2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Modeling and Control of a Hybrid Wheeled Jumping Robot

Traiko Dinev[1], Songyan Xin[1,3], Wolfgang Merkt[1,2], Vladimir Ivan[1], and Sethu Vijayakumar[1]

***Abstract*— In this paper, we study a wheeled robot with a prismatic extension joint. This allows the robot to build up momentum to perform jumps over obstacles and to swing up to the upright position after the loss of balance. We propose a template model for the class of such two-wheeled jumping robots. This model can be considered as the simplest wheeled-legged system. We provide an analytical derivation of the system dynamics which we use inside a model predictive controller (MPC). We study the behavior of the model and demonstrate highly dynamic motions such as swing-up and jumping. Furthermore, these motions are discovered through optimization from first principles. We evaluate the controller on a variety of tasks and uneven terrains in a simulator.**

I. Introduction

Wheels have been used by humanity since the Bronze age. Wheeled vehicles have enabled fast and reliable transport of goods across great distances due to their simplicity and efficiency. However, they require structured terrain such as roads or rails. Legged robots on the other hand are capable of navigating unstructured, rough terrain, including jumping over obstacles and across gaps. This comes at the cost of a more complex mechanical structure. This complexity makes the robot more expensive to control—both in terms of computation as well as energy expenditure. Wheel-legged robots can combine the best of both designs—the fast motion of a wheeled system with the ability to navigate rugged terrain via legged locomotion.

For model-based control, a model of the system is needed. It should be able to handle the rolling contact of the wheels as well as the discrete contact changes typical for stepping and jumping motions of legged robots. However, instead of adding complexity to a legged robot model by adding the rolling contact, we choose to focus on developing a simplified model with only one prismatic joint to 'mimic' the capability of a leg. This provides us with a better tool for modeling the robot behavior than some of the more complex models.

Most wheel-legged robots ([1]–[5]) are high degree of freedom nonlinear systems which do not lend themselves readily to online whole-body trajectory optimization (TO).

Previous work [5]–[7] explored kinematic motion planning where the robot is controlled like a mobile vehicle and the legs act as a suspension. By definition, these techniques do not consider the dynamics of the system.

[1]Authors are with the School of Informatics, The University of Edinburgh, Edinburgh, UK.
[2]Authors are with the Oxford Robotics Institute, University of Oxford, UK.
[3]Authors are visiting researchers at the Shenzhen Institute for Artificial Intelligence and Robotics for Society (AIRS), CUHK-SZ, China.
Email: traiko.dinev@ed.ac.uk.

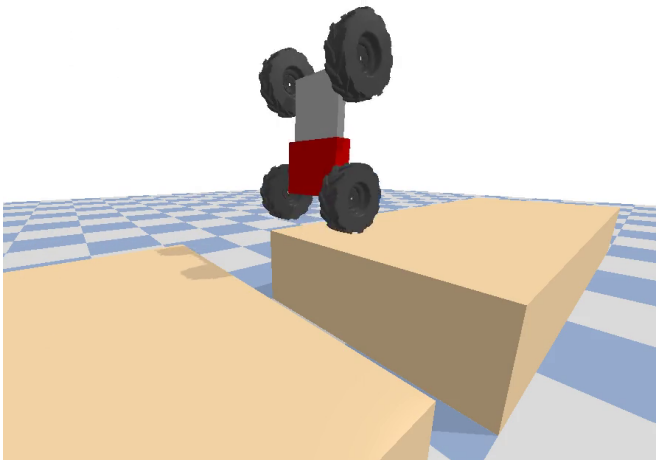


Fig. 1: The wheeled jumping robot jumps across a gap. Please find the accompanying video at https://youtu.be/j2sIWL8m2pQ

The authors of [1] use a linear quadratic regulator (LQR) for balancing the Ascento robot. They linearize the nonlinear system dynamics around the fixed point at the upright configuration. This provides an efficient approximation but it limits the range of motion around the fixed point [8, Chapter 3].

By contrast, ANYmal [2] uses a centroidal dynamics model [9] in a two-stage controller: One for the center of mass (CoM) and one for the wheels. This approach, therefore, fails to exploit potential combined synergies of the wheel and body dynamics.

Skaterbots [4] solve the full optimization. However, due to the complexity of the resulting nonlinear program, the constraints are expressed as cost terms and optimized using Newton's method. Compared to Skaterbots, we are able to solve the nonlinear problem due to the simple template model. Additionally, here, we use a Model Predictive Controller scheme, compared to the PD control used in Skaterbots.

Finally, the authors in [10] propose a Quadratic Programming (QP)-based approach to wheel-legged locomotion. In order to make the problem tractable, the z-component of the trajectory, as well as the CoM trajectory are taken as inputs to the QP-solver. In contrast, we directly optimize the z-component of the robot, thus allowing the optimizer to discover jump-like motions.

The Handle and the Flea by Boston Dynamics show remarkably dynamic motions. Flea is capable of jumping to a height of $10\,\mathrm{m}$ using combustive propulsion [11]. Handle

shows jumping motion while driving forward, as well as traversing rugged terrain. However, methods used to control these systems have not been published.

Indeed, there is a gap with respect to the control of wheel-legged robots. Staged optimization and linearization schemes inherently do not allow for the system to fully exploit its dynamics. Motions such as jumping are hard to execute using these schemes. Jumping in particular is often accomplished using a hand-tuned controller (e.g. Ascento [1] and AirHopper [12]).

Instead of considering the full rigid body dynamics, one can model the system using simpler template models that capture the essence of the robot dynamics.

A popular template model for legged robots is the Spring-Loaded Inverted Pendulum (SLIP) [13]. SLIP has been used for high speed running and jumping [14]. For wheeled balancing systems, Wheeled Inverted Pendulum models (WIP) have been extensively used as template models [15]. Template models based on SLIP and WIP can be applied to wheel-legged systems, however they will consider either only the leg or only the wheel, respectively.

In this paper, we propose the simplest template model of wheel-legged robots. Our system consists of two sets of wheels connected by a prismatic joint (see Figure 1). Our main contributions are:

- We propose the Variable-Length Wheeled Inverted Pendulum (VL-WIP) template model for wheeled-legged systems.
- We implement a motion planner using the VL-WIP model and the direct transcription method which we integrate into a Model Predictive Controller (MPC).
- We show that using the VL-WIP model, the robot can execute motions such as jumping that are only achievable using the combined action of wheels and base.
- We validate the controller in a closed loop in a physics simulator, and we evaluate the robustness of the MPC with sensor noise and on a rough terrain locomotion task. We also compare these results to a Proportional Derivative (PD) controller on a driving and balancing task.

The ability to discover dynamic motion and to exploit the predictive power of the controller stems from the dynamics model we propose.

II. Dynamics Model

We begin by modeling the system behavior using a template model we call a Variable-Length Wheeled Inverted Pendulum (VL-WIP). Our model is based on the WIP model which is used to control balancing robots [15]. WIP consists of a wheel and a pole. The pole is modeled as a point mass a fixed distance away from the wheel (see the derivation in [16]). We extend this model to include a prismatic joint and a floating base described by a z-coordinate as shown in Figure 2.

The dynamics model $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ describes the system behavior in terms of its state $\mathbf{x} = [\mathbf{q}^T \ \dot{\mathbf{q}}^T]^T$ and controls $\mathbf{u}$, where $\mathbf{q}$ is the generalized position and $\dot{\mathbf{q}}$ is the generalized velocity. The generalized position is $\mathbf{q} = [x \ z \ \phi \ l \ \theta]^T$, where

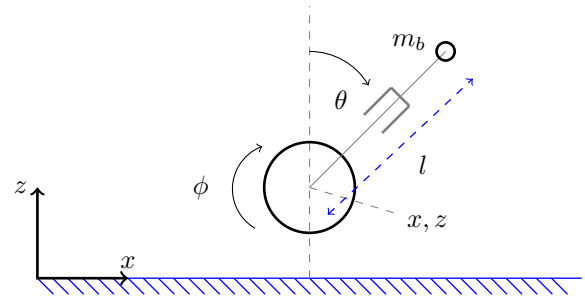


Fig. 2: The proposed Variable-Length Wheeled Inverted Pendulum (VL-WIP) model.

x, z are the coordinates of the center of the wheel, ϕ is the wheel's angle along its axis, l is the distance from the center of the wheel to the body point mass and θ is the rotation of the body from the inertia z-axis. Controls are $\mathbf{u} = [\tau \ f]^T$, where τ is actuation torque of the wheel and f is the linear force in the prismatic joint. The explicit Equations of Motion (EoM) for the system under contact can be written as:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{S}^T\mathbf{u} + \mathbf{J}_C^T\boldsymbol{\lambda} \tag{1}$$

where $\mathbf{M}$ is the mass matrix, $\mathbf{C}$ is the Coriolis matrix, $\mathbf{G}$ is the gravity vector, $\mathbf{S}$ is a selection matrix, $\mathbf{J}_C$ is the contact Jacobian and $\boldsymbol{\lambda}$ stands for the contact forces. Note that, this dynamics equation also applies when the robot is in flight and the contact forces vanish. We compute $\mathbf{M}$, $\mathbf{C}$ and $\mathbf{G}$ using Lagrangian dynamics. For the full derivation, see the Appendix. The results of this derivation are the following matrices:

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} m_t & 0 & 0 & m_b s_\theta & m_b l c_\theta \\ 0 & m_t & 0 & m_b c_\theta & -m_b l s_\theta \\ 0 & 0 & I_w & 0 & 0 \\ m_b s_\theta & m_b c_\theta & 0 & m_b & 0 \\ m_b l c_\theta & -m_b l s_\theta & 0 & 0 & m_b l^2 \end{bmatrix} \tag{2}$$

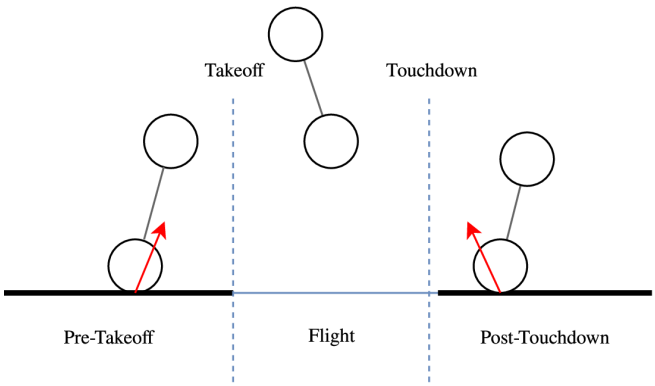
$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} 0 & 0 & 0 & 2m_b c_\theta \dot{\theta} & -m_b l s_\theta \dot{\theta} \\ 0 & 0 & 0 & -2m_b s_\theta \dot{\theta} & -m_b l c_\theta \dot{\theta} \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -m_b l \dot{\theta} \\ 0 & 0 & 0 & 2m_b l \dot{\theta} & 0 \end{bmatrix} \tag{3}$$

$$\mathbf{G}(\mathbf{q}) = \begin{bmatrix} 0 & g m_t & 0 & g m_b c_\theta & -g m_b l s_\theta \end{bmatrix}^T \tag{4}$$

where $m_t = m_b + m_w$ is the total mass which sums up the body mass m_b and the wheel mass m_w, I_w is the inertia of the wheel, $s_\theta = \sin(\theta)$ and $c_\theta = \cos(\theta)$.

The matrix $\mathbf{S}$ translates controls τ and f into the generalized coordinates of the system. f directly maps to the prismatic joint. The torque τ applied on the wheel will lead to a counter reaction torque $-\tau$ on the body of the robot. We write this down as:

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{5}$$



$$\min_{X,U,\Lambda} \sum_i W \mathbf{u}_i^T \mathbf{u}_i \qquad \text{(Minimum Energy Cost)}$$

$$\begin{aligned}
\text{s.t. } & \mathbf{x}_0 = \mathbf{x}_0^* && \text{(Start State)}\\
& \mathbf{x}_N = \mathbf{x}_N^* && \text{(Goal State)}\\
& \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\lambda}_t),\ t \in [0, N] && \text{(Dynamics)}\\
& \mathbf{x}^- \leq \mathbf{x}_t \leq \mathbf{x}^+,\ t \in [0, N] && \text{(State Bounds)}\\
& \mathbf{u}^- \leq \mathbf{u}_t \leq \mathbf{u}^+,\ t \in [0, N-1] && \text{(Control Bounds)}\\
& \text{if } t \in [T_{\text{tf}}, T_{\text{td}}] \ \text{(Flight Phase)}\\
& \boldsymbol{\lambda}_{\text{t}} = \mathbf{0} && \text{(No Ground Force)}\\
& \text{if } t \notin [T_{\text{tf}}, T_{\text{td}}] \ \text{(Ground Phases)}\\
& |\boldsymbol{\lambda}_t^x| \leq \mu \boldsymbol{\lambda}_t^z, (\mu : \text{friction coefficient}) && \text{(Friction Cone)}\\
& \boldsymbol{\lambda}_{\text{t}}^z \geq 0 && \text{(Unilateral Force)}\\
& \dot{x}_t = R_w \dot{\phi}_t && \text{(No Slip on Ground)}
\end{aligned}$$

Fig. 3: Problem formulation. On the top we illustrate the three-phases involved in the planning problem. Below is the mathematical formulation of the problem.

Finally, the contact Jacobian relates the velocity of the contact point $\dot{\mathbf{r}}_C = [\dot{x}_c\ \dot{z}_c]^T$ to the generalized velocities of the robot:

$$\dot{\mathbf{r}}_C = \mathbf{J}_C \dot{\mathbf{q}} = \begin{bmatrix} 1 & 0 & -R_w & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \dot{\mathbf{q}} \tag{6}$$

where R_w is the radius of the wheel.

Because of our system's simplicity, we have derived an analytic dynamics model. Compared to the full-body dynamics of complex systems, e.g. computed using recursive algorithms, analytic dynamics and their derivatives are faster to evaluate. Using the analytic dynamics allows us to plan motions for both the body and the wheels in a unified way. A unified planning approach allows the solver to plan for motions for which the interaction between wheels and base is key.

III. Problem Formulation and Control

This section looks at how the derived system dynamics can be incorporated into a planning and control pipeline. For motion planning, we use an optimal control formulation, namely direct transcription [17]. To account for model mismatch and perturbations during execution, we use Model Predictive Control [18], which calls the planner iteratively to replan the motion from the current robot state.

A. Planning

The generic planning problem formulation is illustrated in Figure 3. We formulate a nonlinear optimization problem (NLP) that finds trajectories $X = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N\}$, $U = \{\mathbf{u}_0, \mathbf{u}_1, \ldots \mathbf{u}_{N-1}\}$, and $\Lambda = \{\boldsymbol{\lambda}_0, \boldsymbol{\lambda}_1, \ldots \boldsymbol{\lambda}_{N-1}\}$ that satisfy the problem constraints and minimize the total energy, weighted by a normalizing term W. Here, $\mathbf{x}$, $\mathbf{u}$, and $\boldsymbol{\lambda}$ refer to the robot state, controls and contact forces respectively.

Firstly, we specify the duration T of the motion in seconds and discretize it into N knot points. At each knot point the optimizer finds states $\mathbf{x}$, controls $\mathbf{u}$, and contact forces $\boldsymbol{\lambda}$ that satisfy the system dynamics $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t, \boldsymbol{\lambda}_t)$ derived in section II.

Next we setup task specific constraints. The start state $\mathbf{x}_0^*$ and the target state $\mathbf{x}_N^*$ are specified for knots $\mathbf{x}_0$ and $\mathbf{x}_N$. At each knot point, state limits $[\mathbf{x}^-, \mathbf{x}^+]$, and control limits $[\mathbf{u}^-, \mathbf{u}^+]$ are enforced. These limits ensure the physical feasibility of the motion generated. For different tasks the state and control limits will differ. For instance, when jumping over a gap, the state limits for the three phases will indicate where the "flight" phase begins and ends. In general, a jumping motion involves three phases: a "pre-takeoff" phase, where the robot is on the ground, a "flight" phase where the robot is in the air, and a "post-touchdown" phase where the robot lands and re-balances. We specify the duration of each phases by setting the time of takeoff T_{tf}, and the time of touchdown T_{td}.

Finally, we specify ground contact constraints. Contact forces must be zero ($\boldsymbol{\lambda}_{\text{t}} = \mathbf{0}$) while the robot is in the flight phase. For the ground phases, they should stay inside the friction cone ($|\boldsymbol{\lambda}_t^x| \leq \mu \boldsymbol{\lambda}_t^z$, where μ is the friction coefficient) and have a positive vertical component ($\boldsymbol{\lambda}_{\text{t}}^z \geq 0$). For kinematics, we enforce a no-slip constraint while the robot is on the ground: $\dot{x}_t = R_w \dot{\phi}_t$, where R_w is the wheel radius. For motions without jumping, we use a similar formulation by removing the flight phase related constraints.

The planner finds an admissible trajectory for the control task. However, for executing it on a real system or in simulation, a controller is needed.

B. Control

With open-loop control, the planned trajectory cannot be tracked precisely due to tracking error introduced by model mismatch and external perturbations. Thus, we use Model Predictive Control [18] to handle the errors. Figure 4 illustrates the scheme. At each timestep, the planner re-plans the trajectory starting at the observed robot state $\mathbf{x}_{obs}$ obtained from the simulator. Then, the first action of the plan $\mathbf{u}_0$ is executed. After that, it recedes the time horizon T by the elapsed time.

A natural problem arises with the recession of T. Ideally we would keep the number of knot points N constant—this enables saving the optimizer's state and avoids costly re-initialization. The solution requires adjusting the limits of the

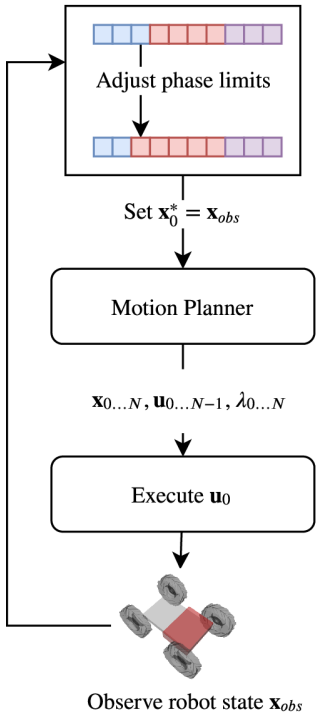


Fig. 4: MPC control pipeline. The controller re-plans a motion every timestep by setting the start state $\mathbf{x}_0^*$ to the current state in simulation $\mathbf{x}_{obs}$. It then executes the first control $\mathbf{u}_0$ and updates the phases and limits of the planner.

problem so that each phase always has the same duration. Thus, we maintain the ratio of pre / flight / post phases by shifting the limits of the problem while keeping N the same.

To implement the optimal control problem, we used CasADI [19]. As the underlying solver, we used KNITRO and selected the Interior/Conjugate-Gradient algorithm that is well-suited for large-scale sparse optimal control problems [20].

IV. Simulation Results

In order to validate our approach, we investigated several tasks, namely swing-up, balancing and jumping in the physics simulator PyBullet [21]. Please find the accompanying video at `https://youtu.be/j2sIWL8m2pQ`.

The simulated robot weighs $6\,\mathrm{kg}$ including the wheels and the dimensions of the base are $0.3 \times 0.4 \times 0.1\mathrm{m}$. The radius of the wheels is $0.17\,\mathrm{m}$ and wheels are $0.15\,\mathrm{m}$ wide.

A. Swing-up and Balance

Firstly, we demonstrate that the robot can switch from a driving mode on four wheels to a balancing mode on two wheels. This demonstrates that the robot can switch between modes without intervention. We used a two-stage controller for this task. Since controlling the robot in driving mode is outside the scope of this paper, we drive forward with a constant acceleration until we reach a velocity of $\dot{x} = 3\mathrm{m\,s^{-1}}$. Then we switch to the proposed MPC scheme.

The trajectory duration (and MPC horizon) was $T = 5\mathrm{s}$ with $N = 20$ knot points. We set control limits to $\pm 10\,\mathrm{N\,m}$ and $\pm 200\,\mathrm{N}$ for τ and f, respectively. Figure 5 shows the resulting motion. The controller planned a motion where the robot coordinates its prismatic joint and wheel to swing-up.

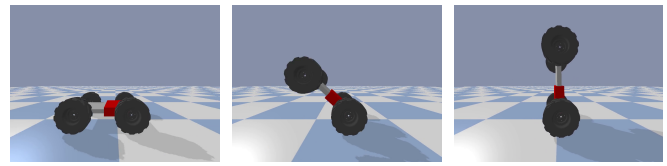


Fig. 5: Swing-up motion.

B. Driving Upright

Next we demonstrate that, in the upright mode, the robot can balance and drive forward using the proposed approach. In this experiment, the goal is to drive the robot forward $1\,\mathrm{m}$ while balancing upright. We defined the same limits on the states $[\mathbf{x}^- \ \mathbf{x}^+]$, and the controls $[\mathbf{u}^- \ \mathbf{u}^+]$ for the entire trajectory. We set torque limits for the wheels to $\pm 10\,\mathrm{N\,m}$ and force limits for the prismatic joint to $\pm 200\,\mathrm{N}$. The total time (MPC horizon) was $T = 3\mathrm{s}$ and $N = 50$ knot points.

The resulting motion is shown in Figure 6. We noticed that the optimizer first chooses to extend the prismatic joint before moving forward. This behavior can, for instance, be explained by noticing that it is easier to balance a body with a higher center of mass, which is achieved through extending the prismatic joint. Note that we did not in any way encode this behavior—the planner discovered it from first principles of optimization.

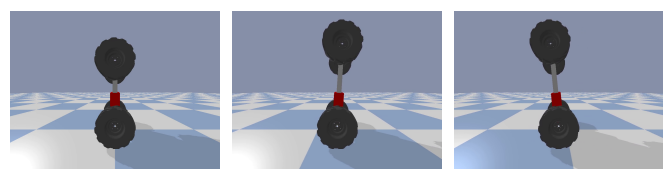


Fig. 6: Moving forward while balancing.

C. Jumping Over a Gap

Finally, we consider jumping over a gap, which uses the full multi-phase optimization described in section III. This task is more complex since different phases are involved in the motion.

We can entirely describe the jumping motion via state and control limits. In addition to torque limits $\pm 10\,\mathrm{N\,m}$ and force limits $\pm 200\,\mathrm{N}$, we set the state limits $\mathbf{q}_{lim}$ for each of the phases. Letting g^-, g^+ specify the bounds of the gap in the x direction, we obtain:

$$\mathbf{q}_{lim}^{pre\text{-}tf} = \begin{bmatrix} -\infty & R_w & -\infty & l^- & -\pi/2 \\ g^- & R_w & \infty & l^+ & \pi/2 \end{bmatrix} \tag{7}$$

$$\mathbf{q}_{lim}^{flight} = \begin{bmatrix} -\infty & R_w & -\infty & l^- & -\pi/2 \\ \infty & z^+ & \infty & l^+ & \pi/2 \end{bmatrix} \tag{8}$$

$$\mathbf{q}_{lim}^{post\text{-}td} = \begin{bmatrix} g^+ & R_w & -\infty & l^- & -\pi/2 \\ \infty & R_w & \infty & l^+ & \pi/2 \end{bmatrix} \tag{9}$$

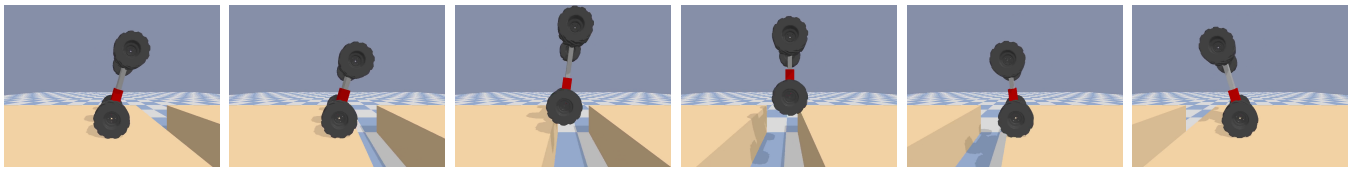


Fig. 7: Jumping over a gap.

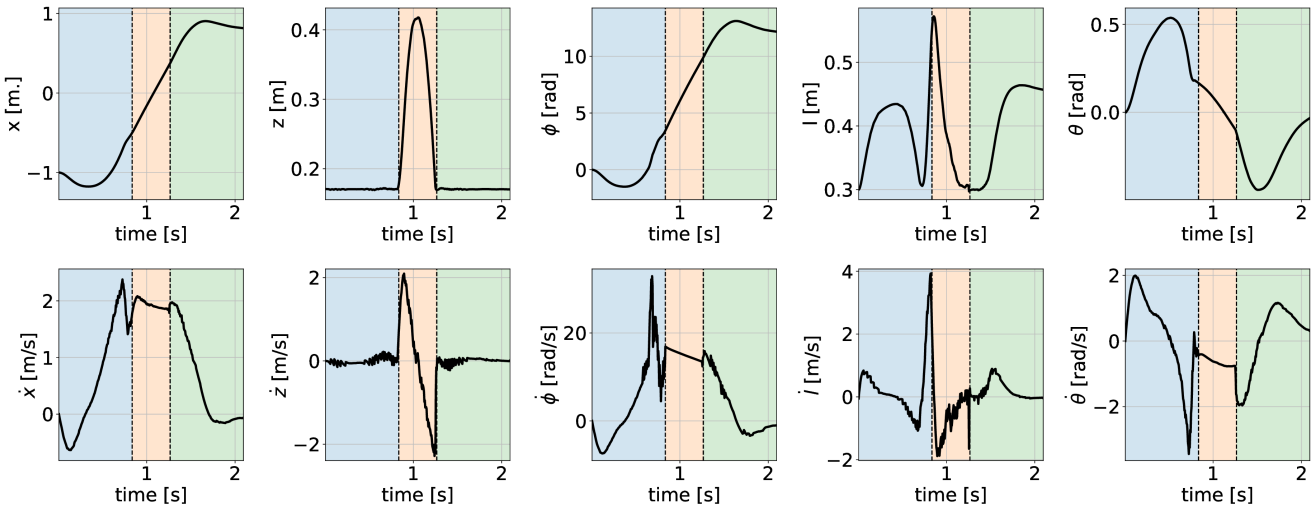


Fig. 8: State evolution for the jumping motion in Figure 7. Phases are—"pre-takeoff" in blue, "flight" in orange, and "post-touchdown"—in green.

where l^- and l^+ are the limits of the prismatic joint, R_w is the wheel radius, and z^+ limits the jumping height. In order to set up the length of the trajectory , we ran time optimization within bounds $T^- = 0.3, T^+ = 5$ and knot points $N_{\textit{pre-tf}} = 40, N_{flight} = 20, N_{\textit{post-td}} = 40$. This ensures that the optimizer chooses the appropriate flight duration that is physically feasible within the set time limits. In this sense, $T = 2.3$s was the optimal time and MPC horizon.

The resulting motion is shown in Figure 7 and its state and control history is plotted in Figure 8. The optimizer first "contracts" the body of the robot by retracting the prismatic joint before takeoff and extends it just before landing to absorb impact. We can see the preparatory movement in the optimized motion in Figure 7 for x, ϕ and θ.

V. Robustness

In the following experiments, we evaluate the robustness of the MPC approach as compared to a Proportional-Derivative (PD) controller with feed-forward torque.

The PD controller has the following control equation:

$$\tau = \tau_{\text{des}} - K_p^\theta e(\theta) - K_d^\theta e(\dot{\theta}) - K_p^x e(x) - K_d^x e(\dot{x}) \quad (10)$$

$$f = f_{\text{des}} + K_p^f e(f) + K_d^f e(\dot{f}) \quad (11)$$

where $e(\cdot)$ is the error between desired and measured state variables, $K_p^\theta = 15$, $K_d^\theta = 2$, $K_p^f = 1,000$, $K_d^f = 100$, $K_p^x = 1.25$, $K_d^x = 2$ are the PD gains we tuned by hand. We add a term dependent on x in the first equation so that position error is taken into account. Note, the larger PD gains for the prismatic joint—this is due to the scale difference in the length and force needed. We tuned the PD controller so that it achieves similar performance when driving on flat terrain as the MPC controller.

A. Robustness to Sensor Noise

This experiment has the same setup as subsection IV-B. We compare the effects of sensor noise on both the MPC and the PD controllers. We added Gaussian noise to sensor readings at each timestep of the simulation:

$$\mathbf{x}'_{\text{obs}} = \mathbf{x}_{\text{obs}} + \mathcal{N}(\mathbf{0}, \sigma\mathbb{I}) \quad (12)$$

where σ is the noise standard deviation, which was varied in the $0 - 0.4$ range and $\mathbb{I}$ is the identity matrix. We computed the L2-distance from the observed state $\mathbf{x}_{\text{obs}}$ to the goal state $\mathbf{x}^*$ at the end of the simulation. For this experiment we ran 20 trials for each value of σ, computing the means as well as the standard deviations of the L2-distance. The results are in Figure 9.

We notice similar performance for the PD controller at lower noise strengths. However, at $\sigma \geq 0.2$ the PD controller has a significant drop in performance. Additionally, as the noise strength increases, the variability in performance for the PD controller becomes larger. In contrast, the MPC controller shows better mean performance and lower performance variability across σ values.

B. Rough Terrain Locomotion

For this experiment we consider the task of driving forward on two wheels, as outlined in subsection IV-B. For the MPC controller we set a target velocity $\dot{x}^* = 1\,\mathrm{m\,s^{-1}}$ and horizon $T = 1\mathrm{s}$. The PD controller has no feed-forward torque reference; it has the same desired velocity target as well as a balance target $\theta^* = 0$. Neither have knowledge of the terrain and the overall duration of each experiment was $T = 5\mathrm{s}$.

We ran 20 experiments for varied terrain heights in the range $0\,\mathrm{m}$ to $0.4\,\mathrm{m}$. We generated random terrain using Perlin noise for every experiment while keeping the same random number generator seed between the PD and MPC. Example generated terrain for heights $0.1\,\mathrm{m}$ and $0.4\,\mathrm{m}$ are shown in Figure 11a and Figure 11b, respectively. We computed the mean absolute angle velocity $\dot{\theta}$ for the entire trajectory for each of the terrain heights and its standard deviation. We also show the episode duration as well as one standard deviation. The episode is terminated when the angle of the robot is more than 85 degrees. The results are in Figure 10a and Figure 10b, respectively.

For flat terrain and for terrain height up to $0.1\,\mathrm{m}$, MPC and PD show similar results. As the terrain height increases, however, MPC is better at stabilizing the robot with much less variability in performance.

VI. Discussion and Future Work

The results demonstrated in this paper show that the VL-WIP model, regardless of its simplicity, is sufficient for generating complex dynamic motions. We consider the combined motion planning pipeline developed here as a first step towards more dynamic locomotion of hybrid wheel-legged systems. Future work will apply several connected VL-WIP models, one for each limb, to wheel-legged bipeds and quadrupeds, using the standard motion planning and control pipeline outlined here. This approach is similar to using a SLIP model for each limb of a biped [22]–[24]. Compared to single rigid-body models, like the one in [25], the stacked VL-WIP model will model leg mass and inertia properties, allowing for more complex in-air maneuvers as well as not requiring inertia compensation.

Furthermore, for all the motions the planner exploited the dynamics of the system without being guided to do so explicitly. Most notably for jumping it discovered the preparatory motion from optimization alone—including contracting and extending the robot before take off, absorbing the impact upon landing, and extending the prismatic joint for easier balancing.

Currently for swing up and jumping we have predefined the switching times. In the future these will be automatically discovered, for instance via phase-based parametrization [25], [26].

When transferring to a real system, we note that work needs to be done to increase the speed of the computation, for instance by warm-starting the solver using trajectory libraries [27] or using policy learning. Our MPC controller ran at 109 ± 64 Hz for balancing, 18 ± 37 Hz for jumping, and 36 ± 13 Hz for swing-up on a Intel Core i9-9980HK at $2.4\,\mathrm{GHz}$ with $16\,\mathrm{GB}$ of DDR4 RAM at $2667\,\mathrm{MHz}$. We have indeed noticed that the solver did not take an equal number of iterations for all initial conditions and it sometimes did not converge. For this reason the speed was significantly lower on the more complex three-phase optimization for the jumping task. The convergence of the motion planner and the impact of warm starts is outside of the scope of this paper but it presents an exciting direction for future research.

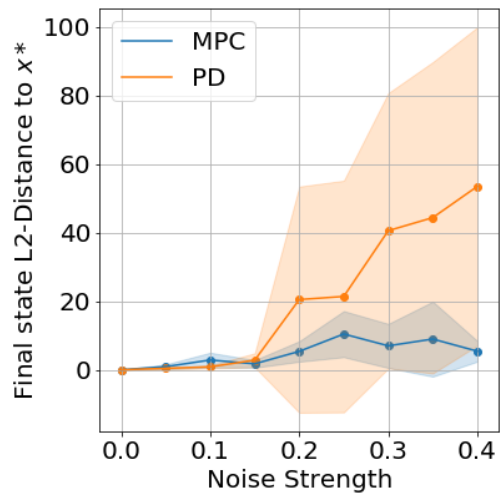


Fig. 9: Driving with sensor noise. L2-distance to $\mathbf{x}^*$ and one standard deviation.

Acknowledgements

This research is supported by the EPSRC Centre for Doctoral Training in Robotics and Autonomous Systems (EP/L016834/1), the EPSRC UK RAI Hub in Future AI and Robotics for Space (FAIR-SPACE, ID:EP/R026092/1) and the EU H2020 project Memory of Motion (MEMMO, ID: 780684). We would like to thank Henrique Ferollho and Matthew Timmons-Brown for their valuable help and feedback,

References

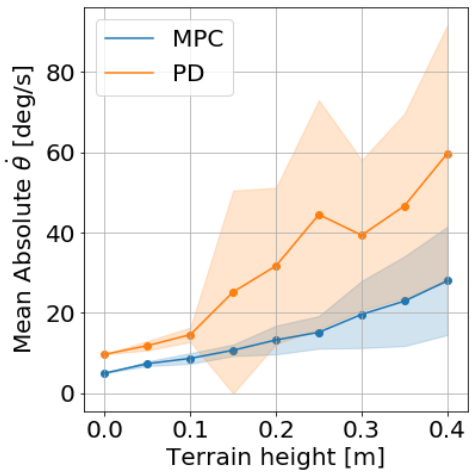
[1] V. Klemm, A. Morra, C. Salzmann, F. Tschopp, K. Bodie, L. Gulich, N. Kung, D. Mannhart, C. Pfister, M. Vierneisel, F. Weber, R. Deuber, and R. Siegwart, "Ascento: A Two-Wheeled Jumping Robot," in *2019 International Conference on Robotics and Automation (ICRA)*. Montreal, QC, Canada: IEEE, May 2019, pp. 7515–7521.

[2] M. Bjelonic, P. K. Sankar, C. D. Bellicoso, H. Vallery, and M. Hutter, "Rolling in the Deep – Hybrid Locomotion for Wheeled-Legged Robots using Online Trajectory Optimization," *arXiv:1909.07193 [cs, eess]*, Jan. 2020.

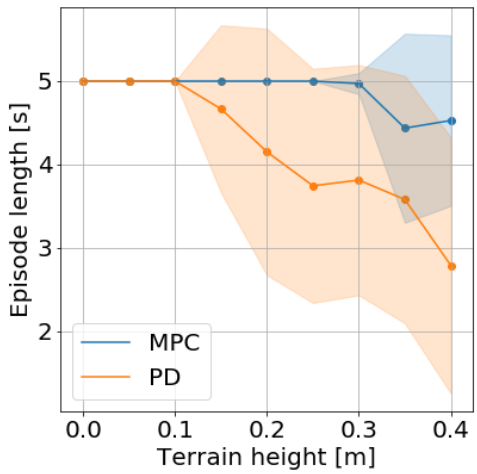
[3] M. Bjelonic, C. D. Bellicoso, Y. de Viragh, D. Sako, F. D. Tresoldi, F. Jenelten, and M. Hutter, "Keep Rollin' - Whole-Body Motion Control and Planning for Wheeled Quadrupedal Robots," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2116–2123, Apr. 2019.

[4] M. Geilinger, R. Poranne, R. Desai, B. Thomaszewski, and S. Coros, "Skaterbots: optimization-based design and motion synthesis for robotic creatures with legs and wheels," *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–12, July 2018.

[5] M. Giftthaler, F. Farshidian, T. Sandy, L. Stadelmann, and J. Buchli, "Efficient kinematic planning for mobile manipulators with non-holonomic constraints using optimal control," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 3411–3417.

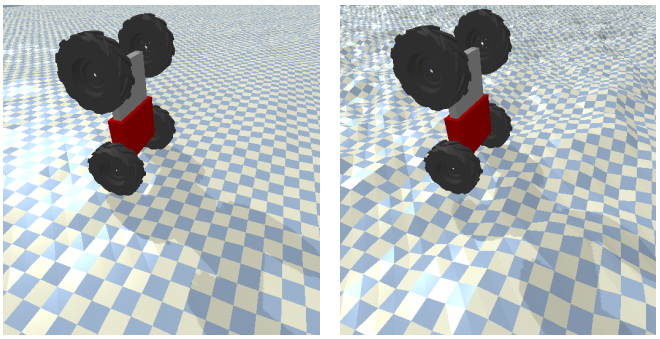


(a) Mean absolute angle velocity $\dot{\theta}$ and one standard deviation.



(b) Episode duration and one standard deviation.

Fig. 10: Rough Terrain Locomotion Results



(a) Rough terrain—height 0.1m. (b) Rough terrain—height 0.4m.

Fig. 11: Examples of rough terrain used.

[6] P. R. Giordano, M. Fuchs, A. Albu-Schaffer, and G. Hirzinger, "On the kinematic modeling and control of a mobile platform equipped with steering wheels and movable legs," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 4080–4087.

[7] K. Nagano and Y. Fujimoto, "The stable wheeled locomotion in low speed region for a wheel-legged mobile robot," in *2015 IEEE International Conference on Mechatronics (ICM)*, Mar. 2015, pp. 404–409.

[8] R. Tedrake, "Underactuated Robotics," Feb. 2018. [Online]. Available: http://underactuated.mit.edu/

[9] D. E. Orin, A. Goswami, and S.-H. Lee, "Centroidal dynamics of a humanoid robot," *Autonomous Robots*, vol. 35, no. 2, pp. 161–176, Oct. 2013.

[10] Y. de Viragh, M. Bjelonic, C. D. Bellicoso, F. Jenelten, and M. Hutter, "Trajectory Optimization for Wheeled-Legged Quadrupedal Robots Using Linearized ZMP Constraints," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1633–1640, Apr. 2019, conference Name: IEEE Robotics and Automation Letters.

[11] A. Saunders, C. E. Thorne, and A. A. Rizzi, "Environmentally sealed combustion powered linear actuator," US Patent US9 238 967B2, 2012.

[12] T. Tanaka and S. Hirose, "Development of leg-wheel hybrid quadruped "AirHopper" design of powerful light-weight leg with wheel," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept. 2008, pp. 3890–3895.

[13] I. Poulakakis and J. W. Grizzle, "The Spring Loaded Inverted Pendulum as the Hybrid Zero Dynamics of an Asymmetric Hopper," *IEEE Transactions on Automatic Control*, vol. 54, no. 8, pp. 1779–1793, Aug. 2009.

[14] P. M. Wensing and D. E. Orin, "Development of high-span running long jumps for humanoids," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 222–227.

[15] R. P. M. Chan, K. A. Stol, and C. R. Halkyard, "Review of modelling and control of two-wheeled robots," *Annual Reviews in Control*, vol. 37, no. 1, pp. 89–103, Apr. 2013.

[16] Y. Ding, J. Gafford, and M. Kunio, "Modeling, Simulation and Fabrication of a Balancing Robot," p. 22, 2012.

[17] M. Kelly, "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, Jan. 2017.

[18] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing Madison, WI, 2017, vol. 2.

[19] J. Andersson, J. Åkesson, and M. Diehl, "CasADi: A Symbolic Package for Automatic Differentiation andOptimal Control," in *Recent Advances in Algorithmic Differentiation*, ser. Lecture Notes in Computational Science and Engineering, S. Forth, P. Hovland, E. Phipps, J. Utke, and A. Walther, Eds. Berlin, Heidelberg: Springer, 2012, pp. 297–307.

[20] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Knitro: An Integrated Package for Nonlinear Optimization," in *Large-Scale Nonlinear Optimization*, P. Pardalos, G. Di Pillo, and M. Roma, Eds. Boston, MA: Springer US, 2006, vol. 83, pp. 35–59.

[21] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," *GitHub repository*, 2016.

[22] P. M. Wensing and D. E. Orin, "High-speed humanoid running through control with a 3D-SLIP model," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 5134–5140.

[23] ——, "3D-SLIP steering for high-speed humanoid turns," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept. 2014, pp. 4008–4013.

[24] Y. Liu, P. M. Wensing, D. E. Orin, and Y. F. Zheng, "Dynamic walking in a humanoid robot based on a 3D Actuated Dual-SLIP model," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5710–5717.

[25] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and Trajectory Optimization for Legged Systems Through Phase-Based End-Effector Parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, July 2018.

[26] T. Stouraitis, I. Chatzinikolaidis, M. Gienger, and S. Vijayakumar, "Dyadic collaborative manipulation through hybrid trajectory optimization." in *CoRL*, 2018, pp. 869–878.

[27] W. Merkt, V. Ivan, and S. Vijayakumar, "Leveraging Precomputation with Problem Encoding for Warm-Starting Trajectory Optimization in Complex Environments," in *IEEE/RSJ IROS*, Oct 2018, pp. 5877–5884.

[28] D. Morin, *Introduction to classical mechanics: with problems and solutions*. Cambridge University Press, 2008.

APPENDIX

We use the Lagrangian method to derive the dynamics of the system [28, Chapter 6]. First, we define the position x_b, z_b of the mass m_b and its velocity $\dot{x}_b, \dot{z}_b$:

$$\begin{aligned} x_b &= x + l\ \sin(\theta) \\ z_b &= z + l\ \cos(\theta) \\ \dot{x}_b &= \dot{x} + \dot{l}\ \sin(\theta) + l\ cos(\theta)\ \dot{\theta} \\ \dot{z}_b &= \dot{z} + \dot{l}\ \cos(\theta) - l\ sin(\theta)\ \dot{\theta} \end{aligned} \tag{13}$$

Lagrange's method states that for a system with total kinetic energy T and potential energy U:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}_i} - \frac{\partial \mathcal{L}}{\partial \mathbf{q}_i} = \mathbf{f}_{ext}, \tag{14}$$

where $\mathcal{L} = T - U$ is the system's Lagrangian and $\mathbf{f}_{ext}$ are external forces applied to the system. We now need to compute the system's kinetic and potential energy.

In general, every link will have a rotational and translational kinetic energy component. For the wheel we include a rotational kinetic energy term $I_w\dot{\phi}^2$ where $I_w = m_w R_w^2$ is the moment of inertia of the wheel. Since the point mass has a zero moment of inertia, it only has a translational kinetic energy $m_b\mathbf{v}_b^T\mathbf{v}_b$, where $\mathbf{v}_b$ is the velocity of the point mass.

The only potential energy component is due to gravity g acting on the wheel and the point mass. This leads to:

$$T = \frac{1}{2}(I_w\ \dot{\phi}^2 + m_w\dot{x}^2 + m_w\dot{z}^2 + m_b\dot{x}_b^2 + m_b\dot{z}_b^2) \tag{15}$$

$$U = m_w g z + m_b g z_b \tag{16}$$

$$\mathcal{L} = T - U \tag{17}$$

Inputting Equations (15), (16), and (17) into Equation (14) leads to a system of five equations, the solution of which is the equations of motion (2)-(4) in section II.