TMMS30 Lab 3

Eric Moringe (erimo668, 000228-5435, 43),
David Wiman (davwi279, 000120-8495, 67)

March 6, 2023

Task 1

When running the commands *cut_joints* and *path_matrix* respectively we get that joint [3] is cut and the path matrix looks as follows:

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1}$$

Task 2

(a) The plots for task 2 a):

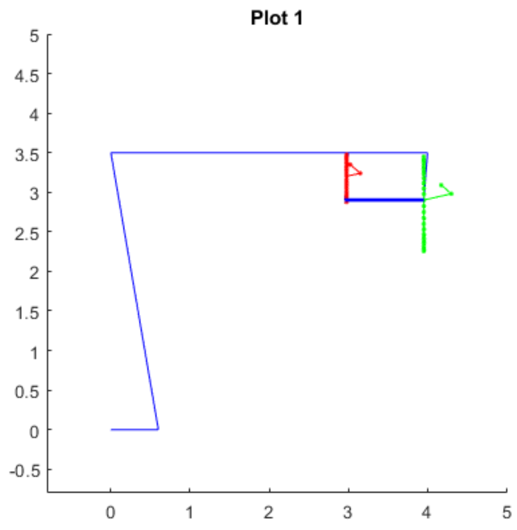


Figure 1: The assembled mechanism after simulation.

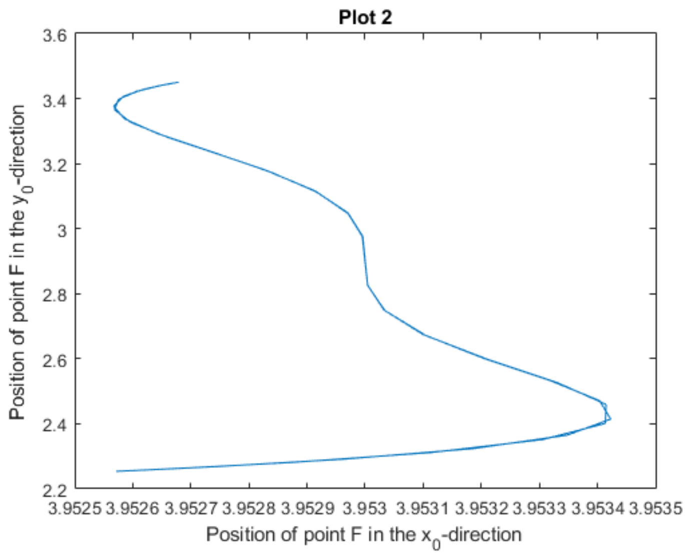


Figure 2: y_0-coordinate plotted against x_0-coordinate of point F.

The stroke of the engine is the vertical distance traveled from the top to the bottom position of the piston. Using the output point F we can take the maximum value and the minimum value in the y direction to find out the stroke. Doing this yields a distance of roughly 1.2 meters.

Doing the same maximum / minimum check for the horizontal movement of point F we get the distance 0.86 millimeters. With a long vertical stroke of 1.2 meters, it is impressive to achieve such a small horizontal movement while using a pantograph.

(b) The full MBS as a graph:

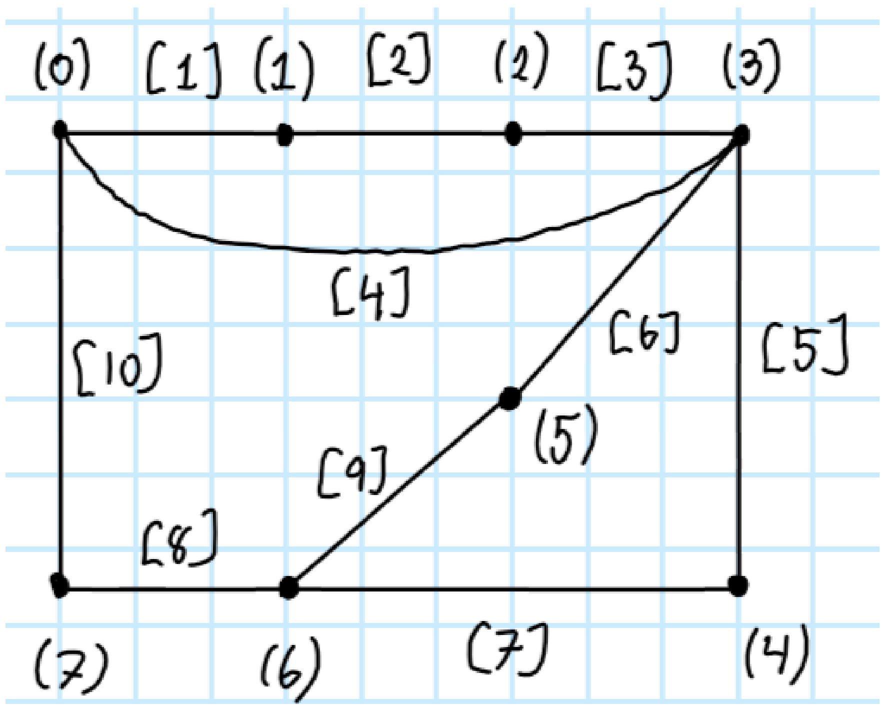


Figure 3: Graph of the mechanism.

When running the command *cut_joints* for this system we get that the joints [3], [7] and [9] are cut.

Using Prim's algorithm the following steps is done to draw the minimum spanning tree (according to how the code does it, it is not the only possible tree).

The steps of the minimum spanning tree:

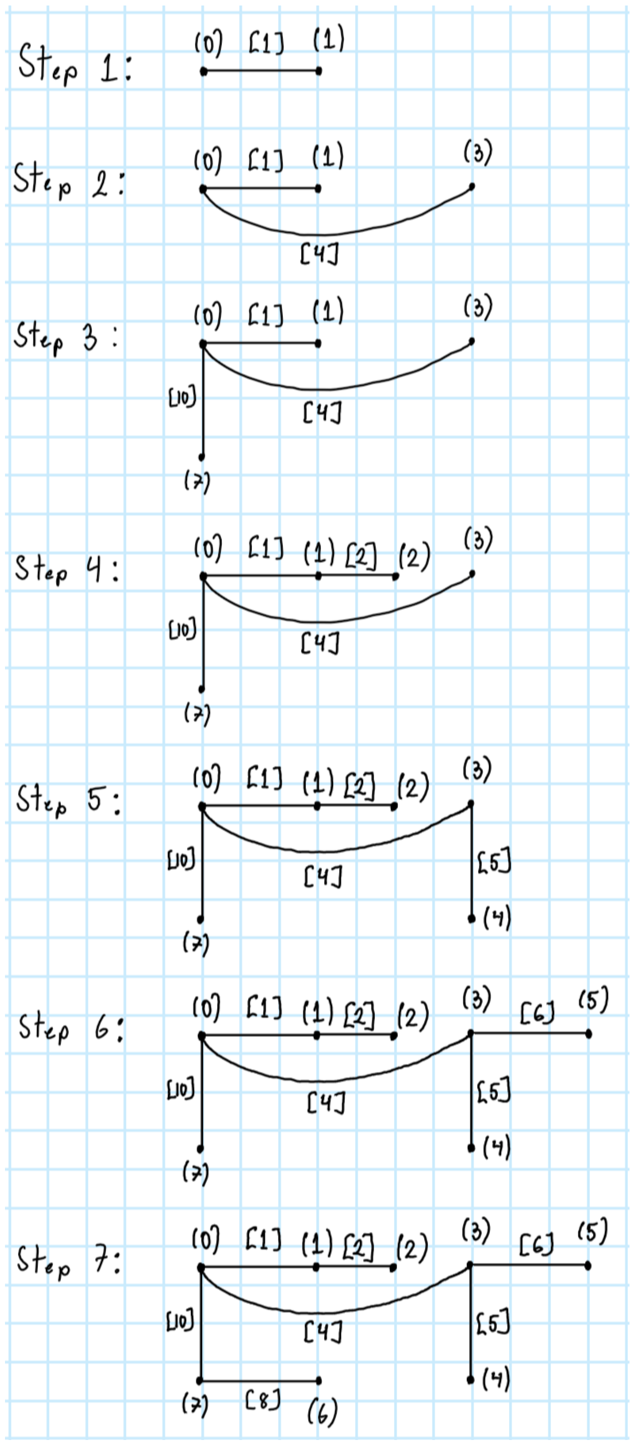


Figure 4: Step-by-step solution of a minimum spanning tree.

Now we are going to explain the variables *Branches*, *Branches_base*, *Branches_body* and *Branches_body_base*.

- Branches: Tells us which joints have to be passed in order to reach the last body of a branch. It only shows every unique branch, so that if a branch extends from anywhere else other than the ground, i.e. a branch splits in two, we only see the joints after the split to the last body for one of the branches.

- Branches_base: Tells us every joint that has to be passed to get to the end of a branch. Here we always begin at the ground and work our way up through all of the branches. When we reach an end of a branch we start over at the ground to check for the next end.

- Branches_body: Tells us what bodies have to be passed in order to reach the end of a branch. It only shows every unique branch, so that if a branch extends from anywhere else other than the ground, i.e. a branch splits in two, we only see the bodies after the split to the end.

- Branches_body_base: Tells us every body that has to be passed to get to the end of a branch. Here we always begin at the ground and work our way up through all of the branches. When we reach an end of a branch we start over at the ground to check for the next end.

Task 3

(a) The B-matrix for a RP-joint is on the form

$$B^{rp}_{j[l]} = \begin{bmatrix} e & \kappa + \tilde{r}_{j[l]} \\ 0 & 1 \end{bmatrix} \tag{2}$$

where

$$\kappa = \begin{cases} \eta_{l_1}, & \text{if i = b} \\ 0, & \text{if i = a} \end{cases} \tag{3}$$

Since e is fixed in the ground, $i = a$ and therefore $\kappa = 0$. This simplifies our B-matrix for an RPg-joint to

$$B^{rpg}_{j[l]} = \begin{bmatrix} e & \tilde{r}_{j[l]} \\ 0 & 1 \end{bmatrix} \tag{4}$$

where indexing with parentheses indicates what row in the corresponding vector is used.

The g-vector for a RP-joint is on the form

$$g_b = g_a + \left[-\dot{\theta}_a^2 s_{\mathcal{P}_a} + 2\dot{\eta}_{l_1}\dot{\theta}_i\tilde{e} - \eta_{l_1}\dot{\theta}_i^2 e + \dot{\theta}_b^2 s_{\mathcal{P}_b} \right] \tag{5}$$

Since a in our case denotes the ground and since e is fixed in the ground (i=a), both $g_a = 0$ and $\dot{\theta}_i = 0$. This simplifies our g-vector for an RPg-joint to

$$g_b = \left[\dot{\theta}_b^2 s_{\mathcal{P}_b} \right] \tag{6}$$

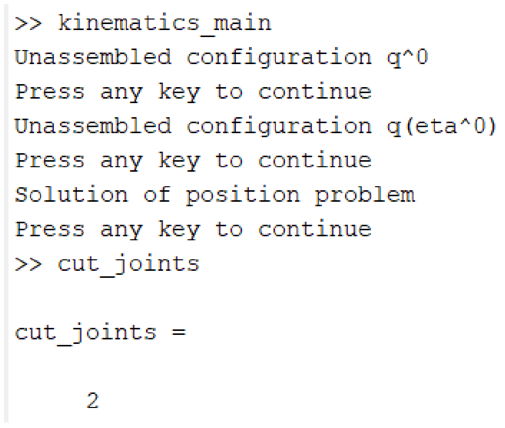
The code added in the get_B.m file is seen below.

```

case{'RPg'}
    %Add your code here!

     local_out=joint(j).inout(2); %Local body number (1 or 2) for the
        outward body of joint number j.
     body_out=joint(j).body(local_out);
         loc_out=joint(j).loc{local_out};


         input_loc=q(3*body_out-2:3*body_out-1)+rot(q(3*body_out),loc_out
            );
                      %Coordinates for input point of output body of
                      %joint j.

     rows=3*i-2:3*i;
         cols=eta_first(j):eta_first(j)+1;

     outer_loc=q(rows(1:2)); %Coordinates for the origin of a body i that
         is further away
                            %from the base than joint j.
     r_vec=outer_loc-input_loc;
         B_sub=get_B_rpg(r_vec,e,kappa);


function B_sub=get_B_rpg(r_vec,e);

%Add your code here!

B_sub=[e(1) -r_vec(2); e(2) r_vec(1); 0 1];

end
```

The code added in the get_g.m file is seen below.

```
case 'RPg'
        %Add your code here!
        g(3*out_body-2:3*out_body-1)=q_dot(3*out_body)^2*rot(q(3*out_body),
            loc_out);

        k=k+2;
```

(b) The result form cut_joints can be seen in figure 5. The joints that was cut was joint 2 which is the R-joint in the piston setup and therefore not the RPg-joint.



```
>> kinematics_main
Unassembled configuration q^0
Press any key to continue
Unassembled configuration q(eta^0)
Press any key to continue
Solution of position problem
Press any key to continue
>> cut_joints

cut_joints =

     2
```

Figure 5: Matlab terminal after the cut_joints function call.

The acceleration of the piston in the x_0-direction and the y_0-direction can be seen in figures 6 and 7 respectively.

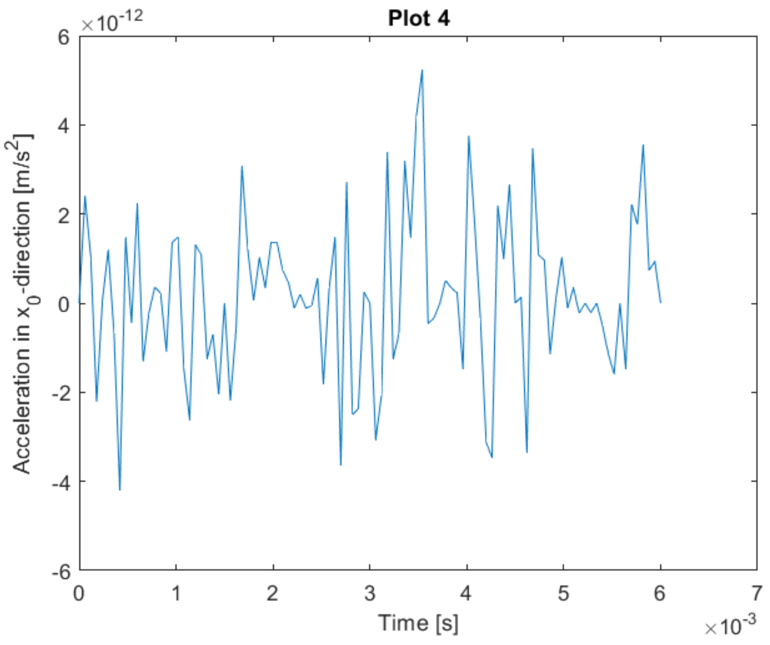


Figure 6: Piston acceleration in the x_0-direction.

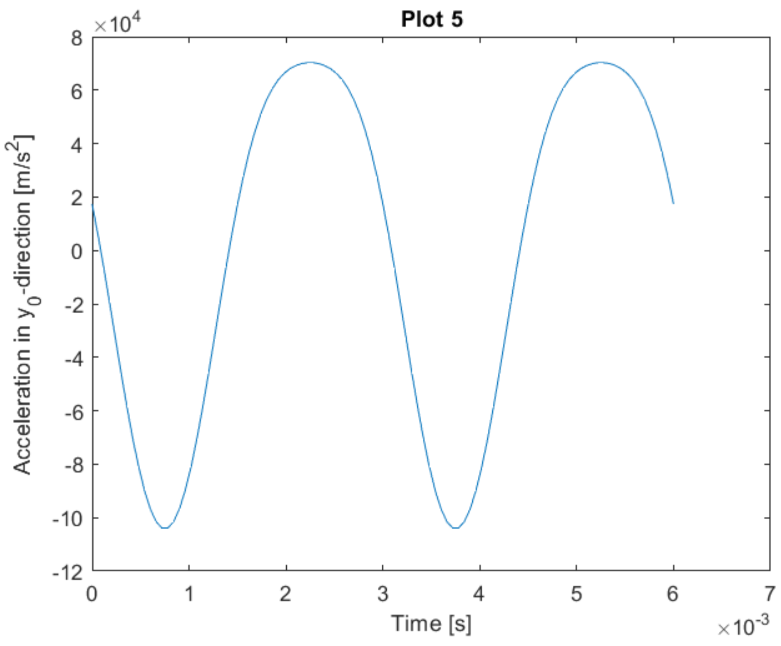


Figure 7: Piston acceleration in the y_0-direction.