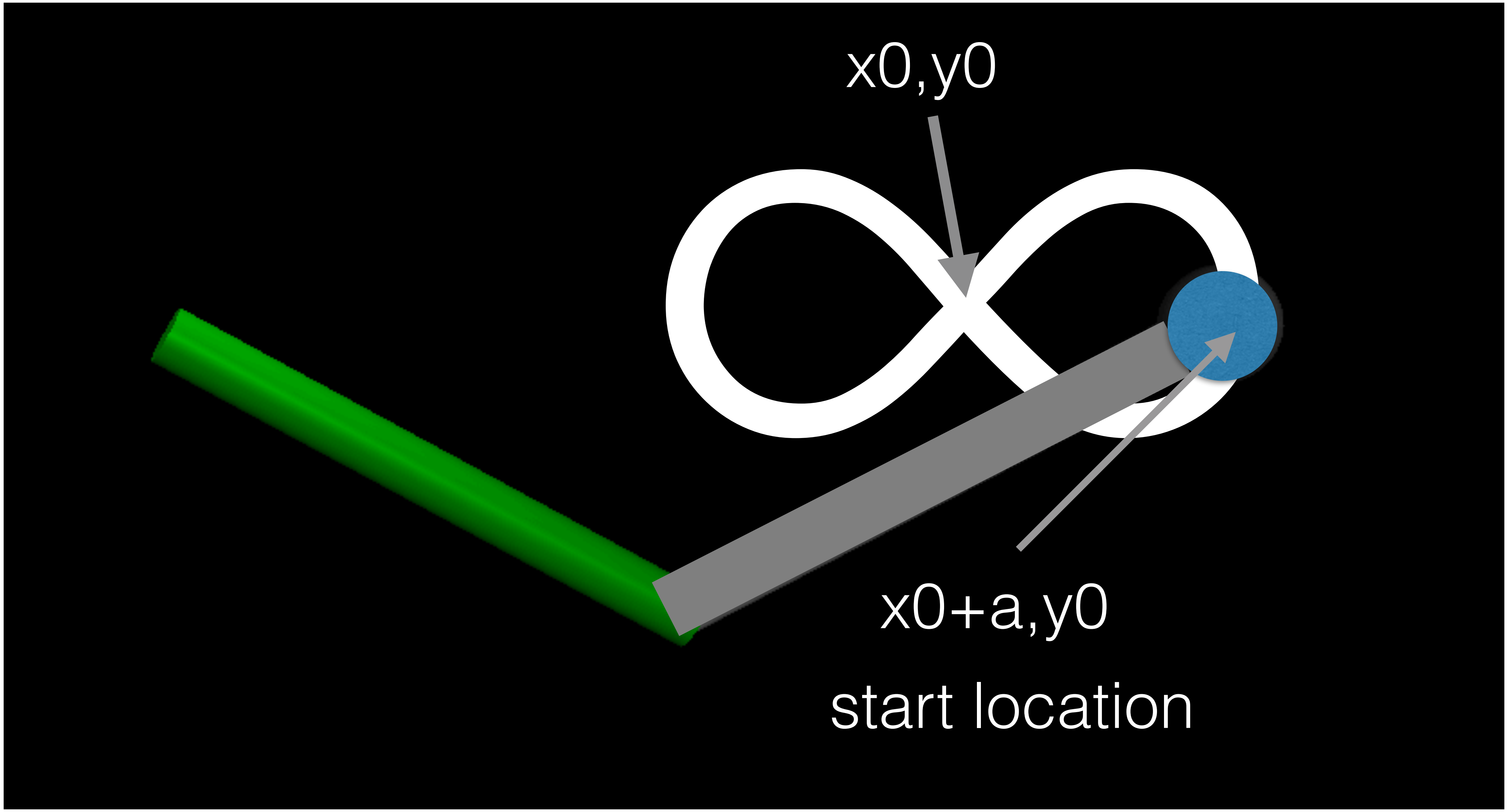
MuJoCo: Inverse Kinematics with Optimization (1)



MuJoCo: Inverse Kinematics using Optimization (2)*

Using template_manipulator.zip to get started

1. From tiny.cc/mujoco download template_manipulator.zip and unzip in myproject
2. Rename folder template_manipulator to manipulator_ik
3. Make these three changes
    1. main.c — line 28, change template_manipulator/ to manipulator_ik/
    2. makefile — change ROOT = template_writeData to ROOT = manipulator_ik also UNCOMMENT (del #) appropriate to your OS
    3. run_unix / run_win.bat change <template_manipulator> to < manipulator_ik>
4. In the shell, navigate to manipulator_ik and type ./run_unix (unix)

* I don't have instructions for Windows. For Windows, use Ubuntu via Virtualbox.

MuJoCo: Inverse Kinematics using Optimization (3)

1. Create a function simulator(Xin, Xout) where Xin is are the joint angles and Xout is the end-effector position
2. Incorporate constrained.c in the code. Include "constrained.c" in main.c and change main() function to inverse_kinematics() in constrained.c
3. Move simulator(Xin,Xout) to constrained.c. We will use two data structures: mjData* d; (data for robot) and mjData* dsim; (data for simulator)

MuJoCo: Inverse Kinematics using Optimization (4)

4. Modify inverse_kinematics to do optimization. Test initial pose (using init_controller).
5. Create the function curve for Xref.
6. Program init_controller to set the curve center and initial the pose
7. Program my_controller to do draw the curve
8. Save the data and plot in MATLAB.

MuJoCo: Inverse Kinematics using Optimization (5)

Internal vs. external callback

Use this for recursive calls (this tutorial)

```
init_controller(){..}

mycontroller() {…}


void main(){

mjcb_control = mycontroller

{

while( termination condition)

 {

 mj_step(m,d);

 }

 }
```

```
init_controller(){..}

my_controller() {…}


void main(){

{

while( termination condition)

 {

 my_controller();

 mj_step(m,d);

 }

}
```