In [1]:
```
import Pkg
Pkg.activate(@__DIR__)
Pkg.instantiate()
import FiniteDiff
import ForwardDiff as FD
import Convex as cvx
import ECOS
using LinearAlgebra
using Plots
using Random
using JLD2
using Test
using MeshCat
const mc = MeshCat
using StaticArrays
using Printf
```

```
Activating environment at `~/OCRL/HW4_S24/Project.toml`
  Updating registry at `~/.julia/registries/General`
 Installed MutableArithmetics ─ v1.4.2
 Installed Zstd_jll ─────────── v1.5.6+0
 Installed StatsBase ────────── v0.34.3
 Installed MathOptInterface ─── v1.27.1
 Installed HTTP ─────────────── v1.10.5
 Installed Plots ────────────── v1.40.3
 Installed XML2_jll ─────────── v2.12.6+0
 Installed OpenSSL_jll ──────── v3.0.13+1
 Installed GR_jll ───────────── v0.73.3+0
 Installed TranscodingStreams ─ v0.10.7
 Installed Contour ──────────── v0.6.3
 Installed GR ───────────────── v0.73.3
 Installed Format ───────────── v1.3.7
  Updating `~/OCRL/HW4_S24/Project.toml`
[5ae59095] + Colors v0.12.10
[f65535da] + Convex v0.15.4
[e2685f51] + ECOS v1.1.2
[6a86dc24] + FiniteDiff v2.22.0
[f6369f11] + ForwardDiff v0.10.36
[b6b21f68] + Ipopt v1.6.2
[033835bb] + JLD2 v0.4.46
[b8f27783] + MathOptInterface v1.27.1
[283c5d60] + MeshCat v0.16.1
[91a5bcdd] + Plots v1.40.3
[90137ffa] + StaticArrays v1.9.3
  Updating `~/OCRL/HW4_S24/Manifest.toml`
[14f7f29c] + AMD v0.5.3
[1520ce14] + AbstractTrees v0.4.5
[79e6a3ab] + Adapt v3.7.2
[4fba245c] + ArrayInterface v7.5.1
[6e4b80f9] + BenchmarkTools v1.5.0
[d1d4a3ce] + BitFlags v0.1.8
[fa961155] + CEnum v0.5.0
[7057c7e9] + Cassette v0.3.13
[d360d2e6] + ChainRulesCore v1.23.0
[9e997f8a] + ChangesOfVariables v0.1.8
[523fee87] + CodecBzip2 v0.8.2
[944b1d66] + CodecZlib v0.7.4
[35d6a980] + ColorSchemes v3.24.0
[3da002f7] + ColorTypes v0.11.4
[c3611d14] + ColorVectorSpace v0.10.0
[5ae59095] + Colors v0.12.10
[bbf7d656] + CommonSubexpressions v0.3.0
[34da2185] + Compat v4.14.0
[f0e56b4a] + ConcurrentUtilities v2.4.1
[187b0558] + ConstructionBase v1.5.5
[d38c429a] + Contour v0.6.3
[f65535da] + Convex v0.15.4
[150eb455] + CoordinateTransformations v0.6.3
[9a962f9c] + DataAPI v1.16.0
[864edb3b] + DataStructures v0.18.18
[e2d170a0] + DataValueInterfaces v1.0.0
[163ba53b] + DiffResults v1.1.0
[b552c78f] + DiffRules v1.15.1
```

```
[ffbed154] + DocStringExtensions v0.9.3
[e2685f51] + ECOS v1.1.2
[460bff9d] + ExceptionUnwrapping v0.1.10
[411431e0] + Extents v0.1.2
[c87230d0] + FFMPEG v0.4.1
[5789e2e9] + FileIO v1.16.3
[6a86dc24] + FiniteDiff v2.22.0
[53c48c17] + FixedPointNumbers v0.8.4
[1fa38f19] + Format v1.3.7
[f6369f11] + ForwardDiff v0.10.36
[46192b85] + GPUArraysCore v0.1.5
[28b8d3ca] + GR v0.73.3
[cf35fbd7] + GeoInterface v1.3.3
[5c1252a2] + GeometryBasics v0.4.10
[42e2da0e] + Grisu v1.0.2
[cd3eb016] + HTTP v1.10.5
[3587e190] + InverseFunctions v0.1.13
[b6b21f68] + Ipopt v1.6.2
[92d709cd] + IrrationalConstants v0.2.2
[c8e1da08] + IterTools v1.4.0
[82899510] + IteratorInterfaceExtensions v1.0.0
[033835bb] + JLD2 v0.4.46
[1019f520] + JLFzf v0.1.7
[692b3bcd] + JLLWrappers v1.5.0
[682c06a0] + JSON v0.21.4
[40e66cde] + LDLFactorizations v0.10.1
[b964fa9f] + LaTeXStrings v1.3.1
[23fbe1c1] + Latexify v0.16.2
[2ab3a3ac] + LogExpFunctions v0.3.27
[e6f89c97] + LoggingExtras v1.0.3
[1914dd2f] + MacroTools v0.5.13
[b8f27783] + MathOptInterface v1.27.1
[739be429] + MbedTLS v1.1.9
[442fdcdd] + Measures v0.3.2
[283c5d60] + MeshCat v0.16.1
[e1d29d7a] + Missings v1.1.0
[99f44e22] + MsgPack v1.2.1
[d8a4904e] + MutableArithmetics v1.4.2
[77ba4419] + NaNMath v1.0.2
[4d8831e6] + OpenSSL v1.4.2
[bac558e1] + OrderedCollections v1.6.3
[d96e819e] + Parameters v0.12.3
[69de0a69] + Parsers v2.8.1
[b98c9c47] + Pipe v1.3.0
[ccf2f8ad] + PlotThemes v3.1.0
[995b91a9] + PlotUtils v1.4.1
[91a5bcdd] + Plots v1.40.3
[aea7be01] + PrecompileTools v1.2.1
[21216c6a] + Preferences v1.4.3
[94ee1d12] + Quaternions v0.7.6
[c1ae055f] + RealDot v0.1.0
[3cdcf5f2] + RecipesBase v1.3.4
[01d81517] + RecipesPipeline v0.6.12
[189a3867] + Reexport v1.2.2
[05181044] + RelocatableFolders v1.0.1
[ae029012] + Requires v1.3.0
```

```
[6038ab10] + Rotations v1.7.0
[6c6a2e73] + Scratch v1.2.1
[efcf1570] + Setfield v1.1.1
[992d4aef] + Showoff v1.0.3
[777ac1f9] + SimpleBufferStream v1.1.0
[a2af1166] + SortingAlgorithms v1.2.1
[276daf66] + SpecialFunctions v2.3.1
[90137ffa] + StaticArrays v1.9.3
[1e83bf80] + StaticArraysCore v1.4.2
[82ae8749] + StatsAPI v1.7.0
[2913bbd2] + StatsBase v0.34.3
[09ab397b] + StructArrays v0.6.18
[3783bdb8] + TableTraits v1.0.1
[bd369af6] + Tables v1.11.1
[62fd8b95] + TensorCore v0.1.1
[3bb67fe8] + TranscodingStreams v0.10.7
[5c2747f8] + URIs v1.5.1
[3a884ed6] + UnPack v1.0.2
[1cfade01] + UnicodeFun v0.4.1
[1986cc42] + Unitful v1.19.0
[45397f5d] + UnitfulLatexify v1.6.3
[41fe7b60] + Unzip v0.1.2
[ae81ac8f] + ASL_jll v0.1.3+0
[6e34b625] + Bzip2_jll v1.0.8+1
[83423d85] + Cairo_jll v1.18.0+1
[c2c64177] + ECOS_jll v200.0.800+0
[5ae413db] + EarCut_jll v2.2.4+0
[2702e6a9] + EpollShim_jll v0.0.20230411+0
[2e619515] + Expat_jll v2.5.0+0
[b22a6f82] + FFMPEG_jll v4.4.4+1
[a3f928ae] + Fontconfig_jll v2.13.93+0
[d7e528f0] + FreeType2_jll v2.13.1+0
[559328eb] + FriBidi_jll v1.0.10+0
[0656b61e] + GLFW_jll v3.3.9+0
[d2c73de3] + GR_jll v0.73.3+0
[78b55507] + Gettext_jll v0.21.0+0
[7746bdde] + Glib_jll v2.80.0+0
[3b182d85] + Graphite2_jll v1.3.14+0
[2e76f6c2] + HarfBuzz_jll v2.8.1+1
[9cc047cb] + Ipopt_jll v300.1400.400+0
[aacddb02] + JpegTurbo_jll v3.0.2+0
[c1c5ebd0] + LAME_jll v3.100.1+0
[88015f11] + LERC_jll v3.0.0+1
[1d63c593] + LLVMOpenMP_jll v15.0.7+0
[dd4b983a] + LZO_jll v2.10.1+0
[e9f186c6] + Libffi_jll v3.2.2+1
[d4300ac3] + Libgcrypt_jll v1.8.7+0
[7e76a0d4] + Libglvnd_jll v1.6.0+0
[7add5ba3] + Libgpg_error_jll v1.42.0+0
[94ce4f54] + Libiconv_jll v1.17.0+0
[4b2f31a3] + Libmount_jll v2.39.3+0
[89763e89] + Libtiff_jll v4.5.1+1
[38a345b3] + Libuuid_jll v2.39.3+1
[d00139f3] + METIS_jll v5.1.2+0
[d7ed1dd3] + MUMPS_seq_jll v5.4.1+0
[e7412a2a] + Ogg_jll v1.3.5+1
```

```
[656ef2d0] + OpenBLAS32_jll v0.3.12+1
[458c3c95] + OpenSSL_jll v3.0.13+1
[efe28fd5] + OpenSpecFun_jll v0.5.5+0
[91d4177d] + Opus_jll v1.3.2+0
[30392449] + Pixman_jll v0.42.2+0
[c0090381] + Qt6Base_jll v6.5.3+1
[a44049a8] + Vulkan_Loader_jll v1.3.243+0
[a2964d1f] + Wayland_jll v1.21.0+1
[2381bf8a] + Wayland_protocols_jll v1.31.0+0
[02c8fc9c] + XML2_jll v2.12.6+0
[aed1982a] + XSLT_jll v1.1.34+0
[ffd25f8a] + XZ_jll v5.4.6+0
[f67eecfb] + Xorg_libICE_jll v1.0.10+1
[c834827a] + Xorg_libSM_jll v1.2.3+0
[4f6342f7] + Xorg_libX11_jll v1.8.6+0
[0c0b7dd1] + Xorg_libXau_jll v1.0.11+0
[935fb764] + Xorg_libXcursor_jll v1.2.0+4
[a3789734] + Xorg_libXdmcp_jll v1.1.4+0
[1082639a] + Xorg_libXext_jll v1.3.4+4
[d091e8ba] + Xorg_libXfixes_jll v5.0.3+4
[a51aa0fd] + Xorg_libXi_jll v1.7.10+4
[d1454406] + Xorg_libXinerama_jll v1.1.4+4
[ec84b674] + Xorg_libXrandr_jll v1.5.2+4
[ea2f1a96] + Xorg_libXrender_jll v0.9.10+4
[14d82f49] + Xorg_libpthread_stubs_jll v0.1.1+0
[c7cfdc94] + Xorg_libxcb_jll v1.15.0+0
[cc61e674] + Xorg_libxkbfile_jll v1.1.2+0
[e920d4aa] + Xorg_xcb_util_cursor_jll v0.1.4+0
[12413925] + Xorg_xcb_util_image_jll v0.4.0+1
[2def613f] + Xorg_xcb_util_jll v0.4.0+1
[975044d2] + Xorg_xcb_util_keysyms_jll v0.4.0+1
[0d47668e] + Xorg_xcb_util_renderutil_jll v0.3.9+1
[c22f9ab0] + Xorg_xcb_util_wm_jll v0.4.1+1
[35661453] + Xorg_xkbcomp_jll v1.4.6+0
[33bec58e] + Xorg_xkeyboard_config_jll v2.39.0+0
[c5fb5394] + Xorg_xtrans_jll v1.5.0+0
[3161d3a3] + Zstd_jll v1.5.6+0
[35ca27e7] + eudev_jll v3.2.9+0
[214eeab7] + fzf_jll v0.43.0+0
[1a1c6b14] + gperf_jll v3.1.1+0
[a4ae2306] + libaom_jll v3.4.0+0
[0ac62f75] + libass_jll v0.15.1+0
[2db6ffa8] + libevdev_jll v1.11.0+0
[f638f0a6] + libfdk_aac_jll v2.0.2+0
[36db933b] + libinput_jll v1.18.0+0
[b53b4c65] + libpng_jll v1.6.43+1
[f27f6e37] + libvorbis_jll v1.3.7+1
[009596ad] + mtdev_jll v1.1.6+0
[1270edf5] + x264_jll v2021.5.5+0
[dfaa095f] + x265_jll v3.5.0+0
[d8fb68d0] + xkbcommon_jll v1.4.1+1
[0dad84c5] + ArgTools
[56f22d72] + Artifacts
[2a0f44e3] + Base64
[ade2ca70] + Dates
[8bb1440f] + DelimitedFiles
```

```
  [f43a241f] + Downloads
  [9fa8497b] + Future
  [b77e0a4c] + InteractiveUtils
  [b27032c2] + LibCURL
  [76f85450] + LibGit2
  [8f399da3] + Libdl
  [37e2e46d] + LinearAlgebra
  [56ddb016] + Logging
  [d6f4376e] + Markdown
  [a63ad114] + Mmap
  [ca575930] + NetworkOptions
  [44cfe95a] + Pkg
  [de0858da] + Printf
  [9abbd945] + Profile
  [3fa0cd96] + REPL
  [9a3f8284] + Random
  [ea8e919c] + SHA
  [9e88b42a] + Serialization
  [6462fe0b] + Sockets
  [2f01184e] + SparseArrays
  [10745b16] + Statistics
  [4607b0f0] + SuiteSparse
  [fa267f1f] + TOML
  [a4e569a6] + Tar
  [8dfed614] + Test
  [cf7118a7] + UUIDs
  [4ec0a83e] + Unicode
  [e66e0078] + CompilerSupportLibraries_jll
  [deac9b47] + LibCURL_jll
  [29816b5a] + LibSSH2_jll
  [c8ffd9c3] + MbedTLS_jll
  [14a3606d] + MozillaCACerts_jll
  [4536629a] + OpenBLAS_jll
  [05823500] + OpenLibm_jll
  [efcefdf7] + PCRE2_jll
  [bea87d4a] + SuiteSparse_jll
  [83775a58] + Zlib_jll
  [8e850ede] + nghttp2_jll
  [3f19e933] + p7zip_jll
[ Info: Listening on: 127.0.0.1:8700, thread id: 1
┌ Info: MeshCat server started. You can open the visualizer by visiting the
following URL in your browser:
└ http://127.0.0.1:8700
[ Info: Server on 127.0.0.1:8700 closing
[ Info: MeshCat server closed.
```

In [2]:
```
include(joinpath(@__DIR__, "utils","ilc_visualizer.jl"))
```

```
update_car_pose! (generic function with 1 method)
```

# Q1: Iterative Learning Control (ILC) (40 pts)

In this problem, you will use ILC to generate a control trajectory for a Car as it swerves to avoid a moose, also known as "the moose test" (wikipedia, video). We will model the dynamics of the car as with a simple nonlinear bicycle model, with the following state and control:

$$x = \begin{bmatrix} p_x \\ p_y \\ \theta \\ \delta \\ v \end{bmatrix}, \qquad u = \begin{bmatrix} a \\ \dot{\delta} \end{bmatrix} \tag{1}$$

where $p_x$ and $p_y$ describe the 2d position of the bike, $\theta$ is the orientation, $\delta$ is the steering angle, and $v$ is the velocity. The controls for the bike are acceleration $a$, and steering angle rate $\dot{\delta}$.

In [3]:

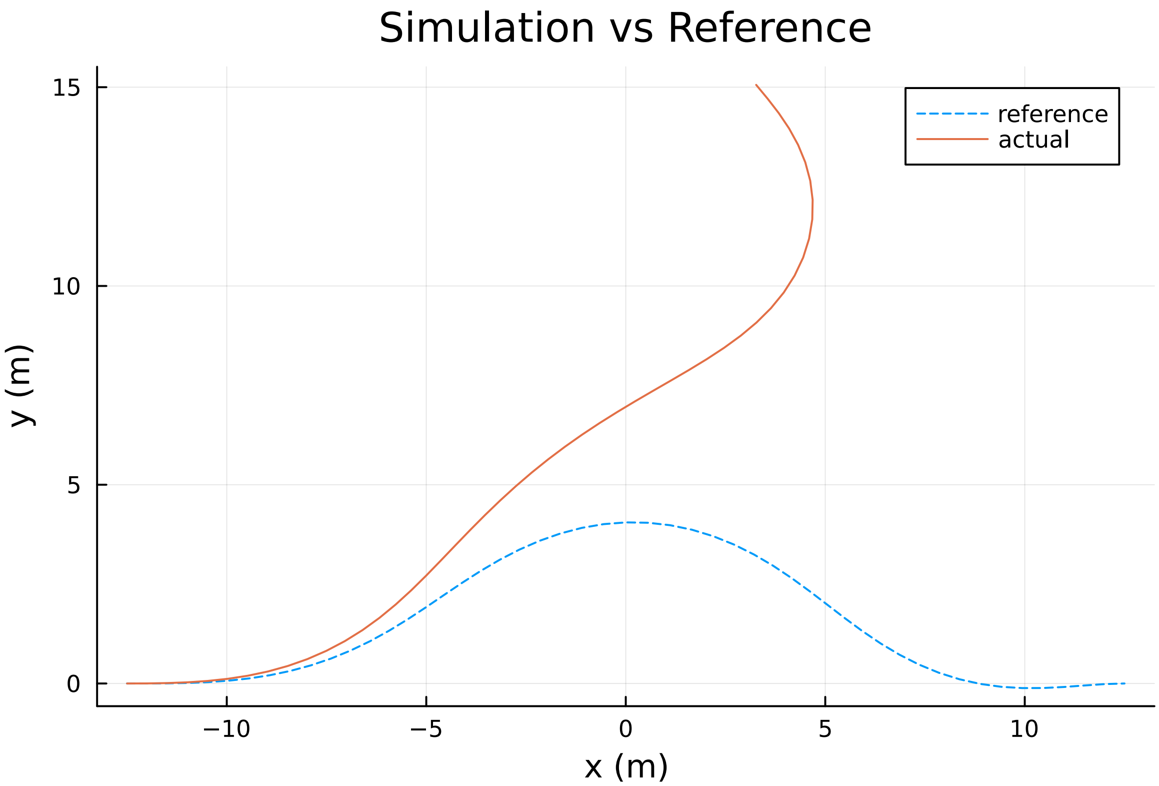
```
function estimated_car_dynamics(model::NamedTuple, x::Vector, u::Vector)::Ve
    # nonlinear bicycle model continuous time dynamics
    px, py, θ, δ, v = x
    a, δdot = u

    β = atan(model.lr * δ, model.L)
    s,c = sincos(θ + β)
    ω = v*cos(β)*tan(δ) / model.L

    vx = v*c
    vy = v*s

    xdot = [
        vx,
        vy,
        ω,
        δdot,
        a
    ]

    return xdot
end
function rk4(model::NamedTuple, ode::Function, x::Vector, u::Vector, dt::Rea
    k1 = dt * ode(model, x,        u)
    k2 = dt * ode(model, x + k1/2, u)
    k3 = dt * ode(model, x + k2/2, u)
    k4 = dt * ode(model, x + k3,   u)
    return x + (1/6)*(k1 + 2*k2 + 2*k3 + k4)
end
```

```
rk4 (generic function with 1 method)
```

We have computed an optimal trajectory $X_{ref}$ and $U_{ref}$ for a moose test trajectory offline using this `estimated_car_dynamics` function. Unfortunately, this is a highly approximate dynamics model, and when we run $U_{ref}$ on the car, we get a very different trajectory than we expect. This is caused by a significant sim to real gap. Here we will show what happens when we run these controls on the true dynamics:

In [4]:
```
function load_car_trajectory()
    # load in trajectory we computed offline
    path = joinpath(@__DIR__, "utils","init_control_car_ilc.jld2")
    F = jldopen(path)
    Xref = F["X"]
    Uref = F["U"]
    close(F)
    return Xref, Uref
end
function true_car_dynamics(model::NamedTuple, x::Vector, u::Vector)::Vector
    # true car dynamics
    px, py, θ, δ, v = x
    a, δdot = u

    # sluggish controls (not in the approximate version)
    a = 0.9*a - 0.1
    δdot = 0.9*δdot - .1*δ + .1

    β = atan(model.lr * δ, model.L)
    s,c = sincos(θ + β)
    ω = v*cos(β)*tan(δ) / model.L

    vx = v*c
    vy = v*s

    xdot = [
        vx,
        vy,
        ω,
        δdot,
        a
    ]

    return xdot
end

@testset "sim to real gap" begin
    # problem size
    nx = 5
    nu = 2
    dt = 0.1
    tf = 5.0
    t_vec = 0:dt:tf
    N = length(t_vec)
    model = (L = 2.8, lr = 1.6)

    # optimal trajectory computed offline with approximate model
    Xref, Uref = load_car_trajectory()

    # TODO: simulated Uref with the true car dynamics and store the states i
    Xsim = [zeros(nx) for i = 1:N]
    Xsim[1] = Xref[1]

    for k = 1:N-1
        Xsim[k+1] = rk4(model,true_car_dynamics,Xsim[k], Uref[k], dt)
    end
```

```
    # --------testing-----------------
    @test norm(Xsim[1] - Xref[1]) == 0
    @test norm(Xsim[end] - [3.26801052, 15.0590156, 2.0482790, 0.39056168, 4

    # --------plotting/animation-------
    Xm= hcat(Xsim...)
    Xrefm = hcat(Xref...)
    plot(Xrefm[1,:], Xrefm[2,:], ls = :dash, label = "reference",
         xlabel = "x (m)", ylabel = "y (m)", title = "Simulation vs Referenc
    display(plot!(Xm[1,:], Xm[2,:], label = "actual"))

end
```




```
Test Summary:    | Pass  Total
sim to real gap |    2      2
Test.DefaultTestSet("sim to real gap", Any[], 2, false, false)
```

In order to account for this, we are going to use ILC to iteratively correct our control until we converge.

To encourage the trajectory of the bike to follow the reference, the objective value for this problem is the following:

$$J(X,U) = \sum_{i=1}^{N-1} \left[ \frac{1}{2}(x_i - x_{ref,i})^T Q (x_i - x_{ref,i}) + \frac{1}{2}(u_i - u_{ref,i})^T R (u_i - u_{ref,i}) \right] + \frac{1}{2}(x_N - x_{ref,N})^T Q_f (x_N - x_{ref,N})$$

Using ILC as described in Lecture 18, we are to linearize our approximate dynamics model about $X_{ref}$ and $U_{ref}$ to get the following Jacobians:

$$A_k = \frac{\partial f}{\partial x}\Big|_{x_{ref,k}, u_{ref,k}}, \qquad B_k = \frac{\partial f}{\partial u}\Big|_{x_{ref,k}, u_{ref,k}}$$

where $f(x, u)$ is our **approximate discrete** dynamics model ( `estimated_car_dynamics + rk4` ). **You will form these Jacobians exactly once, using `Xref` and `Uref`.** Here is a summary of the notation:

- $X_{ref}$ ( `Xref` ) - Optimal trajectory computed offline with approximate dynamics model.
- $U_{ref}$ ( `Uref` ) - Optimal controls computed offline with approximate dynamics model.
- $X_{sim}$ ( `Xsim` ) - Simulated trajectory with real dynamics model.
- $\bar{U}$ ( `Ubar` ) - Control we use for simulation with real dynamics model (this is what ILC updates).

In the second step of ILC, we solve the following optimization problem:

$$\begin{aligned}
\min_{\Delta x_{1:N}, \Delta u_{1:N-1}} \quad & J(X_{sim} + \Delta X, \bar{U} + \Delta U) && (2)\\
\text{st} \quad & \Delta x_1 = 0 && (3)\\
& \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k \quad \text{for } k = 1, 2, \ldots, N-1 && (4)
\end{aligned}$$

We are going to initialize our $\bar{U}$ with $U_{ref}$, then the ILC algorithm will update $\bar{U} = \bar{U} + \Delta U$ at each iteration. It should only take 5-10 iterations to converge down to $\|\Delta U\| < 1 \cdot 10^{-2}$. You do not need to do any sort of linesearch between ILC updates.

In [9]:
```
# feel free to use/not use any of these

function trajectory_cost(Xsim::Vector{Vector{Float64}}, # simulated states
                         Ubar::Vector{Vector{Float64}}, # simulated controls
                         Xref::Vector{Vector{Float64}}, # reference X's we w
                         Uref::Vector{Vector{Float64}}, # reference U's we w
                         Q::Matrix,                     # LQR tracking cost
                         R::Matrix,                     # LQR tracking cost
                         Qf::Matrix                     # LQR tracking cost
                         )::Float64                     # return cost J

    J = 0
    # TODO: return trajectory cost J(Xsim, Ubar)
end

function vec_from_mat(Xm::Matrix)::Vector{Vector{Float64}}
    # convert a matrix into a vector of vectors
    X = [Xm[:,i] for i = 1:size(Xm,2)]
    return X
end

function ilc_update(Xsim::Vector{Vector{Float64}}, # simulated states
                    Ubar::Vector{Vector{Float64}}, # simulated controls (ILC
                    Xref::Vector{Vector{Float64}}, # reference X's we want t
```

```
                        Uref::Vector{Vector{Float64}}, # reference U's we want t
                        As::Vector{Matrix{Float64}},   # vector of A jacobians a
                        Bs::Vector{Matrix{Float64}},   # vector of B jacobians a
                        Q::Matrix,                     # LQR tracking cost term
                        R::Matrix,                     # LQR tracking cost term
                        Qf::Matrix                     # LQR tracking cost term
                        )::Vector{Vector{Float64}}     # return vector of ΔU's

    # solve optimization problem for ILC update
    N = length(Xsim)
    nx,nu = size(Bs[1])

    # create variables
    ΔX = cvx.Variable(nx, N)
    ΔU = cvx.Variable(nu, N-1)

    # TODO: cost function (tracking cost on Xref, Uref)
    cost = 0.0
    for k =1:N-1
        cost += 0.5*cvx.quadform(ΔX[:, k] + Xsim[k] - Xref[k], Q) + 0.5*cvx.
    end

    # problem instance
    prob = cvx.minimize(cost)

    # TODO: initial condition constraint
    prob.constraints += (ΔX[:,1]== zeros(size(Xsim[1],1)))
    # TODO: dynamics constraints
    for k=1:N-1
        prob.constraints += (ΔX[:,k+1] == As[k]*ΔX[:,k]+Bs[k]*ΔU[:,k])
    end
    cvx.solve!(prob, ECOS.Optimizer; silent_solver = true)

    # return ΔU
    ΔU = vec_from_mat(ΔU.value)

    return ΔU
end
```

```
ilc_update (generic function with 1 method)
```

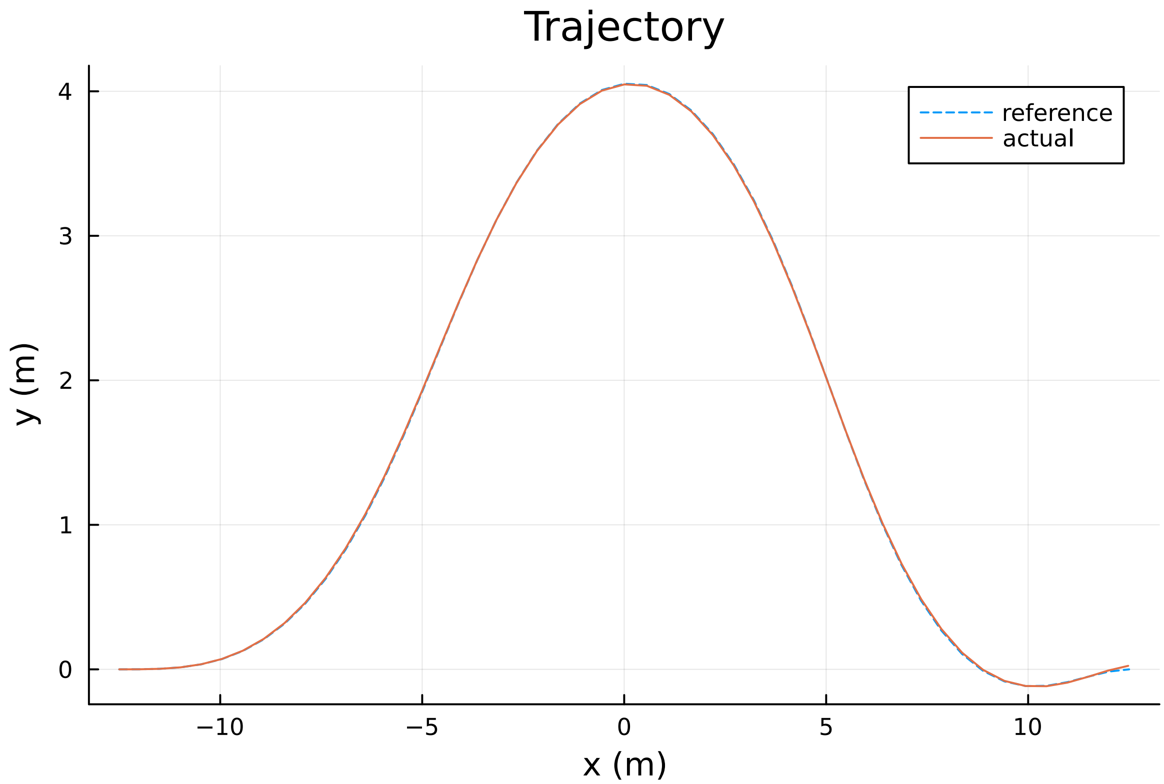
Here you will run your ILC algorithm. The resulting plots should show the simulated trajectory `Xsim` tracks `Xref` very closely, but there should be a significant difference between `Uref` and `Ubar`.

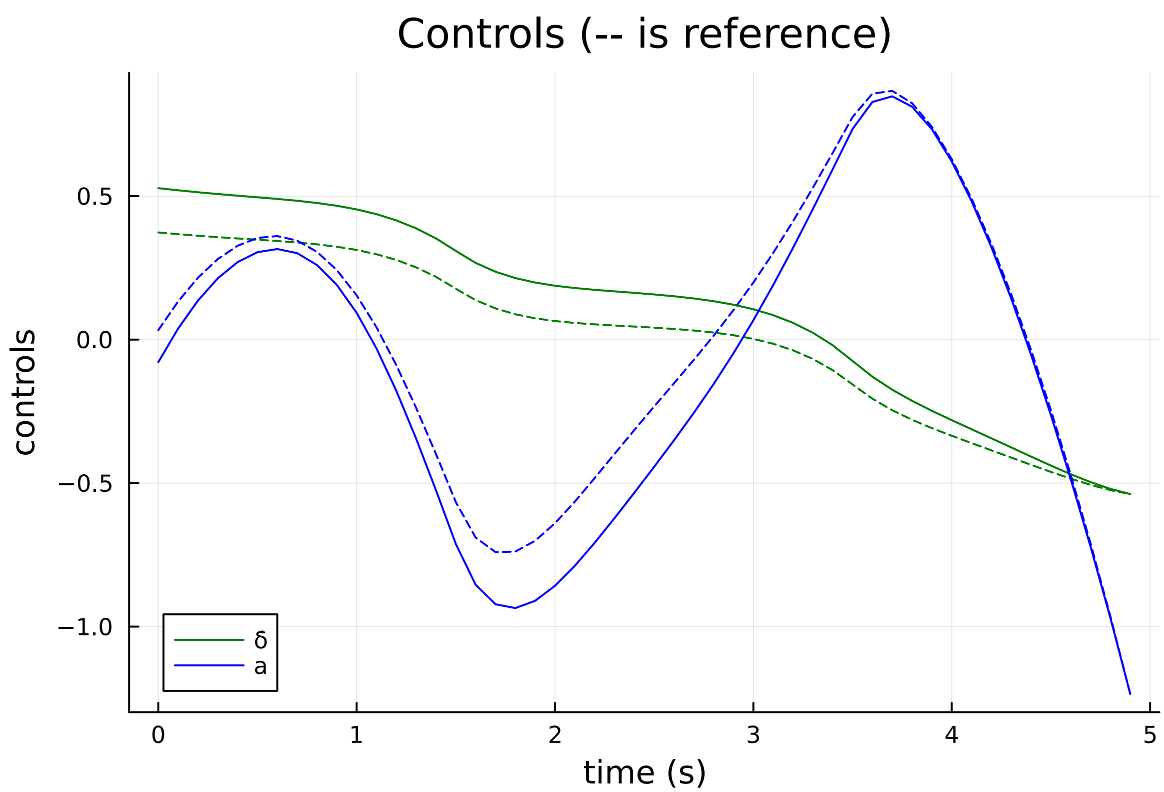
In [10]:
```
@testset "ILC" begin

    # problem size
    nx = 5
    nu = 2
    dt = 0.1
    tf = 5.0
    t_vec = 0:dt:tf
    N = length(t_vec)
```

```
# optimal trajectory computed offline with approximate model
Xref, Uref = load_car_trajectory()

# initial and terminal conditions
xic = Xref[1]
xg = Xref[N]

# LQR tracking cost to be used in ILC
Q = diagm([1,1,.1,.1,.1])
R = .1*diagm(ones(nu))
Qf = 1*diagm(ones(nx))

# load all useful things into params
model = (L = 2.8, lr = 1.6)

params = (Q = Q, R = R, Qf = Qf,xic = xic, xg = xg, Xref=Xref,Uref=Uref,
      dt = dt,
      N = N,
      model = model)


# this holds the sim trajectory (with real dynamics)
Xsim = [zeros(nx) for i = 1:N]

# this is the feedforward control ILC is updating
Ubar = [zeros(nu) for i = 1:(N-1)]
Ubar .= Uref # initialize Ubar with Uref

# TODO: calculate Jacobians
A = [zeros(nx, nu) for i=1:N-1]
B = [zeros(nx,nu) for i=1:N-1]

for k =1:(N-1)
    A[k] = FD.jacobian(dx -> rk4(model,true_car_dynamics,dx,Uref[k],dt),
    B[k] = FD.jacobian(du -> rk4(model,true_car_dynamics,Xref[k], du,dt)
end

# logging stuff
@printf "iter     objv        |ΔU|     \n"
@printf "------------------------------\n"

for ilc_iter = 1:10 # it should not take more than 10 iterations to conv

    # TODO: rollout
    Xsim[1] = Xref[1]
    for i =1:N-1
        Xsim[i+1] = rk4(model,true_car_dynamics,Xsim[i],Ubar[i],dt)
    end
    # TODO: calculate objective val (trajectory_cost)
    obj_val = 0
    obj_val += trajectory_cost(Xsim,Ubar,Xref,Uref,Q,R,Qf)
    # solve optimization problem for update (ilc_update)
    ΔU = 0
    ΔU = ilc_update(Xsim,Ubar,Xref,Uref,A,B,Q,R,Qf)
    # TODO: update the control
    Ubar = Ubar + ΔU
```

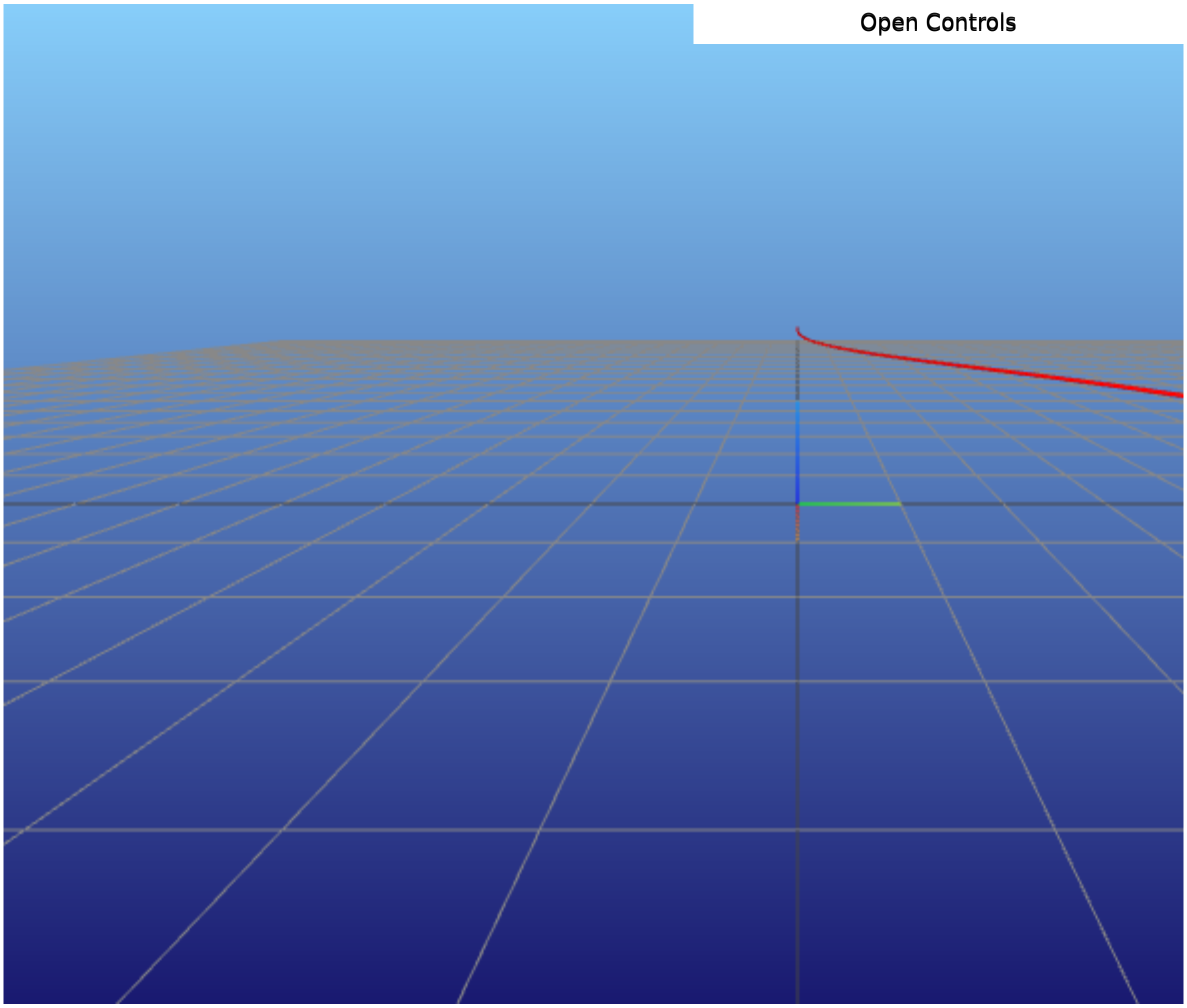
```
        # logging
        @printf("%3d   %10.3e  %10.3e  \n", ilc_iter, obj_val, sum(norm.(ΔU)

    end

    # --------plotting/animation-------
    Xm= hcat(Xsim...)
    Um = hcat(Ubar...)
    Xrefm = hcat(Xref...)
    Urefm = hcat(Uref...)
    plot(Xrefm[1,:], Xrefm[2,:], ls = :dash, label = "reference",
         xlabel = "x (m)", ylabel = "y (m)", title = "Trajectory")
    display(plot!(Xm[1,:], Xm[2,:], label = "actual"))

    plot(t_vec[1:end-1], Urefm', ls = :dash, lc = [:green :blue],label = "",
         xlabel = "time (s)", ylabel = "controls", title = "Controls (-- is
    display(plot!(t_vec[1:end-1], Um', label = ["δ̇" "a"], lc = [:green :blue

    # animation
    vis = Visualizer()
    vis_traj!(vis, :traj, [[x[1],x[2],0.1] for x in Xsim]; R = 0.02)
    build_car!(vis[:car])
    anim = mc.Animation(floor(Int,1/dt))
    for k = 1:N
        mc.atframe(anim, k) do
            update_car_pose!(vis[:car], Xsim[k])
        end
    end
    mc.setanimation!(vis, anim)
    display(render(vis))

    # ----------testing-------------
    @test 0.1 <= sum(norm.(Xsim - Xref)) <= 1.0 # should be ~0.7
    @test 5 <=  sum(norm.(Ubar - Uref)) <= 10 # should be ~7.7

end
```

```
iter     objv        |ΔU|
------------------------------
  1    0.000e+00   5.561e+01
  2    0.000e+00   3.173e+01
  3    0.000e+00   2.609e+01
  4    0.000e+00   1.698e+01
  5    0.000e+00   2.578e+01
  6    0.000e+00   2.002e+01
  7    0.000e+00   1.067e+01
  8    0.000e+00   2.919e+00
  9    0.000e+00   1.647e-01
 10    0.000e+00   1.749e-03
```




```
┌ Info: Listening on: 127.0.0.1:8701, thread id: 1
└ @ HTTP.Servers /home/rsharde/.julia/packages/HTTP/vnQzp/src/Servers.jl:382
┌ Info: MeshCat server started. You can open the visualizer by visiting the
following URL in your browser:
│ http://127.0.0.1:8701
└ @ MeshCat /home/rsharde/.julia/packages/MeshCat/QXID5/src/visualizer.jl:64
```




```
Test Summary: | Pass  Total
ILC           |    2      2
Test.DefaultTestSet("ILC", Any[], 2, false, false)
```

In [1]:
```
import Pkg
Pkg.activate(@__DIR__)
Pkg.instantiate()
import MathOptInterface as MOI
import Ipopt
import FiniteDiff
import ForwardDiff as FD
import Convex as cvx
import ECOS
using LinearAlgebra
using Plots
using Random
using JLD2
using Test
using MeshCat
const mc = MeshCat
using StaticArrays
using Printf
```

```
  Activating environment at `~/OCRL/HW4_S24/Project.toml`
Precompiling project...
  ✓ Contour
  ✓ Format
  ✓ Latexify
  ✓ UnitfulLatexify
  ✓ Plots
  5 dependencies successfully precompiled in 40 seconds (194 already precomp
iled)
```

# Julia note:

incorrect:

```
x_l[idx.x[i]][2] = 0 # this does not change x_l
```

correct:

```
x_l[idx.x[i][2]] = 0 # this changes x_l
```

It should always be `v[index] = new_val` if I want to update v with `new_val` at `index`.

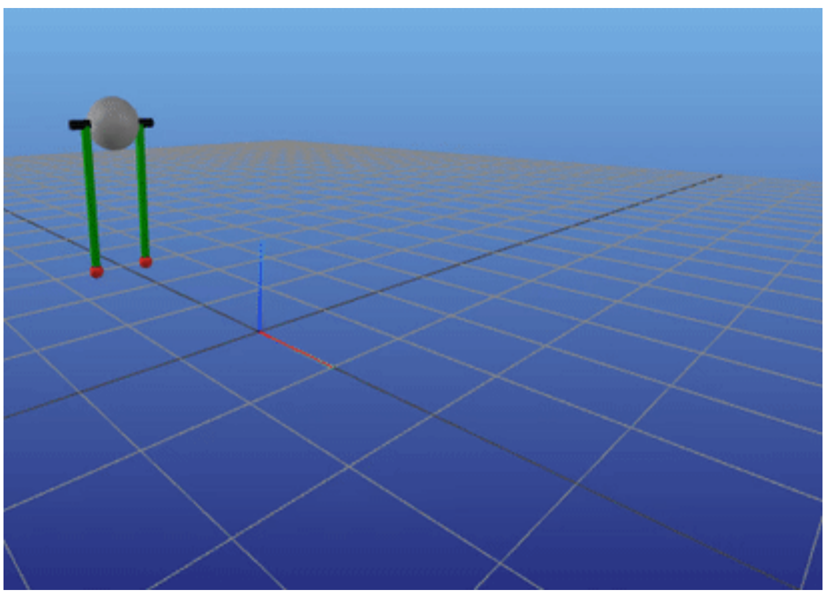
In [2]:
```
let

    # vector we want to modify
    Z = randn(5)

    # original value of Z so we can check if we are changing it
    Z_original = 1 * Z

    # index range we are considering
    idx_x = 1:3
```

```
    # this does NOT change Z
    Z[idx_x][2] = 0

    # we can prove this
    @show norm(Z - Z_original)

    # this DOES change Z
    Z[idx_x[2]] = 0

    # we can prove this
    @show norm(Z - Z_original)


end
```

```
norm(Z - Z_original) = 0.0
norm(Z - Z_original) = 0.8819876395444377
0.8819876395444377
```

In [3]:
```
include(joinpath(@__DIR__, "utils","fmincon.jl"))
include(joinpath(@__DIR__, "utils","walker.jl"))
```

```
update_walker_pose! (generic function with 1 method)
```


(If nothing loads here, check out `walker.gif` in the repo)

**NOTE: This question will have long outputs for each cell, remember you can use `cell -> all output -> toggle scrolling` to better see it all**

# Q2: Hybrid Trajectory Optimization (60 pts)

In this problem you'll use a direct method to optimize a walking trajectory for a simple biped model, using the hybrid dynamics formulation. You'll pre-specify a gait sequence and solve the problem using Ipopt. Your final solution should look like the video above.

# The Dynamics

Our system is modeled as three point masses: one for the body and one for each foot. The state is defined as the x and y positions and velocities of these masses, for a total of 6 degrees of freedom and 12 states. We will label the position and velocity of each body with the following notation:

$$r^{(b)} = \begin{bmatrix} p_x^{(b)} \\ p_y^{(b)} \end{bmatrix} \qquad v^{(b)} = \begin{bmatrix} v_x^{(b)} \\ v_y^{(b)} \end{bmatrix} \tag{1}$$

$$r^{(1)} = \begin{bmatrix} p_x^{(1)} \\ p_y^{(1)} \end{bmatrix} \qquad v^{(1)} = \begin{bmatrix} v_x^{(1)} \\ v_y^{(1)} \end{bmatrix} \tag{2}$$

$$r^{(2)} = \begin{bmatrix} p_x^{(2)} \\ p_y^{(2)} \end{bmatrix} \qquad v^{(2)} = \begin{bmatrix} v_x^{(2)} \\ v_y^{(2)} \end{bmatrix} \tag{3}$$

Each leg is connected to the body with prismatic joints. The system has three control inputs: a force along each leg, and the torque between the legs.

The state and control vectors are ordered as follows:

$$x = \begin{bmatrix} p_x^{(b)} \\ p_y^{(b)} \\ p_x^{(1)} \\ p_y^{(1)} \\ p_x^{(2)} \\ p_y^{(2)} \\ v_x^{(b)} \\ v_y^{(b)} \\ v_x^{(1)} \\ v_y^{(1)} \\ v_x^{(2)} \\ v_y^{(2)} \end{bmatrix} \qquad u = \begin{bmatrix} F^{(1)} \\ F^{(2)} \\ \tau \end{bmatrix}$$

where e.g. $p_x^{(b)}$ is the $x$ position of the body, $v_y^{(i)}$ is the $y$ velocity of foot $i$, $F^{(i)}$ is the force along leg $i$, and $\tau$ is the torque between the legs.

The continuous time dynamics and jump maps for the two stances are shown below:

In [4]:
```
function stance1_dynamics(model::NamedTuple, x::Vector, u::Vector)
    # dynamics when foot 1 is in contact with the ground

    mb,mf = model.mb, model.mf
    g = model.g

    M = Diagonal([mb mb mf mf mf mf])

    rb  = x[1:2]   # position of the body
    rf1 = x[3:4]   # position of foot 1
    rf2 = x[5:6]   # position of foot 2
    v   = x[7:12]  # velocities


    ℓ1x = (rb[1]-rf1[1])/norm(rb-rf1)
    ℓ1y = (rb[2]-rf1[2])/norm(rb-rf1)
    ℓ2x = (rb[1]-rf2[1])/norm(rb-rf2)
    ℓ2y = (rb[2]-rf2[2])/norm(rb-rf2)

    B = [ℓ1x  ℓ2x  ℓ1y-ℓ2y;
         ℓ1y  ℓ2y  ℓ2x-ℓ1x;
          0    0     0;
          0    0     0;
          0  -ℓ2x  ℓ2y;
          0  -ℓ2y -ℓ2x]

    v̇ = [0; -g; 0; 0; 0; -g] + M\(B*u)

    ẋ = [v; v̇]

    return ẋ
end

function stance2_dynamics(model::NamedTuple, x::Vector, u::Vector)
    # dynamics when foot 2 is in contact with the ground

    mb,mf = model.mb, model.mf
    g = model.g
    M = Diagonal([mb mb mf mf mf mf])

    rb  = x[1:2]   # position of the body
    rf1 = x[3:4]   # position of foot 1
    rf2 = x[5:6]   # position of foot 2
    v   = x[7:12]  # velocities

    ℓ1x = (rb[1]-rf1[1])/norm(rb-rf1)
    ℓ1y = (rb[2]-rf1[2])/norm(rb-rf1)
    ℓ2x = (rb[1]-rf2[1])/norm(rb-rf2)
    ℓ2y = (rb[2]-rf2[2])/norm(rb-rf2)

    B = [ℓ1x  ℓ2x  ℓ1y-ℓ2y;
         ℓ1y  ℓ2y  ℓ2x-ℓ1x;
        -ℓ1x   0  -ℓ1y;
        -ℓ1y   0   ℓ1x;
          0    0    0;
          0    0    0]
```

```
    v̇ = [0; -g; 0; -g; 0; 0] + M\(B*u)

    ẋ = [v; v̇]

    return ẋ
end

function jump1_map(x)
    # foot 1 experiences inelastic collision
    xn = [x[1:8]; 0.0; 0.0; x[11:12]]
    return xn
end

function jump2_map(x)
    # foot 2 experiences inelastic collision
    xn = [x[1:10]; 0.0; 0.0]
    return xn
end

function rk4(model::NamedTuple, ode::Function, x::Vector, u::Vector, dt::Rea
    k1 = dt * ode(model, x,        u)
    k2 = dt * ode(model, x + k1/2, u)
    k3 = dt * ode(model, x + k2/2, u)
    k4 = dt * ode(model, x + k3,   u)
    return x + (1/6)*(k1 + 2*k2 + 2*k3 + k4)
end
```

```
rk4 (generic function with 1 method)
```

We are setting up this problem by scheduling out the contact sequence. To do this, we will define the following sets:

$$\mathcal{M}_1 = \{1{:}5, 11{:}15, 21{:}25, 31{:}35, 41{:}45\} \tag{4}$$
$$\mathcal{M}_2 = \{6{:}10, 16{:}20, 26{:}30, 36{:}40\} \tag{5}$$

where $\mathcal{M}_1$ contains the time steps when foot 1 is pinned to the ground ( `stance1_dynamics` ), and $\mathcal{M}_2$ contains the time steps when foot 2 is pinned to the ground ( `stance2_dynamics` ). The jump map sets $\mathcal{J}_1$ and $\mathcal{J}_2$ are the indices where the mode of the next time step is different than the current, i.e. $\mathcal{J}_i \equiv \{k+1 \notin \mathcal{M}_i \mid k \in \mathcal{M}_i\}$. We can write these out explicitly as the following:

$$\mathcal{J}_1 = \{5, 15, 25, 35\} \tag{6}$$
$$\mathcal{J}_2 = \{10, 20, 30, 40\} \tag{7}$$

Another term you will see is set subtraction, or $\mathcal{M}_i \setminus \mathcal{J}_i$. This just means that if $k \in \mathcal{M}_i \setminus \mathcal{J}_i$, then $k$ is in $\mathcal{M}_i$ but not in $\mathcal{J}_i$.

We will make use of the following Julia code for determining which set an index belongs to:

In [5]:
```
let
    M1 = vcat([ (i-1)*10      .+ (1:5)   for i = 1:5]...) # stack the set in
```

```
    M2 = vcat([((i-1)*10 + 5) .+ (1:5)   for i = 1:4]...) # stack the set in
    J1 = [5,15,25,35]
    J2 = [10,20,30,40]

    @show  (5 in M1) # show if 5 is in M1
    @show  (5 in J1) # show if 5 is in J1
    @show !(5 in M1) # show is 5 is not in M1

    @show (5 in M1) && !(5 in J1) # 5 in M1 but not J1 (5 ∈ M_1 \ J1)

end
```

```
5 in M1 = true
5 in J1 = true
!(5 in M1) = false
5 in M1 && !(5 in J1) = false
false
```

We are now going to setup and solve a constrained nonlinear program. The optimization problem looks complicated but each piece should make sense and be relatively straightforward to implement. First we have the following LQR cost function that will track $x_{ref}$ ( `Xref` ) and $u_{ref}$ ( `Uref` ):

$$J(x_{1:N}, u_{1:N-1}) = \sum_{i=1}^{N-1} \left[ \frac{1}{2}(x_i - x_{ref,i})^T Q (x_i - x_{ref,i}) + \frac{1}{2}(u_i - u_{ref,i})^T R (u_i - u_{ref,i}) \right] + \frac{1}{2}(x_N - x_{ref,N})^T Q_f (x_N - x_{ref,N})$$

Which goes into the following full optimization problem:

$$\begin{aligned}
\min_{x_{1:N}, u_{1:N-1}} \quad & J(x_{1:N}, u_{1:N-1}) && && (8)\\
\text{st} \quad & x_1 = x_{ic} && && (1)\\
& x_N = x_g && && (2)\\
& x_{k+1} = f_1(x_k, u_k) && \text{for } k \in \mathcal{M}_1 \setminus \mathcal{J}_1 && (3)\\
& x_{k+1} = f_2(x_k, u_k) && \text{for } k \in \mathcal{M}_2 \setminus \mathcal{J}_2 && (4)\\
& x_{k+1} = g_2(f_1(x_k, u_k)) && \text{for } k \in \mathcal{J}_1 && (5)\\
& x_{k+1} = g_1(f_2(x_k, u_k)) && \text{for } k \in \mathcal{J}_2 && (6)\\
& x_k[4] = 0 && \text{for } k \in \mathcal{M}_1 && (7)\\
& x_k[6] = 0 && \text{for } k \in \mathcal{M}_2 && (8)\\
& 0.5 \leq \|r_k^{(b)} - r_k^{(1)}\|_2 \leq 1.5 && \text{for } k \in [1, N] && (9)\\
& 0.5 \leq \|r_k^{(b)} - r_k^{(2)}\|_2 \leq 1.5 && \text{for } k \in [1, N] && (10)\\
& x_k[2,4,6] \geq 0 && \text{for } k \in [1, N] && (11)
\end{aligned}$$

Each constraint is now described, with the type of constraint for `fmincon` in parantheses:

1. Initial condition constraint **(equality constraint)**.
2. Terminal condition constraint **(equality constraint)**.

3. Stance 1 discrete dynamics **(equality constraint)**.
4. Stance 2 discrete dynamics **(equality constraint)**.
5. Discrete dynamics from stance 1 to stance 2 with jump 2 map **(equality constraint)**.
6. Discrete dynamics from stance 2 to stance 1 with jump 1 map **(equality constraint)**.
7. Make sure the foot 1 is pinned to the ground in stance 1 **(equality constraint)**.
8. Make sure the foot 2 is pinned to the ground in stance 2 **(equality constraint)**.
9. Length constraints between main body and foot 1 **(inequality constraint)**.
10. Length constraints between main body and foot 2 **(inequality constraint)**.
11. Keep the y position of all 3 bodies above ground **(primal bound)**.

And here we have the list of mathematical functions to the Julia function names:

- $f_1$ is `stance1_dynamics + rk4`
- $f_2$ is `stance2_dynamics + rk4`
- $g_1$ is `jump1_map`
- $g_2$ is `jump2_map`

For instance, $g_2(f_1(x_k, u_k))$ is `jump2_map(rk4(model, stance1_dynamics, xk, uk, dt))`

Remember that $r^{(b)}$ is defined above.

In [6]:
```
function reference_trajectory(model, xic, xg, dt, N)
    # creates a reference Xref and Uref for walker

    Uref = [[model.mb*model.g*0.5;model.mb*model.g*0.5;0] for i = 1:(N-1)]

    Xref = [zeros(12) for i = 1:N]

    horiz_v = (3/N)/dt
    xs = range(-1.5, 1.5, length = N)
    Xref[1] = 1*xic
    Xref[N] = 1*xg

    for i = 2:(N-1)
        Xref[i] = [xs[i],1,xs[i],0,xs[i],0,horiz_v,0,horiz_v,0,horiz_v,0]
    end

    return Xref, Uref
end
```

```
reference_trajectory (generic function with 1 method)
```

To solve this problem with Ipopt and `fmincon`, we are going to concatenate all of our $x$'s and $u$'s into one vector (same as HW3Q1):

$$Z = \begin{bmatrix} x_1 \\ u_1 \\ x_2 \\ u_2 \\ \vdots \\ x_{N-1} \\ u_{N-1} \\ x_N \end{bmatrix} \in \mathbb{R}^{N\cdot nx+(N-1)\cdot nu}$$

where $x \in \mathbb{R}^{nx}$ and $u \in \mathbb{R}^{nu}$. Below we will provide useful indexing guide in `create_idx` to help you deal with $Z$. Remember that the API for `fmincon` (that we used in HW3Q1) is the following:

$$\begin{aligned} \min_{z} \quad & \ell(z) && \text{cost function} && (9) \\ \text{st} \quad & c_{eq}(z) = 0 && \text{equality constraint} && (10) \\ & c_L \leq c_{ineq}(z) \leq c_U && \text{inequality constraint} && (11) \\ & z_L \leq z \leq z_U && \text{primal bound constraint} && (12) \end{aligned}$$

Template code has been given to solve this problem but you should feel free to do whatever is easiest for you, as long as you get the trajectory shown in the animation `walker.gif` and pass tests.

In [35]:
```
# feel free to solve this problem however you like, below is a template for
# good way to start.

function create_idx(nx,nu,N)
    # create idx for indexing convenience
    # x_i = Z[idx.x[i]]
    # u_i = Z[idx.u[i]]
    # and stacked dynamics constraints of size nx are
    # c[idx.c[i]] = <dynamics constraint at time step i>
    #
    # feel free to use/not use this

    # our Z vector is [x0, u0, x1, u1, …, xN]
    nz = (N-1) * nu + N * nx # length of Z
    x = [(i - 1) * (nx + nu) .+ (1 : nx) for i = 1:N]
    u = [(i - 1) * (nx + nu) .+ ((nx + 1):(nx + nu)) for i = 1:(N - 1)]

    # constraint indexing for the (N-1) dynamics constraints when stacked up
    c = [(i - 1) * (nx) .+ (1 : nx) for i = 1:(N - 1)]
    nc = (N - 1) * nx # (N-1)*nx

    return (nx=nx,nu=nu,N=N,nz=nz,nc=nc,x= x,u = u,c = c)
end

function walker_cost(params::NamedTuple, Z::Vector)::Real
    # cost function
    idx, N, xg = params.idx, params.N, params.xg
```

```
    Q, R, Qf = params.Q, params.R, params.Qf
    Xref,Uref = params.Xref, params.Uref

    # TODO: input walker LQR cost

    J = 0
    for i =1:N-1
        xi = Z[idx.x[i]]
        ui = Z[idx.u[i]]
        J +=0.5*transpose(xi-xg)*Q*(xi-xg)+0.5*transpose(ui)*R*ui
    end

    xn = Z[idx.x[N]]
    J+= 0.5*transpose(xn-xg)*Q*(xn-xg)
    return J
end


function walker_dynamics_constraints(params::NamedTuple, Z::Vector)::Vector
    idx, N, dt = params.idx, params.N, params.dt
    M1, M2 = params.M1, params.M2
    J1, J2 = params.J1, params.J2
    model = params.model

    # create c in a ForwardDiff friendly way (check HW0)
    c = zeros(eltype(Z), idx.nc)

    # TODO: input walker dynamics constraints (constraints 3-6 in the opti p
    for i = 1:(N-1)
    xi = Z[idx.x[i]]
    ui = Z[idx.u[i]]
    xip1 = Z[idx.x[i+1]]

        # TODO: hermite simpson
        if ((i in M1) && !(i in J1))
            c[idx.c[i]] = xip1 - rk4(model, stance1_dynamics, xi, ui, dt)
        elseif ((i in M2) && !(i in J2))
            c[idx.c[i]] = xip1 - rk4(model, stance2_dynamics, xi, ui, dt)
        elseif (i in J1)
            c[idx.c[i]] = xip1 - jump2_map(rk4(model, stance1_dynamics, xi,
        elseif (i in J2)
            c[idx.c[i]] = xip1 - jump1_map(rk4(model, stance2_dynamics, xi,
        end
    end
    return c

   end

function walker_stance_constraint(params::NamedTuple, Z::Vector)::Vector
    idx, N, dt = params.idx, params.N, params.dt
    M1, M2 = params.M1, params.M2
    J1, J2 = params.J1, params.J2

    model = params.model

    # create c in a ForwardDiff friendly way (check HW0)
```

```
    c = zeros(eltype(Z), N)

    # TODO: add walker stance constraints (constraints 7-8 in the opti probl
    for i =1:N
        xi = Z[idx.x[i]]
        if (i in M1)
            c[i] = xi[4]
        else
            c[i] = xi[6]
        end
    end

    return c
end


function walker_equality_constraint(params::NamedTuple, Z::Vector)::Vector
    N, idx, xic, xg = params.N, params.idx, params.xic, params.xg

    # TODO: stack up all of our equality constraints

    # should be length 2*nx + (N-1)*nx + N
    # inital condition constraint (nx)       (constraint 1)
    # terminal constraint         (nx)       (constraint 2)
    # dynamics constraints        (N-1)*nx   (constraint 3-6)
    # stance constraint           N          (constraint 7-8)

    constraint_1 = params.xic - Z[idx.x[1]]
    constraint_2 = params.xg - Z[idx.x[N]]
    constraint_3_6 = walker_dynamics_constraints(params, Z)
    constraint_7_8 = walker_stance_constraint(params, Z)

    return [constraint_1; constraint_2; constraint_3_6; constraint_7_8]
    # return [params.xic-Z[idx.x[1]]; params.xg-Z[idx.x[N]]; walker_dynamics
end

function walker_inequality_constraint(params::NamedTuple, Z::Vector)::Vector
    idx, N, dt = params.idx, params.N, params.dt
    M1, M2 = params.M1, params.M2

    # create c in a ForwardDiff friendly way (check HW0)
    c = zeros(eltype(Z), 2*N)

    # TODO: add the length constraints shown in constraints (9-10)
    # there are 2*N constraints here

    for i = 1:N
        x = Z[idx.x[i]]
        rb = x[1:2]
        r1 = x[3:4]
        r2 = x[5:6]
        c[(i-1)*2 + 1] = norm(rb - r1)^2
        c[(i-1)*2 + 2] = norm(rb - r2)^2
    end
    return c
end
```

```
walker_inequality_constraint (generic function with 1 method)
```

In [47]:
```
@testset "walker trajectory optimization" begin

    # dynamics parameters
    model = (g = 9.81, mb= 5.0, mf = 1.0, ℓ_min = 0.5, ℓ_max = 1.5)

    # problem size
    nx = 12
    nu = 3
    tf = 4.4
    dt = 0.1
    t_vec = 0:dt:tf
    N = length(t_vec)

    # initial and goal states
    xic = [-1.5;1;-1.5;0;-1.5;0;0;0;0;0;0;0]
    xg =  [1.5;1;1.5;0;1.5;0;0;0;0;0;0;0]

    # index sets
    M1 = vcat([ (i-1)*10      .+ (1:5)   for i = 1:5]...)
    M2 = vcat([((i-1)*10 + 5) .+ (1:5)   for i = 1:4]...)
    J1 = [5,15,25,35]
    J2 = [10,20,30,40]

    # reference trajectory
    Xref, Uref = reference_trajectory(model, xic, xg, dt, N)

    # LQR cost function (tracking Xref, Uref)
    Q = diagm([1; 10; fill(1.0, 4); 1; 10; fill(1.0, 4)]);
    R = diagm(fill(1e-3,3))
    Qf = 1*Q;

    # create indexing utilities
    idx = create_idx(nx,nu,N)

    # put everything useful in params
    params = (
        model = model,
        nx = nx,
        nu = nu,
        tf = tf,
        dt = dt,
        t_vec = t_vec,
        N = N,
        M1 = M1,
        M2 = M2,
        J1 = J1,
        J2 = J2,
        xic = xic,
        xg = xg,
        idx = idx,
        Q = Q, R = R, Qf = Qf,
        Xref = Xref,
        Uref = Uref
    )
```

```julia
# TODO: primal bounds (constraint 11)
x_l = -Inf*ones(idx.nz) # update this
x_u =  Inf*ones(idx.nz) # update this

[x_l[idx.x[i][j]] = 0 for i in 1:N, j in [2, 4, 6]]


# TODO: inequality constraint bounds
c_l = 0.25*ones(2*N) # update this
c_u = 2.25*ones(2*N) # update this

# TODO: initialize z0 with the reference Xref, Uref
z0 = zeros(idx.nz) # update this
for i =1:N
    z0[idx.x[i]] = Xref[i]
    if(i!=N)
        z0[idx.u[i]] = Uref[i]
    end
end
# adding a little noise to the initial guess is a good idea
z0 = z0 + (1e-6)*randn(idx.nz)

diff_type = :auto

Z = fmincon(walker_cost,walker_equality_constraint,walker_inequality_con
            x_l,x_u,c_l,c_u,z0,params, diff_type;
            tol = 1e-6, c_tol = 1e-6, max_iters = 10_000, verbose = true

# pull the X and U solutions out of Z
X = [Z[idx.x[i]] for i = 1:N]
U = [Z[idx.u[i]] for i = 1:(N-1)]

# -----------plotting-------------
Xm = hcat(X...)
Um = hcat(U...)

plot(Xm[1,:],Xm[2,:], label = "body")
plot!(Xm[3,:],Xm[4,:], label = "leg 1")
display(plot!(Xm[5,:],Xm[6,:], label = "leg 2",xlabel = "x (m)",
              ylabel = "y (m)", title = "Body Positions"))

display(plot(t_vec[1:end-1], Um',xlabel = "time (s)", ylabel = "U",
             label = ["F1" "F2" "τ"], title = "Controls"))

# ---------animation------------
vis = Visualizer()
build_walker!(vis, model::NamedTuple)
anim = mc.Animation(floor(Int,1/dt))
for k = 1:N
    mc.atframe(anim, k) do
        update_walker_pose!(vis, model::NamedTuple, X[k])
    end
end
mc.setanimation!(vis, anim)
display(render(vis))
```

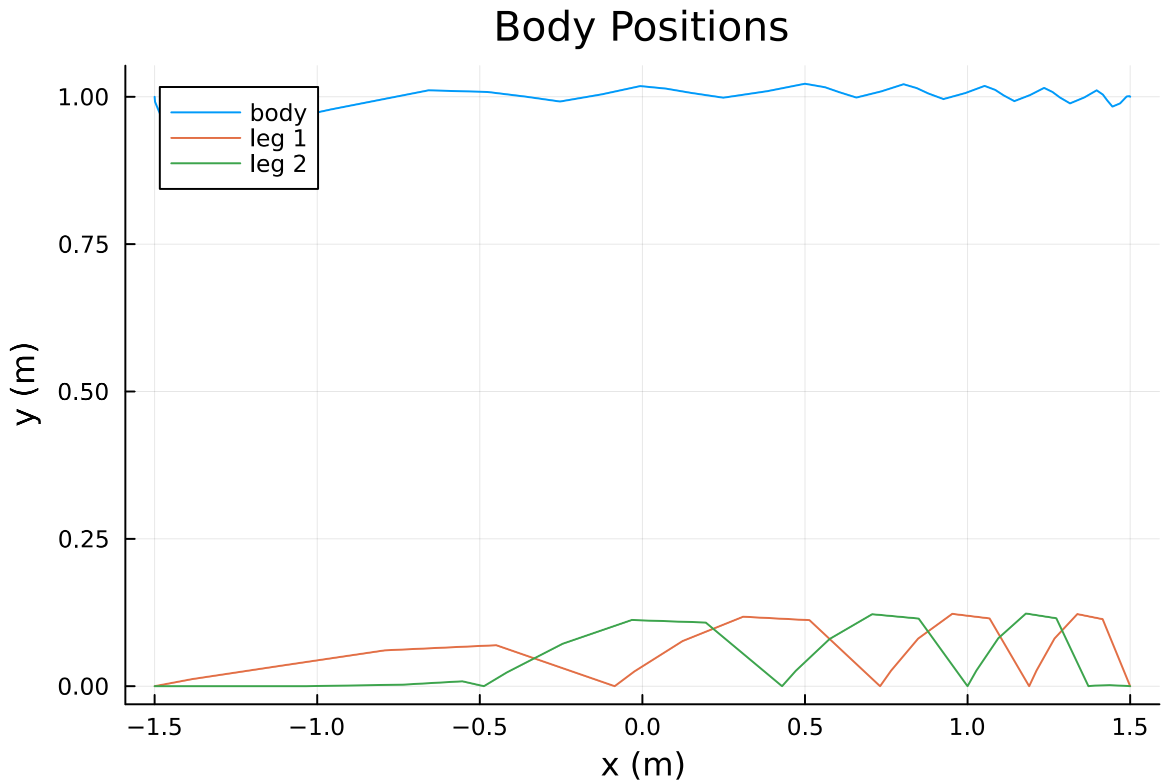
```
    # --------testing----------------

    # initial and terminal states
    @test norm(X[1] - xic,Inf) <= 1e-3
    @test norm(X[end] - xg,Inf) <= 1e-3

    for x in X

        # distance between bodies
        rb = x[1:2]
        rf1 = x[3:4]
        rf2 = x[5:6]
        @test (0.5 - 1e-3) <= norm(rb-rf1) <= (1.5 + 1e-3)
        @test (0.5 - 1e-3) <= norm(rb-rf2) <= (1.5 + 1e-3)

        # no two feet moving at once
        v1 = x[9:10]
        v2 = x[11:12]
        @test min(norm(v1,Inf),norm(v2,Inf)) <= 1e-3

        # check everything above the surface
        @test x[2] >= (0 - 1e-3)
        @test x[4] >= (0 - 1e-3)
        @test x[6] >= (0 - 1e-3)

    end

end
```

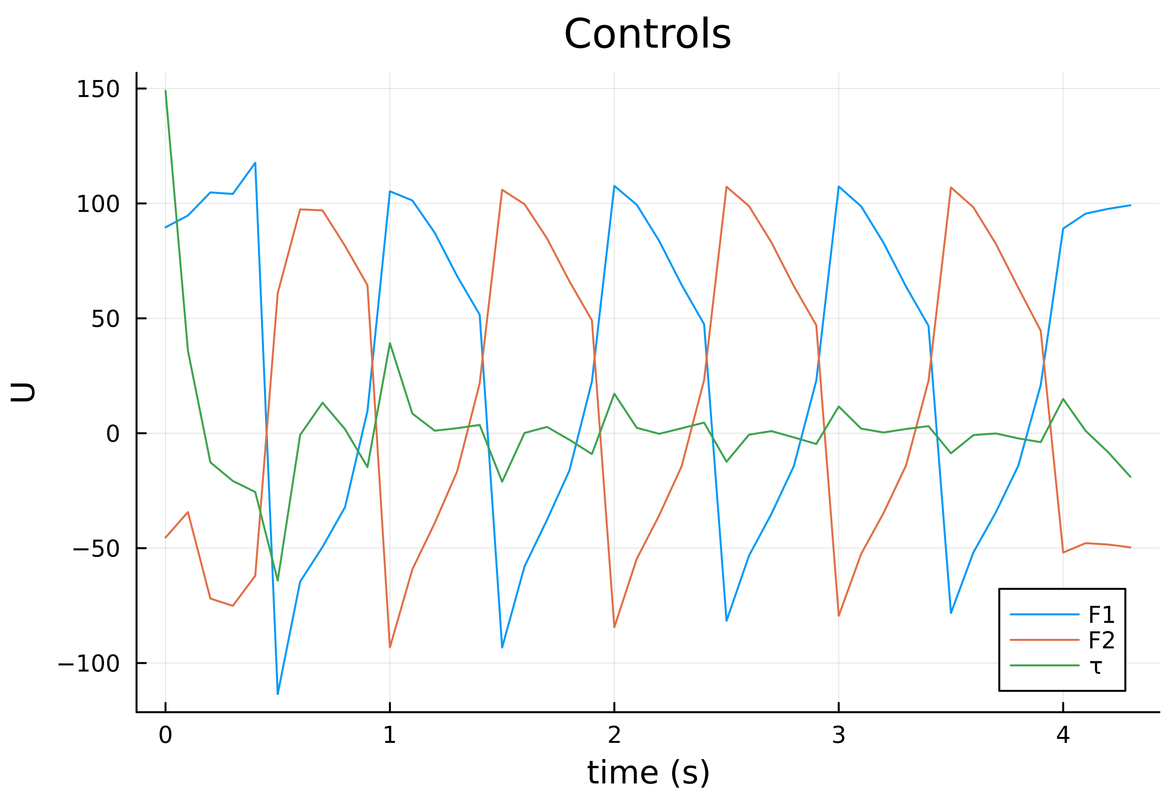
```
---------checking dimensions of everything----------
---------all dimensions good-----------------------
---------diff type set to :auto (ForwardDiff.jl)----
---------testing objective gradient-----------------
---------testing constraint Jacobian----------------
---------successfully compiled both derivatives-----
---------IPOPT beginning solve---------------------
This is Ipopt version 3.14.4, running with linear solver MUMPS 5.4.1.

Number of nonzeros in equality constraint Jacobian...:   401184
Number of nonzeros in inequality constraint Jacobian.:    60480
Number of nonzeros in Lagrangian Hessian.............:        0

Total number of variables............................:      672
                     variables with only lower bounds:      135
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:      597
Total number of inequality constraints...............:       90
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:       90
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
   0  2.5993724e+02 1.47e+00 3.00e+00   0.0 0.00e+00    -  0.00e+00 0.00e+00
0
   1  3.3682679e+02 1.06e+00 4.52e+03  -0.7 1.18e+02    -  4.10e-01 3.62e-01
h  1
   2  5.1585429e+02 1.03e+00 5.53e+03   1.0 1.76e+02    -  1.00e+00 2.42e-01
f  1
   3  7.1939721e+02 9.16e-01 1.81e+03   0.8 7.90e+01    -  7.80e-01 9.25e-01
h  1
   4  7.1970815e+02 3.93e-01 9.15e+03   0.8 3.72e+01    -  2.16e-01 6.90e-01
f  1
   5  7.5049714e+02 3.46e-01 3.92e+03   1.3 6.13e+01    -  9.41e-01 1.00e+00
f  1
   6  6.5949457e+02 3.27e-02 3.09e+02   1.0 2.61e+01    -  1.00e+00 1.00e+00
h  1
   7  5.8755694e+02 3.69e-02 7.41e+01   0.4 3.44e+01    -  9.78e-01 1.00e+00
h  1
   8  5.4987927e+02 6.46e-03 4.95e+01   0.1 2.49e+01    -  1.00e+00 1.00e+00
H  1
   9  5.2327199e+02 3.53e-03 8.50e+01  -0.2 3.50e+01    -  9.76e-01 1.00e+00
H  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
  10  5.0203069e+02 7.35e-02 5.49e+01  -0.5 2.74e+01    -  1.00e+00 1.00e+00
f  1
  11  4.9538745e+02 1.71e-01 2.45e+02  -0.5 6.55e+01    -  9.89e-01 6.79e-01
f  1
  12  4.9687806e+02 1.11e-01 3.66e+01  -0.5 2.86e+01    -  1.00e+00 1.00e+00
h  1
  13  4.8067314e+02 9.78e-02 9.02e+00  -0.3 2.29e+01    -  1.00e+00 1.00e+00
f  1
  14  4.7602217e+02 5.15e-02 6.20e+01  -0.5 1.07e+01    -  9.24e-01 1.00e+00
```

```
h  1
  15  4.6942158e+02 6.52e-03 3.28e+00  -0.9 7.19e+00    -  1.00e+00 1.00e+00
h  1
  16  4.6805616e+02 9.94e-04 1.17e+00  -1.7 3.97e+00    -  9.92e-01 1.00e+00
h  1
  17  4.6617736e+02 8.09e-03 3.44e+00  -2.4 1.17e+01    -  9.99e-01 1.00e+00
f  1
  18  4.6576380e+02 7.94e-03 6.26e+01  -2.1 5.02e+01    -  1.00e+00 1.05e-01
f  3
  19  4.6441933e+02 5.53e-03 5.97e+00  -2.5 1.49e+01    -  1.00e+00 9.70e-01
f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
  20  4.7078366e+02 2.37e-03 1.85e+01  -2.8 2.71e+01    -  9.69e-01 8.02e-01
H  1
  21  4.6481699e+02 9.76e-03 4.07e+01  -2.9 8.22e+00    -  1.34e-01 8.94e-01
f  1
  22  4.6400065e+02 4.92e-03 4.07e+01  -2.9 1.04e+01    -  6.58e-01 1.00e+00
f  1
  23  4.6351600e+02 3.05e-03 3.46e+01  -3.5 3.74e+00    -  1.00e+00 3.80e-01
f  1
  24  4.6335968e+02 1.46e-03 6.18e-01  -4.2 3.73e+00    -  1.00e+00 1.00e+00
f  1
  25  4.6324950e+02 5.62e-04 3.86e-01  -4.3 9.71e-01    -  1.00e+00 9.82e-01
h  1
  26  4.6321930e+02 6.69e-05 1.80e-01  -5.6 4.22e-01    -  1.00e+00 1.00e+00
h  1
  27  4.6321299e+02 2.85e-05 4.66e-01  -6.1 4.03e-01    -  1.00e+00 9.91e-01
h  1
  28  4.6318515e+02 3.08e-05 4.21e-01  -7.1 1.12e+00    -  1.00e+00 1.00e+00
h  1
  29  4.6317698e+02 2.47e-05 8.93e+01  -8.3 1.94e+00    -  1.00e+00 2.50e-01
h  3
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
  30  4.6316745e+02 3.29e-05 1.96e-01  -8.0 7.01e-01    -  1.00e+00 1.00e+00
h  1
  31  4.6316328e+02 3.69e-05 2.63e-01  -9.6 8.88e-01    -  1.00e+00 1.00e+00
h  1
  32  4.6316978e+02 6.42e-07 2.23e-01 -10.6 3.66e-01    -  1.00e+00 1.00e+00
H  1
  33  4.6315510e+02 2.20e-05 9.66e-02 -10.3 2.43e-01    -  1.00e+00 1.00e+00
f  1
  34  4.6315448e+02 2.69e-06 4.24e-02 -11.0 9.21e-02    -  1.00e+00 1.00e+00
h  1
  35  4.6315387e+02 1.60e-06 2.44e-02 -11.0 1.28e-01    -  1.00e+00 1.00e+00
h  1
  36  4.6316171e+02 3.57e-08 1.71e-01 -11.0 5.90e-01    -  1.00e+00 1.00e+00
H  1
  37  4.6315325e+02 6.25e-06 3.70e-02 -11.0 3.89e-01    -  1.00e+00 1.00e+00
f  1
  38  4.6315335e+02 5.59e-07 3.85e-02 -11.0 7.64e-02    -  1.00e+00 1.00e+00
h  1
  39  4.6315292e+02 1.94e-07 8.94e-03 -11.0 4.72e-02    -  1.00e+00 1.00e+00
h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
```

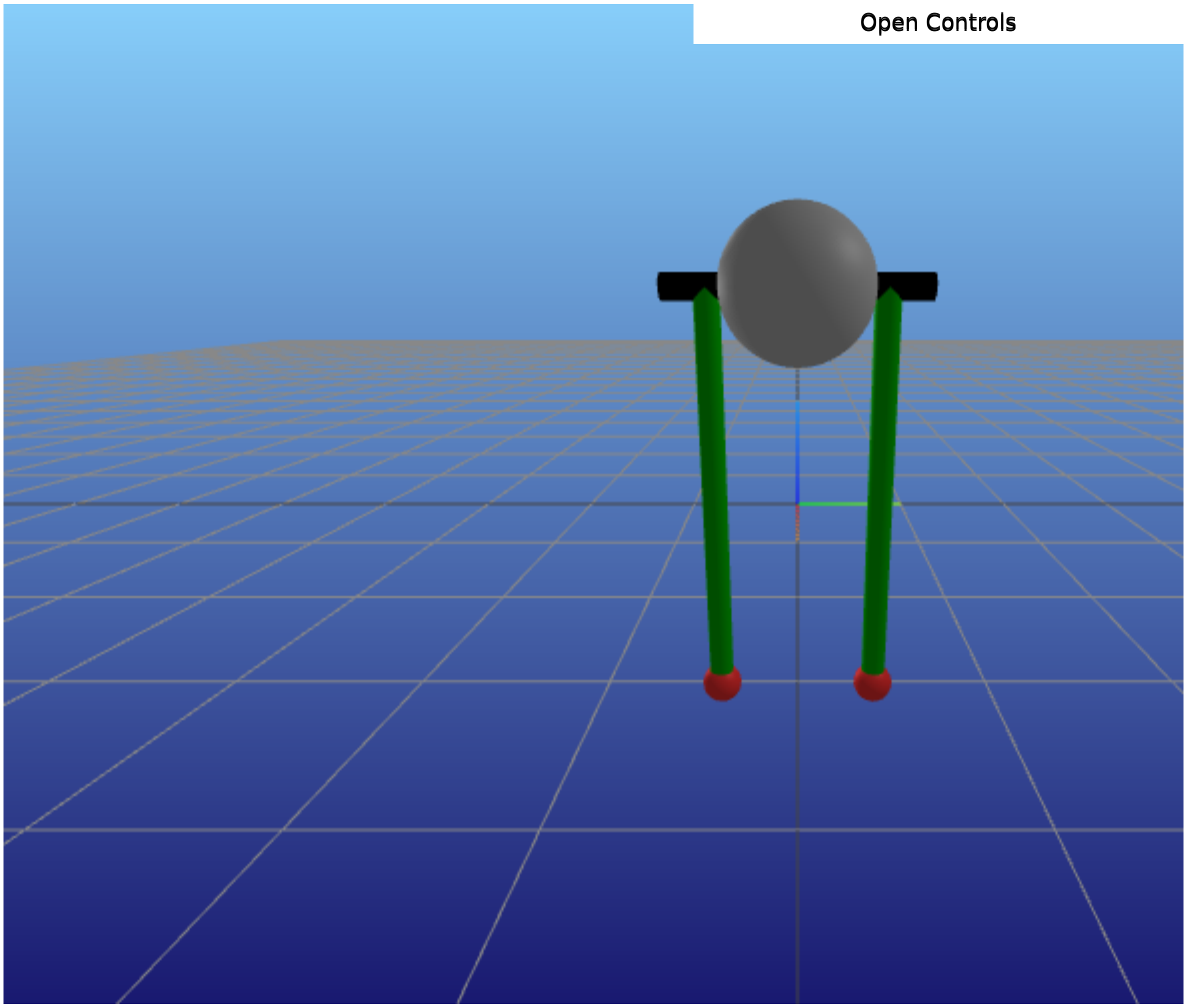
```
ls
  40  4.6315288e+02 4.05e-08 6.84e-03 -11.0 2.30e-02    -  1.00e+00 1.00e+00
h  1
  41  4.6315276e+02 5.17e-07 2.69e-02 -11.0 1.34e-01    -  1.00e+00 1.00e+00
h  1
  42  4.6315566e+02 2.83e-08 1.21e-01 -11.0 4.77e-01    -  1.00e+00 1.00e+00
H  1
  43  4.6315432e+02 4.59e-06 5.62e+02 -11.0 2.28e-01    -  1.00e+00 5.00e-01
f  2
  44  4.6315419e+02 1.12e-05 5.92e-02 -11.0 4.95e-01    -  1.00e+00 1.00e+00
h  1
  45  4.6315363e+02 8.42e-06 8.44e+02 -11.0 2.31e-01    -  1.00e+00 2.50e-01
h  3
  46  4.6315261e+02 2.29e-06 1.48e-02 -11.0 2.12e-01    -  1.00e+00 1.00e+00
h  1
  47  4.6315284e+02 4.20e-07 2.41e-02 -11.0 8.50e-02    -  1.00e+00 1.00e+00
h  1
  48  4.6315259e+02 2.18e-07 6.23e-03 -11.0 7.17e-02    -  1.00e+00 1.00e+00
h  1
  49  4.6315278e+02 1.00e-08 2.51e-02 -11.0 7.26e-02    -  1.00e+00 1.00e+00
H  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
  50  4.6315258e+02 2.99e-07 2.38e-03 -11.0 5.58e-02    -  1.00e+00 1.00e+00
h  1
  51  4.6315270e+02 1.00e-08 1.37e-02 -11.0 3.45e-02    -  1.00e+00 1.00e+00
H  1
  52  4.6315258e+02 2.43e-07 7.04e-04 -11.0 2.76e-02    -  1.00e+00 1.00e+00
h  1
  53  4.6315258e+02 1.00e-08 5.38e-04 -11.0 2.63e-03    -  1.00e+00 1.00e+00
h  1
  54  4.6315332e+02 1.00e-08 5.78e-02 -11.0 2.49e-01    -  1.00e+00 1.00e+00
H  1
  55  4.6315261e+02 1.10e-06 7.98e-03 -11.0 2.00e-01    -  1.00e+00 1.00e+00
f  1
  56  4.6315320e+02 1.00e-08 5.21e-02 -11.0 1.23e-01    -  1.00e+00 1.00e+00
H  1
  57  4.6315257e+02 1.01e-06 3.01e-03 -11.0 1.03e-01    -  1.00e+00 1.00e+00
f  1
  58  4.6315258e+02 1.60e-08 7.38e-03 -11.0 1.01e-02    -  1.00e+00 1.00e+00
h  1
  59  4.6315257e+02 1.73e-08 1.22e-03 -11.0 6.35e-03    -  1.00e+00 1.00e+00
h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
  60  4.6315257e+02 1.00e-08 7.03e-04 -11.0 7.70e-04    -  1.00e+00 1.00e+00
h  1
  61  4.6315257e+02 1.00e-08 1.38e-04 -11.0 3.71e-04    -  1.00e+00 1.00e+00
h  1
  62  4.6315257e+02 1.00e-08 2.32e-04 -11.0 3.89e-04    -  1.00e+00 1.00e+00
h  1
  63  4.6315257e+02 1.00e-08 9.10e-04 -11.0 1.38e-03    -  1.00e+00 1.00e+00
h  1
  64  4.6315257e+02 1.00e-08 1.56e-03 -11.0 1.99e-03    -  1.00e+00 1.00e+00
H  1
  65  4.6315257e+02 1.00e-08 3.26e-04 -11.0 1.29e-03    -  1.00e+00 1.00e+00
```

```
h  1
  66  4.6315257e+02 1.00e-08 8.62e-05 -11.0 1.74e-04    -  1.00e+00 1.00e+00
h  1
  67  4.6315257e+02 1.00e-08 9.10e-05 -11.0 2.67e-04    -  1.00e+00 1.00e+00
h  1
  68  4.6315257e+02 1.00e-08 2.74e-04 -11.0 1.13e-03    -  1.00e+00 1.00e+00
H  1
  69  4.6315257e+02 1.00e-08 5.62e+02 -11.0 6.50e-04    -  1.00e+00 5.00e-01
h  2
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr
ls
  70  4.6315257e+02 1.00e-08 1.56e-04 -11.0 4.65e-04    -  1.00e+00 1.00e+00
H  1
  71  4.6315257e+02 1.00e-08 9.25e-05 -11.0 2.40e-04    -  1.00e+00 1.00e+00
h  1
  72  4.6315257e+02 1.00e-08 2.55e-05 -11.0 1.29e-04    -  1.00e+00 1.00e+00
h  1

Number of Iterations....: 72

                                   (scaled)                 (unscaled)
Objective...............:   4.6315256781851082e+02    4.6315256781851082e+02
Dual infeasibility......:   2.5538525068036222e-05    2.5538525068036222e-05
Constraint violation....:   9.9999997937763180e-09    9.9999997937763180e-09
Variable bound violation:   9.9999997937763180e-09    9.9999997937763180e-09
Complementarity.........:   1.0000000028850007e-11    1.0000000028850007e-11
Overall NLP error.......:   5.1535350911644129e-07    2.5538525068036222e-05


Number of objective function evaluations             = 108
Number of objective gradient evaluations             = 73
Number of equality constraint evaluations            = 108
Number of inequality constraint evaluations          = 108
Number of equality constraint Jacobian evaluations   = 73
Number of inequality constraint Jacobian evaluations = 73
Number of Lagrangian Hessian evaluations             = 0
Total seconds in IPOPT                               = 22.970

EXIT: Optimal Solution Found.
```




```
┌ Info: Listening on: 127.0.0.1:8709, thread id: 1
└ @ HTTP.Servers /home/rsharde/.julia/packages/HTTP/vnQzp/src/Servers.jl:382
┌ Info: MeshCat server started. You can open the visualizer by visiting the
following URL in your browser:
│ http://127.0.0.1:8709
└ @ MeshCat /home/rsharde/.julia/packages/MeshCat/QXID5/src/visualizer.jl:64
```




```
Test Summary:                 | Pass  Total
walker trajectory optimization |  272    272
Test.DefaultTestSet("walker trajectory optimization", Any[], 272, false, fal
se)
```

In [ ]: