Robotics MVA – Final exam topics

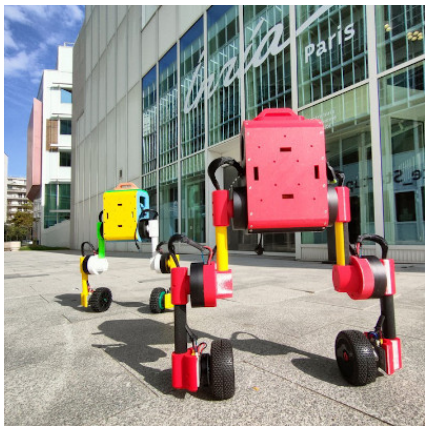
Topic * – Your own project

Context: You are welcome, and encouraged, to propose your own validation project, whether it is an article study or a project.

Goals: Practice the robotics you have learned on something interesting. Have fun.

Project: you (= your team of size 2) decide on a question of interest. It has to deal with robotics and at least one of the main topics seen in class. You should then do some literature review to explain the state of the art, propose a solution, implement it and show your results.

Topic 1 – RL to balance a wheeled biped robot



Context: Upkie [1] is a wheeled biped robot developed in the Willow team at Inria. It can balance itself upright and traverse various terrains. One metric for the robustness of a balancing policy is the maximum push that can be applied to the robot before it falls. In this topic, we focus on the maximum sagittal force over one second, which we will write MSFOS for short.

Goal: evaluate the balancing performance of a linear feedback controller, then train a more robust behavior by reinforcement learning of a neural-network policy.

Project plan:

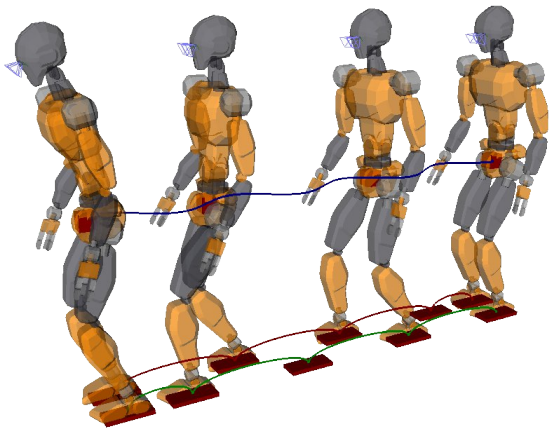
1. Start up a simulation following the instructions from the upkie repository
2. Try out a first balancing policy using only pure linear feedback of the base pitch angle
3. Using the environment API, apply a *sagittal* force to the robot for 1 second. What is the resulting MSFOS (in N) that you can apply before the robot falls ?
4. Update your policy to linear feedback of the full observation vector. How much improvement in MSFOS can you achieve ?
5. From there on, let us switch to training a neural-network policy by proximal policy optimization (PPO) [3]. Clone the PPO balancer and run the pre-trained policy. How well does it perform in MSFOS ?
6. Run the training script to train a new policy, adjusting the number of processes to your computer. What is the best number of processes for your computer ?
7. Run your trained policy and check its MSFOS
8. Improve the policy using one of the techniques seen in class that is not implemented in the PPO balancer : either curriculum learning, or reward shaping, or teacher-student distillation.
9. *Extension* **:** the *UpkieServos* Gymnasium environment allows us to send position or velocity commands to each joint of the robot. Check out the new observations and actions from this environment : do they contain everything we need for balancing ?
10. Train a balancing policy for the *UpkieServos* environment . Does it improve MSFOS out of the box ?
11. *Action shaping* is a way to make the policy less end-to-end by adding some engineering to the environment. For instance, since the two limbs of the robot have roughly the same length, it is usually a good idea to set knee ankle = ±2 * hip angle so that the action becomes [crouching, wheel velocity] for each leg. Add some action shaping to your environment. Can you improve the MSFOS of the resulting policies ?

References:

- [1] Upkie wheeled biped robots (2022–2024)
- [2] Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementations.
- [3] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Topic 2 – Model predictive control for horizontal bipedal locomotion

Context: Model predictive control (MPC) was historically one of the first successful applications of optimal control to robot locomotion. It was used to generate center-of-mass trajectories in an open-loop fashion ("walking pattern generation"), which were then executed on a real robot with the addition of a proper feedback controller ("stabilization"). Today model predictive control is directly used in closed loop (performing both walking pattern generation and stabilization in a single step) and with larger models than a point-mass (for instance a lumped rigid body), but the core ideas from center-of-mass MPC are still alive and well.



Goals: In this topic, we will start from the seminal MPC formulation as a quadratic program (QP) on a linear reduced model, and connect it to a visualization of our humanoid robot.

Project plan:

1. Read reference [1] below.
    a. Is it entirely correct? List what you find.
2. Reproduce the formulation and resolution of such a QP problem in Python with the library of your choice.
    a. You can use qpsolvers package to solve quadratic programs.
    b. Some tips and code snippets in this tutorial. Note that the *pymanoid* library mentioned in this tutorial has been discontinued, so you will need to adapt anything you see there to Pinocchio and your own knowledge.
3. Visualize the resulting COM trajectories (with the COM represented as a sphere moving in 3D) in the framework of your choice.
    a. *Bonus (optional) step***:** Visualize the robot tracking the trajectory as well, as in the above figure.
        i. You can use the Pink package to load a robot model in Pinocchio, make it track the COM target, and visualize it.
    b. MeshCat is a convenient visualizer used in robotics, ready-to-use with Pinocchio.
4. Choose one of the following extensions:
    a. **Extension F:** external forces are applied to the robot and the MPC should recompute footsteps accordingly.
        i. Check out reference [2].
        ii. How much external force can it sustain, and with respect to which relevant parameters?
    b. **Extension R:** the robot wants to track a circular rather than linear path. How can we do this?

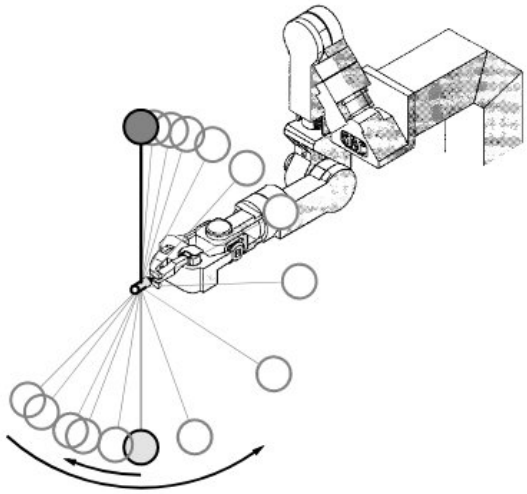
i. Hint: Constrain the ZMP to a small square inside the intersection of footstep areas with various orientations.
ii. Check out references [2] and [3].
iii. What is the minimum radius of curvature it can track, and with respect to which relevant parameters?

References:

- [1] Pierre-Brice Wieber. Trajectory Free Linear Model Predictive Control for Stable Walking in the Presence of Strong Perturbations. IEEE-RAS International Conference on Humanoid Robots, 2006, Genova, Italy.
- [2] Andrei Herdt, Holger Diedam, Pierre-Brice Wieber, Dimitar Dimitrov, Katja Mombaur, et al. Online Walking Motion Generation with Automatic Foot Step Placement. Advanced Robotics, 2010, Special Issue: Section Focused on Cutting Edge of Robotics in Japan 2010, 24 (5-6), pp.719-737.
- [3] Scianca, Nicola, et al. MPC for humanoid gait generation: Stability and feasibility. IEEE Transactions on Robotics 36.4 (2020): 1171-1188.

Topic 3 – Learning pendulum swing up from demonstration

Context: In their 1997 work *Robot Learning from Demonstration* [1], Atkeson and Schaal learned both reward and model from human demonstrations. They showed pendulum swing ups performed by a real robotic arm. Jump forward 25 years, and some of our best manipulation systems [2] are based on behavior cloning [3].



(Figure adapted from [1].)

Goals: In this topic, we will reproduce [1] with concepts seen in class and modern robotics software. We would like to compare it to behavior cloning [3].

Project plan:

1. Read [1].
2. Adapt section 5, *Learning to balance*, using the Cart Pole environment of Gymnasium. What are the main steps to take?
    a. Note that the cart-pole environment will implement its own dynamics, replacing equation (3).
    b. You can find a ready-to-use Python implementation of *dlqr* in control.dlqr of the Python Control Systems library.
    c. Alternatively, you can document yourself on the Linear-Quadratic regulator and implement your own lqr function using for instance scipy.linalg.solve_continuous_are. (If you do, the effort will be appreciated at eval time!)
    d. Alternatively, you can use Stable-Baselines3 to train a linear policy.
3. Pick a robot arm description and load it in PyBullet.
4. Apply *pybullet.TORQUE_CONTROL* with PD gains to make the robot go to a desired configuration.
    a. That is, we have a configuration vector q (of size model.nq) and a tangent velocity vector v (of size model.nv, for your tests you can start with v=0), and we want each joint i to reach an angle q[i] and angular velocity v[i]. To achieve this, PD control sends a joint torque computed as:
        i. *tau = Kp * (q[i] – q_measured[i]) + Kd * (v[i] – v_measured[i])*.
    b. You will have to find values of the two parameters *Kp* and *Kd* that work well for the robot arm model you have selected.
5. Use closed-loop inverse kinematics to command the robot's end effector position along a horizontal line of your choosing.
    a. You can implement your own, as in the inverse kinematics assignment, or use Pink.
6. Modify the URDF file of the robot description to add a pendulum at the end effector of the arm. The pendulum will consist in:
    a. A revolute joint that will be passive (i.e. always command it with zero torque).

    b. A mass at the end of the child link of the revolute joint.
7. Can you adapt your solution from step 2 to this more realistic simulation?
8. Propose a way to adapt section 6, *Learning the swing up task*, using this simulation and any input device connected to your computer.
    a. You can pick either a parametric or nonparametric model; no need to implement both.
    b. Your solution should be feasible on a real robot as well, using some of the systems you have seen in class or in the lab.
    c. Explain how you would perform the experiment on a real robot.
9. Implement and test it, report how many samples you need to realize a successful swing up.
    a. You can use CasADi for nonlinear trajectory optimization of pendulum trajectories, or Aligator.
10. Explain the similarities and differences between this approach and behavior cloning [3].
11. *Bonus (optional) step:* Implement behavior cloning in your simulation. How many samples do you need to realize a successful swing up?

References:

[1] Atkeson, C. G., & Schaal, S. (1997, July). Robot learning from demonstration. In ICML (Vol. 97, pp. 12-20).

[2] Vanhoucke, V., (2023). Shifting Winds in Robot Learning Research.

[3] Torabi, F., Warnell, G., & Stone, P. (2018). Behavioral cloning from observation. arXiv preprint arXiv:1805.01954.

Article studies

Format: you (= your team of 2 students) read and report on a research paper from the following list.

Note that we expect you to go *beyond* just reading and summarizing a paper. We encourage you to be (1) critical of the works you read and (2) creative. For instance, **try to reproduce them** (re-implement them, test an algorithm in simulation, ...) to identify shortcomings or limitations of the assumptions made, try variations of the method to explain some of its choices, propose next steps to overcome limitations, etc.

- Approximating Robot Configuration Spaces with few Convex Sets using Clique Covers of Visibility Graphs
    - Related classes: motion planning
- Estimating 3D Motion and Forces of Person-Object Interactions from Monocular Video
    - Related classes: perception and estimation, optimal control
- Legged Locomotion in Challenging Terrains using Egocentric Vision
    - Related classes: reinforcement learning, perception, inverse kinematics
- Biped Walking Stabilization Based on Linear Inverted Pendulum Tracking
    - Example of outcome: reproduce the algorithm in PyBullet with a humanoid robot description
    - Related classes: inverse kinematics, optimal control
- Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning
    - Related classes: reinforcement learning, motion planning
- Temporal Difference Learning for Model Predictive Control
    - Related classes: optimal control, reinforcement learning
- Hierarchical quadratic programming: Fast online humanoid-robot motion generation
    - Example of outcome: Python implementation of the main algorithm
    - Related classes: inverse kinematics
- Proximal and Sparse Resolution of Constrained Dynamic Equations
    - Example of outcome: simulator prototype with bilateral constraints
    - Related classes: optimal control
- Collision Detection Accelerated: An Optimization Perspective
    - Example of outcome: reimplementation of the GJK algorithm in 2D
    - Related classes: motion planning, optimal control
- On the similarities and differences among contact models in robot simulation
    - Example of outcome: re-implement in Python a contact simulation algorithms
    - Related classes: optimal control
- Towards Generalizable Vision-Language Robotic Manipulation: A Benchmark and LLM-guided 3D Policy
    - Example of outcome: reproduce the results with the released codes and checkpoints; visualize and analyze the failure cases; train on the most challenging tasks
    - Related classes: configuration space

- ViViDex: Learning Vision-based Dexterous Manipulation from Human Videos
    - Example of outcome: re-implement object relocation in Python
    - Related classes: configuration space, reinforcement learning
- Object Goal Navigation with Recursive Implicit Maps
    - Example of outcome: reproduce the results with the released codes and checkpoints; visualize and analyze the failure cases; train and evaluate a model on a few scenes
    - Related classes: motion planning, reinforcement learning
- Another article that piques your curiosity: e-mail us about it!