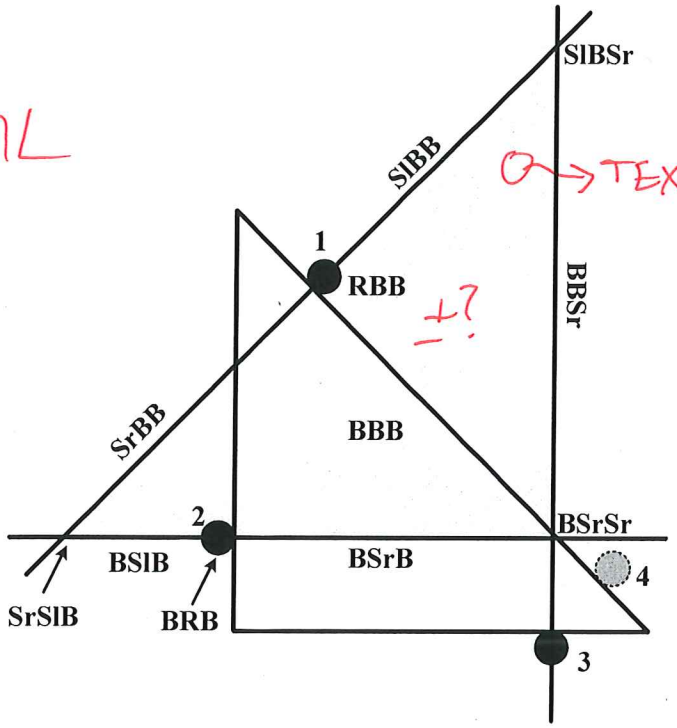
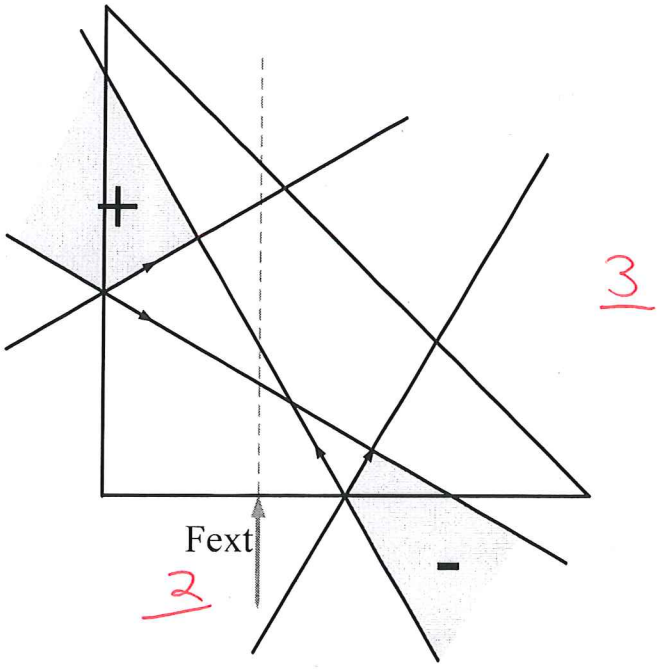
HW6 Chi Zhang

1. The feasible rotation center and are shown in the figure below. By adding finger No.4, there would be form closure.



2. The moment-labeling are shown below. The force "Fext" shown in the figure cannot be resisted by the contacts, because after adding this force the moment labeling area are not fully cancelled in the whole plane.



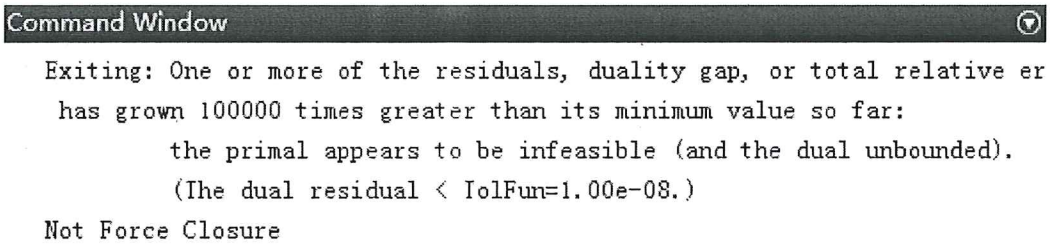
3. The code are shown below. When mu = 0.5 it is not force closure, and when mu = 2 it is force closure. The input and output of the program are shown below.

87/90

Input:

```
num_contact = 2;     %Number of Contact points
cont = [1,0;1,1];    %Coordinates of contact points
center = [0,0];      %Coord of the origin (center)
ang_c = [pi/2,5*pi/4]; %Normal direction of each contact force in angle
mu = 0.5 (2);        %Frictional coefficient
```

Output when mu = 0.5:



Output when mu = 2:

Code:

```
clear;
clc;
num_contact = 2;     %Number of Contact points
cont = [1,0;1,1];    %Coordinates of contact points
center = [0,0];      %Coord of the origin (center)
ang_c = [pi/2,5*pi/4]; %Normal direction of each contact in angle
mu = 0.5;              %Frictional coefficient
Fext = [0,0,0]';     %External force: 0 in this case

thetamu = atan2(mu,1); %Calculate angle of the half friction cone
k = 1;

for i=1:num_contact %For each contact point, calculate the wrench
    F(:,k) = [0,cos(ang_c(i)-thetamu),sin(ang_c(i)-thetamu)]';
    %Calculate the vector between the contact point and the origin
    r = [(cont(i,1)-center(1)),(cont(i,2)-center(2)),0];

    if ang_c(i)-thetamu ~= 0
        force = [F(2:3,k)',0];
        torque = cross(r,force); %Torque term = r cross f
        F(1,k) = torque(3);
    else
        F(1,k) = 0;
    end
    k = k + 1;

    F(:,k) = [0,cos(ang_c(i)+thetamu),sin(ang_c(i)+thetamu)]';
    C1 = - F(3,k)*cont(i,1) + F(2,k)*cont(i,2);
    if ang_c(i)+thetamu ~= 0
        force = [F(2:3,k)',0];
        torque = cross(r,force);
        F(1,k) = torque(3);
    else
```

```
        F(1,k) = 0;
    end
    k = k + 1;
end
```

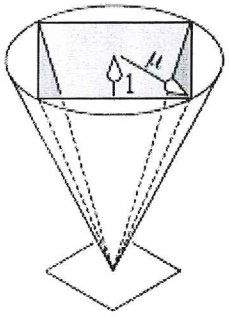
```
%Linear programming: Minimize K with each K>=1 and check if
%the convex contains the origin.
C = ones(1,2*num_contact);
A = zeros(1,2*num_contact);
b = 0;
for i = 1:2*num_contact
    LB(i) = 1;
    UB(i) = Inf;
end
[K,FVAL,EXITFLAG] = linprog(C,A,b,F,Fext,LB,UB);

%Print "Force Closure" if there is a solution of K,
%and print "No" otherwise.
if EXITFLAG == 1
    fprintf('Force Closure\n');
else
    fprintf('Not Force Closure\n');
end
```

4. How to generalize the previous program to work for 3D rigid bodies.

The general idea is still using linear programming to check if there exists a set of k such that Fk = 0 satisfying $k \geq 1$. 2 modifications should be made:

(1) All positions, velocities, accelerations, wrenches and external forces contain 6 components instead of 3.

(2) For each friction cone on the contact point, there needs to be several wrenches (usually 4) estimating the cone instead of 2 planar vectors, usually we use an inscribed cone, as shown in the figure below.



The detailed process of writing each component in the wrenches and the linear programming is the same to problem 3.

5. The code is shown below. Note that though 2*12=24 different wrenches are written, there are only 16 independent ks, as the magnitude of wrenches acting on block 3 is the same as those on the same points but acting on block 1 and 2, though the value of wrenches are not the same (the torque term and the direction of force).

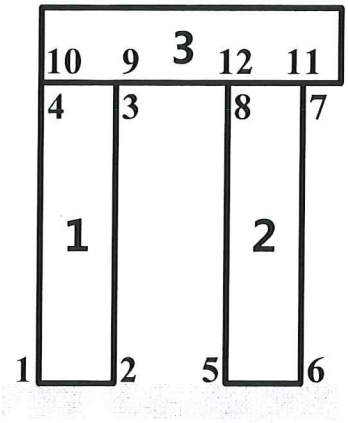
```
function [EXITFLAG] = P5(acc_x)
%Everything is calculated in the hybrid frame
```

```
position = [0,0,0];  %theta, x, y
velocity = [0,0,0];  %dtheta, dx, dy
acceleration = [0,acc_x,0];  %ddtheta, ddx, ddy
gamma = [pi-atan(4/3)+position(1),atan(2)+position(1),pi/2+position(1)];
%Angle of centripetal acceleration

acceleration_rot = [0,-cos(gamma(1))*(velocity(1))^2*0.05,-
sin(gamma(1))*(velocity(1))^2*0.05;
                    0,-cos(gamma(2))*(velocity(1))^2*0.02*sqrt(5),-
sin(gamma(2))*(velocity(1))^2*0.02*sqrt(5);
                    0,-cos(gamma(3))*(velocity(1))^2*0.09,-
sin(gamma(3))*(velocity(1))^2*0.09];
                      %linear acceleration of blocks caused by rotation

gamma = [atan(3/4)+position(1),-atan(0.5)+position(1),position(1)];
%Angle of tangential acceleration

acceleration_ang =
[0,cos(gamma(1))*acceleration(1)*0.05,sin(gamma(1))*acceleration(1)*0.05;

0,cos(gamma(2))*acceleration(1)*0.02*sqrt(5),sin(gamma(2))*acceleration(1)*0.02*sqrt(5);

0,cos(gamma(3))*acceleration(1)*0.09,sin(gamma(3))*acceleration(1)*0.09];
                      %linear acceleration of blocks caused by angular acceleration

acceleration_tray = [acceleration(1),acceleration(2),acceleration(3);
                      acceleration(1),acceleration(2),acceleration(3);
                      acceleration(1),acceleration(2),acceleration(3)];
                  % Acceleration of blocks caused by the acceleration ofthe tray

acceleration_block = acceleration_rot + acceleration_ang + acceleration_tray;
                  % Acceleration of blocks caused by the acceleration ofthe tray

G = diag([9e-4,1,1]);

num_contact = 12;     %Number of Contact points
cont_point = 0.01*[-4*cos(position(1))+position(2),-4*sin(position(1))+position(3);
              -2*cos(position(1))+position(2),-2*sin(position(1))+position(3);
              -2*cos(position(1))-8*sin(position(1))+position(2),-
2*sin(position(1))+8*cos(position(1))+position(3);
              -4*cos(position(1))-8*sin(position(1))+position(2),-
4*sin(position(1))+8*cos(position(1))+position(3);
              1*cos(position(1))+position(2),1*sin(position(1))+position(3);
              3*cos(position(1))+position(2),3*sin(position(1))+position(3);
              3*cos(position(1))-
8*sin(position(1))+position(2),3*sin(position(1))+8*cos(position(1))+position(3);
              1*cos(position(1))-
8*sin(position(1))+position(2),1*sin(position(1))+8*cos(position(1))+position(3);
              -2*cos(position(1))-8*sin(position(1))+position(2),-
2*sin(position(1))+8*cos(position(1))+position(3);
              -4*cos(position(1))-8*sin(position(1))+position(2),-
4*sin(position(1))+8*cos(position(1))+position(3);
              3*cos(position(1))-
8*sin(position(1))+position(2),3*sin(position(1))+8*cos(position(1))+position(3);
              1*cos(position(1))-
8*sin(position(1))+position(2),1*sin(position(1))+8*cos(position(1))+position(3)];

         %Coordinates of contact points

center = 0.01*[-3*cos(position(1))-4*sin(position(1))+position(2),-
3*sin(position(1))+4*cos(position(1))+position(3);
```

```
          2*cos(position(1))-
 4*sin(position(1))+position(2),2*sin(position(1))+4*cos(position(1))+position(3);
          -9*sin(position(1))+position(2),+9*cos(position(1))+position(3)];
       %Coord of the centers of each block

 ang_c = [pi/2+position(1),pi/2+position(1)...
          -pi/2+position(1),-pi/2+position(1)...
          pi/2+position(1),pi/2+position(1)...
          -pi/2+position(1),-pi/2+position(1)...
          pi/2+position(1),pi/2+position(1)...
          pi/2+position(1),pi/2+position(1)];
      %Normal direction of each contact point in angle

 mu = 0.25;          %Frictional coefficient
 Fext_b1 = G*acceleration_block(1,:)'+[0,0,9.8]';
 Fext_b2 = G*acceleration_block(2,:)'+[0,0,9.8]';
 Fext_b3 = G*acceleration_block(3,:)'+[0,0,9.8]';

 theta_mu = atan2(mu,1); %Calculate angle of the half friction cone
 k = 1;

 for i=1:num_contact %For each contact point, calculate the wrench
     Fric(:,k) = [0,cos(ang_c(i)-theta_mu),sin(ang_c(i)-theta_mu)]';
     %Calculate the vector between the contact point and the center
     r = [(cont_point(i,1)-center(ceil(i/4),1)),(cont_point(i,2)-center(ceil(i/4),2)),0];

     if ang_c(i)-theta_mu ~= 0
         force = [Fric(2:3,k)',0];
         torque = cross(r,force); %Torque term = r cross f
         Fric(1,k) = torque(3);
     else
         Fric(1,k) = 0;
     end
     k = k + 1;

     Fric(:,k) = [0,cos(ang_c(i)+theta_mu),sin(ang_c(i)+theta_mu)]';
     if ang_c(i)+theta_mu ~= 0
         force = [Fric(2:3,k)',0];
         torque = cross(r,force);
         Fric(1,k) = torque(3);
     else
         Fric(1,k) = 0;
     end
     k = k + 1;
 end

 %Linear programming: Minimize K with each K>=0 and check if
 %the convex contains the origin.
 C = ones(1,16);
 A = zeros(1,16);
 b = 1;
 for i = 1:16
     LB(i) = 0;
     UB(i) = Inf;
 end
 Fext = [Fext_b1;Fext_b2;Fext_b3];
 F = [Fric(:,1:8),zeros(3,8);
      zeros(3,8),Fric(:,9:16);
      zeros(3,4),Fric(:,17:20),zeros(3,4),Fric(:,21:24)];

 [K,FVAL,EXITFLAG] = linprog(C,A,b,F,Fext,LB,UB);
```

```
%Print "Stable" if there is a solution of K,
%and print "No" otherwise.
if EXITFLAG == 1
    fprintf('Stable\n');
else
    fprintf('Untable\n');
end
end
```

10



By using binary search (code shown below), the max and min acceleration in x direction can be found:

```
ub = 20;
lb = 0;
while(abs(ub-lb)>0.001)
    if P5(ub)==1
        lb = ub;
        ub = ub+0.5*length;
    else
        length = ub-lb;
        ub = lb+0.5*(length);
    end
end
```

5

When the tray is horizontal, the max acceleration in - x and x direction is -2.45m/s^2 and 2.45 m/s^2, respectively.

6. How to generalize the solution to arbitrary assemblies for planar rigid bodies?

The process is similar to the previous problem. Note that all the calculations should be done in a consistent coordinate, in this example we use the hybrid frame.

(1) For each object in the assembly, calculate its acceleration (angular, x and y) caused by the motion of the base (tray), i.e. its position, velocity and acceleration.
(2) If there is gravity, then the RHS of the force balancing equation could be G*a + g, where G is the inertia matrix (3 by 3), a is the acceleration calculated in (1), and g is gravity.
(3) For each object, write the wrenches of each contact point in the current configuration, then the coefficients of the LFS of the force balancing equation should be the certain wrenches acting on this object.
(4) Combine all the LHS and RHS part together. For the RHS, just write all the forces in a column in the order of object numbers for a vector Fext. For the LHS, if a wrench related to an independent magnitude k_m is

15

acting on the object *n*, then write the corresponding wrench in the coefficient matrix in the position (n, m); otherwise write zeroes. The final coefficient matrix F should be similar to the table shown below.

(5) Use linear programming to find k:

Minimize 1^T*k, satisfying:

$$F*k = Fext$$

$$k \geq 0$$

5

	K_1	K_2	K_3	K_4	...	K_8
OBJECT 1	F_1	F_2	0	0	...	0
OBJECT 2	0	0	F_3	F_4	...	0
...	...	...	...	...	...	...
OBJECT N	0	0	F_9	F_{10}	...	F_{16}

If such k exists, then the assembly is stable; otherwise it's unstable.

7. How to generalize the solution to arbitrary assemblies for 3D rigid bodies?

A little modification should be made on the solution in problem 6, and the idea of modification is similar to problem 3. 2 modifications should be made:

(1) All positions, velocities, accelerations, wrenches and external forces contain 6 components instead of 3.

(2) For each friction cone on the contact point, there needs to be several wrenches (usually 4) estimating the cone instead of 2 planar vectors, usually we use an inscribed cone. 5

The detailed process of writing each component in the wrenches and the linear programming is the same to problem 6.

8.

(a). When the path is given, what we are doing is actually velocity planning along the path in the (S, S dot) phase plane.

With the tool shown above in problem 6, for any certain point in the (S, S dot) phase plane, we can find the max and min possible acceleration representing by the upper and lower bound of a cone, similar to the method in chapter 9. Note that we are also considering assembly stability together with torque limit, the cone should be drawn in a little different way:

First, calculate the dynamics of the robot, in this process the anti-force that the robot is acting on the assembly should be added on the force term. Specifically, if we apply the Newton-Euler inverse dynamics algorithm, the anti-force of the combination of all forces acting on the assembly becomes the F_{tip} term (F_{n+1} in Eq. 8.56, as shown below).

5

15

- **Backward Iteration**: for $i = n$ to 1 do

$$\mathcal{F}_i = \mathrm{Ad}^T_{T_{i,i+1}}(\mathcal{F}_{i+1}) + \mathcal{G}_i\dot{\mathcal{V}}_i - \mathrm{ad}^T_{\mathcal{V}_i}(\mathcal{G}_i\mathcal{V}_i) \quad (8.56)$$

$$\tau_i = \mathcal{F}_i^T\mathcal{A}_i. \quad (8.57)$$

Now for each point in the phase plane, the cone determined by torque limit of the robot can be drawn.

Second, draw the cone of assembly stability. With the scaling, each S, dS and ddS value is related to a certain configuration, velocity and acceleration based on the definition and Eq. 9.1 - 9.2 as shown below:

$$\dot{\theta} = \frac{d\theta}{ds}\dot{s} \quad (9.1)$$

$$\ddot{\theta} = \frac{d\theta}{ds}\ddot{s} + \frac{d^2\theta}{ds^2}\dot{s}^2. \quad (9.2)$$

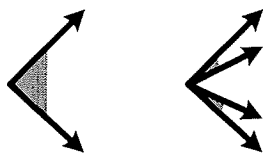
Therefore, for each point on the phase plane, given a ddS value, the program shown in problem 6 returns whether the assembly stability can hold or not. With this tool, we can draw the cone of assembly stability using binary search to find the max and min possible acceleration. **

Finally, by choosing the intersection of the two cones, we can obtain the cone representing the acceleration limit without breaking the torque limit and assembly stability.

For each cone drawn on a certain point on the phase plane, if the upper bound coincides with the lower bound, then it would be on a limit boundary that the curve we planned in the phase plane can never run into. By connecting the continuous boundary points, we get the boundaries, sometimes isolated islands on the plane that we can never run into, as shown in the figure below.

Then we can do the time-optimal time-scaling method that can deal with isolated islands for this graph using the method shown in chapter 9 (page 244, second bullet). The general idea is similar to the conditions of running into the velocity limit curve, we find a curve we can go along that is tangent to the boundary of the island, and trace back along this curve till it intersects with the original one.

** (1) We have to prove that the assembly stability cone is a single continuous one as shown on the left, instead of discrete regions on the right in the figure below, so that binary search can be used to find the upper and lower bound of the cone. As the combination of wrenches is always convex and positive definite, and the relationship between the wrench and acceleration is linear (F=G*a), and G matrix is positive definite, so the accelerations should also be convex and positive definite. When projecting them onto the ddS direction, it should also be a convex cone. (Assumption and general idea proposed by Prof. Lynch in class, no detailed proof yet).



** (2) A specific binary search method for this problem:

Searching for lower and upper bounds can be done separately, but with the similar method.

First, find a vector that is in the feasible region of the acceleration cone. This can be done using the following method:

checkpoint = 2;

while (the acceleration is not feasible)

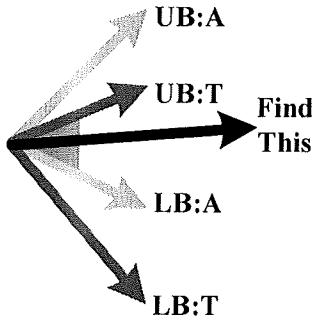
for (i = 1:2:k-1)

Check acceleration of (i/k)*(LB+UB);

end for

k = 2*k;

end while



UB:A – Assembly stability upper bound LB:A – Assembly stability lower bound

UB:T – Torque limit upper bound LB:T – Torque limit lower bound

Once a possible acceleration is found, binary search can be applied as the algorithm shown in problem 5:

```
ub = UB:T;
lb = Find This;
while(abs(ub-lb)>0.001)
    if P5(ub)==1
        lb = ub;
        ub = ub+0.5*length;
    else
        length = ub-lb;
        ub = lb+0.5*(length);
    end
end
```

For the upper bound case, ub of binary search is the upper bound of torque limit (UB:T), and lb is the one found above.

Similarly, the lower bound of the assembly stability cone can also be found.

(b). This problem can be divided into 2 sub problems:

First, find a feasible path.

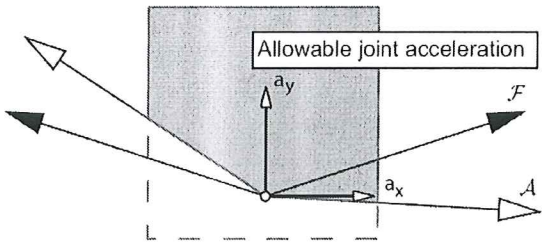
Second, find feasible velocity and acceleration planning for the path.

A possible way is draw the visibility graph, then find a feasible path using RRT or RPM.

For this path, try to find a time-scaling as shown in problem (a). If the result exists, then we find a trajectory that satisfies both torque limits and assembly stability from one state to another; if the time-scaling for this path is not feasible, then go back to the path finding process.

9.

(1). Fig. 2:

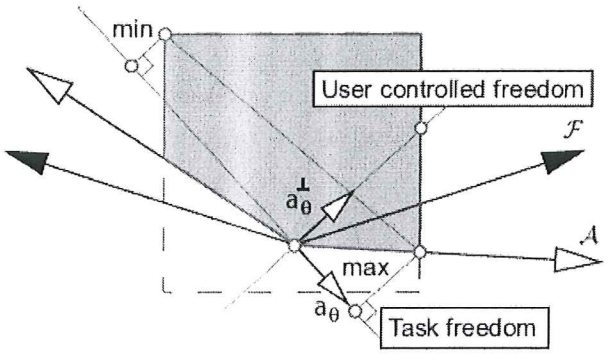


This graph shows the feasible friction cone and acceleration cone of the robot. The dark polygon (square) shows the feasible acceleration bounded by torque limits. As x and y are independent, they would form a square. We can also see that y is not symmetric, mainly because the robot cannot accelerate too much in –y direction or it will break contact with the object. The friction cone is the same as what we talked about in class, and the acceleration cone is mapped from the friction cone using the following equation:

$$\mathcal{A} = G^T M^{-1} G(\mathcal{F})$$

The idea is just using $a = f/m$, but everything is in matrix form. G relates contact forces to wrenches on the object, which is actually a coordinate switching. When calculating the acceleration boundary, first switch to the object frame, calculate the acceleration boundaries of the object, and then switch back to the robot frame so that we obtain the robot acceleration cone. By knowing the acceleration cone of the robot, we know that for any acceleration inside the feasible region, the object will be in rolling contact with the robot, which is one condition problem would follow.

(2). Actually Figure 2 does not lead to Figure 1 (ii) directly, but Figure 3 does. So Figure 3 is explained here.



The acceleration of the object can be written in the following form:

$$\ddot{q}_o = \mathrm{A}\begin{pmatrix} a_x \\ a_y \end{pmatrix} + b \qquad (10)$$

Where $(a_x, a_y)^T$ is the contact acceleration of the robot, which is bounded by the green polygon shown in the figure above. Then we can try to separate the acceleration terms:

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{pmatrix} = \begin{bmatrix} \mathrm{A}_{xy} \\ \mathrm{A}_\theta \end{bmatrix} \begin{pmatrix} a_x \\ a_y \end{pmatrix} + \begin{bmatrix} b_{xy} \\ b_\theta \end{bmatrix} \qquad (11)$$

So that we can project the A matrix into two parts, the upper part relates to x and y acceleration, which we don't care much, and the lower part relates to angular acceleration, and that is what we care about. By using the following equations we can obtain the orthogonal vectors a_θ and $a_\theta^\perp$:

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} = a_\theta \alpha + a_\theta^\perp \beta$$

$$\mathrm{A}_\theta a_\theta^\perp = 0$$

Therefore, we can find a direction of acceleration of the robot that only influences the angular acceleration $\ddot{\theta}$ of the object:

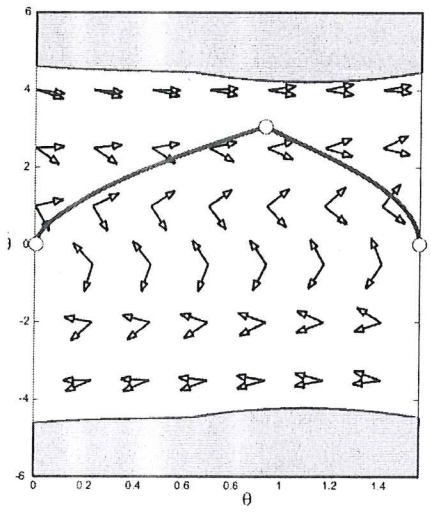
$$\ddot{\theta} = \mathrm{A}_\theta a_\theta^T \alpha + b_\theta \qquad (13)$$

$$\begin{pmatrix} \ddot{x} \\ \ddot{y} \end{pmatrix} = \mathrm{A}_1 \alpha + \mathrm{A}_2 \beta + b_{xy} \qquad (14)$$

Which is the a_θ direction shown in Figure 3. Then following the boundary of feasible accelerations, we can obtain the max and min possible acceleration of the robot that influences the angular acceleration of the object along a_θ axis, and by substituting them into equation (13), we can find the limits of angular acceleration $\ddot{\theta}$ when $(\theta, \dot{\theta})$ is given in the task freedom space.

The user controlled freedom direction is the one that influences the linear acceleration of the block, as this will not influence the angular term, users can choose feasible values by themselves, that's why when there could be several different trajectories though the rotating motion of the block is chosen.

5



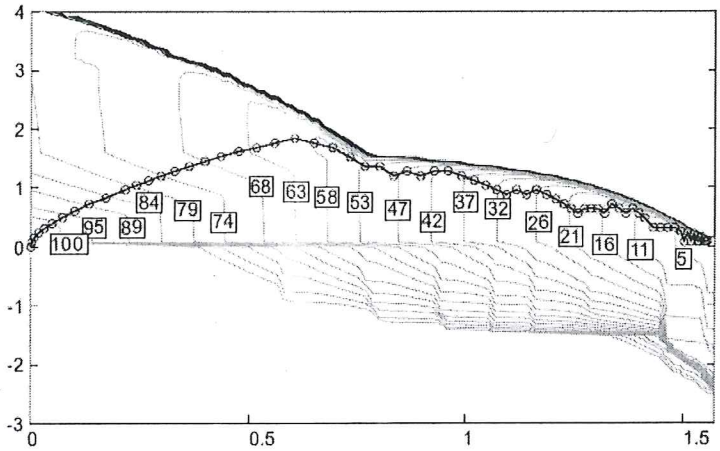
5

(3). How Figure 2 (3) leads to Figure 1 (ii)? As discussed above, after finding the max and min feasible accelerations in a_θ direction, we can calculate the max and min possible angular acceleration of the object, $\ddot{\theta}$ for a certain point in the $(\theta, \dot{\theta})$ plane. Similar to what we do in the $(S, \dot{S})$ phase plane, i.e. drawing two arrows representing for the upper and lower bound of acceleration $\ddot{S}$ by using arctan $(\ddot{S}/S)$, we can draw a cone representing for feasible region of angular accelerations of the object for each $(\theta, \dot{\theta})$ without breaking rotating contact or exceeding the robot's torque limits. When the lower bound coincides with the upper bound, there would be a limit on the $(\theta, \dot{\theta})$ plane, which is the pink region shown in the figure above. Then using the similar idea of bang-bang algorithm, we can generate a curve representing for the angular velocity of the object at each angle configuration from 0 to pi/2.

5

(4). This figure shows a trajectory in the $(\theta, \dot{\theta})$ plane. This is an isocontour, which means that on each light green line, the cost function has the same value, and the values are the time from the state to the final state in milliseconds, shown as the numbers in the graph. For each point on the state, the direction-to-go is one that is closest to the gradient of the isoline (as the contour is discrete as shown in Figure 6, it cannot be exactly the gradient). The isolines are the cost functions (minimum time) calculated using backward dynamic programming. Thus by substituting in the starting point, backward dynamic programming will return the optimal cost-to-go to the end point, which is the optimal time of the whole motion. Following the specific steps in the dynamic programming, the optimal path could be found, as the thick curve shown below. The highly duplicated part of the gray lines above x axis means that points above this area cannot reach (pi/2, 0), so there is no isoline.

5



(5). For generating the curve in the $(\theta, \dot{\theta})$ plane, it is similar to time-optimal time scaling, i.e. going along the feasible direction from the start to the end, which is (0, 0) and (pi/2, 0) in this case. The difference is that for time-optimal time-scaling, we focused on the acceleration of the robot itself, which is motion planning for the robot directly. Once we obtain the curve in the $(S, \dot{S})$ phase plane, we can calculate the configuration, velocity, acceleration and how much force or torque for the robot at state so that the robot can go along the trajectory. While for the method in this paper, we are actually planning for the angle of the object θ, but we are planning the object's angular motion. Once obtaining that, we calculate feasible motions of the robot by choosing feasible user controlled values, and by using the combination of these two values, we can obtain the actual acceleration of the robot. After this step, we now have the direct access to do motion planning for the robot. In conclusion, the time-optimal time-scaling is more straight forward, and is not as "free" as Sidd's problem, as the time-optimal time-scaling plans the motion for the robot directly from the phase plane, while Sidd's method actually plans for the object first, then choose feasible values for the robot so that the object can follow the planned θ "trajectory".

5

15