sDotson.com

Autodesk Inventor Tutorials

by Sean Dotson
www.sdotson.com
sean@sdotson.com

Advanced Motion Part Two

Latest Revision: 5/14/02

This tutorial assumes that the user is familiar with basic constraints and how to drive those constrains. It also assumes that they are familiar with creating and editing parameters. It furthermore assumes that the reader has read the **Advanced Motion Part One** tutorial.

To see an .avi file of the final animation lesson click **here**. Now that we know what we want to do let's begin the lesson.

We begin with a basic assembly. The assembly consists of the checkerboard (made from two parts), checkers, timer body and two different timer hands. While not all of these parts are necessary for the lesson (many were added for aesthetic reasons) they are all provided in a zip file located **here**.

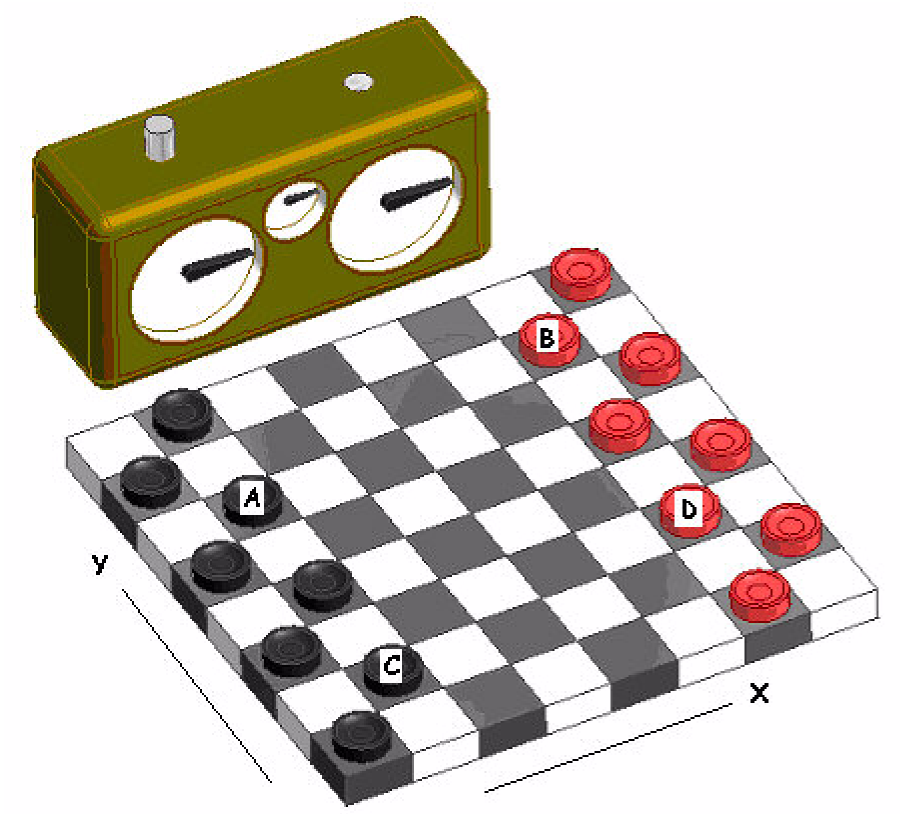

Figure 1 - Basic Constrained Assembly with Checkers Identified

Constrain the assembly as shown above. To constrain the checkers use a mate to attach them to the board then use the checkers' origin planes to constrain them to two sides of the board using flush constraints. Since we are not moving the back row of checkers I used an assembly array to place them however the checkers we want to move must be placed and constrained individually.

The timer body is placed and grounded and the hands are constrained via an angular constraint to a horizontal surface. We then rename the right hand's angular constraint (via the parameters menu) to **angle**.

The smaller middle hand is constrained to a horizontal surface and has the equation

```
angle * 2
```

This step is simply aesthetic and could be skipped.

We now can begin to create the equations needed to move the checkers. Using **angle** as our driven constraint we tie in the motion of Checker A (see Figure 1) to **angle**. Find the flush constraint that controls the motion in the X direction. Edit the equation to read:

```
0.750 in + min(( 0.100 ul * angle * 1.000 in / 1.00 deg );0.5 in)
```

Breaking down this equation reveals that the 0.750in value is the initial displacement of the checker. (Since the squares are 0.5" square and it is in the second row (.5)+(0.5/2)=0.75) We then add to this the minimum value of (0.1*angle) or 0.5in. This means the checker will move until the value of **angle** reaches 5 deg (0.1*5=0.5)

NOTE: Be sure to keep units consistent. Also depending on what surface off of which you initially constrained the checker, the initial value (0.75 in our case) may be different. Furthermore the values may be negative. If this is the case simply switch the signs in your equations.

Now let's look at the motion in the Y direction. Edit this flush constraint to read:

```
2.750 in + min(( 0.100 ul * angle * 1.000 in / 1.00 deg );0.5 in)
```

Again 2.750in is our initial displacement (I constrained the checker off the near side of the board) and again we are adding to that the minimum value of (0.1***angle**) or 0.5in

Now drive **angle** from 0 to 100 deg. The checker should move diagonally one square in the positive X direction and one square in the positive Y direction. (see Figure 2)

We can now work on moving checker B. Again edit the flush constraint parameter to read:

```
0.750 in + min(max(( 0.100 ul * ( angle - 10 deg ) * 1.000 in / 1.00
                    deg );0 in);0.5 in)
```

for the X direction and

```
0.750 in + min(max(( 0.100 ul * ( angle - 10.00 deg ) * 1.000 in / 1.00
                  deg );0.000 in);0.500 in)
```

for the Y direction.

Here you will notice that the equations are a bit more complex. Let's first look at the equation for the X direction. As usual we have the initial displacement of 0.750in. We then have a nested min/max function. Let's look at the inner function first.

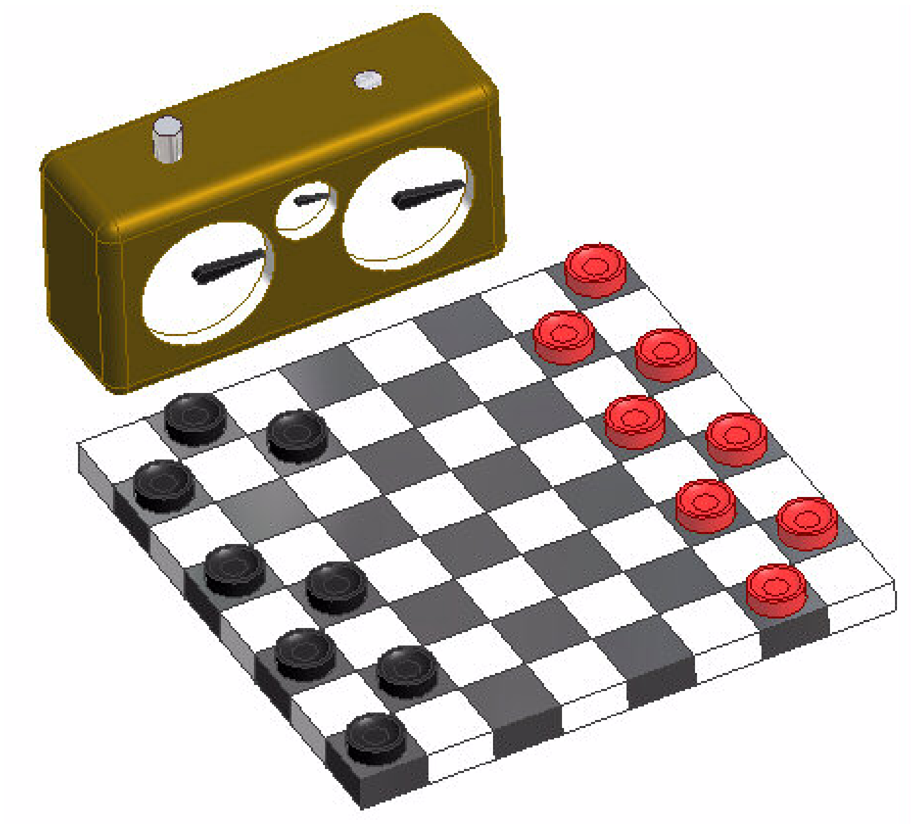

Figure 2 - Checker "A" Moved into Position

```
    max(( 0.100 ul * ( angle - 10 deg ) * 1.000 in / 1.00 deg );0 in
```

This function returns the maximum value of either 0.1*(**angle**-10) or 0 in. This means the return value will be 0 for **angle**<10 degrees since **angle**-10 is negative for all values less than 10 degrees. So our checker will not move until **angle** reaches 10 degrees.

NOTE: Since the previous checker stops moving at 5 degrees we choose the 10 degree value in this equation to prove 5 degrees of "dwell" time. We could have chosen any value for the dwell. When creating animations such as this, a pause between moves makes the motion more realistic. Your actual design will dictate the real dwell time.

Now we go up one level in the equation structure and examine the min function.

```
    min(max(( 0.100 ul * ( angle - 10.00 deg ) * 1.000 in / 1.00 deg
                          );0.000 in);0.500 in)
```

This takes the minimum value of the nested equation (which will range from 0 to (0.1*(angle-10)) or 0.5in. This means that the motion of the checker will stop when **angle** becomes greater than or equal to 15 degrees.

The same logic holds true for the motion in the Y direction. The readers are left to prove this to themselves. (After reading countless engineering textbooks with this phrase I always wanted to use it. Sorry, couldn't resist. ☺)

Again drive **angle** to 100 degrees. You should see the motion of the first checker followed by the motion of the second checker.

The technique of nesting min and max functions is the basic method to get movement of a part to start at a determined time and stop at another.

The motion of checkers C & D are achieved in the same manner. Remember that the dwell angle must be set properly to make the movement consecutive. Below I have listed the equations for checkers C & D

Checker C:

X direction:

```
 0.750 in + min(max(( 0.100 ul * ( angle - 20.00 deg ) * 1.000 in / 1.00
                        deg );0.000 in);0.500 in)
```

Y direction:

```
 0.750 in + min(max(( 0.100 ul * ( angle - 20.00 deg ) * 1.000 in / 1.00
                        deg );0.000 in);0.500 in)
```

Checker D:

X direction:

```
 0.750 in + min(max(( 0.100 ul * ( angle - 30.00 deg ) * 1.000 in / 1.00
                        deg );0.000 in);0.500 in)
```

Y direction:

```
   -1.250 in - min(max(( 0.100 ul * ( angle - 30.00 deg ) * 1.000 in /
                      1.00 deg );0.000 in);0.500 in)
```

Note that in the Y equation for checker D that the initial value is negative. Since this is the case we subtract the rest of the equation from this value to get the correct motion. You will have to use your own judgment to determine if the value of the min/max equations should be added to or subtracted from the initial value.

It is also often useful to construct a graph of the desired movement before you begin. See Figure 3 for an example. For this example you would also want to construct one for the Y direction.

Drive **angle** from 0 to 100 degrees. You should have each checker move one square in succession. If you watch the AVI video however you will notice that checker A moves once (then the others move), then it moves once again. For this motion we will need to edit the constraint equatin. Find the parameter for the X and Y directions and edit them to read as follows.

X direction:

```
  0.750 in + min(( 0.100 ul * angle * 1.000 in / 1.00 deg );0.500 in) +
                   sign(angle - 40.00 deg) * movetwox
```

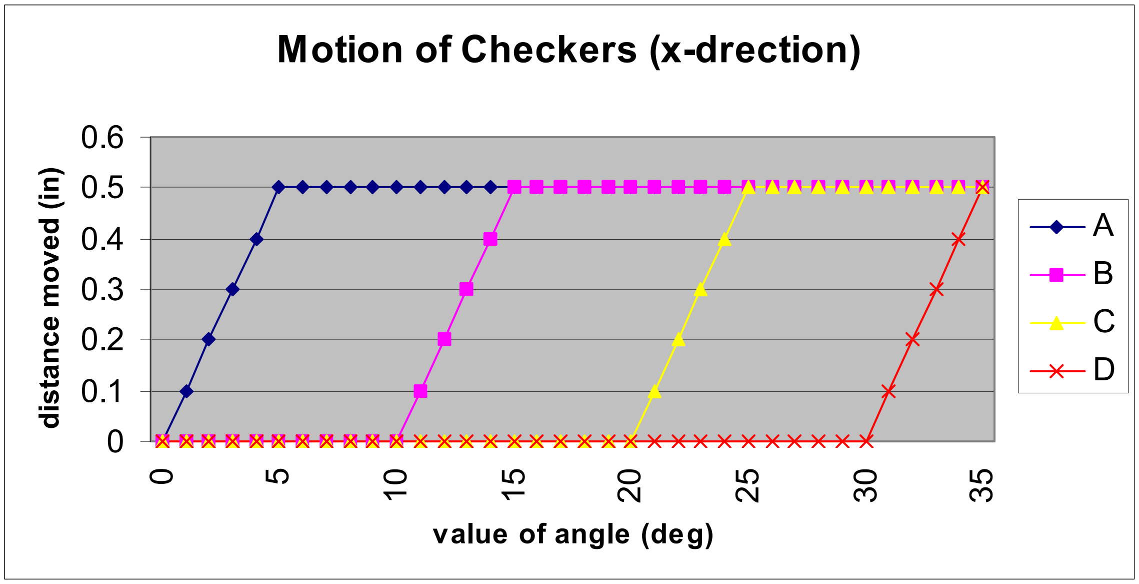

Figure 3 – Chart detailing the motion of checkers (X direction)

Y direction:

```
 2.750 in + min(( 0.100 ul * angle * 1.000 in / 1.00 deg );0.500 in) -
                    sign(angle - 40.00 deg) * movetwoy
```

These are basically the same equations but with the addition (and subtraction) of the term(s):

`sign(angle - 40.00 deg) * movetwox` and

`sign(angle - 40.00 deg) * movetwoy`

movetwox and **movetwoy** are user parameters that we added in parameters dialogue box. Got to the bottom of the parameters dialogue (where you will see user parameters) and click Add.

Enter the values of **movetwox** and **movetwoy** as shown below (units are in inches):

```
movetwox = min(max(( 0.100 ul * ( angle - 40 deg ) * 1.000 in / 1.00
                    deg );0 in);0.5 in)

movetwoy = min(max(( 0.100 ul * ( angle - 40 deg ) * 1.000 in / 1.00
                    deg );0 in);0.5 in)
```

We placed these in the user parameters area so that we did not have to type out the entire formula in the upper parameters box. By breaking down the function into sub functions and calling their names it simplifies the process of writing the equations. We could have written the full equation as:

```
2.750 in + min(( 0.100 ul * angle * 1.000 in / 1.00 deg );0.500 in) -
 sign(angle - 40.00 deg) * min(max(( 0.100 ul * ( angle - 40 deg ) *
                1.000 in / 1.00 deg );0 in);0.5 in)
```

But this gets rather messy.

So what have we done? We know that the first part of the equation moves checker A one square in the x and y directions but what does the

```
sign(angle - 40.00 deg) * movetwox
```

part accomplish?

The sign function returns a binary value depending on the value of its argument. If the argument is positive it returns a 1, if negative a zero. In this example we are using the sign function as a primitive if..then function. If the value of **angle-40** is positive (i.e. **angle**>40) then sign returns 1 and we then add (or subtract) the value of **movetwox** to the first part of the equation. If it is negative (**angle**<40) then sign retuns 0 and we do not add the value of **movetwox** (since 0***movetwox** = 0). Similar logic follows for the Y direction equation.

Now let's look at what **movetwox** does. Again **movetwox** is:

```
movetwox = min(max(( 0.100 ul * ( angle - 40 deg ) * 1.000 in / 1.00
                          deg );0 in);0.5 in)
```

movetwox returns the minimum of either 0.5in or the maximum of either (0.1*(angle-40)) or 0in). This nested function is similar to the one described in the movement of checker B.

So when angle<40 the max function returns 0 and hence the min equation returns 0 as well.

When 45>angle>40 the max function returns the value of (0.1*(angle-40)) and hence the min function returns the value of (0.1*(angle-40)).

When angle>45 the max function returns (0.1*(angle-40)) and hence the min function returns 0.5in

Again a chart can be helpful in planning the motion of the parts. (see Figure 4).

Now drive **angle** from 0 to 100 degrees. You should have motion similar to that as shown in the **AVI example.**

I encourage the readers to edit the equations to see get the checkers to make different movements. For example make checker B's move be in the positive Y direction instead of the negative Y direction. Then try to get checker B, C or D to make another move (using the example of checker A's movement).

Multiple part, multiple motion can get tricky but if you lay out the equations in a good format before you begin you will have no problems performing very advanced motion simulations in Inventor.

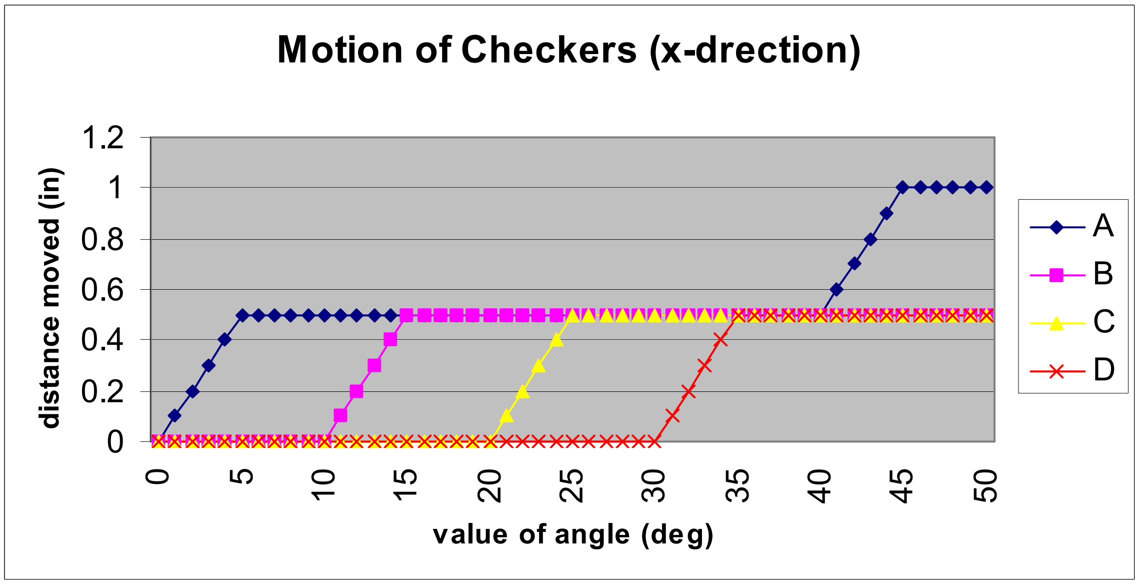

Figure 4 - Chart detailing the final motion of checkers (X direction)