Humanoid and wheeled-legged controllers in C++ and Python: balancing at different frequencies

Stéphane Caron

November 28, 2022

Motion control software

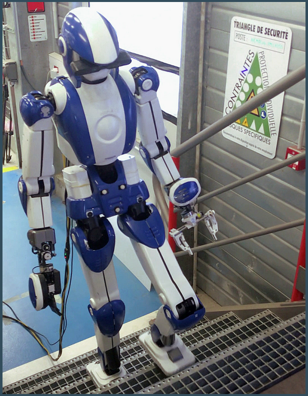


Figure 1: LIPM walking controller

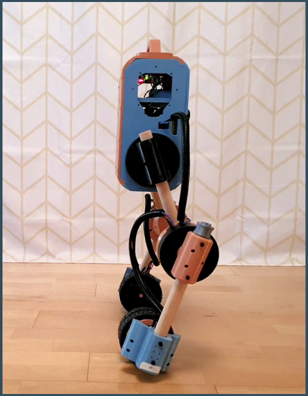


Figure 2: Pink controller

Make a robot **move** (motion) to achieve some **tasks** (control).

Examples:

- Locomotion: change position w.r.t. the world
- Manipulation: change the pose of an object w.r.t. the robot
- Folding: change the configuration of a deformable object
- Breaking: add free-flyer joint to another system ;)

Key part of the work: task formulation.

Software to implement motion control.

Part of it is specialized:

- Robot descriptions: URDF, MJCF, SDFormat, ...
- Lie algebra: Rotations $SO(3)$, transformations $SE(3)$, twists $se(3)$, ...
- Rigid body dynamics: Forward kinematics, inverse dynamics, ...
- Physics simulators: AISTsimulator, Bullet, MuJoCo, RaiSim, ...
- Optimal control: Model predictive control, reinforcement learning, ...

A lot of it is more general, *e.g.*:

- Timers and loop frequency regulation
- Logging and analysis of time series data
- Build systems, packaging and continuous integration
- Serial (I2C, SPI, ...) and data-comms protocols (CAN-FD, EtherCat, ...)

Today's scope

Motion control software for **research projects**.

(Not in today's scope: motion control software for production.)

THE POWER OF COLLABORATION

GitHub

Packaging system

`git clone this-repo-I-try`

`pip install this-pkg-I-use`

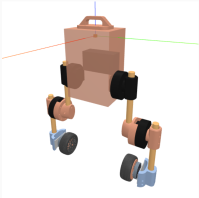
Interlude 1: Robot descriptions

Load a robot description:

```
from robot_descriptions.loaders.pinocchio import load_robot_description

robot = load_robot_description("upkie_description")
```

Visualize it:

```
from pinocchio.visualize import MeshcatVisualizer

robot.setVisualizer(MeshcatVisualizer())
robot.initViewer(open=True)
robot.loadViewerModel()
robot.display(robot.q0)
```



Setup: `pip install meshcat pin robot_descriptions`

Choose a description for the rest of the tutorial:

List: `https://github.com/robot-descriptions/robot_descriptions.py`

C++/Python motion control software

TWO PROGRAMMING LANGUAGES

Pros:

- Faster programs
- Type system

Cons:

- Build complexity
- No packaging system

Pros:

- Packaging system(s)
- Thriving ecosystem

Cons:

- Slower interpreted code
- Real-timeness?

Not covered today: Rust and Julia.

A common approach is to use **bindings**[1]:

- Pro: Performance
- Con: Overhead when API changes

An alternative is **interface description languages**:

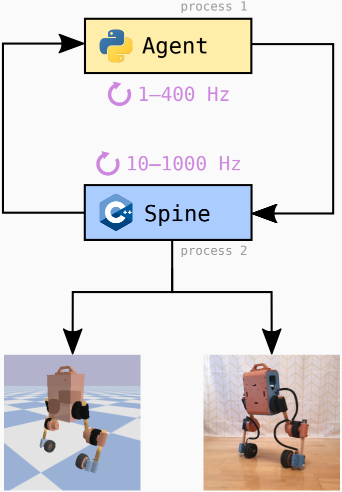
- Pro: Versioning, breaking-change detection
- Con: Overhead when API changes

Can we do better for **prototyping**?

[1] For instance nanobind: `https://github.com/wjakob/nanobind`

observation dictionary


```
{
  "imu": {
    "linear_acceleration": [
      0.10456651449203491,
      0.1155887097120285,
      -0.19683143496513367
    ],
    "angular_velocity": [
      -0.018338500218555813,
      -0.004894516096685997 5,
      0.013782314291377813
    ],
  },
  "servo": {
    "left_hip": {
      "voltage": 19,
      "velocity": -0.0188,
      "torque": -0.73,
      "q_current": -1.90,
      "mode": 10,
      "temperature": 22,
      "position": -0.637,
      "fault": 0,
      "d_current": -0.2
    },
    "right_hip": {
      "voltage": 18.5,
      "velocity": 0.0486,
      "torque": 0.62,
      "q_current": 1.8,
      "mode": 10,
      "temperature": 22,
      "position": 0.638,
      "fault": 0,
      ...
```

simulation

real thing

action dictionary


```
{
  "servo": {
    "left_hip": {
      "position": -0.6597,
      "velocity": 6.0939e-06,
      "kp_scale": 2.0,
      "kd_scale": 1.7
    },
    "left_knee": {
      "position": 1.0698,
      "velocity": -2.5212e-05,
      "kp_scale": 2.0,
      "kd_scale": 1.7
    },
    "left_wheel": {
      "position": null,
      "velocity": -2.6646
    },
    "right_wheel": {
      "position": null,
      "velocity": 2.66464
    }
    "right_hip": {
      "position": 0.658,
      "velocity": -5.149,
      "kp_scale": 2.0,
      "kd_scale": 1.7
    },
    "right_knee": {
      "position": -1.070,
      "velocity": 2.1e-05,
      "kp_scale": 2.0,
      "kd_scale": 1.7
    },
    ...
```

Vulp is an inter-process communication (IPC) protocol:

- Lightweight: fits in a 6-state, 14-transition state machine
- Based on **dictionaries** for serialization/logging
- Reference libraries in C++, Python, (Rust?), (Julia?), ...

Vulp is designed to:

- Start prototyping in a high-level language like Python
- Move to C++ **as needed** for performance
- Provide a simulation/real switch

We will see why this is suited to **balancing** in particular.

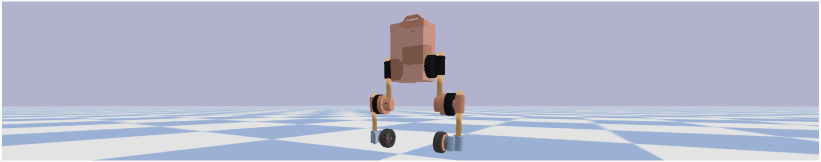
Repository: `https://github.com/tasts-robots/vulp`

Vulp comes batteries included:

```
git clone https://github.com/tasts-robots/upkie_locomotion.git
cd upkie_locomotion
./tools/bazelisk run -c opt //agents/blue_balancer:bullet
```

Bazel will download and build everything (no installation required).

Battery warning for attendees: the first build is consuming.



Repository: `https://github.com/tasts-robots/upkie_locomotion`

Definition

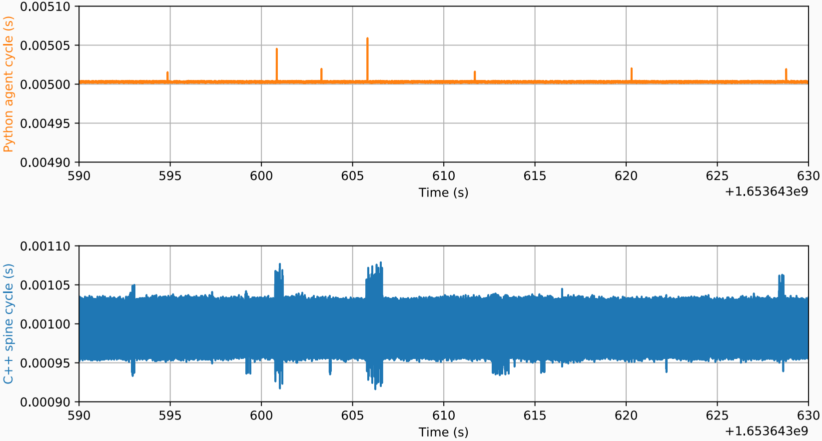
Real-time: of a system that responds to events within a predictable time.

Can Python execute control-critical code with predictable timings?

Let's run an experiment:

- **Agent** (Python) running at 200 Hz:
  - Inverse kinematics by quadratic programming
  - Wheeled balance control
- **Spine** (C++) running at 1,000 Hz:
  - Joint controller: moteus position/velocity/torque
  - State observers: floor contact, wheel odometry
  - I/O: logging, joystick, temperature

Run on a Raspberry Pi Model B (Quad core ARM Cortex-A72 @ 1.5GHz) using the default Raspberry Pi OS kernel (no PREEMPT_RT patch).



Details: `https://github.com/tasts-robots/vulp#performance`

Interlude 2: inverse kinematics

Define IK tasks:

```
from pink.tasks import BodyTask

tasks = [
    BodyTask(f"{leg}_contact", position_cost=1., orientation_cost=1.)
    for leg in ("left", "right")  # adapt to the robot you picked
]
```

Initialize task targets:

```
from pink import apply_configuration

configuration = apply_configuration(robot, robot.q0)
for task in tasks:
    task.set_target_from_configuration(configuration)
```

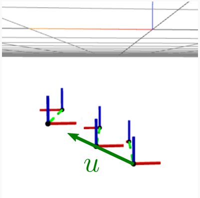
Setup: `pip install pin-pink`

Let's display our targets for convenience:

```
import meshcat_shapes

for task in tasks:
    meshcat_shapes.frame(robot.viewer[f"{task.body}_target"])
```

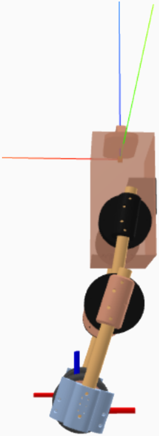
And define the trajectory of our task targets:

```
def update_targets(tasks, t, dt):
    for task in tasks:
        u = 0.2 * np.array([2.0, 0.0, 1.0])
        T = task.transform_target_to_world
        T.translation += np.sin(2 * t) * u * dt

        frame = robot.viewer[f"{task.body}_target"]
        frame.set_transform(T.np)
```



Setup: `pip install meshcat_shapes`

```python
from pink import solve_ik
from pink.utils import RateLimiter

rate = RateLimiter(frequency=200)
dt = rate.period
for t in np.arange(0., 5., dt):
    update_targets(tasks, t, dt)
    velocity = solve_ik(
        configuration,
        tasks,
        dt,
        solver="proxqp",
    )
    q = configuration.integrate(velocity, dt)
    configuration = apply_configuration(robot, q)
    robot.display(q)
    rate.sleep()
```



Setup: `pip install proxsuite pin-pink`

Inverse kinematics by numerical optimization:

- **Joint limits:** position, velocity, acceleration, torque, …
- **Regularization:** fully-define behavior by *e.g.* damping or posture tasks
    - Posture task helps define how knees should bend after stretching
- **Unfeasible targets:** handled when task error is large enough[2]
    - Task morphs into a damping task when unfeasible

Tasks can exit the feasibility workspace and re-enter elsewhere.

Achilles' heel (as of today): feasible target at task singularity.

[2] Tomomichi Sugihara. "Solvability-unconcerned inverse kinematics by the Levenberg–Marquardt method". In: *IEEE transactions on robotics* 27.5 (2011), pp. 984–991.

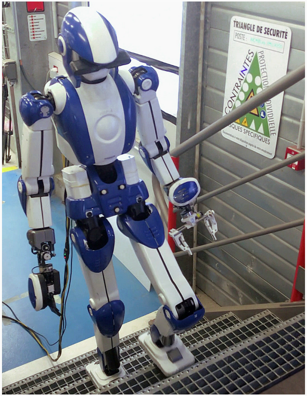
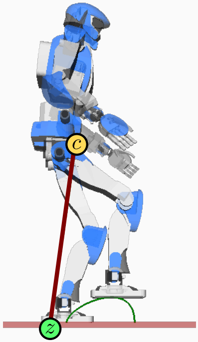


Figure 3: LIPM walking controller

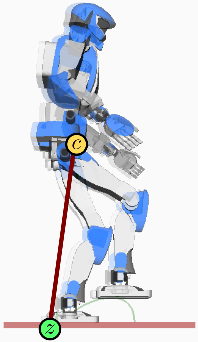
Figure 4: Pink controller

Balance control

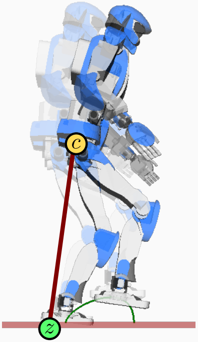
MOTIVATION



Plan$(t + \Delta t)$



Simulation$(t + \Delta t)$



Real$(t + \Delta t)$

- Whole-body dynamics:

$$M\ddot{q} + N = S^T \tau + J^T f$$

- Centroidal dynamics:

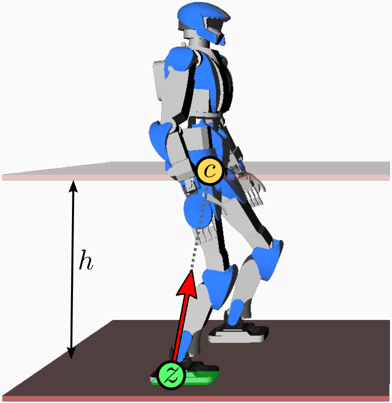
$$\ddot{c} = \frac{1}{m} \sum_i f_i$$

$$\dot{L}_c = \sum_i (p_i - c) \times f_i$$

- Linear inverted pendulum:

$$\ddot{c} = \omega^2 (c - z)$$

with $\omega^2 = g/h$ and z the ZMP



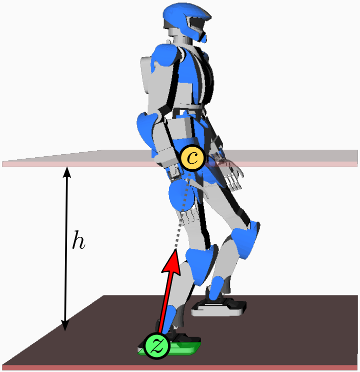
Assumptions:

- Rigid joints, sufficient power
- Conservation of angular momentum
- Constant CoM height

Equation of motion

$$\ddot{c} = \omega^2 (c - z)$$

- $\omega^2 = g/h$ is a constant
- z: *zero-tilting moment point* (ZMP)



[3]Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. "The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking pattern generation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2001.

- Linear inverted pendulum:

$$\ddot{c} = \omega^2 (c - z)$$

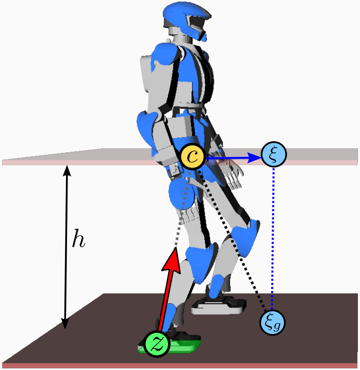
- Divergent component of motion:

$$\xi = c + \frac{\dot{c}}{\omega}$$

- Decoupled dynamics:

$$\dot{\xi} = \omega(\xi - z)$$
$$\dot{c} = \omega(\xi - c)$$

- We only need to regulate ξ



[4] Tomomichi Sugihara. "Standing stabilizability and stepping maneuver in planar bipedalism based on the best COM-ZMP regulator". In: *IEEE International Conference on Robotics and Automation*. 2009.

- DCM dynamics:

$$\dot{\xi} = \omega(\xi - z)$$

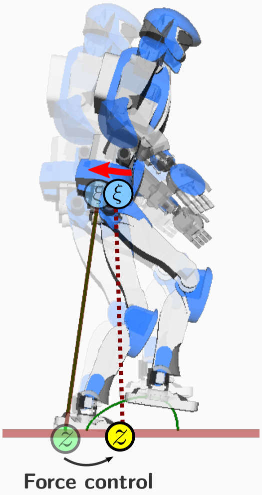
- Regulate the ZMP by **force control**:

$$z = z^d + \xi - k(\xi^d - \xi)$$

- Closed loop: $\xi \to \xi^d$

$$\dot{\xi} = k\omega(\xi^d - \xi)$$

- As long as the ZMP target is feasible...



- DCM dynamics:

$$\dot{\xi} = \omega(\xi - z)$$

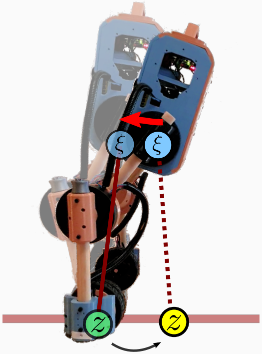
- Regulate the ZMP by **velocity control**:

$$z = z^d + \xi - k(\xi^d - \xi)$$

- Closed loop: $\xi \to \xi^d$

$$\dot{\xi} = k\omega(\xi^d - \xi)$$

- As long as the ZMP target is feasible...



Velocity control

BALANCING IS A LOW FREQUENCY TASK

We can discretize DCM dynamics $\dot{\xi} = \omega(\xi - z)$ with control period δt:

Property[5]

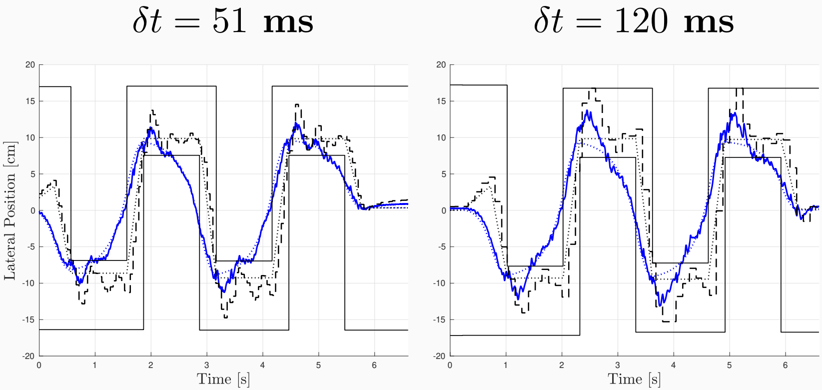
The maximum ZMP tracking error is not impacted by δt, as long as:

$$\delta t \leq \delta t_{max} := \frac{1}{\omega} \ln \left(1 + \frac{1}{k-1} \right)$$

For HRP-4 ($\omega \approx 3.5\,\mathrm{s}^{-2}$) with the LIPM walking controller ($k = 5$), this yields $\delta t_{max} = 62.5$ ms, *i.e.* a minimum control frequency of 16 Hz.

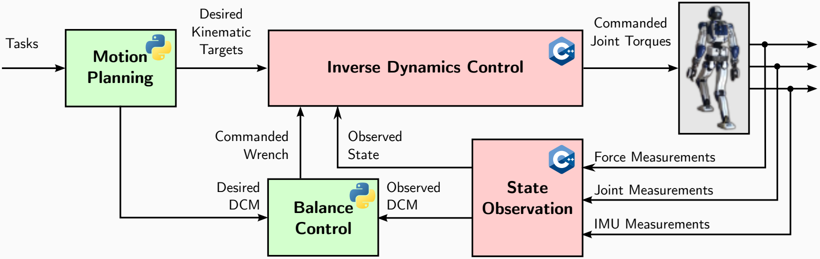
This shows that **balance control** is a low-frequency task (!)

[5] Nahuel Alejandro Villa, Johannes Englsberger, and Pierre-Brice Wieber. “Sensitivity of legged balance control to uncertainties and sampling period”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3665–3670.



[6] Nahuel Alejandro Villa, Johannes Englsberger, and Pierre-Brice Wieber. "Sensitivity of legged balance control to uncertainties and sampling period". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3665–3670.

Force control



NB: C++/Python icons denote frequency, not actual programming language.

[7] Twan Koolen, Sylvain Bertrand, Gray Thomas, Tomas de Boer, Tingfan Wu, Jesper Smith, Johannes Englsberger, and Jerry Pratt. "Design of a Momentum-Based Control Framework and Application to the Humanoid Robot Atlas". In: *International Journal of Humanoid Robotics* (2016).

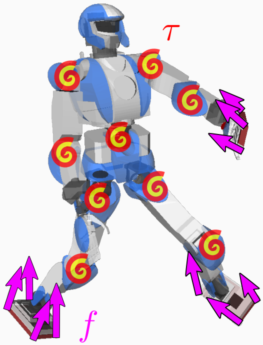
- Whole-body dynamics:

$$M\ddot{q} + N = S^T \tau + J^T f$$

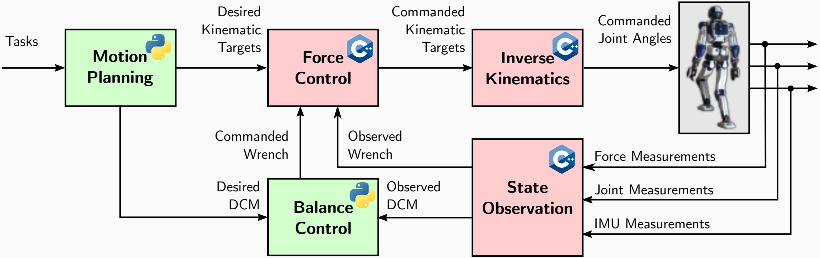
- Linear inverted pendulum task:

$$\ddot{c} = (M\ddot{q} + N)_{[0:3]} = \omega^2 (c - z^d)$$
$$\dot{L}_c = (M\ddot{q} + N)_{[3:6]} = 0$$

- Solution τ^* sent to torque controller
- Requires accurate contact estimation
- **Always** used with some impedance[8]



[8] Twan Koolen, Sylvain Bertrand, Gray Thomas, Tomas de Boer, Tingfan Wu, Jesper Smith, Johannes Englsberger, and Jerry Pratt. "Design of a Momentum-Based Control Framework and Application to the Humanoid Robot Atlas". In: *International Journal of Humanoid Robotics* (2016).



NB: C++/Python icons denote frequency, not actual programming language.

[9] Stéphane Caron, Abderrahmane Kheddar, and Olivier Tempier. “Stair Climbing Stabilization of the HRP-4 Humanoid Robot using Whole-body Admittance Control”. In: *IEEE International Conference on Robotics and Automation*. May 2019.

- Linear model:

$$\tau = K_e(\theta - \theta_e)$$

- Damping control:

$$\dot{\theta} = A(\tau^d - \tau)$$

- Closed-loop behavior:

$$\dot{\tau} = AK_e(\tau^d - \tau)$$

- Closed-loop stability: $AK_e > 0$

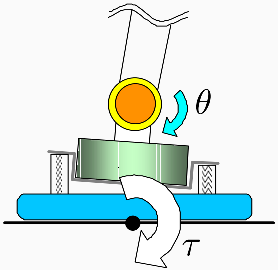


Figure adapted from [Kaj+01b]

[10] Shuuji Kajita, Kazuhito Yokoi, Muneharu Saigo, and Kazuo Tanie. "Balancing a Humanoid Robot Using Backdrive Concerned Torque Control and Direct Angular Momentum Feedback". In: *IEEE International Conference on Robotics and Automation*. 2001.

- Damping control:

$$\dot{\theta}[k] = A(\tau^d - \tau[k])$$

- Closed-loop behavior for $\tau^d = 0$:

$$\tau[k+1] = (1 - AK_e\delta t)\tau[k]$$

Closed-loop stability condition

$$A\delta t < \frac{2}{K_e}$$

- Lowering $K_e \Rightarrow$ larger A or δt
- Force control **can be** low frequency

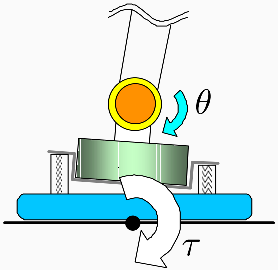
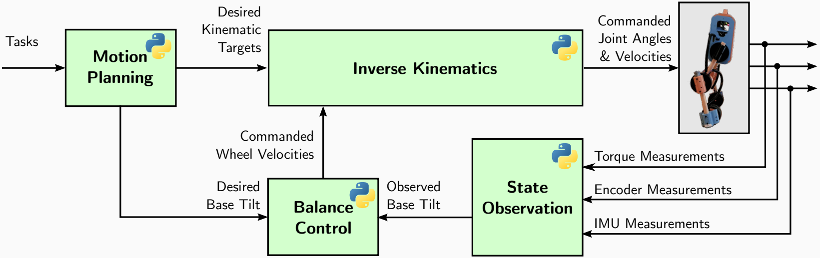


Figure adapted from [Kaj+01b]

Writeup: `https://scaron.info/robot-locomotion/contact-flexibility.html`

FULL PYTHON PIPELINE



NB: C++/Python icons denote frequency, not actual programming language.

What did we see?

Software for research projects:

- Collaborate on GitHub, release packages
- C++ when needed, higher-level language otherwise

Combining C++ and Python for motion control:

- **Vulp** action-observation loop between:
    - Agent process running at 1-400 Hz
    - Spine process running at 10-1,000 Hz
- Python **can perform real-timely** at low frequencies

Several motion control sub-tasks are in the “slow” range:

- Balance control is low frequency
- Force control can be low frequency, depending on design/scope

THANK YOU FOR PARTICIPATING!

Bibliography

[CKT19] Stéphane Caron, Abderrahmane Kheddar, and Olivier Tempier. “Stair Climbing Stabilization of the HRP-4 Humanoid Robot using Whole-body Admittance Control”. In: *IEEE International Conference on Robotics and Automation*. May 2019.

[Kaj+01a] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa. “The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking pattern generation”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2001.

[Kaj+01b] Shuuji Kajita, Kazuhito Yokoi, Muneharu Saigo, and Kazuo Tanie. “Balancing a Humanoid Robot Using Backdrive Concerned Torque Control and Direct Angular Momentum Feedback”. In: *IEEE International Conference on Robotics and Automation*. 2001.

[Koo+16] Twan Koolen, Sylvain Bertrand, Gray Thomas, Tomas de Boer, Tingfan Wu, Jesper Smith, Johannes Englsberger, and Jerry Pratt. “Design of a Momentum-Based Control Framework and Application to the Humanoid Robot Atlas”. In: *International Journal of Humanoid Robotics* (2016).

[Sug09] Tomomichi Sugihara. “Standing stabilizability and stepping maneuver in planar bipedalism based on the best COM-ZMP regulator”. In: *IEEE International Conference on Robotics and Automation*. 2009.

[Sug11] Tomomichi Sugihara. “Solvability-unconcerned inverse kinematics by the Levenberg–Marquardt method”. In: *IEEE transactions on robotics* 27.5 (2011), pp. 984–991.

[VEW19] Nahuel Alejandro Villa, Johannes Englsberger, and Pierre-Brice Wieber. “Sensitivity of legged balance control to uncertainties and sampling period”. In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3665–3670.