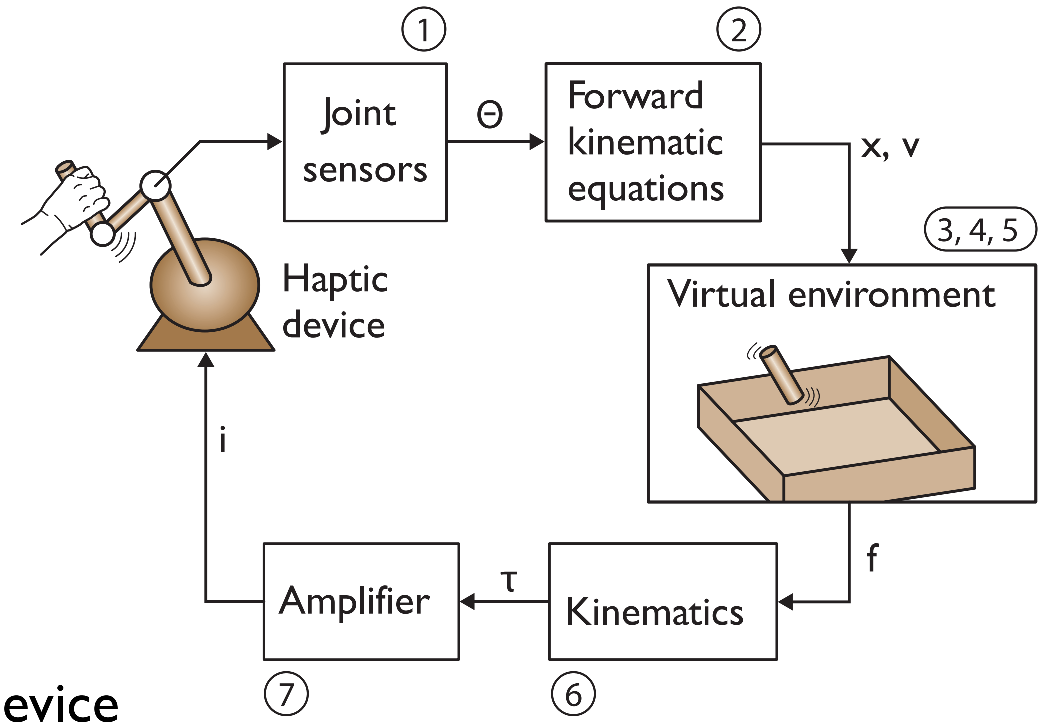
ME 20N: Haptics: Engineering Touch
Autumn 2017

Week 7:
2-D Haptic Rendering

Allison M. Okamura
Stanford University

2-D Rendering

The Haptic Loop



To begin, the user **moves** the haptic device

1. **Movement** of the device is sensed
2. **Kinematic equations** are used to find the motion of the haptic interaction point
3. If necessary, **contact** with object(s) in the virtual environment are detected
4. If necessary, the relevant point of the **surface** of the virtual object is detected
5. The **force** to be displayed to the user is calculated
6. Kinematics are used to determine **actuator commands**
7. An **amplifier** is used to send current/voltage to the actuator

The user **feels a force** from the haptic device

rendering (inside) a box

$$F_x = 0$$

$$F_y = 0$$

$$\text{if } x_{user} > x_{wall-max}$$

$$F_x = F_x + k(x_{wall-max} - x_{user})$$

$$\text{if } x_{user} < x_{wall-min}$$

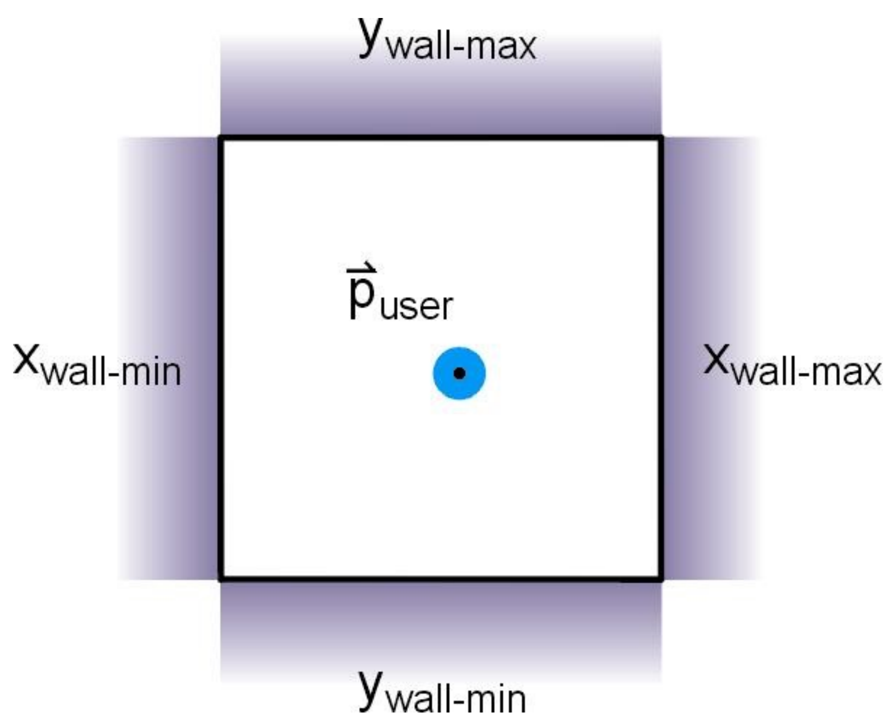
$$F_x = F_x + k(x_{wall-min} - x_{user})$$

$$\text{if } y_{user} > y_{wall-max}$$

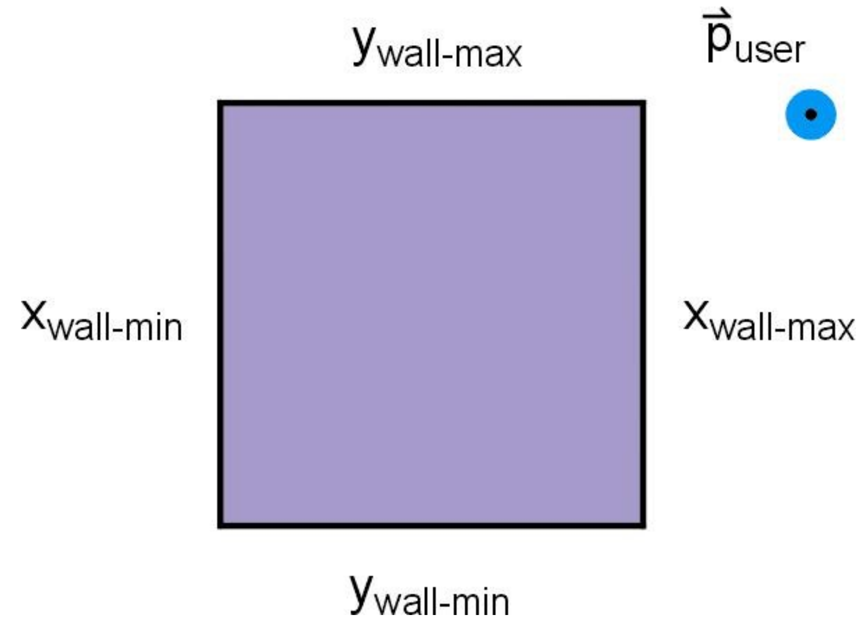
$$F_y = F_y + k(y_{wall-max} - y_{user})$$

$$\text{if } y_{user} < y_{wall-min}$$

$$F_y = F_y + k(y_{wall-min} - y_{user})$$



rendering (outside) a box



$$F = 0$$

$$\text{if } [(x_{user} < x_{wall-max}) \ \& (x_{user} > x_{wall-min})$$
$$\& (y_{user} < y_{wall-max}) \& (y_{user} > y_{wall-min})]$$

Then… what force should be displayed??

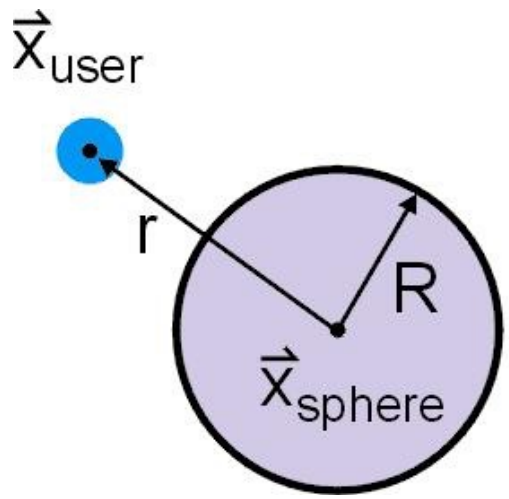
rendering (outside) a circle

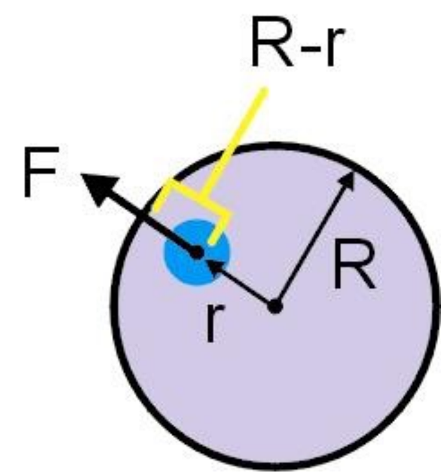
$$r = \sqrt{(x_{user} - x_{sphere})^2 + (y_{user} - y_{sphere})^2}$$

$$\hat{r} = \frac{1}{r}\left[\begin{array}{c} x_{user} - x_{sphere} \\ y_{user} - y_{sphere} \end{array}\right]$$

$$\text{if } \; r < R, \text{then } \; F = k(R - r)\hat{r}$$

$$\left[\begin{array}{c} force_x \\ force_y \end{array}\right] = \left[\begin{array}{c} k(R - r)(x_{user} - x_{sphere})/r \\ k(R - r)(y_{user} - y_{sphere})/r \end{array}\right]$$





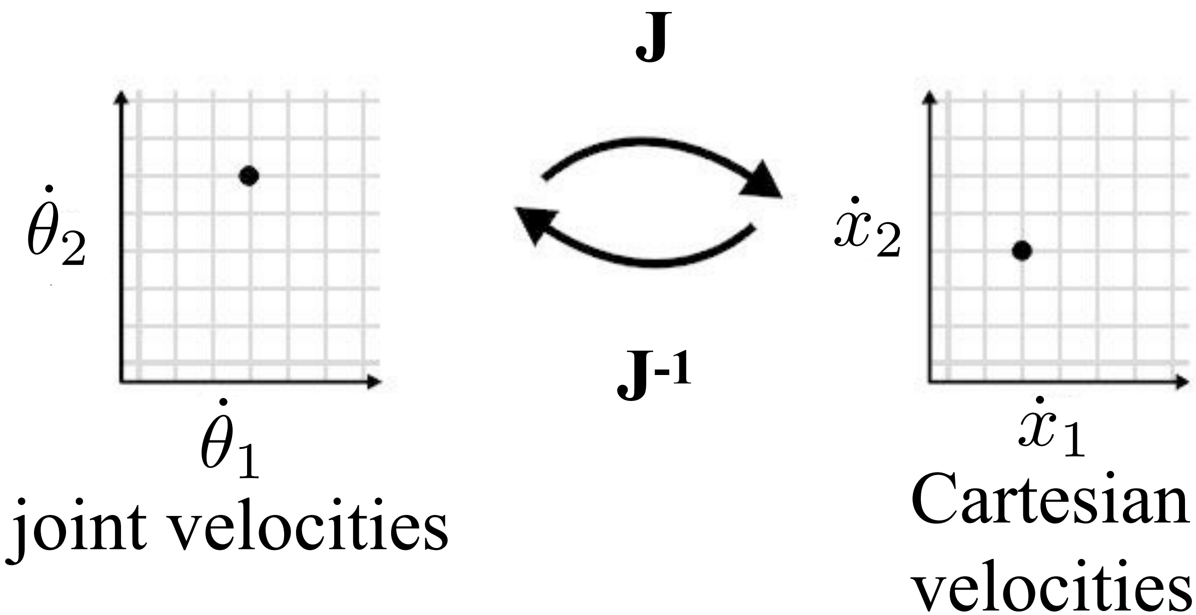
output the computed force

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = J^T \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

what is this magical J^T ?

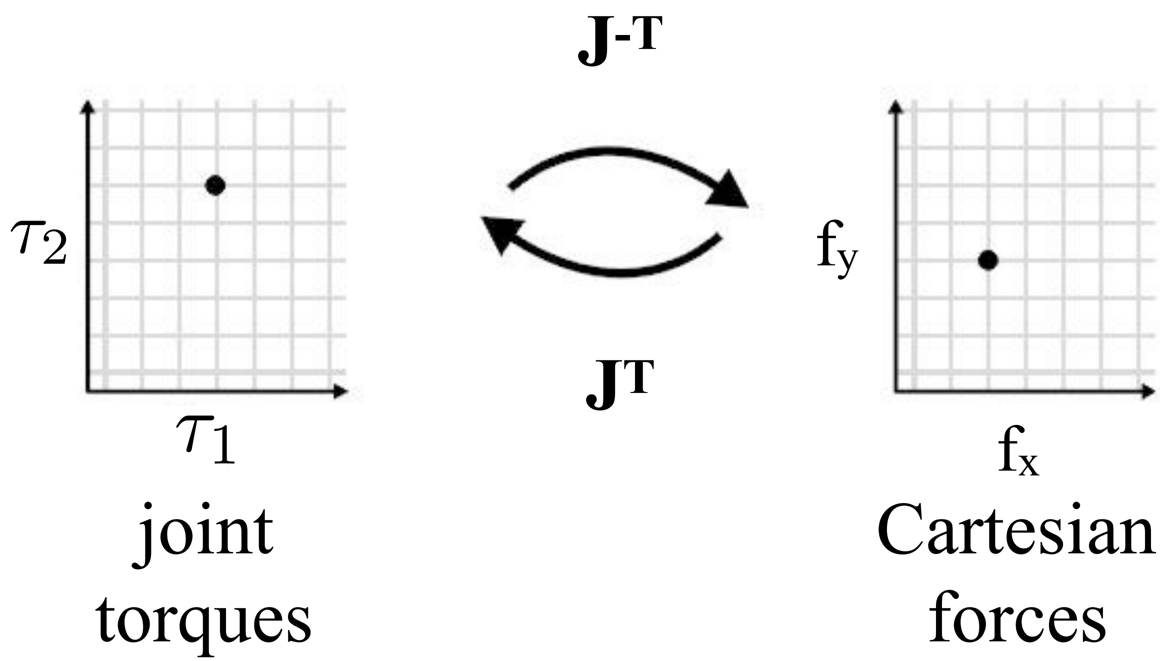
The Jacobian

Jacobian



The Jacobian is a matrix that can transform between joint velocities and Cartesian velocities

Jacobian



The Jacobian can also transform between joint torques and Cartesian forces

computing end-effector velocity

- forward kinematics tells us the endpoint *position* based on joint positions
- how do we calculate endpoint velocity from joint velocities?
- use the ***Jacobian*** matrix

$$\dot{x} = J\dot{\theta}$$

formulating the Jacobian

multidimensional form of the chain rule:

$$\dot{x} = \frac{\partial x}{\partial \theta_1}\dot{\theta}_1 + \frac{\partial x}{\partial \theta_2}\dot{\theta}_2 + \ldots$$

$$\dot{y} = \frac{\partial y}{\partial \theta_1}\dot{\theta}_1 + \frac{\partial y}{\partial \theta_2}\dot{\theta}_2 + \ldots$$

assemble in matrix form:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix}$$

$$\dot{x} = J\dot{\theta}$$

Singularities

- Many devices will have configurations at which the Jacobian is singular
- This means that the device has lost one or more degrees of freedom in Cartesian Space
- Two kinds:
  - Workspace boundary
  - Workspace interior

Singularity Math

- If the matrix is invertible, then it is non-singular.

$$\dot{\theta} = J^{-1}\dot{\mathbf{x}}$$

- Can check invertibility of J by taking the determinant of J. If the determinant is equal to 0, then J is singular.
- Can use this method to check which values of θ will cause singularities.

compute the necessary joint torques

the Jacobian can also be used to relate ***joint torques*** to ***end-effector forces***:

$$\boldsymbol{\tau} = J^T \mathbf{f}$$

this is a key equation for multi-degree-of-freedom haptic devices

how do you get this equation?

the **Principle of virtual work**

states that changing the coordinate frame does not change the total work of a system

$$\mathbf{f} \cdot \delta \mathbf{x} = \boldsymbol{\tau} \cdot \delta \mathbf{q}$$

$$\mathbf{f}^T \delta \mathbf{x} = \boldsymbol{\tau}^T \delta \mathbf{q}$$

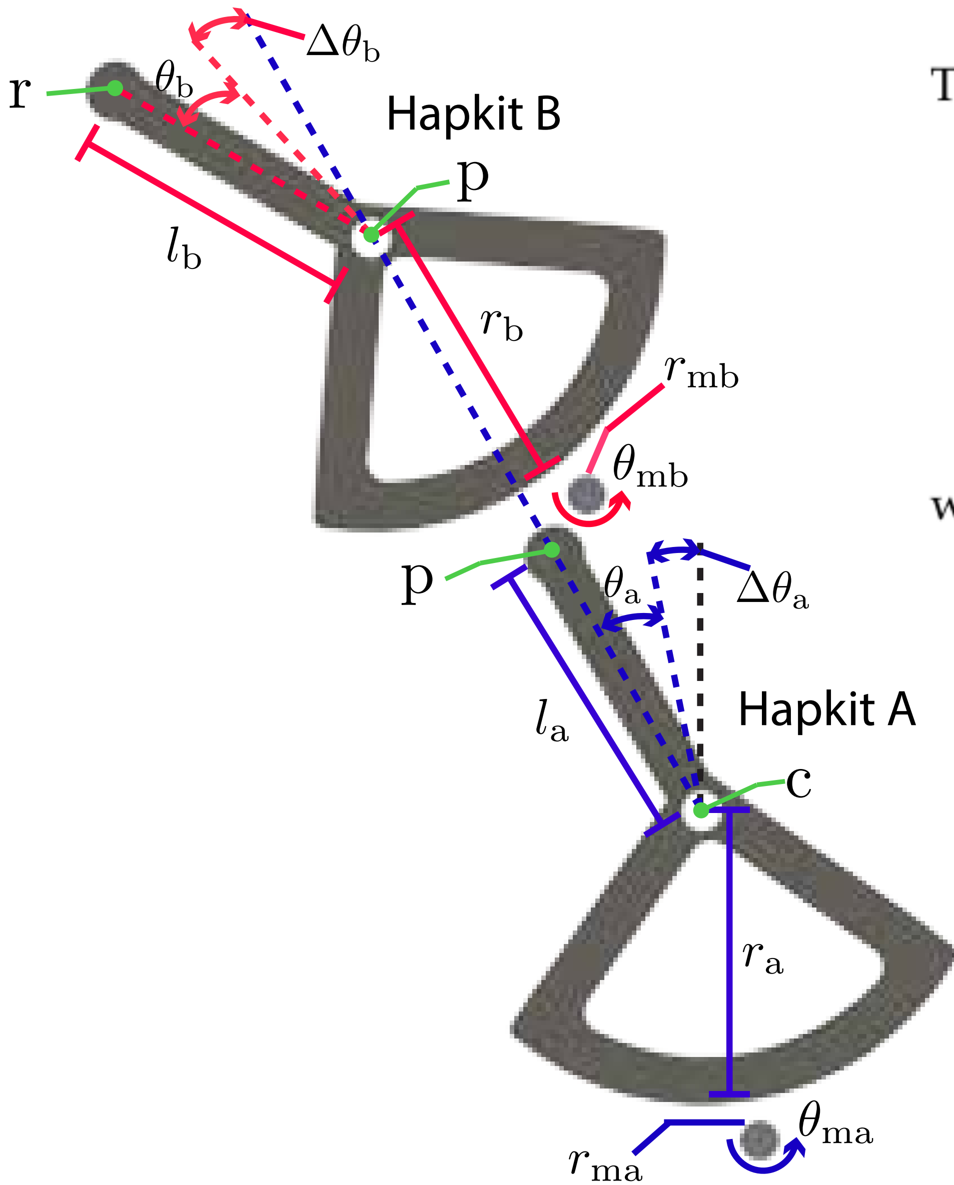
$$\mathbf{f}^T J \delta \mathbf{q} = \boldsymbol{\tau}^T \delta \mathbf{q}$$

$$\mathbf{f}^T J = \boldsymbol{\tau}^T$$

$$J^T \mathbf{f} = \boldsymbol{\tau}$$

Haplink

Forward Kinematics



The forward kinematic equations are:

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} -l_a \sin(\tilde{\theta}_a) + c_x \\ l_a \cos(\tilde{\theta}_a) + c_y \end{bmatrix}$$

$$\begin{bmatrix} r_x \\ r_y \end{bmatrix} = \begin{bmatrix} -l_b \sin(\tilde{\theta}_a + \tilde{\theta}_b) + p_x \\ l_b \cos(\tilde{\theta}_a + \tilde{\theta}_b) + p_y \end{bmatrix}$$

where:

$$\tilde{\theta}_a = \theta_a + \Delta\theta_a$$

$$\tilde{\theta}_b = \theta_b + \Delta\theta_b$$

$$\theta_{ma} = -\frac{r_a}{r_{ma}}\theta_a$$

$$\theta_{mb} = -\frac{r_b}{r_{mb}}\theta_b$$

Jacobian

$$J = \begin{bmatrix} \frac{\partial x}{\partial \tilde{\theta}_a} & \frac{\partial x}{\partial \tilde{\theta}_b} \\ \frac{\partial y}{\partial \tilde{\theta}_a} & \frac{\partial y}{\partial \tilde{\theta}_b} \end{bmatrix}$$

$$J = \begin{bmatrix} L_B \cos(\tilde{\theta}_b + \tilde{\theta}_a) - L_A \cos(\tilde{\theta}_a) & -L_B \cos(\tilde{\theta}_b + \tilde{\theta}_a) \\ -L_B \sin(\tilde{\theta}_b + \tilde{\theta}_a) - L_A \sin(\tilde{\theta}_a) & -L_B \sin(\tilde{\theta}_b + \tilde{\theta}_a) \end{bmatrix}$$

$$J = \begin{bmatrix} J00 & J01 \\ J10 & J11 \end{bmatrix}$$

inside the function `calculatePositionHandleAndJacobian`

```
// Compute the angle of the paddles in radians
theta_ma = (double)(getCountsSensor1())*2*3.1416/TOTAL_ENCODER_COUNTS;
theta_mb = (double)(getCountsSensor2())*2*3.1416/TOTAL_ENCODER_COUNTS;
theta_a = theta_ma*R_MA/R_A;
theta_b = -theta_mb*R_MA/R_A + THETA_B_OFFSET_RAD;

// Compute px and py
tildetheta_a = theta_a + DELTATHETA_A;
px = -(L_A *sin(tildetheta_a)) + CX;
py = L_A * cos(tildetheta_a) + CY;

//Compute rx and ry in n
tildetheta_b = theta_b + DELTATHETA_B;
rx =  -(L_B *sin(tildetheta_b + tildetheta_a))+ px;
ry =  L_B * cos(tildetheta_b + tildetheta_a)+ py;

//build the Jacobian
J00 = -L_B*cos(tildetheta_b + tildetheta_a) - L_A * cos(tildetheta_a);
J01 = -L_B*cos(tildetheta_b + tildetheta_a);
J10 = -L_B*sin(tildetheta_b + tildetheta_a) - L_A*sin(tildetheta_a);
J11 = -L_B*sin(tildetheta_b + tildetheta_a);
```

inside your haptic rendering function, for example

`haplinkForceOutput`

```
ForceX = 0.5;
ForceY = 0.5;

TorqueX = (J00*ForceX + J10*ForceY)*0.001;
TorqueY = (J01*ForceX + J11*ForceY)*0.001;

TorqueMotor1 = -((TorqueX*R_MA)/R_A);
TorqueMotor2 = -((TorqueY*R_MB)/R_B);

outputTorqueMotor1(TorqueMotor1);
outputTorqueMotor2(TorqueMotor2);
```

Changes to make in main.h and main.cpp

First, use the code for Haplink, not Hapkit:

```
// #define HAPKIT          1
 #define HAPLINK         2
```

Second, depending on the starting angle you use for Hapkit B, you might need to change the offset:

```
#define THETA_B_OFFSET   -40.0
```

WHY?

Call functions in the section `#else //then you are using HAPLINK`

```
calculatePositionHandleAndJacobian();
```