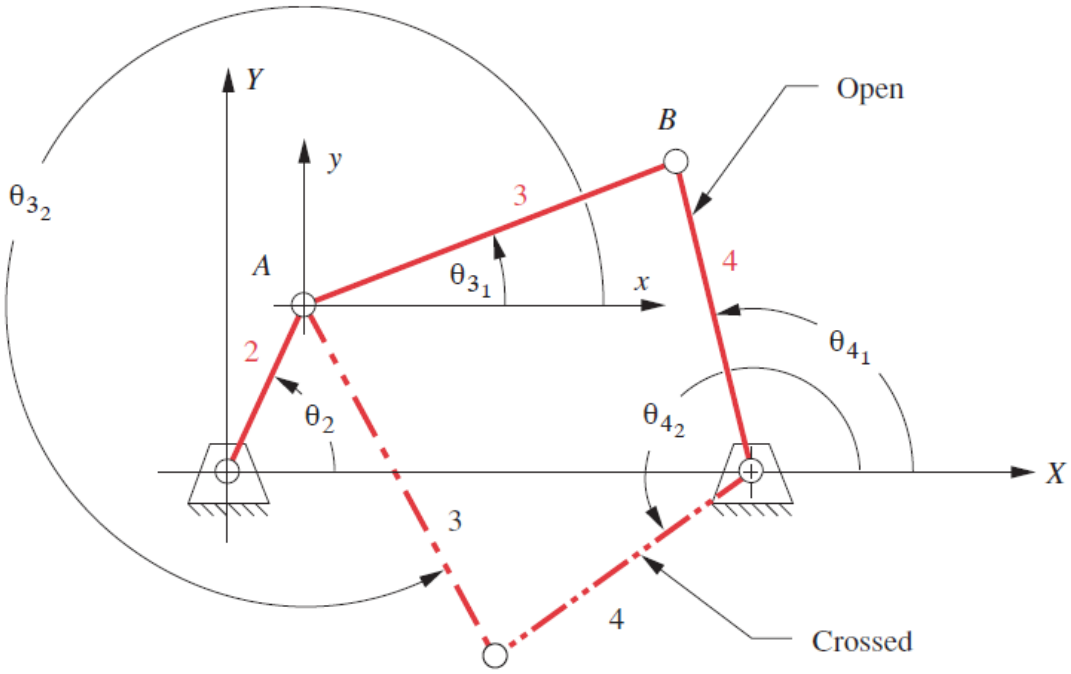
HOMEWORK #2



The value of θ_2 and the lengths of the links in the fourbar linkage shown above are as follows:

- L_1 = 7 inches
- L_2 = 4 inches
- L_3 = 5 inches
- L_4 = 6 inches
- θ_2 = 85 degrees

1a. (10 pts.) Using the notation given in Norton's book as shown above, determine θ_3 and θ_4 for both the open and closed configurations of the fourbar linkage. Use the position analysis equations and show your work. (Mathcad recommended)

1b. (10 pts.) Verify your results for both configurations using Norton's linkage software. Include screenshots of the linkage in both configurations and the relevant tabular results.

1c. (5 pts.) Is the fourbar linkage Grashof?

1d. (10 pts.) Make link 3 an equilateral triangle and show the path of the vertex that is not connected to one of the other links using the Norton linkage software for both open and crossed configurations.

2. (35 pts.) Creating a Chassis class using encoders

In class we looked at creating a class and a sample class was shown that had methods that would drive forward for some distance and turn for some number of degrees. In class the example used time to control the driving and turning. As we know, time is a poor method of controlling the robot operation because there are many variables that effect the speed that the motors might turn. For example, when driving, for a given effort and time, the result may vary depending on the:

- Charge on the batteries
- Slope the robot was driving up or down
- Surface the robot was driving on, for example, carpet vs hard floor

A much better solution is to use sensors to measure the rotations on the wheels. The motors on the Romi Chassis are equipped with encoders that measure 1440 counts per wheel rotation making it possible to get very precise movements.

The Chassis class

The definition of the Chassis class presented in class is shown below.

```
class Chassis
{
public:
    void driveDistance(float inches);
    void turnAngle(float degrees);


    const float wheelDiameter = 2.8;
    const int CPR = 1440;
    const float wheelTrack = 5.75;


private:
    Romi32U4Motors motors;
    Romi32U4Encoders encoders;

};
```

In this class there are 2 public methods, one to drive for some distance and another to turn for some number of degrees. This is just the class definition and is almost always saved in a header file (.h) for the class. The actual code to implement those methods is stored in a separate implementation file (.cpp). The implementation is shown below.

```
#include "Chassis.h"

/**
 * Assume the robot drives about 12 inches / second
 * Take the number of inches, divide by 12 and drive that long
 */
void Chassis::driveDistance(float inches)
{
    motors.setEfforts(100, 100);
    delay(inches / 12 * 1000);
    motors.setEfforts(0, 0);
}

/**
 * Assume the robot turns at about 180 degrees per second
 */
void Chassis::turnAngle(float degrees)
{
    motors.setEfforts(100, -100);
    delay(1.0 / 180 * degrees * 1000);
    motors.setEfforts(0, 0);
}
```

Notice that for each method (function) in the implementation, the method name is prefixed by the class name for example

```
Chassis::driveDistance(float inches)
```

This is required to identify this as a class method instead of a function. Class methods may access class data where normal functions can not.

Changing the class to use encoders

Ideally there would be a class definition for each subsystem on your robot. In this case the Chassis class would contain all the code that operates the robot chassis. Because all the chassis code is *encapsulated* inside the class it is easy to find everything that needs to be changed when the subsystem is improved. In this case, the encoders are used instead of the less repeatable timing. But since the rest of your program only uses class methods to interface with the chassis, then you are free to change those implementations without worrying about all the places that the chassis might be used.