ImperiSim 2.0: Bidirectional Communication Between SolidWorks and KISS-Sim
for 3D Robot Simulation with Motors, Sensors, and Physics (Part 2)
Jeremy Rand and Marty Rand
Norman Advanced Robotics
jeremy.rand@ou.edu, mario.tech.boy@gmail.com

ImperiSim 2.0: Bidirectional Communication Between SolidWorks and KISS-Sim for 3D Robot Simulation with Motors, Sensors, and Physics (Part 2)

12 Welcome to Part 2!

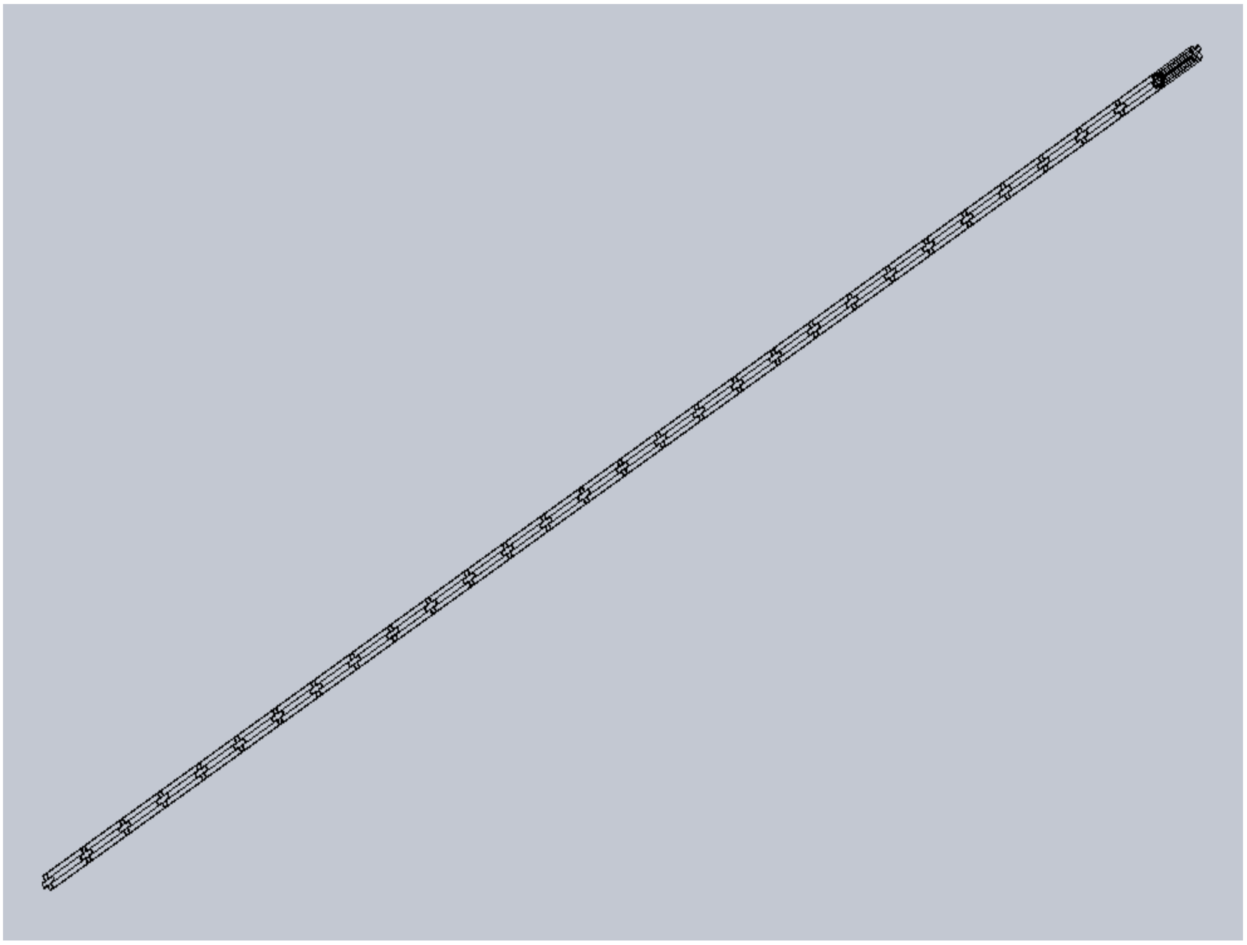
Welcome back to ImperiSim 2.0. In Part 2, we'll cover sensor implementations, an operational guide, and other things that didn't fit in Part 1.

13 Implementation: Touch Sensors

Touch sensors were implemented using SolidWorks's collision detection feature, known as Check Interference. Each segment, the touch-sensitive part of the sensor is compared with the environment to find any collision; detected collisions are reported as touch sensor values of 1. The big issue with this method is that Check Interference usually produces multiple collisions, some of which are typically noise (e.g. an object interfering with itself). Our short-term solution is to measure the number of collisions at the beginning of the simulation, store it in a calibration value within KISS-C for the entire simulation session, and only report collisions additional to that calibration value to the user program.

14 Implementation: Rangefinder Sensors

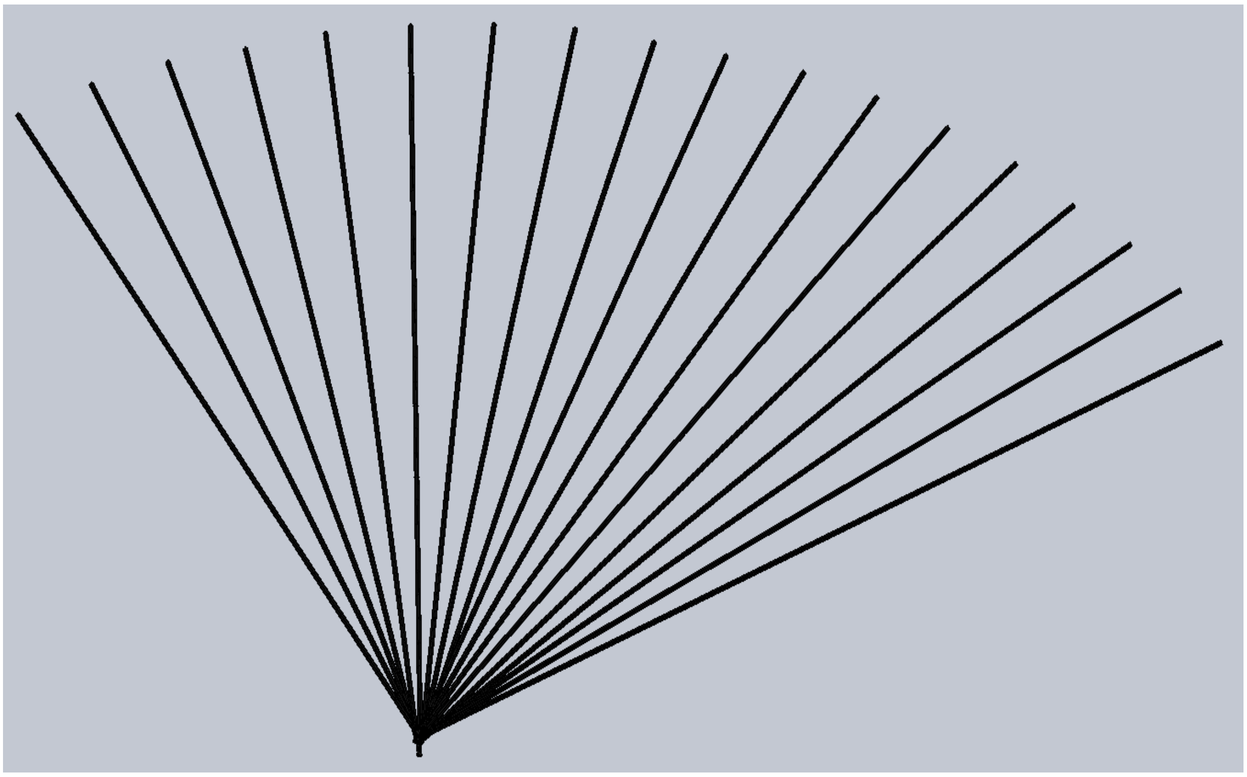
We initially planned to implement rangefinders using SolidWorks's ray-tracing API. Unfortunately, we were unable to make those API functions work consistently (we hope to revisit this issue later). So we improvised. A rangefinder is essentially a very long and thin touch sensor which passes through the objects with which it collides. Based on this principle, we created an assembly of 30 small Lego axles (960mm long according to SolidWorks's measurements), and treated each axle as a touch sensor. The triggered touch sensor which is closest to the robot is the distance reported.


(Wireframe Model of Rangefinder Beam)

Rangefinders in Botball are not necessarily comparable with each other. The IR reflectance sensors ("large top hat" and "small top hat"), Sharp GP2D12 ("ET"), and Maxbotix EZ-1 (CBC sonar) all have their own mapping between distance and analog voltage. We're still in the process of implementing these transformations, but when finished, we expect the operation to be transparent to the user. At the moment, a rangefinder takes approximately 0.5 seconds to fully evaluate; we're looking into ways to improve this performance.

15 Implementation: Camera Sensors

Our implementation of cameras is quite reminiscent of our implementation of rangefinders. Just as we treat rangefinders as arrays of touch sensors, we treat cameras as arrays of rangefinders. A camera is simulated using a rangefinder for each pixel. Instead of reporting the distance from the robot, we report which object was hit and which rangefinder was triggered.


(Camera Pixels Modeled as Rangefinder Beams)

A camera currently takes approximately 8 seconds to fully evaluate at default 18-pixel resolution; we're not satisfied with this delay and are investigating methods of improving this.

16 Inter-Segment Synchronization

At the end of each segment, after sensor data is sent to KISS-Sim, SolidWorks caches the end position (transformation matrix) of each game component (robots and game pieces), copies this data to the beginning position of each component, and restarts the calculation with new motor data. Some particularly interesting hacking was necessary to make this work, because while API calls can read and write the component positions, the new positions are not applied until the user manually clicks a button. A workaround hack was to use GlovePIE (a game enhancer which allows scripting of input devices) to generate mouse clicks as necessary. The user presses the right Ctrl key to trigger GlovePIE to simulate mouse clicks to apply the new positions and begin another segment. Not quite the prettiest hack in the world, but as Jeremy does game enhancement research [2], we enjoyed applying our game enhancement knowledge to robotics.

17 Guide to Operation

A quick guide to operating ImperiSim 2.0 is included in this paper. Operation is quite similar to ImperiSim 1.0 [1], so this guide will only cover major differences.

1. On the KISS-C side, nothing is different from ImperiSim 1.0. Refer to the 1.0 paper for information on setting up KISS-C to use ImperiSim. However, you will be including ImperiSimRT.h.

2. On the SolidWorks side, motors and physics are identical to ImperiSim 1.0; refer to the 1.0 paper for information on setting up motors and physics.
3. The SolidWorks option: “Tools --> Options --> External References --> Update component names when documents are replaced” must be unchecked. This is because you will need to rename subassemblies.
4. Large Assembly Mode must be disabled from the Tools menu, and the option “Tools --> Options --> Assemblies --> Use Large Assembly Mode to improve performance” must be disabled as well. This is because Large Assembly Mode causes collision detection to fail.
5. Restart SolidWorks after changing these setting for the first time, before continuing to run ImperiSim 2.0. (Otherwise collision detection might fail.)
6. To use sensors from SolidWorks’s physics system instead of KISS-Sim, name the component forming the sensor according to Section 7. Each robot should be a subassembly, with the robot subassembly having the name RTBot1. (Additional robots will be named RTBot2, etc., once we get that working; see Section 15.)
    a. Rangefinders and cameras require that one of the collision assemblies (included with ImperiSim 2.0) be used. These should be mated to the robot such that the collision assembly coincides with the rangefinder beam or the camera field of vision. The collision assembly should be named according to Section 7.
7. In SolidWorks, click Tools --> Macro --> Edit, and open ImperiSimRT.swp. Visual Basic for Applications will open.
8. Install GlovePIE [4], and open the script ImperiSimRT.PIE.
9. Click the Run button in GlovePIE. You can now minimize GlovePIE.
10. Return to SolidWorks and VBA, and make sure that both windows are visible on your desktop. (Dual-monitor setups are useful, though not necessary.)
11. Make sure that your speakers are enabled.
12. Press the right Shift button on your keyboard.
13. GlovePIE will ask you to move your mouse pointer to the Update Key button in SolidWorks’s Motion Study timeline (it’s a green diamond with a plus), and then press the right Shift button. Do so.
14. GlovePIE will then ask you to move your mouse pointer to the Run Sub button in VBA (it looks like a Play symbol), and then press the right Shift button. Do so.
15. GlovePIE will announce that the mouse calibration is complete. If you made a mistake, you can press the right Shift button to repeat the calibration.
16. Begin your KISS-C program in KISS-Sim.
17. Move your robot to the appropriate starting position in SolidWorks, and make sure that the Motion Study duration is as short as the temporal resolution you want (0.5 to 0.75 seconds works well).
18. Press the right Ctrl button on your keyboard to begin the simulation.
19. The first simulation segment will begin. Wait for the simulation to render in SolidWorks, and for the mouse pointer to stop showing that SolidWorks is busy. At the end, the robot and any important game pieces will be highlighted.
20. If you want to save the simulation step for later, you can take a screenshot now. (See Section 16.)
21. Now press the right Ctrl button on your keyboard again. Another segment will commence.
22. Continue pressing the right Ctrl button until you’re finished with the simulation.

Feel free to contact us via the Botball Community [3] with further questions if this guide is insufficient.

18 Multiple Robots -- Not Quite Yet

ImperiSim 1.0 supported multiple robots, but we haven't yet had a chance to integrate that feature with the segmentation and Named Pipe synchronization of ImperiSim 2.0. So for now at least, multiple robots are not supported by ImperiSim 2.0 (we hope to fix this soon).

19 Recording a Simulation Session

Unfortunately, recording a full simulation session is more difficult in ImperiSim 2.0 than in 1.0; this is because the segments are wiped after a new segment is formed. We're currently investigating what it would take to save some data about each segment to disk, so that the simulation could be reconstructed. For the moment, the solution is to manually save the session as an AVI file or series of screenshots before authorizing GlovePIE to begin the next segment.

20 Conclusion

ImperiSim 2.0 will be released on the Botball Community [3], and support will be offered there as well. If you'd like to contact us, please stop by the Community and say hello. Happy Simulating!

References

[1] J. Rand. ImperiSim: Integrating SolidWorks and KISS-Sim for Advanced Robot Simulation.
[2] J. Rand and M. Michelsen. GCARS-CS and GeckoTunnel: Using Video Game Enhancement Technology to Enable Online Play of Modern Offline Multiplayer Games. University of Oklahoma Undergraduate Research Day Conference. http://www.youtube.com/watch?v=1M3lCFOs7cs , March 2011.
[3] Botball Youth Advisory Council. Botball Community. http://community.botball.org , June 2011.
[4] C. Kenner. GlovePIE - Programmable Input Emulator. http://glovepie.org/ , November 2010.