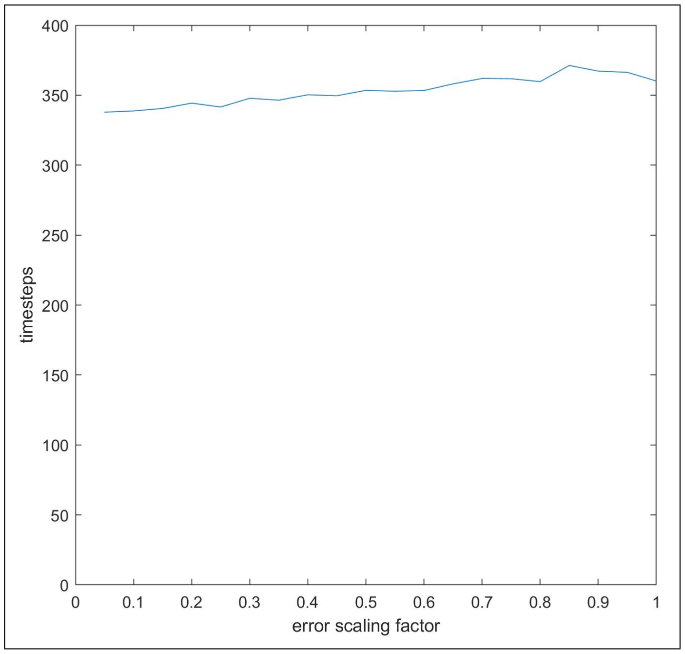

Fig. 12. Mean time steps taken for error scaling factors between 0.05 and 1.00 (incremented by 0.05).

[15]Statistically, the shepherd would only ever move to a collecting position as the high error amount would certainly lead to the shepherd deciding the flock is not clustered.

for the shepherd do not vary significantly; although, they do increase in a somewhat linear fashion. To determine if the scaling factors still allowed the shepherd to be successful in the majority of cases, data on the failure rate of the shepherd was also collected, as per Figure 13. It can be seen that in

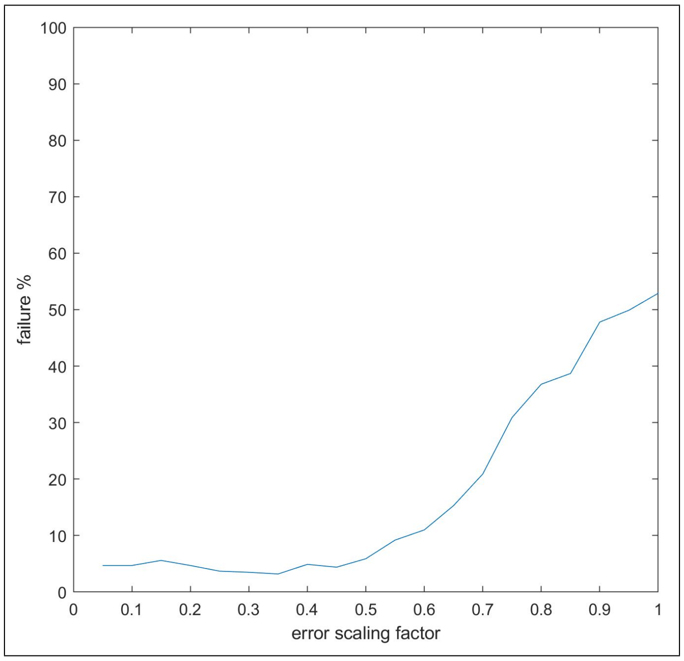

Fig. 13. Mean failure rates for error scaling factors between 0.05 and 1.00 (incremented by 0.05).

this plot, for scaling factors greater than 0.50, the proportion of failures significantly increases. After investigating this further, it was found that the higher error rates corresponded to the collecting and driving positions being far outside of the arena boundaries. As the shepherd could not move through these boundaries, it would remain at the edge of the arena until the simulation ends as a failure.

VIII. LIMITS OF NEURAL NETWORK-BASED SHEPHERDING

As Section VII showed, under clearly defined non-ideal conditions that the neural networks have not been trained on, there are limits to the effectiveness of the shepherd. Based on the results from testing, these are quantified as both minimum and maximum limits of effectiveness in Table VI. The number

Condition	Minimum limit	Maximum limit
Arena size	150 m $\times$ 150 m	230 m $\times$ 230 m
Number of sheep	5	100
Error scaling factor	0.00	0.50

TABLE VI
MINIMUM AND MAXIMUM LIMITS OF EFFECTIVE FOR PARTICULAR NON-IDEAL CONDITIONS THAT WERE PRESENTED

of sheep maximum could potentially be extended further but for the sake of reality, herding more than 100 sheep in a 150 m $\times$ 150 m paddock was deemed impractical.

IX. Conclusion

For this research report, four neural networks were trained to provide information to a shepherding agent. This shepherding agent was then able to use this information to herd a flock of 20 sheep with a success rate comparable to the heuristic-based shepherd. Further testing of the limits of the neural networks included changing the size of the arena the shepherd worked in, the number of sheep to be herded and including error in position-based inputs for the sheep. The limits of how far these parameters could be changed before significant detriment occurred were then quantified. This report shows that for specific pattern-matching tasks, neural networks are viable alternatives to heuristic methds and despite being trained on a specific problem, they can still be relevant within a limit of parameter modification.

References

[1] Daniel Strömbom, Richard Mann, Alan Wilson, Stephen Hailes, Jennifer Morton, David Sumpter and Andrew King. *Solving the Shepherding Problem: Heuristics for Herding Autonomous, Interacting Agents*. Journal of the Royal Society Interface, Volume 11, Issue 100, 2014.

[2] Sheng-Yuan Tu and Ali Sayed. *Cooperative Prey Herding Based on Diffusion Adaptation*. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2011.

[3] Kaoru Fujioka. *Effective Herding in Shepherding Problem in V-formation Control*. Transactions of the Institute of Systems, Control and Information Engineers, Volume 31, Issue 1, 2018.

[4] Meng Li, Zhiwei Hu, Jiahong Liang and Shilei Li. *Shepherding Behaviors with Single Shepherd in Crowd Management*. International Computer Science Conference: Communications in Computer and Information Science, Volume 326, 2012.

[5] Cameron Clark, Sergio Garcia, Kendra Kerrisk, James Underwood, Juan Nieto, Mark Calleija, Salah Sukkarieh and Greg Cronin. *Herding Cows with a Robot: The Behavioral Response of Dairy Cows to an Unmanned Ground Vehicle*. ADSA-ASAS-CSAS Joint Annual Meeting, 2014.

[6] Daniel Baxter. *Swarm Intelligence and its use in Robotic Shepherding: Literature Review*. Internal source.

[7] Daniel Baxter. *Towards a Swarm of UAV Shepherds*. Internal source.

[8] Bogdan Wilamowski and Hao Yu. *Improved Computation for Levenberg-Marquardt Training*. IEEE Transactions on Neural Networks, Volume 21, Issue 6, 2010.

APPENDIX A
NEURAL NETWORK TRAINING SCRIPT

```
height = 150; % length of arena
width = 150; % width of arena
x = 1; % easy for row indexing
y = 2;
numSheep = 20; % the number of sheep being simulated
sheepRadius = 2; % this value is how far the sheep need to be from other sheep
clusterRadius = sheepRadius*numSheep^(2/3);
drivingDistance = sheepRadius*sqrt(numSheep);
trainingNumber = 20000;

% create training instances
for n = 1:trainingNumber
    % pseudo-random limits on sheep spawn locations
    outerModx = rand(1);
    innerModx = outerModx*rand(1);
    outerMody = rand(1);
    innerMody = outerMody*rand(1);
    % reset all variables for this iteration
    sheepMatrix = zeros(20,2);
    directions = zeros(20,2);
    norms = zeros(1,20);
    GCM = zeros(1,2);
    collectingPos = zeros(1,2);
    temp = zeros(1,2);
    furthestIndex = 0;
    furthestNorm = 0;
    scalingFactorCollect = 0;

    % calculate random values for the (x,y) coordinates for each sheep
    for s = 1:numSheep
        sheepMatrix(s,x) = (outerModx*width - innerModx*width)*rand(1)+ innerModx*width;
        sheepMatrix(s,y) = (outerMody*height - innerMody*height)*rand(1)+ innerMody*height;
    end
    % calculate the GCM of the flock
    GCM(x) = mean(sheepMatrix(:,x)); GCM(y) = mean(sheepMatrix(:,y));
    GCMNorm = norm(GCM);
    % calculate the direction vectors from each sheep to the GCM
    for s = 1:numSheep
        directions(s,x) = sheepMatrix(s,x) - GCM(x);
        directions(s,y) = sheepMatrix(s,y) - GCM(y);
    end
    % determine furthest sheep and how far away it is from the GCM
    for s = 1:numSheep
        temp = [directions(s,x) directions(s,y)];
        norms(s) = norm(temp);
        if (norms(s) > furthestNorm)
            furthestNorm = norms(s);
            furthestIndex = s;
        end
    end
    if furthestNorm < clusterRadius
        boolClustered = true;
    else
        boolClustered = false;
    end
    furthestSheepCoords = [sheepMatrix(furthestIndex,1) sheepMatrix(furthestIndex,2)];
    % calculate position of collecting position based on furthestIndex and GCM
    scalingFactorCollect = (furthestNorm + sheepRadius)/furthestNorm;
    collectingPos(x) = (directions(furthestIndex,x)*scalingFactorCollect+GCM(x));
    collectingPos(y) = (directions(furthestIndex,y)*scalingFactorCollect+GCM(y));
    % calculate driving position based on GCM location, back of the flock, and the driving distance
    scalingFactorBackofFlock = (GCMNorm + furthestNorm)/GCMNorm;
    backOfFlock(x) = (GCM(x)*scalingFactorBackofFlock);
    backOfFlock(y) = (GCM(y)*scalingFactorBackofFlock);
    scalingFactorGCM = (GCMNorm + drivingDistance+furthestNorm)/GCMNorm;
    drivingPos(x) = (GCM(x)*scalingFactorGCM);
    drivingPos(y) = (GCM(y)*scalingFactorGCM);

    % reshape to fit the needs of the output
    sheepMatrixReshaped = reshape(sheepMatrix, [numSheep*2,1]);
    GCMReshaped = reshape(GCM,[2,1]);
    backOfFlockReshaped = reshape(backOfFlock,[2,1]);
    furthestSheepReshaped = reshape(furthestSheepCoords,[2,1]);
    collectingPosReshaped = reshape(
        collectingPos ,[2 ,1]);
    drivingPosReshaped = reshape(drivingPos ,[2 ,1]);

    % create collectingNetwork input vector (for test only)
    collectingInput = [GCMReshaped; furthestSheepReshaped];
    isClusteredInput = collectingInput;
    % create collectingNetwork input vector (for test only)
    drivingInput = [GCMReshaped; backOfFlockReshaped];

    % store this training instance
    for s = 1:numSheep*2
        inputGCM(s,n) = sheepMatrixReshaped(s,1);
    end
    for s = 1:2
        outputGCM(s,n) = GCMReshaped(s,1);
        inputCollectingPos(s,n) = GCMReshaped(s,1);
        inputDrivingPos(s,n) = GCMReshaped(s,1);
        outputCollectingPos(s,n) = collectingPosReshaped(s,1);
        outputDrivingPos(s,n) = drivingPosReshaped(s,1);
    end
    for s = 3:4
        inputCollectingPos(s,n) = furthestSheepReshaped(s-2,1);
        inputDrivingPos(s,n) = backOfFlockReshaped(s-2,1);
    end
    inputClustered = inputCollectingPos;
    outputClustered(n) = boolClustered;
end
```

APPENDIX B
CODE TO FORCE INPUT NUMBERS TO 20 SHEEP

```
% we now need to expand or compress our input array into exactly 20
if numSheep == 20
    fillUpSheepMatrix = sheepMatrix;
else
    % if less than 20, we do this by filling the extra spots with averages
    % of all the other sheep positions
    if numSheep < 20
        meanX = mean(sheepMatrix(:,x));
        meanY = mean(sheepMatrix(:,y));
        for i = 1:numSheep
            fillUpSheepMatrix(i,x) = sheepMatrix(i,x);
            fillUpSheepMatrix(i,y) = sheepMatrix(i,y);
        end
        for i = (numSheep + 1):20
            fillUpSheepMatrix(i,x) = meanX + 10*rand(1);
            fillUpSheepMatrix(i,y) = meanY + 10*rand(1);
        end
    % otherwise if we have more than 20, then
    else
        % check up to 400 sheep (this is a lot but just to be safe) - we
        % don't check 1 because that is less than 20
        for a = 2:20
            if numSheep < a*20
                sheepUpperLimit = a;
                break;
            end
        end
        % we now have the upper limit in multiples of 20 for the number of
        % sheep
        % if numSheep is 38, this will go for i = 1:18
        for i = 1:(numSheep-(sheepUpperLimit-1)*20)
            % collect the multiples together
            for b = 0:(sheepUpperLimit-1)
                tempSheepPosition((b+1),x) = sheepMatrix((i+b*20),x);
                tempSheepPosition((b+1),y) = sheepMatrix((i+b*20),y);
            end
            % those values averaged become the filled up index values
            fillUpSheepMatrix(i,x) = mean(tempSheepPosition(:,x));
            fillUpSheepMatrix(i,y) = mean(tempSheepPosition(:,y));
        end
        % and the rest are just single values added in
        for i = (numSheep-(sheepUpperLimit-1)*20+1):20
            fillUpSheepMatrix(i,x) = sheepMatrix(i,x);
            fillUpSheepMatrix(i,y) = sheepMatrix(i,y);
        end
    end
end
```