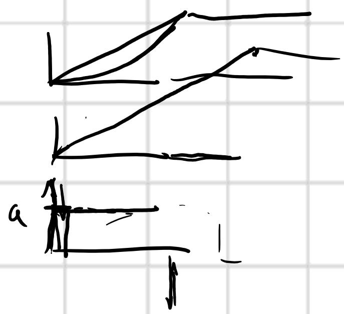
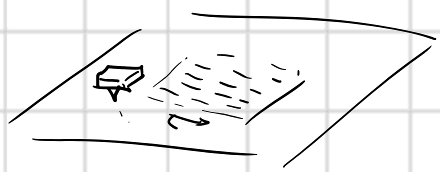
Path and Trajectory Planning

Overview:

Path / Trajectory planning are used to define robot motion along the task.

Consider an example where the robot is performing a task. Starts at home position, moves to a table, welds a circular motion on part on top of the table, moves back home.





This motion is considered in task space, and has to be specified to the controller in joint or configuration space.

This set of notes considers two general approaches:

1) Path planning using potential fields

2) Trajectory planning via polynomials - 2 method. linear segments w/ parabolic blends

Definitions:

Path – Defines robot motion considering position only.

Trajectory – Defines robot motion consider position and time (i.e., when the robot is at certain positions, velocity along the path.

Page 2

Wednesday, April 19, 2017 1:30 PM

1) Path planning using potential fields

The idea here is to treat the robot as a body attracted or repelled to its goal or obstacles. The goal and obstacles create a potential field that acts on the robot. Then, the desired path is one that minimizes the potential energy of the robot. The force on the robot is calculated as the negative gradient of the potential and the joint torques are given as the transpose of J times the force.

Page 3

Wednesday, April 19, 2017 1:30 PM

2) Trajectory Planning Via Polynomials:

Step 1) Define the General desired motion in tool space
Step 2) Define the position, velocity, time at key points along the task
Step 3) Using Inverse kinematics, find the position, velocity at key points in joint space.
Step 4) Define via points between point to point motion. Do this as described in step 5.
Step 5) In joint space, map the trajectories as splines that satisfy the end conditions. For desirable motion, the spines must have finite jerk (be c2 constinuous)
Step 6) Define a real-time controller update rate (proposed here as 300 Hz).
Step 7) Access the mapped joint-space trajectories at the controller update-rate to get target position and velocity for each input.
Step 8) Pass these target position and velocity inputs to the joint-level PID controller to generate error signal: $e, \dot{e}$ and $\int e$ with $e = (q_d - q_a)$, etc., and corresponding control input, $u = k_p e + k_d \dot{e} + k_i \int e$.
Step 9) Continue steps 4-6, repeat steps 1-3 as additional tool-space trajectory target points are received.

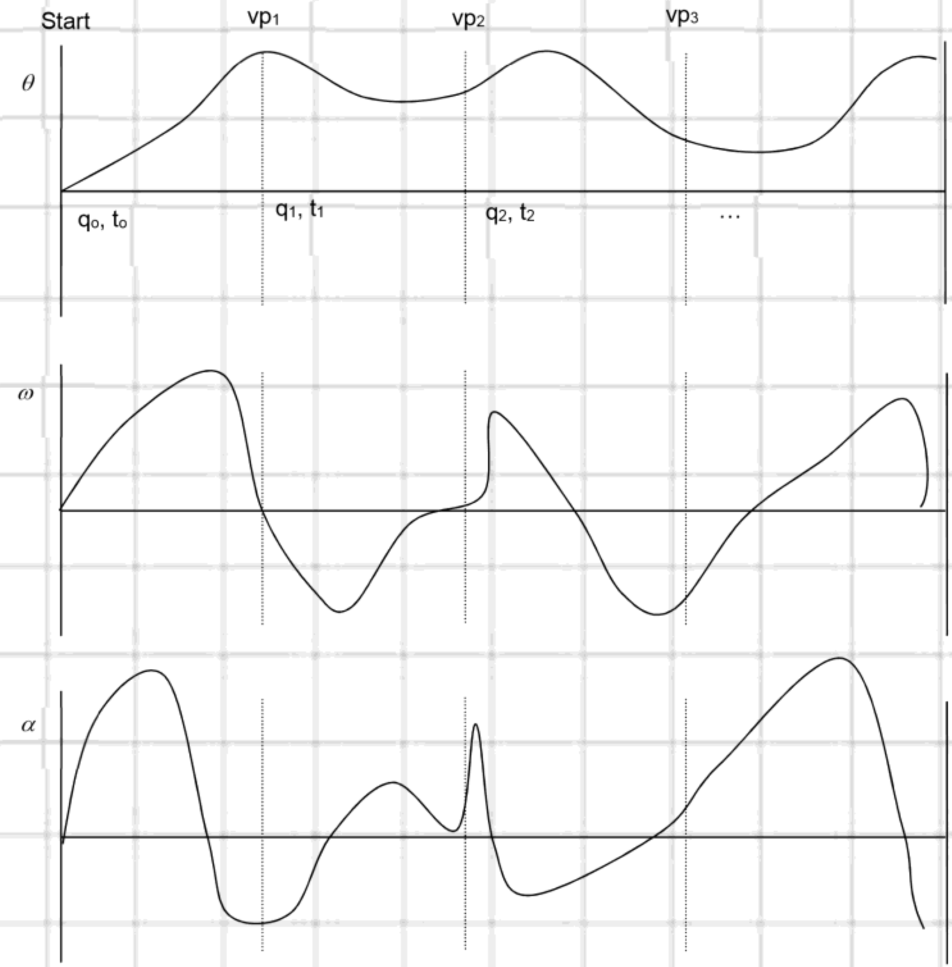
Acceptable curve fitting

Acceptable robot motion will be smooth along path and velocity and continuous through 2nd derivative (finite input torques required).

Consider the curve below:

Page 4

Wednesday, April 19, 2017 1:30 PM



$\leftarrow \frac{d^2}{dt^2}(\theta)$

The motion is desired to be continuous with finite accelerations (and ideally finite jerk). The general approach is to satisfy the trajectory as a spline consisting of a series of segmented trajectories with matching boundary conditions.

Three acceptable curves are suggested (and commonly used):

1) Cubic polynomials $\rightarrow \; a_3t^3 + a_2t^2 + a_1t + a_0 = \theta$
2) higher order polynomials
3) Linear segments with parabolic blends (LSPBs)

These are considered in turn below.

Page 5

Wednesday, April 19, 2017 1:30 PM

cubic splines: The joint motion is defined as a cubic polynomial,

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$
$$\dot{q}(t) = a_1 + 2a_2 t \quad + 3a_3 t^2$$
$$\ddot{q}(t) = 2a_2 + 6a_3 t$$

The cubic spline permits four boundary conditions, generally taken as the boundary conditions on joint position and velocity as:

$$q(t_0) = q_0, \qquad q(t_1) = q_1,$$
$$\dot{q}(t_0) = \dot{q}_0, \ \dot{q}(t_1) = \dot{q}_1,$$

The coefficients of the cubic are solved as:

$$\begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{Bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_1 & t_1^2 & t_1^3 \\ 0 & 1 & 2t_1 & 3t_1^2 \end{bmatrix}^{-1} \begin{Bmatrix} q_0 \\ \dot{q}_0 \\ q_1 \\ \dot{q}_1 \end{Bmatrix}$$

The cubic polynomials match boundary conditions through velocity and guarantee finite acceleration but not finite jerk.

Page 6

Wednesday, April 19, 2017 1:30 PM

Quintic splines: To include boundary conditions on position, velocity and acceleration, a fifth-order polynomial can be used:

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$
$$\dot{q}(t) = a_1 + 2a_2 t \quad + 3a_3 t^2 4a_4 t^3 + 4a_5 t^4$$
$$\ddot{q}(t) = 2a_2 + 6a_3 t \quad + 12a_4 t^2 + +20a_5 t^3$$

The boundary conditions are given as:

$$q(t_0) = q_0, \qquad q(t_1) = q_1$$
$$\dot{q}(t_0) = \dot{q}_0, \; \dot{q}(t_1) = \dot{q}_1$$
$$\ddot{q}(t_0) = \ddot{q}_0, \; \ddot{q}(t_1) = \ddot{q}_1$$

and the coefficients of the quintic polynomial are solved as

$$\begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_3 \\ a_5 \end{Bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_1 & t_1^2 & t_1^3 & t_1^4 & t_1^5 \\ 0 & 1 & 2t_1 & 3t_1^2 & 4t_1^3 & 5t_1^4 \\ 0 & 0 & 2 & 6t_1 & 12t_1^2 & 20t_1^3 \end{bmatrix}^{-1} \begin{Bmatrix} q_0 \\ \dot{q}_0 \\ \ddot{q}_0 \\ q_1 \\ \dot{q}_1 \\ \ddot{q}_1 \end{Bmatrix}$$

Linear splines with parabolic blends: These represent simplistic segments as a combination of quadratic and linear displacement functions (and corresponding constant and zero acceleration functions). These permit simplistic control for minimizing accelerations but do not permit velocity control or limit jerk.

Δt

(1) $$q(t) = q_0 + \frac{a}{2}t^2, 0 \le t \le t_b$$

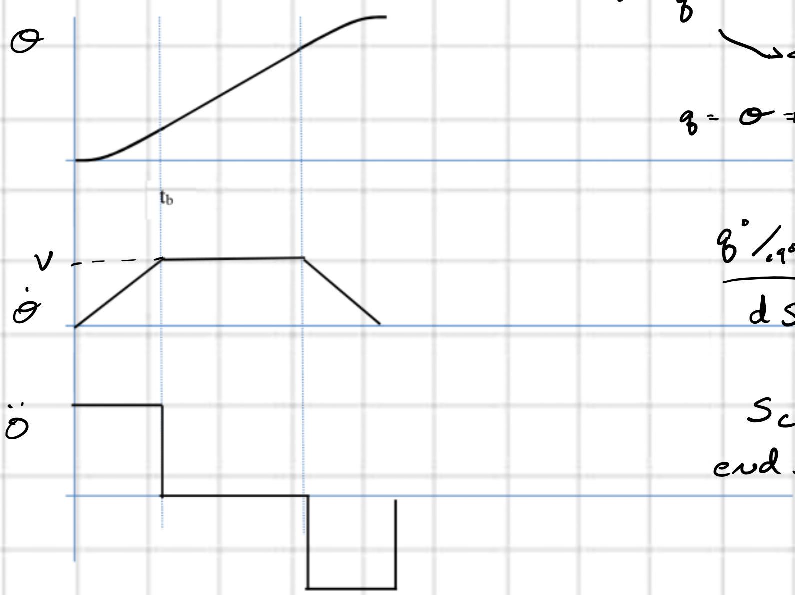
(2) $$q(t) = \frac{q_f + q_0 + Vt_f}{2} + Vt, t_b < t \le t_f - t_b$$

(3) $$q(t) = q_f - \frac{a}{2}t_f^2 + at_f t - \frac{a}{2}t^2, t_f - t_b < t \le t_f$$

q_0, q_f
t_b, t_f
a, V

Then solve for q, $\dot{q}$ along the profile.

→ q → d.c. RC servo.
→ steps stepper.



$q = \theta$ in degrees:
half stepping (.9 degrees/step)
0 steps = 0 degrees.

$q^\circ / .9^\circ/\text{step} = \overset{(int)}{S}$ steps.

$dS = S - S_{current}$.
send dS steps to your stepper
$S_{current} = S$.
end.

Scenario 1:
Choose $t_b = \%t_f \leftarrow$ ramp.
$q_0, q_f, t_f, \%ramp\ (t_b = \%t_f)$, (start, end positions, total time, % dedicated to parabolic blend, i.e., $t_b = \%t_f$)
First find a, V during constant accel, constant vel. period:

$$a = \frac{q_f - 2q_0}{t_f^2(\% - \%^2)}, \quad V = \frac{q_f - 2q_0}{t_f(1 - \%)}$$

↑
$t_b = \%t_f$

Scenario 2:

q_0, q_f given
V, t_b
t_f, a solve

7

tb

1) Given desired displacement, travel velocity and ramp, solve for tf, tb, a:

$$a = V/t_b$$
$$q(t_b) = q_b = q_0 + \frac{\alpha}{2} {t_b}^2 = \frac{V}{2} * ramp * t_f$$
$$t_b = t_f * ramp$$
$$q_f - 2q_b = t_f * (1 - 2ramp) * V$$

These lead to:

4 $$t_f = q_f/(V * (1 - ramp))$$
5 $$t_b = t_f * ramp$$
6 $$a = V/t_b$$

declare variables:

t, qf, t_b, t_f, a, V, ramp.

setup:

initialize RTI → Timer overflow interrupts.

set outputs to drive stepper

serial print.

loop
~~main~~: {

→ 4 5 6

while(1) {

}

}

Timer overflow interrupt: {

get current time: $t = t + \Delta t$ ← timer prescale.

4, 5, ~~6~~ → get current posi → q (cm) use 1, 2, 3

convert q to steps.

get Δsteps

→ Move stepper Δ steps (# steps needed since last time)

}

Page 9

Wednesday, April 19, 2017 1:30 PM

Sample code:

```
#include <stdtypes.h>
#include "rt_nonfinite.h"
#include "main.h"
#include "MCU.h"
#include "math.h"

__interrupt void RealTimeInterrupt( void );   // prototype for our RTI interrupt function

float  V_x, a_x, q_x, q_x_current, qo_x, qf_x, tb_x, tf_x, new_x, recieved_V_x,
recieved_x, previous_x, x_steps_per_unit, t_current_x;
float  V_y, a_y, q_y, q_y_current, qo_y, qf_y, tb_y, tf_y, new_y, recieved_V_y,
recieved_y, previous_y, y_steps_per_unit, t_current_y;
float  V_z, a_z, q_z, q_z_current, qo_z, qf_z, tb_z, tf_z, new_z, recieved_V_z,
recieved_z, previous_z, z_steps_per_unit, t_current_z;
float  V_e, a_e, q_e, q_e_current, qo_e, qf_e, tb_e, tf_e, new_e, recieved_V_e,
recieved_e, previous_e, e_steps_per_unit, t_current_e;

float ramp, t, over, recieved_V, L, V;

int i, ii, step_size, unit;
int dirx, step_x;
int diry, step_y;
int dirz, step_z;
int dire, step_e;

void main( void )
{

  DDRA=0xff;
  DDRH=0xff;
  DDRJ=0xff;
  DDRP=0xff;
  PLL_init();

  // setup the realtime interrupt
  RTICTL = 0x20;  //0b00011100; //0x28;
  CRGINT |= 0b10000000;
  EnableInterrupts;


 ///////////////////////
 /// INPUT PARAMETERS ///
 ///////////////////////
```

Page 10

Wednesday, April 19, 2017 1:30 PM

```
recieved_x=0;      // (units)
recieved_y=0;      // (units)
recieved_z=-1000.0;     // (units)
recieved_e=0;      // (units)

recieved_V=650.0;      // (units/min)
recieved_V_z=2000.0;   // (units/min)

ramp=0.2;           // (unitless)
unit=0;             // unit=1 is inches, unit=0 is milimeters
step_size=2;        // 1,2,4,8,16 step size options

//////////////////////////
// CALCULATED PARAMETERS //
//////////////////////////

 if(recieved_x>0){     // direction to move in x
  dirx=1;
  } else{
  dirx=-1;
  recieved_x=recieved_x*-1;
  }

 if(recieved_y>0){    // direction to move in y
  diry=1;
  } else{
  diry=-1;
  recieved_y=recieved_y*-1;
  }

 if(recieved_z>0){    // direction to move in z
  dirz=1;
  } else{
  dirz=-1;
  recieved_z=recieved_z*-1;
  }

 if(recieved_e>0){    // direction to move in e
  dire=1;
  } else{
  dire=-1;
  recieved_e=recieved_e*-1;
  }
```

Page 11

Wednesday, April 19, 2017 1:30 PM

```
recieved_V=recieved_V/60;      // (units/min)
recieved_V_z=recieved_V_z/60;   // (units/min)

L=sqrt(((recieved_x-previous_x)*(recieved_x-previous_x))+((recieved_y-
previous_y)*(recieved_y-previous_y)));

recieved_V_x=recieved_V*((recieved_x-previous_x)/L);  // (in/s)
recieved_V_y=recieved_V*((recieved_y-previous_y)/L);  // (in/s)



if(unit==1){
  x_steps_per_unit = 103;    // (steps/in)   ///////// **ATTENTION** RECHECK THESE
CONVERSIONS!! //////////////////
  y_steps_per_unit = 103;    // (steps/in)
  z_steps_per_unit = 4081.633;   // (steps/in)
  e_steps_per_unit =480;    // (steps/in)
}
else{
  x_steps_per_unit = 4;      // (steps/mm)
  y_steps_per_unit = 4;      // (steps/mm)
  z_steps_per_unit = 160.694;    // (steps/mm)
  e_steps_per_unit = 19;      // (steps/mm)  NOTE: inches/25.4
}

V_x = (recieved_V_x * x_steps_per_unit)*step_size;   // (steps/s)
V_y = (recieved_V_y * y_steps_per_unit)*step_size;   // (steps/s)
V_z = (recieved_V_z * z_steps_per_unit)*step_size;   // (steps/s)

qf_x = (recieved_x * x_steps_per_unit)*step_size;   // (steps)
qf_y = (recieved_y * y_steps_per_unit)*step_size;   // (steps)
qf_z = (recieved_z * z_steps_per_unit)*step_size;   // (steps)
qf_e = (recieved_e * e_steps_per_unit)*step_size;   // (steps)

qo_x=(previous_x * x_steps_per_unit)*step_size;    // (steps)
qo_y=(previous_y * y_steps_per_unit)*step_size;    // (steps)
qo_z=(previous_z * z_steps_per_unit)*step_size;    // (steps)
qo_e=(previous_e * e_steps_per_unit)*step_size;    // (steps)


tf_x=qf_x/(V_x*(1-ramp));  // (s)
tf_y=qf_y/(V_y*(1-ramp));  // (s)
tf_z=qf_z/(V_z*(1-ramp));  // (s)
tf_e=(tf_x+tf_y)/2;       // (s)
```

Page 12

Wednesday, April 19, 2017 1:30 PM

```
tb_x=ramp*tf_x;            // (s)
tb_y=ramp*tf_y;            // (s)
tb_z=ramp*tf_z;            // (s)
tb_e=ramp*tf_e;            // (s)

V_e=qf_e/(tf_e-tb_e);


a_x=V_x/tb_x; // (steps/s^2)
a_y=V_y/tb_y; // (steps/s^2)
a_z=V_z/tb_z; // (steps/s^2)
a_e=V_e/tb_e; // (steps/s^2)

q_x_current=0; // (steps)
q_y_current=0; // (steps)
q_z_current=0; // (steps)
q_e_current=0; // (steps)

t=0;   // (s)
t_current_z=0;
t_current_x=0;
t_current_y=0;
t_current_e=0;

i=0;
over=0;

  switch(step_size){
  case 1:
    PTP = 0b00000000; // Full step
    break;

  case 2:
    PTP = 0b00000001; // Half step
    break;

  case 4:
    PTP = 0b00000010;  // Quarter step
    break;

  case 8:
    PTP = 0b00000011; // eight step
    break;

   case 16:
    PTP = 0b00000111; // sixteeth step
```

Page 13

Wednesday, April 19, 2017 1:30 PM

```
    break;
    }

while(1) {

//////////////////////////////////////
// Determine what motor needs a step //
//  using Brensham's Line algorithm  //
//////////////////////////////////////

if (qf_z>q_z_current){
    step_z=1;
    }
    else{
         step_z=0;
         }

if(step_z==0){
      step_x=1;
      step_y=1;
      step_e=1;
}


  }
} // main (void)

////////////////////
//  RTI function  //
////////////////////
__interrupt void RealTimeInterrupt( void ){
t_current_x=t_current_x+(1.0/3906);  // Update time
t_current_y=t_current_y+(1.0/3906);  // Update time
t_current_z=t_current_z+(1.0/3906);  // Update time
t_current_e=t_current_e+(1.0/3906);  // Update time

//////////////////////
//// X Acceleration ///
//////////////////////


if(step_x==1){

    if (t_current_x<=tb_x){                // (s)
           q_x = qo_x+((a_x/2)*t_current_x*t_current_x); // (steps)
         }
```

Page 14

Wednesday, April 19, 2017 1:30 PM

```
        else if(t_current_x<=(tf_x-tb_x)){   // (s)
                q_x = ((a_x/2)*tb_x*tb_x)+V_x*(t_current_x-tb_x); // (steps)
        }

        else if(t_current_x<(tf_x)){
                q_x = qf_x-((a_x/2)*tf_x*tf_x)+a_x*tf_x*t_current_x-
((a_x/2)*t_current_x*t_current_x);  // (steps)
        }

        else {
        q_x = qf_x;
        }
}

//////////////////////
//// Y Acceleration ///
//////////////////////
if(step_y==1){
  if (t_current_y<=tb_y){               // (s)
        q_y = qo_y+((a_y/2)*t_current_y*t_current_y); // (steps)
        }

        else if(t_current_y<=(tf_y-tb_y)){   // (s)
                q_y = ((a_y/2)*tb_y*tb_y)+V_y*(t_current_y-tb_y); // (steps)
        }

        else if(t_current_y<(tf_y)){
                q_y = qf_y-((a_y/2)*tf_y*tf_y)+a_y*tf_y*t_current_y-
((a_y/2)*t_current_y*t_current_y);  // (steps)
        }

        else {
        q_y = qf_y;
        }
}

//////////////////////
//// Z Acceleration ///
//////////////////////
if(step_z==1){
   if (t_current_z<=tb_z){               // (s)
        q_z = qo_z+((a_z/2)*t_current_z*t_current_z); // (steps)
        }

        else if(t_current_z<=(tf_z-tb_z)){   // (s)
```

Page 15

Wednesday, April 19, 2017 1:30 PM

```
            q_z = ((a_z/2)*tb_z*tb_z)+V_z*(t_current_z-tb_z); // (steps)
        }

        else if(t_current_z<(tf_z)){
            q_z = qf_z-((a_z/2)*tf_z*tf_z)+a_z*tf_z*t_current_z-
((a_z/2)*t_current_z*t_current_z);  // (steps)
            t_current_x=0;
            t_current_y=0; // These timers start when z move is complete
            t_current_e=0;
        }

        else {
        q_z = qf_z;
        }
}

//////////////////////
//// E Acceleration ///
//////////////////////
if(step_e==1){
    if (t_current_e<=tb_e){                // (s)
        q_e = qo_e+((a_e/2)*t_current_e*t_current_e); // (steps)
        }

        else if(t_current_e<=(tf_e-tb_e)){   // (s)
            q_e = ((a_e/2)*tb_e*tb_e)+V_e*(t_current_e-tb_e); // (steps)
        }

        else if(t_current_e<(tf_e)){
            q_e = qf_e-((a_e/2)*tf_e*tf_e)+a_e*tf_e*t_current_e-
((a_e/2)*t_current_e*t_current_e);  // (steps)
        }

        else {
        q_e = qf_e;
        }
}
////////////////////////
/// Step the steppers ///
////////////////////////

if(step_z==1){               // Step in Z
    if(q_z >= q_z_current){
    if(dirz==1){
      PORTA=0b11011111;
      PORTA=0b11111111;
```

Page 16

Wednesday, April 19, 2017 1:30 PM

```
      }
      if(dirz==-1){
        PORTA=0b11001111;
        PORTA=0b11111111;
        }
       q_z_current++;
      }
  }
  if(step_y==1){              // Step in Y
      if(q_y >= q_y_current){
      if(diry==1){
        PORTA=0b11110111;
        PORTA=0b11111111;
      }
      if(diry==-1){
        PORTA=0b11110011;
        PORTA=0b11111111;
        }
       q_y_current++;
      }
    }

  if(step_x==1){             // Step in X
     if(q_x >= q_x_current){
     if(dirx == 1){
       PTH=0b00000011;
       PTH=0b00000001;
       }
     if(dirx == -1){
       PTH=0b00000010;
       PTH=0b00000000;
       }
       q_x_current++;
      }
   }

  if(step_e==1){              // Step in E
      if(q_e >= q_e_current){
      if(dire==1){
        PTJ=0b00000010;
        PTJ=0b00000000;
      }
      if(dire==-1){
        PTJ=0b00000011;
        PTJ=0b00000001;
        }
```

Page 17

Wednesday, April 19, 2017 1:30 PM

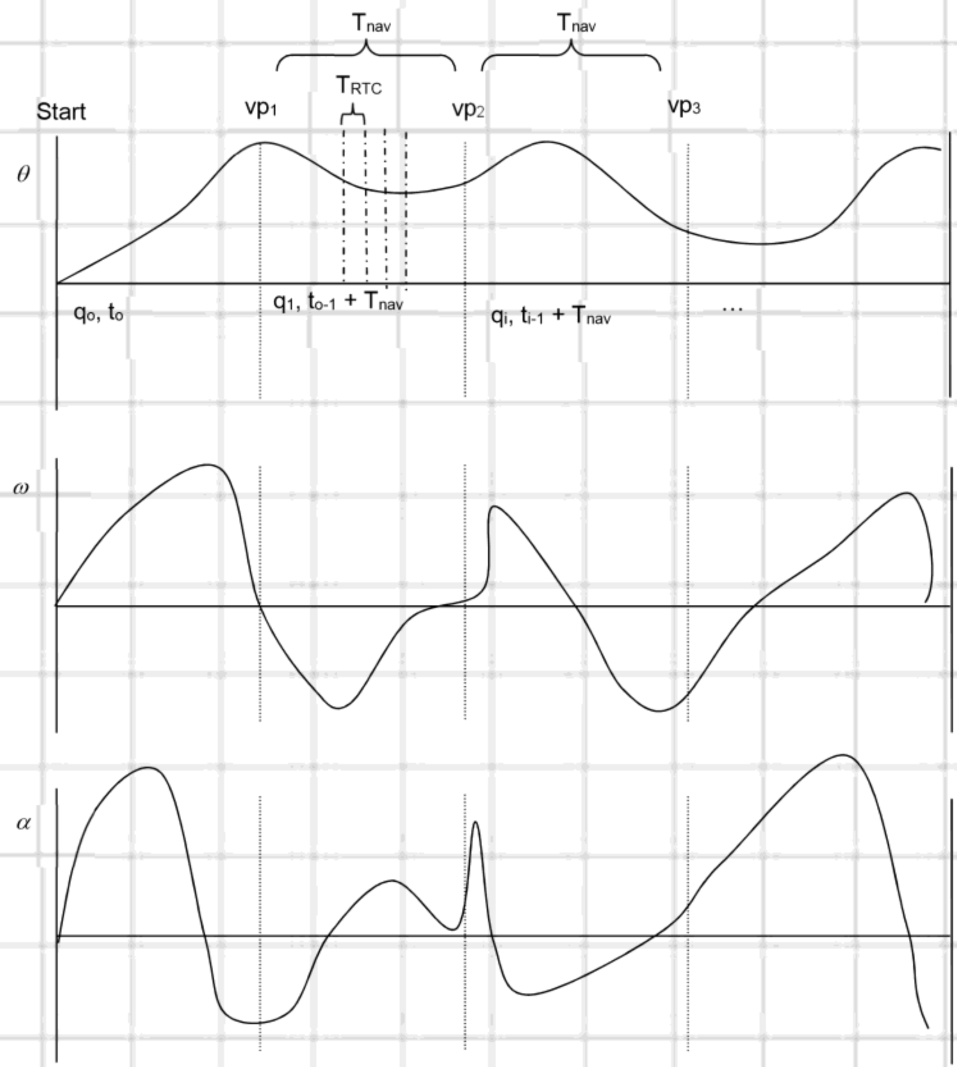
```
            q_e_current++;
        }
    }

    CRGFLG = 0b10000000; // clear RTI flag
}// end RTI function
```

Page 18

Wednesday, April 19, 2017 1:30 PM

Example: Real-Time Trajectory Generation Schemes:

For the initial implementation of the PAW prototype tracking a real-time trajectory (for example from a joystick), the cubic spline scheme will be used. The initial state is given and manipulator startup, while all future states are defined by via points which are generated in real-time by the input device (for example a joystick). The via points are generated at a rate or frequency defined by the rate at which navigation commands are updated, called ω_{nav} (T_{nav} as time between via points). This time is shown on figure 2. The current desired state of the manipulator is accessed by the manipulator controller at a rate of approx. 300-400 Hz (considered real time for most robotic applications). This rate is called ω_{RTC} (T_{RTC} between updated control signals).



Page 19

Wednesday, April 19, 2017 1:30 PM

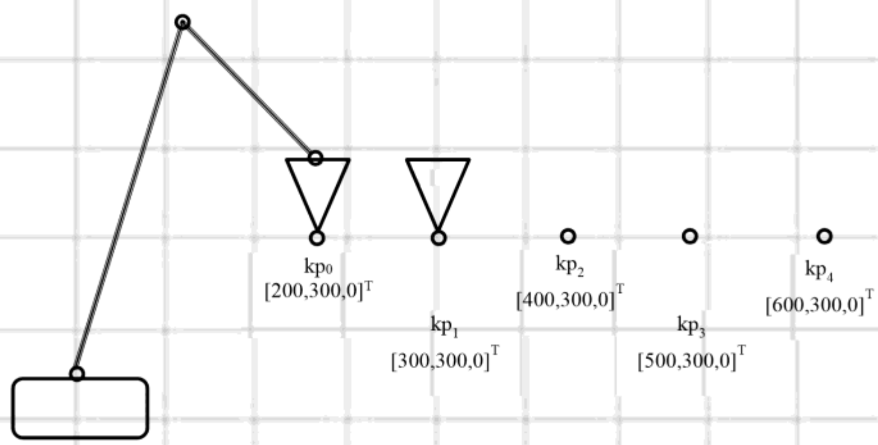
In this scheme, the manipulator controller will first generate the cubic polynomial (calculate coefficients) for the upcoming T_{nav} time period, and then calculate desired position and velocity information every T_{RTC} seconds for the control module. When a new via point is received, the current manipulator position is taken as the starting state and the new position the ending state for an upcoming segment. In this manner, the manipulator lag time should be limited to T_{nav}.

Page 20

Wednesday, April 19, 2017 1:30 PM

Example:
This example demonstrates trajectory synthesis with specified path and velocity. This example assumes there are $m = 5$ keypoints, and an $n=3$ dof manipulator.

Step 1: Define the general motion and task: Move along a horizontal line with end-effector pointing down, maintain constant velocity along path.



Step 2: Define keypoints as in figure above, with velocities defined as start/stop at rest, maintain 100cm/s velocity at internal keypoints.

$$KP = \begin{bmatrix} 200 & 300 & 400 & 500 & 600 \\ 300 & 300 & 300 & 300 & 300 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -90 & -90 & -90 & -90 & -90 \end{bmatrix}$$

$$VKP = \begin{bmatrix} 0 & 100 & 100 & 100 & 100 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Also, need to define times

$$\boldsymbol{t} = [0 \quad 1.1 \quad 2.1 \quad 3.1 \quad 4.1]$$

Step 3) Using Inverse kinematics, find the position, velocity at key points in joint space.

Page 21

Wednesday, April 19, 2017 1:30 PM

This leads to the following arrays:

$$\mathbf{q}_{kp} = \begin{bmatrix} \theta_{1,1} & \theta_{1,2} & \theta_{1,3} & \theta_{1,4} & \theta_{1,5} \\ \theta_{2,1} & \theta_{2,2} & \theta_{2,3} & \theta_{2,4} & \theta_{2,5} \\ \theta_{3,1} & \theta_{3,2} & \theta_{3,3} & \theta_{3,4} & \theta_{3,5} \end{bmatrix}$$

$$\dot{\mathbf{q}}_{kp} = \begin{bmatrix} \dot\theta_{1,1} & \dot\theta_{1,2} & \dot\theta_{1,3} & \dot\theta_{1,4} & \dot\theta_{1,5} \\ \dot\theta_{2,1} & \dot\theta_{2,2} & \dot\theta_{2,3} & \dot\theta_{2,4} & \dot\theta_{2,5} \\ \dot\theta_{3,1} & \dot\theta_{3,2} & \dot\theta_{3,3} & \dot\theta_{3,4} & \dot\theta_{3,5} \end{bmatrix}$$

With $\mathbf{q}_{\mathbf{kp}}$ solved from the inverse kinematic problem at each keypoint value, $\dot{\mathbf{q}}_{kp}$ solved from the Jacobian ($\mathbf{q}$(:,i) = $\mathbf{J}^{-1}\mathbf{Vkp}$(:,i)), i=1:*m* for *m* keypoints.)

Step 4) Define via points between point to point motion. (see Step 5)

Step 5)In joint space, map the trajectories as splines that satisfy the end conditions. For desirable motion, the spines must have finite jerk (be c2 constinuous)
For our problem, we have positions and velocities at the keypoints defined. Therefore, cubic splines are a natural fit. So solve for those:

$$\mathbf{A}_{kp}(:,:,j) = \begin{bmatrix} a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \end{bmatrix}$$

as

$$\mathbf{A}_{kp}(:,i,j) = \begin{bmatrix} 1 & \boldsymbol{t}(i) & \boldsymbol{t}^2(i) & \boldsymbol{t}^3(i) \\ 0 & 1 & 2\boldsymbol{t}(i) & 3\boldsymbol{t}^2(i) \\ 1 & \boldsymbol{t}(i+1) & \boldsymbol{t}^2(i+1) & \boldsymbol{t}^3(i+1) \\ 0 & 1 & 2\boldsymbol{t}(i+1) & 3\boldsymbol{t}^2(i+1) \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{q}_{\mathbf{kp}}(j,i) \\ \mathbf{q}_{\mathbf{kp}}(j,i+1) \\ \dot{\mathbf{q}}_{\mathbf{kp}}(j,i) \\ \dot{\mathbf{q}}_{\mathbf{kp}}(j,i+1) \end{bmatrix}$$

where j = 1:n (n = robot dof, 3 in this example), i = *1:m-1* for *m* keypoints.

Note in this setup, the array holding all of the cubic polynomial coefficients, $\mathbf{A}_{\mathbf{kp}}$ is a 3 dimensional array – 4 rows, *m* columns, and *n* layers.

Step 6) Define a real-time controller update rate (proposed here as 300 Hz).

Step 7) Access the mapped joint-space trajectories at the controller update-rate to get target position and velocity for each input.
For $\mathbf{t}_{\mathbf{kp}}(0) < t <= \mathbf{t}_{\mathbf{kp}}(1)$

$$\theta_1(t) = \mathbf{A}_{kp}(1,i,1) + \mathbf{A}_{kp}(2,i,1)t + \mathbf{A}_{kp}(3,i,1)t^2 + \mathbf{A}_{kp}(4,i,1)t^3$$
$$\dot\theta_1(t) = \mathbf{A}_{kp}(2,i,1) + 2\mathbf{A}_{kp}(3,i,1)t + 3\mathbf{A}_{kp}(4,i,1)t^2$$

$$\theta_2(t) = \mathbf{A}_{kp}(1,i,2) + \mathbf{A}_{kp}(2,i,2)t + \mathbf{A}_{kp}(3,i,2)t^2 + \mathbf{A}_{kp}(4,i,2)t^3$$

Page 22

Wednesday, April 19, 2017 1:30 PM

$$\dot{\theta}_2(t) = \mathbf{A}_{kp}(2,i,2) + 2\mathbf{A}_{kp}(3,i,2)t + 3\mathbf{A}_{kp}(4,i,2)t^2$$

$$\theta_3(t) = \mathbf{A}_{kp}(1,i,3) + \mathbf{A}_{kp}(2,i,3)t + \mathbf{A}_{kp}(3,i,3)t^2 + \mathbf{A}_{kp}(4,i,3)t^3$$
$$\dot{\theta}_3(t) = \mathbf{A}_{kp}(2,i,3) + 2\mathbf{A}_{kp}(3,i,2)t + 3\mathbf{A}_{kp}(4,i,3)t^2$$

For $\mathbf{t}_{\mathbf{kp}}(1) < t <= \mathbf{t}_{\mathbf{kp}}(2)$

$$\theta_1(t) = \mathbf{A}_{kp}(1{,}2{,}1) + \mathbf{A}_{kp}(2{,}2{,}1)t + \mathbf{A}_{kp}(3{,}2{,}1)t^2 + \mathbf{A}_{kp}(4{,}2{,}1)t^3$$
$$\dot{\theta}_1(t) = \mathbf{A}_{kp}(2{,}2{,}1) + 2\mathbf{A}_{kp}(3{,}2{,}1)t + 3\mathbf{A}_{kp}(4{,}2{,}1)t^2$$

And so on for the remaining segments.

Step 8) Pass these target position and velocity inputs to the joint-level PID controller to generate error signal (e, $\dot{e}$, $\int e$), $e = (\theta_d - \theta_a)$, $\dot{e} = (\omega_d - \omega_a)$, and corresponding control input, $u = k_p e + k_d \dot{e} + k_i \int e$

Step 9) Update the time, t, repeat steps 7 and 8 until $t = \mathbf{t}_{\mathbf{kp}}(m)$