Proceedings of the ASME 2019
International Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
IDETC/CIE2019
August 18-21, 2019, Anaheim, CA, USA

DETC2019-98039

ONLINE TRAJECTORY OPTIMIZATION FOR LEGGED ROBOTICS INCORPORATING VISION FOR DYNAMICALLY EFFICIENT AND SAFE FOOTSTEP LOCATIONS

Joshua R. Hooks
Robotics and Mechanisms Laboratory
Department of Mechanical and Aeronautical Engineering
University of California Los Angeles
Los Angeles, California 90095
Email: hooksj@ucla.edu

Min Sung Ahn[*]
Dennis Hong
Robotics and Mechanisms Laboratory
Department of Mechanical and Aeronautical Engineering
University of California Los Angeles
Los Angeles, California 90095
Email: aminsung@ucla.edu

ABSTRACT

This paper presents a trajectory optimization algorithm for legged robotics that uses a novel cost function incorporating point cloud data to simultaneously optimize for footstep locations and center of mass trajectories. This novel formulation transforms the inherently discrete problem of selecting footstep locations into a continuous cost. The algorithm seamlessly balances the desire to choose footstep locations that enhance the dynamic performance of the robot while still choosing locations that are viable and safe. We demonstrate the success of this algorithm by navigating the ALPHRED V2 robotic system over unknown terrain in a simulation environment.

INTRODUCTION

Legged locomotion for robotic systems is an inherently underactuated problem requiring the complex planning of discrete foot placements and center of mass (CoM) trajectories in order to successfully navigate the world. Ground reaction forces at the points of contact and gravity are the only forces acting on a robot, making control of the ground reaction forces vital to achieve a successful motion plan. For this reason, foot placement is a critical component of developing a successful walking trajectory for a legged robot. However, due to the non-linear relationship between foot placement and robot dynamics many of the previous works either solve the problem offline or decouples the problem by separately planning footstep locations before solving for CoM trajectories.

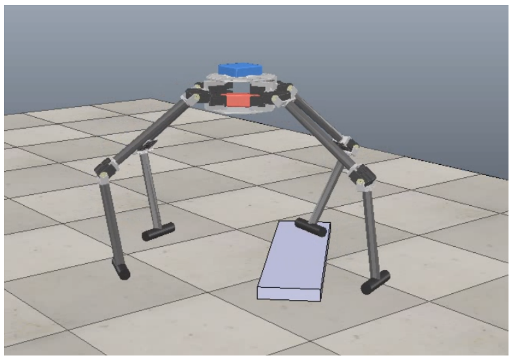

FIGURE 1. SIMULATION OF ALPHRED V2 TRAVERSING OVER AN OBSTACLE

There have been several successful frameworks that perform full-body trajectory optimization (TO) to execute complex tasks [1–3]. However, these algorithms are computationally complex resulting in the need for offline planning and full knowledge of the terrain a-priori making them impractical to use in the real-world. In [4] and [5] the Cheetah robot was able to perform extremely dynamic bounding gaits through the use of optimized

[*]Address all correspondence to this author.

force profiles on the ground. However, these algorithms were tailored for bounding gaits on flat terrain and do not extend to the general case for different gaits or uneven terrain.

Schaal et al. was one of the first to develop a method for trajectory planning over uneven terrain. In [6] they developed an impressive machine learning (ML) algorithm to build a robust footstep planner for a variety of terrains. The developed ML algorithm used a discritized height map to optimize for footstep locations based on the safety of the foothold position and the robot's kinematics. The footstep plan was then used by a Zero Moment Point (ZMP) motion planner to create a safe trajectory. In [7] Deits et al. developed a mixed-integer quadratically-constrained quadratic program (MIQCQP) to solve for an optimal footstep plan based on visual data. From the vision data Deits created a discrete set of viable planes where the robot could place its foot and used the MIQCQP to select the most optimal set of footsteps. In [8] Kuindersma et al. developed a stabilizing QP controller for dynamic walking that would use the MIQCQP as the input to the stabilizing controller. However, once again in this case the footstep planner only considered the kinematics of the robot and not the dynamics. ETH demonstrated impressive results on the ANYmal platform in [9] by simultaneously solving two different optimization problems, one for the CoM and one for the footstep locations. In this case the footstep planner was able to optimize the footsteps based on past dynamics of the robot, however the current and future CoM motion was not included in the optimization of the feet position. These methods all show impressive results but none of them address the inherent coupling between the footstep locations and the dynamics of the robot.

Winkler et al. developed non-linear programs (NLP) to simultaneously optimize for both CoM trajectories and foot placements. In [10] Winkler proved that his novel method of representing the ZMP constraint embedded capture point (CP) [11] dynamics into the optimization of the footstep locations for a position controlled robot. In [12] Winkler developed an NLP to solve for trajectories and footsteps using floating base dynamics for a force controlled robot. Similar to Winkler's floating body dynamics NLP, Bledt [13] developed an NLP that added useful heuristics to the cost function to significantly improve the time required to solve the optimization problem. Both Winkler's and Bledt's implementations optimize feet locations based on the dynamics of the robot, but do not consider the viability or safety of those locations. In addition, these algorithms require a dense height map of the entire terrain a-priori making them impractical for use on uneven terrain.

This paper presents a method for combining the previous works into a single NLP that produces CoM trajectories and foothold positions. A vision algorithm scores potential footstep locations based off of a multitude of criteria which then is used by an NLP to optimize for foothold positions based on safety of the location from the vision data, kinematics of the robot, and dynamics of the robot. The contributions made by this paper are the following:

- Development of a vision algorithm that finds the closest viable footstep location relative to a nominal footstep location. This algorithm provides a cost of that footstep location relaying how safe that position is.
- Development of a formulation that takes advantage of previously developed algorithms to solve the problems of footstep planning and trajectory optimization simultaneously. This formulation avoids the use of mixed integer optimization or multiple optimization problems that might not be able to utilize the dynamics of the robotic system.
- Development of an architecture that allows for continuous solving and stitching of trajectories while perceiving and adapting to the changing environment. This structure guarantees for safe motion of the robot even in the case where the optimization problem fails to find a solution.

Taken together, the contributions we offer enabled the development of a framework that utilizes the relationship between feet placement and robot dynamics while using safe and viable footstep locations. The presented formulation was tested in simulation on a quadrupedal robot, however it should be noted that the algorithms could be generalized to robotic systems with any number of legs.

1 SYSTEM OVERVIEW

An overview of the system framework for teleoperation on the ALPHRED V2 platform is shown in Fig. 2. User inputs (velocity magnitude, heading, and gait type) are passed to three motion planning threads, the first being the footstep planner. Based on user inputs the footstep planner generates footstep times and locations that will result in the desired velocity and direction of the robot. The footstep planner is further discussed in Section 2. The nominal footsteps are passed to the vision algorithm which potentially modifies these footsteps to the closest, viable, and safe locations. Each of these locations is given a score based on how safe these locations are, where the location is found by finding a flat viable plane and the safety is determined by the flatness, friction, and size of the sampled region. The analysis of the vision algorithm is discussed further in Section 3. These modified footsteps and scores are then passed to the TO algorithm. The TO algorithm will produce the final footstep locations and an optimal CoM trajectory for the robot to execute, discussed in Section 4. The optimized trajectory and footstep plan is passed to a Finite State Machine (FSM) which uses position control to execute the given trajectory. The FSM runs at 200 Hz and runs in parallel to the footstep planner, vision footstep adjuster, and trajectory optimization. While the FSM is executing the given trajectory the other three threads are simultaneously working on the next trajectory based on the most current user input. The three

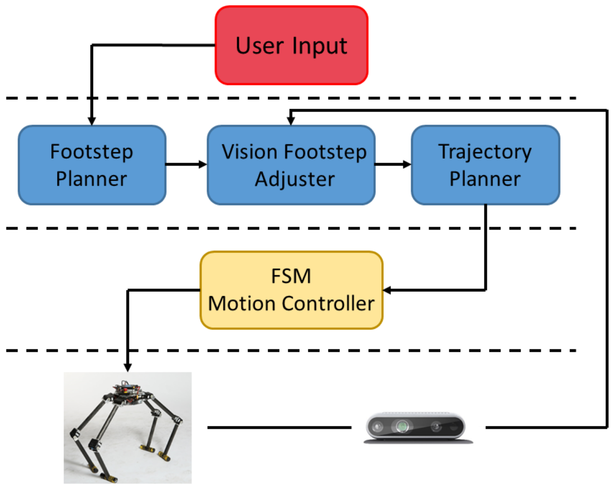

FIGURE 2. HIGH-LEVEL SYSTEM ARCHITECTURE FLOW CHART

motion planning threads run sequentially due to their reliance on the previous threads outcome.

2 FOOTSTEP PLANNER

The footstep planner determines the times and locations for the next 8 steps based on the previous trajectories footstep plan and the user inputs: velocity, direction, and gait type. The footstep planner returns a list of times and positions for each leg, which is passed to the vision footstep adjuster thread.

$$
\begin{aligned}
T_i &= [t_1, t_2, t_3, t_4] \\
F_i &= [f_1, f_2, f_3]
\end{aligned}
\tag{1}
$$

Where $i \in \{1,2,3,4\}$ is the i-th leg, $t_n \in \mathbb{R}$ is a time, and $f_m \in \mathbb{R}^3$ is a 3 dimensional position vector. The entry f_1 for each leg is the current position vector of that leg. The final two position vectors are the new footstep locations that will produce the desired direction and speed of the robot. The time vector consists of a lift off time and touchdown time for each of the new footsteps respectively (no times are need for the first location). This formulation makes it quite simple to change the gait of the robot by simply changing the time vectors for each leg. An indicator function can be derived from the time vector that indicates whether foot i is on the ground or not. Where $C_i(t) = 0$ when the i-th foot is off of the ground.

$$
C_i(t) = \begin{cases} 0 & t_1 < t < t_2 \quad \text{or} \quad t_3 < t < t_4 \\ 1 & otherwise \end{cases}
\tag{2}
$$

The location of the footsteps are determined using Raibert heuristics [14] similar to the method used in [15] and [16].

$$
f_i = f_{nom} + \dot{r}\frac{\Delta t}{2}
\tag{3}
$$

Where, f_{nom} is the nominal position of the foot when the robot is initialized, $\dot{r}$ is the CoM's velocity, and Δt is the contact time of the foot. In this case $\Delta t = t_3 - t_2$.

3 VISION FOOTSTEP ADJUSTER

To achieve online trajectory planning based on vision data where a potential adjustment from the nominal footstep position as well as a cost for the position needs to be passed back within a certain deadline, it was not possible to conduct a brute force search through all of the available vision data. Thus, it was necessary to conduct a series of pre- and post-processing strategies in such a way that less data could be used without loss of performance.

3.1 Pre-processing

Due to strict time constraints imposed by online planning, the order and the types of operations applied to the data were critical. We primarily tried to aggressively reduce the amount of data before conducting any computationally heavy tasks.

3.1.1 Reduced Vision Data All operations are done purely on point cloud data, as opposed to including RGB and depth information as demonstrated in [17]. While losing RGB-D information means a loss of more than half what some sensors can fully provide, considering the task at hand it was unnecessary to include such information.

In addition to removing the RGB data, the point cloud data was further reduced by a series of operations. First, the field of view (FOV) was narrowed to only fit the provided nominal footsteps. Second, there was no need to look past the furthest footsteps hence a passthrough filter was used to cut-off distant point clouds. Lastly, this reduced point cloud was downsampled by restructuring the cloud as a grid and taking the centroid as opposed to the center to better represent surfaces even in a downsampled state.

3.1.2 Planar Leg Removal Due to the camera being located under the body of the robot the vision sensor returns point cloud data about each of the front legs. While this information can be useful for collision detection, for the purposes of this paper, it was unnecessary. Our choice to efficiently remove this data from the cloud was to conduct a planar segmentation using

RANSAC [18]. With the correct parameters, the rod shaped legs would be labeled as outliers. This method retains all the points that are somewhat planar, and hence returns regions that could be realistically stepped on by the robot. However, it runs the risk of ignoring non-planar obstacles, potentially resulting in a point cloud with holes. While we do not purposefully place non-planar obstacles for this approach, these holes are expected to eventually be covered up as the map is built and updated as the robot moves around.

Our procedure results in a 95% reduction in data provided by the out of the box vision sensor. This reduced data is stored in an octree [19] for efficient searching through the cloud.

3.2 Short-Term Robot-Centric Map

A downside of relying on a single camera for footstep adjustments is the requirement to build an internal map because not all footholds are visible from a single frame. Because the camera is mounted on a legged robot that continuously interacts with the environment, the resultant data is noisy and incomplete. Thus, we focus on mapping just the vicinity based only on recent data. We use a variant of the Generalized Iterative Closest Point (GICP) [20], where the older the data is, the less prominent the point cloud data can be in affecting the scores of the footsteps. Furthermore, all the data is relative to the moving global inertial frame, a projection of the robot frame towards the ground. Also, data outside a pre-defined vicinity are removed from tracking.

This approach is effective primarily because of two reasons. First, it allows very little data to be stored in memory at all times. Without impacting ICP performance due to a lack of points, the map can still be stitched within a time limit. Second, by relying more on recent sensor data and forgetting old ones, the robot can still effectively use the built map without having to worry about biasing issues and potential errors from the transformation matrices continuously propagating in the map and having a negative impact.

3.3 Footstep Scoring

Receiving the nominal footstep positions and using the information from the map built in Sub-section 3.2, footstep scores are found for not only the nominal positions, but also potential positions within a predefined radius. The method for determining a cost for a given footstep is identical for the nominal position and the potential positions. After a complete assessment, the cost is clipped between 0 and 1. A cost of 0 has no impact on the proceeding optimization, allowing the problem to freely choose the footstep position, while that of 1 requires the opposite from the optimization problem.

Given a footstep or a point of interest (POI), we search for neighboring points within a pre-defined radius while keeping track of the Euclidean distance between the POI and the neighbor. 8 semi-random points are found, where 4 points are neighbors that are farthest away, while the other 4 points are randomly selected under a Gaussian distribution. In cases when rising/falling edges are among the neighbors, we create another point that is moved away from these edges by a selected magnitude, resulting in potentially 9 points. In the case that there are not enough neighbors to conduct a thorough cost analysis on that POI (i.e. a significant dearth of points) the terrain is assumed flat, resulting in a minimum cost of 0.

After locating all viable footsteps which is a union of the given footstep and its neighbors previously obtained, a cost assessment is done using a region of interest (ROI) identified using the neighbors. Costs for each POI and its neighbors are found by summing the individual scores of the slope, friction, and curvature of the ROI:

$$\alpha_{POI,i} = \sum w_m s_m \tag{4}$$

where w_m are the weights for the different scores obtained by the slope, friction, and curvature, while s_m is the score itself. $i = 1, \cdots, 8$ or 9 and $m \in \{\text{slope}, \text{friction}, \text{curvature}\}$.

To calculate the slope, friction, and curvature, we conduct a Principal Component Analysis (PCA) on the region. For slope, a normal vector is found by solving for the eigenvectors of the covariance matrix of the region. The normal vector's offset to the global Z axis (perpendicular to the ground) is compared, and assuming the offset is less than a pre-defined threshold, the offset in radians is returned. This offset value is used as s_m, where $m = \text{slope}$, resulting in a plane perpendicular to the Z axis allowing the most freedom to the proceeding optimization. It is believed that a slope closer to zero will provide a "safer" foothold position for two reasons 1) small inclines decrease the chances of slipping because the large ground reaction forces to counter act gravity are all normal to the plane 2) higher inclines require greater precision of terrain height estimation and end effector positioning because with higher inclines the terrain heights change dramatically which can cause early or late ground collisions with the end effectors if the end effector is not positioned correctly.

For calculating friction of the ROI, we use the standard deviation of the ROI along its normal vector, and use that as s_m where $m = \text{friction}$. Such calculations are simple, and allow us to avoid theoretically completely flat terrain, which have the potential to be more slippery than those that are not completely flat. In our formulation we prioritize ROI's with larger calculated friction coefficients.

Lastly, the curvature of the ROI is estimated by the relationship:

$$\kappa = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \tag{5}$$

where $\lambda_0 \leq \lambda_1 \leq \lambda_2$ and λ are the eigenvalues of the covariance matrix. For the same reasons as the slope calculations regions with larger curvatures are categorized as less "safe" than areas with smaller curvatures. Identical to the slope and the friction, the curvature score is directly used as s_m and weighted by w_m accordingly for our purposes.

The full algorithm and the scoring mechanism is shown in Algorithm 1 and 2. Some of the trivial co-routines are not explained for conciseness of the paper. We believe that the three selected scoring criteria are sufficient to correctly categorize a region as "safe" or not.

Algorithm 1 Overall Vision Algorithm

```
 1: procedure VisionModule($p_{nom}$)
 2:     $P_{xyz} \leftarrow$ GetPointCloud()
 3:     $P_{xyz} \leftarrow$ PreprocessPointCloud($P_{xyz}$)
 4:     $P_{xyz} \leftarrow$ BuildShortTermMap($P_{xyz}$)
 5:     for i = 1, $\cdots$, 8 do
 6:         $(\alpha_{POI,i,0}, p_{ext,i}) \leftarrow$ ScorePOI($p_{nom}$)
 7:         for j = 1, $\cdots$, len($p_{ext,i}$) do
 8:             $\alpha_{POI,i,j} \leftarrow$ ScorePOI($p_{ext}$)
 9:     $m \leftarrow$ GetMinIndex($\alpha_{POI}$)
10:     $p_{xyz} \leftarrow (p_{nom} \bigcup p_{ext})(m)$
11:     $\alpha_{min} \leftarrow \alpha_{POI}(m)$
12:     return $(p_{xyz}, \alpha_{min})$
```

Algorithm 2 Footstep Scoring Algorithm

```
1: procedure ScorePOI($p_{xyz}$)
2:     $A_{ROI} \leftarrow$ SearchRadius($p_{xyz}$)
3:     $p_{ext} \leftarrow$ GetSemiRndNeighbors($A_{ROI}$)
4:     $(\Lambda, V, C) \leftarrow$ PCA($A_{ROI}$)
5:     $(v_{normal}, s_{slope}) \leftarrow$ GetSlope($A_{ROI}, V$)
6:     $s_{friction} \leftarrow$ GetFriction($A_{ROI}, v_{normal}$)
7:     $s_{curvature} \leftarrow$ GetCurvature($A_{ROI}, \Lambda$)
8:     return $(\sum s_{slope,friction,curvature}, p_{ext})$
```

4 TRAJECTORY OPTIMIZATION

The trajectory algorithm presented in this paper builds on the algorithms in [21] and [10]. The problem is discretized into a discrete number of optimization parameters that fully describe the continuous time trajectory. The set w_c describes the CoM trajectory, w_u describes the ZMP, and w_p describes the x and y location of each foot.

$$\begin{aligned} w_c &= [{}^0a_1, ..., {}^4a_1, ..., {}^0a_n, ..., {}^4a_n]^T \\ w_u &= [{}^1\lambda_1, ..., {}^4\lambda_1, ..., {}^1\lambda_n, ..., {}^4\lambda_n]^T \\ w_p &= [{}^1p_1, ..., {}^4p_1, ..., {}^1p_m, ..., {}^4p_m]^T \end{aligned} \tag{6}$$

Where n is the number of polynomials that describe the trajectory and m is the number of footsteps to be optimized. ${}^ja_i \in \mathbb{R}^2$ describes the i-th polynomial's j-th coefficient for both x and y, ${}^j\lambda_i \in \mathbb{R}$ describes the j-th foot's percentage of the total load at the time of the i-th polynomial, and ${}^jp_i \in \mathbb{R}^2$ describes the j-th foot's x and y position for the i-th step. Using this parameterization of the problem it becomes relatively simple to compute all required dynamics of the robot at any given time. Given,

$$A_i = [{}^0a_i, {}^1a_i, {}^2a_i, {}^3a_i, {}^4a_i,] \quad T = [t^0, t^1, t^2, t^3, t^4]^T \tag{7}$$

the CoM position, velocity, and acceleration can be computed by the following:

$$\begin{aligned} r(t) &= A_i T \\ \dot{r}(t) &= A_i \dot{T} \\ \ddot{r}(t) &= A_i \ddot{T} \end{aligned} \tag{8}$$

The ZMP can also be calculated with the following formulation.

$$u(t) = \sum_{f=1}^{4} {}^f\lambda_i p_{m,f} \tag{9}$$

Given that the duration of the polynomials and the step times are known it can be assumed in (7), (9) and for the remainder of this paper that the correct optimization parameters are chosen from the sets w_c, w_u, and w_p given a time t. Fig. 3 provides a broad overview of the structure of this optimization problem whereas the subsequent sections give a brief explanation of each component.

4.1 Cost Functions

4.1.1 Acceleration This cost minimizes the total acceleration of the CoM. Prioritizing this cost creates smooth natural motion while minimizing the effort required by the actuators.

$$\theta_a \|\ddot{r}(t)\|^2 \tag{10}$$

The gain θ_a is used to change the amount of effect this cost will have on the overall optimization.

$$
\begin{aligned}
\min_{w_c, w_u, w_p} \quad & \theta_a \|\ddot{r}(t)\|^2 && \text{(CM acceleration)} \\
& \theta_l \|\lambda(t) - \lambda^*(t)\|^2 && \text{(Optimal ZMP location)} \\
& \theta_p \|p(t) - p^*(t)\|^2 && \text{(Footstep locations)} \\
\text{s.t.} \quad & && \\
& r(0) = r_0 && \text{(initial state)} \\
& r(T) = r_f && \text{(final state)} \\
& r(t_k^+) = r(t_k^-) && \text{(Continuity)} \\
& \ddot{r}(t) = f(r(t), u(t), p(t)) && \text{(ZMP dynamics)} \\
& |\boldsymbol{v}_i - r(t) + p_i(t)| > r_{xy} && \text{(Kinematic model)} \\
& \sum \lambda_i(t) = 1 && \text{(Total Loading)} \\
& 0 < \lambda_i(t) < c_i && \text{(Single Leg Load)}
\end{aligned}
$$

FIGURE 3. AN OVERVIEW OF THE TRAJECTORY OPTIMIZATION ALGORITHM

4.1.2 Loading The optimal loading cost chooses conservative trajectories for the robot and was first introduced in [10]. This cost compares each foot's loading condition (${}^j\lambda_i$) to the foot's optimal loading condition ($\lambda^*(i,t)$). The optimal loading condition balances the load between all feet that are currently on the ground. This will create a center of pressure directly in the middle of the support polygon.

$$
\begin{aligned}
& \theta_l({}^j\lambda_i - \lambda^*(i,t))^2 \\
\text{where,} \quad & \lambda^*(i,t) = \frac{C_i(t)}{\sum_{i=1}^{4} C_i(t)}
\end{aligned}
\tag{11}
$$

The gain θ_l is used to change the amount of effect this cost will have on the overall optimization.

4.1.3 Foot Placement The foot placement cost tries to minimize the distance between the location of the optimized foot location ${}^jp_i$ and the desired foot location ${}^jp_i^*$ chosen by the vision algorithm. The gain α also comes from the vision algorithm and is bound between 0 and 1. Notice that when α is 0 this cost will have no effect and the foot placement will only be chosen based on the previously mentioned costs.

$$
\theta_p(e^{\alpha} - 1)\|{}^jp_i - {}^jp_i^*\|^2 \tag{12}
$$

The gain θ_p is used to change the amount of effect this cost will have on the overall optimization. Originally, a linear form of the cost was tested but produced undesirable results. If θ_p was set too low then this cost would be ignored regardless of α, however if θ_p was set to high this cost would dominate regardless of α. We found that it was extremely difficult to tune θ_p with a linear cost in α. Through trial an error a heuristic exponential cost in α was found to produce the desired results of balancing the importance of the costs through the vision score α.

4.2 Constraints

4.2.1 Continuity The following is an equality constraint that ensures continuity between polynomials.

$$
\begin{aligned}
A_i T - {}^0a_{i+1} = 0 \\
A_i \dot{T} - {}^1a_{i+1} = 0
\end{aligned}
\tag{13}
$$

4.2.2 ZMP dynamics The dynamic model used in this algorithm is the well known ZMP equation shown in (14). The values $\ddot{c}_z$ and c_z represent the Z acceleration and position of the robot respectively. These values are pre-computed based on the average z height of all the feet at a given time.

$$
\ddot{r}(t) = f(w_c, w_u, w_p, t) = \frac{g + \ddot{c}_z}{c_z}(r(t) - u(t)) \tag{14}
$$

The equality constraint (15) ensures that the optimized trajectory satisfies the ZMP dynamic model to a satisfactory tolerance. Simpson's rule bounds the error by $\propto (t_{k+1} - t_k)^4$ over the interval t_{k+1} to t_k.

$$
A\ddot{T} - f(w_c, w_u, w_p, t) = 0 \quad \forall t \in t_k, \frac{t_{k+1} - t_k}{2}, t_{k+1} \tag{15}
$$

4.2.3 Kinematic The inequality constraint described by (16) assures that the relationship between the CoM position and the i-th foot position is bound by a box with dimensions r_{xy}. The value r_{xy} must be set based on prior knowledge of the robot's kinematics.

$$
-r_{xy} < v_i - r(t) + p_i(t) < r_{xy} \tag{16}
$$

Where v_i is the vector from the robot CoM to the i-th foot in the robot's nominal pose.

4.2.4 Loading The inequality constraint described in (17) bounds all feet on the ground between 0 and 1 and all feet off of the ground are forced to be zero. This ensure that no leg in a swing phase carries any load. The equality constraint (18) makes certain that all of the leg's loading conditions add up to 1, meaning that all load bearing legs are supporting 100 percent of the weight of the robot.

$$
0 < {}^j\lambda_i < C_i(t) \tag{17}
$$

$$\sum^{j} \lambda_i = 1 \tag{18}$$

If both (17) and (18) are satisfied than the ZMP is defined to lie within the support polygon.

4.3 Stitching

Each user input creates a nominal straight line trajectory. The nominal trajectory is created simply by moving along the velocity vector given by the user input until time t_f (the final time of the optimization problem). There are two points of interest along this trajectory: the point at t_{s_1}, where t_{s_1} is the time right after all feet have finished their first step and the point at t_f. Each trajectory solves for the next 8 steps, two steps for each foot, with the desired footsteps being provided by the vision algorithm. The final CoM position is set to the location of the nominal trajectory at time t_f and the velocity is set to zero. Only the first step of each foot is allowed to be an optimization variable whereas the last step for each foot is set to the location provided by the vision algorithm. The final solution will create a trajectory that brings the robot to a complete stop in a nominal pose after taking two steps with each leg.

If a user input is provided another optimization problem will be created for another 8 steps. A new inertial coordinate system is created with it's origin corresponding to the location at t_{s_1} along the previous optimization problem's nominal trajectory, as shown in Fig. 4. The initial conditions for the new optimization problem are the CoM position, CoM velocity, and footstep locations at t_{s_1}. All of these values are transformed into this new coordinate system. The previous trajectory will only execute until t_{s_1} upon which the FSM will start executing the new trajectory. This pattern will occur until no more user inputs are detected and the robot will come to a complete stop. Using a local inertial frame rather than a global inertial frame helped reduce solve times by warm starting the optimization problem with the solution from the previous optimization problem.

This stitching method only uses half of the solution from each optimization problem. There are methods such as path regularization [22–24] that use the entire or nearly entire solution of the computed optimization problem, making these methods more efficient. However, these methods require that a solution to the optimization problem is found before the previous problem has completed execution with failure to do so resulting in a crash. Unfortunately NLP's do not provide bounds on solve times and for this reason we opted to use a less efficient method in order to guarantee safety of the robot. Every trajectory that is created during our algorithm is assured to have a safe and viable stopping point meaning that if a solution to the next optimization problem fails the robot will still be able to safely stop.

5 RESULTS

This section discusses the results from simulation of the ALPHRED V2 system in V-REP [25]. The entire code was written in Python 2.6 implemented on an Intel NUC Quad-Core i7-6770HQ with 8 GB of DDR4 RAM at 2400 MHz. Interior Point Optimizer (Ipopt) [26] was used as the non-linear solver for the TO thread with PyIpopt as the Python wrapper. The simulation camera mimics the Intel Realsense D435. In simulation we tested the algorithm with flat obstacles varying in different heights, sizes, and orientations which can be seen in the supplementary video. The discussion below explains in detail a single test of the robot going over a 0.04m tall obstacle.

In the simulation the robot trots 1.0m and then steps onto and over a 0.04m obstacle. The step length was set to 0.25m with a step time of 0.6 seconds giving the algorithm 1.2 seconds to solve before the next trajectory. During the duration of this task the algorithm stitched together eight trajectories with an average solve time of 400ms (100ms for the vision and 300ms for the TO) per trajectory. Fig. 5 - 7 are three snapshots of different moments during the simulation that best highlight the novelty of this algorithm. The figures show the nominal footsteps provided by the footstep planner (blue squares), the modified footsteps provided by the vision planner (green diamonds), and the optimized footsteps provided by the TO problem (red crosses). Fig. 5 is a footstep plan during the first 1.0m of travel. During this time the terrain is completely flat with no dangerous footstep locations, this is indicative of the fact that the vision algorithm did not modify the nominal footsteps and provided a low cost on

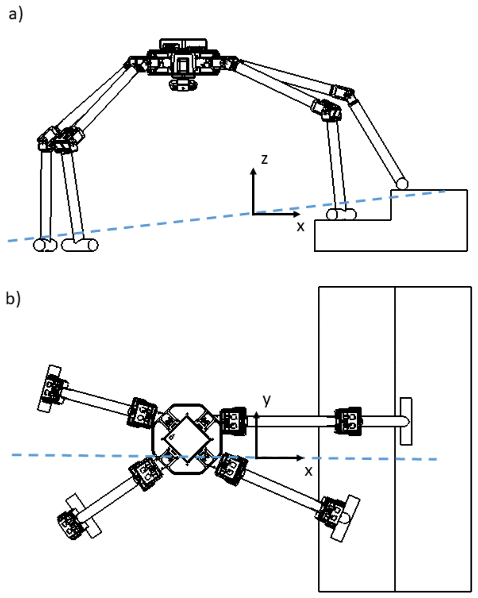

FIGURE 4. NOMINAL TRAJECTORY SHOWN BY BLUE DASHED LINE. A) SIDE VIEW, B) TOP VIEW

all footstep. This allowed the footsteps to be highly optimized shown from the optimized footsteps deviating dramatically from the modified footsteps.

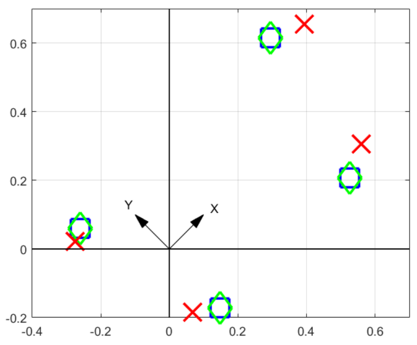

FIGURE 5. FOOTSTEP PLAN FOR FLAT TERRAIN SHOWING DEVIATION FROM THE MODIFIED FOOTSTEPS PROVIDED BY VISION AS OPTIMIZED DYNAMICS ARE PREFERRED OVER LANDING AT THE MODIFIED FOOTSTEPS

In Fig. 6 the robot is planning on stepping onto the obstacle (dashed purple box) with the front two legs. The back two legs are in safe locations and are thus treated as they were in Fig. 5. However, the front two legs must now be modified from the nominal positions in order to provide the robot with secure footing. Due to their precarious position both the front legs have a high cost forcing the optimization algorithm to choose the position selected by the vision algorithm. An interesting result from this is that the back feet locations are now changed to try to counter act the dynamics created from modifying the front feet. This is most clearly seen by the back left's optimized footstep. The change in the back left foot's position mirrors that of the constrained front foot's location. This is a very powerful result where the optimization algorithm naturally tries to account for the adverse dynamics created by traversing safely over the obstacle. Similar phenomenon are seen in Fig. 7 where now the back feet are modified due to the obstacle and the front feet are free to be optimized.

6 CONCLUSION

In this paper we introduced a vision algorithm and NLP that could continuously produce optimal CoM trajectories and foothold positions for a legged robot over varying terrains. By developing a vision algorithm that locates and scores potential foot positions we were able to encode the inherently discrete problem of footstep selection into a continuous time cost that could naturally be added to our NLP formulation. We believe the results of this paper show that this type of problem formulation has great potential in the area of legged robot trajectory optimization, however future work needs to be done to further investigate this methodology. We would like to further investigate the criteria determining whether a footstep location is "safe" or not by quantifying the effects of each criteria on the robot's ability to successfully track the desired trajectory. The gains in both the vision algorithm and NLP were hand tuned, in the future we would like to use a ML algorithm to perform the scoring for the vision algorithm similar to those found in [6] and [27]. Currently, the algorithm has only been tested in a simulated environment, and in the future we plan on testing the algorithm on the real-world robot.

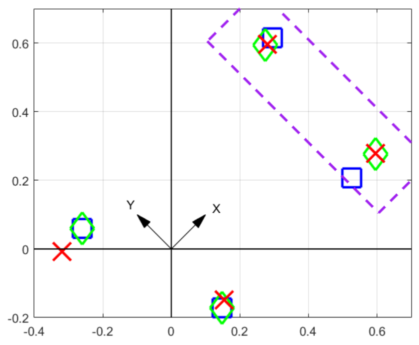

FIGURE 6. FOOTSTEP PLAN FOR STEPPING UP ONTO AN OBSTACLE WITH THE FRONT TWO LEGS, WHERE THE OPTIMIZED FOOTSTEPS ADHERE TO THE MODIFIED FOOTSTEPS PROVIDED BY THE VISION BECAUSE SAFETY IS PREFERRED OVER FOLLOWING A GIVEN FOOTSTEP TRAJECTORY

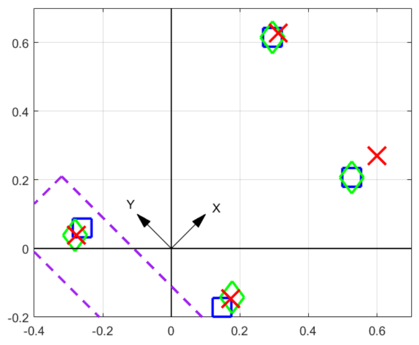

FIGURE 7. FOOTSTEP PLAN FOR STEPPING DOWN FROM AN OBSTACLE, WHERE THE FRONT TWO LEGS OPTIMIZED FOOTSTEPS DO NOT ADHERE TO THE MODIFIED FOOTSTEPS BECAUSE OF LOW RISK WHEREAS THE BACK LEGS ON THE OBSTACLE CONTINUE TO FOLLOW THE MODIFIED FOOTSTEPS TO ENSURE SAFETY

ACKNOWLEDGMENT

This research was supported by a grant (code R2016001) from Gyeonggi Technology Development Program Funded by Gyeonggi Province.

REFERENCES

[1] Posa, M., Cantu, C., and Tedrake, R., 2014. "A direct method for trajectory optimization of rigid bodies through contact". *The International Journal of Robotics Research,* **33**(1), pp. 69–81.

[2] Posa, M., Kuindersma, S., and Tedrake, R., 2016. "Optimization and stabilization of trajectories for constrained dynamical systems". In 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 1366–1373.

[3] Farshidian, F., Neunert, M., Winkler, A. W., Rey, G., and Buchli, J., 2017. "An efficient optimal planning and control framework for quadrupedal locomotion". In 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 93–100.

[4] Park, H.-W., Wensing, P. M., Kim, S., et al., 2015. "Online planning for autonomous running jumps over obstacles in high-speed quadrupeds".

[5] Park, H., Park, S., and Kim, S., 2015. "Variable-speed quadrupedal bounding using impulse planning: Untethered high-speed 3d running of mit cheetah 2". In 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 5163–5170.

[6] Kalakrishnan, M., Buchli, J., Pastor, P., Mistry, M., and Schaal, S., 2011. "Learning, planning, and control for quadruped locomotion over challenging terrain". *The International Journal of Robotics Research,* **30**(2), pp. 236–258.

[7] Deits, R., and Tedrake, R., 2014. "Footstep planning on uneven terrain with mixed-integer convex optimization". In Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on, IEEE, pp. 279–286.

[8] Kuindersma, S., Permenter, F., and Tedrake, R., 2014. "An efficiently solvable quadratic program for stabilizing dynamic locomotion". In Robotics and Automation (ICRA), 2014 IEEE International Conference on, IEEE, pp. 2589–2594.

[9] Bellicoso, C. D., Jenelten, F., Fankhauser, P., Gehring, C., Hwangbo, J., and Hutter, M., 2017. "Dynamic locomotion and whole-body control for quadrupedal robots". In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3359–3365.

[10] Winkler, A. W., Farshidian, F., Pardo, D., Neunert, M., and Buchli, J., 2017. "Fast trajectory optimization for legged robots using vertex-based zmp constraints". *IEEE Robotics and Automation Letters,* **2**, pp. 2201–2208.

[11] Pratt, J., Carff, J., Drakunov, S., and Goswami, A., 2006. "Capture point: A step toward humanoid push recovery". In 2006 6th IEEE-RAS International Conference on Humanoid Robots, pp. 200–207.

[12] Winkler, A. W., Bellicoso, C. D., Hutter, M., and Buchli, J., 2018. "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization". *IEEE Robotics and Automation Letters,* **3**(3), pp. 1560–1567.

[13] Bledt, G., Wensing, P. M., and Kim, S., 2017. "Policy-regularized model predictive control to stabilize diverse quadrupedal gaits for the mit cheetah". In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp. 4102–4109.

[14] Raibert, M. H., 1986. *Legged robots that balance*. MIT press.

[15] Di Carlo, J., Wensing, P. M., Katz, B., Bledt, G., and Kim, S., 2018. "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control". In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp. 1–9.

[16] Bledt, G., Powell, M. J., Katz, B., Di Carlo, J., Wensing, P. M., and Kim, S., 2018. "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot". In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, pp. 2245–2252.

[17] Ahn, M. S., Chae, H., and Hong, D. W., 2018. "Stable, autonomous, unknown terrain locomotion for quadrupeds based on visual feedback and mixed-integer convex optimization". In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (Accepted).

[18] Fischler, M. A., and Bolles, R. C., 1981. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". *Communications of the ACM,* **24**(6), pp. 381–395.

[19] Payeur, P., Hebert, P., Laurendeau, D., and Gosselin, C. M., 1997. "Probabilistic octree modeling of a 3d dynamic environment". In Proceedings of International Conference on Robotics and Automation, Vol. 2, pp. 1289–1296 vol.2.

[20] Segal, A., Haehnel, D., and Thrun, S., 2009. "Generalized-icp.". In Robotics: science and systems, Vol. 2, p. 435.

[21] Hooks, J., and Hong, D., 2018. "Implementation of a versatile 3d zmp trajectory optimization algorithm on a multi-modal legged robotic platform". In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems. Paper accepted for publication.

[22] Bellicoso, C. D., Jenelten, F., Gehring, C., and Hutter, M., 2018. "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots". *IEEE Robotics and Automation Letters,* **3**(3), July, pp. 2261–2268.

[23] Betts, J. T., 1998. "Survey of numerical methods for trajectory optimization". *Journal of Guidance, Control, and Dynamics,* **21**(2), Mar., pp. 193–207.

[24] Pardo, D., Möller, L., Neunert, M., Winkler, A. W., and Buchli, J., 2016. "Evaluating direct transcription and non-

linear optimization methods for robot motion planning". *IEEE Robotics and Automation Letters,* **1**(2), pp. 946–953.

[25] E. Rohmer, S. P. N. Singh, M. F., 2013. "V-rep: a versatile and scalable robot simulation framework". In Proc. of The International Conference on Intelligent Robots and Systems (IROS).

[26] Wächter, A., and Biegler, L., 2009. Ipopt-an interior point optimizer.

[27] Siravuru, A., Wang, A., Nguyen, Q., and Sreenath, K., 2017. "Deep visual perception for dynamic walking on discrete terrain". In 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids), IEEE, pp. 418–424.