Reinforcement Learning for Autonomous Headland Turns

Lukas Pindl[1], Riikka Soitinaho[1], Patrick Behr[1] and Timo Oksanen[1,2]
[1]*Chair of Agrimechatronics, Technical University of Munich (TUM), Germany*
[2]*Munich Institute for Robotics and Machine Intelligence (MIRMI), Technical University of Munich (TUM), Germany*
{*lukas.pindl, riikka.soitinaho, timo.oksanen*}*@tum.de*

Keywords: Reinforcement Learning, Autonomous Vehicles, Agricultural Robotics, Headland Turning, Proximal Policy Optimization.

Abstract: This paper explores the use of reinforcement learning (RL) for the autonomous planning and execution of headland turns, aiming to achieve real-time control without the need for preplanning. We introduce a method based on proximal policy optimization (PPO), and incorporate expert knowledge through Dubins paths to enhance the training process. Our approach models the vehicle kinematics and simulates the environment in Matlab/Simulink. Results indicate that reinforcement learning (RL) can effectively handle the complexity of headland turns, offering a promising solution for enhancing the efficiency and productivity of agricultural operations. We show, that this approach can reach the turns goal point reliably in simulation with a positional error of under 20 cm. We also test the policy on a real vehicle, showing that the approach can run in real conditions, although with reduced accuracy. This study serves as a foundation for future research in more complex scenarios and optimization goals.

1 INTRODUCTION

One main puzzle piece in the automatic coverage of agricultural fields is the planning and execution of headland turns. Headland turning involves the navigation of machinery at the ends of crop rows during fieldwork activities such as plowing, planting, or harvesting. The most common maneuvers are shown in Figure 2. The headland, also known as the end rows or turn rows, is a typically small area at the edges of a field where machinery turns to begin a new pass. This practice, while seemingly straightforward, plays a significant role in the efficiency and productivity of farming. Efficient headland turning minimizes time loss, reduces soil compaction, and mitigates crop damage, contributing to overall operational effectiveness. Various factors, including equipment type, field geometry, and operator skill, influence the complexity and execution of headland turns. Deep reinforcement learning (DRL) is a potential option for this application, that could allow for real time planning and control of the vehicle. It could also enable inclusion of multiple, sometimes competing, optimization objectives and should eventually be able to deal with more complex vehicle models, due to it's inherent non-linearities. The goal for this paper is, to explore the general viability of reinforcement learning (RL) for this task. We design the simulation to represent our research tractor (see Figure 1), so that we can test the final agent on it.

Figure 1: The AMX G-Trac is the chair's own tractor. The final agent was able to do a headland turn with this vehicle.

To our knowledge, it is the first example of an RL based controller that can fully plan and execute a headland turn without the need for any preplanning in the field. While the scope is still limited and some simplifying assumptions are made, it is intended to build a basis for future research with more complex

Pindl, L., Soitinaho, R., Behr, P. and Oksanen, T.
Reinforcement Learning for Autonomous Headland Turns.
DOI: 10.5220/0012944700003822

In *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2024) - Volume 1*, pages 318-325
ISBN: 978-989-758-717-7; ISSN: 2184-2809

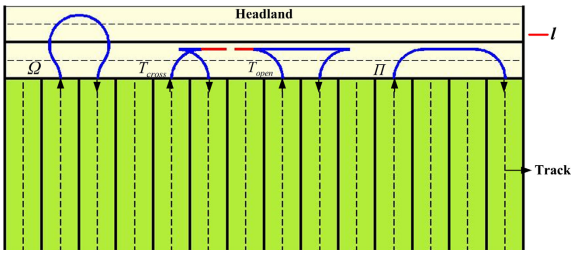


Figure 2: Illustration of the geometrical representation of a field with four common turn types: The forward turn (Ω-Turn), the reverse cross turn (T_{cross} or Fishtail-Turn), the reverse open turn (T_{open}) and the double round corner turn (Π or U-Turn). (Zhou et al., 2015, Figure 5).

models and optimization goals.

2 BACKGROUND

2.1 Related Work

Headland turns can be planned with many different approaches. The problem can be represented through optimal control as in (Oksanen, 2007). Here, a numerical solver is used to find the necessary control inputs and the resulting trajectory. This approach is revisited by (Tu, 2013), where the solution is sped up with some simplifications, as well as newer software and hardware. (Sabelhaus et al., 2013) investigates the use of continuous curvature paths for headland turns. These use a similar approach as the well known Dubins turns (Dubins, 1957), but improve the real world performance by accounting for limited steering rates. (Cariou et al., 2009) and (Yin et al., 2020) plan a turn using geometric approaches and use relatively simple control laws to follow those trajectories. (Gao et al., 2023) starts from scratch by also detecting the available headland area, mainly from camera images.

Much of the research regarding RL for non-holonomic autonomous vehicles covers on-road driving. Off-road problems often have a very different structure, making it necessary to find separate solutions. The agent designed by (Wiberg et al., 2022) uses information about the goal position, several vehicle states like joint angles and wheel slip as well as a local height map to navigate a forestry vehicle through rough terrain. (Sánchez et al., 2023) proposes to learn steering actions end-to-end from lidar based range measurements in uneven terrain with obstacles. Curriculum learning is applied during training with a generic actor-critic algorithm.

While parking often happens on paved areas, the problem structure is still similar, with one goal pose and a comparable vehicle model. (Lazzaroni et al., 2024) simulates parking a car in the CARLA simulator. The agent is trained in simulation to learn the right throttle and steering controls from distance measurements and goal information. Curriculum learning is also applied. (Wu et al., 2023) let human volunteers park in the CARLA simulator to generate expert knowledge, that is used to speed up training.

RL has not yet been extensively tested for either planning or control of vehicles in an agricultural context, but some examples exist. (Makino et al., 1999) employ tabular RL to improve a globally generated path for covering a field. The vehicle model includes an approximation of slip, that also depends on the terrain.

(Olcay et al., 2023) train an agent to cover a field and successfully turn at the end of each lane. They achieved this by predefining a rough corridor, that the vehicle should follow and then giving the DRL agent its distance to the edge of this corridor in multiple directions. A simple tractor-trailer kinematic model is used for this purpose. Multiple learning algorithms are applied and their results compared.

2.2 Headland Representation

Since the headland follows the edge of a field, it can have various shapes. In this paper, to simplify the generation of different situations, we assume the field edge to be a straight line. While this is obviously not true when looking at the entirety of a field, this approximation is valid for most areas, as long as it is only applied locally to a single turn. Such simplifications are typical for headland turning automation (see for example (Oksanen, 2007), (Tu, 2013), and (Yin et al., 2020)). We only investigate left hand turns. For the setup we use, to achieve a right hand turn, all inputs and outputs can simply be mirrored.

Figure 3 shows the geometrical representation that we use in this paper. From any start pose p_{start} the field-headland edge $x_{h,field}(y)$ is easily defined through the headland angle α. The goal pose p_{goal} can be calculated given the working width l_w. Lastly, the headland-outside $x_{h,out}(y)$ edge can be found by shifting the field-headland edge by a fixed headland width l_h. Mathematically, these definitions can be formulated as follows, if the start pose is assumed to be $p_{start} = [0\ 0\ 0]^T$:

$$p_{goal} = [l_w \sin(\alpha)\ l_w\ \pi]^T \tag{1a}$$

$$x_{h,field}(y) = y\sin(\alpha) \tag{1b}$$

$$x_{h,out}(y) = l_h\cos(\alpha) + y\sin(\alpha) \tag{1c}$$

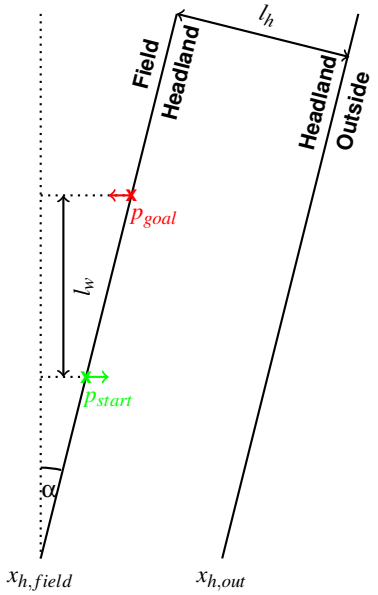


Figure 3: The general setup of the headland turn problem for this paper. Arrows at start and goal point indicate the vehicle orientation.

2.3 Proximal Policy Optimization

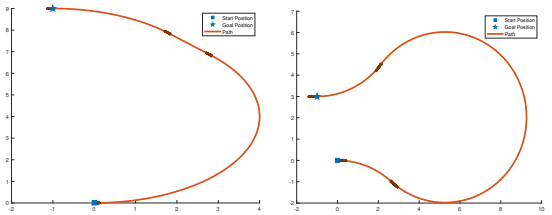
Proximal policy optimization (PPO) is an on-policy actor-critic DRL method that was specifically designed to allow the agent to be trained on samples that were created with policies that were slightly different from the current one. PPO excels due to relatively small computation times during the optimization step, meaning that far more state-action pairs can be included in training in the same amount of time.

PPO was introduced by (Schulman et al., 2017) as a more computationally efficient version of trust region policy optimization (TRPO) (Schulman et al., 2015). PPO clips the change of the policy to a maximum ratio ε. To achieve this, the probability of choosing the action that was taken is stored for each sample as $\pi_{\theta_{old}}(a_t|s_t)$ and the loss for the actor network is then computed as

$$\mathscr{L}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, \text{clip}(k_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)] \tag{2a}$$

$$k_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{2b}$$

with A_t being the samples advantage. Note that the ratio between the new and old policy $k_t(\theta)$ is called $r_t(\theta)$ in the original paper. Since we use r for the reward in this paper, we have changed the nomenclature to avoid any confusion.



(a) U-Turn. (b) Ω-Turn.

Figure 4: Examples for a U-Turn and an Ω-Turn using Dubins connections.

2.4 Dubins Turns

A technique to find optimal trajectories between two points for any vehicle with a fixed minimum turn radius is introduced by (Dubins, 1957). Some simplifying assumptions are made, however. There is an infinite amount of free space in all directions, with no obstacles. Another assumption that is not typically true in real-life applications, is that the vehicle can steer at any angle instantly. While the steering angle is limited, leading to a minimum turn radius, the vehicle needs to switch from a full left turn to a full right turn in an instant to perfectly execute a Dubins turn. Even with such limitations, these can however generate fairly good U-turns and Ω-turns as shown in Figure 4, that can be a starting off point for further learning. Dubins paths can always be split into 3 parts. The first is a curve, either left or right. Curves are always at the minimum turn radius. The second part can either be a straight segment, or a curve in the opposite direction. The third segment is always a curve in the same direction as the first. This yields 6 different path types. For each, the path can be geometrically constructed from the given circles and straight lines. The shortest of the 6 paths is chosen. Because they are relatively simple to generate, we use Dubins turns as expert knowledge (see Section 4.2).

3 REINFORCEMENT LEARNING ENVIRONMENT

3.1 Vehicle Model

The goal in this study is, to model the chairs agricultural research tractor. It is a Lindner Lintrac 130, also called the G-Trac, that can steer the rear wheels independently. We therefore include the four wheel steering capability in the model for later use. However, we fix the rear wheel angle at $\delta_r = 0$ for all experiments in this paper. The vehicle is modeled through its kinematics only. The vehicle state is described by

$$X = [x\ y\ \varphi\ \delta_f\ \delta_r\ v]^T \tag{3}$$

x and y are the vehicle's coordinates, φ its heading, δ_f, δ_r describe the front and rear wheel angles and v is the current velocity. The vehicle's origin is set in the center of the rear axle.

The kinematics are described by a simple differential equation, where l is the vehicle's wheelbase:

$$dX = \begin{bmatrix} v\cos(\varphi) \\ v\sin(\varphi) \\ (v\sin(\delta_f - \delta_r)/(l\cos(\delta_f))) \\ \omega_f \\ \omega_r \\ a \end{bmatrix} \tag{4}$$

The control inputs for acceleration a and steering rates ω_f, ω_r are limited by the given vehicle. For this project, they are simply calculated from the desired velocity and steering inputs, which were set by the controller. To properly represent our G-Trac platform, we used its wheelbase $l = 2.42\,\text{m}$ and maximum steering angle $\delta_{f,max} = 52^\circ$ in our simulation.

The first three components of the vehicle state describe its pose $p_v = [x\ y\ \varphi]$. This pose is based on the position of the middle of the vehicle's rear axle. It is sometimes explicitly referred to as the rear pose. If the position of the front axle is needed, it can be calculated as

$$p_{v,f} = p_v + [l\cos(\varphi)\ l\sin(\varphi)\ 0]^T \tag{5}$$

$$= [x_f\ y_f\ \varphi]^T \tag{6}$$

3.2 Observation + Action Space

The observation space contains all information that is given to the agent. It can therefore have great impact on it's performance. Giving more values is usually better. However, information that doesn't help the agent make decisions, can simply slow down training. It is unfortunately not always immediately obvious, which values are useful and which are not.

Care should also be taken, that all observations are realistically obtainable on the real-world vehicle. Otherwise, the agent can not be transferred to the field.

The basic observation O for all experiments in this paper is

$$O = [x'_{goal}\ y'_{goal}\ \sin(\varphi'_{goal})\ \cos(\varphi'_{goal})\ v\ d_h\ \alpha]^T \tag{7a}$$

$$p'_{goal} = [x'_{goal}\ y'_{goal}\ \varphi'_{goal}], \tag{7b}$$

$$d_h = x_{h,field}(y) - x_f \tag{7c}$$

p'_{goal} is the goal pose, but transformed into the vehicle system. Sine and cosine are used here, to avoid jumps in the angle at $\pm\pi$. The horizontal distance to the field edge is given in d_h. All values are also scaled in the implementation, to keep values relatively small. This helps with keeping gradients reasonable and stabilizes learning.

To keep the setup simple, only the front steering angle is controlled, meaning that the action space consisted of only one variable, δ_f.

3.3 Reward Function

The reward function is comprised of multiple components. The full reward can be described as

$$r_{full} = c_{suc} r_{suc} - c_{fail} r_{fail} - c_\delta r_\delta - c_{time} \tag{8a}$$

$$r_\delta = \max(|a_{steer,f}| - 1.1\delta_{f,max}, 0) \tag{8b}$$

r_{suc} and r_{fail} are 1, when the vehicle is in a success or failure condition, 0 otherwise (see Equation (9) and 10). $a_{steer,f}$ is the desired front wheel angle. Equation (8b) will lead to the steering reward r_δ being big, if the desired angle is significantly over the maximum steering angle of the vehicle. This stops the agent from getting stuck in a learned behavior, where every action constantly exceeds the maximum allowed value. The controller then caps the action to $\delta_{f,max}$, meaning that small, random changes still lead to the same result. Exploration is then hindered, as all tested actions are the same. This part of the reward is only necessary, if the used algorithm does not limit the action output, which is the case for the Matlab implementation of PPO.

A success was counted, when the vehicle pose was close to the goal pose in both position and heading.

$$\sqrt{x'^2_{goal} + y'^2_{goal}} < \varepsilon_{norm} \tag{9a}$$

$$|\varphi'_{goal}| < \varepsilon_\varphi \tag{9b}$$

The accuracy thresholds for the final evaluation of the policy are $\varepsilon_{norm} = 0.2; \varepsilon_\varphi = \frac{\pi}{36}$, but different values are used during most of the training (see Section 4.3). The failure condition is reached, if the front axle passes the headland-outside edge, or if the rear axle passes from the headland into the field by more than 1 m:

$$d_h < 0 \tag{10a}$$

$$x < x_{h,field}(y) - 1 \tag{10b}$$

Typical values for the scaling factors in training are $c_{suc} = 1, c_{fail} = 1, c_\delta = 0.002, c_{time} = 0.001$.

4 IMPLEMENTATION

4.1 Matlab/Simulink Setup

The vehicle model described in Section 3.1 is implemented in Simulink. The differential equation Equation (4) is solved in discrete time steps of $\Delta t = 100\,\text{ms}$ through a simple integrator. The control block receives the current state from this simulation and sends the next steering angle as a control input. Inside the control block, the vehicle state is used in conjunction with the current scenario parameters to compute the agent's observations. The Simulink *RL Agent* block is used to easily learn and execute the policy. A visualization is also provided, which can be turned off during training.

The training setup is done in Matlab scripts. This includes the reset function for the environment. At the start of each episode, the agent starts at the origin and the headland angle is chosen at random between $\pm 30°$.

4.2 Inclusion of Expert Knowledge

Using purely random actions, the agent is extremely unlikely to reach the goal without exiting the headland area. With so little positive reward, the agent will probably not learn a successful policy. To avoid this problem, agents can be pretrained from expert knowledge, that is at least good enough to get close to the goal, some of the time. For this purpose, Dubins turns, as introduced in Section 2.4, are used in this paper. At the start of each episode, the inbuilt Matlab function for generating Dubins turns is employed. This function outputs the distances, for which the vehicle has to drive at a full left/right turn. This information is used to directly control the vehicle in simulation, by continuously integrating the distance traveled, and changing the steering angle accordingly.

Since only tight, left-hand turns were tested for this paper, all turns are of the type *right-left-right*. Since the vehicle model uses limited steering rates, strictly following these turns leads to a significant mismatch. By simply adjusting the distance for each of the 3 stages of the turn by a constant length, a much better result can be achieved. For the specific setup in this paper, the three stages are changed by -0.5m, -0.8m and +0.1m respectively. While this is by far not sufficient for reaching the goal very accurately and consistently (see Table 2), it is good enough to give some positive examples to the agent, allowing it to learn from there.

The decision whether the agent will follow its current RL policy or this expert knowledge, is made randomly at the beginning of each episode. The chance of using expert knowledge starts at almost 100%, and is lowered linearly with the number of episodes all the way to 0% shortly before the end of training.

Table 1: Hyperparameters used for the most successful training with PPO. Bold Parameters were varied in the random search.

Parameter	Value
Learning Rate Actor (Part 1)	5.448×10^{-5}
Learning Rate Actor (Part 2)	9.712×10^{-6}
Learning Rate Critic (Part 1)	3.202×10^{-5}
Learning Rate Critic (Part 2)	1.030×10^{-5}
Action Log Prob (Part 2)	-2.677
Experience Horizon	4096
Minibatch Size	256
Clip Factor	0.1
Discount Factor	0.995
EntropyLossWeight	0.001

4.3 Training and Hyperparameters

The Matlab RL-Pipeline makes it hard to dynamically change certain hyperparameters during training. To allow for fast learning in the beginning, but accurate adjustments in the end, the training is split into two parts, where the learning rates are lowered for the second one. In the secondary training, the log probability of the action's normal distribution is also set to a fixed value. The first training is for 17000 episodes, the second for 5000. The second training is started from the saved model that achieved the best success rate at 20 cm accuracy during the first training. After manual tuning to find a good baseline, a random search over the most impactful hyperparameters is performed. The agent with the best performance was trained using the values listed in Table 1. Parameters that are not mentioned use the default values in the Matlab PPO implementation.

Even with the initial use of pregenerated Dubins turns the agent would fail most attempts at the start of the training. A lack of positive reinforcement can make it harder for the algorithm to learn the desired behavior. To increase the likelihood of success, the values for $\varepsilon_{norm}, \varepsilon_\varphi$ as introduced in Section 3.3 are initially set to fairly high values (typically 1.5 m and $\frac{\pi}{3}$). As the training progresses the difficulty is increased by lowering these thresholds. In the best case, this would be done dynamically, whenever the agent reaches sufficient success rates on the current difficulty. Due to limitations with the learning setup in Matlab, this was not easily possible. Instead, the thresholds where lowered linearly over the course of the training and reached their minimum shortly before the episode limit of the training was reached. Changing the difficulty throughout the process brings some

problems with it. The learning rate must be very well tailored to the rate at which the thresholds are adjusted. Otherwise the agent either overfits to the current difficulty, or the agent can not keep up with the increase in difficulty. In the end, a good balance was found that allowed the agent to slowly adjust to the new thresholds.

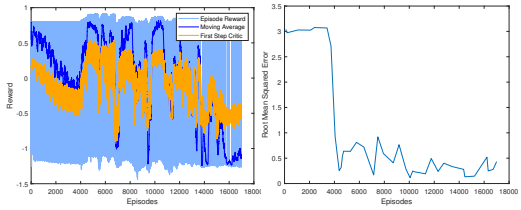
4.4 Real World Vehicle Integration

For testing the final policy in the real world, we use the G-Trac that the chair has modified with multiple sensors and controllers. The position of the vehicle is gained from an RTK-GNSS receiver. From the position of the antenna as well as current pitch and roll angles from an IMU sensor, the position of the rear axle near the ground can be calculated. An interface on the G-Trac allows controlling the speed and steering angles. The data is sent via CAN-Bus to the G-Trac at a rate of 10 Hz, which then executes the commanded actions. The G-Trac is shown in Figure 1. The policy was tested on a sloped grass field as well as a flat, paved yard.

5 RESULTS

5.1 Training Progression

The results shown in this section are for the most successful agent found during the hyperparameter search, as discussed in Section 4.3. The agent learned to complete a 3 m turn ($l_w = 3$). Figure 5a indicates, how the cumulative reward per episode developed. The dark blue line shows the average reward over the previous 100 episodes. The yellow line tracks the critics prediction in the first step of the episode, which includes the discount factor γ. The early decline of the reward is expected, since the difficulty is increasing. The ratio of precomputed Dubins turns to RL based episodes is decreasing at the same time. At around 5000 episodes the steep improvement indicates that the agent has managed to learn how to complete turns for most of the tested headland angles at this point. The frequent drops are most likely due to overfitting to the current difficulty, meaning that the learning rate may have been just slightly too high. The agent was mostly able to recover however. The final accuracy required to be considered successful was set at 10 cm. Since actions are drawn semi-randomly during training, this threshold could rarely be reached, leading to the final episodes getting fairly low rewards. During training the agent is saved regularly, when performance is especially good. This consists



(a) Reward. (b) RMSE.

Figure 5: Indicators for training progression with the basic setup.

of checking if a new global or local maximum was reached in the average reward. To avoid saving too many models, which would have significant storage and time costs, some hysteresis is added. Then the performance of each saved model is tested: The agent is set to always choose the mean of the action distribution. With this deterministic agent 100 random episodes are completed. For each episode the error is computed as the smallest distance that the vehicle ever had to the target. In our test we let the agent drive past the goal to find this minimal distance, meaning that the last recorded position was further away from the goal than the best one. In a real application, it would more likely be stopped as soon as the distance starts increasing again. The way we measure the root mean squared error (RMSE) reflects the second approach. Figure 5b shows how the resulting RMSE progressed over training. While it clearly had a major improvement around 4000 episodes, the value still varies strongly afterwards, indicating that the training was not especially stable.

5.2 Generated Trajectories

As expected, the final results are similar to the Dubin's turns, which we use as expert knowledge. To successfully finish a 3 m turn, the vehicle needs to steer right first, then do the left turn, and finally steer back to straighten out the tractor. This can be seen in the trajectories in Figure 6, as well as in the generated actions in Figure 7b. Since we are using the deterministic policy for evaluation, we can also see, that the actions are fairly smooth. There is some minor overcorrection in the desired steering angle at around steps 30-40. It is likely that a trajectory without this correction would be faster, but the agent simply is not able to learn this. It is a good example of a situation where the blackbox type approach of artificial neural networks (ANNs) makes it very hard to find the exact reasoning for the selected action. All generated turns reached within 20 cm of the target point. Table 2 shows the metrics we used to evaluate how well the final agent did on different difficulties. Comparing the

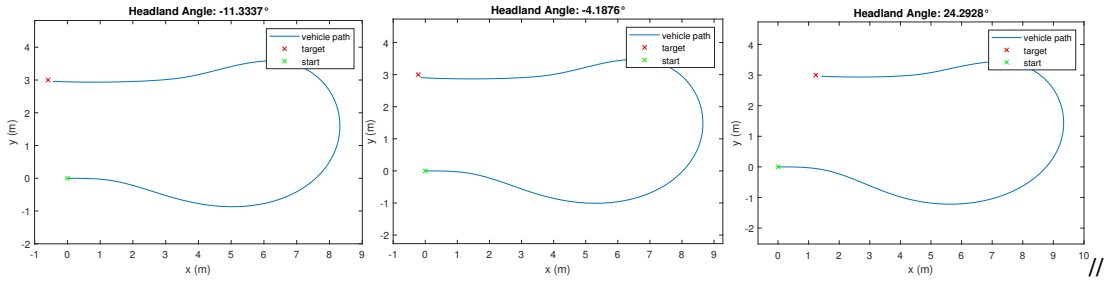
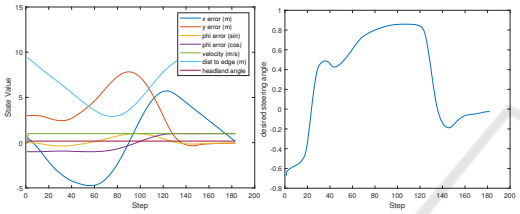


Figure 6: Paths generated for multiple headland angles with the basic setup.

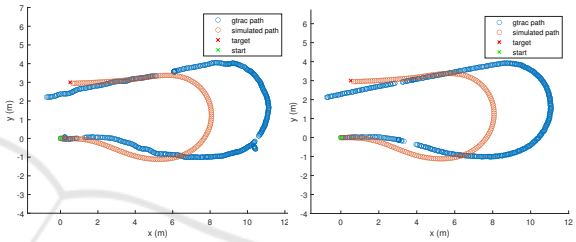
Table 2: Success Metrics, tested on random headland angles. TTS is time to success. Results over 1000 episodes.

Strategy	Success Rate (%)			RMSE (m)	Heading RMSE (°)	TTS (s)
	0.1m	0.2m	0.5m			0.2m
Dubins turn	27.0	32.8	63.7	0.873	3.71	22.16
PPO agent	32.4	100	100	0.133	1.83	20.35



(a) Observations. (b) Generated Action.

Figure 7: Inputs and outputs to the RL agent with the basic setup.



(a) Field. (b) Yard.

Figure 8: Paths generated in the real world using the G-Trac.

metrics to the untrained Dubins turns, that are used as expert knowledge to initiate the training, it is clear that the trained agent performs significantly better in all categories.

During training, the heading is an important part of deciding whether or not an episode was successful or not (see Equation (9)). This means, that the agents learns to line up with the intended final heading. For the success rates shown in Table 2, this angle is not checked. This is mainly to keep the difficulty categories simpler, since it is hard to decide which heading threshold to combine with which distance threshold. The heading error shown in the tables is the angle that was reached for the time step that was closest to the goal in distance. It is possible, that there is a time step in the turn, where the sum of positional and angular error is smaller, but this point would depend on the weight that we would apply to either error. We instead opt to just use the point with the lowest positional error. The time to success was also measured to this time step. Since a constant velocity is used, this value directly corresponds to the distance traveled.

5.3 Real World Vehicle

A total of 7 turns with the RL agent were performed on the G-Trac on different surfaces. One of the turns had to be excluded from the results. It automatically stopped in the middle of the turn without any of the normal failure conditions being met. This was most likely due to a configuration error, that had nothing to do with the agent itself.

Figure 8 shows 2 of the recorded turns. Since all turns have very similar performance, the ones in the figure are simply chosen for how complete the recorded position data is. The ones that are presented have the least amount of GNSS gaps, making them more useful for visual inspection. In the performance metrics, no major differences between the different surfaces can be seen. All of them are therefore combined in the results: Only one turn reaches the goal within 50 cm, but all reach within 60 cm. The RMSE is 0.532 m. The average time to reach the closest point to the goal pose is 42.5 s. The final heading error is relatively high with an average error of 12.6°.

Figure 8 also shows the agent's path in the simulation for the same headland angle. It is obvious that the tractor behaves differently in real life, leading to a longer turn. This results in observations that were never seen during training. The agent still manages to turn with these unexpected observations, but fails to correct in the final stages. The fact that this small difference stops the agent from finishing correctly in-

dicates that it has overfitted to the simulated environment.

6 DISCUSSION AND CONCLUSION

The results show that an RL based agent is capable of planning a headland turn and controlling the vehicle to follow it. The final policy can easily be executed in under 100ms, meaning that the approach is viable for running in real time. However, the stability of the training is still somewhat lacking. While we are able to find a great policy even with the instability, improving on this would be desirable to speed up training and allow for more difficult problems.

Since this paper is intended as a proof-of-concept, no in-depth comparison to other headland turning algorithms is made. This should be an important component for future research into the area.

It is also clear that this implementation does not use many of the theoretical advantages of RL. Research that focuses on using more complex vehicle models and optimization goals could greatly benefit from those strengths.

For the real world application, more techniques to improve the sim-to-real transfer could be applied. The results could likely be improved significantly, if the policy was forced to achieve better generalization.

ACKNOWLEDGMENTS

We would like to thank Michael Maier and Ruben Hefele for their help with implementing and testing the policy on the real tractor. We are also very greatful for Henri Hornburg's input on possible algorithms for the expert knowledge generation.

REFERENCES

Cariou, C., Lenain, R., Thuilot, B., and Martinet, P. (10.10.2009 - 15.10.2009). Motion planner and lateral-longitudinal controllers for autonomous maneuvers of a farm vehicle in headland. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5782–5787. IEEE.

Dubins, L. E. (1957). On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics*, 79(3):497.

Gao, G., Guo, H., Zhang, J., Zhang, Z., Wu, T., Lu, H., Qiu, Z., Chen, H., and Lingxuan, Z. (2023). An efficient headland-turning navigation system for a saffflower picking robot. *Journal of Agricultural Engineering*, 54(3).

Lazzaroni, L., Pighetti, A., Bellotti, F., Capello, A., Cossu, M., and Berta, R. (2024). Automated Parking in CARLA: A Deep Reinforcement Learning-Based Approach. In Bellotti, F., Grammatikakis, M. D., Mansour, A., Ruo Roch, M., Seepold, R., Solanas, A., and Berta, R., editors, *Applications in Electronics Pervading Industry, Environment and Society*, volume 1110 of *Lecture Notes in Electrical Engineering*, pages 352–357. Springer Nature Switzerland.

Makino, T., Yokoi, H., and Kakazu, Y. (28-30 July 1999). Development of a motion planning system for an agricultural mobile robot. In *SICE '99. Proceedings of the 38th SICE Annual Conference. International Session Papers (IEEE Cat. No.99TH8456)*, pages 959–962. Soc. Instrum. & Control Eng.

Oksanen, T. (2007). *Path planning algorithms for agricultural field machines*, volume no. 31 0031 of *Helsinki University of Technology Automation Technology Laboratory Series A Research reports*. Helsinki University of Technology, Espoo.

Olcay, E., Rui, X., and Wang, R. (26.08.2023 - 30.08.2023). Headland Turn Automation Concept for Tractor-Trailer System with Deep Reinforcement Learning. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–7. IEEE.

Sabelhaus, D., Röben, F., Meyer zu Helligen, L. P., and Schulze Lammers, P. (2013). Using continuous-curvature paths to generate feasible headland turn manoeuvres. *Biosystems Engineering*, 116(4):399–409.

Sánchez, M., Morales, J., and Martínez, J. L. (2023). Reinforcement and Curriculum Learning for Off-Road Navigation of an UGV with a 3D LiDAR. *Sensors (Basel, Switzerland)*, 23(6).

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust Region Policy Optimization.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms.

Tu, X. (2013). *Robust navigation control and headland turning optimization of agricultural vehicles*. PhD thesis, Iowa State University.

Wiberg, V., Wallin, E., Nordfjell, T., and Servin, M. (2022). Control of Rough Terrain Vehicles Using Deep Reinforcement Learning. *IEEE Robotics and Automation Letters*, 7(1):390–397.

Wu, Y., Wang, L., Lu, X., Wu, Y., and Zhang, H. (27.10.2023 - 29.10.2023). Reinforcement Learning-based Autonomous Parking with Expert Demonstrations. In *2023 7th CAA International Conference on Vehicular Control and Intelligence (CVCI)*, pages 1–6. IEEE.

Yin, X., Wang, Y., Chen, Y., Jin, C., and Du, J. (2020). Development of autonomous navigation controller for agricultural vehicles. *International Journal of Agricultural and Biological Engineering*, 13(4):70–76.

Zhou, K., Jensen, A. L., Bochtis, D. D., and Sørensen, C. G. (2015). Quantifying the benefits of alternative field-work patterns in a potato cultivation system. *Computers and Electronics in Agriculture*, 119:228–240.