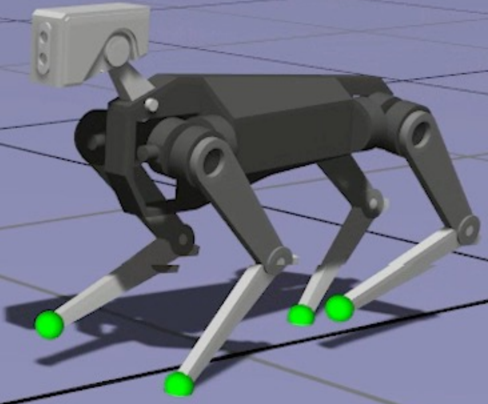
CMM21 - Assignment 2

Kinematic Walking Controller



Dongho Kang
kangd@ethz.ch

ETH zürich

CRL

Quick and Efficient Recap

Unconstrained Optimization & Inverse Kinematics

Unconstrained Optimization

Remember, this is scalar!

Objective function $f(\cdot)\colon \mathbb{R}^n \to \mathbb{R}$

- Our goal

$$\mathbf{q}^* = \operatorname{argmin}_{\mathbf{q}} f(\mathbf{q})$$

Optimal $\mathbf{q}$

Optimization variable $\mathbf{q} \in \mathbb{R}^n$

- Sidenote. Constrained optimization

$$\mathbf{q}^* = \operatorname{argmin}_{\mathbf{q}} f(\mathbf{q})$$

$$\text{s.t.}\ \ \mathbf{g}(\mathbf{q}) = \mathbf{0}$$

Equality constraint $\mathrm{g}(\cdot)\colon \mathbb{R}^n \to \mathbb{R}^{n_{ec}}$

$$\mathbf{h}(\mathbf{q}) \leq \mathbf{0}$$

Inequality constraint $\mathrm{h}(\cdot)\colon \mathbb{R}^n \to \mathbb{R}^{n_{ic}}$

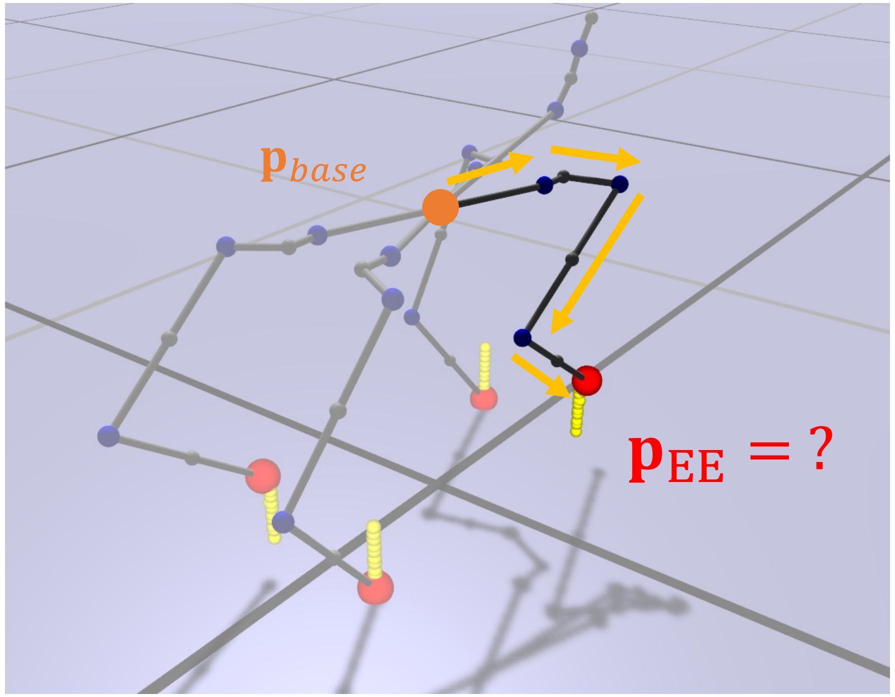
We will see this again soon!

Forward Kinematics

- Generalized coordinates

$$\mathbf{q} = \left[\mathbf{p}_{\text{base}}, \mathbf{\Theta}_{\text{base}}, \theta_1, \theta_2, \dots, \theta_{n_j}\right]^{\mathrm{T}}$$

Base position Base orientation Joint angles



Foot position (in world frame)

- Forward kinematics

$$\mathbf{p}_{\text{EE}} = \mathbf{FK}(\mathbf{q})$$

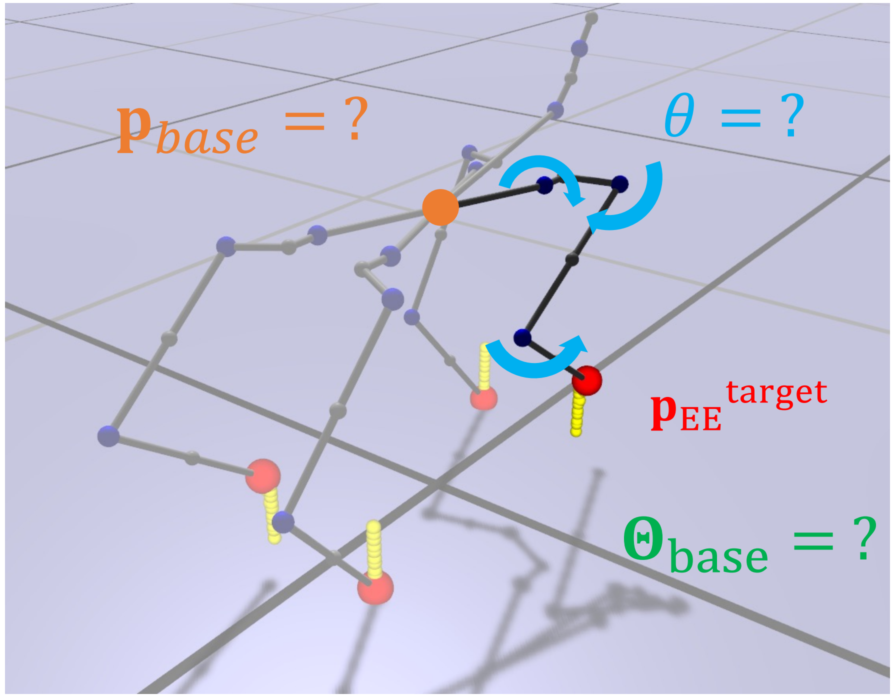
This is what we want to find!

Inverse Kinematics

- Generalized coordinates

$$\mathbf{q} = \left[\mathbf{p}_{\mathrm{base}}, \boldsymbol{\Theta}_{\mathrm{base}}, \theta_1, \theta_2, \ldots, \theta_{n_j}\right]^{\mathrm{T}}$$

Base position   Base orientation   Joint angles



- Inverse kinematics $\quad \mathbf{q}^{\mathrm{desired}} = \mathbf{IK}(\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}})$

This is what we want to find!

Inverse Kinematics

$$\mathbf{q}^{\text{desired}} = \mathbf{IK}({\mathbf{p}_{\text{EE}}}^{\text{target}})$$

Can you guess why?

Unfortunately… no closed-form (in most cases)

- Inverse kinematics as unconstrained optimization

$$\mathbf{q}^{\text{desired}} = \text{argmin}_{\mathbf{q}} \parallel {\mathbf{p}_{\text{EE}}}^{\text{target}} - \mathbf{FK}(\mathbf{q}) \parallel_2{}^2$$

"Least squares problem"

Gradient Descent (first-order method)

What we want to solve…

$$\mathbf{q}^* = \mathrm{argmin}_{\mathbf{q}}\, f(\mathbf{q})$$

- Gradient descent update

$$\nabla f(\mathbf{q}^i) = \left[\frac{\partial f(\mathbf{q}^i)}{\partial q_1}, \frac{\partial f(\mathbf{q}^i)}{\partial q_2}, \dots, \frac{\partial f(\mathbf{q}^i)}{\partial q_n}\right]^{\mathrm{T}}$$

$$\mathbf{q}^{i+1} = \mathbf{q}^i - \gamma \nabla f(\mathbf{q}^i)$$

Iterate until converges…

Gradient Descent (first-order method)

- Gradient $\nabla f\colon\ \mathbb{R}^n \rightarrow \mathbb{R}^n$ returns a vector pointing the steepest slope at the point of evaluation.

$$\nabla f(\mathbf{q}^i) = \left[\frac{\partial f(\mathbf{q}^i)}{\partial q_1}, \frac{\partial f(\mathbf{q}^i)}{\partial q_2}, \dots, \frac{\partial f(\mathbf{q}^i)}{\partial q_n}\right]^{\mathrm{T}}$$

- We go down along the steepest slope!

$$\mathbf{q}^{i+1} = \mathbf{q}^i - \gamma \nabla f(\mathbf{q}^i)$$

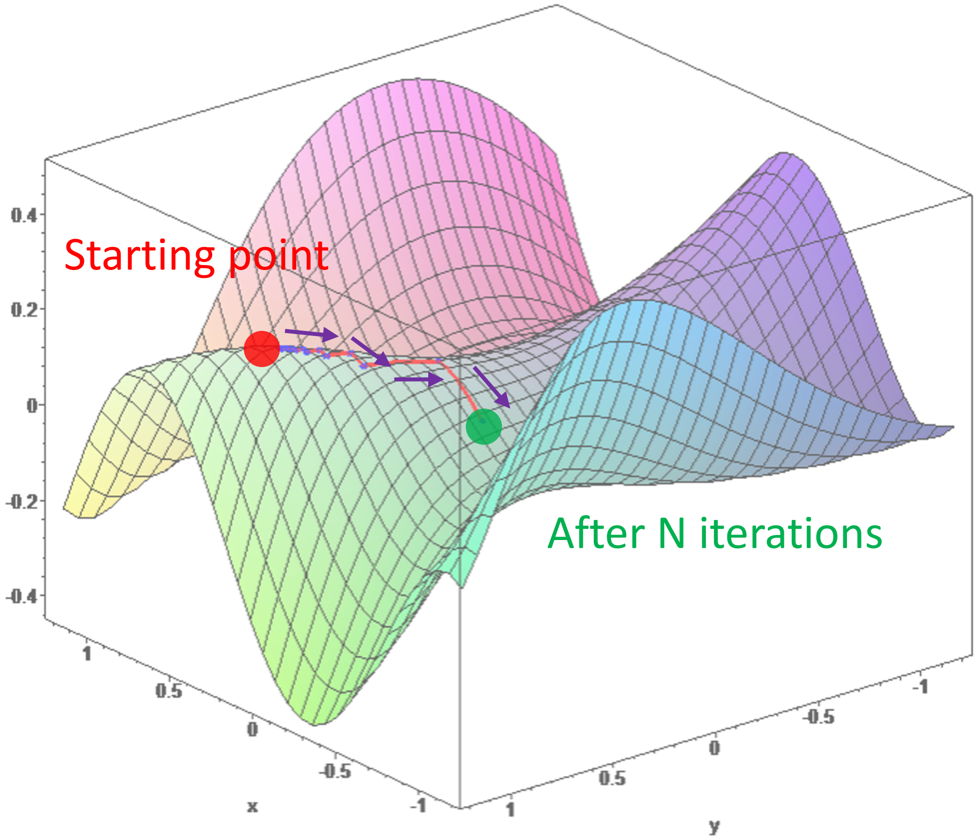


Image courtesy of Wikipeda: Gradient Descent

Newton's Method (second-order method)

What we want to solve…

$$\mathbf{q}^* = \mathrm{argmin}_{\mathbf{q}}\, f(\mathbf{q})$$

- Newton's method update

$$\nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial^2 q_1} & \frac{\partial^2 f}{\partial q_1 \partial q_2} & \cdots & \frac{\partial^2 f}{\partial q_1 \partial q_n} \\ \frac{\partial^2 f}{\partial q_2 \partial q_1} & \frac{\partial^2 f}{\partial^2 q_2} & \cdots & \frac{\partial^2 f}{\partial q_2 \partial q_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial q_n \partial q_1} & \frac{\partial^2 f}{\partial q_n \partial q_2} & \cdots & \frac{\partial^2 f}{\partial^2 q_n} \end{bmatrix}$$

Hessian matrix

$$\mathbf{q}^{i+1} = \mathbf{q}^i - \left(\nabla^2 f(\mathbf{q}^i)\right)^{-1} \nabla f(\mathbf{q}^i)$$

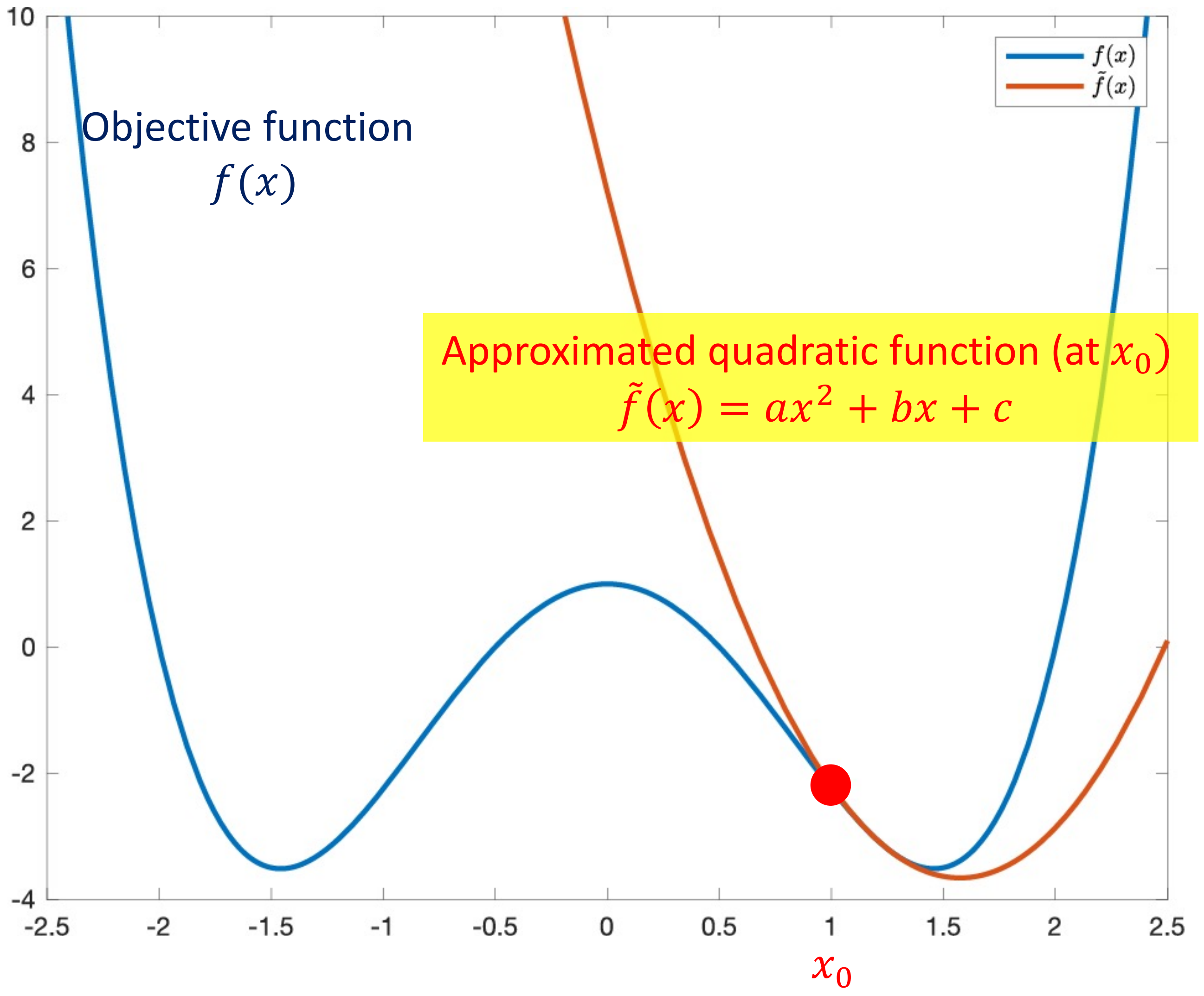
Newton's Method (second-order method)

- What the hack is this?
  - Let's see 1D case
- We approximate $f(x)$ using Taylor expansion

$$\tilde{f}(x) \approx f(x_0) + \nabla f(x_0)^{\mathrm{T}}(x - x_0) + \frac{1}{2}(x - x_0)^{\mathrm{T}}\nabla^2 f(x_0)(x - x_0)$$

- Let's say approximated quadratic function is

$$\tilde{f}(x) = ax^2 + bx + c$$



Newton's Method (second-order method)

- Newton's method update

$$\nabla f(x_0) = \nabla \tilde{f}(x_0) = 2ax_0 + \mathrm{b}$$

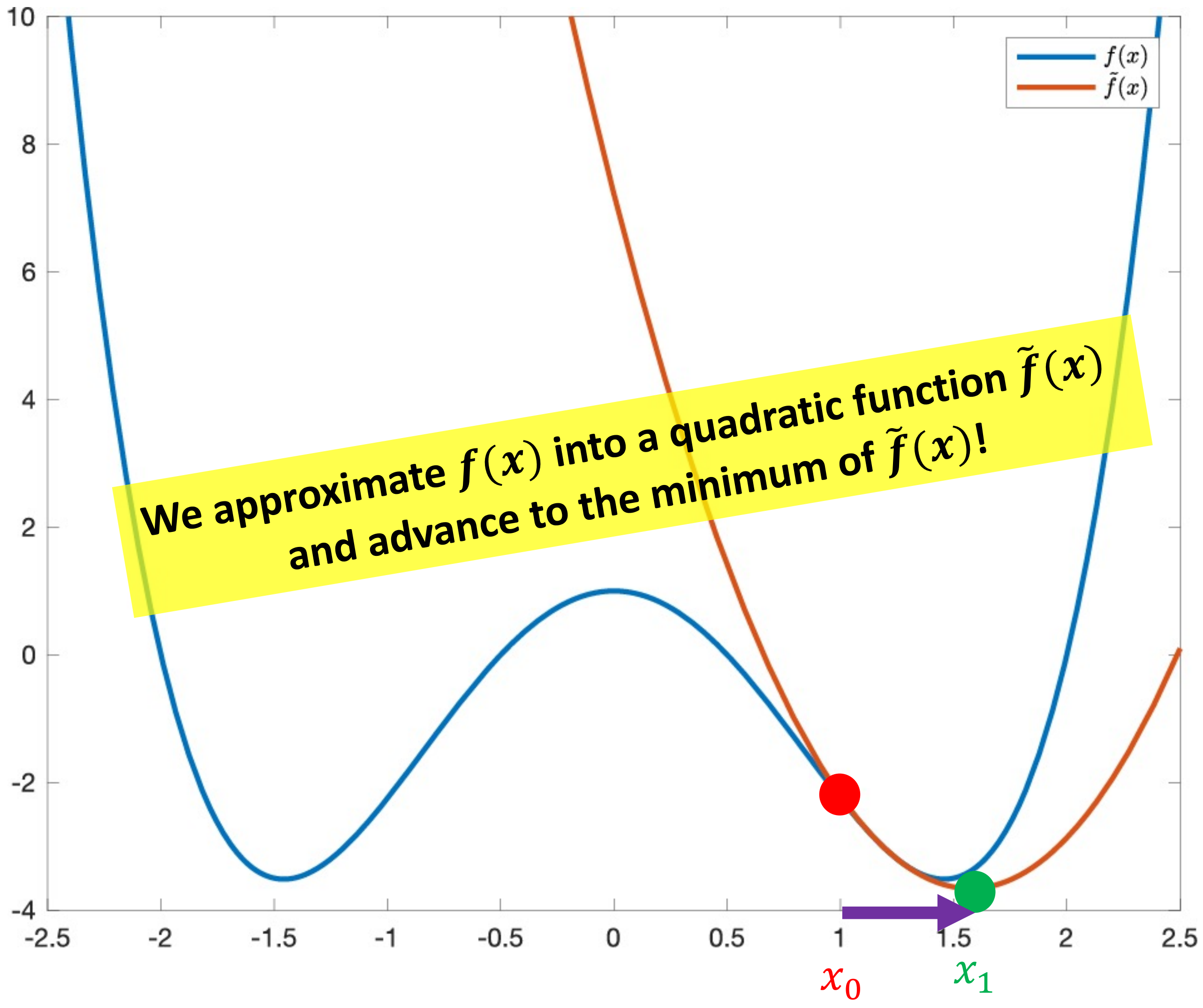
$$\nabla^2 f(x_0) = \nabla^2 \tilde{f}(x_0) = 2a$$

f and $\tilde{f}$ have same gradient and Hessian (curvature) at x_0

$$x_1 = x_0 - (\nabla^2 f(x_0))^{-1} \nabla f(x_0)$$

$$= x_0 - \frac{2ax_0 + b}{2a} = -\frac{b}{2a}$$

This is a minimum of $ax^2 + bx + c$



Inverse Kinematics

$$\mathbf{FK}(\mathbf{q}) = \left[\mathrm{FK}_x(\mathbf{q}), \mathrm{FK}_y(\mathbf{q}), \mathrm{FK}_z(\mathbf{q})\right]^{\mathrm{T}}$$

- Inverse kinematics objective function

$$f(\mathbf{q}) = \frac{1}{2} \| \mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q}) \|_2{}^2$$

Just for brevity…

$$\mathbf{J} = \frac{\partial \mathbf{FK}}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial \mathrm{FK}_x}{\partial q_1} & \cdots & \frac{\partial \mathrm{FK}_x}{\partial q_n} \\ \frac{\partial \mathrm{FK}_y}{\partial q_1} & \cdots & \frac{\partial \mathrm{FK}_y}{\partial q_n} \\ \frac{\partial \mathrm{FK}_z}{\partial q_1} & \cdots & \frac{\partial \mathrm{FK}_z}{\partial q_n} \end{bmatrix}$$

Jacobian matrix

$$\nabla f(\mathbf{q}) = -\mathbf{J}^{\mathrm{T}}(\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q}))$$

$$\nabla^2 f(\mathbf{q}) = -\frac{\partial \mathbf{J}}{\partial \mathbf{q}}^{\mathrm{T}} (\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q})) + \mathbf{J}^{\mathrm{T}}\mathbf{J}$$

$$\nabla f(\mathbf{q}) = -\mathbf{J}^{\mathrm{T}}(\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q}))$$

$$\nabla^2 f(\mathbf{q}) = -\frac{\partial \mathbf{J}}{\partial \mathbf{q}}^{\mathrm{T}} (\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q})) + \mathbf{J}^{\mathrm{T}}\mathbf{J}$$

Inverse Kinematics

- Gradient descent

Often called "Jacobian transpose method"

$$\mathbf{q}^{i+1} = \mathbf{q}^{i} - \gamma \nabla f\left(\mathbf{q}^{i}\right) = \mathbf{q}^{i} + \gamma \mathbf{J}^{\mathrm{T}}(\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}\left(\mathbf{q}^{i}\right))$$

- Newton's method

Usually Newton method converges faster…
But not always better. Why?

$$\mathbf{q}^{i+1} = \mathbf{q}^{i} - (\nabla^2 f(\mathbf{q}^{i}))^{-1} \nabla f(\mathbf{q}^{i})$$

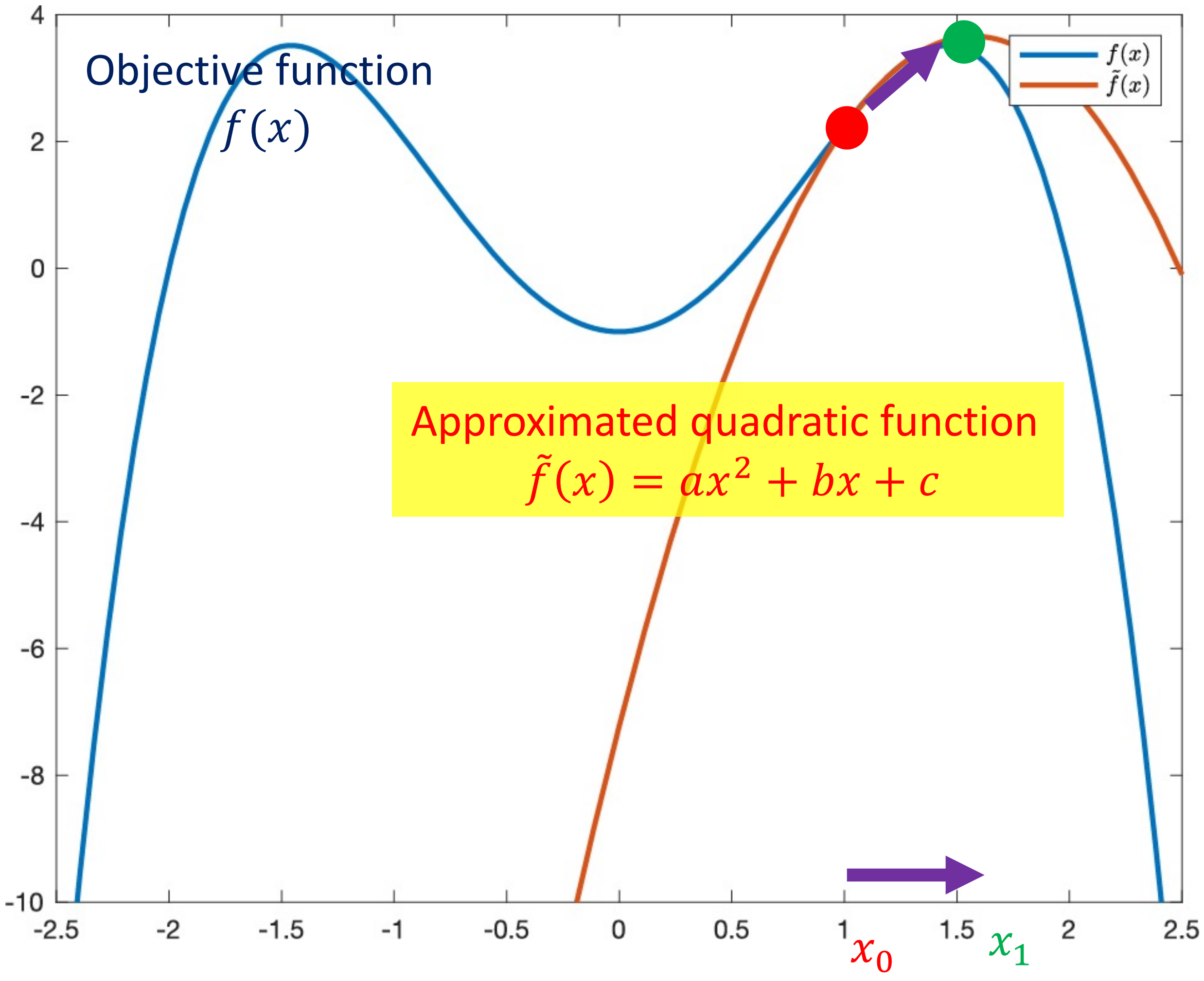
$$= \mathbf{q}^{i} + [-\frac{\partial \mathbf{J}}{\partial \mathbf{q}}^{\mathrm{T}} (\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}\left(\mathbf{q}^{i}\right)) + \mathbf{J}^{\mathrm{T}}\mathbf{J}]^{-1} \mathbf{J}^{\mathrm{T}} (\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}\left(\mathbf{q}^{i}\right))$$

Newton's Method – Convergence

- What if $\nabla^2 f(x_0) < 0$?
  - Back to 1D… ($a < 0$)

- Then we advance to a maximum!

- That's why we use a regularization term.

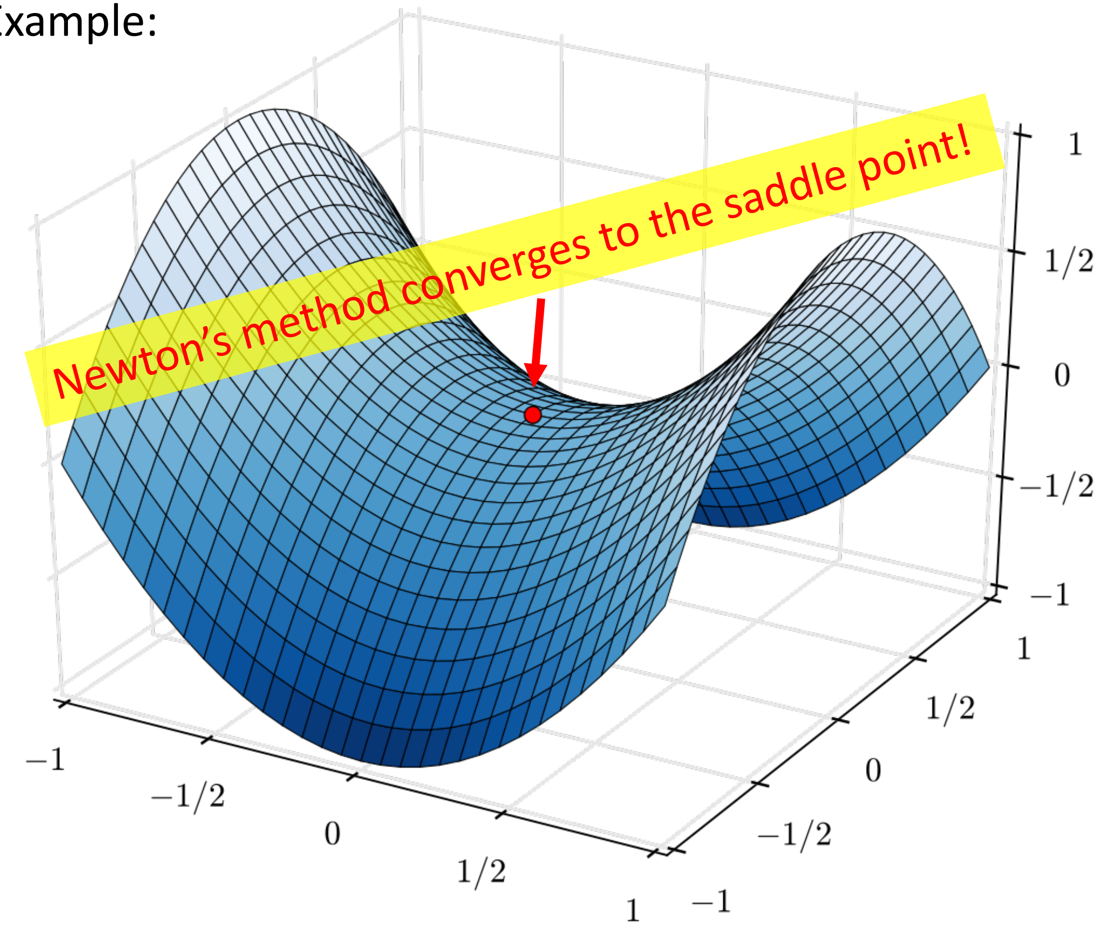


Newton's Method - Convergence

- In N-dimension…
- If $\nabla^2 f(x_0)$ is not positive definite, we cannot guarantee convergence to a minimum.
- Sidenote: Positive Definiteness

$$\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > \mathbf{0} \quad \forall \mathbf{x} \in \mathbb{R}^{\boldsymbol{n}}$$

Example:



Gauss-Newton Method

$$\mathbf{q}^{i+1} = \mathbf{q}^{i} + [\cancel{\frac{\partial \mathbf{J}}{\partial \mathbf{q}}^{\mathrm{T}} (\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q}))} + \mathbf{J}^{\mathrm{T}}\mathbf{J}]^{-1}\mathbf{J}^{\mathrm{T}}(\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q})$$

- Hessian approximation
  - $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ is always positive-definite (Why?)

$$\nabla^2 f(\mathbf{q}) \approx \mathbf{J}^{\mathrm{T}}\mathbf{J}$$

Thus, we advance to minimum.

- Update

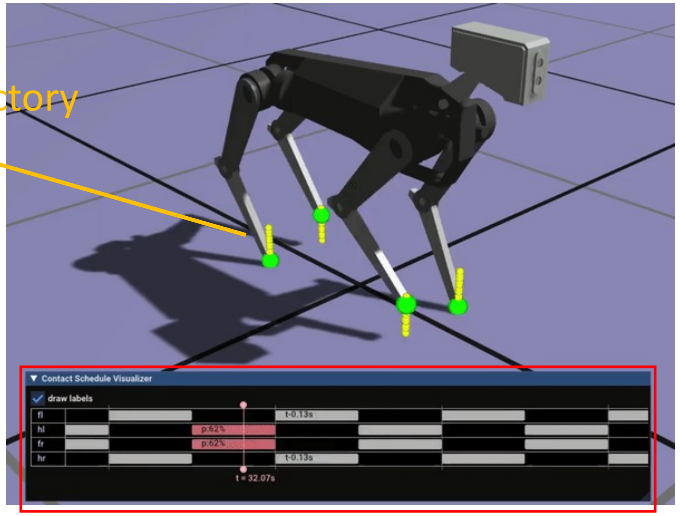
$$\boxed{\mathbf{q}^{i+1} = \mathbf{q}^{i} + (\mathbf{J}^{\mathrm{T}}\mathbf{J})^{-1}\mathbf{J}^{\mathrm{T}}(\mathbf{p}_{\mathrm{EE}}{}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q}^{i}))}$$

$\mathbf{J}^{+} \coloneqq (\mathbf{J}^{\mathrm{T}}\mathbf{J})^{-1}\mathbf{J}^{\mathrm{T}}$ is called Moore-Penrose pseudo inverse. Thus, this method is often called “Jacobian transpose method”

Assignment 2

Kinematic Walking Controller

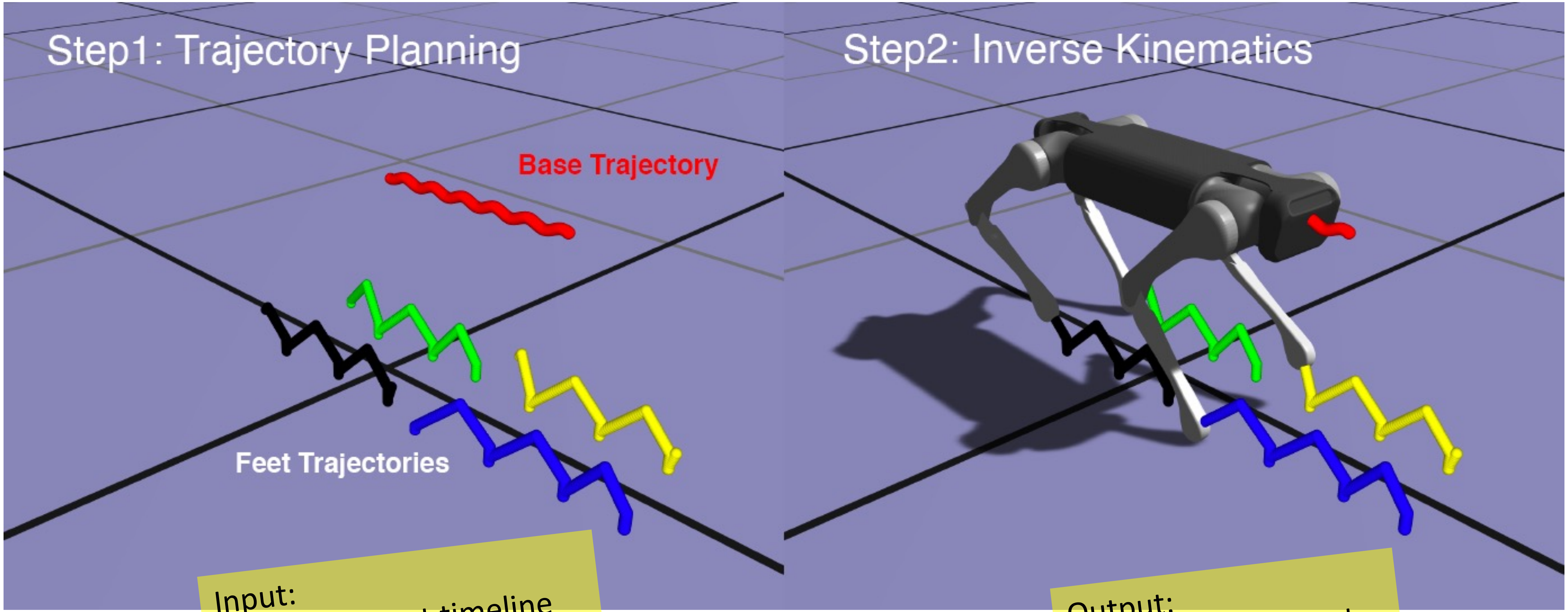
Goal



- We will implement a kinematic walking controller for a legged robot!

- Learning Objective
  - Formulating Inverse Kinematics as an unconstrained oprimization problem.
  - FK and IK Implementation for complicated systems.
  - Sneak peek of legged locomotion planning strategy.

Control Pipeline Overview



Input:
- Foot contact timeline
- User velocity command

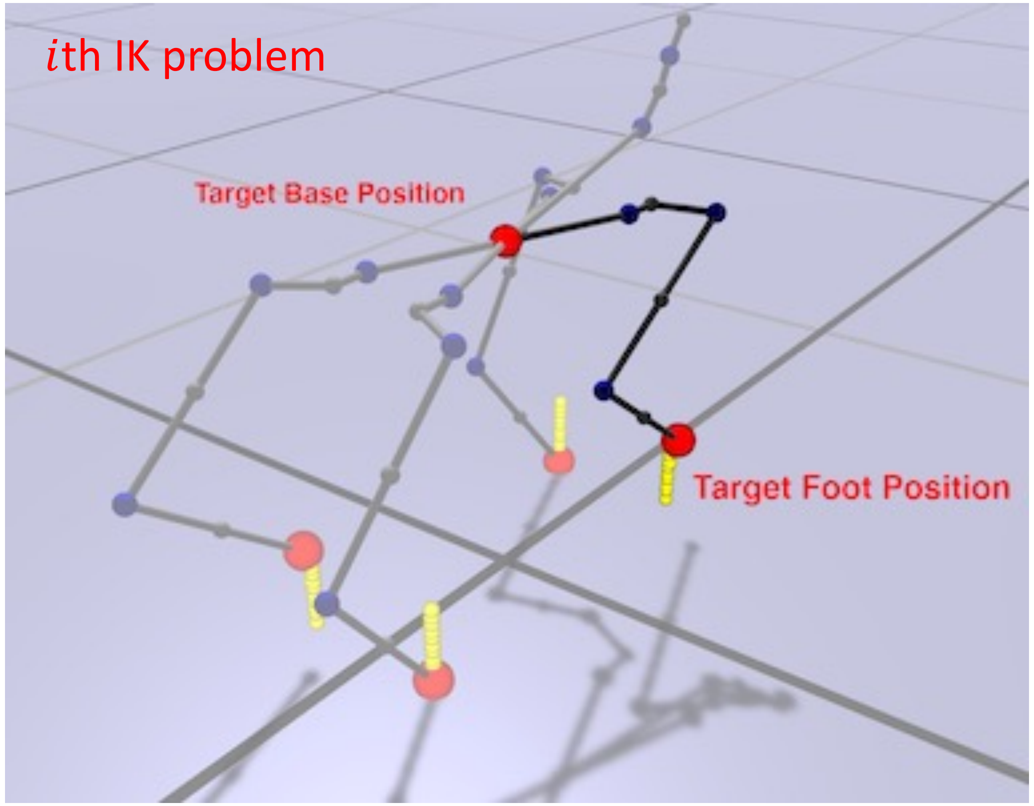
Output:
- Desired joint angles

Inverse Kinematics Overview

- We will assume the base perfectly follows the target.
- Thus, we only solve IK for the legs.
  - One leg = one IK problem

For ith IK problem

$$\mathbf{q}^{\text{desired},i} = \text{argmin}_{\mathbf{q}} \parallel \mathbf{p}_{\text{EE},i}{}^{\text{target}} - \mathbf{FK}_i(\mathbf{q}) \parallel_2{}^2$$



- Total 4 **independent** IK problems
  → Sum up solutions for $\mathbf{q}^{\text{desired}}$

$$\mathbf{q}^{\text{desired}} = \sum_{i=1}^{4} \mathbf{q}^{\text{desired},i}$$

Baseline Exercises

- Ex.1 Forward Kinematics (20%)
- Ex.2-1 Inverse Kinematics - Jacobian by Finite Difference (20%)
- Ex.2-2 Inverse Kinematics - IK solver (20%)
- Ex.3 Trajectory Planning (20%)

Ex.2-1 Inverse Kinematics – Jacobian by FD

- For now, we will compute Jacobian matrix with FD

$$\mathbf{FK}(\mathbf{q}) = \left[\mathrm{FK}_x(\mathbf{q}), \mathrm{FK}_y(\mathbf{q}), \mathrm{FK}_z(\mathbf{q})\right]^{\mathrm{T}}$$

$$\mathbf{J} = \frac{\partial \mathbf{FK}}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial \mathrm{FK}_1}{\partial q_1} & \cdots & \frac{\partial \mathrm{FK}_1}{\partial q_j} & \cdots & \frac{\partial \mathrm{FK}_1}{\partial q_n} \\ \frac{\partial \mathrm{FK}_2}{\partial q_1} & \cdots & \frac{\partial \mathrm{FK}_2}{\partial q_j} & \cdots & \frac{\partial \mathrm{FK}_2}{\partial q_n} \\ \frac{\partial \mathrm{FK}_3}{\partial q_1} & \cdots & \frac{\partial \mathrm{FK}_3}{\partial q_j} & \cdots & \frac{\partial \mathrm{FK}_3}{\partial q_n} \end{bmatrix}$$

"Central difference"

$$\mathbf{J}_{i,j} = \frac{\mathrm{FK}_i(\mathbf{q} + \mathbf{h}_j) - \mathrm{FK}_i(\mathbf{q} - \mathbf{h}_j)}{2h}$$

jth component

$$\mathbf{h}_j = h[0, \dots, 0, 1, 0, \dots, 0]^{\mathrm{T}}$$

Small perturbation

Ex.2-2 IK Solver

You may need to implement line search and regularization for a better convergence...

- Choose one of the following methods.
  - Gradient descent

$$\mathbf{q}^{i+1} = \mathbf{q}^{i} + \gamma \mathbf{J}^{\mathrm{T}} ({\mathbf{p}_{\mathrm{EE}}}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q}))$$

  - Newton's method

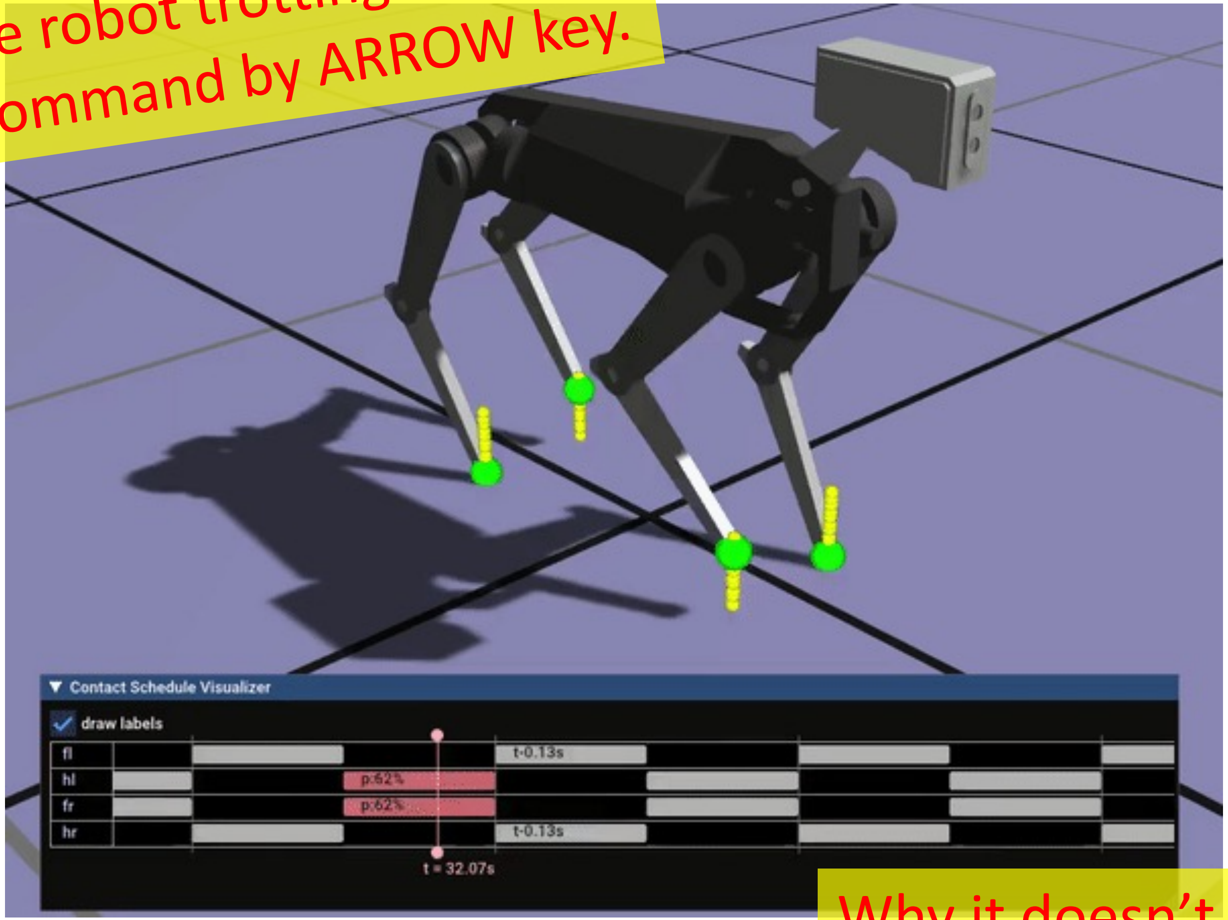
$$\mathbf{q}^{i+1} = \mathbf{q}^{i} + [-\frac{\partial \mathbf{J}}{\partial \mathbf{q}}^{\mathrm{T}} ({\mathbf{p}_{\mathrm{EE}}}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q})) + \mathbf{J}^{\mathrm{T}}\mathbf{J}]^{-1} \mathbf{J}^{\mathrm{T}} ({\mathbf{p}_{\mathrm{EE}}}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q}))$$

  - Gauss-Newton method

$$\mathbf{q}^{i+1} = \mathbf{q}^{i} + (\mathbf{J}^{\mathrm{T}}\mathbf{J})^{-1} \mathbf{J}^{\mathrm{T}} ({\mathbf{p}_{\mathrm{EE}}}^{\mathrm{target}} - \mathbf{FK}(\mathbf{q}))$$

Once you finish Ex.2 …

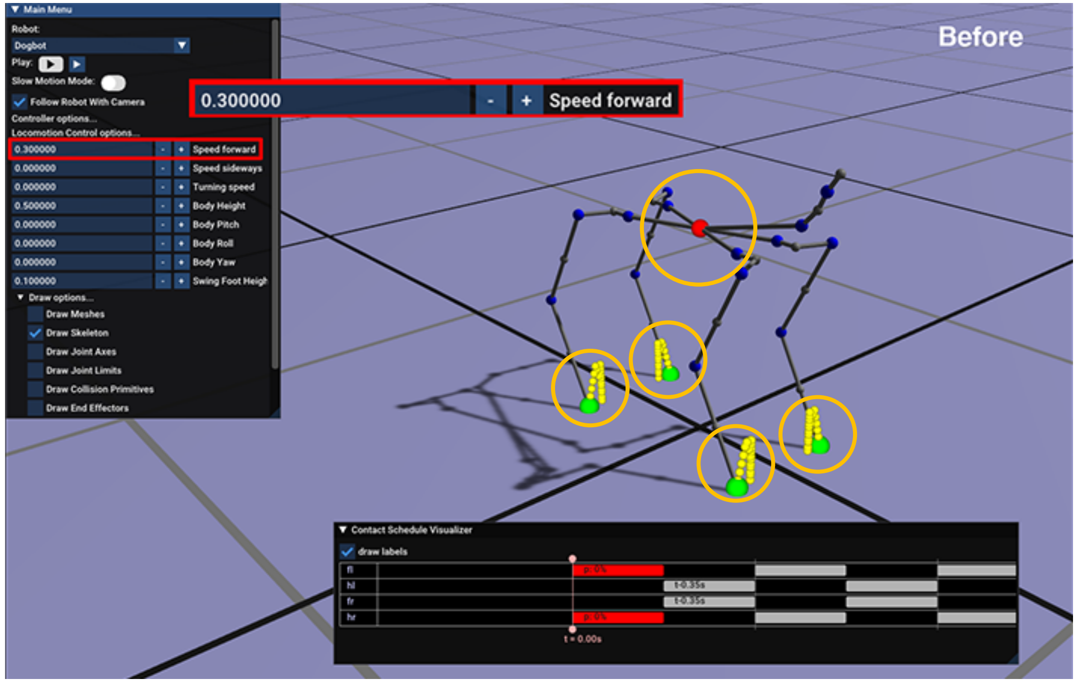
You should see the robot trotting in place.
Now let's give a command by ARROW key.



Why it doesn't its body move at all?

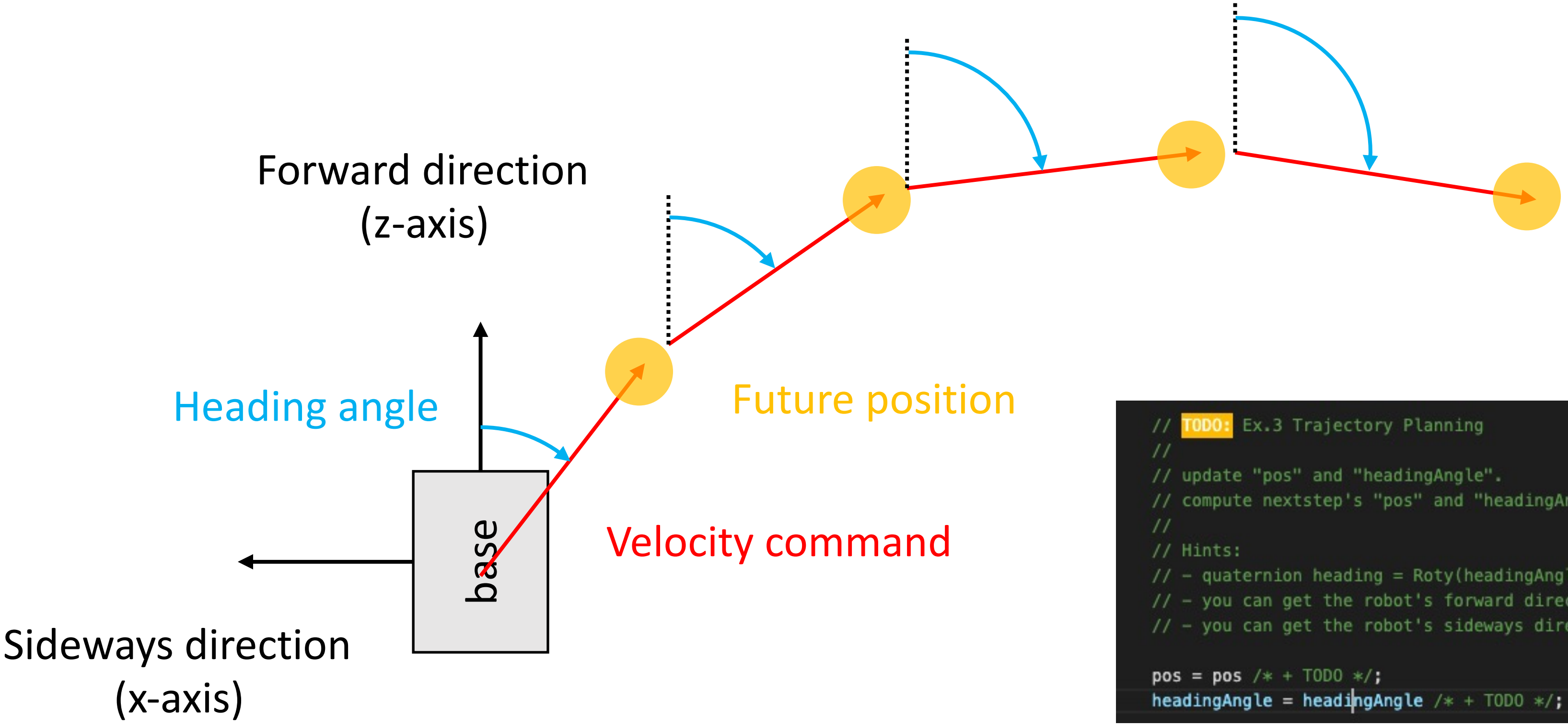
Ex.3 Trajectory Planning

- The target trajectories are not updated at all!
  - We should update the trajectories according to user command.



Ex.3 Trajectory Planning

- We are going to integrate target velocity command to get future position of the base.

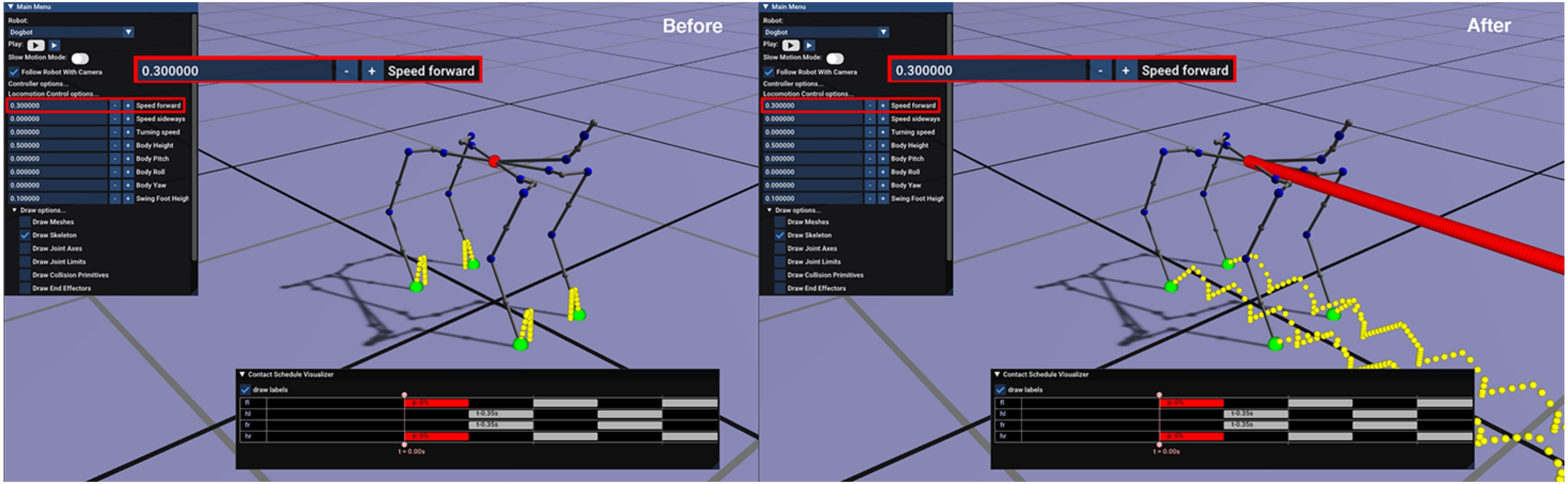


Note. We use y axis as up-axis

```
// TODO: Ex.3 Trajectory Planning
//
// update "pos" and "headingAngle".
// compute nextstep's "pos" and "headingAngle" by integrating "vForward", "vSideways" and "turningSpeed".
//
// Hints:
// - quaternion heading = Roty(headingAngle). (we use y-up world frame coordinate)
// - you can get the robot's forward direction vector from robot->forward
// - you can get the robot's sideways direction vector by RBGlobals::worldUp.cross(robot->forward)

pos = pos /* + TODO */;
headingAngle = headingAngle /* + TODO */;        You, seconds ago • Uncommitted changes
```

Ex.3 Trajectory Planning



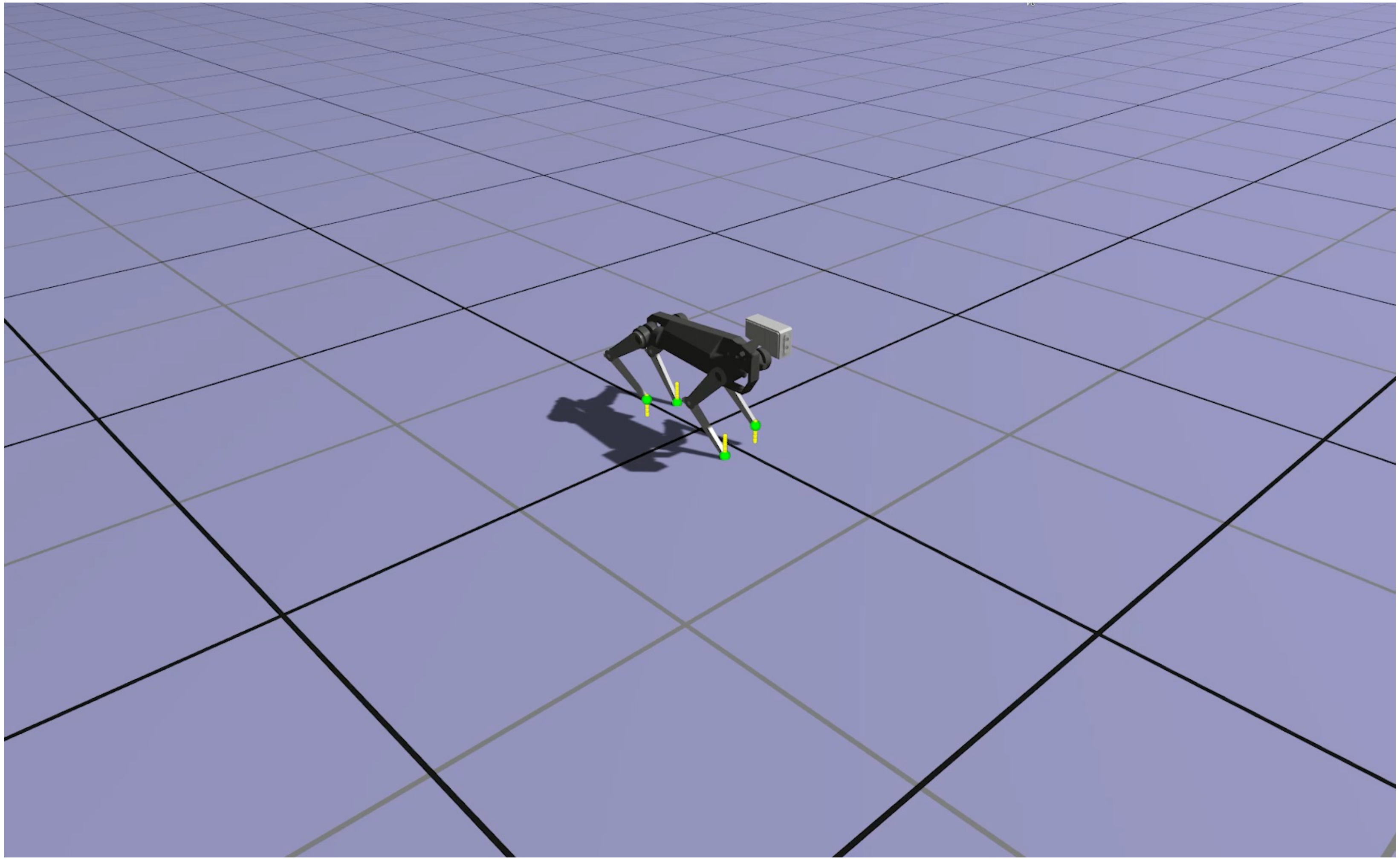
Ex.3 Trajectory Planning

- Sidenote
  - Once you complete Ex.3, the feet trajectory is also updated by user commands. (I implemented a foot trajectory planning strategy already)
  - Planning the feet trajectories is a bit more tricky.
    - We use a strategy called “*Raibert Heuristic*”
    - If you are interested, please read

M. Raibert et al., Experiments in Balance with a 3D One-Legged Hopping Machine, 1984

Marc Raibert was a professor at MIT and founded Boston Dynamics Company

Baseline Exercises



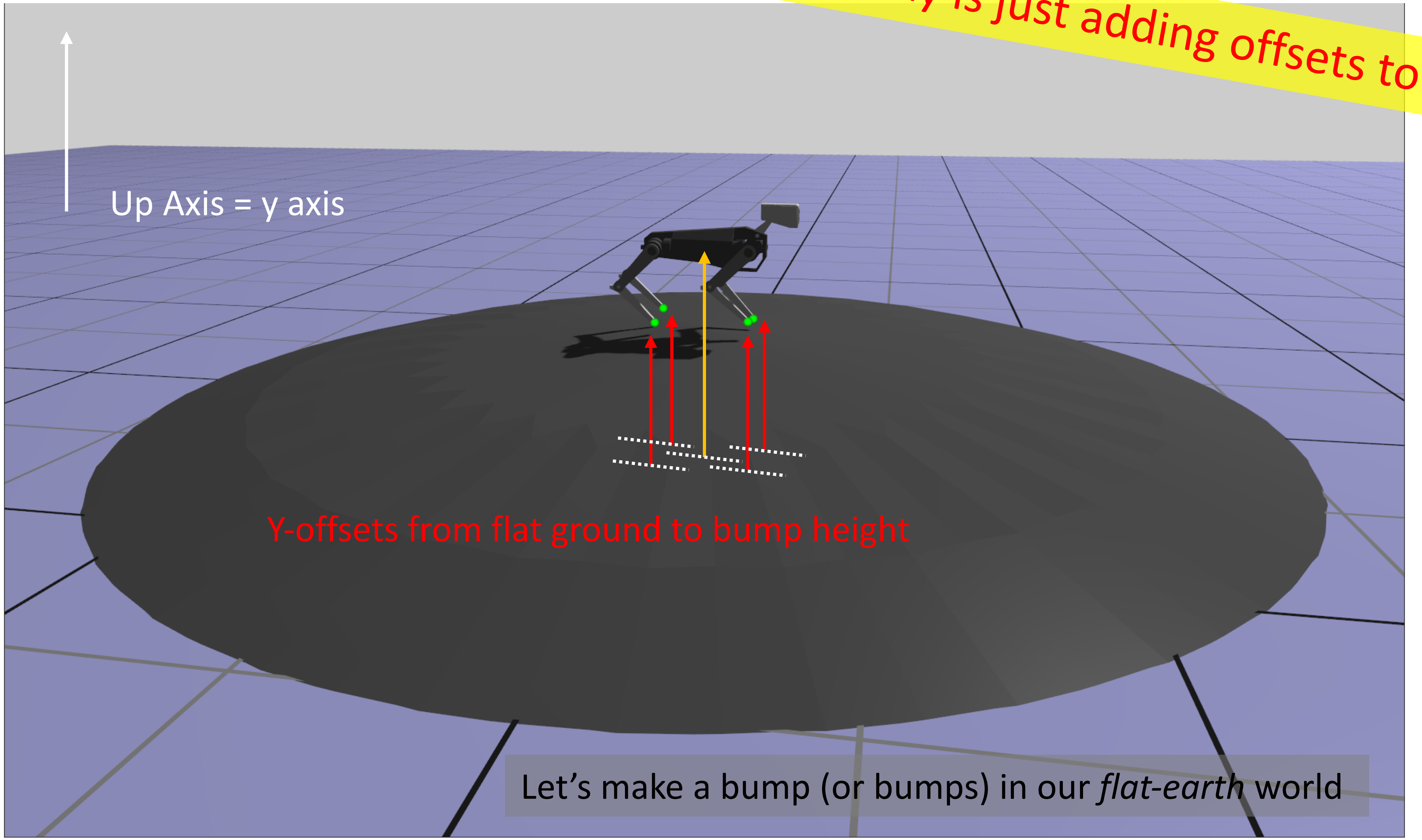
Advanced Exercises

- Ex.4 Analytic Jacobian – Unit test will check your implementation
- Ex.5 Uneven Terrain
  - 5% for walking on terrain
  - 5% for visualization

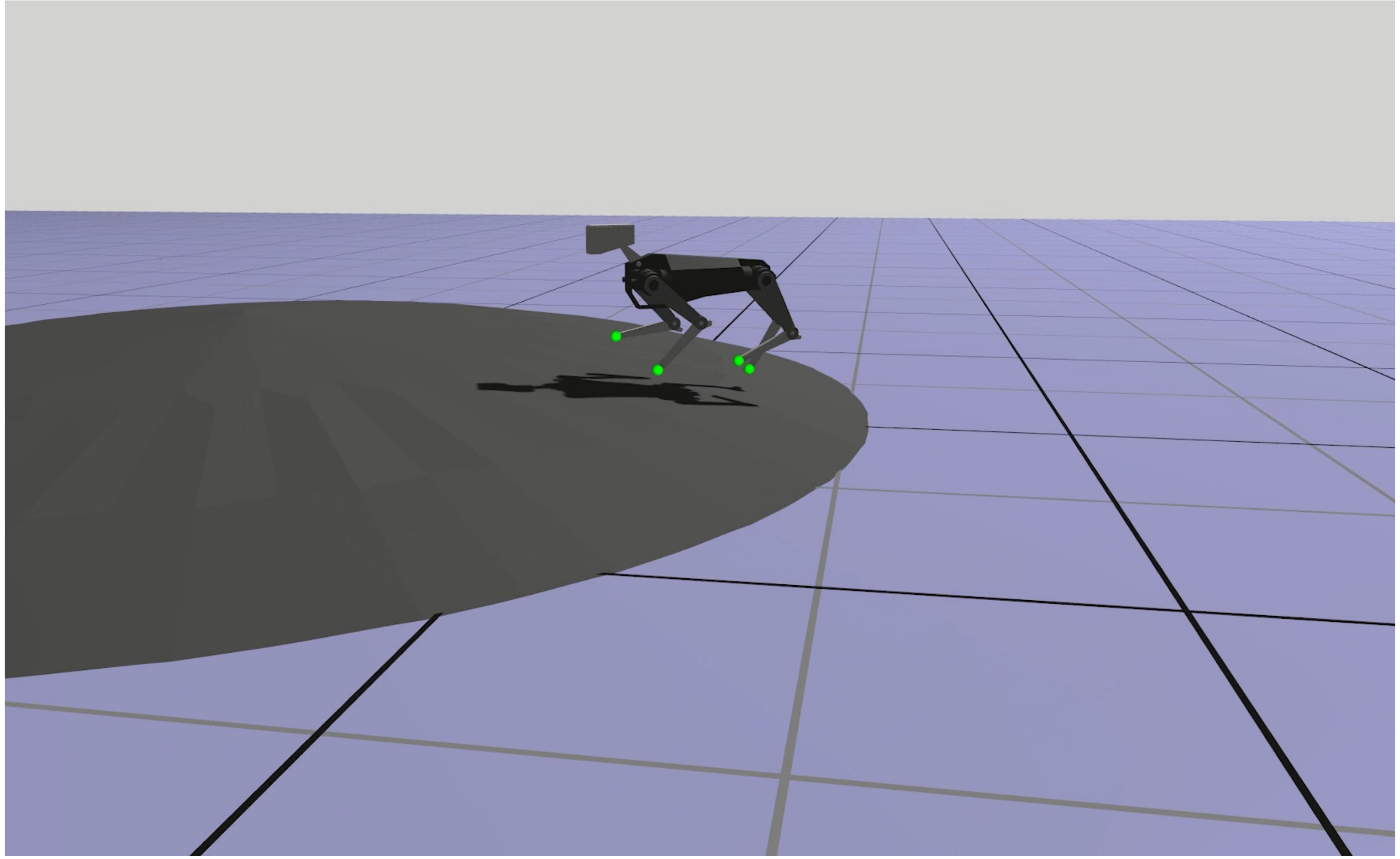
You have a full freedom. Create your own terrain.

Ex.5 Uneven Terrain

Easiest way is just adding offsets to your targets



Ex.5 Uneven Terrain



Hand-in and Grading Scheme

- Hand-in (April 2nd, 2021, 18:00 CEST)
  - A short demo video for baseline exercises (Ex.1 - Ex. 3)
  - A short demo video for Ex.5
  - Unit test all passes
  - Code pushed to your github repo.
- Evaluation
  - Baseline exercises implementation (80%)
  - Advanced exercises implementation (20%)

Please do read README.md very carefully!

Can we use kinematic walking controller for a real robot?

- Unfortunately… it doesn’t work well in practice

Can you guess why? Please leave your idea in GitHub issue.

Questions?

- Please actively use GitHub issue.
  - https://github.com/cmm-21/a2/issues

- Contact me if you have other questions.
  - kangd@ethz.ch