A Walking Controller for Humanoid Robots using Virtual Force

by

VINAYAK JAGTAP

A Dissertation
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in
Robotics Engineering

September 2019

APPROVED:

Professor Michael Gennert, Thesis Advisor

Professor Gregory Fischer

Professor Taşkın Padır

Professor Mohammad Mahdi Agheli Hajiabadi

Abstract

Current state-of-the-art walking controllers for humanoid robots use simple models, such as Linear Inverted Pendulum Mode (LIPM), to approximate Center of Mass(CoM) dynamics of a robot. These models are then used to generate CoM trajectories that keep the robot balanced while walking. Such controllers need prior information of foot placements, which is generated by a walking pattern generator. While the robot is walking, any change in the goal position leads to aborting the existing foot placement plan and re-planning footsteps, followed by CoM trajectory generation. This thesis proposes a tightly coupled walking pattern generator and a reactive balancing controller to plan and execute one step at a time. Walking is an emergent behavior from such a controller which is achieved by applying a virtual force in the direction of the goal. This virtual force, along with external forces acting on the robot, is used to compute desired CoM acceleration and the footstep parameters for only the next step. Step location is selected based on the capture point, which is a point on the ground at which the robot should step to stay balanced. Because each footstep location is derived as needed based on the capture point, it is not necessary to compute a complete set of footsteps. Experiments show that this approach allows for simpler inputs, results in faster operation, and is inherently immune to external perturbing and other reaction forces from the environment. Experiments are performed on Boston Dynamic's Atlas robot and NASA's Valkyrie R5 robot in simulation, and on Atlas hardware.

Acknowledgements

The work presented in this dissertation is an outcome of about five years. Several people have directly or indirectly helped, contributed, or influenced the direction of this research.

I would like to thank:

- My parents, Vasant Jagtap and Shobha Jagtap, and my sisters, Ujwala Bavisker and Jayashree Bhandarkar, for their sacrifices and the constant effort put in to make me capable.

- My wife, Nirupama Talele, for the encouragement and motivation throughout this journey. This would not have been possible without her support.

- My Advisor, Professor Michael A. Gennert, for trusting and guiding me. For the financial and organizational support, and for providing me the opportunity (and freedom) to work at WPI Humanoid Robotics Lab (WHRL). I would also like to thank Professor Gregory Fischer, Professor Mohammad Mahdi Agheli Hajiabadi, and Professor Taşkın Padır for providing insights into different aspects of the research presented.

- NASA's Space Robotics Challenge team and WHRLers for the support in maintaining the robot and creating an environment that facilitated humanoid research.

- Jayam Patel, Rahul Krishnan, and Lening Li for long discussions about research and grad-life in general.

- Finally, I would like to acknowledge the Dean of Engineering, CS, ME, ECE departments, and the Robotics Engineering program for their resources and constant support.

Contents

List of Figures

Chapter 1

Introduction

One of the major motivations for robotics research is to pass on dangerous or unpleasant tasks from humans to robots. Given that these situations appear in environments that cannot always be traversed on wheels, there arises a need for legged locomotion. Several different types of legged robots exist and a majority of them are bio-inspired. Legged robots allow smaller footprint on ground at the cost of increased complexity in the controller design. Walking of these legged robots is achieved by planning and executing trajectories for each leg. In many cases, the same trajectory is phase shifted and executed on different legs. A special type of these legged robots that are inspired from humans are the biped humanoid robots. The world that we live in is built for humans. The infrastructures, machines, tools, vehicles, etc. are all built for use by humans. For robots to perform tasks in this environment, either they should be able to use existing tools or we must develop specialized tools for such robots. Redesigning every single tool or machine for use by robots is not practical, so exploring the first option seems more logical for a general purpose robot. One can argue that the control of inherently balanced robot can be less complex and the robot could be similarly able. However, to traverse any kind of terrain, climb ladders, climb stairs, open doors, etc. there is a need for active balancing for some or most part of the task. Humanoid robotics provide the starting point required for such kind of walking and balancing requirements. The earliest research on humanoid robots dates back to the 1970s when Wabot-1, a full-scale anthropomorphic robot, was developed[31]. Wabot-1 could communicate in

Japanese, measure distances to objects, grasp objects, and walk. In the 40 years following development of Wabot-1, several other humanoid robots were developed in different research labs with motivation to build personal assistants, entertainment robots, and understanding balancing/walking of humans. However, these robots significantly lacked good locomotion capabilities. PETMAN[41], developed in 2012, was the first biped robot that walked dynamically like human beings. In the same year, the Defense Advanced Research Projects Agency (DARPA) organized DARPA Robotics Challenge (DRC)[46] catapulting the humanoid robotics research for real world applications like disaster response in dangerous, degraded, and human-engineered environments. This competition exposed the challenges in humanoid balancing and walking to the world. One of the most viewed videos from this competition is a compilation of falling robots during the competition[1]. There has been a steady progress in research and development of humanoid walking and balancing controllers since the competition, leading to more advanced balancing and walking controllers for humanoid robots.

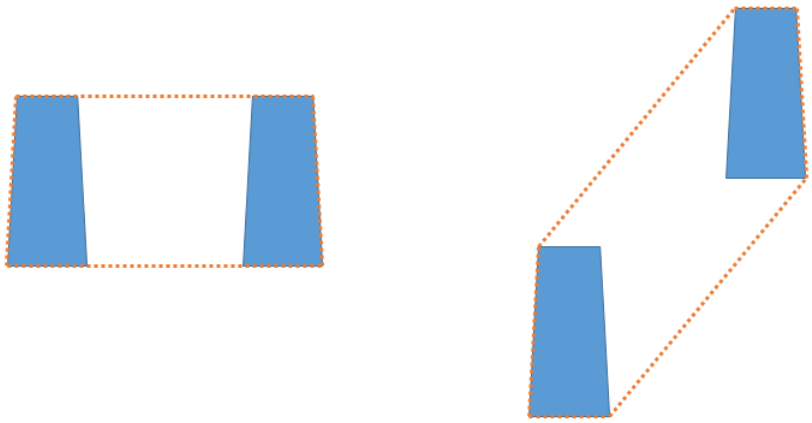


Figure 1.1: Projection of feet on ground when the robot is standing with feet next to each other (left) and during double support while walking (right). Orange border shows the support polygon

Most of the current walking controllers for humanoid robots use simple models like Linear Inverted Pendulum Mode (LIPM) to generate motion trajectories for the Center of Mass (CoM) of the robot. The controller minimizes error between desired and actual

[1]https://www.youtube.com/watch?v=g0TaYhjpOfo

values of certain ground reference points which results in minimizing error of CoM position or velocity while following these trajectories. As the robot has high degrees of freedom (DOF), there are infinite values for joint positions to get a specific desired position of CoM. The joint positions, velocities, and/or acceleration are selected using optimization techniques like quadratic solvers, optimal control, or reinforcement learning and sent to the robot. In this method, one controls either the position and velocity of the CoM or the centroidal momentum acting on the CoM or both. The robot is balanced as long as the projection of CoM is inside support polygon formed by joining all the contacts of the robot with the environment as shown in Figure 1.1 and the total momentum acting on centroid of the robot is zero. These controllers require footstep to be planned before generating the desired CoM trajectories. The footstep planners run at a lower frequency than the control loop and need re-planning in dynamic environments[5][34] [24]. The footsteps provided by the planners are validated and used for planning the robot trajectories[15] [2] [10].

In the approach mentioned above, the robot tracks desired footsteps with high accuracy at the cost of time required to plan these steps. The requirement of desired footstep comes from the system dynamics which helps in deciding current commands to the robot based on the future position/velocity of CoM[2]. However, if we can control the robot such that decisions based on future of CoM position and velocity can be relaxed, we can build reactive controllers for humanoid robots. These reactive controllers can be used to make a robot walk, but the operator would lose control on positioning the steps exactly at the desired location. It is acceptable to lose precision in footstep planning as humanoid robots have a very large taskspace as seen in Figure 1.2. The green cubes in the figure can be reached in any orientation by the 7 DOF arm whereas the red cubes represent the space where all orientations of hand are not possible. When we use 10 DOF kinematic chain by adding 3 DOF of pelvis along with the arm, the reachable area increases further. Hence the standing pose of the robot need not be precise for most tasks. On most terrains, the footstep positions of robot can be approximate as long as object to be manipulated is reachable and arm manipulation is performed with required precision.

[2]More details on this are provided in following chapters.

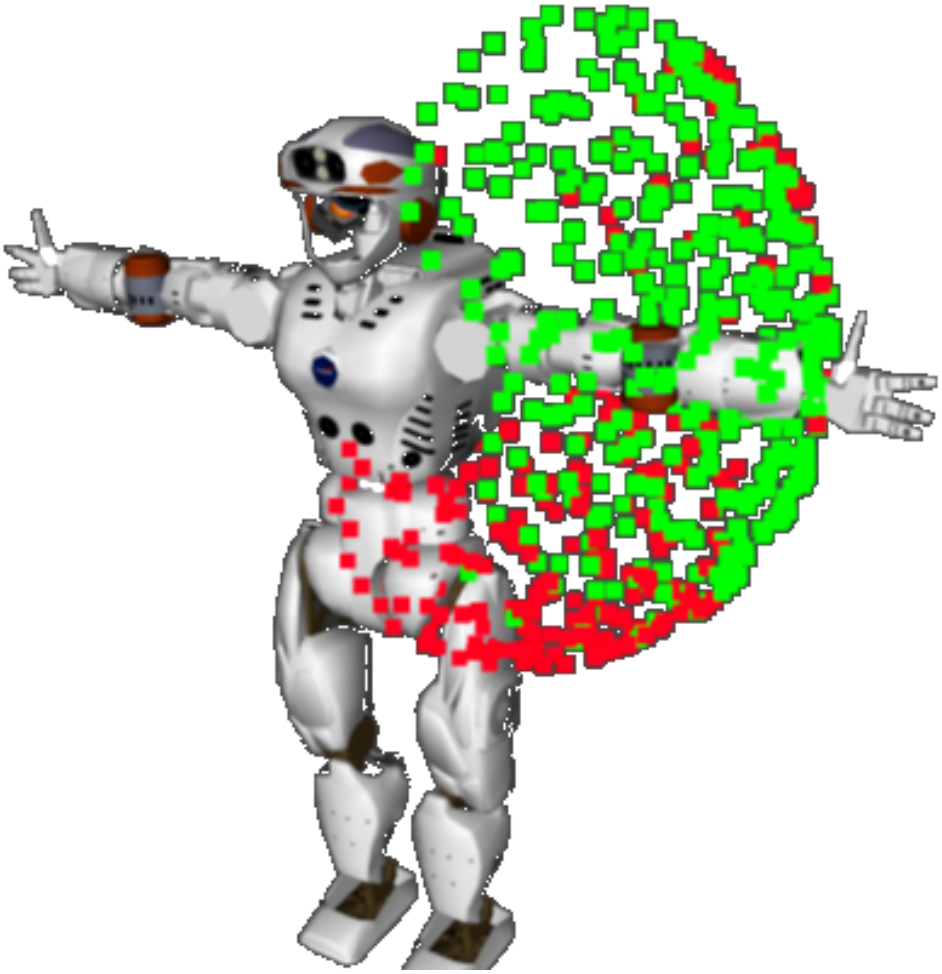


Figure 1.2: Taskspace of 7-DOF left arm of Valkyrie robot. Green cubes can be reached in any orientation and red cubes can be reached with limited orientations.

1.1 Problem Statement

In the current state, humanoid robots plan footsteps before walking. This approach is offline, where a planner uses knowledge of robot kinematics to plan footsteps from start to goal. Once planned, the low-level controller follows the steps. Minor online step adjustment is performed for a stable walk. However, the planned steps are inflexible.

This thesis proposes a novel method for humanoid robots to walk one step at a time using "Virtual Force". The virtual force is used for generating desired footstep parameters using the same dynamics that is used by the low-level controller. This allows us to move the

robot such that its CoM velocity would reach the required value for stepping in the desired direction. As the dynamics of the robot is used for generating the desired footstep, it ensures that the low-level controller would be able to execute the footstep trajectory. This provides a tight coupling between the walking pattern generator and the walking controllers as opposed to footstep planners where only kinematics is used for planning footsteps. In this entire process, the desired state is generated based on virtual force alone. The location of footstep cannot be precisely defined by the control input in this method, but the computation can be executed with real time constraints.

1.2 Related Work

For humanoid robots to walk stable, the resultant of external forces on the robot has to pass through the support polygon. Hence the traditional tracking control of multiplying gains with errors computed using desired and actual position, velocity, or acceleration of CoM to generate control input cannot be used. Those controllers do not allow adding constraints to position external forces inside the support polygon. To add on to the complexity, the CoM follows a non-linear trajectory and its dynamics is reset with every step the robot walks. Different control strategies exist depending on the type of actuation of the robot as explained in subsections below.

1.2.1 Passive Walkers

Passive (biped) walkers are mechanisms with open-loop control that produce walking behaviour by utilizing the forces on the system due to gravity or system state. Passive walkers rely on the mechanical design of the robot to generate a basin of attraction such that the robot stays stable. The dynamics of such mechanisms is explored in [37]. Coleman et.al. [7] proposed one such mechanism which used only gravity to walk down a slope. This toy-like mechanism is stable near a statically unstable configuration but does not depend on spinning parts. Collins et. al. [9] built a 3-dimensional passive walker with knee joints. An efficient 3D walker was later presented in [8] that had actuators to trigger different states in a finite-state-machine. These mechanisms provide an insight into passive dynamics of the

system, however they are limited to simple walking behaviours.

1.2.2 Underactuated Biped Robots

A steady walking cycle is a non-trivial periodic motion with non-linear hybrid dynamics. The continuous portion of the dynamics is underactuated [59]. Approaches that store energy when the leg touches the ground and use the stored energy for underactuated motion when the robot swings its leg have been explored in [19] [45]. Stable gait using hybrid zero dynamics is explored in [1]. These systems can be formally verified to stay stable and be highly efficient, however they lack complex behavior.

1.2.3 Fully Actuated Humanoid Robots

One of the solutions to generate stable walking using fully actuated robots, is using simplified models for the system dynamics (explained later in Chapter 2). This approach controls the trajectory of Zero Moment Point (ZMP) or Center of Pressure (CoP), which can be approximated to be linear with few specific assumptions proposed by Kajita et. al. [29] in Linear Inverted Pendulum Mode (LIPM) for biped walking. Now the controllers can be designed to stabilize CoM along the planned ZMP trajectory. LIPM paved a path for walking controllers that track a desired walking pattern by stabilizing ZMP trajectory using various methods like - preview control [28], Model Predictive Control (MPC) [51] [36], Receding Horizon Control [14]. In [60] ZMP tracking is done using a linearly constrained quadratic program for a Model Predictive Control. Tedrake et. al. [56] has proposed a closed-form solution for ZMP gait generation and feedback stabilization using linear-quadratic regulator (LQR). [27] [48] proposes online modification of a generated trajectory for walking. All of these methods require a pre-planned trajectory using a walking pattern generator. The generated trajectory is stabilized online using ZMP stabilizer.

Planning and executing a single step has been explored in various fall recovery strategies as seen in [52], [62], and [39]. A Model Predictive Control approach has been proposed in [53] for fall recovery by stepping. [11] proposes a method for push-recovery using Divergent Component of Motion(DCM). These approaches are intended for push recovery and do not focus on continued walking. Capture points [47] use the orbital energy of LIPM to find a

stepping location in order to stop. This is extended by [13] to design walking controllers using capture points.

The approach described in this thesis does not require a walking pattern generator. When commanding a robot to walk, only the next step is decided based on the virtual force provided to the controller. The robot walks one step at a time until the virtual force is acting on the robot. When walking multiple steps, every subsequent step is decided such that the existing momentum of the robot assists the movement, if it is in the same direction.

1.3 Assumptions

1. We will assume the terrain to be flat. However, this approach can be extended to slopes, stairs, and different leveled planes, as discussed in the last chapter.

2. We will assume constant height constraint as required by the LIPM model for generating footsteps. This is discussed in more details in chapter 2.

3. The robot is assumed to be in bent knee configuration as opposed to straight knee configuration. This serves two purposes 1) Avoid control complexities arising due to singular configuration of the legs, 2) Avoids non-linearity in the system dynamics. The low-level controllers proposed in [18] overcome this limitation.

1.4 Contributions

1. Novel approach to walking using Virtual Force for footstep generation and capturability based low-level controller for executing the planned step.Chapter 3 explains the math and intuition behind computing the next footstep for a humanoid robot. It also discusses design of a capture point based controller that is used for experiments in chapter 5.

    - A journal paper titled “One Step at a Time: Biped Walking with Virtual Force”, is submitted to RA-L and International Conference of Robotics and Automation 2020 that describes the footstep generation using virtual forces.

2. Transportable Opensourse API & UI for Generic Humanoids (TOUGH): This library is built for developing 'high level controllers' and 'state machines for complex tasks' using humanoid robots. It provides interfaces for perception, manipulation, motion planning, and walking capabilities of a humanoid robot. Chapter 4 explains the design and use of TOUGH. The code repository is available at https://github.com/WPI-Humanoid-Robotics-Lab/tough

    - A journal paper describing TOUGH library is currently under review at International Journal of Advanced Robotic Systems.
    - A conference paper titled "Extended State Machines for Robust Robot Performance in Complex Tasks", which is published at 2018 IEEE-RAS 18th International Conference on Humanoid Robots.

1.5 Structure of Thesis

The next chapter provides a quick overview of important concepts and requirements for balancing and walking of a humanoid robot along with the dynamics for simplified models used for controlling fully actuated humanoid robots. Chapter 3 provides derivation of a reactive controller using capture point followed by the desired capture point generator. It also presents the derivation of walking parameters for a humanoid robot based on virtual force and dynamics of the robot. Chapter 4 explains the software library – TOUGH, developed for integrating controls, manipulation, perception, and motion planning of humanoid robots. Chapter 5 presents the experiments performed in simulation and on the hardware with analysis of the results. The last chapter discusses the conclusions, applications, and the future work.

Chapter 2

Background

2.1 Balancing

Humanoid robots are inherently unbalanced. If there is no control input to the robot, the robot would fall. Worse yet, most humanoid robots cannot stand up by themselves from the fallen state. Due to this fact, any task for these robots is secondary. The primary task is to maintain its balance. At any given instant, the robot is balanced if the projection of its center of mass falls inside the convex hull of the foot-support area[40], also known as support polygon or base of support. Few more ground reference points are useful for understanding and controlling the balance of a humanoid robot, viz., 1. Zero Moment Point (ZMP), 2. Center of Pressure (CoP), and 3. Centroidal Momentum Pivot (CMP) point.

2.1.1 Zero Moment Point

When a robot is standing, the resultant force of all the inertial and gravitational forces of the robot can be assumed to be acting on the ankle. The horizontal components of this force are balanced by the static friction between the feet and the ground. The vertical component of the force can be balanced by positioning the reaction force such that net moment in vertical planes is zero. Under these conditions, the point at which reaction force is acting is defined as the Zero Moment Point(ZMP)[58]. It should be noted that there can be other links on the robot where the resultant moment is zero. However, for the balancing

of humanoid robots, we are only interested in the ZMP located inside the support polygon. If the forces don't balance out, ZMP ceases to exist.

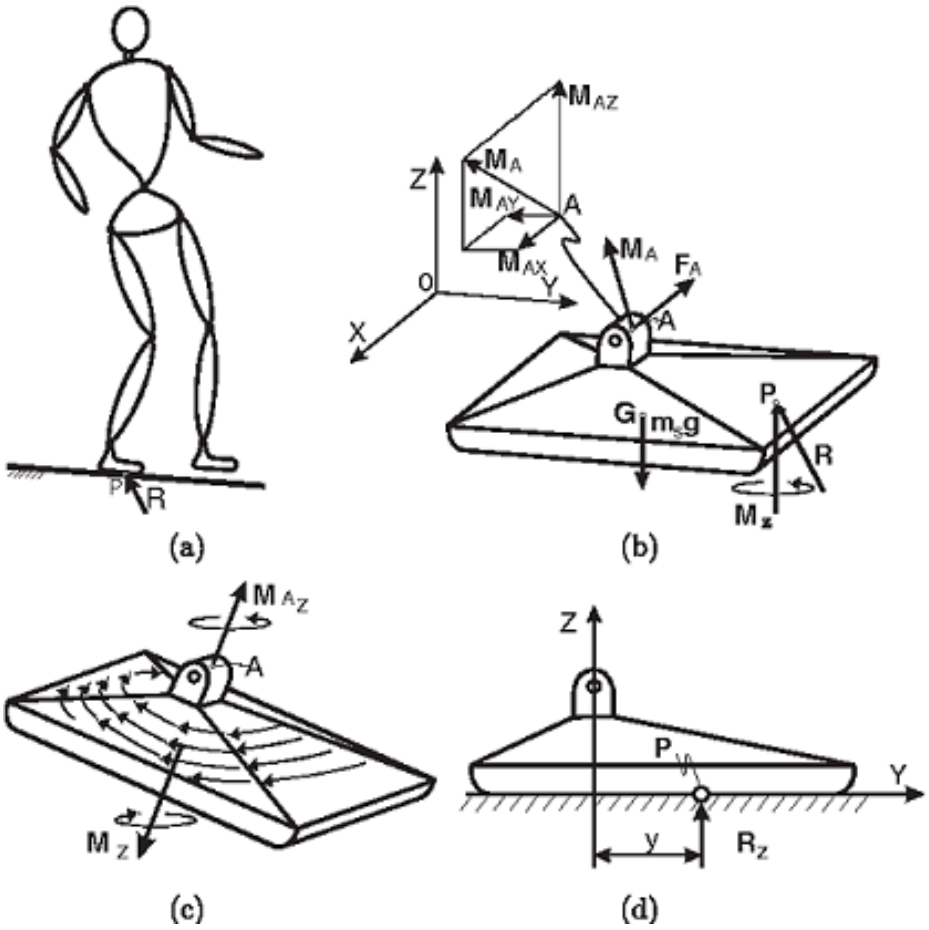


Figure 2.1: Biped mechanism and forces acting on its sole[58]

When a humanoid robot is balanced on one foot during walking, the forces acting on the sole of the robot are shown in Figure 2.1. $F_A : (F_{Ax}, F_{Ay}, F_{Az})$ is the resultant of all the inertial and gravitational forces of the robot acting at the ankle joint A. Ground reaction force on foot is shown as $R : (R_x, R_y, R_z)$ acting on point P. The mass of the foot is m_s, and the weight of the foot is acting at point G. The static equilibrium condition for the robot is given by equations 2.1 and 2.2,

$$R + F_A + m_s g = 0 \tag{2.1}$$

$$\overrightarrow{OP} \times R + \overrightarrow{OG} \times m_s g + M_A + M + \overrightarrow{OA} \times F_A = 0 \tag{2.2}$$

$$(\overrightarrow{OP} \times R)^{XY} + \overrightarrow{OG} \times m_s g + M_A^{XY} + (\overrightarrow{OA} \times F_A)^{XY} = 0 \tag{2.3}$$

$\overrightarrow{OP}$,$\overrightarrow{OG}$, and $\overrightarrow{OA}$ are radius vectors from the origin of the coordinate system O. M_A is the resultant moment acting at the ankle joint. $M : (M_x, M_y, M_z)$ is the ground reaction moment. Horizontal components of ground reaction forces (frictional forces) are balanced by horizontal components of F_A and the vertical ground reaction moment M_z is balanced by the vertical component of M_A. This is under no-slip condition. The horizontal components of M_A can be reduced to zero by positioning the point P such that moments M_x and M_y balances the corresponding components of M_A. Under these conditions, the resultant moments acting on point P are zero, and point P is the Zero Moment Point (ZMP). Equation 2.3 is the projection of Equation 2.2 on XY plane. It is the basis for positioning R such that ZMP exists inside the support polygon.

2.1.2 Center of Pressure

The pressure applied on the robot's foot(or feet) by the ground can be replaced by a force acting at a single point, known as Center of Pressure (CoP)[50]. On a flat surface, if this force balances out all the forces acting on the robot during the motion and the net moment is zero, it coincides with ZMP, and the robot is balanced. If there is a non-zero net moment acting around the CoP, in that case, ZMP does not exist, and the robot is not balanced[58]. It should be noted that in both balanced and unbalanced conditions, CoP always lies inside the support polygon.

2.1.3 Centroidal Momentum Pivot Point

For the angular momentum of the system to be conserved, the resultant external moment on a humanoid robot must be equal to the rate of change of its angular momentum. In Figure 2.2, point P is the CoP and F_g is the ground reaction force acting on the robot. In the left figure, the rate of change of angular momentum ($\dot{H}$) is zero, as the resultant external force pass through C, the Center of Mass (CoM). In the figure on the right, the rate of change of angular momentum is non-zero. If we move the resultant external force

on the ground plane such that it intersects the CoM, we get the point A on the ground. This is the CMP point[44][20]. $\dot{H}$ is proportional to the distance between CoP and CMP. It is essential to regulate the angular momentum while balancing.

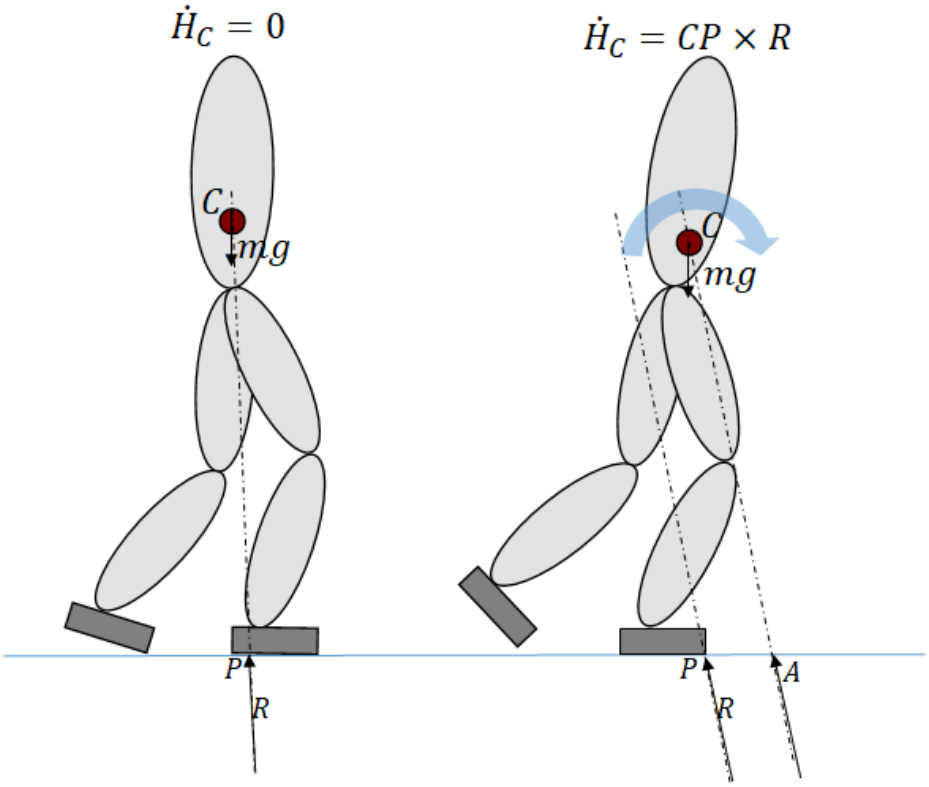


Figure 2.2: a) Rate of change of angular momentum acting on the CoM is zero b) Rate of change of angular momentum acting on the CoM is greater than zero

The CMP point(A) is always on the ground and can be computed if CoM location in ground plane (C_x, C_y, C_z) and the ground reaction force (F_g) is known using the equations below,

$$A_x = C_x - \frac{F_{gx}}{F_{gz}} C_z \tag{2.4}$$

$$A_y = C_y - \frac{F_{gy}}{F_{gz}} C_z \tag{2.5}$$

$$\tag{2.6}$$

2.2 Walking

Oxford dictionary defines walking as "Moving at a regular pace by lifting and setting down each foot in turn, never having both feet off the ground at once." While walking, the leg that is on the ground is referred to as stance leg, support leg, or anchor leg. The foot that is off the ground is called the swing leg. The maximum height a swing leg reaches is called the swing height and the heel to heel distance between two feet when they both are touching the ground while walking is called step length. Thus, walking can be split into 4 phases, as shown in Figure 2.3.

1. Liftoff: In this phase, the robot lifts the swing leg and balances on the stance leg. The liftoff can be instantaneous, if the entire foot is lifted at once, or it can be over an extended period if the toe is kept in contact with the ground as seen in the figure to allow bigger support polygon for a longer time.

2. Swing: In this phase, the robot moves its swing leg in a trajectory reaching swing height and also moving the body forward in the process.

3. Touch down: In this phase, the robot touches the swing leg on the ground and thus increasing the size of the support polygon.

4. Transfer: In this phase, the robot transfers its weight from one leg to the other, the swing and stance legs switch roles. The robot body keeps moving forward in this phase.

To maintain balance while walking, the ZMP should stay in the support polygon. In statically stable poses, the ZMP and the projection of CoM on the ground is the same point. Immediately after the liftoff phase, the size of support polygon shrinks to less than 50%, as seen in Figure 2.3. Hence, the entire weight of the robot must be transferred on the stance leg before liftoff. The transfer of weight ensures that the ZMP is inside support polygon, and the robot stays balanced. During the swing phase, the ZMP and the CoM move from one edge of the stance foot to the other. The time for which the support polygon is the same as the area of one foot is called swing time. The touchdown phase is triggered

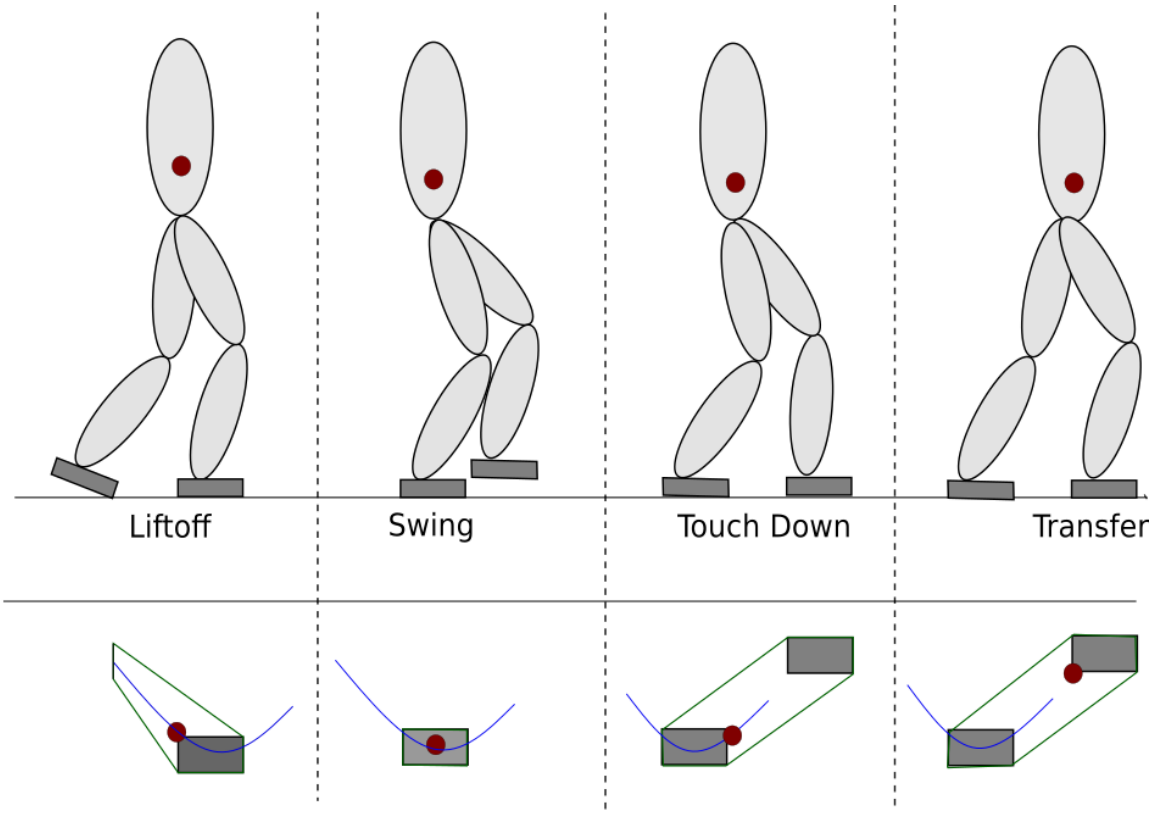


Figure 2.3: Walking phases. Upper part of the image is the side view and the lower part is the top view. The maroon circle is the CoM, the support polygon is shown in green, and the blue line is the CoM trajectory for a statically stable walk.

such that the support polygon grows before ZMP leaves the stance foot projection on the ground. During the transfer phase, the legs switch roles, and the ZMP and CoM move towards the new stance foot. The time required to move the weight from one stance foot to the other as the legs switch roles is called transfer time. Another common terminology used for walking is – single-support and double-support phase. In the single-support phase, the robot is balanced with just one foot on the ground, and in the double-support phase, both the feet are touching the ground. Hence, the stance foot is also referred to as support foot in literature. With this terminology, the swing time is the duration of the single-support phase, and the transfer time is the duration of the double-support phase.

Walking can be classified into two types – static walk, and dynamic walk. As the name suggests, the robot is in static equilibrium during a static walk. The projection of CoM on

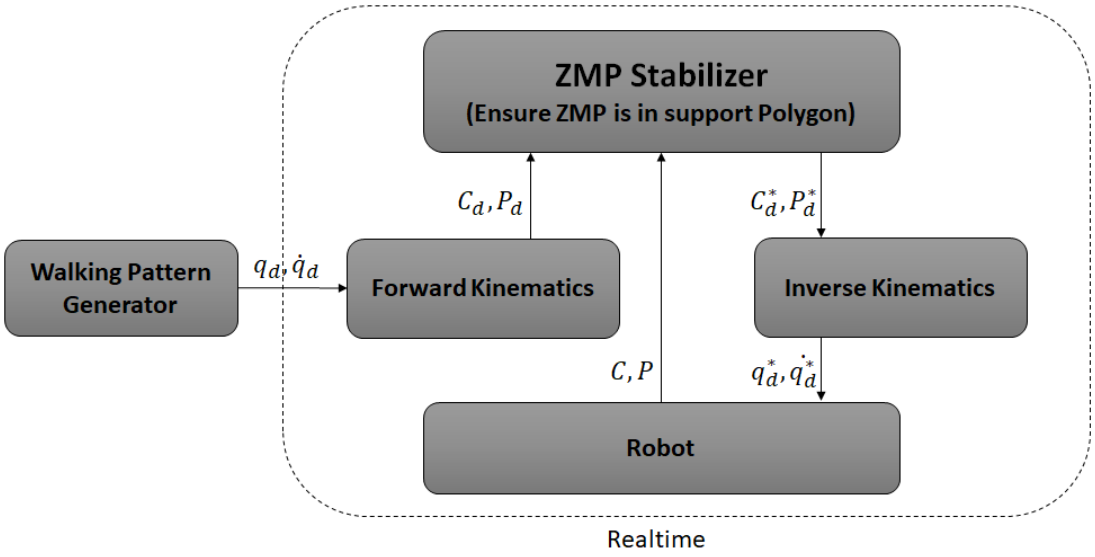


Figure 2.4: ZMP-Based Walking Controllers

the ground plane is always inside the support polygon. In a dynamic walk, the CoM might leave the support polygon momentarily, but the motion of the robot brings it back into the support polygon. ZMP always stays in the support polygon.

With the notion of ZMP, dynamic walking controllers can be designed by isolating walking pattern generation and stabilization around it. This is done by planning footsteps from a starting location to goal, generating CoM trajectories for those footsteps, and modifying the trajectories to ensure ZMP stays inside the support polygon before sending it to the robot. The process of modifying CoM trajectories to stabilize ZMP is known as stabilization. Simplest form of walking controller, as shown in Figure 2.4, solves forward kinematics for the desired joint angles(q) and velocities($\dot{q}$) to compute the desired CoM (C_d). Then, use the desired CoM position to determine the desired ZMP (P_d) location. Update the desired CoM based on stable ZMP ($P^*{}_d$) position computed using a simplified model, and then solving inverse kinematics to get the joint angles ($q^*{}_d$) and velocities ($\dot{q}^*{}_d$) for stable trajectory.

There is another class of biped walker called "Passive walkers"[9][8][59]. Passive walkers assume that walking pattern generation and stabilization are closely coupled and should

not be isolated. Rather, the design of the robot should generate a basin of attraction such that the robot walks instead of falling when a force is acting on it. In this thesis, we aim for a middle ground where the walking pattern generation and stabilizer run with real-time constraints, and the walking pattern generator is a function of stabilizer.

2.3 Modelling System Dynamics For Walking

At a higher level, there are two approaches for generating walking trajectories for a humanoid robot. The first one is using the exact model of the robot to generate CoM trajectories for walking. This requires precise knowledge of the robot's physical properties and external forces that will act on the robot during operation[22][61][42][25][43]. This approach provides good control over the robot, but it is of little use when the robot's exact model is unknown. The second approach uses a simplified model to generate the CoM trajectories and then controls the robot to follow those trajectories. This approach is useful when the robot's physical properties cannot be precisely known, computed, or modeled. It is also robust to changes in the environment and external forces acting on the robot, as long as there are corresponding sensors to provide feedback to the controller. In this research, we focus on the latter case.

2.3.1 Linear Inverted Pendulum Mode (LIPM)

Walking dynamics of a humanoid robot can be modeled similar to a linear inverted pendulum by assuming that the entire mass of the robot is concentrated at the CoM position and it is balanced on the ZMP with a torque-free joint. The legs are assumed to be massless and extensible. Constant height CoM trajectories can then be generated using the dynamics of LIPM[30][28]. As seen in Figure 2.5, the force due to gravity (mg) on the CoM of an inverted pendulum is balanced by the vertical component of the ground reaction force (F_g) acting on the inverted pendulum. The inertial force of the pendulum ($m\ddot{C}_x$) is balanced by the horizontal component of F_g. The horizontal component of F_g corresponds to the friction force on the ground. The resultant dynamic equations are linear and decoupled. Thus, it can be derived for one dimension as given by Equation 2.7 and can be used independently

for both x and y dimensions.

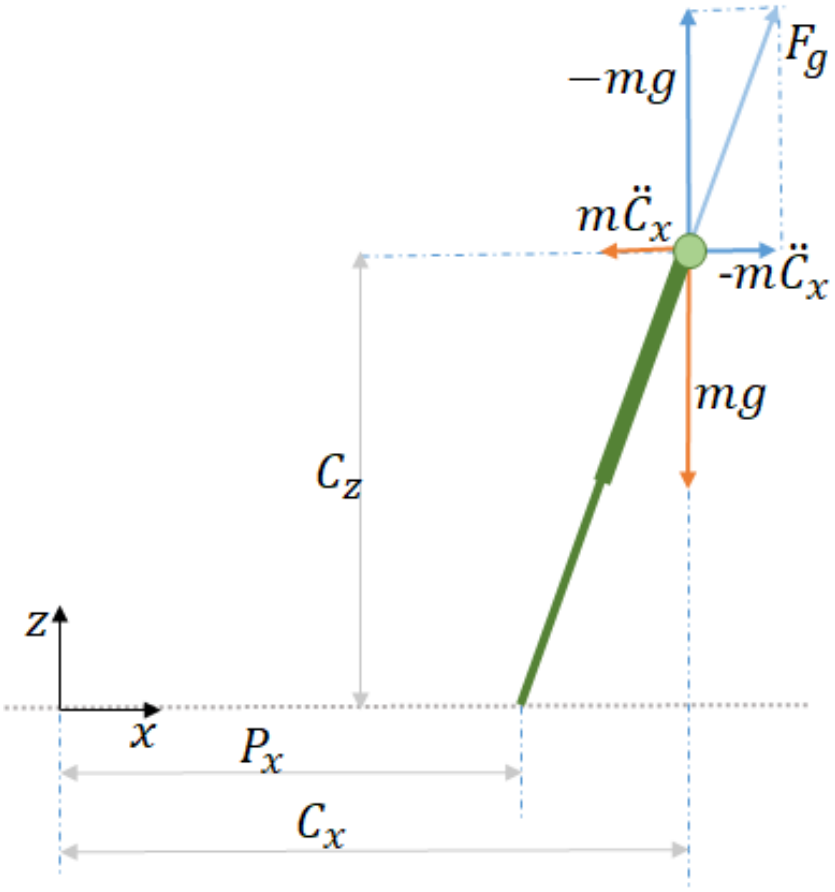


Figure 2.5: 2D Inverted Pendulum in Equilibrium

$$\ddot{C}_x = \frac{g}{C_z}(C_x - P_x) \tag{2.7}$$

Let C be the 2D coordinates of CoM (C_x, C_y) in XY-Plane, P be the 2D coordinates of ZMP, $\omega = \sqrt{g/C_z}$ and $\sigma = [C, \dot{C}]^T$, where σ is the system state consisting of position and velocity of CoM in XY-plane. Then the complete system dynamics of LIPM is given by,

$$\dot{\sigma} = \begin{bmatrix} 0 & 1 \\ \omega^2 & 0 \end{bmatrix} \sigma + \begin{bmatrix} 0 \\ -\omega^2 \end{bmatrix} P \tag{2.8}$$

If σ_0 is the initial state of the system, then the explicit solution for LIPM dynamics in Equation 2.8 is given by,

$$\sigma(t) = \begin{bmatrix} cosh(\omega t) & \frac{1}{\omega} sinh(\omega t) \\ \omega sinh(\omega t) & cosh(\omega t) \end{bmatrix} \sigma_0 + \begin{bmatrix} 1 - cosh(\omega t) \\ -\omega sinh(\omega t) \end{bmatrix} P \tag{2.9}$$

The above equations assume CoM is at a constant height and provides its position and velocity at time t. This can also be imagined as the extension of the leg required to maintain a constant height of CoM. It is obvious that the leg cannot extend beyond its kinematic limits. This provides the maximum step a robot can take for a specific pelvis height. As the robot touches the other foot on the ground, the dynamics is reset to the initial state.

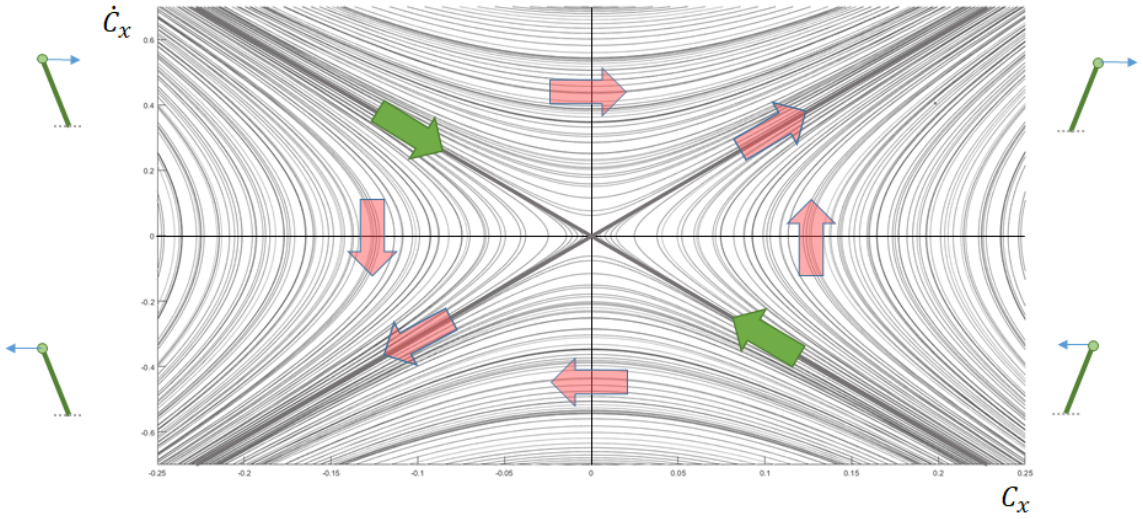


Figure 2.6: Trajectories of CoM when X-direction position is plotted against its velocity

Figure 2.6 shows the trajectories of CoM when the position of CoM is plotted against its velocity. In the first quadrant, CoM is ahead of its ZMP, and its velocity is increasing. This is the unstable region and always results in an ever-increasing velocity of CoM. Taking a step past ZMP in this state moves the system state to the second quadrant. In the second quadrant, the CoM is behind ZMP and is moving towards ZMP. During the motion, CoM velocity decreases, and if the velocity of the CoM is enough to cross the ZMP, it will pass ZMP and start accelerating after crossing it. If the velocity of CoM is not enough to reach ZMP, it will start accelerating in the opposite direction after reaching zero velocity. If

however, the system state lies on the eigenvector shown in a bold black diagonal line in the second quadrant, the pendulum will reach equilibrium state with CoM exactly above the ZMP, and the velocity would approach zero. The third quadrant is similar to the first quadrant with different directions for velocity and position. Forth quadrant is similar to the second, where the pendulum can reach equilibrium if the state lies on the eigenvector; otherwise, the CoM either overshoots the equilibrium or undershoots. At every step, the dynamics of the pendulum is reset.

2.3.2 Linear Inverted Pendulum Plus Flywheel

LIPM assumes that the entire mass of the robot is concentrated at CoM, and this point mass lacks rotational inertia. However, many human motions use angular momentum to maintain balance. In this model, the angular momentum is modeled using a flywheel that replaces the point mass of LIPM, as shown in Figure 2.7. This model allows acceleration of the CoM by changing the angular momentum stored in the flywheel[47]. A similar model for compensating angular momentum of the robot is used in Reaction Mass Pendulum[35].

The dynamics of the LIP with flywheel is given by,

$$\ddot{C}_x = \omega^2 C_x - \frac{1}{mC_z}\tau_h \tag{2.10}$$

$$\ddot{\theta}_b = \frac{1}{J}\tau_h \tag{2.11}$$

where, m is the mass of the flywheel, J is the rotational inertia of the flywheel, C_x and C_z are the coordinates of CoM, θ_b is the flywheel angle with respect to the vertical axis, and τ_h is the motor torque on the flywheel.

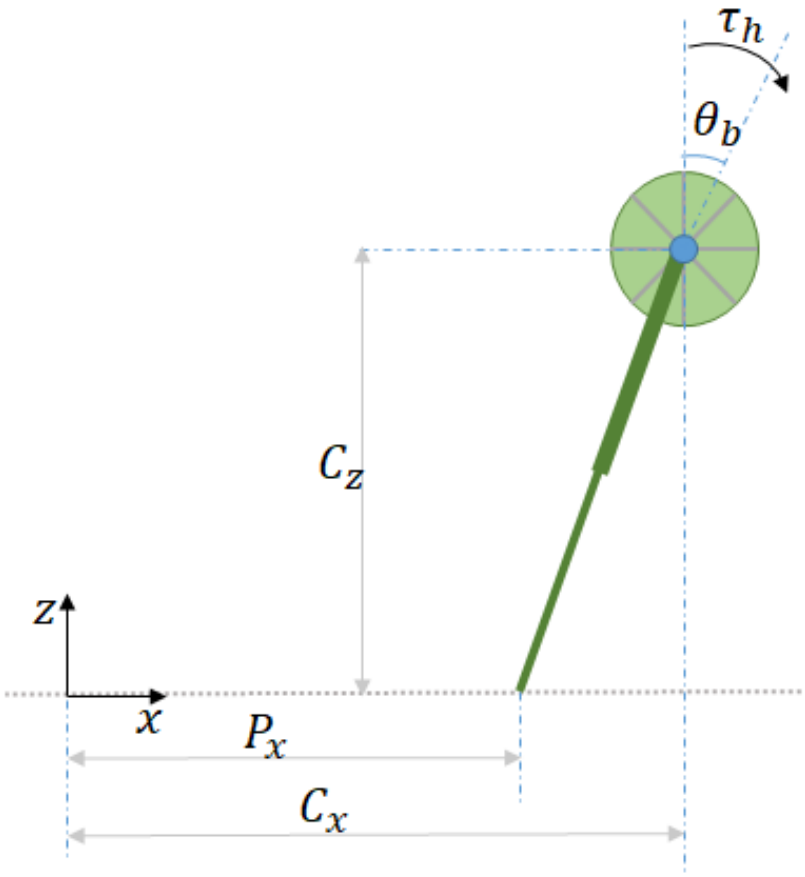


Figure 2.7: Abstract model of a biped in the single support phase with a flywheel body and massless legs. The swing leg is not shown. Two actuators of the biped are located at the flywheel center (also the CoM of the biped) and the leg.

Chapter 3

Stepping with Virtual Forces

We seek a region for stepping such that if a robot in a steady-state or moving, places its swing leg in that region, the robot will stay balanced and can stop if required. Various push recovery strategies already rely on a similar concept. Capture points and capture regions[47] provide an orbital energy-based approach to bring a robot to complete rest with N-Step Capturability. The same capture points are derived using the natural dynamics of LIPM in[13]. [52] shows a comparison of CoP based recovery, CMP based recovery, and stepping based recovery. The velocity of the CoM is used to determine step length for push recovery in this research. A passive model of a rimless wheel with two spokes is used in[62] to determine stepping location for a complete stop. A significant difference between existing push recovery mechanisms and the proposed method is the impact of swing foot. During push recovery, the impact of swing foot on the ground helps the robot to stop quickly, whereas this impact is not desirable while walking. The presented approach is an extension to the biped walking proposed in[13], and it obviates the entire footstep planning component. The commands for the robot to move in the direction of interest are expressed as a virtual force vector(F_v) that pushes the robot in the desired direction.

3.1 Capture Points and Capture Region

A capture point is a reference point on the ground, where the robot has to step to come to a complete stop. At this point, the total orbital energy of the robot is zero. External

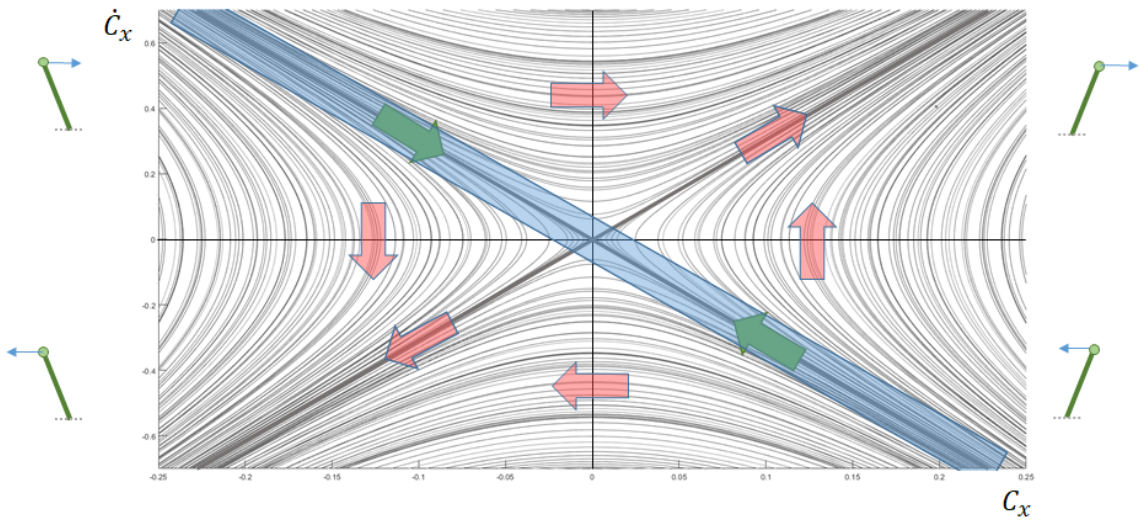


Figure 3.1: Trajectories of CoM. A stable region, shown in gray, can be achieved if we replace point foot with foot of width w

forces acting on the robot can move the capture point away from the robot. As a robot always have reaction mass and feet, one can adjust the CoP in support polygon to keep the robot stable. As the robot can use its mass to move the CoP position, there exist multiple capture points for a given state of the robot. Set of all the capture points in a given state is known as Capture region. Capture point lies on the eigenvector of the phase plot shown in Figure 2.6 and hence, it is guaranteed to reach equilibrium. With the addition of feet to the LIPM, that phase plot changes to Figure 3.1 allowing more states to reach equilibrium and thus expanding the capture point to capture region. Any point inside of the blue shaded region in the figure can be brought to zero velocity. Figure 3.2 shows three cases where the capture region can lie based on the forces acting on the robot. If the capture region is within the support polygon, the robot need not take any step. If the capture region is outside the support polygon but within the kinematic range of the robot, the robot needs just one step to come to a stop. If the capture region is outside the kinematic range, capture points can be found for multiple steps such that every step takes the robot closer to the 1-step capture region. Koolean et. al. [33] provides analysis of N-step capture regions with a conclusion that only the 4-step capture region is sufficient as long as every next step is (N-1)-step capturable.

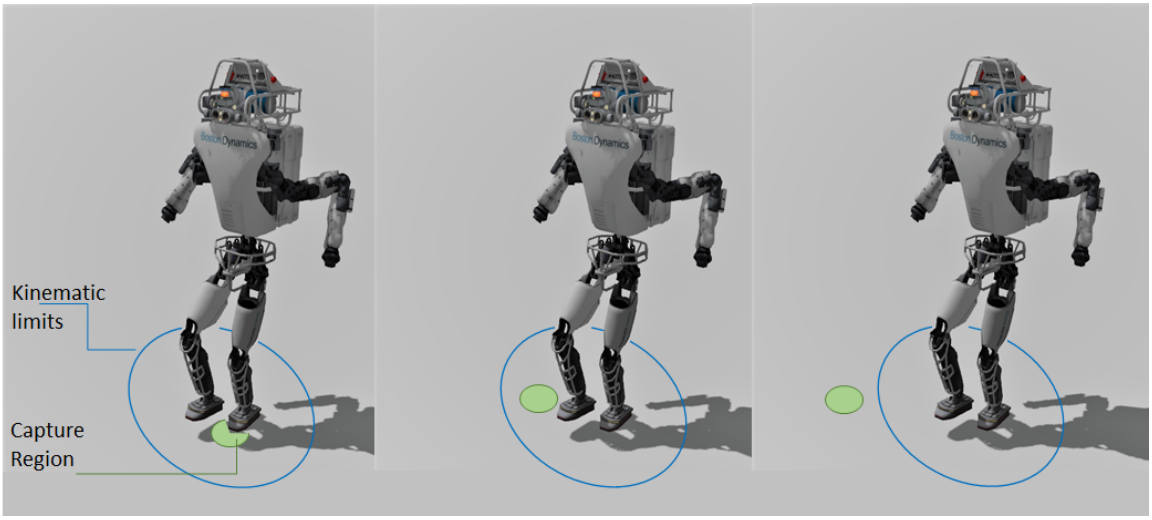


Figure 3.2: Capture region: a) No step needed as the capture region is inside support polygon. b) stepping in the capture region can bring the robot to complete stop. c) capture region is outside of kinematic limits, hence more than one step is required for the robot to stop.

3.2 Capture Point Dynamics

A capture point is a reference point on the ground, where the robot has to step to come to a complete stop. For LIPM, this implies that the CoM is located exactly over the ankle or $C_x = P_x$ in Equation 2.7. Location of capture point can be found by solving Equation 2.9 such that C_x tends to P_x as time tends to infinity. The Capture Point ξ_x is given by,

$$\xi_x = C_x + \frac{\dot{C}_x}{\omega} \tag{3.1}$$

Once we have the above equation, the CoM velocity($\dot{C}_x$) can now be represented in terms of capture point.

$$\dot{C}_x = -\omega(C_x - \xi_x) \tag{3.2}$$

Equation 3.2 shows that C_x has a stable first-order open loop dynamics with time constant $\frac{1}{\omega}$. Also knowing that the same equation applies to both x and y dimensions, we can assume $\xi = [\xi_x \quad \xi_y]^T$, $C = [C_x \quad C_y]^T$, and $P = [P_x \quad P_y]^T$. The dynamics of the capture point can be derived by differentiating Equation 3.1 as follows,

$$\dot{\xi} = \dot{C} + \frac{\ddot{C}}{\omega}$$
$$\dot{\xi} = -\omega(C - \xi) + \frac{\omega^2(C - P)}{\omega}$$
$$\dot{\xi} = \omega(\xi - P) \tag{3.3}$$

This shows that ξ has unstable first-order open-loop dynamics. As the CoM has a stable first-order dynamics, we can ignore it and use only the Capture Point dynamics to formulate a control law. CoM will follow the Capture Point, and when we stabilize the Capture Point, the entire system will be stable. In Figure 3.3, we can see that ξ moves in a straight line away from P, it's velocity $\dot{\xi}$ is proportional to the distance between P and ξ. The projection of CoM on the ground, C, moves towards ξ with a velocity proportional to the distance between ξ and C.

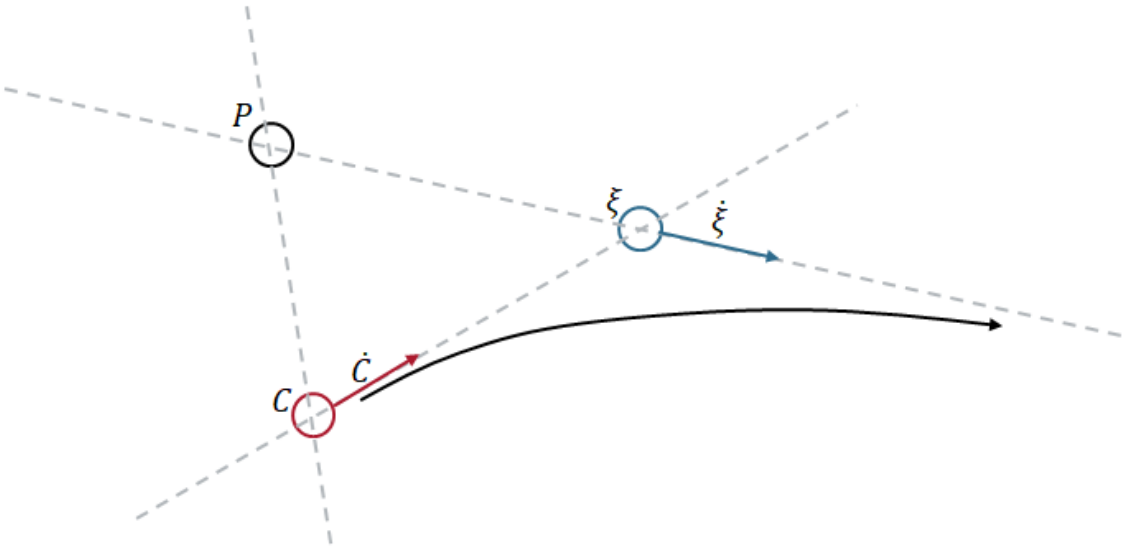


Figure 3.3: Capture point dynamics

For a constant ZMP position, P, the solution of equation 3.3 is given by,

$$\xi(t) = e^{\omega t}\xi_0 + (1 - e^{\omega t})P \tag{3.4}$$

$$P = \frac{\xi_d - e^{\omega dT}\xi}{1 - e^{\omega dT}} \tag{3.5}$$

$$P = \frac{1}{1 - e^{\omega dT}}\xi_d - \frac{e^{\omega dT}}{1 - e^{\omega dT}}\xi \tag{3.6}$$

In the above equation, dT is the desired time to reach the desired capture point ξ_d. Using this control law, we can set the desired capture point as the location of the next step to reach in time dT. If the desired value of the capture point changes when the robot is in motion, it tries to reach the new point in the remaining time dT and hence it can be used for writing reactive controllers. The point P used here is ZMP. However, the equations remain unchanged if CoP is used instead of ZMP [47]. As the CoP always lies inside the support polygon and can be controlled by shifting robots weight, it is convenient to consider CoP instead of ZMP. Figure 3.4 provides the complete picture of computing CoM coordinates C based on the desired capture point ξ_d. The desired capture point generator takes in a force vector to compute ξ_d. More details on the generator are presented in the next section.

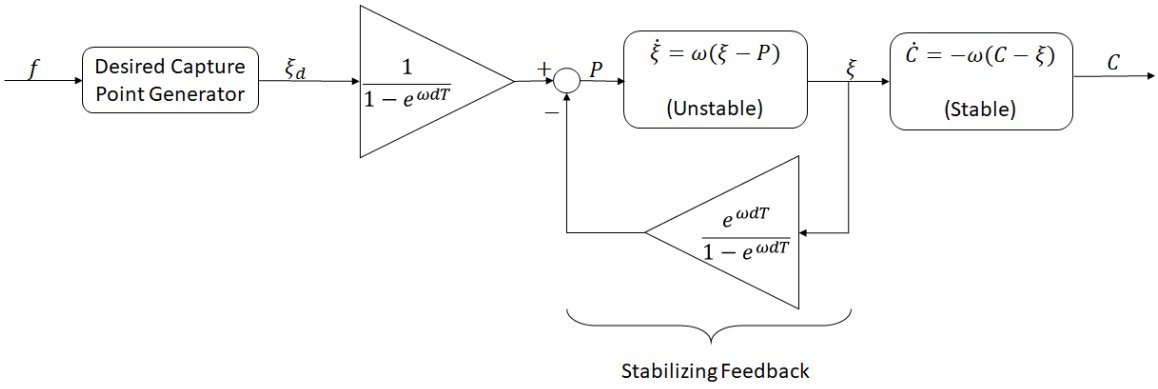


Figure 3.4: Controlling CoM based on desired Capture Point

3.3 Desired Capture Point Generator

As seen in the previous section, a biped robot can walk stably if it always steps inside the capture region by controlling the instantaneous capture point trajectory. The ability to step in the capture region is constrained by the kinematic limits of the robot's leg. In the following subsections related to LIPM with flywheel model, the kinematic limits are ignored for simplicity. However, these limits are added back in later subsections for completeness.

3.3.1 LIPM with Flywheel

For the LIPM model, only a single capture point exists, and hence it is of little value while generating the desired capture point. When a flywheel is added to LIPM, the capture point grows into a capture region. Figure 3.5 shows side view (XZ-plane) and top view(XY-plane) of a stable inverted pendulum with flywheel. The ground reaction forces (F_g) compensate for the frictional forces and weight of the pendulum. CoP (P) and CMP (A_0) are located at the ankle of the pendulum. When a virtual force (F_v) is applied at the CoM (C), the desired CMP (A_d) is computed using equations 3.7 and 3.8. F_v is assumed to be constrained in XY-plane. The virtual ground reaction forces under application of F_v are shown as F'_g. The blue circle in the XY-plane shows the capture region of the model. If A_d is inside the capture region, the next step should be at the point A_d; otherwise, the next step should be at the intersection of capture region perimeter and the line connecting A_0 and A_d as shown in Figure 3.5.

Each of the reference points used in this description have three coordinates. The vector form of these points is given below:

Center of mass: $C = [C_x \quad C_y \quad C_z]^T$

Ground reaction force : $F_g = [F_{gx} \quad F_{gy} \quad F_{gz}]^T$

Center of pressure : $P = [P_x \quad P_y \quad 0]^T$

Initial centroidal moment pivot : $A_0 = [A_{0x} \quad A_{0y} \quad 0]^T$

Virtual force : $F_v = [F_{vx} \quad F_{v_y} \quad 0]^T$

Desired centroidal moment pivot : $A_d = [A_{dx} \quad A_{dy} \quad 0]^T$

Desired capture point : $\xi_d = [\xi_{dx} \quad \xi_{dy} \quad 0]^T$

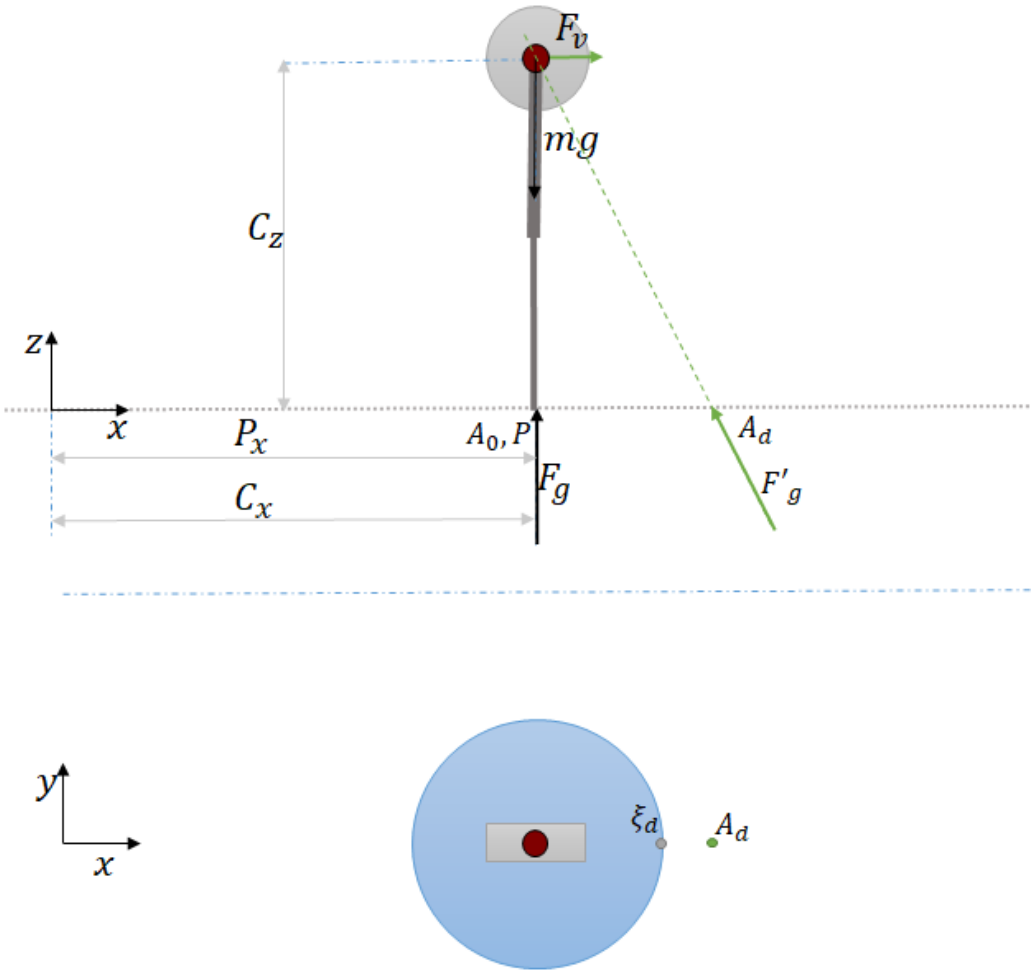


Figure 3.5: Virtual force F_v moves the initial CMP A_0 to desired CMP A_d. ξ_d is bounded by the capture region shown in blue circle in the XY-plane and lies on the line connecting A_0 and desired CMP A_d.

$$A_{dx} = C_x - \frac{F_{gx} - F_{vx}}{F_{gz}} C_z \tag{3.7}$$

$$A_{dy} = C_y - \frac{F_{gy} - F_{vy}}{F_{gz}} C_z \tag{3.8}$$

While walking, a robot spends non-zero time in double-support phase. As soon as the swing foot touches the ground, the next desired capture point is generated and sent to the robot. This happens before the robot has finished transferring its mass on both feet. In

Figure 3.6, CoM is already in motion after the first step such that the force in X-direction on the CoM is given by $m\ddot{C}_x$. The support foot before the touch down is not shown in the image. If the virtual force (F_v) is still acting on the CoM, the desired CMP is computed by using both the current acceleration and the virtual force. This approach results in the first step always being shorter than the following steps. This is expected behavior as the robot is stationary for the first step; however, the momentum generated while taking the step moves the capture region forward, allowing longer steps. At the last step, the flywheel absorbs additional energy to bring the robot to a complete stop. Acceleration in Y-direction would move ξ_d along Y-axis, and the stepping direction would change accordingly. As the capture point dynamics are decoupled and linear, the equations for both axes are similar.

3.3.2 Multibody Model

Leg Workspace

Figure 3.7 shows the top view of the ground plane with left foot anchored on the ground. The workspace of the right foot is bordered with a blue curve. This workspace includes safety margins to avoid self collisions of the robot. In (a), the pelvis is centered between the feet in top view. The pelvis can move in X or Y direction by moving the hip joints of the left leg, expanding the workspace of the right leg as seen in (b). We assume that the pelvis height stays constant and the knees of the robot are bent to avoid singular position of the legs. The expanded workspace in X-direction is shown in (c). The blue filled circles denote end position of the pelvis in top view. The workspace of one foot in the reference frame of other is always constant for a constant pelvis height. Hence it can pre-computed for a given robot. Pelvis movement in Y-direction is not accounted for; however, similar calculations would apply for Y-axis movement of the pelvis. Given the robot is standing with bent knees, the workspace can be computed by tracing the part of the circle on the ground formed by the ankle joint by extending the knees and moving the hip joints. Even though we consider completely stretched legs in computing the workspace, reaching of that singular position of the leg is not required as joints of the other leg can be actuated to move pelvis in the required direction.

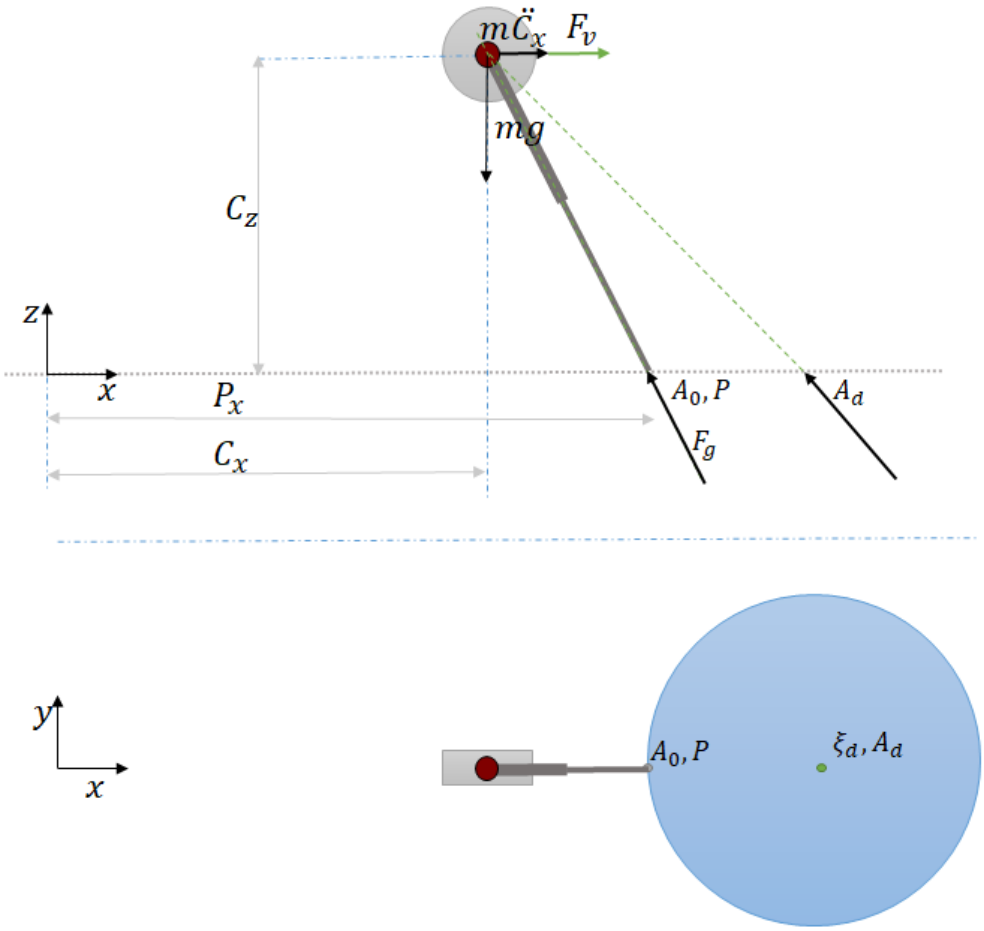
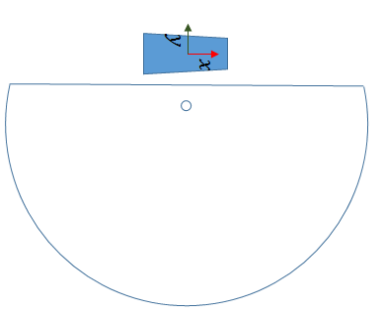


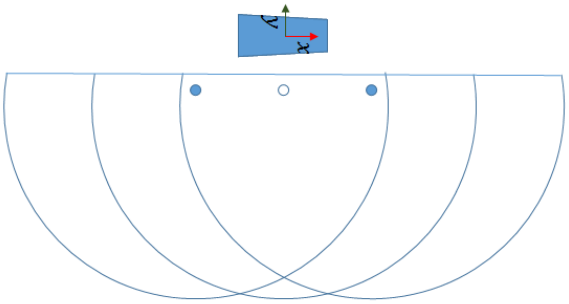
Figure 3.6: Desired CMP (A_d) is computed using current acceleration ($c\ddot{C}_x$) and the virtual force (F_v)

Capture Region

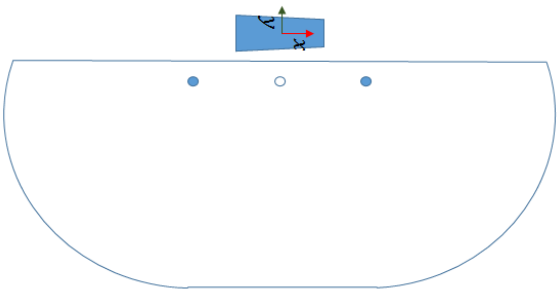
The dynamics of Linear Inverted Pendulum with Flywheel is given by equations 2.10 and 2.11. In these equations, τ_h is a variable that we can choose. This variable is bounded by the position, velocity, and acceleration limits of the upper body joints of the robot. If the value of τ_h is zero, it represents the LIPM dynamics. In this condition, only one capture point exists, and it is given by Equation 3.1. With an increase in the value of τ_h, this point expands to an area as described by the dynamics of LIP with flywheel model. Figure 3.8 shows the intersection of the Capture Region and foot workspace when external forces are acting on the robot. This area is computed under the application of virtual force F_v. The



(a) Pelvis is fixed.



(b) Pelvis can move to minimum and maximum values in X-axis.



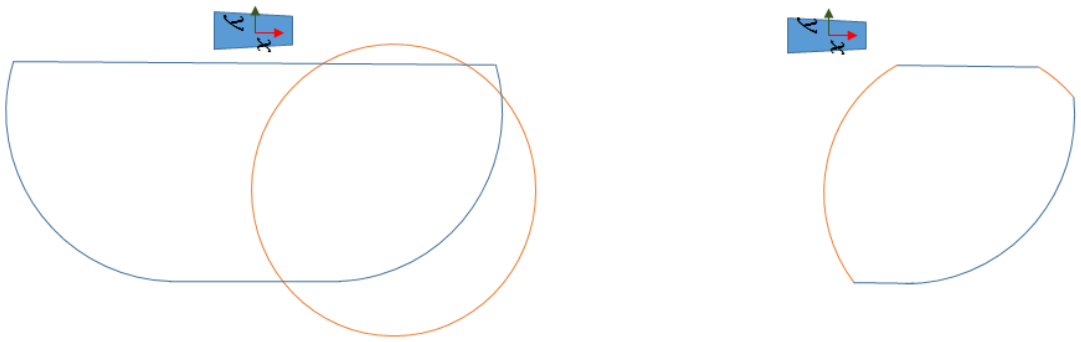
(c) Complete workspace, when pelvis can move in X axis.

Figure 3.7: Workspace of right foot with respect to left foot

capture region shown in the figure is circular and in front of the support foot for ease of understanding. With the upper body mass of the robot and support foot with area ¿ 0, the capture region is much bigger than what is portrayed in this figure.

Desired CMP and Desired Capture Point

The user provides a virtual force (F_v) as an input to the controller. This force is assumed to be acting at the height of CoM in X-Y (horizontal) plane. The force is positioned at the midpoint of the two feet frames in XY-plane. Current CMP (A_0) of the robot is computed using Equation 2.4. For computing desired CMP (A_d), the virtual force is subtracted from ground reaction forces (F_g) and the resultant force (F_g') is used in 2.4. If A_d lies inside the safe region to step, as shown in Figure 3.9 (a), then that point is considered as desired



(a) Workspace and Capture Region Superimposed

(b) Safe Region to Step.

Figure 3.8: Intersection of Foot Workspace and Capture Region

capture point. If A_d lies outside of the safe region as shown in (b), then the desired capture point lies on the farthest intersection of the vector $(A_d - A_0)$ and the safe region.

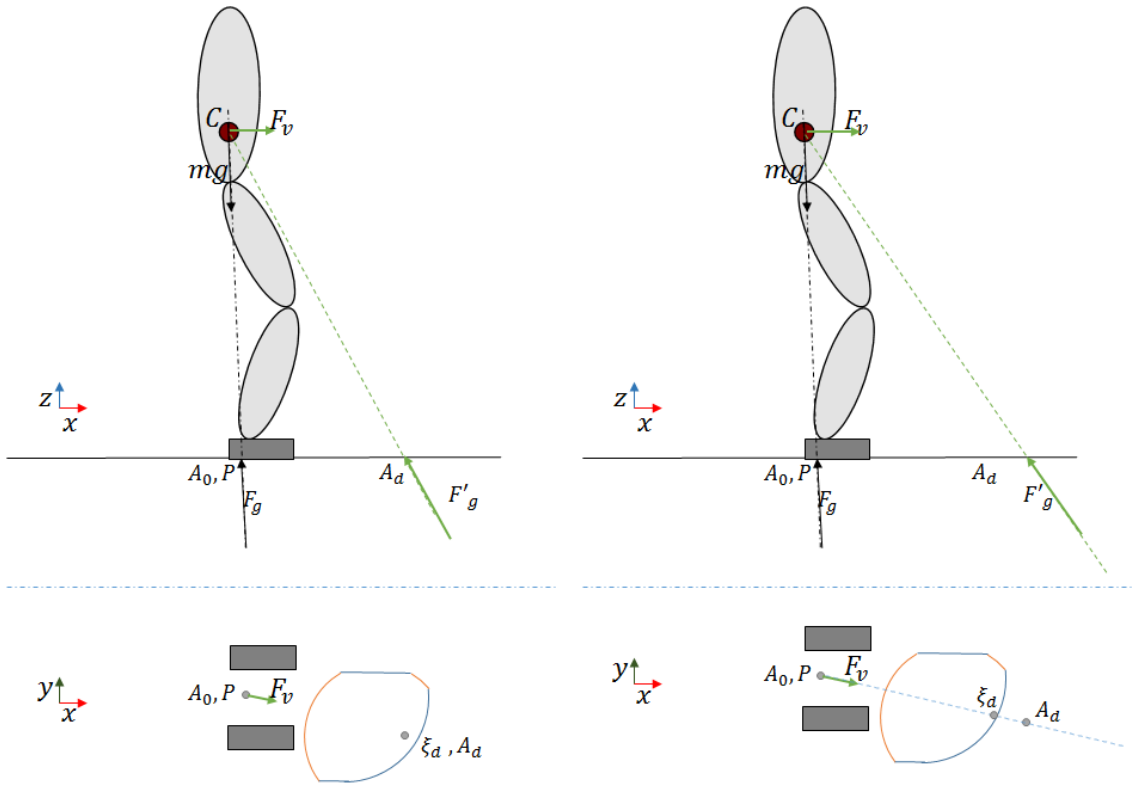
Foot Separation & Orientation

The derivations using LIPM can provide both X and Y coordinates for foot placement, however for a more natural-looking gait, the foot separation can be kept constant if the modified footstep lies in the safe step region. For the Atlas robot, the foot separation distance is set to 0.32m. This distance comes from the zero configuration of the robot. For the Valkyrie robot, it is 0.25m in simulation. Figure 3.10 shows 3 possible cases of application of virtual force. The center of a planned footstep, which coincides with the foot frame, is planned on the dotted line that marks the foot separation distance in a direction parallel to the applied virtual force.

The orientation of the footstep is based on the direction of the virtual force. If F_{vx} and F_{vy} are the x and y components of the virtual force (F_v), then the yaw(ϕ) of the foot frame can be calculated as:

$$\phi = \arctan \frac{F_{vx}}{F_{vy}} \tag{3.9}$$

We use arctan instead of arctan2 as that would guarantee the steps generated behind the robot are also feasible. When a virtual force is applied to the robot in the negative x-



(a) Desired CMP is inside safe region.

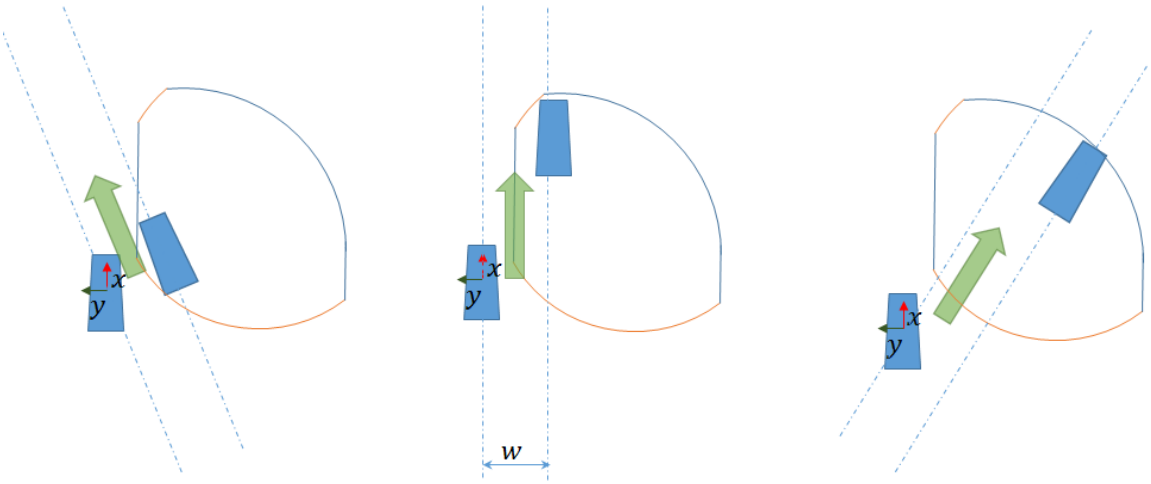
(b) Desired CMP is outside safe region.

Figure 3.9: Computing desired Capture Point ξ_d

direction, the robot will plan a step such that it walks backward instead of turning around. In the current implementation, the angle ϕ is rounded to nearest 5 degrees to avoid minor rotations of feet, while calculating the angle of a footstep. To adhere to the kinematic limits of Atlas robot, the foot rotation is modified to maintain positive angle for the right foot and negative angle for the left foot. This constraint does not apply to Valkyrie.

Frame of Application for the Virtual Force

The CoM of the robot follows a non-linear path when the robot is walking, as shown in Figure 3.11. If we apply a virtual force at the CoM, it will not allow the robot to walk straight as the CoM continuously moves in a non-linear path between both the feet. In order to avoid this situation, a new frame is created at the midway of both feet. The rotation of



(a) Virtual force is inclined towards left.

(b) Virtual force is aligned with X axis.

(c) Virtual force is inclined towards left.

Figure 3.10: Foot Orientation when virtual force is applied in different direction. w is the foot separation distance.

the new frame is average of rotations of both the foot frames. This new frame is at the same height as that of CoM. If L, R, and C are the vectors representing x, y, and z coordinates of the left foot frame, right foot frame, and the CoM respectively, the coordinates of new frame C' are given by equation 3.10.

$$C' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \frac{(L+R)}{2} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} C \tag{3.10}$$

The rotation of this frame is the average of rotations of left foot frame R_l and right foot frame R_r. Rotations are expressed as SO3 rotation matrices. Hence these cannot be averaged directly. In order to get the average, SO3 matrix is converted into its tangential plane denoted by *so3*. so3 is the Lie algebra of the Lie bracket SO3. so3 can now be divided by two and converted back to SO3. This process of averaging 3D rotations is known as "Spherical Linear Interpolation" or Slerp. Equation 3.11 shows the computation. The exponent and logarithm functions used in this equation are matrix functions, and the

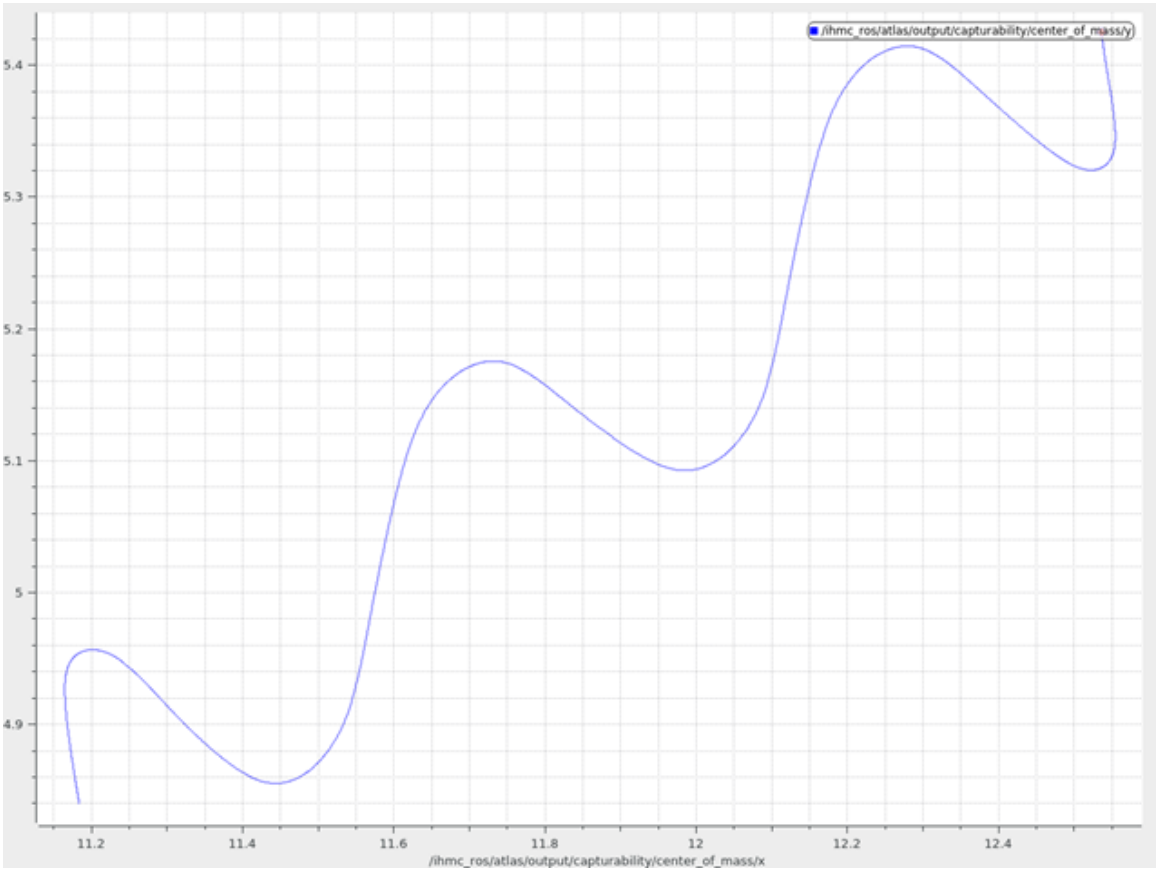


Figure 3.11: CoM trajectory as seen in the top view

values are derived using Taylor's series.

$$R_{c'} = \exp\left(R_l \frac{\ln\left(R_l^T R_r\right)}{2}\right) \tag{3.11}$$

This approach ensures that the virtual force is always applied at the center of both feet. As the desired footstep is computed when both feet are on the ground, the position or orientation of the foot frames while walking do not impact the calculation of the desired capture point. Y-coordinate of the generated footstep is adjusted to maintain a constant distance between both the feet.

3.4 Walking Parameters

The low-level controller expects the following walking parameter to generate leg trajectories and execute walking motion:

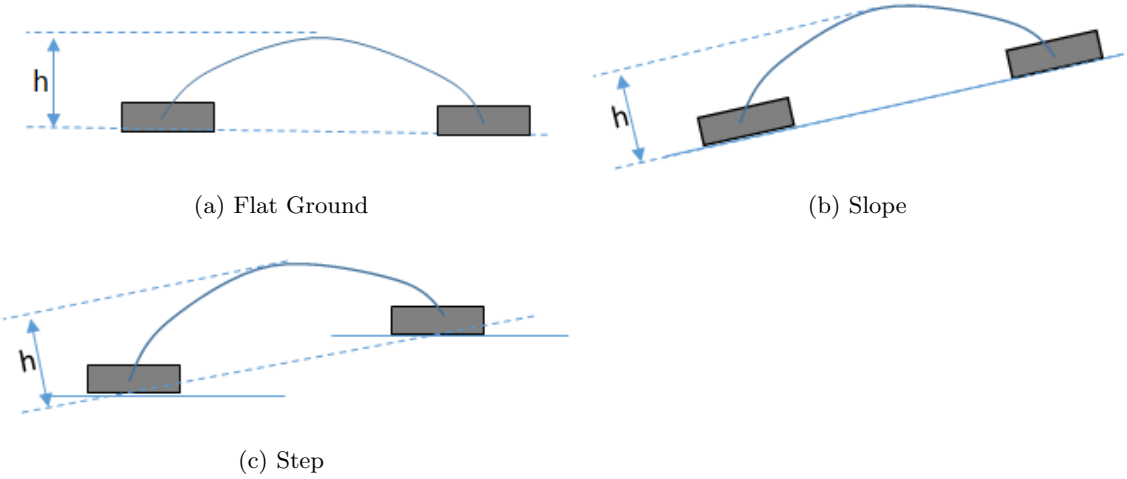
1. Step length
2. Swing height
3. Swing time
4. Transfer time

3.4.1 Step Length

Step Length is the heel to heel distance between feet. When the robot is standing in zero-configuration, the first step usually is around half the length of the maximum step length, as seen in Figure 3.5. The smaller footstep is because the velocity of the CoM is zero at time $t = 0$. When the robot starts walking, the step length is longer due to the acceleration of the CoM. The step length is maximum for every next step if F_v is high enough to move the desired CMP outside of the safe region to step. The foot separation distance is adjusted in the final position before sending it to the controller, such that the distance in y-axis between the 2 feet stays nearly constant, except for when the robot is turning. This modification allows for the steps to look more natural and human-like.

3.4.2 Swing Height

Swing height is the maximum height that the swing foot reaches while executing the leg trajectory. Swing height is always referred in the plane that contains the center of current footstep and the center of next footstep. Hence, we can use a constant swing height for flat ground, slopes, or steps. However, if there are obstacles that are to be stepped over, swing height must be increased. A constant swing height of 0.18m is found to be working well for the Atlas robot in both the simulation and on the real robot.



(a) Flat Ground

(b) Slope

(c) Step

Figure 3.12: Swing height on different terrains

3.4.3 Swing Time

Swing time is the time required for executing swing trajectory. During this time, the robot is in single support phase. Swing time can be computed by rearranging Equation 3.4 and solving for t when $\xi(t) = \xi_d$ where ξ_d is the desired capture point. Let $t = T_s$ when $\xi(t) = \xi_d$,

$$\xi_d - P = (\xi_0 - P)e^{\omega T_s} \tag{3.12}$$

$$T_s = -\frac{1}{\omega} \ln\left(\frac{\xi_0 - P}{\xi_d - P}\right) \tag{3.13}$$

In Equation 3.13, the fraction inside ln is inverted to avoid divide by zero error at runtime. The lower bound of T_s is defined by the achievable acceleration of the actuators of the robot. On Atlas robot, it is experimentally found out to be 1.2 seconds. If the value of T_s is high, the instantaneous capture point will move further away than the desired location, and ideally, the robot would need multiple steps to come to a stop. However, in this approach, we only use the capture point dynamics to generate walking parameters, and the low-level controllers execute the trajectory by planning for the desired values. Hence, the increase in swing time results in a static walk instead of unstability in the system.

3.4.4 Transfer Time

Transfer time is the time for which the robot stays in double support phase while walking. This time is required for transferring the weight of the robot from one foot to the other. While writing the dynamics equations for a simplified model, we often consider an instantaneous switch of the swing foot and the support foot. In reality, however, when switching the support foot, we have to wait until CMP enters the convex hull of the support leg and the dynamics of the robot is such that the acceleration of CoM would stay in the forward direction even after lifting the other foot. The CoP can be controlled to move from heel to toe during a walk resulting in smooth transition in-to and out-of transfer phase. On Atlas robot, this was found to be 0.8 seconds, for the existing controllers.

Chapter 4

Transportable Opensource API & UI for Generic Humanoids

Humanoid robotics is a complex and highly diverse field. Most humanoid robots have more than 20 degrees of freedom and may have up to several dozen sensors. Developing software for such robots is challenging because one needs to correctly integrate all the actuators and sensors to operate in a highly coordinated manner. TOUGH solves this problem for two state-of-the-art humanoid robots — Boston Dynamics' Atlas V5 and NASA's Valkyrie R5 (henceforth referred to simply as "Atlas" and "Valkyrie" by extending software provided by the Florida Institute for Human & Machine Cognition (IHMC)[54] and integrating several ROS libraries. Most humanoid robot tasks require the integration of perception, manipulation, and locomotion realized through planning and control subject to constraints that are specific to each type of humanoid robot. For example, a humanoid manipulation task may require the planning of arm motions while ensuring that the robot remains balanced; footstep planning should consider robot-specific limitations; perception algorithms should be robust to vision sensor vibration. To successfully integrate these modules, they must be developed by considering their impact on each other.

TOUGH uses the legged locomotion algorithms and optimization based momentum controller framework provided by IHMC Open Robotics Software [54]. The algorithms are generic to several humanoids and quadrupeds. The controllers are written in Java with

a network interface supporting ROS-Java integration through defined messages. ROS [49] is one of the most widely used middleware for research and education in robotics. The TOUGH APIs discussed in this chapter communicate with the software mentioned above using ROS topics to read robot state, sensor data, and send custom commands to the robot. Algorithms developed to work on one robot can be tested on another with great ease to make comparisons. The minimal setup and independent modules of TOUGH are designed to help new researchers and students focus on specific tasks without losing the impact of other modules on the task in hand. This framework attempts to provide a generic and integrated software stack which would aid new researchers, educational programs and the robotics community get started with humanoid robotics.

Using TOUGH, users can focus on high-level algorithms without worrying about robot specific configuration, system setup, message generation, and communication. IHMC provides open-source access to their low-level controllers along with several robotics libraries; however their mission control repository, which is used for operating the robot is closed source. TOUGH utilizes most of the features provided by IHMC software and abstracts overwhelming details required by the low-level controller. It also adds customizable GUI, ease of use, and several examples.

The next section talks about the existing open-source libraries commonly used in humanoid robotics, followed by the design of TOUGH. Getting Started section provides an example task highlighting the ease of use followed by performance comparison. The section after that explains available Docker images. The last section presents three use cases of the APIs.

4.1 Related Work

There are several open-source libraries for control of legged robots. Pinocchio[4] specializes with fast rigid body dynamics algorithms used for robotics, computer animation, and biomechanical applications. OpenSoT[23] allows rapid prototyping of controllers for high degrees of freedom (DoF) robots. It can be used for computing inverse kinematics, inverse dynamics, or contact force optimization, which is commonly used while controlling hu-

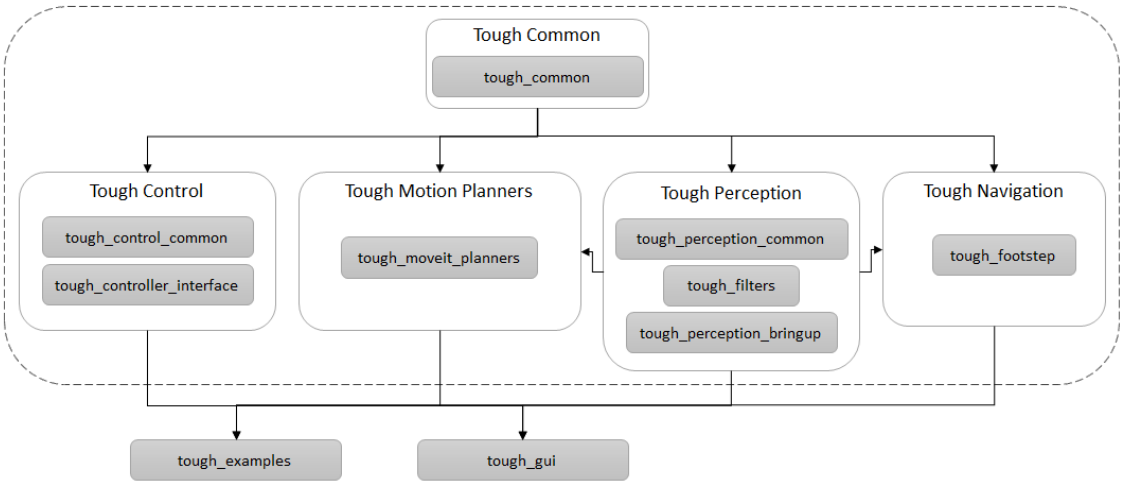


Figure 4.1: Packages in TOUGH APIs

manoid robots. The iDyntree[43] library was designed specifically for control of free-floating robot as part of the iCub project. Drake[57] provides tools to analyze the dynamics, design control systems, navigation, and planning algorithms for all kinds of robots. Controlit[16] provides a software framework for whole-body operational space control. These libraries provide a particular set of tools for dynamics and control of legged robots.

On the contrary, TOUGH is designed to provide seamless integration of manipulation, navigation, perception, and control. There is a heavy emphasis on high-level user experience and effectively managing low-level control for complex robots. TOUGH allows novice users to control the robot at a higher level by abstracting low-level details. More advanced users, on the other hand, can write algorithms for control, motion planning, or perception based on sensor feedback from the robot and send the control input at joint level.

4.2 Design

TOUGH uses ROS as middleware to communicate with the momentum-based controller[32] running on the robot or simulator. At a higher level, TOUGH takes input from user converts it into appropriate ROS messages and sends it to the robot. TOUGH APIs consists of five modules and two ROS packages, as shown in Figure 4.1. The modules inside the

dotted lines form the core of TOUGH. These modules communicate directly with the robot. tough_examples provide examples of using the core module, and tough_gui provides a graphical user interface (GUI) for operating the robot. Tough common contains classes that are required by all the other packages. *Tough control* allows sending joint level commands to different parts of the robot. *Tough perception* provides utilities for data processing of vision sensors. *Tough motion planners* allows planning of taskspace trajectories which can be executed using the controller interfaces. Though the motion planners are not directly dependent on *Tough control*, a user would generally need to make use of *Tough control* for sending planned trajectories to the robot. *Tough navigation* is used for sending leg trajectories to the robot. Although a user can move the legs in the desired position in taskspace using this package, it is most commonly used for sending either footsteps or goal to which footsteps should be planned and executed by the robot.

TOUGH acts as an abstraction layer to hide details required by the low-level controller that does not concern the user. For example, to change the pelvis height of the robot to 0.8m in 1s using ROS message, a user has to send the following values as a ROS message:

```
taskspace_trajectory_points:
- time: 1.0
  position:{x:0.0, y:0.0, z:0.8}
  linear_velocity:{x:0.0, y:0.0, z:0.0}
  unique_id: 0

frame_information:
 {trajectory_reference_frame_id:-102,
 data_reference_frame_id: -102}

use_custom_control_frame: false

control_frame_pose:
  translation: {x: 0.0, y: 0.0, z: 0.0}
```

```
  rotation: {x:0.0, y:0.0, z:0.0, w:0.0}


execution_mode: 0
previous_message_id: 0
unique_id: 1
```

Listing 4.1: ROS Message to Change Pelvis Height

The above snippet is an example of ROS message to set pelvis height. It has some variables whose values are essential for the user to know, like that of time and position in *taskspace_trajectory_points*. However, there are some variables whose values are not required or could be computed without user input in most cases. For example, the reference frame IDs in *frame_information* mentioned as -102 is a hash for world frame, which need not be known to the user. Same is the case with the last variable, unique_id, if its value is 0, then the message is discarded by the controller. The above message is reduced to the following two lines using TOUGH

```
/* nh is a ros nodehandle */
PelvisControlInterface pc(nh);
pc.controlPelvisHeight(0.8, 1.0);
```

Listing 4.2: TOUGH example

4.2.1 Tough Common

Tough Common module is used for fetching details of robot model and robot state. It consists of two classes – RobotDescription and RobotStateInformer. Both the classes follow the singleton pattern. Singleton implementation allows the creation of only one object which is shared with all the classes that access information through these classes. RobotDescription provides information about robot model, like the frame names, joint names, joint limits, etc. RobotStateInformer provides the current robot state, i.e., values of the robot's joint angles, velocities, and efforts. It also provides functions to get the external forces from force sensors and the accelerations from the Inertial Measurement Unit (IMU)

sensor. The robot state is updated at 1KHz. It also provides methods for querying current pose of frames and transforming points or frames between different base frames. Figure 4.2 shows the class diagram of tough_common package. In the figure, RobotState is a struct with a joint name, position, velocity, and effort of a joint. RobotSide is an enum with two values, LEFT and RIGHT. It is used to specify the side of the robot when sending commands to arms, grippers, or legs.

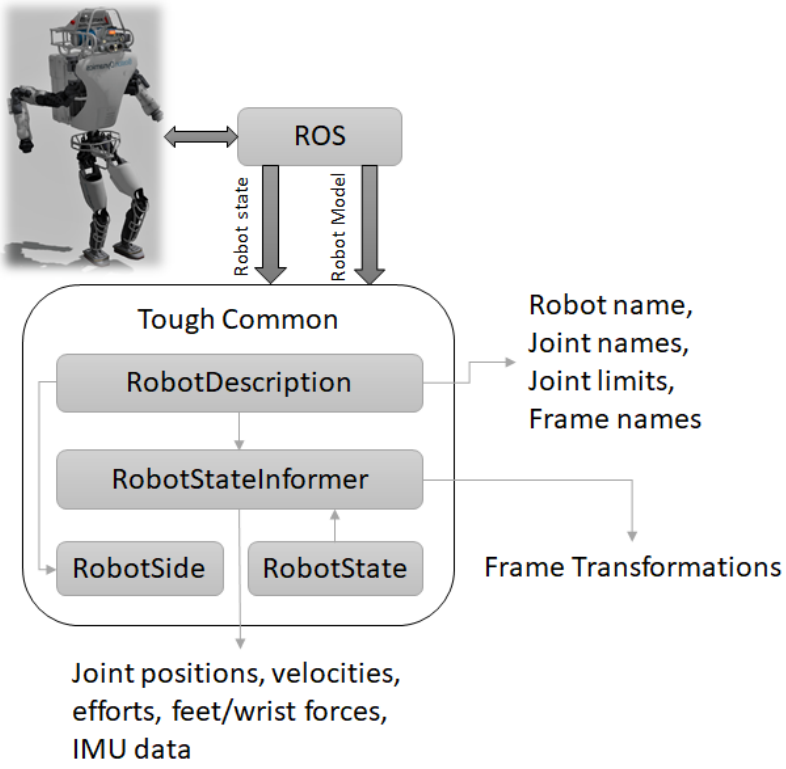


Figure 4.2: Overview of Tough Common module. This module uses ROS to fetch sensor data from robot/simulator and makes it available to the user via functions from tough_common library.

4.2.2 Tough Perception

Tough Perception module is an interface to access Multisense SL sensor using two classes, MultisenseImage and MultisensePointCloud. It also provides a few utility nodes that can be used with any other sensor. Figure 4.3 shows an overview of tough perception. Multisense

SL, has a stereo camera and a spinning Hokuyo lidar. The right camera of the stereo camera pair provides Monochrome image, and the left camera is RGB. Using images from both cameras, we can determine depth information. MultisenseImage class provides images from the stereo camera along with organized RGBD pointcloud which can be used to find world coordinates of a pixel in the image.

Lidar sensor provides laser scan, which consists of range and intensity, of points in a single plane. The Lidar spins about an axis parallel to the ground, to provide laserscan in all planes that can be assembled to form a 3D pointcloud representation. MultisensePointCloud class provides access to the lidar data. As the Lidar needs to spin to gather 3D data, it takes about 3 seconds to generate an assembled 3D pointcloud. On the other hand, the stereo camera provides a stereo pointcloud at higher update rates and lower data size but with the cost of less accuracy.

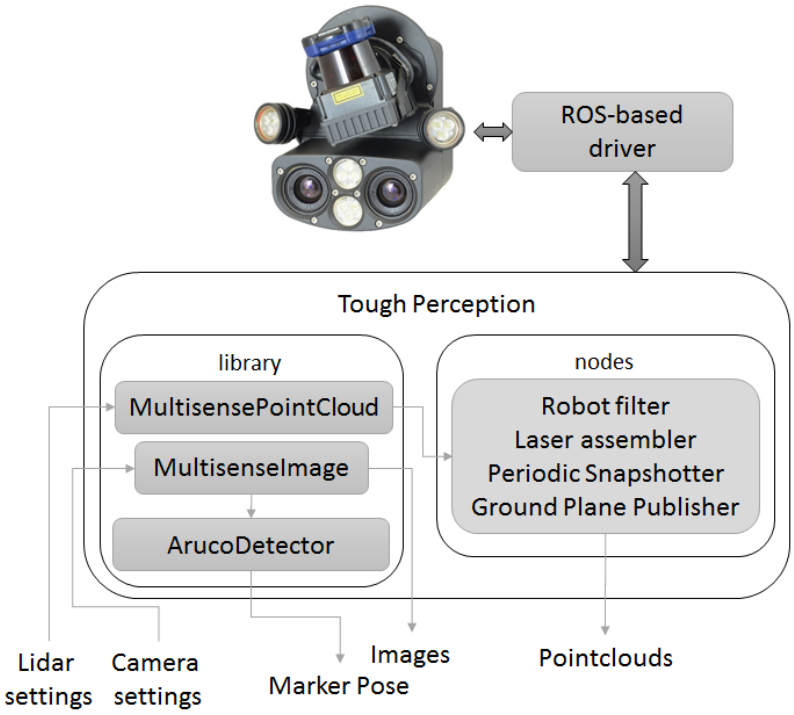


Figure 4.3: Overview of Tough Perception. The libraries shown on the left provide access to the Multisense SL sensor and the nodes shown on the right provides pre-configured pointclouds.

This package includes utilities that assemble pointcloud, provide registered pointcloud,

and provide a ground plane. Laserscan assembler assembles all laserscans within a set time to form a pointcloud. This assembled pointcloud is merged with previously assembled pointcloud to provide a registered pointcloud. The registered pointcloud has all the detected points filtered to a density of 1 point per 5cm voxel. The ground plane is detected based on the lowest foot height of the robot and surface normals from the detected plane. ArucoDetector class provides detection of objects using ArUco markers[17]. This class detects and provides pose of registered ArUco markers in the field of view of the robot.

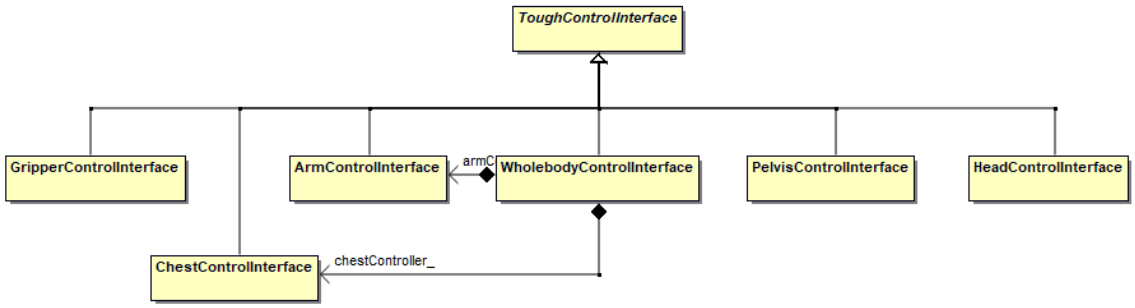


Figure 4.4: Class Diagram of Tough Controller Interface.

4.2.3 Tough Control

Tough Control module provides classes to interface with robot controllers using ROS. Figure 4.4 shows class diagram of the tough_controller_interface package. Each of the body parts has a controller interface class associated with it. These interface classes allow a user to send commands to the required part of the robot body. Each class provides functions for that specific part. For example, Chest controller interface allows rotation of torso about x, y, or z axes whereas pelvis controller interface allows setting pelvis height of the robot. Commands to multiple controller interfaces can be sent simultaneously, and the robot executes the most feasible trajectory that keeps the robot balanced. If planners are used for generating whole-body trajectories, these trajectories can be sent to the robot using wholebody controller interface. All the controllers accept only joint level trajectories; however, motion_planners explained in the next subsection can be used for taskspace planning.

4.2.4 Tough Motion Planners

Tough Motion Planners provides planning capabilities using MoveIt[6] configuration for four predefined planning groups. Two groups for 7 degrees of freedom (DOF) planning of each side, starting from the shoulder until hand and two groups for 10 DOF planning of each side that include roll, pitch and yaw of the chest along with 7 DOF arms. TaskspacePlanner class provides methods to generate joint trajectories for a given 6D point in taskspace. It can also be used for generating trajectories to follow a set of waypoints in taskspace. These joint trajectories can then be executed on the robot using tough_controller_interface. It also provides an inverse kinematics solution using TracIK[3] for any of the planning groups.

4.2.5 Tough Navigation

Tough Navigation module provides RobotWalker class that can be used to send footstep locations to the robot. It allows the user to customize footstep parameters like the step length, swing height, swing time, and transfer time. Footsteps can be generated based on either fixed offset that user provides or based on a goal location. Search based footstep planner[24] is configured for Atlas and Valkyrie based on the configuration of each robot. The footstep planner needs a 2D occupancy map which is generated by map_generator node in this same module. It uses the ground plane filtered by Tough Perception module to create a map.

This module also provides a class FrameTracker, that can be used to track motion between 2 frames. This class is useful in cases where one needs to programmatically check if the robot is walking. Though the FrameTracker class is developed in the context of walking, it can be used for tracking motion between any two frames. Another utility class in this package is FallDetector. FallDetector provides a way of knowing if the robot is standing on its feet or has fallen down. This is useful in scenarios where an operator cannot see the robot or in cases where the robot is working in complete autonomy.

4.2.6 tough_examples

tough_examples are split into four categories, namely control, manipulation, navigation, and perception. Each of these categories provides a comprehensive set of examples which describe the proper usage of the APIs. The examples provided here are also used for sending any quick commands to the robot, like reset_robot to its default position or rotate the neck to see around or walk a few steps. All of these examples take arguments and perform one specific task.

4.2.7 Tough GUI

Tough GUI provides the necessary information required for operating a humanoid robot along with a view of current joint angles, the current pose of the robot, and buttons to send different commands to the robot. If required, RViz[21] can be used along with the GUI for visualizing the data from different angles. A default configuration file for RViz is included in tough_gui package. As seen in Figure 4.5, Tough GUI provides RViz based render panel and other visualization tools. The robot model is displayed in either 3D or 2D view in the central panel. The lower left widget provides a view of the robot's camera. The lower central widget displays the current values of various joint angles and the current position of the robot. The widget on the lower right consists of different tabs to control each of the body parts. These tabs are

- Nudge - allows nudging either right or left hand by 5cm in task space using direction buttons.
- Arm/Chest/Neck/Gripper - provides sliders to move individual joints to move into a specific configuration.
- Walk - provides an interface to move a fixed number of steps by a fixed offset, change walking parameters, and to change the pelvis height.

The top toolbar has RViz tools that can be used to fetch the coordinates of a clicked point in the render panel, measure the distance between 2 points, send a 2D navigation goal for the robot to walk. When *2D Nav Goal tool* is used to send a goal location, the footstep

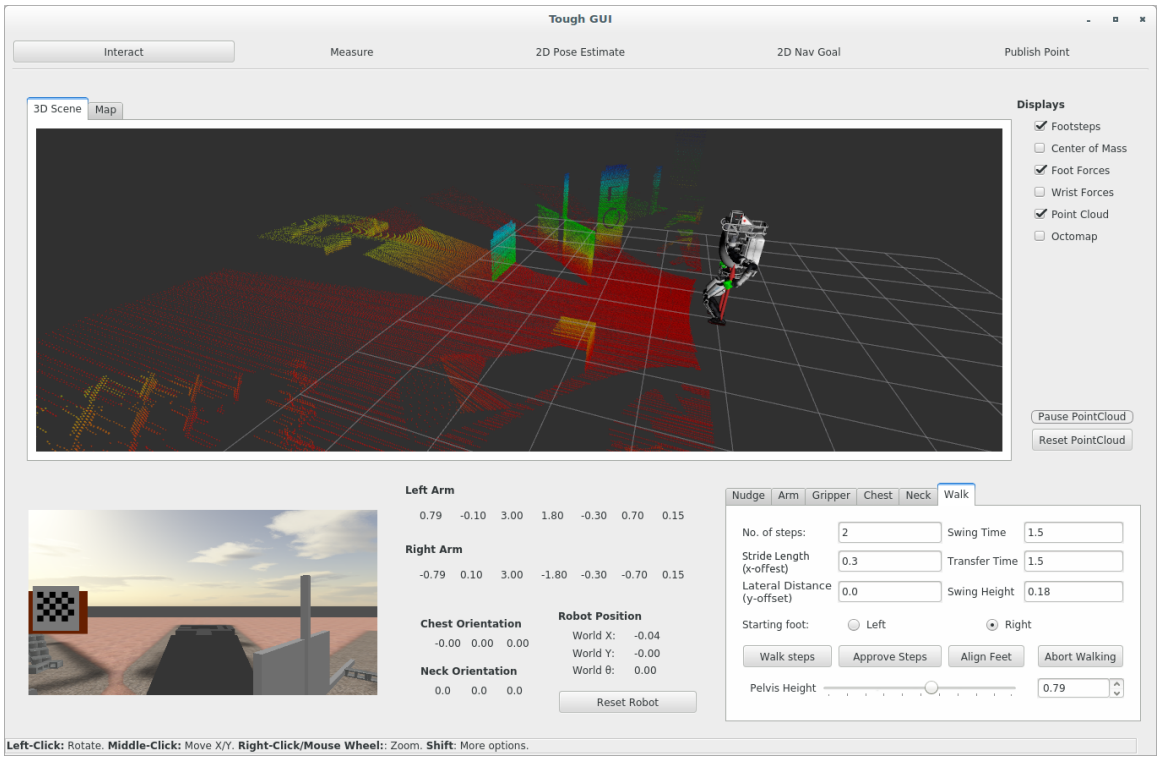


Figure 4.5: TOUGH GUI connects to the robot or simulator to show a 3D rendered display of the robot-state and pointcloud. It also has several widgets to control different body parts.

planner generates a set of footsteps for the robot to follow. This planner takes care of robot specific constraints, like the kinematic limits of the robot. For example, Atlas robot cannot place its foot such that the toe is pointing inwards; the footstep planner is configured to consider that. Once a plan is ready, those footsteps are seen in GUI and the user must click on "Approve Steps" button for the robot to start walking.

4.3 Getting Started

This section provides code snippets to perform a pick and place task using TOUGH APIs. The task is to detect an object, navigate to it, pick it up, navigate to delivery location, and place the object.

4.3.1 Initialization

First step is to initialize all the required objects as shown in Listing 4.3.

```
/* Initialize a ros node */
ros::init(argc, argv, "example_node");
ros::NodeHandle nh;
ros::AsyncSpinner spinner(1);
spinner.start();

/* Get pointers to the objects of RobotDescription and
   RobotStateInformer */
RobotStateInformer* rsi;
rsi=RobotStateInformer::getRobotStateInformer(nh);

RobotDescription* rd;
rd =RobotDescription::getRobotDescription(nh);

/* Controllers */
ArmControlInterface armCont(nh);
HeadControlInterface headCont(nh);
WholebodyControlInterface wbCont(nh);
GripperControlInterface gripCont(nh);

RobotWalker walkCont(nh);

/* Planner */
TaskspacePlanner planner(nh);

/* Object Detection*/
```

```
ArucoDetector detector(nh);
```

Listing 4.3: Initialization

4.3.2 Object Detection

The next step is to detect the object of interest. Assuming the object has an ArUco marker with id 15, we can check the presence of the object in the field of view using the code snippet below

```
/* Detector updates an existing map of string and PoseStampedPtr.
    So create an empty map first.
*/
std::map<std::string,
  geometry_msgs::PoseStampedPtr> detected_objects;

/* Fetch all the detected objects using detector
*/
detector.getDetectedObjects(detected_objects);

// Check if the object is detected, print its pose
geometry_msgs::PoseStampedPtr objectPose;
if (detected_objects.count("15") > 0 )
{
  objectPose= detected_objects["15"];
 std::cout<< "Position of the object is :"
        << objectPose.pose.position.x
        << objectPose.pose.position.y
        << objectPose.pose.position.z;
```

```
// pose variable also contains the orientation and fixed frame
   information
}
```

Listing 4.4: Object Detection

In case the object is not in visible range, HeadControlInterface can be used to turn the vision sensor as shown in code snippet below

```
/* define the required roll pitch and yaw in radians */
float roll = 0;
float pitch = 0;
float yaw = M_PI_4 ;

headCont.moveHead(roll, pitch, yaw);
```

Listing 4.5: Move Head to Look Around

4.3.3 Navigation

Assuming a walking goal is computed based on the detected object pose, such that the robot stands near the object, we can plan footsteps and make the robot walk to the goal position using the following code snippet:

```
// goal is geometry_msgs::Pose2D object computed based on object
   pose

bool blockingCall = true;
walkCont.walkToGoal(goal, blockingCall);
```

Listing 4.6: Navigate to a Goal Pose

4.3.4 Motion Planning and Manipulation

Let us assume that the object is on the right side of the robot and it is in reachable space of right arm after the robot completed its walk. The pose of the object is stored in *objectPose* variable. We can then compute a trajectory to go to the desired pose. Assuming the desired pose is same as objectPose. In a real application, it can be different and computed based on object shape and placement of the marker on the object.

```
// create a blank trajectory message to be populated with values
moveit_msgs::RobotTrajectory trajMessage;

/* Available planning groups are
1. TOUGH_COMMON_NAMES::RIGHT_ARM_10DOF_GROUP;
2. TOUGH_COMMON_NAMES::RIGHT_ARM_7DOF_GROUP;
3. TOUGH_COMMON_NAMES::LEFT_ARM_10DOF_GROUP;
4. TOUGH_COMMON_NAMES::LEFT_ARM_7DOF_GROUP;
*/
std::string planner_group = TOUGH_COMMON_NAMES::
   RIGHT_ARM_10DOF_GROUP;

// Allowable tolerances for the planner can be modified before
   planning
if(planner.getTrajectory(objectPose, planner_group, trajMessage))
{
    // execute the computed trajectory
    wbCont.executeTrajectory(trajMessage);
}
else
{
   // planning failed
```

```
}
```

Listing 4.7: Trajectory Planning and Execution

Once the trajectory is executed, we can confirm that the end effector pose has reached the required location.

```
// fetch right hand end effector frame name
std::string rEndEffFrame = rd->getRightEEFrame();

// lets assume we want to query the pose wrt pelvis frame.
std::string pelvisFrame = rd->getPelvisFrame();

//create a pose object to store the pose of end effector
geometry_msgs::Pose rEndEffPose;

rsi->getCurrentPose(rEndEffFrame, rEndEffPose, pelvisFrame);

// now eEndEffPose can be compared with objectPose to confirm
   that the end effector has reached the desired pose.
```

Listing 4.8: Query Pose of End-Effector

Grasping can be performed using gripper controller as shown below

```
// closing right gripper
gripCont.closeGripper(RobotSide::RIGHT);

//opening right gripper
gripCont.openGripper(RobotSide::RIGHT);
```

Listing 4.9: Operating Grippers

Placing of the object can be performed similarly using object detection to locate the pose for placing object, then navigating to it, planning hand motion to place the object, and

placing it.

4.4 Performance Comparison

The primary purpose of TOUGH is to act as an abstraction layer between the low-level controller implementation and the user input. It allows achieving more by writing less code. For most algorithms, the APIs depend on other open-source libraries, as mentioned above. It does not add any overhead on time complexity to the existing algorithms. Hence, it is apt to use "Lines of Code per Command" to compare the programs written using TOUGH APIs with direct ROS-based nodes. Figure 4.6 provides a comparison of executables using TOUGH APIs and basic ROS nodes for performing the same task. The code used for comparison is available in tough_examples package. The examples are elementary, and yet we see code reduction by ∼75%. When compared with more significant tasks like writing high-level controllers or executing required motion trajectories, the overall code reduction would reach more than 80% as seen in a more complex example like "Walk N Steps" in the chart.

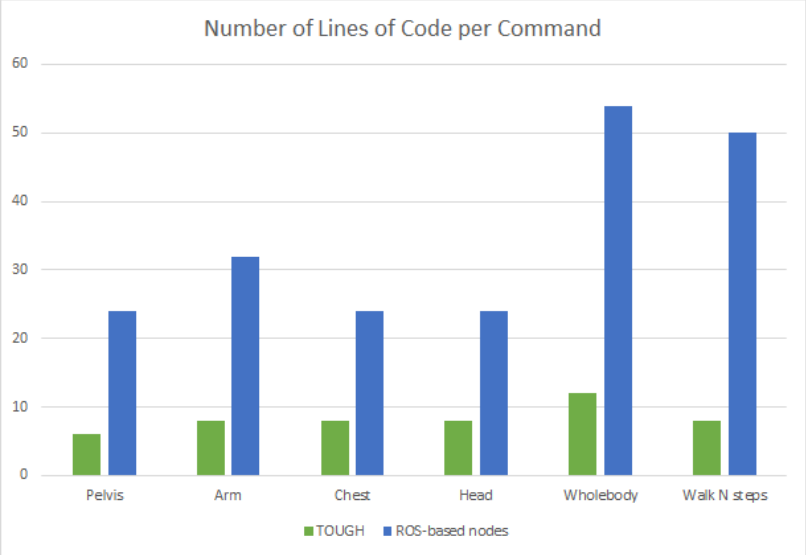


Figure 4.6: TOUGH APIs vs ROS nodes

4.5 Docker Container Images

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application [38]. Docker images for Atlas and Valkyrie are created which help in quick configuration and set up of the entire system. These images are self-contained and run the simulator in headless mode, i.e., without graphical front end. User can connect to the simulation server from the local machine and visualize the robot with data from its sensors or send commands to the robot. Using docker images allows us to separate the entire simulation with controllers from the TOUGH APIs or other code written using those APIs. Docker container can be replaced by the robot to execute the code on the robot.

Generally, docker images are independent of the operating system on which they are running. However, due to specific requirements of controllers, ROS, and Gazebo simulator, the current version only supports the use of docker images on Ubuntu 16.04. The host computer must also have a dedicated graphics card for simulation and processing of vision sensor data. Following docker images are made available on github account of WPI Humanoid Robotics Lab (WHRL). https://github.com/WPI-Humanoid-Robotics-Lab

- Repository name : drcsim_docker
  - provides Atlas simulation in Gazebo.
- Repository name : srcsim_docker
  - provides Valkyrie simulation in Gazebo.

The repositories are named based on the original simulation software names which were created for DARPA Robotics Challenge (DRC) and NASA Space Robotics Challenge (SRC) by Open Robotics, Inc. The original simulation software has been modified to work on Ubuntu 16.04 and ROS kinetic. The controllers used in the original simulation software for atlas has been replaced with the momentum-based controllers[32].

4.6 Use Cases

4.6.1 NASA Space Robotics Challenge

NASA Space Robotics Challenge, organized in 2016-17, focused on developing software that would allow increasing autonomy of humanoid robots. TOUGH APIs are an outcome of that competition and provides a complete suite for developing autonomous solutions for known tasks. The premise of the competition was, some time in future a sand storm on mars has misaligned communication dish, disconnected power, and caused a leak in the habitat. A Valkyrie R5 robot that is present onsite should fix the alignment of the communication dish, fix the solar array by deploying a solar panel and plugging the power cable into it, finding and fixing an air leak inside a habitat. The restrictions on bandwidth and latency made it extremely difficult to control the robot manually. Jagtap et al. [26] designed extended state machine to complete two of these tasks autonomously. The state machine used TOUGH APIs which were specific to Valkyrie R5 at that time and have been modified and tested to work on Atlas robot since.

4.6.2 Humanoid Robotics Course

For the very first offering of a graduate-level special topic course on Humanoid Robotics at Worcester Polytechnic Institute, a virtual server was used, where all the students could log in and use atlas simulation for assignments and course projects.

The virtual system is set up in a unique way to allow multiple users to access the system simultaneously. ROS uses random ports for different ROS executables, known as nodes, and all these nodes communicate with a single roscore. For multiple users, we needed multiple roscores configured. Moreover, the robot controllers use a fixed set of ports for real-time communication. Docker images explained in the previous section are used to avoid crosstalk between nodes of different users. Each user has his docker container with a unique IP address. This set up allows running roscore and controllers on ports that are user-specific and present in their docker containers. The complete setup is shown in Figure 4.7. Any command that needs graphics driver is run using Virtual GL. Such commands are shown with green blocks in the figure. Users can now execute their nodes by setting ROS

related environment variables to talk to the roscore running inside the docker container. Running the docker container and setting correct environment variables requires knowledge of docker and running various commands. To simply user experience, intuitive aliases for the commands and scripts are provided. The user can thus stay oblivious of the gritty details and yet efficiently use the simulation using these aliases.

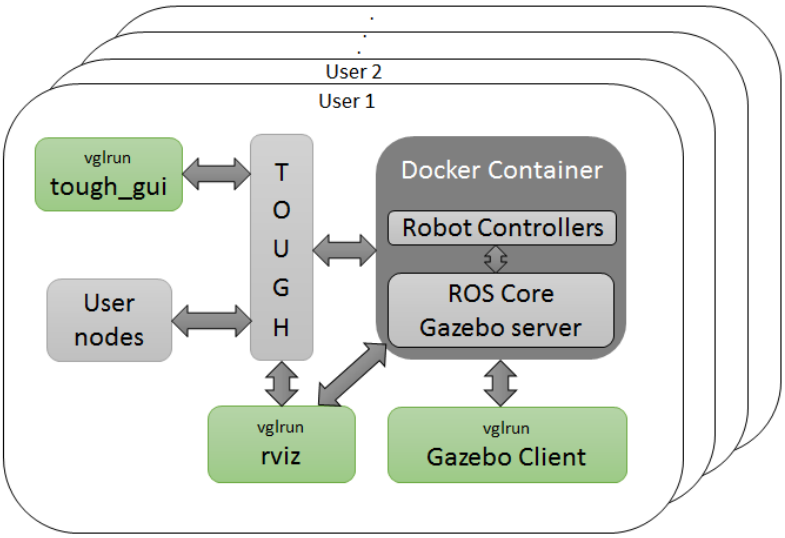


Figure 4.7: Multiuser Setup for Humanoid Robotics Course. Each user session is represented by the encapsulating rectangle. Every session has a Docker container that runs the robot controllers and gazebo simulator. Green boxes are commands that need graphics card access via Virtual GL for visualization.

4.6.3 Research in Humanoid Robotics

In WPI Humanoid Robotics Lab, TOUGH APIs are used for research projects related to perception, manipulation, motion planning, and locomotion. Use of TOUGH eliminates the inter-dependencies of these projects. Though in real-world, manipulation or motion planning is dependent on perception, it can be entirely ignored by using registered point-cloud provided by TOUGH for collision avoidance or ArUco markers for pose detection. Similarly, perception projects can test out their algorithms when the robot is walking or performing other motions and stay assured that the robot will be stable. There are minor differences in simulation compared to the actual robot, and the user should be wary of those.

However, these differences are due to simulators in general and not specific to TOUGH. For example, the robot can walk much faster and perform faster motions in simulations, but those trajectories had to be slowed down when executing on a real robot.

Chapter 5

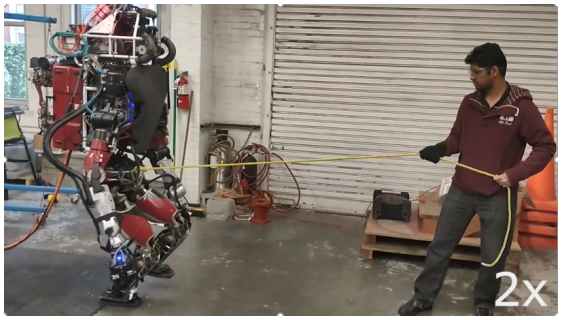
Experiments

The experiments are split into two sections. In the first section, the robot reacts to external forces by stepping in the direction of the force. The external force is applied by pulling the robot with a rope or pushing the robot with a wooden stud. The second section uses virtual force instead of pulling or pushing the robot. This method allows an exact amount of force to be applied to the robot. Accompanying videos for these experiments are available at https://youtu.be/DxSJrTB3cAQ and https://youtu.be/ilbqVM9fLwI.

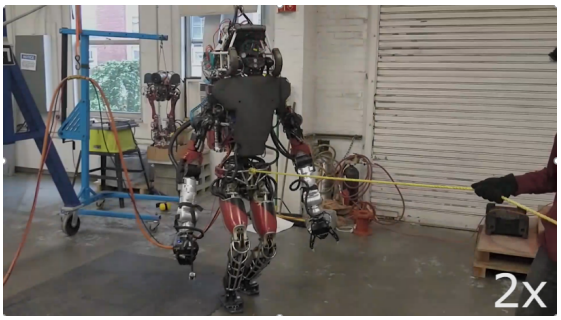
5.1 Reacting to External Forces by Stepping

When external forces are applied to the robot, these forces are estimated using the measurements of joint torque sensors, foot force sensors, and the Inertial Measurement Unit (IMU) sensor. The state estimator implementation from [55] was used in the experiments to adjust ground reaction forces based on the applied force. In the first experiment, the robot reads the ground reaction forces at a rate of 250Hz and steps in the direction of applied force. A rope is tied to the robot and pulled to exert a pulling force. Figure 5.1 shows the robot stepping in the direction in which the rope is pulled. As the force is manually exerted, it was not possible to precisely measure the applied force.

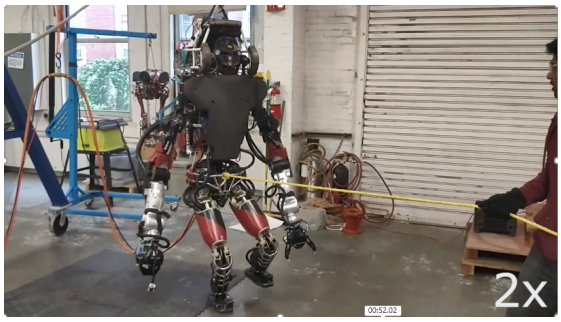
The same controller works as expected when the robot is pushed instead of pulled. This is seen in Figure 5.2. When the robot is pushed back with a wooden stud, it steps back to balance the forces. The applied force can be any direction, and the controller computes the



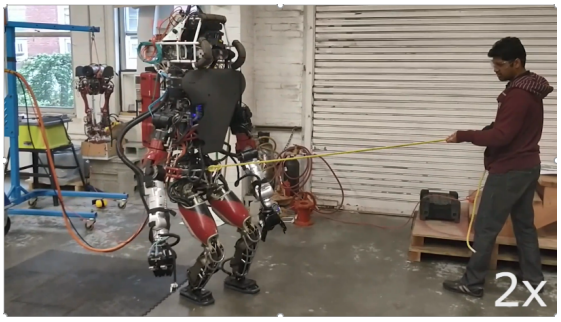
(a) Pulling straight forward



(b) Pulling at an angle



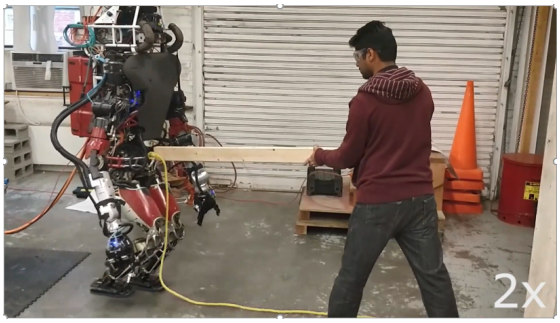
(c) Pulling at an angle



(d) Pulling in a different direction when robot is walking

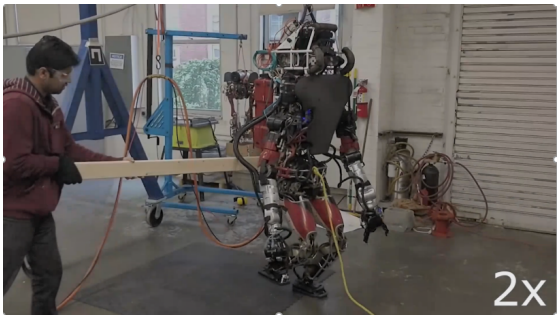
Figure 5.1: Atlas reacting to the pulling force by walking in the direction of applied force

required footstep online.

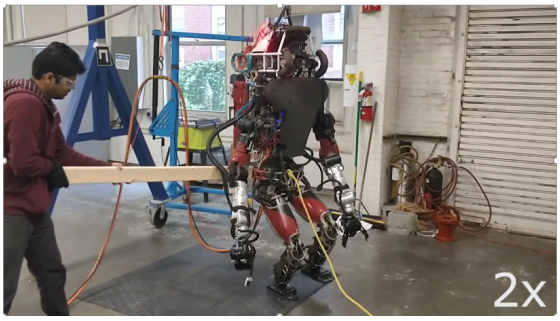


(a) Pushing backwards

(b) Pushing backwards



(c) Pushing sideways



(d) Pushing sideways

Figure 5.2: Atlas reacting to the pushing force by walking in the direction of applied force

5.2 Walking with Virtual Force

The reactive controller explained in Section 5.1 allows the robot to walk in the direction of applied force; however, the magnitude and direction of the applied force cannot be precisely regulated. To allow better control over where the robot moves and how long or short the step is, we use virtual force explained in Chapter 3. This virtual force can be applied precisely in the desired direction. When the robot is in double support such that both the feet are aligned and the robot is standing, the step length varies linearly with force in the forward direction as shown in Figure 5.3. The robot does not take any step if the desired CMP lies inside the support polygon. This is the lower limit of the required virtual force. When the desired CMP is outside the intersection of leg workspace and the capture

region, the step length is maximum. This is the upper limit on the virtual force. As seen in Figure 5.3, the virtual force should be more than 140N for the robot to start moving, and if it exceeds 660N, the step length stays constant at 0.45m. When the robot is walking, based on external forces on each of the legs, the lower bound and the upper bound of the virtual force change as the size of support polygon and capture region changes during each phase of the walk.

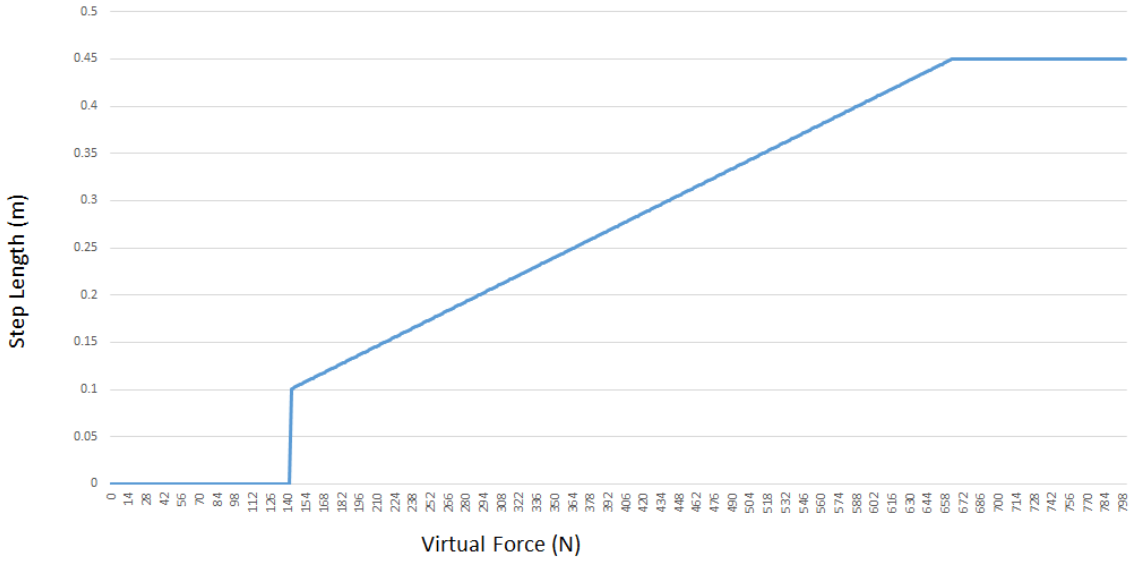


Figure 5.3: Virtual Force vs Step Length in double-support standing pose of Atlas

5.2.1 Simulation

In this experiment, a constant virtual force is applied to the robot in simulation to walk 20 steps forward and 20 steps backward. The workspace of each leg is kept at the maximum allowable per the joint limits, which allows aggressive stepping resulting in instability of the robot. The Virtual force is increased from 300N to 650N in steps of 50N. Figure 5.4 shows the plot of virtual force vs. step length for the forwards steps in every 50N increment of virtual force. The step length variation is due to the ground reaction forces acting on the robot while walking. When a computed step is too long or too short, the next step is adjusted based on the external forces acting on the robot. As seen in the plot, the variation

in step length is higher for the virtual force values of 600N and 650N. When a higher force is applied to the robot, the robot tries to take longer steps that are feasible, and as the workspace is maximum possible, the robot opts for steps that are at the border of the feasible region. This results in a less stable step; however, the next step is shortened to balance out the effects of the current step.

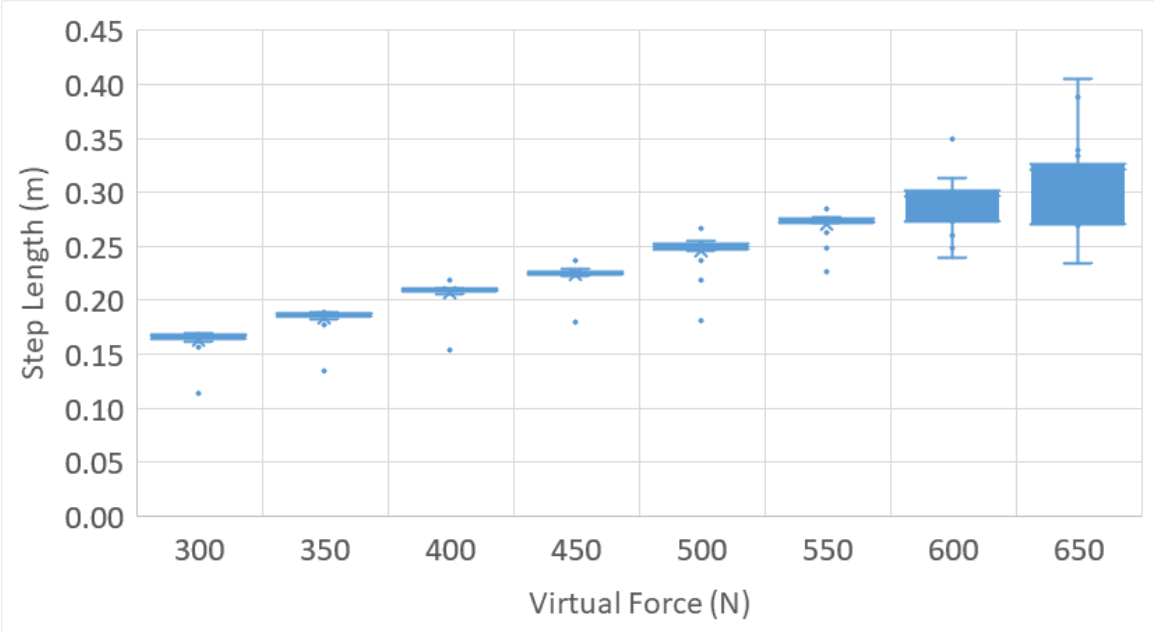


Figure 5.4: Virtual Force vs. Step Length

5.2.2 Atlas

On Atlas hardware, a constant virtual force of 575N is applied. Figure 5.5 shows the capture point trajectories for the 5 planned steps of approximately 0.3m in length. In this figure, capture points are of two types, end of step capture points shown as black diamonds and intermediate capture points shown as blue diamonds. The curves in the figure are capture point trajectories, and the yellow triangles are points when the next step is planned. Vertical dotted lines separate the Single Support (SS) and Double Support (DS) phases.

In the initial state, the robot is standing with both feet aligned. The capture point is at

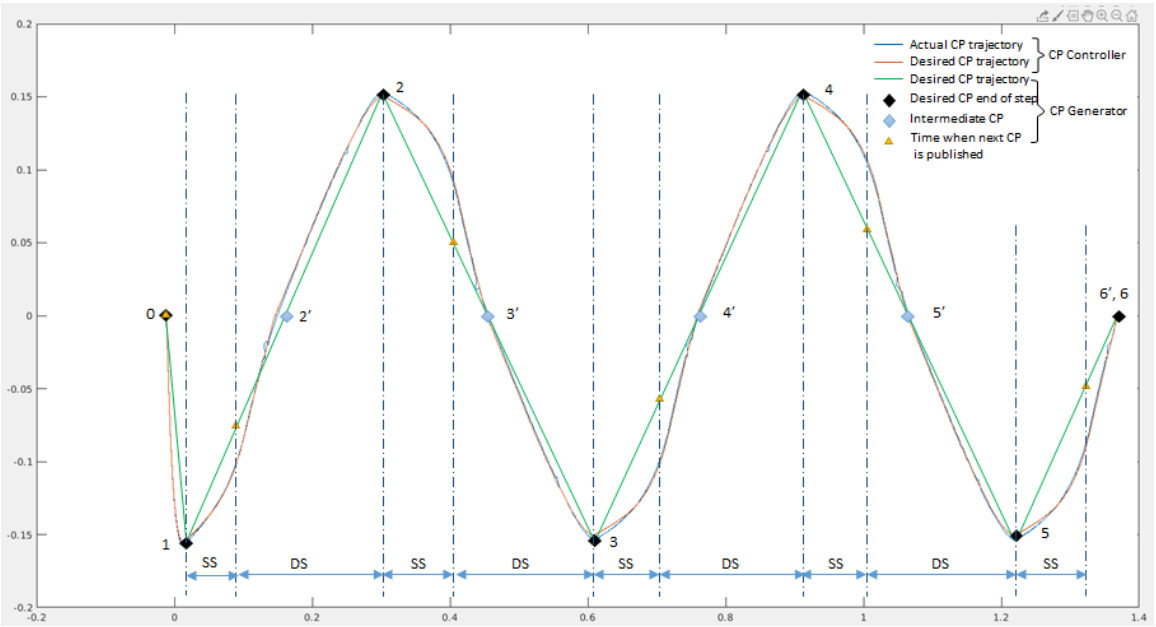


Figure 5.5: Desired and actual Capture Point (CP) trajectories by the low-level controller and the desired capture point generator when a constantvirtual force of 575N is applied. DS = double support, SS = single support.

the center of the support polygon, which approximately coincides with the origin, shown as capture point 0. The first step generates two capture points, one for transferring the weight of the robot to the right foot at capture point 1 (black diamond) and another intermediate capture point at 2' shown in blue. Point 2' is the center of support polygon when the robot places its left foot on capture point 2. The capture point 2 is also referred to as the End of Step capture point. At every yellow triangle, two more capture points are computed, one for transferring the weight on support leg at the previous end of step capture point and the other intermediate point occurring at the center of support polygon due to the current step sent to the robot. For example, at the yellow triangle after capture point 1, the intermediate point 2' moves to capture point 2 and a new intermediate point 3' is created. If no force is acting on the robot when the yellow triangle on the trajectory is reached, the intermediate capture point becomes the last capture point, and the robot stops walking with both feet touching the ground. This is seen at the last point where 6 and 6' coincides. The controller creates a trajectory for instantaneous capture point and follows that trajectory to achieve the desired capture point. The trajectories of the actual and the desired capture points by

the controller are shown in blue and orange lines. The ones generated by the capture point generator are shown in green.

The next step is planned as soon as the robot enters the double support phase, and the virtual force is still acting on the robot. This is shown by yellow triangle in Figure 5.5. Delay in planning the next step reduces the horizontal components of the ground reaction forces. This results in loss of momentum as the robot tries to stop after reaching the double support phase causing jerks by sudden deceleration and acceleration of CoM. This is avoided by planning the next step as soon as the robot enters the double support phase.

Chapter 6

Conclusions

Use of virtual force for generating the next footstep and controlling the robot using the same dynamics used for footstep generation provides a tight coupling between walking pattern generator and the walking controller. This is possible because of the capture regions and end of step capture point control. Experiments show that the robot can plan only one step at a time and execute multiple steps in the desired direction. The first step takes typically an additional 0.3 to 0.5 seconds as the robot has to transfer its weight to one of the legs before starting the computed step. All the footstep parameters are computed based on the direction and magnitude of the applied virtual force. This thesis provides the implementation on flat ground, which is tested in both the simulation and on real hardware. Simulation results are shown for Atlas and Valkyrie robots, whereas the hardware experiments are done on the Atlas robot.

Simpler Input for Teleoperation: Virtual force is the only input required to make the robot walk. It is a 2D vector compared to a series of footsteps required by most other methods. For teleoperation, the operator can command the robot to move in the desired direction using a virtual force as an input to the controller, and the robot starts walking immediately.

Always Execute a Feasible Step: In this approach, we computed a safe region to step before sending the walk command to the robot. As the safe step region is always a subset of capture region, the robot can stop after completion of any given step. The planned

step is still feasible as both kinematic and dynamic constraints are used while computing the safe step region. If a step is not feasible, the robot does not walk and comes to a complete stop at the end of the transfer phase in walking. The controller can resist external forces to keep the robot balanced because the step parameters are generated outside of the controller. For example, if the robot is pushed moderately, the robot resists the push to maintain its balance, but if a virtual force of similar magnitude is applied, it starts walking in the desired direction.

Inherent Immunity to External Forces: Next step is generated based on the forces that are acting on the robot along with the virtual force. If there is any additional force acting on the robot at any given state, it is taken into account before planning the next step. The low-level controller handles the unexpected forces during the step execution, whereas the impact of those forces is taken care of by the footstep generator while planning the next step. This makes the robot immune to external forces.

6.1 Future Work

6.1.1 Integrate perception

The method presented in this thesis can be integrated with perception, to limit the safe region to step based on detected planes. In order to achieve this, we can compute an intersection of the leg workspace and the capture region as seen in 3.3.2 with the planes detected from the perception module. For low-level control of the robot in those situations, Divergent Component of Motion (DCM) [12] would be required instead of the capture points as the steps would be planned in a 3D space. This integration would allow walking on slopes or different leveled planes like stairs. Integrating perception relies heavily on availability of enough sensor data to detect planes when the robot is already in motion.

6.1.2 Virtual Wrench

The concept of “Virtual Force” can be extended to “Virtual Torque”. In the current implementation, the robot cannot turn in place with any orientation or magnitude of the virtual force. To overcome this behavior, a combination of force and torque can be expressed

as a "Virtual Wrench". The desired footstep position and orientation can then be computed based on the effect of the virtual wrench.

Bibliography

[1] Aaron D. Ames, Eric A. Cousineau, and Matthew J. Powell, *Dynamically stable bipedal robotic walking with nao via human-inspired hybrid zero dynamics*, Proceedings of the 15th ACM International Conference on Hybrid Systems: Computation and Control (New York, NY, USA), HSCC '12, ACM, 2012, pp. 135–144.

[2] C. G. Atkeson, B. P. W. Babu, N. Banerjee, D. Berenson, C. P. Bove, X. Cui, M. DeDonato, R. Du, S. Feng, P. Franklin, M. Gennert, J. P. Graff, P. He, A. Jaeger, J. Kim, K. Knoedler, L. Li, C. Liu, X. Long, T. Padir, F. Polido, G. G. Tighe, and X. Xinjilefu, *No falls, no resets: Reliable humanoid behavior in the darpa robotics challenge*, 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), Nov 2015, pp. 623–630.

[3] Patrick Beeson and Barrett Ames, *Trac-ik: An open-source library for improved solving of generic inverse kinematics*, 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), IEEE, 2015, pp. 928–935.

[4] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard, *The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives*, IEEE International Symposium on System Integrations (SII), 2019.

[5] Joel Chestnutt, Manfred Lau, German Cheung, James Kuffner, Jessica Hodgins, and Takeo Kanade, *Footstep planning for the honda asimo humanoid*, Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, IEEE, 2005, pp. 629–634.

[6] Sachin Chitta, Ioan Sucan, and Steve Cousins, *Moveit![ros topics]*, IEEE Robotics & Automation Magazine **19** (2012), no. 1, 18–19.

[7] Michael J. Coleman and Andy Ruina, *An uncontrolled walking toy that cannot stand still*, Phys. Rev. Lett. **80** (1998), 3658–3661.

[8] Steve Collins, Andy Ruina, Russ Tedrake, and Martijn Wisse, *Efficient bipedal robots based on passive-dynamic walkers*, Science **307** (2005), no. 5712, 1082–1085.

[9] Steven H Collins, Martijn Wisse, and Andy Ruina, *A three-dimensional passive-dynamic walking robot with two legs and knees*, The International Journal of Robotics Research **20** (2001), no. 7, 607–615.

[10] Mathew DeDonato, Felipe Polido, Kevin Knoedler, Benzun PW Babu, Nandan Banerjee, Christoper P Bove, Xiongyi Cui, Ruixiang Du, Perry Franklin, Joshua P Graff, et al., *Team wpi-cmu: Achieving reliable humanoid behavior in the darpa robotics challenge*, Journal of Field Robotics **34** (2017), no. 2, 381–399.

[11] J. Englsberger, G. Mesesan, and C. Ott, *Smooth trajectory generation and push-recovery based on divergent component of motion*, 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sep. 2017, pp. 4560–4567.

[12] Johannes Englsberger, Christian Ott, and Alin Albu-Schäffer, *Three-dimensional bipedal walking control based on divergent component of motion*, IEEE Transactions on Robotics **31** (2015), no. 2, 355–368.

[13] Johannes Englsberger, Christian Ott, Maximo A. Roa, Alin Albu-Schäffer, and Gerhard Hirzinger, *Bipedal walking control based on capture point dynamics*, IEEE International Conference on Intelligent Robots and Systems (2011), 4420–4427.

[14] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson, *Optimization based full body control for the atlas robot*, 2014 IEEE-RAS International Conference on Humanoid Robots, Nov 2014, pp. 120–127.

[15] Siyuan Feng, *Online hierarchical optimization for humanoid control*, (2016).

[16] Chien-Liang Fok, Gwendolyn Johnson, John D Yamokoski, Aloysius Mok, and Luis Sentis, *Controlit!a software framework for whole-body operational space control*, International Journal of Humanoid Robotics **13** (2016), no. 01, 1550040.

[17] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas, and Manuel Jesús Marín-Jiménez, *Automatic generation and detection of highly reliable fiducial markers under occlusion*, Pattern Recognition **47** (2014), no. 6, 2280–2292.

[18] R. J. Griffin, G. Wiedebach, S. Bertrand, A. Leonessa, and J. Pratt, *Straight-leg walking through underconstrained whole-body control*, 2018 IEEE International Conference on Robotics and Automation (ICRA), May 2018, pp. 1–5.

[19] Ayonga Hereid, Shishir Kolathaya, Mikhail S Jones, Johnathan Van Why, Jonathan W Hurst, and Aaron D Ames, *Dynamic multi-domain bipedal walking with atrias through slip based human-inspired control*, Proceedings of the 17th international conference on Hybrid systems: computation and control, ACM, 2014, pp. 263–272.

[20] Hugh Herr and Marko Popovic, *Angular momentum in human walking*, Journal of Experimental Biology **211** (2008), no. 4, 467–481.

[21] Dave Hershberger, David Gossow, and Josh Faust, *Rviz, 3d visualization tool for ros*, URL: http://wiki. ros. org/rviz (2009).

[22] Kazuo Hirai, Masato Hirose, Yuji Haikawa, and Toru Takenaka, *The development of honda humanoid robot*, Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on, vol. 2, IEEE, 1998, pp. 1321–1326.

[23] E. M. Hoffman, A. Rocchi, A. Laurenzi, and N. G. Tsagarakis, *Robot control for dummies: Insights and examples using opensot*, 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids), Nov 2017, pp. 736–741.

[24] Armin Hornung, Andrew Dornbush, Maxim Likhachev, and Maren Bennewitz, *Anytime search-based footstep planning with suboptimality bounds*, Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on, IEEE, 2012, pp. 674–679.

[25] Qiang Huang, Shuuji Kajita, Noriho Koyachi, Kenji Kaneko, Kazuhito Yokoi, Hirohiko Arai, Kiyoshi Komoriya, and Kazuo Tanie, *A high stability, smooth walking pattern for a biped robot*, Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on, vol. 1, IEEE, 1999, pp. 65–71.

[26] V. Jagtap, S. Agarwal, S. Nirmal, S. Kejriwal, and M. A. Gennert, *Extended state machines for robust robot performance in complex tasks*, 2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids), Nov 2018, pp. 264–270.

[27] Jong Hyeon Park and Hyun Chul Cho, *An online trajectory modifier for the base link of biped robots to enhance locomotion stability*, Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), vol. 4, April 2000, pp. 3353–3358 vol.4.

[28] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kiyoshi Fujiwara, Kensuke Harada, Kazuhito Yokoi, and Hirohisa Hirukawa, *Biped walking pattern generation by using preview control of zero-moment point*, ICRA, vol. 3, 2003, pp. 1620–1626.

[29] Shuuji Kajita, Fumio Kanehiro, Kenji Kaneko, Kazuhito Yokoi, and Hirohisa Hirukawa, *The 3d linear inverted pendulum mode: A simple modeling for a biped walking pattern generation*, Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180), vol. 1, IEEE, 2001, pp. 239–246.

[30] Shuuji Kajita and Kazuo Tani, *Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode*, Proceedings. 1991 IEEE International Conference on Robotics and Automation, IEEE, 1991, pp. 1405–1411.

[31] Ichiro Kato, *Development of wabot 1*, Biomechanism **2** (1973), 173–214.

[32] Twan Koolen, Sylvain Bertrand, Gray Thomas, Tomas de Boer, Tingfan Wu, Jesper Smith, Johannes Englsberger, and Jerry Pratt, *Design of a momentum-based control*

framework and application to the humanoid robot atlas, International Journal of Humanoid Robotics **13** (2016), no. 01, 1650007.

[33] Twan Koolen, Tomas De Boer, John Rebula, Ambarish Goswami, and Jerry Pratt, *Capturability-based analysis and control of legged locomotion, Part 1: Theory and application to three simple gait models*, International Journal of Robotics Research **31** (2012), no. 9, 1094–1113.

[34] James J Kuffner Jr, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue, *Footstep planning among obstacles for biped robots.*, IROS, 2001, pp. 500–505.

[35] Sung-Hee Lee and Ambarish Goswami, *Reaction mass pendulum (rmp): An explicit model for centroidal angular momentum of humanoid robots*, Robotics and Automation, 2007 IEEE International Conference on, IEEE, 2007, pp. 4667–4672.

[36] Sean Mason, Nicholas Rotella, Stefan Schaal, and Ludovic Righetti, *An mpc walking framework with external contact forces*, 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 1785–1790.

[37] Tad McGeer et al., *Passive dynamic walking*, I. J. Robotic Res. **9** (1990), no. 2, 62–82.

[38] Dirk Merkel, *Docker: Lightweight linux containers for consistent development and deployment*, Linux J. **2014** (2014), no. 239.

[39] M. Morisawa, K. Harada, S. Kajita, K. Kaneko, J. Sola, E. Yoshida, N. Mansard, K. Yokoi, and J. Laumond, *Reactive stepping to prevent falling for humanoids*, 2009 9th IEEE-RAS International Conference on Humanoid Robots, Dec 2009, pp. 528–534.

[40] Jerzy Mrozowski, Jan Awrejcewicz, and Piotr Bamberski, *Analysis of stability of the human gait*, Journal of theoretical and applied mechanics **45** (2007), no. 1, 91–98.

[41] Gabe Nelson, Aaron Saunders, Neil Neville, Ben Swilling, Joe Bondaryk, Devin Billings, Chris Lee, Robert Playter, and Marc Raibert, *Petman: A humanoid robot for testing chemical protective clothing*, Journal of the Robotics Society of Japan **30** (2012), no. 4, 372–377.

[42] K Nishtwaki, Kenichirou Nagasaka, Masayuki Inaba, and Hirochika Inoue, *Generation of reactive stepping motion for a humanoid by dynamically stable mixture of predesigned motions*, Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on, vol. 6, IEEE, 1999, pp. 902–907.

[43] Francesco Nori, Silvio Traversaro, Jorhabib Eljaik, Francesco Romano, Andrea Del Prete, and Daniele Pucci, *icub whole-body control through force regulation on rigid noncoplanar contacts*, Frontiers in Robotics and AI **2** (2015), no. 6.

[44] Marko B Popovic, Ambarish Goswami, and Hugh Herr, *Ground reference points in legged locomotion: Definitions, biological trajectories and control implications*, The International Journal of Robotics Research **24** (2005), no. 12, 1013–1032.

[45] Ioannis Poulakakis and Jessy W Grizzle, *The spring loaded inverted pendulum as the hybrid zero dynamics of an asymmetric hopper*, IEEE Transactions on Automatic Control **54** (2009), no. 8, 1779–1793.

[46] Gill Pratt and Justin Manzo, *The darpa robotics challenge [competitions]*, IEEE Robotics & Automation Magazine **20** (2013), no. 2, 10–12.

[47] Jerry Pratt, John Carff, Sergey Drakunov, and Ambarish Goswami, *Capture point: A step toward humanoid push recovery*, Proceedings of the 2006 6th IEEE-RAS International Conference on Humanoid Robots, HUMANOIDS (2006), 200–207.

[48] Qiang Huang, K. Kaneko, K. Yokoi, S. Kajita, T. Kotoku, N. Koyachi, H. Arai, N. Imamura, K. Komoriya, and K. Tanie, *Balance control of a piped robot combining off-line pattern with real-time modification*, Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065), vol. 4, April 2000, pp. 3346–3352 vol.4.

[49] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng, *Ros: an open-source robot operating system*, ICRA workshop on open source software, vol. 3, Kobe, 2009, p. 5.

[50] Philippe Sardain and Guy Bessonnet, *Forces acting on a biped robot. center of pressure-zero moment point*, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans **34** (2004), no. 5, 630–637.

[51] Nicola Scianca, Marco Cognetti, Daniele De Simone, Leonardo Lanari, and Giuseppe Oriolo, *Intrinsically stable mpc for humanoid gait generation*, 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids), IEEE, 2016, pp. 601–606.

[52] Benjamin Stephens, *Humanoid push recovery*, Humanoid Robots, 2007 7th IEEE-RAS International Conference on, IEEE, 2007, pp. 589–595.

[53] Benjamin J Stephens and Christopher G Atkeson, *Push recovery by stepping for humanoid robots with force controlled joints*, 2010 10th IEEE-RAS International Conference on Humanoid Robots, IEEE, 2010, pp. 52–59.

[54] SylvainBertrand, Duncan Calvert, Jerry Pratt, Georg Wiedebach, Robert Griffin, btshrewsbury, Jesper Smith, Twan Koolen, Nathan Mertins, Douglas Stephen, Stephen McCrory, neyssette, mahopkins, cmschmid, jcarff, Kyle Cesare, aprvs, egalbally, mj fl, Peter Abeles, edilee, pneuhaus, Antnio Bento Filho, Will Rifenburgh, Nathaniel Choe, Karel Petrnek, stespenc83, OlgerSiebinga, Shlok Agarwal, and lbunch, *ihmcrobotics/ihmc-open-robotics-software: 0.11 Release Notes*, January 2018.

[55] SylvainBertrand, Jerry Pratt, Duncan Calvert, Georg Wiedebach, Jesper Smith, Twan Koolen, Robert Griffin, Nathan Mertins, Douglas Stephen, neyssette, Stephen McCrory, mahopkins, jcarff, Kyle Cesare, egalbally, mj fl, Peter Abeles, edilee, pneuhaus, Antnio Bento Filho, Will Rifenburgh, Nathaniel Choe, Karel Petrnek, OlgerSiebinga, lbunch, Koen Krmer, Vladimir Ivan, admimou, aprvs, and turbancov2, *ihmcrobotics/ihmc-open-robotics-software: SRC Finals*, July 2017.

[56] R. Tedrake, S. Kuindersma, R. Deits, and K. Miura, *A closed-form solution for real-time zmp gait generation and feedback stabilization*, 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), Nov 2015, pp. 936–940.

[57] Russ Tedrake and the Drake Development Team, *Drake: Model-based design and verification for robotics*, 2019.

[58] Miomir Vukobratović and Branislav Borovac, *Zero-moment pointthirty five years of its life*, International journal of humanoid robotics **1** (2004), no. 01, 157–173.

[59] Eric R Westervelt, Jessy W Grizzle, and Daniel E Koditschek, *Hybrid zero dynamics of planar biped walkers*, (2003).

[60] Pierre-Brice Wieber, *Trajectory free linear model predictive control for stable walking in the presence of strong perturbations*, IEEE-RAS international conference on humanoid robots, 2006.

[61] Jin'ichi Yamaguchi, Eiji Soga, Sadatoshi Inoue, and Atsuo Takanishi, *Development of a bipedal humanoid robot-control method of whole body cooperative dynamic biped walking*, Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on, vol. 1, IEEE, 1999, pp. 368–374.

[62] Seung-kook Yun and Ambarish Goswami, *Momentum-based reactive stepping controller on level and non-level ground for humanoid robot push recovery*, Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, IEEE, 2011, pp. 3943–3950.