Wits FC Team Description Paper (RoboCup 2022)

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

February 28, 2022

Abstract. This document provides an outline of the Wits FC team, as an entry into the RoboCup 2022 3D Soccer Simulation League. We describe how we have built off the base code made freely available by UT Austin Villa [9]. This paper describes the two main contributions that distinguish us from the original code base, namely a low level parameter optimisation approach and infrastructure improvements in the form of a python wrapper.

1 Introduction

Wits FC first started competing in the 3D Simulation League of RoboCup in 2021. The team forms part of the RAIL Lab [6] from the School of Computer Science and Applied Mathematics at the University of the Witwatersrand (Wits) [8] in South Africa. Although our research group has long-standing experience in robotics and reinforcement learning, we are relatively new to RoboCup, with a small mix of undergraduate and postgraduate students. In order to avoid starting from scratch, we bootstrapped from the base code release of UT Austin Villa [9]. This allowed us to focus our team development on aspects more closely related to our algorithmic and machine learning expertise.

For our performance at RoboCup 2021, we were awarded the "Best New Team" award. More importantly, the experience presented a significant learning curve encouraging us to continue to develop as a team. We saw better-than-expected performance against multiple teams who had many years of experience, highlighting the fact that even without fundamental components such as optimised walking or kicking, our team could still reveal weaknesses in other teams. 2021 was also the first year that Wits FC competed in the Brazil Open 3D simulation league. Here we performed even better as we were able to win our first match. These leagues have been really beneficial to our development since there are currently no local leagues in South Africa, or even Africa. We are aiming to export our infrastructure to other universities to help them get started in the future. However, this lack of a local ecosystem has meant that we have had to rely exclusively on internal competitions within our own group in order to develop our skills. It is hoped that with this momentum and the continued experiences of international competitions we can continue to grow in the years to come.

The experience we have gained over the last year has heavily influenced our efforts to improve our team. Our most glaring weakness that we saw was our un-optimised walk and kick behaviours, which were slower and weaker than the other teams. These behaviours are defined by a set of parameters values, and we sought to bring our experience in machine learning and optimisation to bear on improving these parameters. Our approach described in Section 3 to optimisation looks to utilise simulated annealing and genetic algorithms to obtain improved behaviours. Additionally we sought ways in which we could leverage our team's experience with reinforcement learning and other machine learning algorithms. To this end we looked to develop our code base infrastructure as described in Section 4 in a way which could better support our needs and connect to our theoretical research programme.

2 Overview

As mentioned above, 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 provides many data structures that keep track of the state of the game as well as each agent.

Our focus in the first year of competing was in adding our own high-level behaviours, which enabled the agent to make appropriate decisions according to different situations. Specifically, we implemented a dynamic formation assignment strategy that uses the Stable Marriage algorithm [4] to pair players to positions on the field as seen in Figure 1.

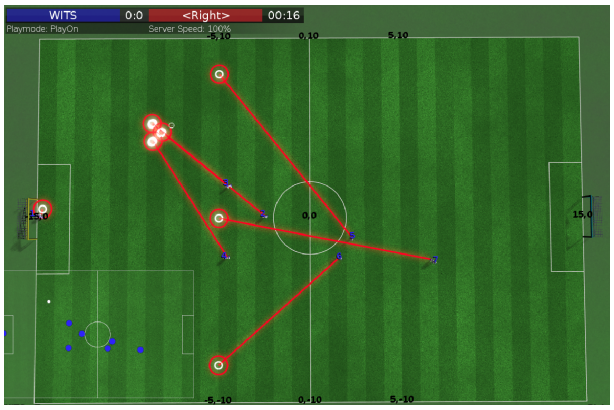


Figure 1: Optimised Formation Assignment using Stable Marriage

In addition, we developed systems to evaluate the viability of kicking to different kinds of target locations. This was utilised in both our kick type selection routines as well as passing behaviours. Lastly, through our own internal smaller format tournaments we developed a system that minimised fouls when it came to tackling. This was achieved by implementing a ray-casting system to determine the viability of passes based on a bounding rectangle, as seen in Figure 2

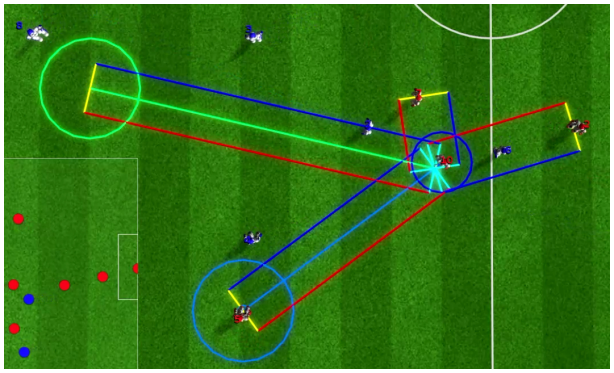


Figure 2: Optimised Passing Bounding Rectangle

3 Low-level Optimisation of Skills

Similarly to the Austin Villa team [3], we optimise the various parameters of the walking and kicking systems to obtain optimised behaviours in an automatic way. These systems are parameterised by values stored in a

parameter file, with each configuration setting affecting some aspect of the existing controller, like walking (e.g. stride length), getting up, and kicking (e.g. how far away from the ball the kick should start).

3.1 Method

The way we approach this is to view the parameter vector v as the input to some function $f(v)$ we want to optimise and use heuristic optimisation methods to solve this problem. In particular, as [3] describes, there are many different parameters, and not all of them affect all parts of the robot (e.g., the kick parameters do not affect walking speed). We thus only consider a subset of parameters for each task and change only these while keeping the rest of the parameters fixed. This has the added benefit of reducing the complexity of the optimisation, as there are fewer degrees of freedom.

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

3.1.1 Genetic Algorithms

Genetic Algorithms are a class of search algorithms which are loosely based on the mechanics of natural selection [1]. The main idea behind genetic algorithms is that there is a population of individuals, where each individual is represented by a genotype, which can be anything from bit strings or integer vectors, to the weights of a neural network. In our case, this genotype is the parameter vector v. Each individual represents a solution to the optimisation problem, and the main goal of these methods is to find the best solution according to some criteria.

Every individual is then assigned a *fitness* score, which measures the quality of the individual with respect to some desired solution. This is important because it then allows the highest fitness individuals to be chosen to "reproduce and pass on their genetic information".

This reproduction method is called crossover, where the genotypes of two parents are combined to form the genotype of the child. Mutation is then performed on the child, which randomly alters its genotype to maintain genetic diversity and facilitate exploration. Ultimately, by adding individuals to the reproduction pool in proportion to their fitness, the highest fitness individuals reproduce more often. This results in good traits that perform well being maintained and ill-performing traits being discarded.

This procedure is then repeated many times or until a sufficiently good solution is found.

3.1.2 Simulated Annealing

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 [5]. This process can be applied to a range of optimisation problems, where the current solution (which has energy E_1) is perturbed slightly, and the energy level E_2 is determined. The energy can be simply any metric that we want to minimise, like distance in the travelling salesman problem [5, 2], or, for example, the negative of the distance travelled by the robot in 10 seconds. If $E_2 < E_1$, we simply accept the perturbed solution. Otherwise, we accept the perturbed solution with a probability $\exp(-\frac{E_2-E_1}{T})$, where T is the current temperature, which starts off large and gradually decreases. This allows the algorithm to explore initially and escape from local optima, but as time goes on it explores less and focuses on the current best solution.

3.1.3 Scenarios

We mainly focused on creating scenarios for the walking and kicking tasks. For walking, the goal was to improve the agents' strides to move around the field faster. As a proof of concept, we start with a task where the agent must walk in a straight line, and its fitness/performance is 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 of the UT Austin Villa codebase, see Table 1) but ended up not

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 3a

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

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 3b 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

being very useful in actual games as the parameters were somewhat overfitted to the simple setting, and did not fare well when needing to deal with e.g. turns. As shown in Table 1, the optimised parameters were also somewhat inconsistent, with much higher standard deviation than the base parameters. This manifests as performing well most of the time, but failing completely some of the time.

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. We still optimise the total distance travelled within a certain amount of time.

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 2-3$\times$ 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. The results for this scenario are shown in Table 2, and while the genetic algorithm improved upon the default parameters, the simulated annealing parameters doubled even this score.

We remedied this problem by again creating a more complex scenario that potentially involves multiple kicks, while also moving towards the ball. The robot starts at location $(x, y) = (-2, 0)$, with the ball at the origin, and the objective is to obtain a goal within 50 seconds.

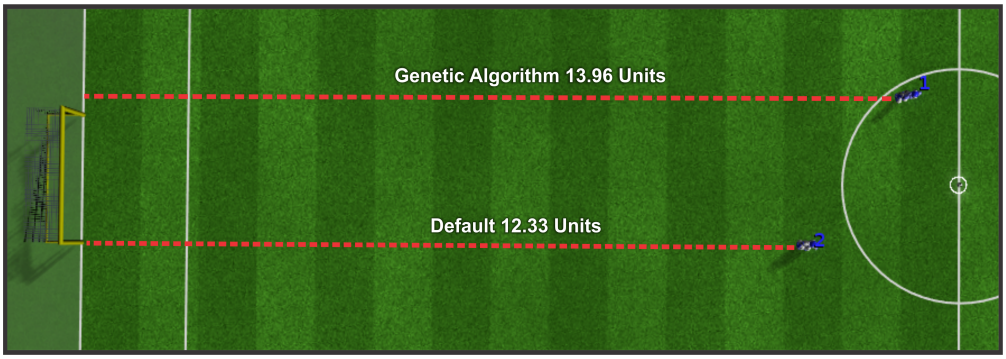
3.2 Limitations

Naïvely implementing this method does lead to some problems, however. Here we detail a few of these problems and our solutions.

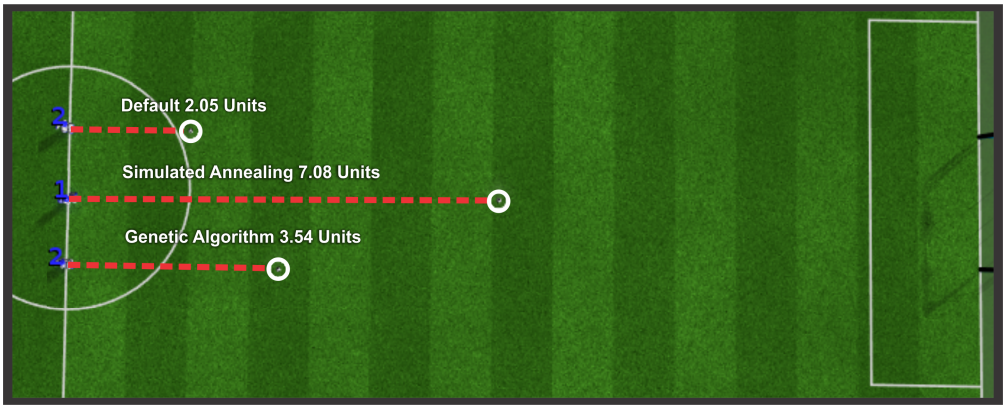
3.2.1 Noise

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. This leads to potential problems where, if we were to simply run one instance of a scenario per parameter configuration, a truly good v might perform badly, or a truly bad v might perform well. This could derail the entire optimisation process, resulting in worse solutions that are erroneously marked as high-performing.

To overcome this, for each parameter vector v, we run N $(40 \leq N \leq 80)$ simulations starting from the same state. The performance of this vector is then the mean of these N results. Whenever the agent catastrophically fails, we assign it a large negative value as fitness (which makes the overall mean very



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

low). This means that parameters that consistently perform decently will be rated higher than those that sometimes perform well and sometimes fail completely.

3.2.2 Computational Resources

When implementing a population-based method such as genetic algorithms, the amount of computing required to obtain high-performing results can be prohibitively high. In part, this is because it takes an entire generation (consisting of many individuals) to be simulated before any improvement can take place. This can be alleviated through the use of parallelism, but only to a point. Because of this, we mostly focused our efforts on Simulated Annealing (SA), as one only needs to evaluate one parameter vector at a time, and progress is faster. We still leverage parallelism here, however, mainly when simulating the N runs simultaneously.

3.2.3 Challenging Scenarios

As discussed above, we constructed more complex scenarios to improve the parameters in more realistic settings. This, however, increased the difficulty of the optimisation problem and resulted in marginal gains. Ways to alleviate this problem include optimising a smaller subset of the parameters at a time, although this might hamper the quality of the parameters, as some are interconnected and only changing one affects the performance negatively. These are the areas and approaches that we are looking to still explore further.

4 Infrastructure Improvements

The original UTAustinVilla codebase has been immensely useful in greatly reducing the time required to develop a platform where we could begin to start competing in RoboCup. One limitation of the codebase, however, is that it was developed in C++. Therefore, we are constrained by which libraries we can use for further development as well as optimisation purposes. To help circumvent this limitation we have developed a wrapper (https://github.com/SD-Group-17/RobocupPy) that is used to parse C++ objects into Python objects. These include, but are not limited to, the following data objects and methods:

- player_number – object storing the player's UNUM
- play_mode – object storing the server assigned play mode
- side – object storing which side of the field our team was loaded in on
- player_pos – object storing a 3D vector of the current agent's position
- ball_pos – object storing a 3D vector of the ball's position
- team_positions – object storing the 3D positions of all the agents from our team
- opponent_positions – object storing the 3D positions of all the agents from the opponent's team
- team_dist_ball – a method calculating all the distances between all agents from our team and the ball
- opp_dist_ball – a method calculating all the distances between all agents from our team and the opponent's team

The system was implemented in a manner that was flexible enough where we could add additional objects when required. Our wrapping system works by directly integrating itself within the C++ codebase. During an agent's decision-making process the objects mentioned above are converted to "PyObject*" objects and piped directly into a python script. This script then reconstructs the same data structures so that they can be used to perform any decision-making which would have originally been performed in C++. The result of this decision is then returned back to the C++ codebase where a low-level skill is executed. The benefit of this approach is that we are able to utilise the multitude of libraries available with python. Additionally, changes made to the python script do not require any rebuilding of the C++ codebase. Through testing, we also determined that the overhead of this added pipeline did not exceed the maximum time allowed for agent decision making. We have yet to utilise this new infrastructure, however, it is part of our future plans. Additionally, since we have benefited from the work of others, we would also like to make this tool public. It is hoped that through it's continued development that Wits FC can help contribute to the community.

5 Future Work

The past year has seen the team's focus shift towards developing systems that can allow us to catch up to the other teams. These systems are our low-level optimisation framework and our python wrapper. Whereas before, we could only focus on high-level strategy now we have the tools to tackle a wider range of problems in more intelligent ways. Many of these problems that we want to explore were identified as a result of our experiences in RoboCup 2021.

Our primary project is to utilise our newly implemented optimisation framework in order to learn optimal parameters for our walk and kick skills. In particular, we are aiming to implement a reinforcement learning optimisation approach where the action space covers a range of possible values for our parameters. This experimental setup is akin to a traditional bandit problem [7]. In addition, we are looking to improve our robustness by decreasing the variation when running the same parameter file multiple times. We would hope to aim to learn multiple different styles of kicks, which could be useful in various situations. This could be

achieved by developing behaviours catered towards more niche skills. For example, being able to use different kick skills for different target distances.

Additionally, we look to utilise our python wrapper to employ reinforcement learning (RL) techniques to learn optimal high-level action selection. Using the state (s) based information such as player positions we can define a set of actions (a) from our set of skills which include "KICK" and "GOTO". By using this framework and the RoboCup simulation as our environment we could use a range of RL techniques to learn an optimal policy $\pi(a, s)^*$. Here the state information gathered by the individual agent will be sent to the python script via our wrapper and an action will be selected and passed back to the agent. The agent will process this action by executing the corresponding skill. This cycle will repeat for each episode of the simulation. The simulation allows for freedom in defining the reward function as it is episodic in nature and we can pass information between both frameworks during each episode. This means we could define a sparse positive reward only for when we score, or we could define a dense reward dependant on the distance of the ball to the opponent's goal.

References

[1] David E. Goldberg. "Genetic Algorithms in Search, Optimization and Machine Learning". In: 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. Chap. 1. ISBN: 0201157675.

[2] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680.

[3] Patrick MacAlpine et al. "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.

[4] David G McVitie and Leslie B Wilson. "The stable marriage problem". In: *Communications of the ACM* 14.7 (1971), pp. 486–490.

[5] Vic Rayward-Smith et al. *Modern Heuristic Search Methods*. Jan. 1996.

[6] *Robotics, Autonomous Intelligence, and Learning (RAIL) Lab*. `https://www.raillab.org/`. (accessed: 16.04.2021).

[7] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[8] *University of the Witwatersrand, Johannesburg*. `https://www.wits.ac.za/`. (accessed: 16.04.2021).

[9] *UT Austin Villa Robocup 3D Simulation Team*. `https://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/`. (accessed: 16.04.2021).