2018 IEEE International Conference on Robotics and Automation (ICRA)
May 21-25, 2018, Brisbane, Australia

Distance-based multi-robot coordination on pocket drones

Bastian Broecker[1], Karl Tuyls[2] and James Butterworth[3]

Abstract— **We present a fully realised system illustrating decentralised coordination on Micro Aerial Vehicles (MAV) or pocket drones, based on distance information. This entails the development of an ultra light hardware solution to determine the distances between the drones and also the development of a model to learn good control policies. The model we present is a combination of a recurrent neural network and a Deep Q-Learning Network (DQN). The recurrent network provides bearing information to the DQN. The DQN itself is responsible for choosing movement actions to avoid collisions and to reach a desired position. Overall we are able provide a complete system which is capable of letting multiple drones navigate in a confined space only based on UWB-distance information and velocity input. We tackle the problem of neural networks and real world sensor noise, by combining the network with a particle filter and show that the combination outperforms the traditional particle filter in terms of converge speed and robustness. A video is available at: https://youtu.be/yj6Qqh0zpok.**

I. Introduction

Coordination is a challenging issue in multi-robot systems and the more global knowledge a robot has the easier the problem becomes [6]. Such knowledge includes the global position and velocity of a robot and it's coordination partners. Most of the time this information is obtained by employing a global map in combination with vision or laser based localisation [7]. This information is then passed on to a centralised system, or to each robot individually, in order to execute the best movement action.

When working with smaller robots, for instance pocket drones, these techniques are usually infeasible since most localisation techniques like "Vision-based localisation and mapping" are computationally too expensive and the robots might not even be able to carry the required hardware. The MAVs used in this research have a payload of around $20g$, therefore, we are aiming for a light weight hardware solution, which can provide us with enough information to perform multi robot coordination, but doesn't require extensive post-processing such as laser scan matching or image processing. Some work is already going on in a similar direction, for example the system proposed in [10] consists of speakers and an array of microphones. By employing the obtained distance and bearing information, and by applying particle filters, the UAVs are able to detect and track neighbouring robots. Another system is proposed in [18] where ground robots are equipped with infra-red emitters and receivers in a spherical layout; both systems only perform well on smaller distances and need additional processing power to calculate distances and filter external audio/light noises. The system proposed in this paper is based on ultra wide band (UWB) distance calculations (see section III). These sensors are sending radio messages back and forth between two devices. The distance is then calculated by multiplying the duration by the speed of travel. UWB systems are mostly used for global localisation - by placing multiple UWB-chips in fixed known locations, a mobile chip can be tracked by triangulation. Due to the small footprint and payload of our MAVs, we are only able to carry one chip, triangulation is therefore not an option and we have to rely on sequences of distance measurements to provide a sufficient movement strategy. Programming and designing movement policies, e.g. based on state machines, can be a difficult and time consuming job because all edge cases have to be covered and extensively tested. The difficulty is also intensified if the input data is only a partial representation of the environment. In order to tackle this problem we are applying a recurrent neural network in combination with a particle filter, to extract features from a sequence of states to make the world more observable and therefore the application of a movement policy possible.

In the last couple of years recurrent neural networks have shown great performance in tasks which rely on understanding state sequences, for example: natural language processing or playing 3D games like Doom (e.g. [15],[5],[8]). In the field of robot coordination are deep-neural networks mainly applied on ground robots or simulation, where sophisticated sensors (e.g. $360°$ lidar) are able to provide mostly fully observable states, allowing for a more direct input-output mapping (e.g. [13], [17]).

The system presented in this work, is capable of providing navigation and collision avoidance on MAVs with limited state-informations, specifically one dimensional distance measurements to other drones and obstacles. The system applies a learned policy to control the movement of a drone towards its desired target position and avoid all approaching drones on the way. Our model consists of a recurrent neural network for sequential feature extraction, a particle filter to ensure stability in state predictions and a Deep Q-Learning Network (DQN) providing the movement policy.

The remainder of this paper is organised as follows. Section II introduces the main technique the work is relying on. Section III continues with a description of a hardware. Following in Section IV, is a detailed structure description of the employed network, its training and performance is illustrated in Section V. Concluded is the paper with the experimental result in Section VI and the conclusion in

[1]Department of Computer Sciences, University of Liverpool, broecker@liv.ac.uk
[2]DeepMind, karltuyls@google.com
[3]Department of Computer Sciences, University of Liverpool, sgjbutte@student.liv.ac.uk

Section VII.

II. Background

Below we give a brief introduction to radio-based distance estimation, Deep Q-Networks, Particle Filter and Recurrent Neural Networks.

A. Radio-wave-based distance estimation

There have been a number of different ranging method proposed, one of the most well know methods is based on the fact that signal strength decreases over distance; this technique is called RSS (Received Signal Strength). RSS a simple and affordable technique, but it suffers from high inaccuracy especially indoors [2].

More accurate methods are time based and are mostly referred to as TOF (Time of Flight) or TOA (Time of Arrival). The basic idea of these methods is the employment of communication to calculate the distance between the two devices. The devices are sending messages back and forth, the distance is than approximated by multiplying the duration of the messaging process by the speed of travel (mostly light speed). The amount of messages necessary can be reduced by synchronising the device clocks, but this process is rather complicated especially with mobile devices.

B. Deep Q-Networks

Our approach relies on Q-learning [19]. In Q-Learning $Q^*(s,a)$ is the optimal value function for calculating the expected reward, for a given state-action pair. In order to scale this algorithm it is common to approximate Q^*. DQNs [12] use a neural network parametrised by θ, to estimate a Q-function, Q_θ, as close as possible to Q^*.

$$Q^*(s,a) = \mathbb{E}[r + \gamma \max_{a'} Q^*(s',a') | s,a] \tag{1}$$

This leads to the following loss function, which the network θ is minimizing by applying gradient decent:

$$L_r(\theta_t) = \mathbb{E}_{s,a,r,s'}\left[(y_t - Q_{\theta_t}(s,a))^2\right] \tag{2}$$

t marks the current time step and y_t is defined as:

$$y_t = r + \gamma \max_{a'} Q_{\theta_t}(s',a') \tag{3}$$

$\gamma \in (0,1]$ is a parameter that discounts future rewards. For further details about DQNs, see [12].

C. Recurrent Neural networks

Recurrent Neural Networks (RNN) have the ability of remembering previous network inputs, unlike normal feed-forward networks and are therefore particularly useful in learning from partially observable sequential data. Typical use cases are natural language and music processing. RNNs have recurrent connections which allow the network to hold information across inputs. These connections can be thought of as similar to memory. There are different types of recurrent networks with different configurations of connections, but the most common ones are Long Short-Term Memory (LSTM) [14] cells and Gated Recurrent Units (GRU). Both have mechanisms to protect their internal memory from losing essential previous inputs, allowing them to handle longer sequences.

D. Particle Filter

Particle filters also known as Sequential Monte Carlo (SMC) methods, are commonly used in robotics for filtering and state estimation problems in partially observable dynamic environments ([4], [9]), such as localisation in known maps. The particle filter is designed to provide a conditional probability of states based on noisy observations. They present a distribution by a set of particles drawn from this distribution. Each particle $x_t^{[m]}$, in a set of size M, represents a possible state at time t. At each time step particles are updated by the control u_t (e.g. velocity) and afterwards weighted by its probability $w_t^{[m]} = p(z_t|x_t^{[m]})$ based on the observation z_t. Subsequently the filter draws M particles from the current set, and the probability of drawing a specific particle is defined by its normalised weight. This increases the probability to converge towards particles representing the actual state. For further detail see [20].

III. Hardware

A. UWB distance chip noise and communication

The chip used in this project is produced by the company DecaWave and has a claimed accuracy of $\pm$10cm. We modelled the chip performance by comparing its measurement with the data from a motion capture system and estimated a normal distributed error with $\sigma^2 = 0.061295$ and $\mu = 0.0085569$. Two way ranging [16] is used for the distance approximation. A problem with this method is, that it requires a few message exchanges to estimate a distance, and depending on how many drones are present this can lead to a cross-talking problem where messages get lost. Currently we are using a token-ring [3] protocol to handle this problem. A token-ring only allows the drone with a token to initialise connections, this reduces the likelihood of cross-talking, but slows down the individual communication-rate with the increasing number of drones. In future work we will address synchronising the drone-clocks in order to reduce the amount of messages required.

B. Drone and Mocap setup

The platform we use are crazyflies produced by the company BitCraze. It is a 27g micro drone, based on a STM32 micro processor. For stable control it requires velocity information currently provided by a motion capture system, which will be replaced by an optical flow-sensor in the future.

C. Neural network hardware interface

From a software architecture perspective the proposed model (Section IV) resides on top of the drones hardware controller passing on velocity commands, which are then executed in a closed loop fashion. The model receives the required distance information from the DecaWave chip and the velocity data from the motion capture system. Possible actions for the model are a fixed acceleration in eight

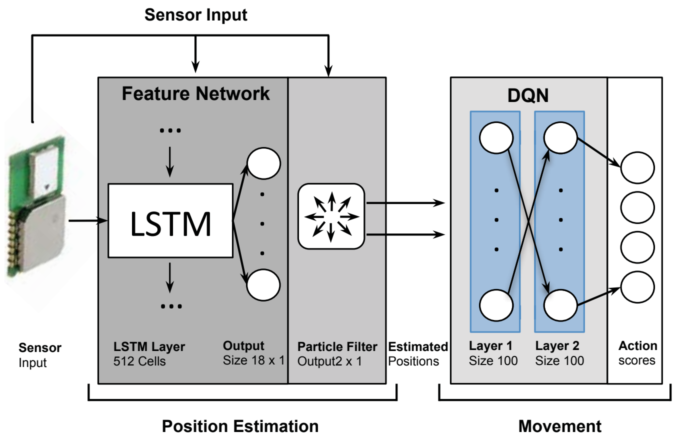


Fig. 1: Architecture and data flow of the employed model.

different direction with a 45° spacing. Additionally it has the option to slow down, this adds up to nine actions in total. The speed of the drone is capped at 1m/s. The model is currently run externally, but will be added to the drones' software stack after the next hardware iteration.

IV. Model

Our proposed model consists of two different subtasks (Figure 1): The first task is the extraction of direction information based on a state sequences and is accomplished by a recurrent neural network (Section IV-A) in combination with a particle filter (Section IV-B). The second subtask is Q-value approximation for the action values depending on the current state, this is realised by a DQN (Section IV-C). Both neural networks can be trained in parallel in a relatively short time. The networks are trained in simulation, described in Section V.

A. Recurrent Network Model

Based on a sequence of input states, the RNNs aim is to estimate the direction to a certain object (goal or drone) and the objects relative orientation. Using the estimated direction α (in radians) and the distance d received from the UWB-chip, we can calculate the relative x and y position of the object, assuming the drones fly at the same height. The estimated direction can either be pointing towards a goal or to another drone. Rather than feeding a combined state of goal and drone distances and returning a direction to both; the state only includes one or the other. This makes its use case more flexible, since it is easier to estimate directions of multiple drones. The estimated orientation is representing the translation between the own frame and the frame of the object. This information is required by the particle filter, in order to translate the objects velocity into the local frame of the current drone. The RNN requires as input the delta time dt since the last update, the distance to an object d_o, its own velocity (v_x, v_y) and the velocity of the object (v_{ox}, v_{oy}). Therefore is the input state defined as $(dt, d_o, v_x, v_y, v_{ox}, v_{oy})$.

Tests in the beginning showed that the inclusion of the velocity information results in a high gain in the networks performance. Since the ranging requires a message transfer, we are able to include this information into the ranging message.

The RNN illustrated in Figure 1 consists of one layer of 512 LSTM-cells. The direction towards an object and its orientation are estimated in degrees, these angles are discretised in steps of 20 degrees, resulting in two vectors with the size of 1×18. Each output is a one hot encoding vector marking the predicted angle-class with a one and all others with a zero. The loss-function of the network is the combined cross-entropy of the predicted vectors and a one hot encoded vector of the actual direction/orientation angle.

B. Particle Filter

The main purpose of the particle filter in the proposed model is to provide a stable and sensible state estimation to the DQN. Even with a perfectly trained RNN it might happen that, due to sensor noise or dropouts, the network estimates two sequential states which are physically impossible. The particle filter is able to prevent this. The employed particles are a representation of the other drones and the goal positions, with their x, y coordinates and their yaw rotation. As mentioned in Section II-D, the filter updates the particles based on the control u_t for every time step t. In the case of our model u_t is the combination of the x and y velocities of the own drone and the drone/goal represented by the particles. The weighting is done based on the noise model of the DecaWave chip, shown in Section III. The filter itself works mostly like a normal particle filter, but instead of initialising random particles in a larger area, it spawns particles with x, y and yaw coordinates similar to the estimation of the RNN. The resampling step has a similar difference, 90% of the samples are drawn from the previous set and 10% are new particles based on the estimation of the RNN. Since the RNN provides a pretty good estimation on its own, we don't require a lot of particles to offer a stable estimation and a good converging speed (see Section VI).

C. DQN

The DQN has two fully connected hidden layers with of size 100 each. The input is a vector of size 1 x 7 and the output is a Q-action value vector with a size of 9. The possible actions are fixed accelerations in eight different directions with a $45°$ step size, the remaining action slows the drone down. The DQN requires relative 2D coordinates to the goal (d_x,d_y) and the obstacle (o_x, o_y), provided by the particle filter, its own velocity (v_x, v_y) and the delta time dt since the last update. Therefore the input state is defined as $(v_x, v_y, d_x, d_y, o_x, o_y, dt)$.

V. Training

This section will detail the training of our model.

A. Simulation

To speed up the training we created a simple 2D python simulator with simple inertia and noise models based on the MAVs and the UWB chips. We decided to keep the simulation in 2D and restrict the drones to fly at the same height. The problem with flying on top of each other is that the drone on top is producing a down wind (known as prop-wash), which most likely causes the drone below to oscillate or crash.

B. Frame skipping (update rate)

Frame skipping [1] is a common technique, used for playing 2D or 3D games with neural networks. The technique uses the fact that some consecutive states, or in their case images, look very similar. This makes it pretty difficult to extract features from a sequence, since all states in this sequence look the same. The solution to the problem is to skip steps in between, however it is important that the action which was applied before the skipping is held until the next frame, otherwise the sequence becomes quite unpredictable. In the hardware setting we are working in, this could also be referred to as update rate. In order to accomplish this, both our networks have to be synchronised. At each update step the DQN is choosing an action and the RNN is recording the current state and predicting the required features, the action is held until the next step and so on.

C. Recurrent Network

The RNN is trained in a supervised fashion, by gathering data over a longer time period and training the network offline.

1) Training data collection and sensor noise: The data is a collection of 3000 episodes. One episode is a 10 minute scenario of two drones flying in a 2m x 2m area. We are collecting data of both drones, this adds up to around 41 days worth of data. The simulated drones use a pre trained DQN policy on perfect position information and to a certain extent random moves. We are recording the direction to the other drone, the baring to a fixed target point and the required input states. Since there is a relation between the goal position and the direction the movement policy is travelling, we are not recording the actual goal position, instead we are choosing another fixed point at random. The data is recorded without any sensor noise, new sensor noise is added to the data everytime a new batch is drawn from the data set; this prevents the network from over fitting.

2) Sequential updates: Similar to the approach presented in [15] we are drawing sequences of observations of length n from a replay buffer. Since the states in the beginning of the sequence don't have a lot of information to rely on, we are only considering states, which have at least a k number of predecessor states. Therefore we only backpropagate errors through the network for states k and higher as illustrated in Figure 2. The number of states used for the update can be quite important and shown in Figure 3.

3) Stability and Training parameters: RMSProp is used for the optimisation with a minibatch size of 30. For stability we divide the data into validation and training sets and check the network performance on the validation set during training. As a metric we apply the accuracy mean of all predicted angles and their target value. We only save the weights when the accuracy improves over a fixed number of validations.

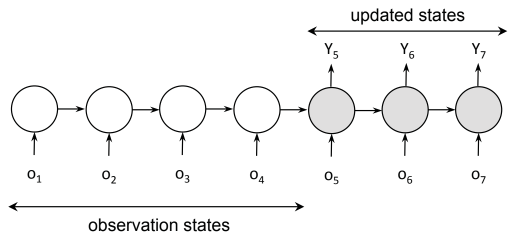


Fig. 2: In this example only errors of $k > 4$ are backpropagated through the network.

4) Performance: The performance during training is measured on a validation set during training. As heuristic we use the percentage of correctly predicted outcomes on a known sequence of states. The training takes around eight hours on our setup and converges towards a accuracy of 95%. The converging speed depends also on number of states considered for the update. Figure 3 shows that it takes the optimiser, which considered the last ten states, a lot less iterations to reach a 90% accuracy, than the optimiser which only considered one.

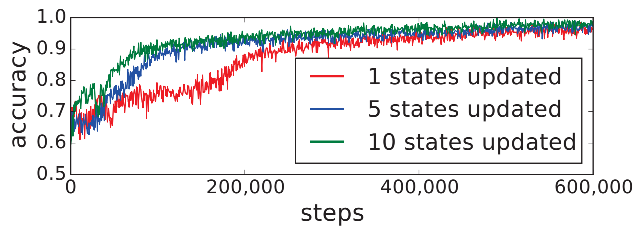


Fig. 3: Converging behaviour

D. DQN

1) Reward shaping: The main goal of the network is to learn a policy to guide the drones towards a target position and avoid other drones on the way. Therefore we define the reward as the negative distance to the goal. If the distance to another agent is under a defined threshold range c, we add a second negative reward, the inverse of this distance. Both rewards are normalised and can be defined as follows:

$$r = \begin{cases} -d1, & \text{if } d2 \geq c \\ -d1 + \frac{d2-c}{c}, & \text{otherwise} \end{cases} \tag{4}$$

With $d1$ as the goal distance, $d2$ the distance to the closest drone and c the critical collision distance.

E. Multi-agent Q-learning

We are training one policy which is applied on all agents. In the current training setting, we simulate two drones, each drone has the same network. The network is constantly updated, changing the policy during training, this means the behaviour of the other drone is constantly altering. This can cause unstable learning or a moving target problem. In general the learning performed well under normal settings. As a comparison we applied Hysteretic Q-Learning [11], which is applied in particularly to fully observable multi-agent settings. It uses two different learning rates, a smaller learning rate β is applied if the current update has a negative influence on the updated Q-Value. We tried two different βs shown in figure 4, the difference is minor and all approaches reach a similar reward average, but the more we ignore the negative reward, the weaker the network performs.

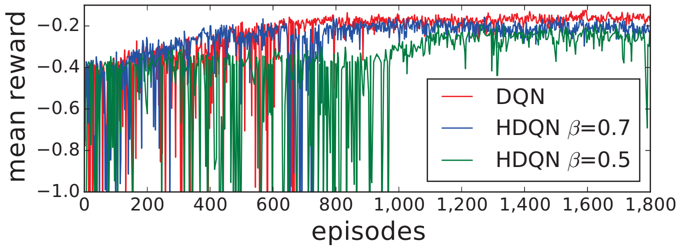


Fig. 4: HDQN training performance

F. *Stability*

Unlike supervised learning settings, in reinforcement learning we can't evaluate the model on a validation set. This makes the validation of the network a little bit more difficult. Since our reward shaping is quite simple and linear we have applied the evaluation metric suggested by [12], where we use the average reward over the number of episodes to validate the networks performance. Weights resulting in a new top score are stored.

G. *Training parameters and performance*

Depending on the machine, the training takes around two hours to converge to a stable state. For the action selection we use ε-greedy. The learner has a probably of ε of choosing a random action and a $1-\varepsilon$ probability of selecting an action based on the networks Q-Value. To explore the reward-function, ε is initialised with 80% and decayed over time to its final value of 10%. The network updates are done on a random batch of size 30.

VI. Real-world experiments

A practical use-case for pocket drones is mostly an indoor environment, since they are light in weight and therefore sensitive to strong winds and gust. Their advantage against ground robots is of course that they are able to fly and are not effected by a bad ground condition or lying obstacles. Furthermore they are able to observe the environment from a top-down perspective, which can be an advantage in certain situations, like surveillance and exploration. Therefore the tests are applied exclusively indoors, in the robotic-lab of the University of Liverpool. The test area measures a size of around 4m $\times$ 4m, the drones itself are a open-hardware platform called crazyflie. The required velocity information is obtained by an optical tracking system on the ceiling, later on we are planning to use an optical flow system on the drones itself in order to provide this information. As described before the is system is divided in two sequential subtasks, both task are evaluated separately. The first part of the model is responsible for the estimation of the direction/position of goals and other drones, this is a accomplished by a recurrent neural network in combination with a particle filter, its evaluation is illustrated in section VI-A. The second sub-system, a DQN, is responsible for choosing the best movement action, based on the input state its got passed from its predecessor. The system is tested in two scenarios, one with fixed goal positions and the other one with random goal positions, each scenario is tested up to ten times with two, three and four drones respectively. Section VI-B describes those scenarios in greater detail.

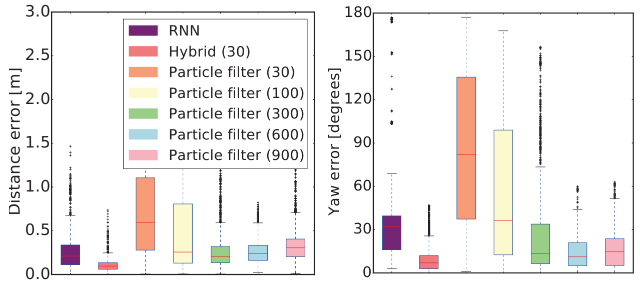


Fig. 5: Average error over all test runs.

A. *Position accuracy*

To test and compare the performance of the position estimation part of the system, we tried different system configurations. We used the RNN without an particle filter, particle filter with different particle configurations and both combined. Sensor data of all flights is collected in order to test different particle filter offline. As described in Section IV the estimation part of the system is using the input data to estimate the direction to a object (goal or drone) and its relative orientation. The estimated direction pointing at a neighbour drone (same height) and the distance information of the UWB-chip are combined to a polar-coordinate to represent the neighbour position in the original drones frame. The distance error between estimated and actual positions is used as heuristic to evaluate the performance, as well as the difference between estimated and actual orientation (yaw-rotation). Both results are illustrated in Figures 5 and 6. The number in the brackets behind the name of the approach, indicates the amount of particles used.

Figure 5 shows the average error over all runs and reveals that the RNN has a similar error profile as the particle filter configurations with higher particle counts. The distance error between estimated obstacle/goal position is around 15cm and the average error of the estimated obstacle orientation is around 30°. The performance of the particle filter only configuration is improved by increasing the particle count, but the tests show that configuration with higher particle count have the tendency to converge to multi possible solutions/particle clouds, which explains the performance drop between the 600 and 900 particle configuration. A lower particle count configuration is faster in terms of computation, but it less likely to converge to a correct solution and if the particles are drifting away from the correct solution it's less likely to recover. The evaluation shows that combination of RNN and particle filter is the best compromise, it doesn't requires a lot of particles to keep the error low (see Figure 5) and it converges quicker to a good solution than the best performing particle filter configurations (see Figure 6). The hybrid approach is also less likely to get stuck in a local

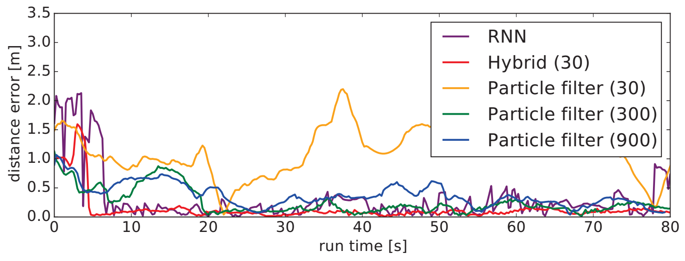
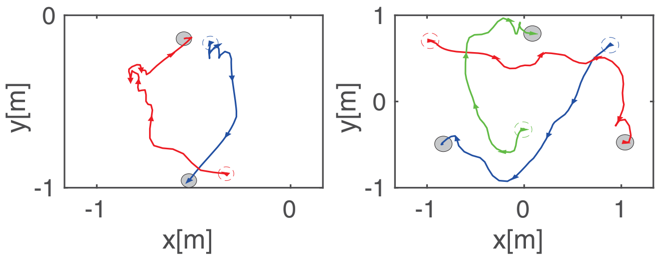
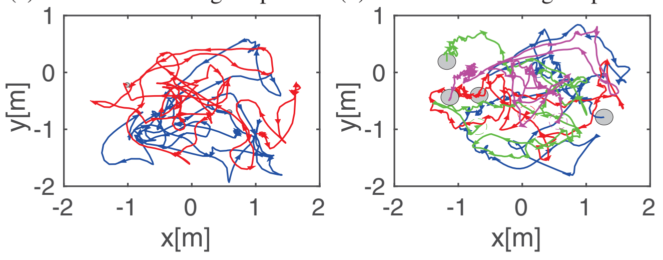


Fig. 6: Performance over a single run



(a) 2 Drones: Fixed goal points. (b) 3 Drones: Fixed goal points.



(c) 2 Drones: Random goal points. (d) 4 Drones: Random goal points.

Fig. 7: Path-recordings of the real-world tests.

minima, since the filter is sampling new particles based on the RNN estimation and these are more likely to be the correct solution than any random sampled particles.

Figure 6 shows the performance of each configuration in an average run. It shows that the pure RNN and the hybrid-configuration are converging quicker, than the pure particle filter configurations, to a good position estimation of the other drone. The estimation error of the hybrid approach is mostly stable, where the RNN has some outliers.

B. *Navigation performance*

Besides estimating the position of other drones and goal positions based on distance and velocity information, we are also providing motion control using a DQN. The network is tested in in two scenarios, one with fixed goal positions and the other one with random goals. The fixed goal setting is designed to avoid head-on collisions and crossing path with other drones, shown in Figures 7a and 7b. The random scenario (Figure 7c and 7d) was designed to check if the drones are able to operate in a relative confined space for a longer period of time without crashing. In total we performed over 30 test flights with no mayor collision and all the goal positions were reached. In terms of smoothness of the travelled trajectories, Figures 7c and 7d show that the DQN is sometimes choosing slightly curvy trajectories, mostly in the beginning of the path, this is due to the discrete action space and the time it takes the particle filter/RNN to converge to the correct goal position.

VII. Conclusion

We introduced a complete system that is capable of navigating multiple autonomous drones in a confined space only based on UWB-distance information and velocity input. We tackled the problem of real world sensor noise, by combining neural networks with a particle filter and show that the combination outperforms the traditional particle filter approach in terms of convergence speed and robustness. While the proposed approach performs well, there are some limitations we need to address and tackle in future work. The UWB-chips require communication between each other to calculate distances, we apply a token ring protocol to avoid cross-talking, but this is slowing down the individual communication-rate with the increasing number of drones. Currently we are only able to scale to five drones. In future work we aim to synchronise the drone-clocks in order to reduce the amount of messages needed. Finally, we plan to employ an optical-flow sensor as we are currently relying on a mocap-system to provide the velocity information.

References

[1] M. Bellemare and et al., "The arcade learning environment: An evaluation platform for general agents," *CoRR*, 2012.

[2] M. Bernas and B. Płaczek, *Energy Aware Object Localization in Wireless Sensor Network Based on Wi-Fi Fingerprinting*. Springer International Publishing, pp. 33–42.

[3] W. Bux, F. Closs, K. Kuemmerle, H. Keller, and H. R. Mueller, *Chapter 2 The token ring*. Springer Berlin Heidelberg, pp. 36–63.

[4] A. D. et al., Ed., *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[5] A. J. et al., "Domain adaptation of recurrent neural networks for natural language understanding," *CoRR*, 2016.

[6] B. R.-S. et al., "Bio-inspired multi-robot systems," in *Biomimetic Technologies*, 2015, pp. 273 – 299.

[7] D. H. et al., "Multi-robot collision avoidance with localization uncertainty," in *Proceedings of AAMAS 2012*, 2012, pp. 147–154.

[8] H. S. et al., "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *CoRR*, 2014.

[9] J. L. et al., "Sequential monte carlo methods for dynamic systems," *Journal of the American Statistical Association*, pp. 1032–1044, 1998.

[10] M. B. et al., "Audio-based Relative Positioning System for Multiple Micro Air Vehicle Systems," in *RSS2013*, 2013.

[11] S. O. et al., "Deep decentralized multi-task multi-agent reinforcement learning under partial observability," *CoRR*, 2017.

[12] V. M. et al., "Playing atari with deep reinforcement learning," 2013, vol. abs/1312.5602.

[13] Y. C. et al., "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," *CoRR*, 2016.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[15] M. Kempka and et al., "Vizdoom: A doom-based AI research platform for visual reinforcement learning," *CoRR*, 2016.

[16] S. Lanzisera and et al., "Radio frequency time-of-flight distance measurement for low-cost wireless sensor localization," vol. 11, pp. 837 – 845, 04 2011.

[17] P. Long, W. Liu, and J. Pan, "Deep-learned collision avoidance policy for distributed multiagent navigation," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 656–663, April 2017.

[18] J. Roberts and et al., "3-d relative positioning sensor for indoor flying robots," *Autonomous Robots*, vol. 33, no. 1-2, pp. 5–20, 2012.

[19] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998.

[20] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.