Noon Team Description Paper
Humanoid KidSize League, RoboCup 2026

Máté Somfai, Miklós Máté Varga, Sámuel Virgo

Abstract. In this paper we introduce the newly formed Team noon, the team representing Hungary in the Humanoid SmallSize League in RoboCup. We present all of our plans for this year and outline a progression for future competitions. We describe every major system in the robot, including vision, movement, tactics, electronics and hardware.

Keywords: RoboCup • Humanoid Robot

1 Introduction

Our team was formed in 2025 in Veszprém, but we have members from all over the country from years of meeting people in competitions, for example in RoboCup Junior[1] and CanSat[2]. We have a mix of skills that each contribute something to our team but it is also important for our long-term sustainability to recruit more talented and aspiring teammates.

2 Hardware

We aim for our robot to be in the small, yet powerful category.
With our target height of 60 cm and weight of 2 kg, we would be at the lower limit of the BMI, which is the goal.
The arms are as simple as it gets, but the legs are where it gets interesting.
Thanks to our sponsor, Maxon, we have first-in-class motors, for which we need to design adequate gearboxes to support all forms of motion we have planned.

For the first iteration, we are making a 1:60 reduction spiroid gearbox, which will give us a nominal speed of approximately 140 rpm, a nominal torque of around 2 Nm and a stall torque just shy of 8 Nm (this will be lower due to current limits).

In the future, we will also be focusing on the development of active compliance, since achieving backdrivability at such high ratios in such a small form factor is practically impossible.
Our early attempts will include an elastic element, such as silicone in series with the gear (possibly with an exponential elasticity curve) to soften any momentary impacts and give us a higher gain on measuring force through deflection, and actively driving the motors in the opposite directions.

We want to experiment with running and other high-energy movements, for which absorbing impacts is essential.

3 Electronics

3.1 Compute

Electronics is divided into 4 major parts; the head assembly, the chest, the motor drivers and the battery packs.
These all have one or more microcontrollers per unit and communicate via a shared CAN bus.

We are using a dual-core CPU with an integrated 0.5 TOPs NPU that comes with 64 MB of RAM.
This is an extremely underpowered system compared to other teams' approaches, but it forces us to deeply think through every system in order to achieve the highest efficiency possible.

Through the rough calculations and estimates, we have concluded that, while limited, it will be performant enough for our first season.
Once we have the tools and libraries built out around this constrained system, migrating to a more powerful base will come with huge performance benefits.

3.2 Vision

We've chosen an old camera sensor for a few reasons.
It is widely available, affordable and its SMD package means we do not need cables to connect it to our processing system.
While it is in a "last shipments" phase, we have ordered 30+ (due to its discounted price) meaning it could be sufficient for up to 3 seasons.
For the second season, we are planning on creating a stereo vision setup (possibly with the same sensor) with either a faster, or two of the same CPU as now.

3.3 Power

We have two smaller battery packs, each powering one side of the robot, but they are redundant, so a single pack is able to power the whole robot under light loads.
This system allows us to rapidly replace batteries without any systems going unpowered.
Since we'll have plenty of these packs, we've also added USB-PD Source support, so each individual pack can act as 50Wh power banks, aiming to create less waste.

4 Tactics

We're using the lightweight Lua programming language to interface with our robots quickly and efficiently. We are currently working on a tactics library which will provide a framework for us to implement dynamic decision making for our robots. Two projects exist currently: the game management system which renders a virtual match, and the player logic system which we're planning to use for the competition. One of the robots is a captain, who will make decisions about how to proceed during the match. Network communication will be used to broadcast information to the captain, who will make up an effective strategy for the current state of the match.

We are also considering implementing a graphical behavior tree system, where we can easily update our tactics without the need of writing code.
It would export into a text-like/binary format that is then processed by the Lua script(s).

5 Balancing

To balance the robot reliably we need to build a very robust system, we achieved this by breaking up this daunting task into 3 smaller subsystems. A keyframing animator for standing up from any position, a keyframe generator for generating the animation for kicking the ball at a certain angle and a neural network for bipedal locomotion.
Balancing currently runs on the second, 700 MHz core of our CPU, but we will be experimenting with running it on a separate 400 MHz microcontroller that has no operating systems and is more direct with the IMU and the motor drivers in hopes of reducing latency, even at the cost of performance.

5.1 Keyframing

The keyframing animator is just as simple as it sounds; for each axis where a joint can move, it gets a series of keyframes it linearly interpolates between. But this simple system serves a lot of purposes. It is the base for the kicking animation generation and it is the way we made the standing up animations. In the making of it we also discovered a lot of minor and quite major bugs in our training environment.
On the robot, we are using a system we wrote for a previous RoboCup competition[3], which can efficiently interpolate between poses. We will be updating it to support a full robot, as it was only designed to support hands, but the specification for it was flexible enough so that it will be easy to do so.

5.2 The kicking animation generator

Making the robot kick the ball is also quite a complex problem because of the sheer amount of different variables at play. The angle the robot needs to kick at, the distance between the ball and the robot, the force it should kick with and where it should kick are all very important variables at play. So we needed to make a system that takes all these variables into account and alters the animation based off of them.

5.3 The neural network

The most complex and most important part of our balancing system is the neural network that generates the walking animation. The development of it has taken up a lot of our time and effort because we initially started working on it in Unity because our team was familiar with C# and the Unity workflow, and after implementing the majority of it we ran into the issue of not being able to simulate the joints properly and because of that we switched to Webots because from our research in bipedal walking[4] the best papers we found used Webots.

5.3.1 The reward system

Currently the robot receives 2 major rewards and 5 more minor rewards and penalties. The 2 major ones are for the main functions it has to achieve, which are a constant forward/backwards velocity, and a constant turn rate for the robot. With these two controls working reliably we will be able to calculate an approximate position for the robot, based on the input given to the neural network and the time that has passed.

We use the minor rewards to make the neural network more stable, and to not adapt a weird or unrealistic policy. Each one was added to address a specific problem we encountered during the development process. For example a penalty was added if the feet left the floor for too long, this was added because the robot learned to balance on one leg, and retraining with this penalty added made the robot not balance on one leg anymore. We also have a penalty for side to side motion, a penalty for vertical motion, a slight reward for the feet being on the ground and a slight penalty for moving the motors, so that it tries to conserve the amount of energy it uses.

5.3.2 The neurons and the training algorithm

The robot currently has 2 hidden layers with 64 neurons each, and for the neuron activation function we use leaky ReLU. The input layer consists of 29 values, since the neural network will not be in charge of moving the neck, it only receives the state of the other 18 motors, for the brushless DC motors it receives the actual current position of the motors, but since for the servos won't have that data it just receives their target position. Besides the current motor positions it also receives the angle (x, y and z) of the robot's body that will be measured with a gyroscope, the angular (x, y and z) and the linear (x, y and z) acceleration of the body, these will be measured using an accelerometer. Besides the data collected from sensors and the motor positions, it also receives 2 inputs, one that specifies the desired walking speed of the robot (in m/s), and one that specifies the speed the robot should turn at (in rad/s)

We are training the neural network with PPO in python because it has the largest ecosystem of machine learning libraries and resources we can learn from. PPO is very convenient for us because of its sample efficiency which is crucial because Webots is a very slow environment, but to speed it up we can run lots of environments in parallel. We achieved this by having a central trainer process that starts the specified amount of Webots instances, and in each instance starts the specified amount of robots. Each robot communicates with the trainer via local Linux sockets.

5.3.3 The future of our balancing

As of the writing of this paper the neural network hasn't yielded the greatest results. To fix this we are looking into implementing one more reward system which will reward the neural network based on how close the actual motor positions are to the target ones. This way we will be able to give it reference motions (like in reference 2) and make the walking more stable and for the robot to be able to balance during the kicking and standing up animations.

6 Acknowledgements

Our team wants to thank every open source and free resource available on the internet, these include GitHub where we can host the whole of our project, Onshape for the great service they provide for free to students, Webots, Kicad and the 2 major Python libraries we use which are PyTorch and Stable-Baselines3.

In addition we would like to thank our two sponsors that make this project possible, Infuze Robotics Hungary and Maxon Motors.

References

1. hu-more-bot: GitHub repository of the RoboCup Junior projects.
   Software, GitHub (2022-25). https://github.com/hu-more-bot
2. CanSat Hungary: Official website for CanSat Hungary competitions.
   Website (n.d.). https://cansathungary.hu/en/front-page/
3. hu-more-bot: PoseSim—A keyframing system for the RoboCup Junior OnStage 2024 competition.
   Software, GitHub (2024). https://github.com/hu-more-bot/RCJ2024/tree/main/src/pose-sim
4. Itahashi, N., Itoh, H., Fukumoto, H., Wakuya, H.: Reinforcement learning of bipedal walking using a simple reference motion. Applied Sciences 14(5), 1803 (2024). https://doi.org/10.3390/app14051803