Jeremy Rand
Norman Advanced Botball
biolizard89@gmail.com

ImperiSim: Integrating SolidWorks and KISS-Sim for Advanced Robot Simulation Part 2

10 Welcome to Part 2!

Welcome back. Hopefully you've already read Part 1 by now (if not, what are you doing here at Part 2?). So far we've covered the basics of simulation and how to export CSV data. In Part 2, we'll discuss the remaining CSV exporting, loading the data into SolidWorks and other fun features.

11 Logging Servo Positions

Servo positions are somewhat harder than Motor positions because your program doesn't know where the servos are (it only knows their goal positions, and it may take several seconds for the servos to reach their goals). In addition, servo motion is absolute, while DC motor motion is relative. As a result, the code for servos is significantly more complex than that for DC motors.

This code goes into the while loop:

```
// For each servo
for(servo_count=0; servo_count<4; servo_count++)
{
    // Move the servo slowly rather than all at once, since that's what it
    // will do in real life
    if(abs(get_servo_position(servo_count) -
    servo_positions[servo_count]) < imperisim_servo_step[servo_count])
        servo_positions[servo_count] = get_servo_position(servo_count);
    else
        servo_positions[servo_count] += imperisim_servo_step[servo_count] *
        sign(get_servo_position(servo_count) - servo_positions[servo_count]);

    // Scale positions - similar to DC motors, but relative to the starting
    // position instead of 0 ticks
    servo_positions_scaled[motor_count] = (servo_positions[servo_count] -
    start_servo_positions[servo_count]) * 360.0 /
```

```
    imperisim_servo_ticks_per_revolution[servo_count];
}

// Output servo positions
fprintf(outFileS0, "%f, %f\n", (time), servo_positions_scaled[0]);
fprintf(outFileS1, "%f, %f\n", (time), servo_positions_scaled[1]);
fprintf(outFileS2, "%f, %f\n", (time), servo_positions_scaled[2]);
fprintf(outFileS3, "%f, %f\n", (time), servo_positions_scaled[3]);
```

These two lines of code go under the similar line for motors (before the `imperisim()` function):

```
int imperisim_servo_ticks_per_revolution[] = {4094, 4094, 4094, 4094};
int imperisim_servo_step[] = {186, 186, 186, 186};
```

The `sign()` function is in the source code included with this paper. 4094 is an approximation based on 2047 tick range for servos and an estimated angular range of π radians (0.5 revolutions). 186 is based on a black DC gear motor (with the same power as a servo) turning at 1000 ticks per second. 1000 motor ticks/s * revolution/1100 motor ticks * 4094 servo ticks/ revolution = 3722 servo ticks/s. Multiplying this by 50ms=0.05s (the period for the while loop) yields 186 servo ticks per step. As with DC motors, it is best to measure these yourself due to natural variance between servos (particularly the step, which will lower with increasing load). Both variables should be set prior to starting the `imperisim()` thread.

Now we can dump servo positions to CSV files as the program runs.

12 Logging Robot Coordinates

While it is theoretically possible to rig SolidWorks to simulate robot locomotion simply based on the DC motor values, it is easier to simply log that data from the `imperisim()` function as well. This also allows us to use the iRobot Create chassis [5]. Be warned, there is a lot of math here. An excellent discussion of the math behind robot locomotion is at The Rossum Project [6].

```
if(imperisim_create) // Create chassis
{
    trans = gc_distance;
    rotate = gc_total_angle;
}
else // Differentially steered, DC motor driven chassis
{
    trans = motor_positions_scaled[imperisim_rdrive] +
        motor_positions_scaled[imperisim_ldrive];
    rotate = motor_positions_scaled[imperisim_rdrive] -
        motor_positions_scaled[imperisim_ldrive];
```

```
}

// Scale from ticks
trans /= imperisim_trans_scale;
rotate /= imperisim_rotate_scale;

// Compute how far we moved since the last loop
delta_trans = trans - old_trans;
delta_rotate = trans - old_rotate;

// Trig approximation of XY coordinates based
// on translation and rotation axes
if(delta_trans > 1.0 || delta_trans < -1.0)
{
    x += (delta_trans * cos(rotate*6.283/360.0));
    y += (delta_trans * sin(rotate*6.283/360.0));
}

// Store values for the next loop
old_x = x;
old_y = y;
old_trans = trans;
old_rotate = rotate;

// Output locomotion variables
fprintf(outFileX, "%f, %f\n", (time), x);
fprintf(outFileY, "%f, %f\n", (time), y);
fprintf(outFileTheta, "%f, %f\n", (time), rotate);
```

We've added several global variables here:

```
int imperisim_create = 0;
int imperisim_rdrive = 0;
int imperisim_ldrive = 1;
float imperisim_trans_scale = 1000.0;
float imperisim_rotate_scale = 180.0/
3.14159265358979323846264338327950288419 71;
```

The globals should be set as follows prior to starting the imperisim() thread:

- `imperisim_create` should be 1 if using the Create chassis and 0 otherwise.
- `imperisim_rdrive` should be the port number of the right drive wheel if not using the Create.

- `imperisim_ldrive` should be the port number of the left drive wheel if not using the Create.
- If using the Create, `imperisim_trans_scale` should be the number of Create encoder-measured millimeters in a meter of forward movement; otherwise, it should be the number of wheel rotations (sum of right and left) in a meter of forward movement.
- If using the Create, `imperisim_rotate_scale` should be the number of Create encoder-measured degrees in a radian of angular movement; otherwise, it should be the number of wheel rotations (sum of right and left) in a radian of angular movement.

That's all the CSV data we need to dump for now. Ready to load it into SolidWorks?

13 CSV Data Import in SolidWorks

SolidWorks makes CSV import quite easy. First, open the Motion Study, and make sure that all the motors are turned on for the first 135 seconds of the Motion Study, and turned off afterwards. (If you're not simulating an entire Botball 2010 International game, you may wish to use a different time period.) Next, look in the list of components on the left-hand side. Right-click the Motor for which you want to import data, and click Edit Feature. Click the drop-down menu under Motion, and click Interpolated. A second drop-down will appear, allowing you to choose between Displacement, Velocity, and Acceleration. Click Displacement. Interpolation Type doesn't make much difference for this purpose as far as I can tell. To load the data, click Load From File, and navigate to the CSV file for that Motor. There may be a brief delay while SolidWorks imports the data. Finally, click on the green checkmark to finalize the Motor.

Simply do this for each motor which your robot uses, and then click the Calculate button in the Motion Study. The Motion Study will usually calculate fairly quickly, although the speed of your PC is a major factor. Once it's calculated, click the Play button, and watch the animation!

14 Automation using VBA

Manual CSV import in SolidWorks can be a pain, especially when done repeatedly. Luckily, SolidWorks offers an API to automate tasks using the Visual Basic for Applications (VBA) scripting language [7]. For documentation on the SolidWorks API, open SolidWorks and click Help → API Help. We're going to use the `AssemblyDoc::IFeatureByName` function to access the Motors, `SimulationMotorFeatureData::InterpolatedMotor` to set the motors to use CSV data, and `SimulationMotorFeatureData::LoadSplineData` to load the CSV file.

The `ImperiSim.swp` VBA macro included in this paper's companion .zip file does what we want here. To use it, open the Motion Study in SolidWorks, and click Tools → Macro → Run. Navigate to and choose the ImperiSim.swp file included with this paper. It will prompt for a directory where the CSV data was dumped, ask what Bot Number the CSV data should be assigned to (this is usually 1 unless multiple robots are being used), and load all of the data into

the Motors. It even shows its progress (and any errors) in SolidWorks's Status Bar. You can click Tools → Macro → Edit to look at the source code (it's fairly straightforward).

15 More Physics with Basic Motion

Basic Motion supports physics other than Motors. Springs, Contact, and Gravity are also available. These can be added by clicking their associated buttons in the Motion Study timeline (immediately to the right of the Motor button). Springs can be used to simulate rubber bands. Contact can be used to allow a robot to bulldoze or grab game pieces without simply passing through them. And Gravity allows projectiles (both from your entry and game pieces) to be simulated. Keep in mind that additional physics elements will reduce the speed of your calculations.

16 Multiple Robots

Simulating multiple robots can be achieved by inserting the additional bot(s) into the assembly, and creating their motors with a different Bot Number (e.g. the motors for the second robot would be named Bot2X, Bot2Y, etc.). Enter the corresponding Bot Number into the VBA script when loading their CSV files.

17 Importing Data from a Physical Robot

Sometimes KISS-Sim's physics aren't accurate enough to determine how your robot's motors will behave. In these cases, you may wish to record CSV data from a physical robot on the actual game board. Doing this is easy. In your program, add the following line prior to calling `start_process(imperisim)`:

```
system("cd /mnt/browser/usb/");
```

Before running your program on the physical robot, insert a flash drive into the CBC's USB port, and mount it from the CBC Firmware's GUI. Now, when you run the program, the CSV files will be dumped to the flash drive's root directory. When the program has ended, unmount the flash drive, and insert it into your PC to access the CSV files. Run the VBA script as usual, providing the flash drive's root directory when asked.

If you don't want to move a flash drive back and forth, you're running the NHS Patchset [8], and your CBC has network access, you can replace the `system()` command above with the following:

```
system("mkdir /tmp/imperisim");
system("cd /tmp/imperisim/");
```

This stores the files in the Chumby's RAM instead of a flash drive. To retrieve the data, open a Cygwin or Linux bash prompt on your PC, navigate with `cd` to where you want the files to go, and execute the following command, where `<CBCIP>` is your CBC's IP address:

```
scp -C -r root@<CBCIP>:/tmp/imperisim ./
```

You can rerun the robot's program, and then rerun the `scp` command to retrieve updated data.

18 Optimizing Calculation Times

If the Motion Study is calculating slower than you'd like, try clicking on the Motion Study Properties (right-most icon on toolbar above the timeline). You can lower the Basic Motion framerate and the Geometry Accuracy and 3D Contact Resolution levels. I've found that setting the Geometry Accuracy and 3D Contact Resolution to the minimum and the framerate to 1 often still yields acceptable results, and is much faster.

19 Why No Sensor Feedback?

You may have noticed that we're only linking the motors to SolidWorks; sensors stay within KISS-Sim. I am investigating methods of linking sensors as well, but suffice to say that it is a lot harder, and probably also a good bit slower. However, I believe it is doable. If I get sensors to work, expect a paper at GCER 2011 on the subject.

20 Conclusion

ImperiSim is still very new, and I'm sure that a hacker with more SolidWorks API and VBA expertise could find many aspects to improve. Still, I think ImperiSim will encourage more Botball students to use SolidWorks, which was my original goal.

A few people deserve thanks as well: Matthew Oelke of KIPR for modeling the Botball parts in SolidWorks and introducing me to SolidWorks; Jeffrey Hsu of Nease High School for giving me SolidWorks advice and letting me bounce ideas off him during the development of ImperiSim; Dassault Systèmes SolidWorks API Support staff for answering many questions about the details of the SolidWorks API; and finally, Dr. David Miller of KIPR for inspiring the project by telling me at GCER 2009 that he had tried to integrate SolidWorks and KISS-Sim but hadn't figured out how to make it work -- I always like a good challenge!

If you've done something interesting regarding SolidWorks simulation or improvements to KISS-Sim, or if you have suggestions for ImperiSim, I'd like to hear about it! Contact me on the Botball Forums or the Botballer's Chat on the Botball Community [9].

References

[5] iRobot. iRobot Create Programmable Robot. http://store.irobot.com/product/index.jsp?productId=2586252&cp=3311368.2591511&parentPage=family, June 2010.
[6] G.W. Lucas. A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators. http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html, January 2006.
[7] Wikipedia contributors. Visual Basic for Applications. http://en.wikipedia.org/wiki/Visual_Basic_for_Applications, June 2010.
[8] J. Rand, M. Thompson, B. McDorman. CBC Hacking 2010. Proceedings of the 2010 Global Conference on Educational Robotics, July 2010.
[9] Botball Youth Advisory Council. Botball Community. http://community.botball.org, May 2010.