Wits FC Team Description Paper (RoboCup 2023)

Branden Ingram, Michael Beukman, Andries Bingani, Marcell Nagy,
Benjamin Rosman, Pravesh Ranchod
School of Computer Science and Applied Mathematics
University of the Witwatersrand
South Africa

February 22, 2023

Abstract. This document provides an outline of the Wits FC team, as an entry into the RoboCup 2023 3D Soccer Simulation League. We describe how we have built off the base code made freely available by UT Austin Villa [MacAlpine and Stone]. This paper describes the two main contributions that distinguish us from our previously presented work, namely a low-level parameter optimisation approach using Reinforcement Learning and a kicking optimisation for FatProxy.

1 Introduction

Wits FC is a team which forms part of the RAIL Lab[1] which looks to compete and develop solutions for RoboCup. The RAIL Lab is a research group which is situated at the University of Witwatersrand (Wits) [wit] in South Africa. We started competing at RoboCup in 2021 and have continued to develop and improve over the years from winning the "Best New Team" award, to scoring our first goal and now to winning our first game. In 2022, we competed for the first time in person and through this experience, we were better able to identify our weaknesses and formulate a plan to overcome them. Moreover, our success in the FatProxy contest has demonstrated to us that we are not significantly worse than the leading teams in terms of high-level strategic proficiency. However, the weakness in our team was the lack of competitive low-level skills in the main league, essential for us to succeed. The simulated Nao robot is a difficult platform to learn optimal low-level control skills such as kicking and walking due to it being bi-pedal with a large state and action space. As a result much of our energy over the last year has been towards learning/designing a longer distance kick. In recent years, Reinforcement Learning (RL) has been applied successfully in a number of robotics domains specifically for control problems [Haarnoja et al., 2018, Berseth et al., 2018, Tan et al., 2018]. Given our lab's deep expertise in reinforcement learning, we are well-positioned to tackle this problem with a tailored approach that draws on our past successes in this area.

In this paper, we describe our efforts of utilising RL in order to learn a kicking policy in Section 3. Finally, in an attempt to maintain competitiveness in the FatProxy competition, we outline in Section 4 an optimisation approach for kick parameter selection.

2 Overview

Initially, we built on the UT Austin Villa Base Code Release as a template for our codebase, to focus our efforts on components of the systems better aligned to our research expertise. This code release provides a fully functioning agent, with a number of basic skills which we have utilised. Most importantly, it provided a parameterised walking and kicking policy. In this section, we provide an overview of our previous years'

[1] `https://www.raillab.org/`

attempts at optimising the parameters which define the basic kicking and walking skills. In particular, as MacAlpine et al. [2012] describes, there are many different parameters, and not all of them affect all parts of the robot. We thus only considered a subset of parameters for each task and change only these while keeping the rest of the parameters fixed.

We considered two main optimisation algorithms, namely simulated annealing (SA) and genetic algorithms (GA).

- Genetic Algorithms are a class of search algorithms which are loosely based on the mechanics of natural selection [Goldberg, 1989].
- Simulated annealing (SA) is a heuristic optimisation algorithm that is loosely based on the physical process of annealing, in which metals are heated up and cooled slowly, to remove internal stresses and make the metal stronger [Rayward-Smith et al., 1996].

For walking, the goal was to improve the agents' strides to move around the field faster. As a proof of concept, we started with a task where the agent must walk in a straight line, and its fitness/performance was determined by how far it moves in 10 seconds. This did lead to improved results in this one specific case (up to a 10-20% improvement in straight walking speed from the default parameters, however, with a larger standard variation due to over-fitting, see Table 1). In light of this, we created a more detailed task that involved the robot walking in a circuit by following some preset waypoints placed around the field, which is a more realistic setting.

Table 1: Results for the simple walk task. Here we show the straight-line distance that the agent walked in 20s. Results are averaged over 100 runs. These results are visually illustrated in Figure 1a

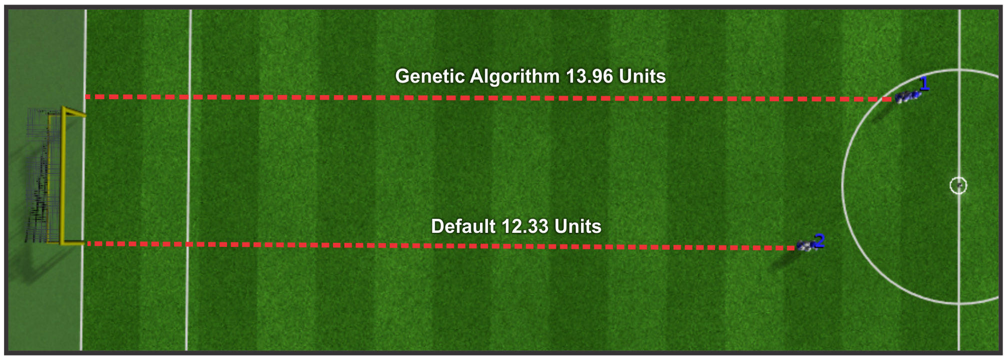
Parameters	**Mean**	**Standard Deviation**
Base	12.33	0.02
Genetic Algorithm	**13.96**	2.97

For the kicking, we initially let the robot just kick a ball that was placed next to it, and the fitness was the distance travelled by the ball. Here we observed impressive results and stronger kicks (up to a $100 - 150\%$ improvement in the kick distance over the default parameters, see Table 2), although they were again not easily transferable to real play, this time due to the agent only optimising when stationary, leading to it struggling to kick the ball when moving. We remedied this problem by again creating a more complex scenario that potentially involves multiple kicks, while also moving towards the ball.

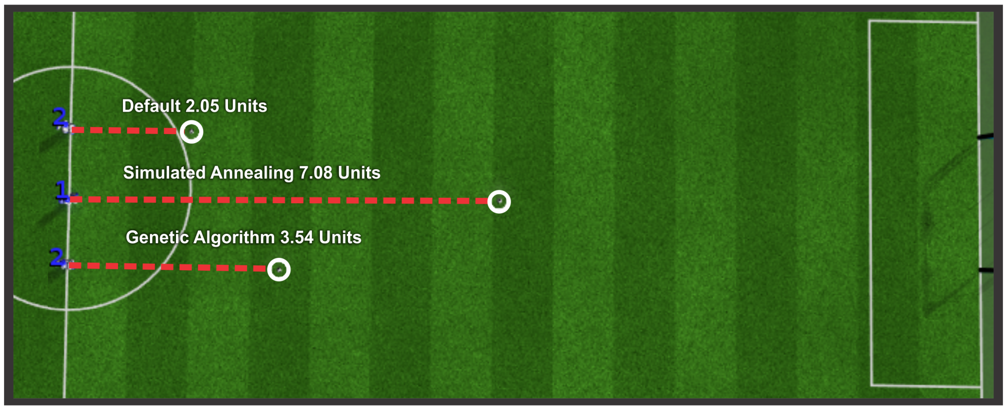
Table 2: Results for the simple kick task. Here we show the straight-line distance that the agent kicked the ball. Results are averaged over 100 runs. Figure 1b shows a visual example of these kick results.

Parameters	**Mean**	**Standard Deviation**
Base	2.05	0.03
Genetic Algorithm	3.54	0.38
Simulated Annealing	**7.08**	0.14

Overall we found these approaches produced better performing parameters however, naïvely implementing this method does lead to some problems. Since we are simulating a very complex system, there is bound to be some noise, such as the robot starting at slightly different locations in different runs, the same kick performing differently, etc. To overcome this, we ran multiple simulations starting from the same state, with the performance now being the average over all runs. In the case an agent catastrophically failed, we



(a) Results on the simple walking task. Optimising the walk parameters led to a slight improvement in speed when travelling in a straight line.



(b) Results for the simple kick. The genetic algorithm improved the kick distance slightly, and the simulated annealing approach nearly doubled that.

Figure 1: Optimisation results for the (a) walk and (b) kick tasks.

assigned it a large negative fitness value. This means that parameters that consistently perform decently will be rated higher than those that sometimes perform well and sometimes fail completely. One additional technique we found useful was to penalise high variance when running the same parameters multiple times, to improve the consistency of the parameters in-game. Empirically we found that these techniques resulted in improved parameters that were useful and robust in actual games. In particular, using simulated annealing with an objective of mean($distance$) $-$ standard deviation($distance$), followed by genetic algorithms once this stagnated resulted in a high-performing kick of around 5 units. However, overall we were not satisfied by these improvements as they were still lacking when compared with those of the top performing teams. For this reason we deemed it necessary to try alternative approaches to optimisation as well as learning skills from scratch.

3 Reinforcement Learning Kick Optimisation

Our next attempt at obtaining improved low-level skills involved reinforcement learning [Sutton and Barto, 2018, RL]. The aim here was to learn an agent policy π that corresponds to specific low-level skills, such as walking and kicking. We particularly focused on kicking as the initial task, as that is one major weak point of our current team.

3.1 Methodology

To start our investigation into reinforcement learning, we initially used the Robocup RL environment from Bahia-RT.[2] This system used a proxy that connects to both the agent and server and routes their commands appropriately, allowing a Python-based RL agent to send messages to the actual agent code, which can then be used to make decisions.

We built upon this, significantly improved the speed of the framework and allowed agents to directly control the low-level joints of the robot instead of sending messages to the base code. Our environment receives actions from the agent, transforms them into server messages and sends these to the server. The environment also listens for server updates, parses these into observations and sends them to the agent. We designed a general continuous-control framework that allows us to choose the exact state and action spaces of the agent (i.e. which sensors are used, and which actuators the agent can affect), and specify a reward function we wish to maximise. Furthermore, since some of our reward functions required information that is not normally given to the agent, we also listened on the visualiser socket, obtaining and parsing visualisation information relevant for these calculations, such as the robot's foot positions. Finally, since we needed to reset the agent at each episode to a known starting point and configuration, we used proportional control to set the robot's joints to their default positions, and then moved the robot to the starting location. We chose to use this approach as disconnecting and reconnecting the agent from the server (which also served as a reset) proved to be significantly slower, and resulted in the server crashing frequently.

The interface of our environment adheres to the standard API in RL, allowing us to easily use off-the-shelf libraries. In particular, we use Stable Baselines 3, as it provides high-performing implementations of many RL algorithms in a simple interface.[3]

Each kick instance is modelled as a single episode in RL, with each episode consisting of multiple timesteps. At each timestep, the robot receives an observation, performs an action and receives a reward. The aim of the RL algorithm, then, is to choose actions such that the discounted sum of future rewards is maximised.

Using this framework, we experimented with many different setups, aiming to obtain a high-performing kick to replace our current one. Each environment had the same base observations, actions and reward, and we detail the experiments we performed in the next sections. Overall, much of our experimentation is based on prior research into using RL in the context of RoboCup [Abreu et al., 2019, Teixeira et al., 2020, Spitznagel et al., 2021]. We also choose to use Proximal Policy Optimisation [Schulman et al., 2017, PPO] as our RL algorithm, due to its general high-performance.

The base environment had the following properties:

Base Observations The observations the agent has access to include all of its joint angles, as well as the ball position, the information from the gyroscope and accelerometer as well as the resistance on the robot's feet.

Base Actions The agent's actions corresponded to setting a desired speed (a continuous value) for each joint, resulting in a 22-dimensional continuous action space.

Base Rewards We experimented with numerous different reward settings, but the main commonality was that the agent was rewarded for moving the ball a large distance. In particular, the reward at each step was the overall distance the ball has travelled. This outperformed using only the difference in positions between subsequent steps, possibly due to some server noise.

3.2 Experiments

In trying to obtain a high-performing kick, we explored numerous different experimental setups, consisting of using different observations, actions and reward functions. For the following results, we evaluated our trained policies by kicking 100 times, and we report the mean and standard deviation of the distances we achieved.

[2]https://bitbucket.org/bahiart3d/bahiart-gym

[3]https://github.com/DLR-RM/stable-baselines3

3.2.1 Initial Attempt

Our initial attempt consisted of using the base observations, actions and rewards as above. We also penalised the agent, and stopped the episode, if the agent fell over. This, however, proved to be problematic, as the agent ended up standing completely still, to not receive the large negative reward associated with falling. For the rest of the experiments, we stopped giving the agent a negative reward for falling, instead just ending the episode once it does so. This still penalises falling (as the agent cannot kick the ball as far), but incentivises a poor kick over no kick at all.

Eventually, this approach led to an agent that managed to kick the ball; however, the ball moved only a short distance (less than 3 units). This behaviour persisted even when training for a long time. This inspired us to explore additional avenues.

3.2.2 Curriculum

We next followed the approach of the Magma Offenburg team [Bohlinger et al., 2022] in using a curriculum, training the agent to first perform a back swing, and using this agent to learn the kick behaviour. The idea behind this approach is that training an agent directly on maximising the ball distance will lead to it not learning to perform a back swing (which will lead to longer distances in the long term), instead favouring a short term gain by weakly kicking the ball. Our approach had two phases. First, we incentivised a back swing and follow through, using the following reward function:

$$R_t^{swing} = \begin{cases} d_t - d_{t-1} & \text{if } t < \frac{T}{2} \\ -(d_t - d_{t-1}) & \text{otherwise} \end{cases} \tag{1}$$

with t being the current timestep, T being the total number of timesteps in the episode and d_t being the distance between the robot's right foot and the ball at time t. In essence, this reward incentivises the agent to bring its foot back in the first half of the episode and then follow through to the ball in the second half. When using this reward, we also added the current timestep to the agent's observations, to be able to distinguish between the first half and the second half of the episode.

Training with this reward function resulted in an agent that can perform the back swing and follow through very well. When training with different values of T, the speed of the resulting behaviour changed accordingly.

We then used this agent as the starting point for phase two. Here we wished to maximise distance. However, in many cases, if we just used the dense distance reward described above (i.e. simply rewarding the agent for the ball's distance from its starting location), the agent quickly reverted to the greedy behaviour, and did not perform the back swing anymore.

Our final reward function for phase two was therefore

$$R_t = \begin{cases} R_t^{swing} & \text{if } t \leq T \\ D_t^b & \text{otherwise} \end{cases} \tag{2}$$

with D_t^b representing the x-distance the ball is from its starting point. For this case, we extended the length of the episode to include both the swing as well as the final kick behaviour. The idea behind this reward structure is that the agent is still incentivised to perform a back swing and use it to kick the ball with significant force. In some cases, only having the distance reward after training on phase one resulted in a much further kick than training from scratch on this setting, even though neither approach still performed the back swing.

Overall, after experimenting with different setups, the best curriculum we used was as follows, using three phases:

Initial This simply uses the initial, back swing reward, with 20 timesteps in total, the first 10 rewarding swinging the foot back and the next 10 rewarding the opposite behaviour. This was trained for around 12M total timesteps.

Middle This rewarded only the ball distance, with more steps than before, at 30, trained for 70M steps.

End This phase reduced the number of steps to 20 again, and rewarded the maximum distance, as well as the maximum height obtained during the kick. This was trained for 2M steps.

The initial phase resulted in a good, fast and consistent backswing, but the ball only moved around 1 unit, with a large standard deviation. After the second phase, the robot kicked the ball an average of 4.2 units, with a standard deviation of 2.3. Finally, after the final phase, the kick was significantly improved, with a mean distance of 6.4 units, while being more consistent, with a standard deviation of only 0.75. This kick reaches a further distance than any of our previous kicks, including our best simulated annealing and genetic algorithm-optimised kick (which had a distance of around 5.2), see Table 3 for a comparison of all of these results. However, since there is no explicit penalty for falling (besides the episode ending), the robot falls down immediately after performing the kick.

Table 3: Results for the simple kick task. Here we show the straight-line distance that the agent kicked the ball. Results are averaged over 100 runs. While the best RL agent had more variance than the best parameter file, it also kicked the ball further. Here we do not include the simulated annealing-based approach that was infeasible to use during a game.

Method	**Mean**	**Standard Deviation**
Base	2.05	0.03
Best GA & SA	5.2	0.2
RL (Second Phase)	4.2	2.3
RL (Third Phase)	**6.4**	0.75

Interestingly, while the agent does not perform the back swing behaviour anymore, it still manages to kick the ball relatively far. Incentivising the back swing at the same time as the distance did not seem to perform better, but this may be caused by poor reward balancing, and is an avenue we wish to explore more deeply.

For all of these phases, the action space consisted of the desired angle and an associated maximum speed for each joint instead of the joint velocities directly. This was then converted into a joint velocity (as required by the server) using a simple proportional formula of $\text{clip}(a_{desired} - a_{current}, -\text{speed}_{max}, \text{speed}_{max})$, with $a_{desired}$ and $a_{current}$ being the desired and current angle for each joint, respectively.

3.2.3 Lift

As an additional goal, we desired to obtain a kick that propels the ball high up in the air. This was for two main reasons; firstly, as a reward-shaping tool to obtain further travelling kicks, and secondly, to obtain a kick behaviour that would result in fewer interceptions from opponents. The curriculum we used for this had two phases. The first one was the same as the above setting, but it trained for nearly twice the time, around 2.8M steps. The second phase, trained for 180M steps, incentivised only kicking the ball high up in the air.

The overall results here were that the ball had a maximum height of 0.42 ± 0.04 units, and travelled a total distance of 3.3 ± 0.6 units. This shows that we were able to obtain this lobbing behaviour, but it did not result in an overall improved kick, and can still be improved.

4 Fat Proxy Optimisation

The Fat Proxy is a proxy for client agents, connecting to the Simspark server. It extends the magmaProxy by supporting high-level dash and kick commands for agents. By using this proxy, teams are put on an equal playing field with respect to kicking and walking skills. During RoboCup 2021 we competed successfully for the first time in the FatProxy competition, where we were able to win a number of games. However, we were outmatched by Magma Offenburg and FCPortugal who demonstrated far better walking and kicking. We later learned this was as a result of their optimised parameter selection for the dash and kick action. In this section we outline our own approach for identifying the optimal parameter selection for kicking given a desired kick destination.

4.1 Background

Fat Proxy for the RoboCup 3D Soccer Simulation League was created by the magmaOffenburg team [4]. To have simplified access to the simspark simulation, the Fat Proxy takes over all motor control of the simulated Nao robots. Agents control the robot with high-level dash and kick commands. This is achieved by forwarding simspark observations to both the agent and our magma fatproxy agent.

The following server message is utilised to make use of the Fat Proxy kick :
(proxy kick <power> <horizontalAngle> <verticalAngle>)

- <power>: The desired kick power ($0 \leq$ power ≤ 10)
- <horizontalAngle>: The horizontal kick direction relative to the player ($-180 \leq$ horizontalAngle ≤ 180, 0 is straight)
- <verticalAngle>: The (positive) vertical angle for high kicking ($0 \leq$ verticalAngle ≤ 70)

For example, the message "(proxy kick 8.5 -50.028 10.0)" would result in a kick with a power of 8.5, 50 degrees to the right and an initial 10 degrees vertical angle.

4.2 Methodology

Now, while the Fat Proxy kick allows for powerful and accurate kicks, it is not immediately clear which input parameters (power, horizontal and vertical) would result in which final ball location, in terms of x, y position relative to the player. When deciding to kick, however, we only have a target in mind, and would thus like to find a mapping between the input parameters and resultant ball position after performing the kick.

To do this, we performed a grid search over the three input parameters, measured the resultant ball location and stored this in a look-up-table. To automate this, we built a test environment, which takes in these three values, as well as the robot's orientation and position relative to the ball, and simply performs a Fat Proxy kick with these parameters. This process is outlined in Figure 2.

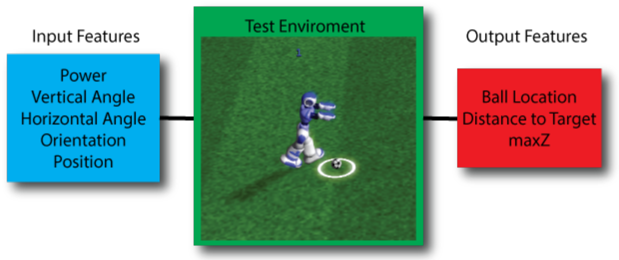


Figure 2: FatProxy Kick Parameter Selection Optimisation

We measure the performance of each set of input features by tracking both the distance to the target location as well as the vertical height achieved. The reason for this is to learn a set of features which can be used for two different scenarios. The first being where we have an open passing lane between the current and

[4]https://github.com/magmaOffenburg/magmaFatProxy

a target positions, meaning there is no obstructions and therefore we are free to pass in the most accurate manner. The second scenario is when there is an obstruction and therefore we are required to lob the ball into the air towards a target. The best performing parameters are selected and configured into a look up database. For any given situation, the lookup table can then be used to predict the appropriate kick power and angle through the use of interpolation.

4.3 Results

With this approach, it was found that reliable passing to within the radius of the "kickable margin" of 0.44 units was achievable. In the test environment, it was found that tuning the agent's ball approach parameters further improved the stability of the agent (and therefore the passing performance) for targets in a greater arc around the player - by taking advantage of remote interaction between the agent and the ball when performing a Fat Proxy kick.

5 Future Work

While our current reinforcement learning endeavours are promising, and have resulted in kicks that are better than our current ones, much work remains to be done. We must still integrate the Python-based RL agents with the C++-based code, investigate and improve the generalisation of the kicks, and improve stability after the kick. We have additional ideas for this project; in particular, investigating training the agents using a learned dynamics model, which would allow us to train much faster than the server simulation allows. Furthermore, we would like to explore the effect of using additional observations, such as derivatives of each joint's angle [Abreu et al., 2019, Spitznagel et al., 2021], or adding in the temporal history of the observations. Finally, once we have successfully done all of this, we would like to obtaining optimised behaviour for other tasks, such as walking, or even more complex, real-world scenarios such as penalty kicks or certain passing manoeuvres.

In terms of FatProxy, the question still remains on how to efficiently move the ball up from a defensive position in our own half to an attacking one in the opponent's. To this end, we are considering the team of players to be a weighted connected graph through which the ball needs to travel. Here the weight along each edge would be dependent on a number of factors, for example distance. We hypothesise that through this approach we can utilise a shortest path algorithm to determine the optimal path to the goal.

References

University of the Witwatersrand, Johannesburg. https://www.wits.ac.za/.

M. Abreu, N. Lau, A. Sousa, and L. P. Reis. Learning low level skills from scratch for humanoid robot soccer using deep reinforcement learning. In *ICARSC*, pages 1–8. IEEE, 2019.

G. Berseth, C. Xie, P. Cernek, and M. Van de Panne. Progressive reinforcement learning with distillation for multi-skilled motion control. *arXiv preprint arXiv:1802.04765*, 2018.

N. Bohlinger, H. Braun, K. Dorer, L. Ehlers, S. Glaser, D. Huber, H. Huber, R. Schillings, J. Scholz, and M. Wolffram. Magma offenburg team description paper, 2022.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, chapter 1. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989. ISBN 0201157675.

T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103*, 2018.

P. MacAlpine and P. Stone. UT Austin Villa Robocup 3D Simulation Team. https://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/.

P. MacAlpine, S. Barrett, D. Urieli, V. Vu, and P. Stone. Design and optimization of an omnidirectional humanoid walk: A winning approach at the robocup 2011 3d simulation competition. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

V. Rayward-Smith, I. Osman, C. Reeves, and G. Simth. *Modern Heuristic Search Methods*. 01 1996.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

M. Spitznagel, D. Weiler, and K. Dorer. Deep reinforcement multi-directional kick-learning of a simulated robot with toes. In *2021 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 104–110. IEEE, 2021.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

H. Teixeira, T. Silva, M. Abreu, and L. P. Reis. Humanoid robot kick in motion ability for playing robotic soccer. In *ICARSC*, pages 34–39. IEEE, 2020.