Prototyping fast and agile motions for legged robots with Horizon

Francesco Ruscelli [1], Arturo Laurenzi [1], Nikos G. Tsagarakis [1], and Enrico Mingo Hoffman[2]

***Abstract*— For legged robots to perform agile, highly dynamic and contact-rich motions, whole-body trajectories computation of under-actuated complex systems subject to non-linear dynamics is required. In this work, we present hands-on applications of *Horizon*, a novel open-source framework for trajectory optimization tailored to robotic systems, that provides a collection of tools to simplify dynamic motion generation. Horizon was tested on a broad range of behaviours involving several robotic platforms: we introduce its building blocks and describe the complete procedure to generate three complex motions using its intuitive and straightforward API.**

***Paper Type* – Recent Work [1].**

I. INTRODUCTION

Optimal control (OC) is a powerful tool for designing complex and dynamic motions for robotics platforms. It seeks a trajectory that minimizes a performance index while satisfying a set of constraints to shape the resulting motion, obtaining the desired robot behaviour that guarantees feasibility and physics consistency. Horizon is an intuitive tool based on CasADi [2] that uses OC to rapidly generate robot trajectories, providing the user with a pipeline encompassing the whole problem of loading a robot model, imposing the preferred behaviour, and obtaining a dynamically-feasible motion that satisfies the user's instructions. It allows to set up and solve trajectory optimization (TO) problems in both an offline and receding horizon fashion, seeking to simplify the pipeline of optimal motion planning for robotic systems. Horizon is generic enough to include all the necessary tools to prototype a dynamic manoeuvre, co-design a robot structure, or produce periodic gaits.

II. BUILDING BLOCKS

The process of prototyping robot motions follows a well-defined structure that can be adopted for different goals and robots. This Section offers descriptions and API examples of the main blocks required to generate a dynamic motion by constructing and solving an OC problem using Horizon.

A. Model acquisition

The parser module retrieves the model so as to be compatible with the underlying symbolic framework. It provides the robot dynamics in a symbolic form, so that it can be injected into the optimization: throughout the problem definition, model information can be obtained as symbolic variables that can be used in constraints and objective functions. Leveraging Pinocchio [3], it offers useful robotics-oriented algorithms, such as direct and inverse kinematics and dynamics. The *HorizonParser* guarantees ease-of-use and compatibility, as it supports the Universal Robot Description Format (URDF).

[1]Istituto Italiano di Tecnologia (IIT), Humanoids and Human Centered Mechatronics (HHCM), Via Morego 30, 16163 Genova, Italy

[2]PAL Robotics, Carrer Pujades 77-79, 08005 Barcelona, Spain

Fig. 1. The Horizon framework is available at: `https://github.com/ADVRHumanoids/horizon`

```
urdf = open(urdf_file, 'r').read()
model = CasadiKinDyn(urdf)
```

B. Horizon specification

The TO problem can be set up as a fixed horizon, to design non-periodic manoeuvres with a final goal, or in a receding horizon fashion, to generate periodic motion or continuous movements. Horizon allows both scenarios, given a suitable number of nodes to discretize the time window of the problem. This choice implies a trade-off between the resolution of the time grid and the problem dimension.

```
prb = Problem(n_nodes)
```

C. Transcription selection

Transcription methods are used for transcribing the OC problem into a non-linear programming (NLP) one, which can be efficiently solved by state-of-the-art solvers. The input and state trajectories, parametrized as decision variables over the horizon, are connected by junction nodes and gap constraints to enforce dynamic feasibility. The user can choose among two available built-in strategies: multiple shooting and direct collocation. Once specified, Horizon formalizes it as a set of constraints on each node to guarantee continuity.

```
tr = 'multiple_shooting' #'collocation'
int_type = 'RK4' # 'leapfrog', 'euler'
Transcriptor(tr, prb, int_type)
```

D. System formulation

While its core functionalities can be used even outside the robotics scope, Horizon's main target is robotic applications, as it provides specific built-in methods and robotics-oriented

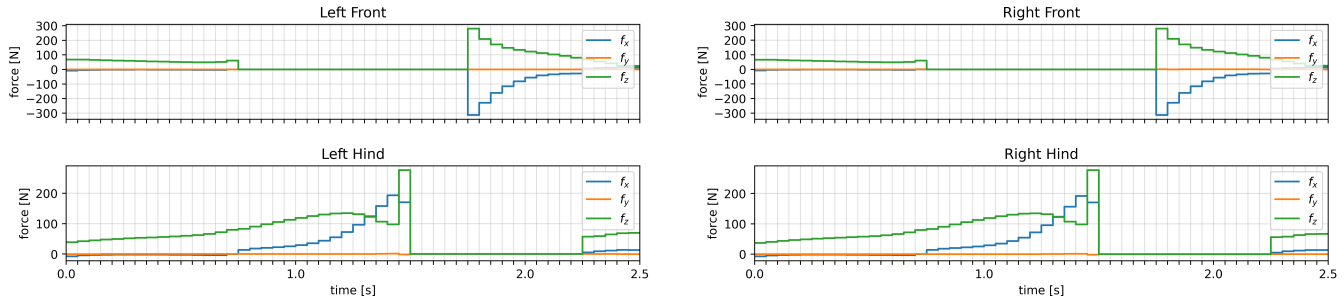

Fig. 2. Time histories of the contact forces during the leap. Notice how the robot jumps with the hinds leg and lands using the forward legs.

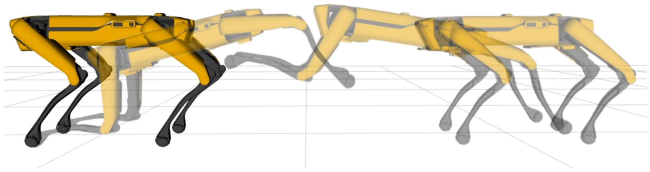

Fig. 3. Frame sequence of Spot® while performing a leaping motion. The quadruped robot jumps using the hind legs to provide thrust, and finally lands on the front legs at a specified position.

utilities. Therefore, the system taken into consideration in this work is the generic floating-base robotic system:

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{\nu}} + \boldsymbol{h}(\boldsymbol{q}, \boldsymbol{\nu}) = \mathbf{S}\boldsymbol{\tau} + \mathbf{J}_c^T(\boldsymbol{q})\boldsymbol{f}_c \tag{1}$$

which describe its dynamics with the usual symbols. The optimization problem is constructed from the generalized coordinates $\boldsymbol{q}$ and velocities $\boldsymbol{\nu}$ of the floating-base system:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{\rho} \\ \boldsymbol{\theta} \end{bmatrix}, \quad \boldsymbol{\nu} = \begin{bmatrix} \dot{\boldsymbol{p}} \\ \boldsymbol{\omega} \\ \dot{\boldsymbol{\theta}} \end{bmatrix} \tag{2}$$

where $\boldsymbol{p}, \boldsymbol{\rho}, \boldsymbol{\theta}$ are the linear position, the orientation of the floating-base and the vector of joint positions respectively. Consequently, $\dot{\boldsymbol{p}}, \boldsymbol{\omega}, \dot{\boldsymbol{\theta}}$ are the linear and angular velocity of the floating base and the vector of joint velocities.

```
q = prb.createStateVar('q', q_dim)
v = prb.createStateVar('v', v_dim)
```

E. Feasibility injection

A necessary ingredient for floating-base systems is the set of constraints to impose feasibility, i.e. the system under-actuation. The equations of motion are provided by the model (see Subsection II-A). Given the vector of Cartesian forces at each contact, the inverse dynamics torques can be retrieved from the general form of (1). The first six elements, corresponding to the effort on the virtual joints $\boldsymbol{\tau}_{fb}$, are constrained to zero, to enforce under-actuation.

```
i_d = model.InverseDynamics(contacts)
fb_tau = i_d.call(q, v, a)[:6]
prb.createConstr('feas', fb_tau)
```

F. Contact schedule

A generic manoeuvre consists of a sequence of contacts of the robot with the environment that are established and broken. In this work, contact scheduling is enforced as follows: when the contact is established, the Cartesian velocity at the end-effector frame $\dot{\mathbf{p}}_c$ is constrained to be zero and the contact forces $\boldsymbol{f}_c$ greater than zero.

```
# contact on, n_st = [stance_nodes]
DFK = model.frameVelocity('contact')
v_c = DFK(q, q_dot)['ee_vel_linear']
prb.createConstr('v_c', v_c, n_st)
prb.createConstr('f_c', f_c, n_st,
↪  upper_bounds=inf)
```

Conversely, breaking the contact is encoded by constraining the Cartesian forces at the end-effector to be zero. The constrains presented above can be combined along the horizon to assign phases of the robot motion to specific portions of the trajectory, ultimately shaping the resulting motion.

```
# contact off, n_sw = [swing_nodes]
prb.createConst('f_c', f_c, n_sw)
```

G. Solver selection

According to the authors' experience, and given the wide variety of NLP problems that come out of an OC problem, it is very likely that a one-size-fits-all solver does not exist. On the contrary, a specific solver can be more prone to finding a suitable solution to the task at hand. Horizon makes several different solvers available to the user, allowing to switch easily between them:

1) A custom implementation of a sparse, Gauss-Newton Sequential Quadratic Programming (GN-SQP) algorithm;
2) A custom implementation of a multiple-shooting Iterative Linear-Quadratic Regulator (ILQR);
3) IPOPT [4] and similar state-of-the-art solvers.

```
slv_type = 'ipopt' # 'ilqr', 'gnsqp'
solver = Solver(slv_type, prb, opts)
```

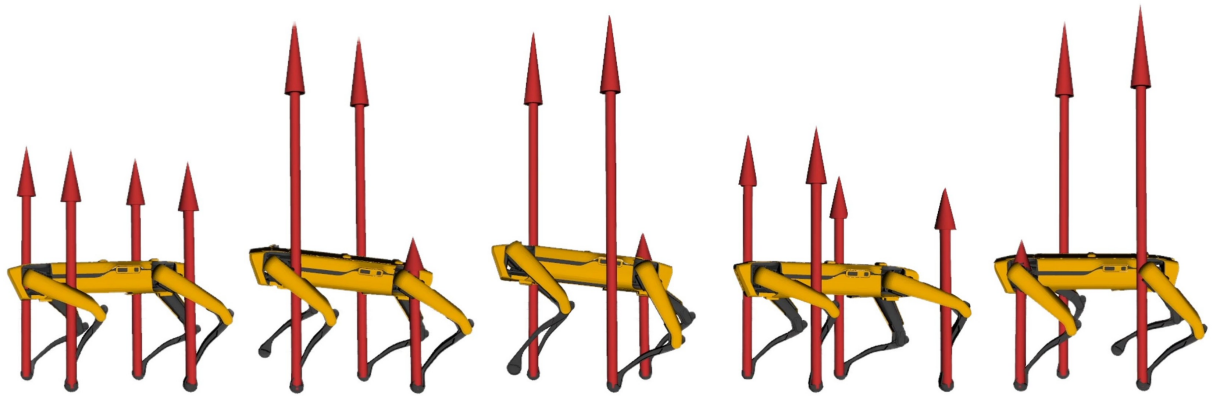

Fig. 4. Frame sequence of Spot® walking in a receding horizon fashion.

III. PROTOTYPING A MOTION

The building blocks presented in Section II can be assembled to generate a desired dynamic motion of a robot. Horizon allows to combine them seamlessly in a simple and intuitive way: this Section describes how three motions are generated, drawing parallels between the theoretical design of the OC problems and their implementation in Horizon.

A. Common features

Two features are common to the robotic applications presented in this section: first, the formulation of the dynamical system: we choose to assemble the state vector of the system as joint position and velocity combined, whereas the input is chosen as the vector of generalized accelerations and forces:

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{q} \\ \boldsymbol{\nu} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \dot{\boldsymbol{\nu}} \\ \boldsymbol{f}_c \end{bmatrix} \tag{3}$$

```
# acc dim: n_v # f dim: n_f
a = prb.createInputVar('a', n_v)
f_c = prb.createInputVar('f_c', n_f)
```

Given the state and the input as decision variables, Horizon allows to inject the system dynamics in the problem:

```
xdot = getDynamics()
prb.setDynamics(xdot)
prb.setDt(dt)
```

The second common feature consists of the TO initial conditions: on the first node the joint position is set to the initial robot configuration while the velocity is bound to zero.

```
# initial bounds (at node 0)
q.setBounds(lb=q0, ub=q0, n=0)
v.setBounds(lb=0, ub=0, n=0)
```

Finally, regularization terms are introduced for the input: both the velocity $\boldsymbol{\nu}$ and the contact forces $\boldsymbol{f}_c$ are minimized.

```
prb.setCost('min_f_c', f_c)
prb.setCost('min_v', v)
```

B. Leaping

The first optimization problem ought to find joint acceleration, torques and contact forces that correspond to a leap motion of a quadruped robot. The model acquired (see II-A) is Spot®. The finite-horizon (see II-B) is selected, as the motion is aperiodic. Generally, a finite-horizon trajectory requires terminal constraints: we impose the zero velocity at the end of the motion, as the robot should be standing still. This rule amounts to adding a single constraint at the last node forcing the joint velocity to be zero:

```
prb.setFinalConstr('v_final', v)
```

The transcription method in II-C is chosen as the multiple shooting. The system is formulated as in II-D and II-E. The contact schedule (see II-F) to enforce a leaping motion is chosen (Figure 2 shows the contact pattern as the contact forces are constrained to zero). Finally, we decide upon IPOPT, the large scale non-linear solver. To obtain the forward motion in the jump, a constraint is added for the floating base position $\boldsymbol{p}$ at the last node. Additionally, a postural for the robot is set to ensure a valid final pose:

$$\boldsymbol{p}^N = \boldsymbol{p}_{\text{des}}, \quad \boldsymbol{\rho}^N = \boldsymbol{\rho}^0, \quad \boldsymbol{\theta}^N = \boldsymbol{\theta}^0 \tag{4}$$

```
prb.setFinalConstr(x[:3] - fbp_des)
prb.setFinalConstr(x[3:6] - fbr0)
prb.setFinalConstr(x[7:] - q0)
```

C. Crawling

The goal for this optimization is prototyping a 90 deg turn of the robot CENTAURO within two full gait cycles (8 steps). The robot URDF, the system formulation presented in Subsection III-A, the finite-horizon (II-B), the multiple shooting transcription method (II-C) and the contact scheduling (II-F) are selected. To obtain the twisting motion during the

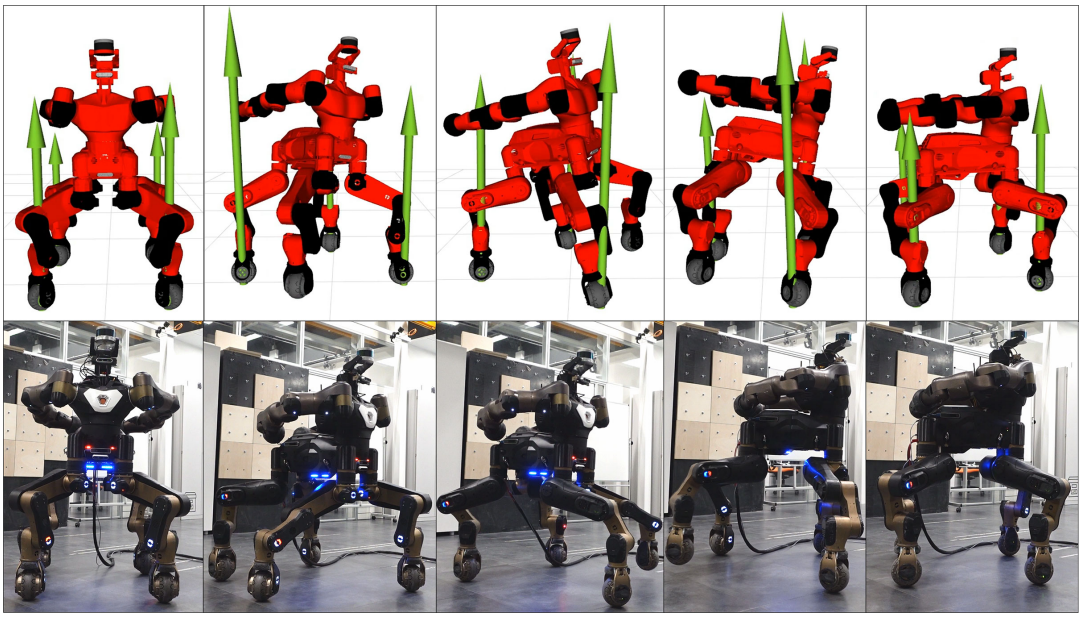

Fig. 5. Frame sequence of Centauro while performing a 90 deg in-place turn. The green arrows represent the contact forces at each foot.

jump, it is enough to constrain the robot postural and the floating base at the last node. Specifically, the orientation of the floating base is set to the desired one, while the remaining elements, i.e. the floating base $\boldsymbol{p}$ and the joint $\boldsymbol{\theta}$ positions, are constrained to the initial configuration:

$$\boldsymbol{p}^N = \boldsymbol{p}^0, \quad \boldsymbol{\rho}^N = \boldsymbol{\rho}^{\text{des}}, \quad \boldsymbol{\theta}^N = \boldsymbol{\theta}^0 \tag{5}$$

where $\boldsymbol{\rho}_{\text{des}}$ is the quaternion corresponding to a rotation of 90 deg. Finally, a so-called *clearance constraint* imposes, for each foot, a fixed swing trajectory on the z-axis.

```
FK = model.fk(frame)
p_c = FK(q)['ee_pos_linear']
prb.setConstr('clear', p_c[2] - z_sw)
```

D. Walking

This application shows how Horizon can be used in a Horizon receding horizon scenario, i.e. a continuous walking motion of Spot® (see Figure 4). Once again, the common formulation of the problem presented in Subsection III-A is used. However, the horizon (II-B) is chosen to be receding. This implies some design choices: the contact schedule presented in Subsection II-F is defined for one cycle only and fed to Horizon. Then, instead of imposing final constraints, a constant velocity reference is set as a cost function for the base link of the robot.

```
prb.createCost('v_ref', v - v_ref)
```

This translates into the optimizer searching for a periodic gait given the contact schedule. Finally, a postural cost is added during the whole motion to keep the robot configuration, while walking, as close as possible to the initial pose.

```
prb.createCost('postural', q[7:] - q0)
```

The resulting problem was solved online with the custom ILQR solver.

IV. CONCLUSIONS

Horizon was successfully tested on several robots[1] to generate a vast range of behaviours, proving to be a suitable tool for prototyping motions. Horizon aims at simplifying the process of generating agile motions of robotic platforms without losing the performance that state-of-the-art algorithms can offer. While Horizon is mature enough for complex applications, this project is still ongoing: the next milestone will extend the pipeline with a motion control layer to open the path for MPC applications, bridging the gap between simulation and real robot deployment.

REFERENCES

[1] F. Ruscelli, A. Laurenzi, N. G. Tsagarakis, and E. Mingo Hoffman, "Horizon: a trajectory optimization framework for robotic systems," *Frontiers in Robotics and AI*, p. to appear, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2022.899025

[2] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[3] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integrations (SII)*, 2019.

[4] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[1]Clips of more applications are gathered at: `https://youtube.com/playlist?list=PL7c1ZKncPan6ngCiYJHP0uciQygY1hl5I`.