Jeremy Rand
Norman Advanced Botball
biolizard89@gmail.com

# ImperiSim: Integrating SolidWorks and KISS-Sim for Advanced Robot Simulation Part 1

## 1 Introduction to ImperiSim

Dassault Systèmes SolidWorks Corporation has sponsored Botball since 2007, providing free licenses of the SolidWorks Student Design Kit CAD software [1] to all teams. SolidWorks is a powerful piece of software, allowing students to construct 3D models of their robots and how they move. SolidWorks can be highly useful in testing designs prior to construction or when a physical kit isn't available, but many teams complain that SolidWorks doesn't deliver enough capability to justify spending the time to learn its interface or to actually build the CAD models. ImperiSim's goal is to increase adoption of SolidWorks by Botball teams, by integrating SolidWorks with a tool which many teams already use, KISS-Sim. The result is a robot simulator which simulates both the hardware and software components of the robot, in a relatively user-friendly manner. As prerequisites, readers should already have basic familiarity with SolidWorks and KISS-C as applied to Botball robots.

## 2 Introduction to KISS-Sim

KISS-Sim is a tool (included with KISS-C [2]) created by KISS Institute for Practical Robotics which simulates the software of robots on a PC. It is a quite high-level simulator: rather than emulating the Chumby and CBOB hardware and running those processors' actual firmwares, KISS-Sim simply links the robot code with different libraries targeted to the PC platform. KISS-Sim contains some graphical display and some physics simulation, but they are limited to 2D and only simulate the physics of locomotion; additional effectors such as arms are not available. ImperiSim relies on KISS-Sim for the software simulation, as well as simulating hardware components such as sensors, which would be unwieldy in SolidWorks.

## 3 Introduction to SolidWorks Motion Studies

SolidWorks includes a feature called Motion Studies, which allows the motion of assemblies such as robots to be simulated in full 3D with physics such as motors, springs, contact, and gravity. Motion Studies offer a lot of control over the simulated hardware, but the software simulation is very primitive: most Botball teams drag and drop their bot's effectors to the expected locations within the SolidWorks interface, since SolidWorks cannot run a KISS-C

program to find the locations itself. ImperiSim relies on a Motion Study for hardware simulation.

# 4 Animation, Basic Motion, and Motion Analysis

SolidWorks Motion Studies can use three different solvers to calculate motion: Animation, Basic Motion, and Motion Analysis. Animation is usually used to generate simple videos; it is very fast, but does no physics calculation, instead using interpolation between keyframes to calculate where components move. Because Animation is so fast and simple, it is commonly used for simple Motion Studies, but has some disadvantages as well. Animation cannot handle collision detection, gravity, or springlike components such as rubber bands. When such features are necessary for a Motion Study, Basic Motion is preferable. Basic Motion is slower than Animation, but still relatively fast, and can handle collisions, gravity, and springs. It cannot, however, handle friction. When friction analysis is necessary, Motion Analysis is usable. Motion Analysis is very accurate and handles friction, forces, and dampers, but is much, much slower than Basic Motion. Motion Analysis is also not available in the SolidWorks Student Design Kit which is given to Botball teams; a purchase of SolidWorks Premium (around $100 for an educational license) is necessary.

So what solver should be used in ImperiSim? Because of Motion Analysis's slowness and extra monetary cost, I do not recommend it. One might expect Animation to work well for simple ImperiSim simulations, but a problem arises with Animation. Because Animation solves the entire simulation at once, rather than sequentially, it assumes that the axis of rotation for the robot is unmoving. This means that rather than your robot turning around its center, it will turn around wherever its center was in the robot's starting position. (If you know of a workaround for this, please let me know!) Basic Motion solves the simulation sequentially, so it allows the axis of rotation to move. As a result, I recommend Basic Motion for the vast majority of ImperiSim simulations.

For more information on the differences between the Motion Study solvers, see the SolidWorks website [3].

# 5 The Botball Game Board in a Motion Study

KIPR distributes a SolidWorks model of the Botball game board at the workshop. To use the game board in a Motion Study, copy the game board file to a new `.sldasm` file, open it in SolidWorks, and click Insert → New Motion Study. Next, go back to the Model tab at the bottom of your SolidWorks window (leaving the Motion Study) and click Insert → Component → Existing Part/Assembly. Select the SolidWorks model of your robot, and place it somewhere near the game board. Right-click the robot in the tree on the left-hand side, and click the Component Properties icon. By default, the robot will be set to Solve as Rigid; change it to Solve as Flexible; otherwise your robots' motors will be unable to move.

You'll notice that your robot is not directly on the game board; in my tests the robot usually ends up below the board. The easiest way to place your robot on the board is to mate the bottom of the robot to the surface of the game board. Mating wheels to the game board surface can be tricky; newbies to SolidWorks should expect to expend some effort on this. At one point when I got frustrated, I simply mated the bottom plate of the Create to the board surface, allowing the wheels to stick through the board by a centimeter or so. Readers who go with this simpler method should be aware that the height of their robot will be offset, and that any vertical arms may be thrown off by the height difference.

Finally, make certain that your robot is facing perpendicular to one of the game board axes. This will make the locomotion simulation much easier (the robot will be able to face any direction once the simulation begins.)

# 6 Adding Motors

Now that your robot is present in the model, click on the tab at the bottom for the Motion Study that you added (KIPR usually already includes a Motion Study which shows the camera panning around the board, so your Motion Study will likely be called "Motion Study 2"). If the solver is set to Animation or Motion Analysis, change it to Basic Motion. We're going to create three Motors to simulate the robot's 2D locomotion along the ground (X, Y, and Theta), as well as the 4 DC motors and 4 RC servos which the CBC contains. At the bottom of your screen will be a Motion Study timeline. Click the Motor button to open the Motor creation dialog. For Motor Type, choose Linear Motor. For Component/Direction, click a component of the robot which corresponds to the X axis of the game board. Finally, for the Move Relative to Another option, click a PVC pipe which is parallel to the X axis of the game board. You can leave the Motion options at their default (Constant Speed). Click the green checkmark, and the Motor should have been created. Rename the Motor (it will be on the component list as "Linear Motor 1") to "Bot1X".

Now that you've created the X Motor, do the same thing for the Y Motor. All steps are the same; simply click a component and PVC pipe which correspond to the Y axis instead of the X axis. Name this one "Bot1Y".

Next is the Theta Motor. The steps are similar, except choose the Rotary Motor type instead of Linear Motor, and choose a circular component whose center is near the center of your robot. Don't choose anything for Move Relative to Another. Name it "Bot1Theta".

We now have locomotion working. To test the Motors, you can enable and disable them on the timeline, click Calculate, and watch the robot move by clicking Play.

Finally, create Rotary Motors for each of the DC motors and RC servos on the robot, naming the DC motors "Bot1M0" through "Bot1M3" and the RC servos "Bot1S0" through "Bot1S3", as they

correspond to the motor/servo ports on the CBC. Feel free to enable/disable the motors on the timeline to verify that the robot's effectors move as they should.

# 7 CSV Data Export in KISS-Sim

We're now going to add a library to KISS-Sim which allows it to export motor motion in CSV (Comma-Separated Value) format [4]. CSV is a common, very simple format for storing tables of data. In our case, we want a table that relates time and the motors' positions. The full source code is included in this paper's companion .zip file. Let's start with this function which will run as a separate thread:

```
#include "math.h"

int imperisim_active = 0;

void imperisim()
{
    // For DC motor simulation
    float motor_positions_scaled[] = {0.0, 0.0, 0.0, 0.0};

    // For servo simulation
    int servo_count;
    int start_servo_positions[] = {get_servo_position(0),
        get_servo_position(1), get_servo_position(2),
        get_servo_position(3)};
    int servo_positions[] = {get_servo_position(0),
        get_servo_position(1), get_servo_position(2),
        get_servo_position(3)};
    float servo_positions_scaled[] = {0.0, 0.0, 0.0, 0.0};

    // For dead-reckoning
    static double x, y, trans, rotate, old_x, old_y, old_trans,
        old_rotate;
    double delta_trans, delta_rotate;

    // Keep track of time
    double start_time = seconds();
    double time = 0.0;

    // Used in the while loop below
    imperisim_active = 1;

    // Now we open the data files for writing
```

```
    FILE *outFileM0 = fopen("BotM0.csv", "w");
    FILE *outFileM1 = fopen("BotM1.csv", "w");
    FILE *outFileM2 = fopen("BotM2.csv", "w");
    FILE *outFileM3 = fopen("BotM3.csv", "w");

    FILE *outFileS0 = fopen("BotS0.csv", "w");
    FILE *outFileS1 = fopen("BotS1.csv", "w");
    FILE *outFileS2 = fopen("BotS2.csv", "w");
    FILE *outFileS3 = fopen("BotS3.csv", "w");

    FILE *outFileX = fopen("BotX.csv", "w");
    FILE *outFileY = fopen("BotY.csv", "w");
    FILE *outFileTheta = fopen("BotTheta.csv", "w");

    printf("ImperiSim Started\n");

    while(imperisim_active)
    {
     // keep track of time
        time = seconds() - start_time;

        // We're going to dump data here

        msleep(50);
    }

    // Robot is done; close the files.

    fclose (outFileM0);
    fclose (outFileM1);
    fclose (outFileM2);
    fclose (outFileM3);

    fclose (outFileS0);
    fclose (outFileS1);
    fclose (outFileS2);
    fclose (outFileS3);

    fclose (outFileX);
    fclose (outFileY);
    fclose (outFileTheta);

}
```

To use this code, simply copy it into the file `imperisim.h`, `#include` it in your robot's program, and add the following to your robot's code:

Right before your bot starts moving (probably immediately after `wait_for_light()` returns):

```
start_process(imperisim);
```

And right after your bot finishes moving (probably immediately after `run_for()` returns and you shut off motors):

```
imperisim_active = 0;

msleep(1000); // Wait for files to close
```

You'll notice that we don't log any data yet. That's in the next section!

# 8 Logging DC Motor Positions

By far the easiest motion to log is that of DC motors. Since both KISS-Sim and the CBC can sense the DC motors' position, simply read that position, multiply by 2π radians, divide by the ticks per revolution (thus converting ticks to radians, which SolidWorks uses), and output it to the files. Insert this code into that while loop:

```
// For each motor
for(motor_count=0; motor_count<4; motor_count++)
{
    motor_positions_scaled[motor_count] =
get_motor_position_counter(motor_count) * 2.0 *
3.14159265358979323846264338327950288419 71 /
imperisim_motor_ticks_per_revolution[motor_count];
}

// Output DC motor positions
fprintf(outFileM0, "%f, %f\n", (time), motor_positions_scaled[0]);
fprintf(outFileM1, "%f, %f\n", (time), motor_positions_scaled[1]);
fprintf(outFileM2, "%f, %f\n", (time), motor_positions_scaled[2]);
fprintf(outFileM3, "%f, %f\n", (time), motor_positions_scaled[3]);
```

You'll notice that we've used an array of values, `imperisim_ticks_per_revolution[]`, and a counter variable. Let's add these lines right under the declaration of `imperisim_active`:

```
int motor_count;
```

```
int imperisim_motor_ticks_per_revolution[] = {1100, 1100, 1100, 1100};
```

Your user program may wish to modify these values prior to starting the imperisim thread. 1100 is a fairly accurate estimate of the black gear motors from the 2010 Botball kit, but since individual motors vary, you are best off if you measure the values empirically for each motor on your robot.

Now that we've added this code, the motor positions will dump to CSV files when your program runs.

# 9 End of Part 1

That's it for Part 1. Part 2 will finish explaining how ImperiSim works and how to utilize it yourself. See you in Part 2!

# References


[1] Dassault Systèmes SolidWorks Corporation. SolidWorks 3D CAD Design Software. http://www.solidworks.com/, June 2010.
[2] KISS Institute for Practical Robotics. KISS-C. http://botball.org/kiss, December 2009.
[3] J. Galliera. Solvers used for Animation, Basic Motion & Motion Analysis. https://forum.solidworks.com/community/solidworks_simulation/motion_studies/blog/2010/02/08/solvers-used-for-animation-basic-motion-motion-analysis, February 2010.
[4] Wikipedia contributors. Comma-separated values. http://en.wikipedia.org/wiki/Comma-separated_values, May 2010.