# Macro: crank_rocker.mac

| | |
|---|---|
| Macro | crank_rocker.mac |
| Description | Demonstrates four bar mechanism (as a crank-rocker) |
| CM version | Any |
| See also | macro: crank_rocker_dsp.mac |

## What the macro does

This macro creates a “stick diagram” of a four bar mechanism. This means that the links are represented by simple lines. The crank and driven links are just a single line each; the coupler link is here formed by three lines in a triangle. These links are shown on the left in figure 1. Here the links are assembled correctly by the modeller.

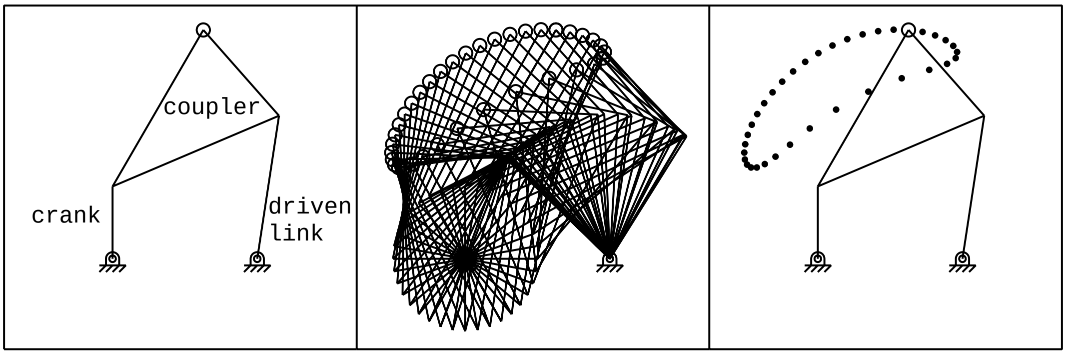


Figure 1: Crank-rocker (four bar) mechanism

In the middle of figure 1, the crank has been rotated in a number of steps. At each stage, the other two links are reassembled with the crank and hence a simulation of the motion is obtained.

There is a point (shown as a small circle) at the free vertex of the coupler triangle. The position of this point is tracked as the mechanism moves. This track is shown in the third part of the figure. This is an example of a possible output motion from the mechanism.

## How the main part of the macro works

The listing of the macro is given below. The lines of the macro are numbered for ease of reference.

A number of geometric objects are used.

| | |
|---|---|
| `p1` | first fixed pivot point |
| `p2` | second fixed pivot point |
| `l1` | line to represent the crank |
| `l2` | line to represent the main part of the coupler, it goes between the joints with the crank and with the driven link |
| `l2a` | line for one side of the coupler triangle |
| `l2b` | line for the other side of the coupler triangle |
| `ptip` | point at the tip of the coupler triangle |
| `l3` | line to represent the driven link |

These objects are declared as global variables at the start of the macro (lines 0009 and 0010) and are defined in function `setup` (lines 0030-0039). The definition is in terms of real variables declared in line 0008 and given values in lines 0014–0019. Each line goes between two end-points; for example, the first end-point of line `l1` is denoted by `l1:e1`, and the second end-point by `l1:e2`.

Each line is embedded in a model space. A model space is essentially a transformation with which a number of geometric entities can be associated. If the transform changes, then the entities move together.

Three model spaces are used. These are declared in line 0012 and are defined in the `setup` function in lines 0027–0029. The translation components of each model space are initially zero (first two argument in each use of function `mod2`). The rotation angles are set to non-zero values: this is mainly for convenience here so that the objects appear rotated on the screen and so are easier to identify.

Note that models space `m2` is embedded into model space `m1` by including `m1` as the fourth argument in line 0028. This means that if `m1` moves, the `m2` moves with it. A simple hierarchy of model spaces has been constructed as shown in figure 2.

In the definitions of the geometric objects in the `setup` function, it is seen that the lines and the point `ptip` are each embedded in an appropriate space (`l1` in `m1`, and so on).

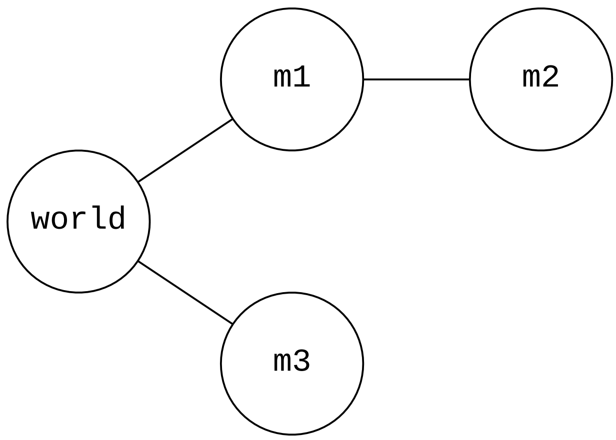


Figure 2: Hierarchy of model spaces

Initially the geometric objects are not connected, as suggested by the left hand part of figure 3. The `setup` function uses the `pivot` function three times in lines 0040–

0042. The first of these makes an adjustment to the translation part of model space `m1`. This is to bring `l1:e1`, the first end-point of line `l1` onto `p1`, the first pivot point. The second adjusts model space `m2` to bring `l2:e1`, the first end-point of line `l2` onto `l1:e2`, the second end-point of line `l1`. The third use of the `pivot` function adjusts `m3` so that `l3:e1` lies on `p2`. The second part of figure 3 shows the result of these uses of the `pivot` function.

What each use of `pivot` does is to fix the translation components of the relevant model space. All that can change is its rotation angle.

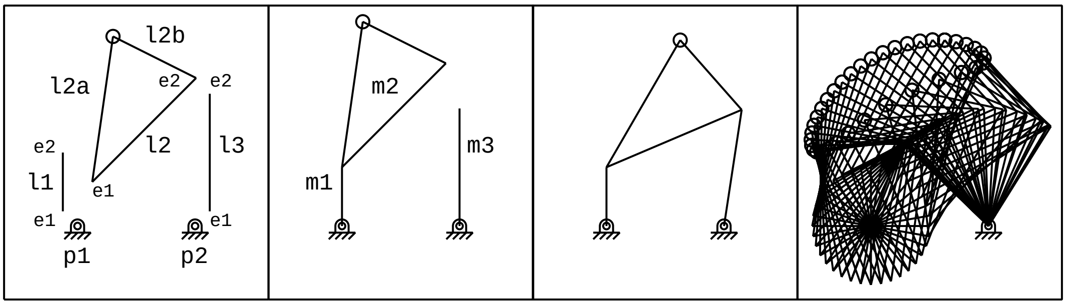


Figure 3: Stages in assembling the mechanism

It remains to join the ends of the coupler and driven link together. This cannot be achieved using the `pivot` function. Instead a constraint rule needs to be introduced. This is done in the `assemble` function (lines 0045–0050). The constraint rule says that point `l2:e2` needs to be on point `l3:e2`. In line 0049, the `on` function finds the distance between these points. The rule is becomes true when this distance is zero.

The `var` list (line 0047) just contains `m2` and `m3`. Since the translation components of each of these are fixed by one of the `pivot` commands, all that can be changed are the rotation angles. Hence there are just two degrees of freedom that can be used to resolve the constraint rule.

As its name suggest, the `cycle` function is used to simulate a cycle of the mechanism running. This function is defined in lines 0052–0071. An angular step, `astep`, is first determined; this is the rotation required of the crank at each step of the cycle. Within a loop, the rotation angle of space `m1` is repeatedly incremented (line 0062) and the mechanism is reassembled (line 0063). To track the end of the coupler, the point `ptip` is transformed into world space (by the `transf` function, line 0064). The result is assigned to a member of a global array called `qq` (declared in line 0011). Its colour and font are then set and the screen is repainted (line 0067). The argument for th `rpnt` function is provided by the argument passed in to the `cycle` function. Finally in the `cycle` function, the rotation angle of space `m1` is restored from a saved value: there is no need to do this, except that it does prevent the rotation angle increasing greatly beyond 360 degrees.

Finally in the macro, a menu is created.

GM
May 2013

# Listing

```
$ ========================================================================
$   crank_rocker.mac -- simple four bar mechanism (crank-rocker)
$ ========================================================================

dec int   npoint;                           $ number of points in cycle
npoint = 36;

dec real  d0, d1, d2, d3, d2x, d2y;         $ declare various lengths
dec geom  p1, p2, l1, l2, l3, l2a, l2b;     $ declare various geometry
dec geom  ptip;                             $ point for tip of coupler
dec geom  qq[npoint];                       $ array of geom (points)
dec mod2  m1, m2, m3;                       $ declare 2D model spaces

d0  =  8;                                   $ distance between pivots
d1  =  4;                                   $ crank length
d2  = 10;                                   $ coupler length
d2x =  8;                                   $ coupler x offset
d2y =  6;                                   $ coupler y offset
d3  =  8;                                   $ driven link length

function setup
{
   p1 = pnt( 0, 0, 0 );                     $ one pivot (at origin)
   p2 = pnt( d0, 0, 0 );                    $ second pivot
   ccol( blue(), p1, p2 );                  $ change colour
   cfont( 7, p1, p2 );                      $ change font of points
   m1 = mod2( 0, 0, 90 );                   $ crank model space
   m2 = mod2( 0, 0, -45, m1 );              $ coupler model space
   m3 = mod2( 0, 0, 90 );                   $ driven model space
   l1 = lin( 0,0,0, d1,0,0, m1 );           $ line to represent crank
   l2 = lin( 0,0,0, d2,0,0, m2 );           $ line to represent coupler
   l2a = lin( 0,0,0, d2x,d2y,0, m2 );       $ line of coupler triangle
   l2b = lin( d2,0,0, d2x,d2y,0, m2 );      $ line of coupler triangle
   ptip = pnt( d2x, d2y, 0, m2 );           $ point at tip of triangle
   l3 = lin( 0,0,0, d3,0,0, m3 );           $ line to represent driven
   ccol( red(), l1 );                       $ make crank red
   ccol( green(), l2, l2a, l2b, ptip );     $ make coupler green
   ccol( yellow(), l3 );                    $ make driven link yellow
   cfont( 4, ptip );                        $ make it a circle
   pivot( m1, l1:e1, p1 );                  $ attach crank to p1
   pivot( m2, l2:e1, l1:e2 );               $ attach coupler to crank
   pivot( m3, l3:e1, p2 );                  $ attach driven to p2
}

function assemble
{
   var  m2, m3;                             $ just change m2, m3 (angles)

   rule( l2:e2 on l3:e2 );                  $ connect coupler and driven
}
```

Figure 4: Listing of macro crank_rocker.mac (part 1)

```

function cycle
{
   dec int  i, code;                         $ declare local variables
   dec real ahold, astep;
   inp code;                                 $ function has one argument

   ahold = m1:a;                             $ hold current crank angle
   astep = 360/npoint;                       $ angular step

   loop( i, 0, npoint )                      $ start loop for cycling
   { m1:a = ahold + i*astep;                 $ advance crank angle
     assemble();                             $ reassemble
     qq[i] = transf( ptip );                 $ transform ptip to world space
     ccol( magenta(), qq[i] );               $ change colour of held point
     cfont( 6, qq[i] );                      $ and its font
     rpnt( code );                           $ repaint the graphics
   }                                         $ end of loop

   m1:a = ahold;                             $ restore original crank angle
}

graphics();                                  $ open graphics window
setup();                                     $ call setup function
assemble();                                  $ do initial assembly
rpnt();                                      $ repaint to show graphics
zoom();                                      $ zoom to fit graphics area
zoom(0.8);                                   $ zoom down a little

menu fbc                                     $ start menu definition
{ button Reset
  { setup();                                 $ call setup functio
    rpnt();                                  $ repaint (clearing screen)
  }
  button Assemble
  { assemble();                              $ do the assembly
    rpnt( 0 );                               $ repaint without clearing
  }
  button Cycle(0)
  { cycle( 0 );                              $ call cycle function
  }
  button Cycle(1)
  { cycle( 1 );                              $ call cycle function
  }
}

remmenu();                                   $ remove any existing menu
addmenu( fbc );                              $ put up new menu

$ End of file
```

Figure 5: Listing of macro crank_rocker.mac (part 2)