To appear in *Proceedings of the RoboCup International Symposium 2016* (RoboCup 2016), Leipzig, Germany, July 2016.

UT Austin Villa RoboCup 3D Simulation Base Code Release

Patrick MacAlpine and Peter Stone

Department of Computer Science, The University of Texas at Austin, USA
{patmac,pstone}@cs.utexas.edu

Abstract. This paper presents a base code release by the UT Austin Villa RoboCup 3D simulation team from the University of Texas at Austin. The code release, based off the 2015 UT Austin Villa RoboCup champion agent, but with some features such as high level strategy removed, provides a fully functioning agent and good starting point for new teams to the RoboCup 3D simulation league. Additionally the code release offers a foundational platform for conducting research in multiple areas including robotics, multiagent systems, and machine learning.

1 Introduction

The RoboCup 3D simulation environment is a 3-dimensional world that models realistic physical forces such as friction and gravity, in which teams of autonomous soccer playing humanoid robot agents compete with each other. Programming humanoid agents in simulation, rather than in reality, brings with it several advantages, such as making simplifying assumptions about the world, low installation and operating costs, and the ability to automate experimental procedures. All these factors make the RoboCup 3D simulation environment an ideal domain for conducting research in robotics, multiagent systems, and machine learning.

The UT Austin Villa team, from the University of Texas at Austin, first began competing in the RoboCup 3D simulation league in 2007. Over the course of nearly a decade the team has built up a strong state of the art code base enabling the team to win the RoboCup 3D simulation league four out of the past five years (2011 [14], 2012 [8], 2014 [9], and 2015 [11]) while finishing second in 2013. It is difficult for new RoboCup 3D simulation teams to be competitive with veteran teams as the complexity of the the RoboCup 3D simulation environment results in an often higher than expected barrier of entry for new teams wishing to join the league. With the desire of providing new teams to the league a good starting point, as well as offering a foundational platform for conducting research in the RoboCup 3D simulation domain, UT Austin Villa has released the base code for its agent team. This paper presents the code release. Due to space constraints some details of the agent code release are left out, but are covered in other documents including a team technical report [13].

The remainder of the paper is organized as follows. In Section 2 a description of the 3D simulation domain is given. Section 3 gives an overview of the code release and what it includes. A high level view of the agent's architecture is provided in Section 4 with several of the agent's features (walk engine, skill description language, and optimization task infrastructure) highlighted in Section 5. Section 6 references other code releases, and Section 7 concludes.

2 RoboCup 3D Simulation Domain Description

The RoboCup 3D simulation environment is based on SimSpark [17], a generic physical multiagent system simulator. SimSpark uses the Open Dynamics Engine[1] (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. ODE also provides support for the modeling of advanced motorized hinge joints used in the humanoid agents.

Games consist of 11 versus 11 agents playing on a 30 meters in length by 20 meters in width field. The robot agents in the simulation are approximately modeled after the Aldebaran Nao robot,[2] which has a height of about 57 cm, and a mass of 4.5 kg. Each robot has 22 degrees of freedom: six in each leg, four in each arm, and two in the neck. In order to monitor and control its hinge joints, an agent is equipped with joint perceptors and effectors. Joint perceptors provide the agent with noise-free angular measurements every simulation cycle (20 ms), while joint effectors allow the agent to specify the speed and direction in which to move a joint.

Visual information about the environment is given to an agent every third simulation cycle (60 ms) through noisy measurements of the distance and angle to objects within a restricted vision cone (120°). Agents are also outfitted with noisy accelerometer and gyroscope perceptors, as well as force resistance perceptors on the sole of each foot. Additionally, agents can communicate with each other every other simulation cycle (40 ms) by sending 20 byte messages.

In addition to the standard Nao robot model, four additional variations of the standard model, known as heterogeneous types, are available for use. These variations from the standard model include changes in leg and arm length, hip width, and also the addition of toes to the robot's foot. Teams must use at least three different robot types, no more than seven agents of any one robot type, and no more than nine agents of any two robot types.

Figure 1 shows a visualization of the Nao robot and the soccer field.

3 Code Release Overview

The UT Austin Villa base code release, written in C++ and hosted on GitHub,[3] is based off of the 2015 UT Austin Villa agent. A key consideration when releasing the team's code is what components should and should not be released. A complete full release of the team's code could be detrimental to the RoboCup

[1] `http://www.ode.org/`
[2] `http://www.aldebaran-robotics.com/eng/`
[3] UT Austin Villa code release: `https://github.com/LARG/utaustinvilla3d`

Fig. 1: A screenshot of the Nao robot (left), and a view of the soccer field during a game (right).

3D simulation community if it performs too strongly. In the RoboCup 2D soccer simulation domain the former champion Helios team released the Agent2D code base [1] that allowed for teams to be competitive by just typing `make` and running the code as is. Close to 90% of the teams in the 2D league now use Agent2D as their base effectively killing off their original code bases and resulting in many similar teams. In order to avoid a similar scenario in the 3D league certain parts of the team's code have been stripped out. Specifically all high level strategy, some optimized long kicks [2], and optimized fast walk parameters for the walk engine [7] have been removed from the code release. Despite the removal of these items, which are described in detail in research publications [13, 14, 7, 6, 2, 12, 10], we believe it should not be too difficult for someone to still use the code release as a base, and develop their own optimized skills (we provide an example of how to do this with the release) and strategy, to produce a competitive team.

The following features are included in the release:

– Omnidirectional walk engine based on a double inverted pendulum model [7]
– A skill description language for specifying parameterized skills/behaviors
– Getup (recovering after having fallen over) behaviors for all agent types
– A couple basic skills for kicking one of which uses inverse kinematics [14]
– Sample demo dribble and kick behaviors for scoring a goal
– World model and particle filter for localization
– Kalman filter for tracking objects
– All necessary parsing code for sending/receiving messages from/to the server
– Code for drawing objects in the RoboViz [15] monitor
– Communication system previously provided for drop-in player challenges[4]
– An example behavior/task for optimizing a kick

What is not included in the release:

– The team's complete set of skills such as long kicks [2] and goalie dives
– Optimized parameters for behaviors such as the team's fastest walks (slow and stable walk engine parameters are included, as well as optimized parameters for positioning/dribbling [7] and approaching the ball to kick [9])
– High level strategy including formations and role assignment [6, 12]

[4] `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/2015_dropin_challenge/`

4 Agent Architecture

At intervals of 0.02 seconds, the agent receives sensory information from the environment. Every third cycle a visual sensor provides distances and angles to different objects on the field from the agent's camera, which is located in its head. It is relatively straightforward to build a world model by converting this information about the objects into Cartesian coordinates. This of course requires the robot to be able to localize itself for which we use a particle filter incorporating both landmark and field line observations [4, 9]. In addition to the vision perceptor, the agent also uses its accelerometer readings to determine if it has fallen and employs its auditory channels for communication.

Once a world model is built, the agent's control module is invoked. Figure 2 provides a schematic view of the UT Austin Villa agent's control architecture.

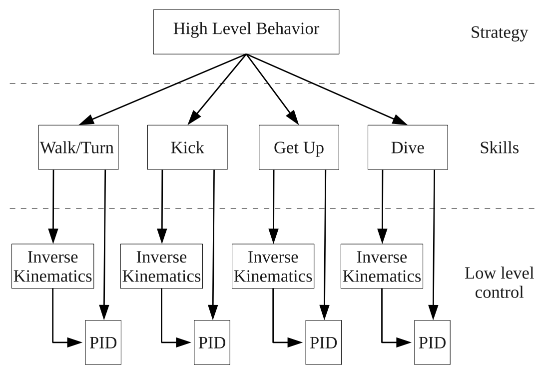

Fig. 2: Schematic view of UT Austin Villa agent control architecture.

At the lowest level, the humanoid is controlled by specifying torques to each of its joints. This is implemented through PID controllers for each joint, which take as input the desired angle of the joint and compute the appropriate torque. Further, the agent uses routines describing inverse kinematics for the arms and legs. Given a target position and pose for the hand or the foot, the inverse kinematics routine uses trigonometry to calculate the angles for the different joints along the arm or the leg to achieve the specified target, if at all possible.

The PID control and inverse kinematics routines are used as primitives to describe the agent's skills. In order to determine the appropriate joint angle sequences for walking and turning, the agent utilizes an omnidirectional walk engine which is described in Section 5.1. Other provided useful skills for the robot are kicking and getting up from a fallen position. These skills are accomplished through a programmed sequence of poses and specified joint angles as discussed in Section 5.2. One of the kicking skills provided in the code release uses inverse kinematics to control the kicking foot such that it follows an appropriate trajectory through the ball as described in [14].

High level strategy has been removed from the code release, however some sample behaviors such as dribbling and kicking the ball into the goal are included.

5 Feature Highlights

The following subsections highlight several features of the UT Austin Villa code release. These features include an omnidirectional walk engine (Section 5.1), skill description language (Section 5.2), and optimization task infrastructure (Section 5.3). When combined together these features provide a nice platform for machine learning and optimization research.

5.1 Omnidirectional Walk Engine

Agents use a double inverted pendulum omnidirectional walk engine [7] to move. The omnidirectional walk is crucial for allowing the robot to request continuous velocities in the forward, side, and turn directions, permitting it to approach continually changing destinations (often the ball).

The walk engine has parameterized values that control such things as step height, length, and frequency. Walk engine parameters are loaded at runtime from parameter files and can be switched on the fly for different walking tasks (e.g. approaching the ball, sprinting, and dribbling). A slow and stable set of walk engine parameters is included with the release, and these parameters can be optimized to produce a faster walk [7].

5.2 Skill Description Language

The UT Austin Villa agent includes skills for getting up and kicking, each of which is implemented as a periodic state machine with multiple *key frames*, where a key frame is a static pose of fixed joint positions. Key frames are separated by a waiting time that lets the joints reach their target angles. To provide flexibility in designing and parameterizing skills, we designed an intuitive skill description language that facilitates the specification of key frames and the waiting times between them. Below is an illustrative example describing a kick skill.

```
SKILL KICK_LEFT_LEG

KEYFRAME 1
setTarget JOINT1 $jointvalue1 JOINT2 $jointvalue2 ...
setTarget JOINT3 4.3 JOINT4 52.5
wait 0.08

KEYFRAME 2
increaseTarget JOINT1 -2 JOINT2 7 ...
setTarget JOINT3 $jointvalue3 JOINT4 (2 * $jointvalue3)
wait 0.08
.
.
.
```

As seen above, joint angle values can either be numbers or be parameterized as `$<varname>`, where `<varname>` is a variable value that can be loaded after being learned. Values for skills and other configurable variables are read in and loaded at runtime from parameter files.

5.3 Optimization Task Infrastructure

A considerable amount of the UT Austin Villa team's efforts in preparing for RoboCup competitions has been in the area of skill optimization and optimizing parameters for walks and kicks. An example agent for optimizing a kick is provided with the code release. Optimization agents perform some task (such as kicking a ball) and then determine how well they did at the task (such as how far they kicked the ball) which is known as the agent's *fitness* for the task. Optimization agents are able to adjust the values of parameterized skills at runtime by loading in different parameter files as mentioned in Section 5.2, thus allowing the agents to easily try out and evaluate different sets of parameter values for a skill. After evaluating itself on how well it did at a task, an optimization agent writes its *fitness* for the task to an output file.

Optimization agents can be combined with machine learning algorithms to optimize and tune skill parameters for maximum *fitness* on a task. The UT Austin Villa team uses the CMA-ES [3] policy search algorithm for this purpose. During optimization, agents try out different parameter values from loaded parameter files written by CMA-ES, and then the agents write out their *fitness* values indicating how well they performed with those parameters so that CMA-ES can attempt to adjust the parameters to produce higher *fitness* values. UT Austin Villa utilizes overlapping layered learning [10] paradigms with CMA-ES to optimize skills that work well together.

When performing an optimization task, agents are able to change the world as needed (such as move themselves and the ball around) by sending special training command parser commands[5] to the server.

6 Other Code Releases

There have been several previous agent code releases by members of the RoboCup 3D simulation community. These include releases by magmaOffenburg[6] (Java 2014), libbats[7] (C++ 2013), Nexus[8] (C++ 2011), and TinMan[9] (.NET 2010). The UT Austin Villa code release (C++ 2016) expands on these previous code releases in a number of ways. First the UT Austin Villa code release offers a proven base having won the RoboCup 3D simulation competition four out of the past five years. Second the release provides an infrastructure for carrying out optimization and machine learning tasks, and third the code is up to date to work with the most recent version of the RoboCup 3D simulator (rcssserver3d 0.6.10).

[5] `http://simspark.sourceforge.net/wiki/index.php/Network_Protocol#Command_Messages_from_Coach.2FTrainer`

[6] `http://robocup.hs-offenburg.de/uploads/media/magmaOffenburg3D-2014Release.tar.gz`

[7] `https://github.com/sgvandijk/libbats`

[8] `http://nexus.um.ac.ir/index.php/downloads/base-code`

[9] `https://github.com/drewnoakes/tin-man`

7 Conclusion

The UT Austin Villa RoboCup 3D simulation team base code release provides a fully functioning agent and good starting point for new teams to the RoboCup 3D simulation league. Additionally the code release offers a foundational platform for conducting research in multiple areas including robotics, multiagent systems, and machine learning. We hope that the code base may both inspire other researchers to join the RoboCup community, as well as facilitate non-RoboCup competition research activities akin to the reinforcement learning benchmark keepaway task in the RoboCup 2D simulation domain [16].

Recent and ongoing work within the RoboCup community is the development of a plugin[10] for the Gazebo[11] [5] robotics simulator to support agents created for the current RoboCup 3D simulation league simulator (SimSpark). The UT Austin Villa code release has been tested with this plugin and provides an agent that can walk in the Gazebo environment. As the development of the plugin continues further support of the Gazebo plugin by the UT Austin Villa code release, such as providing getup behaviors that work in Gazebo, is planned.

A link to the UT Austin Villa 3D simulation code release, as well as additional information about the UT Austin Villa agent, can be found on the UT Austin Villa 3D simulation team's homepage.[12]

Acknowledgments

This code release is based upon the work of all past UT Austin Villa RoboCup 3D simulation team members. We thank all the previous team members for their important contributions. This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287), ONR (21C184-01), and AFOSR (FA9550-14-1-0087). Peter Stone serves on the Board of Directors of, Cogitai, Inc. The terms of this arrangement have been reviewed and approved by UT Austin in accordance with its policy on objectivity in research.

References

1. Akiyama, H., Nakashima, T.: Helios base: An open source package for the robocup soccer 2d simulation. In: RoboCup 2013: Robot World Cup XVII, pp. 528–535. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
2. Depinet, M., MacAlpine, P., Stone, P.: Keyframe sampling, optimization, and behavior integration: Towards long-distance kicking in the robocup 3d simulation league. In: RoboCup 2014: Robot Soccer World Cup XVIII. Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin (2015)
3. Hansen, N.: The CMA Evolution Strategy: A Tutorial (January 2009), `http://www.lri.fr/~hansen/cmatutorial.pdf`, `http://www.lri.fr/~hansen/cmatutorial.pdf`
4. Hester, T., Stone, P.: Negative information and line observations for monte carlo localization. In: IEEE Int. Conf. on Robotics and Automation (May 2008)

[10] `https://bitbucket.org/osrf/robocup3ds`
[11] `http://gazebosim.org/`
[12] `http://www.cs.utexas.edu/~AustinVilla/sim/3dsimulation/`

5. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on. vol. 3, pp. 2149–2154 vol.3 (Sept 2004)
6. MacAlpine, P., Barrera, F., Stone, P.: Positioning to win: A dynamic role assignment and formation positioning system. In: RoboCup-2012: Robot Soccer World Cup XVI. Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin (2013)
7. MacAlpine, P., Barrett, S., Urieli, D., Vu, V., Stone, P.: Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI-12) (July 2012)
8. MacAlpine, P., Collins, N., Lopez-Mobilia, A., Stone, P.: UT Austin Villa: RoboCup 2012 3D simulation league champion. In: RoboCup-2012: Robot Soccer World Cup XVI. Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin (2013)
9. MacAlpine, P., Depinet, M., Liang, J., Stone, P.: UT Austin Villa: RoboCup 2014 3D simulation league competition and technical challenge champions. In: RoboCup 2014: Robot Soccer World Cup XVIII. Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin (2015)
10. MacAlpine, P., Depinet, M., Stone, P.: UT Austin Villa 2014: RoboCup 3D simulation league champion via overlapping layered learning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15) (January 2015)
11. MacAlpine, P., Hanna, J., Liang, J., Stone, P.: UT Austin Villa: RoboCup 2015 3D simulation league competition and technical challenges champions. In: RoboCup 2015: Robot Soccer World Cup XIX. Lecture Notes in Artificial Intelligence, Springer (2015)
12. MacAlpine, P., Price, E., Stone, P.: SCRAM: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15) (January 2015)
13. MacAlpine, P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., Stone, P.: UT Austin Villa 2011 3D Simulation Team report. Tech. Rep. AI11-10, The Univ. of Texas at Austin, Dept. of Computer Science, AI Laboratory (December 2011)
14. MacAlpine, P., Urieli, D., Barrett, S., Kalyanakrishnan, S., Barrera, F., Lopez-Mobilia, A., Ştiurcă, N., Vu, V., Stone, P.: UT Austin Villa 2011: A champion agent in the RoboCup 3D soccer simulation competition. In: Proc. of 11th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2012) (June 2012)
15. Stoecker, J., Visser, U.: Roboviz: Programmable visualization for simulated soccer. In: Rfer, T., Mayer, N., Savage, J., Saranl, U. (eds.) RoboCup 2011: Robot Soccer World Cup XV, Lecture Notes in Computer Science, vol. 7416, pp. 282–293. Springer Berlin Heidelberg (2012)
16. Stone, P., Sutton, R.S., Kuhlmann, G.: Reinforcement learning for RoboCup-soccer keepaway. Adaptive Behavior 13(3), 165–188 (2005)
17. Xu, Y., Vatankhah, H.: Simspark: An open source robot simulator developed by the robocup community. In: RoboCup 2013: Robot World Cup XVII, pp. 632–639. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)