Chapter 10
Dynamic Balancing and Flexible Task Execution for Dynamic Bipedal Walking Machines

Andreas Hofmann

Abstract Effective use of robots in unstructured environments requires that they have sufficient autonomy and agility to execute task-level commands with temporal constraints successfully. A challenging example of such a robot is a bipedal walking machine, particularly one of humanoid form. Key features of the human morphology include a variable base of support and a high center of mass. The high center of mass supports the ability to support a high "sensor package"; when standing erect, the head can see over obstacles. The variable base of support allows both for operation in tight spaces, by keeping the feet close together, and stability against disturbances, by keeping the feet further apart to widen the support base. The feet can also be placed in specific locations when there are constraints due to challenging terrain. Thus, the human morphology supports a range of capabilities, and is important for operating in unstructured environments as humans do. A bipedal robot with human morphology should be able to walk to a particular location within a particular time, while observing foot placement constraints, and avoiding a fall, if this is physically possible. This is a challenging problem because a biped is highly nonlinear and has limited actuation due to its limited base of support. This chapter describes a novel approach to solving this problem that incorporates three key components: (1) a robust controller that is able to use angular momentum to enhance controllability beyond the limits imposed by the support base; (2) a plan specification where task requirements are expressed in a qualitative form that provides for spatial and temporal execution flexibility; and (3) a task executive that compiles the plan into a form that makes the dynamic limitations explicit, and then executes the compiled form using the robust controller.

Keywords Bipedal • Walking • Flexible plan execution • Feedback linearization • Angular momentum

A. Hofmann (✉)
Massachusetts Institute of Technology, 77 Massachusetts Ave.,
Cambridge, MA 02139, USA
e-mail: hofma@csail.mit.edu

D. Zhang, B. Wei (eds.), *Dynamic Balancing of Mechanisms and Synthesizing of Parallel Robots*, DOI 10.1007/978-3-319-17683-3_10

10.1 Introduction

Effective use of autonomous robots in unstructured human environments requires that they have sufficient autonomy to perform useful tasks independently, have sufficient size, strength, and speed to accomplish these tasks in a timely manner, and that they operate robustly and safely in the presence of disturbances. These requirements are more challenging than the ones for today's factory robots, which are stationary, work in very restricted environments, and have very limited autonomy.

A particularly challenging example of an autonomous robot in an unstructured environment is a bipedal walking machine, as shown in Fig. 10.1. An example task for such a system is to walk to a moving soccer ball and kick it, as shown in Fig. 10.2. Stepping movement must be synchronized with ball movement so that the kick

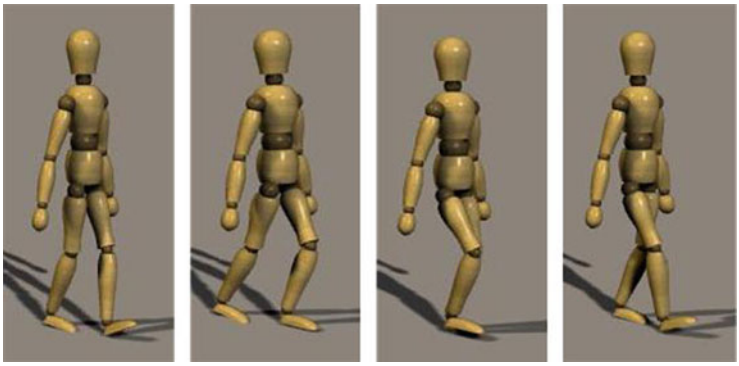


Fig. 10.1 A humanoid biped performing a walking task

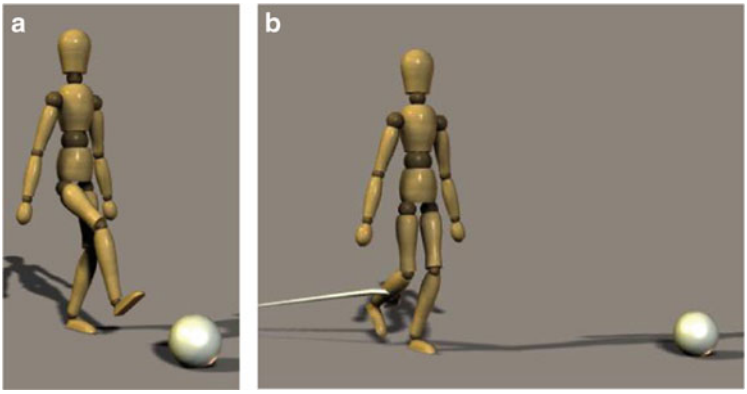


Fig. 10.2 Kicking soccer ball task, interrupted by trip disturbance. In (**a**), the goal is to kick a possibly moving soccer ball; the biped must be in an acceptable location at an acceptable time in order to perform the kick. In (**b**), the task is interrupted by a trip; the biped should try to recover, if this is physically possible

happens when the ball is close enough. More generally, such tasks require that the biped be in the right location at an acceptable time. This implies spatial and temporal constraints for such tasks. There are also important dynamic balance constraints that limit the kinds of movements the biped may make without falling down.

If the system encounters a disturbance while performing a task, it will have to compensate in some way in order to satisfy these constraints. The disturbance may cause a delay, allowing another player to kick the ball, or it may interfere with movement synchronization. For example, a trip, shown in Fig. 10.2b causes disruption of synchronization between the stepping foot, and the overall forward movement of the system's center of mass.

Another example task is walking on a constrained foot path, such as stones across a brook, or on a balance beam. As with the soccer ball example, this task has spatial, temporal, and dynamic constraints, but in this case, the spatial constraints are more stringent; the biped must reach its goal using foot placements that are precisely constrained.

Figure 10.3 shows a biped walking over blocks that constrain foot placement in a similar manner. When foot placement is constrained, the stepping pattern can't be changed arbitrarily to compensate for a disturbance. For example, if a lateral push disturbance occurs, rather than stepping the leg out to the side, other compensating techniques, such as angular movement of the body and swing leg must be used, as shown in Fig. 10.4.

In these examples, and others like them, the key challenge is to move a complex, dynamic system to the right place, at the right time, despite actuation limits, and despite disturbances. The system should be able to recover from disturbances such as slips, trips, pushes, and ground contact instability due to soft terrain, even when foot placement is constrained.

This chapter addresses the class of problems that require movement of a dynamic bipedal system according to stringent state-space and temporal requirements, despite actuation limits and disturbances. Additionally, we consider how the use of flexible link structures changes the problem and the solution. This additional

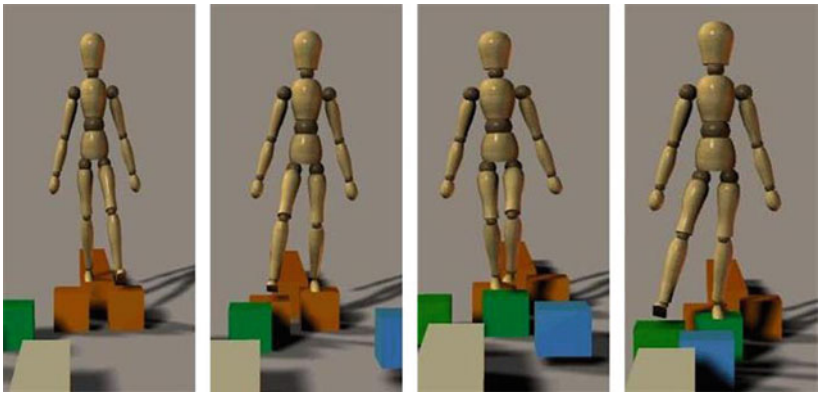


Fig. 10.3 Dynamic walking with foot placement constraints

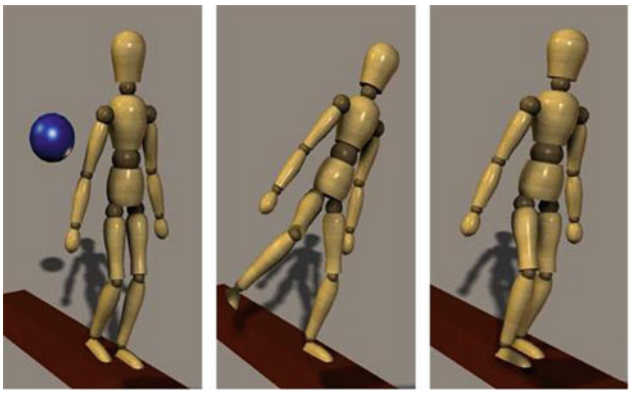


Fig. 10.4 Compensating for lateral push disturbance using angular movement of torso and swing leg

consideration is significant given the importance of using lightweight structures in mobile robots, in order to maximize energy efficiency. Use of lightweight materials for the links implies less rigidity than is the norm in industrial factory manipulators.

The remainder of this introduction describes, in more detail, the motivation for studying this class of problems, a statement of the problem being solved, and associated challenges, and an overview of the approach and innovations used to solve it.

10.1.1 Significance and Motivation

Humanoid bipedal morphologies have unique characteristics that provide significant advantages (and also some disadvantages) over quadruped or wheeled robots. Key functional features of the humanoid morphology include a variable base of support and a high center of mass. Because bipeds have only two legs, their support base is naturally constrained, allowing them to operate in environments where support base space is limited. Many human environments require this (a crowded elevator, for example). The legs can also be placed in wide stances, when space permits, enhancing stability against disturbances. The feet can also be placed in specific locations, allowing for traversal over terrain where foot placement is constrained due to obstacles. Thus, a key advantage of the humanoid morphology is that the variable base of support allows both for operation in constrained spaces, and stability against disturbances, Humanoid bipeds also have a high center of mass. This has the advantage that an elevated "sensor package" can be supported; when standing erect, the head can see over obstacles. Thus, the humanoid biped morphology supports a range of capabilities that are important for operating in unstructured environments, particularly when collaborating with humans.

Solving the problem of balance control, and task-level control of humanoid biped devices is of significant importance, in that it will be necessary for deployment of autonomous humanoid robots that can provide assistance to humans in human environments such as the home, office, construction sites, loading docks, and many others. Additionally, solution of this problem will permit the deployment of powered exoskeletons that can provide locomotion capability to disabled humans.

10.1.2 Problem Statement and Challenges

We seek to develop a robust plan execution system capable of guiding a robotic biped through a series of walking task goals, in the presence of disturbances. The system must understand commands at the task level; it must take as input a high-level specification of where it should be, and by what time, and then automatically figure out the details of how to move to accomplish these goals. It should also be able to automatically detect whether a task that it is given can be accomplished in the allotted time, and should warn the human operator when this is not the case. If a disturbance occurs during execution of the task, the system should attempt to compensate in order to avoid a fall, and should still try to complete the task on time. If this is not possible, the system should alert the user, or a higher-level control authority.

There are significant challenges to solving this problem.

First, the specification of the walking task itself should represent the true spatial and temporal constraints of the task, rather than arbitrarily setting tight, artificial constraints. For example, if the task is allowed to complete with a duration of 5–10 s, the specification should not restrict the duration any further than this. Similarly, if the goal is properly represented as a set of possible states, the specification should not restrict the goal to a single state. It is important for the specification of the task to not artificially constrain operation. This gives the control system maximum flexibility in selecting actions that maximize robustness and performance. Second, the combination of limited support base and high center of mass presents a challenge in terms of balance control in that such a system is inherently less stable (more sensitive to disturbances) than a quadrupedal or four-wheeled configuration with relatively low center of mass. The limited support base and high center of mass imply that the biped is under-actuated and has significant inertia, so future state evolution is coupled to current state through dynamics that limit acceleration. The control system must consider how current state and actions may limit achievement of future desired state. Third, a biped is a high-dimensional, highly nonlinear, tightly coupled system, so computing control actions that achieve a desired state is a challenging problem. This is complicated by the incorporation of temporal constraints, and the limits that the dynamic system imposes on temporal performance. Fourth, this type of system, because it operates in human environments, must have stringent safety requirements. Balance control is essential both for autonomous legged assistive robots and for a variety of assistive devices, including powered

exoskeletons that provide locomotion to the disabled. For such systems, preventing a fall is of paramount importance. An autonomous robot that falls may damage itself, or may hurt a human in its environment. In the case of an exoskeleton, a fall implies that the human wearer of the exoskeleton has fallen. Thus, a bipedal walking machine should avoid falling, if at all physically possible, even if it encounters a significant disturbance. If a fall is inevitable, the system should recognize this sufficiently early to alert users and surrounding humans.

Addressing these challenges requires investigation of a number of questions, including:

- How should walking task goals be expressed?
- What are the fundamental requirements for achieving these goals?
- What kinds of disturbances may occur while executing walking tasks?
- What fundamental balance strategies can bipeds use?

The following discussion introduces approaches to addressing these challenges, and answering these questions.

10.1.3 Approach and Innovations

We address these challenges with three key innovative techniques.

To address the first challenge (representation of task goals), we use a specification of state-space and temporal requirements called a *Qualitative State Plan* (QSP) [9, 10]. A QSP consists of a set of *Qualitative States*, where each Qualitative State is a region of state space in which all states have a uniform property with respect to the task at hand [23]. For a biped, qualitative states are defined by foot ground contact state. The biped may be in a double-support state, where both feet are in contact with the ground, a single-support state, where either the left or the right foot is on the ground, or a jumping state, where neither foot is on the ground.

Each qualitative state may specify valid operating regions for particular state variables. Foot placement constraints are examples of such operating region constraints. Each qualitative state may also specify goal regions that particular state variables must attain. For example, it may be a requirement that the biped center of mass be in a particular region in order for the biped to kick a soccer ball. Thus, a qualitative state is hybrid in that it is defined by continuous state regions, like allowable regions for the center of mass position, as well as by discrete state, like which feet are in contact with the ground. Transitions from one qualitative state to another are defined by events. For example, the transition from double to single support is defined by a toe-off event, which is the point where the swing foot lifts off the ground. The transition from single to double support is defined by a heel-strike event, which is the point where the swing foot touches the ground after taking a step. Events represent temporal boundaries that can be restricted by temporal constraints. Thus, a QSP permits the representation of a task's true constraints, providing maximum flexibility to the control system in selecting actions to maximize robustness and performance.

To address the third challenge (nonlinearity, high dimensionality, and tight coupling), the biped system is linearized and decoupled into a set of independent, linear systems, resulting in an abstraction of the biped that is easier to control. This is accomplished through a *Dynamic Virtual Model Controller* (DVMC) [11]. The linearization and decoupling provided by this controller allows *reaction points* on the biped to be controlled directly, in a manner similar to the way that a puppeteer controls a marionette. This controller also provides a novel means of using angular momentum to enhance stability. Angular momentum balance techniques are used, for example, by tight-rope walkers in order to achieve balance on a very limited support base.

To address the second challenge (future state evolution given actuation limits), a *Task Executive*, [13, 24], capable of predicting future state is used. The Task Executive utilizes the abstraction provided by the DVMC. It implements a control policy for this abstraction by pre-compiling *Flow Tubes* that define valid operating regions for the state variables and control parameters in the abstracted biped. The Flow Tubes represent bundles of state trajectories that take into account dynamic limitations due to under-actuation, and that also satisfy plan requirements. Off-line generation of these Flow Tubes represents a pruning of infeasible trajectories, so that the on-line controller can focus on executing the plan by using only trajectories in the Flow Tubes.

The Task Executive also uses the Flow Tubes to predict whether the plan will succeed or fail. In particular, if a disturbance occurs that pushes the system into a state where no suitable Flow Tube trajectory can be found, then the Task Executive knows that the plan will fail. In this case, it alerts a higher-level control authority, such as a human user. The ability to predict failure ahead of time in this way is important since it provides some time to change the plan, or take other compensating action. For example, a soccer player chasing a ball will abort if it becomes clear that another player will get to the ball first. Similarly, if a person trips while walking and a fall is inevitable, he will put out his hands to mitigate the effects of the fall. In this way, the Task Executive and Flow Tubes also address the fourth challenge (safety of operation in human environments).

To summarize, the Task Executive interprets plan goals, as specified by an input QSP, monitors biped state, and computes control actions for the biped, as shown in Fig. 10.5. The executive computes a sequence of joint torques for the biped that results in achievement of each successive qualitative state goal in the sequence, as shown in Fig. 10.6.

The rest of this chapter provides details of this approach. We begin with a review of biped balance mechanics in Sect. 10.2. This is followed, in Sect. 10.3 by a description of the DVMC, which utilizes the balance mechanics principles. Next, Sect. 10.4 presents the QSP in detail, and Sect. 10.5 describes the Task Executive, which interprets the QSP and utilizes the abstraction provided by the DVMC. Finally, Sect. 10.6 describes experimental results, and Sect. 10.7 provides a discussion of the results and contributions.

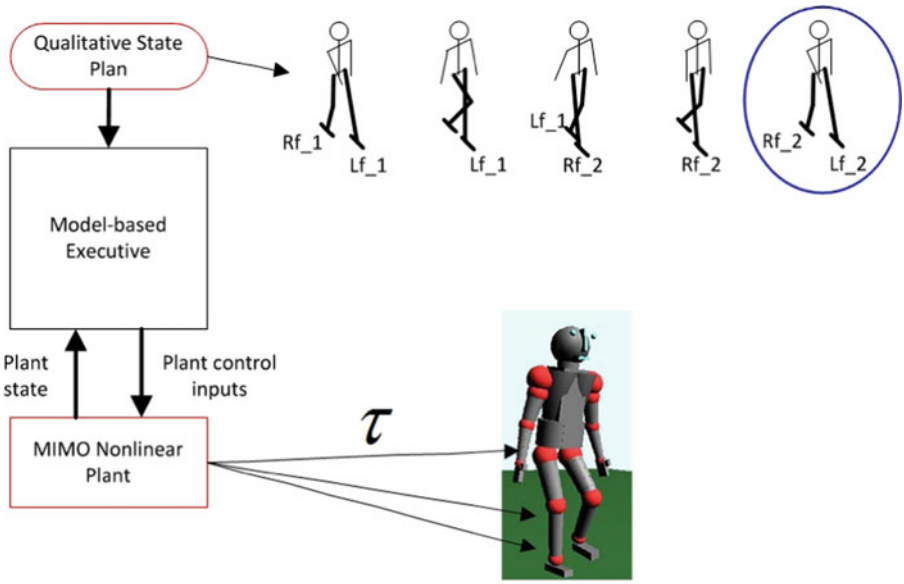


Fig. 10.5 A model-based executive computes a sequence of joint commands for the biped that results in the achievement of the successive qualitative state goals

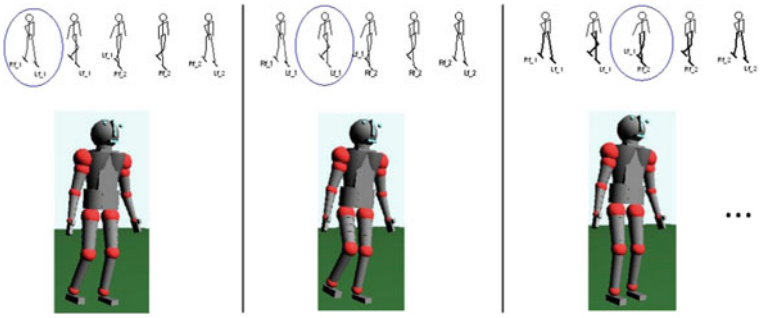


Fig. 10.6 Execution of a sequence in the qualitative state plan

10.2 Analysis of Balance Mechanics and Constraints

Balance control requires the ability to adjust the biped's linear and angular momentum. Due to conservation of momentum laws, such adjustment can only be achieved through force interaction with the environment. For a biped, this force interaction is comprised of gravity and the ground reaction force, the net force exerted by the ground against the biped. The following analysis of physical constraints and requirements for balancing leads to a simple, comprehensive model of balance control that specifies coordination of control actions that adjust the ground reaction

force, and therefore, the momentum of the biped. Similar models have been used previously in a number of gait planning algorithms [12, 15, 19, 25]. These models utilize analysis of inverted pendulum dynamics [5]. A key difference in the model presented here is its ability to purposely sacrifice angular momentum control goals in order to achieve linear control goals when both cannot be met.

The model makes use of a number of physical points that summarize the system's balance state. These points are the center of mass (CM), the zero-moment point (ZMP) [21], and the centroidal-moment point (CMP) [16]. The ZMP is a point on the ground that represents the combined force interaction of all ground contact points. The CMP is the point on the ground from which the ground reaction force would have to emanate if it were to produce no torque about the CM. These points will be defined more formally in the following discussion.

A biped's *support base* [7] is defined as the smallest convex polygon that includes all points where the feet are in contact with the ground. When in single support, that is, where one foot, the stance foot, is on the ground and the other is stepping, the support base is the outline of the part of the stance foot that is in contact with the ground. When in double support, that is, where both feet are on the ground, the base of support is the smallest convex polygon that includes all points where the two feet are in contact with the ground.

The ground reaction force vector, $\mathbf{F}_{\text{gr}}$, is defined as the integral, over the base of support, of the incremental ground reaction forces emanating from each point of contact with the ground:

$$\mathbf{F}_{\text{gr}} = \iint_{\text{BOS}} \mathbf{F}_{\text{gr}}(x, y)\, dxdy \tag{10.1}$$

where $\mathbf{F}_{\text{gr}}(x, y)$ is the incremental force at point x, y on the ground, and BOS refers to the base of support region.

The CM is the weighted mean of the positions of all points in the system, where the weight applied to each point is the point's mass. Thus, for a discrete distribution of masses m_i, located at positions $\mathbf{r}_i$, the position of the center of mass is given by

$$\mathbf{CM} = \frac{\sum_i m_i \mathbf{r}_i}{\sum_i m_i} \tag{10.2}$$

A bipedal mechanism consists of a set of articulated links, each of which is a rigid body with mass m_i. Each rigid body has its own CM at a point $\mathbf{r}_i$.

The CM represents the effective mass of the system, concentrated at a single point. This is valuable because it allows for simplifying the balance control problem to one of keeping the CM in the right place at the right time. Furthermore, the control dynamics of this point is expressed, simply, by Newton's law, $\mathbf{F}_{\text{gr}} = m\mathbf{a}$, where m is the total mass of the system, and $\mathbf{a}$ is the resulting acceleration of the CM.

Vukobratovic and Stepanenko defined the ZMP as the point of resulting reaction forces at the contact surface between the extremity and the ground [22]; it is the

point from which the ground reaction force vector, defined by (10.1), emanates. The ZMP may be defined as the point on the ground surface about which the horizontal component of the moment of ground reaction force is zero [1, 20]. Because the base of support is defined by the convex polygon of points in contact with the ground, and because the ZMP represents the average force contribution of these points, the ZMP is always inside the biped's base of support [6].

We designate the horizontal axes as x and y, where x represents the anterior–posterior direction, and y the medio-lateral direction. We designate the vertical axis as z (positive direction is upwards). The position of the ZMP along these axes, x_{ZMP} and y_{ZMP} can be expressed in terms of CM position, force, and moment as

$$x_{\mathrm{ZMP}} = x_{\mathrm{CM}} - \frac{F_{\mathrm{grx}}}{F_{\mathrm{grz}}} z_{\mathrm{CM}} - \frac{\tau_y}{F_{\mathrm{grz}}} \tag{10.3}$$

$$y_{\mathrm{ZMP}} = y_{\mathrm{CM}} - \frac{F_{\mathrm{gry}}}{F_{\mathrm{grz}}} z_{\mathrm{CM}} + \frac{\tau_x}{F_{\mathrm{grz}}} \tag{10.4}$$

where x_{cm}, y_{cm}, and z_{cm} are the x, y, and z positions of the CM, F_{grx}, F_{gry}, and F_{grz} are the ground reaction forces in the x, y, and z directions, and τ_x and τ_y are the CM moments about the x and y axes, respectively.

Because the gravitational force is purely vertical, F_{grx} and F_{gry} are the net horizontal forces on the CM. The net vertical CM force, F_z, is $F_z = F_{\mathrm{grz}} - Mg$, where g is the gravitational acceleration and M is the total mass. The ZMP is always inside the support base [16]. If the moments in Eq. (10.2) are zero, the ground reaction force vector points directly at the CM, as shown in Fig. 10.7a.

The CMP is the point on the ground, not necessarily within the support base, from which the observed net ground reaction force vector would have to act in order to generate no torque about the CM [for τ_x and τ_y in (10.4) to be 0]. Thus, it is that point

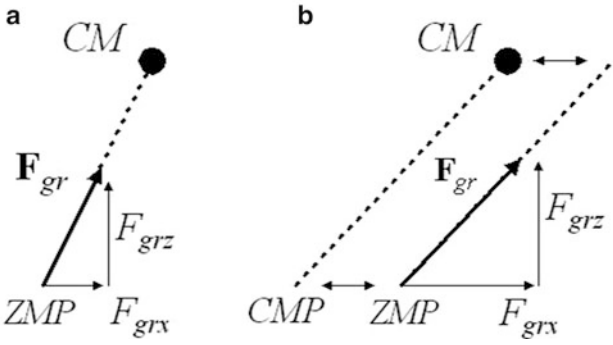


Fig. 10.7 As shown in (**a**), if there is no moment about the CM, the ground reaction force points from the ZMP to the CM position. As shown in (**b**), if there is a moment about the CM, the ZMP and CMP diverge, where the separation distance is the moment arm associated with the vertical force, F_{grz} (©IEEE, 2009, reprinted with permission)

where a line parallel to the ground reaction force vector, passing through the CM, intersects with the ground, as shown in Fig. 10.7b. The CMP can be expressed as

$$(\mathbf{r}_{\mathrm{CMP}} - \mathbf{r}_{\mathrm{CM}}) \times \mathbf{F}_{\mathrm{gr}} = 0. \tag{10.5}$$

Expanding this cross product yields

$$x_{\mathrm{CMP}} = x_{\mathrm{CM}} - \frac{F_{\mathrm{gr}x}}{F_{\mathrm{gr}z}} z_{\mathrm{CM}} \tag{10.6}$$

$$y_{\mathrm{CMP}} = y_{\mathrm{CM}} - \frac{F_{\mathrm{gr}y}}{F_{\mathrm{gr}z}} z_{\mathrm{CM}} \tag{10.7}$$

Note that because $F_{\mathrm{gr}x}$ and $F_{\mathrm{gr}y}$ are the net horizontal forces on the CM, this relation can be used to compute horizontal CM force as a function of CM position, CMP point location, and vertical ground reaction force. Such horizontal forces are critical for maintaining bipedal stability since they can be applied to change CM state to desired values.

By combining Eqs. (10.4) and (10.7), we obtain a relation between ZMP and CMP:

$$x_{\mathrm{CMP}} = x_{\mathrm{ZMP}} + \frac{\tau_y}{F_{\mathrm{gr}z}} \tag{10.8}$$

$$y_{\mathrm{CMP}} = y_{\mathrm{ZMP}} - \frac{\tau_x}{F_{\mathrm{gr}z}} \tag{10.9}$$

Equation (10.9) shows that when there is no horizontal moment about the CM, the CMP and ZMP points coincide. In this case, the ground reaction force vector points directly to the CM, as shown in Fig. 10.7a. Conversely, when there is a horizontal moment about the CM, the CMP and ZMP diverge. The horizontal separation distance between these points is the moment arm for the CM moment due to vertical force, $F_{\mathrm{gr}z}$, as shown in Fig. 10.7b. Note that as the CMP and ZMP diverge, the ZMP must remain within the support base, but the CMP may leave the region of support.

The relationship between the CM and CMP indicates the specific effect that the net ground reaction force has on CM translation. Because the observed net ground reaction force always operates at the ZMP which is within the support base, whenever the net ground reaction force generates no torque about the CM, then the ZMP and CMP coincide. If the net ground reaction force generates torque, however, then the CMP and ZMP differ in location, and, in particular, the CMP may be outside the support base.

It is sometimes desirable to have the CMP and ZMP diverge as shown in Fig. 10.8b so that horizontal CM forces can be more effectively controlled. In this case the CMP can be displaced from the ZMP which reflects the increased ability

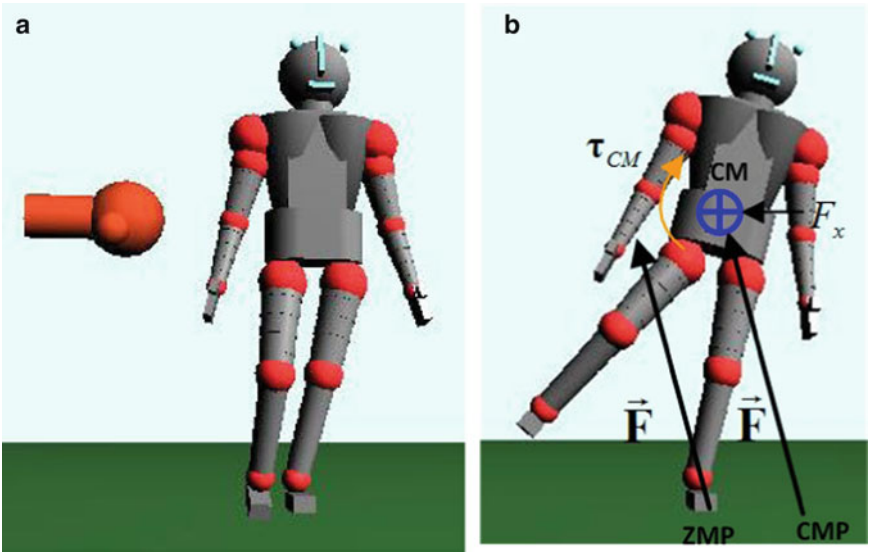


Fig. 10.8 Recovery from a lateral disturbance using CMP. In (**a**), the biped is disturbed by a lateral force. In (**b**), the use of angular momentum to recover from the disturbance is indicated by the divergence of the CMP from the ZMP

of the net ground reaction force to affect translation of the CM. The associated moment about the CM generally produces undesirable effects, such as loss of upright orientation of the upper body. In many cases, these effects are temporary, can be managed, and are well worth the overall positive effect on CM position and velocity. For example, a tightrope walker will tolerate temporary angular instability if this means that he won't fall off the tightrope.

Use of the CMP is demonstrated in Fig. 10.8, which depicts recovery from a lateral disturbance. This sequence shows an initial disturbance that pushes the biped to the right. To compensate, the system takes control actions involving rotation of the body and swing leg, that move its CMP to the right, creating a lateral compensating force to the left. Because the disturbance is significant, the CMP moves beyond the edge of the support polygon, and thus, it does not coincide with the ZMP. This compensating action corresponds to a clockwise torque about the CM, which is manifested by clockwise rotation of the torso and right leg.

The model of balance control presented here, where requirements for balance are expressed in terms of CM, ZMP, CMP, and the support base is extremely useful for planning and control, due to its simplicity. Balance control is then reduced to a problem of adjusting the base of support, adjusting the ZMP within the base of support, and, if necessary, performing motions that generate angular momentum, so that the CMP can be moved, temporarily, outside the base of support, in order to exert additional compensating force on the CM.

10.3 Dynamic Virtual Model Controller

10.3.1 Biped Model

Consider the three-dimensional humanoid biped model, shown in Fig. 10.9. The model has seven segments: two feet, two lower leg segments, two upper leg segments, and a body segment that lumps the torso, head, and arms. The leg and body segments are modeled as cylinders, whereas the feet are modeled as rectangular blocks. Segment dimensions and masses are given in Tables 10.1 and 10.2. Twelve degrees of freedom correspond to joints (six in each leg), and six degrees of freedom correspond to upper body position and orientation. Each leg is modeled with a ball-and-socket hip joint (three degrees of freedom), a pin knee joint (one degree of freedom), and a saddle-type ankle joint (two degrees of freedom). Note that although the humanoid model presented here does not include independently moving arms, the model, and the DVMC control architecture can be easily extended to include them.

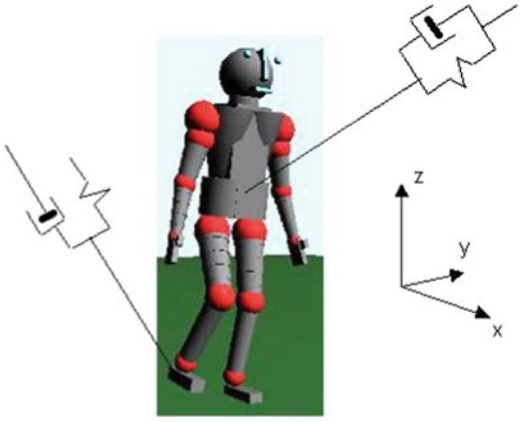


Fig. 10.9 Virtual linear spring-damper elements, attached to reaction points, allow the mechanism to be controlled as if it were a puppet. The coordinate frame is as follows: x is the anterior–posterior axis, y is the medio-lateral axis, and z is the vertical axis (©IEEE, 2009, reprinted with permission)

Table 10.1 Model segment masses

Model segment	Mass (kg)
Foot	1.56
Lower leg	4.48
Upper leg	10.73
Upper body	70.65

Table 10.2 Model segment dimensions

Model segment	Length (m)	Radius (m)
Upper body	0.64	0.18
Upper leg	0.46	0.08
Lower leg	0.48	0.05
Hip spacing	0.25	

Table 10.3 Outputs to be controlled

Index	Output
1	Posterior-anterior CM position
2	Medio-lateral CM position
3	Vertical CM position
4	Upper body roll angle
5	Upper body pitch angle
6	Upper body yaw angle
7	Posterior-anterior swing foot position
8	Medio-lateral swing foot position
9	Vertical swing foot position
10	Swing foot roll angle
11	Swing foot pitch angle
12	Swing foot yaw angle

For single-support case, in the global coordinate frame. For double-support, outputs 7–12 (the ones associated with the swing foot) are omitted

The outputs to be controlled are listed in Table 10.3. These outputs are values relevant to balance control and locomotion, such as CM position, upper body orientation, and stepping foot position. Thus, the purpose of the DVMC is to move the joints so that the desired motion for the outputs is achieved.

10.3.2 Controller Architecture

Desired motion behavior for the outputs is specified in a simple, straightforward way, using a linear proportional-differential (PD) law:

$$\ddot{\mathbf{y}} = \mathbf{k}_s \left(\mathbf{y}_s - \mathbf{y}\right) + \mathbf{k}_d \left(\dot{\mathbf{y}}_s - \dot{\mathbf{y}}\right), \tag{10.10}$$

where $\mathbf{y}$ is the vector of outputs to be controlled, $\mathbf{y}_s$ and $\dot{\mathbf{y}}_s$ are position and velocity setpoint vectors, and $\mathbf{k}_s$ and $\mathbf{k}_d$ are spring and damping gain vectors. Such a control law can be represented as a set of virtual spring-damper elements attached to the output points being controlled, as shown in Fig. 10.9, so that the controlled outputs move as if they were point masses attached to these spring-damper systems.

The difficulty with this is that the robot is not a linear system; the accelerations of the controlled outputs are nonlinear functions of the joint torque actuation inputs. The DVMC solves this problem by providing an abstraction of the plant, shown in Fig. 10.10, which makes it appear linear, and therefore, allows it to follow control laws in the form of Eq. (10.10).

This use of virtual elements is similar, in concept, to the one used in a virtual model controller [17]. However, unlike [17], the DVMC accounts for plant

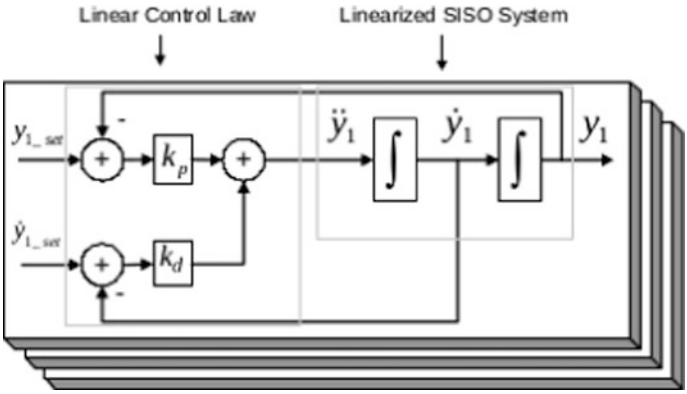


Fig. 10.10 Linear virtual element abstraction consisting of a set of SISO systems with associated linear control laws

dynamics, resulting in a linear system where controlled points move as if they were linear second order systems. Furthermore, through the use of a goal prioritization technique, the DVMC is able to generate moments about the CM in order to generate beneficial forces on the CM.

The DVMC uses a model-based input–output linearization algorithm [18] to linearize and decouple the plant. The input–output linearization approach is augmented with a slack variable relaxation technique to accommodate actuation constraints and prioritize goals. This feature is important because it is not always possible to achieve all control goals simultaneously. Actuation constraints, such as the requirement that the ZMP must remain well inside the support base in the case where foot roll is undesirable, may cause the overall system to become over-constrained, in which case some goals must be deferred. To address this problem, the controller incorporates a goal prioritization algorithm that automatically sacrifices lower-priority goals when the system becomes over-constrained in this way. For example, the system may temporarily sacrifice goals of maintaining upright posture in order to achieve CM state goals. We now describe the linearization and goal prioritization components of the controller in more detail.

10.3.3 Linear Virtual Element Abstraction

A geometric transform, **h**, is used to convert from the joint state to the workspace (output) state representation, according to

$$\mathbf{y} = \mathbf{h}(\mathbf{q}) \tag{10.11}$$

where **q** is the joint position vector, and **y** is the output vector. Thus, **h** is the kinematic transform. The controller uses a feedback linearizing transformation

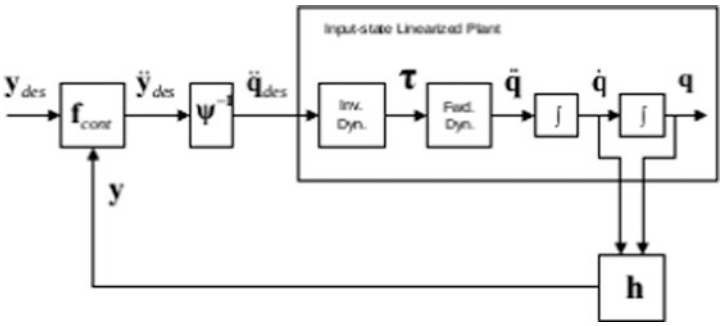


Fig. 10.11 Two-stage linearization

to convert desired workspace variable accelerations, $\ddot{\mathbf{y}}$, into corresponding joint torques. Application of these torques results in a new joint state, and associated workspace state. Use of the linearizing transformation makes the nonlinear plant appear to be a set of decoupled SISO linear second-order systems, as shown in Fig. 10.10. Each SISO system can then be controlled by a proportional-differential (PD) law, as discussed previously, resulting in a *linear virtual element abstraction*.

The linearization is accomplished using a two-stage process, as shown in Fig. 10.11. Given a desired output acceleration vector, $\ddot{\mathbf{y}}_{\text{des}}$, which is computed by the PD law, we first compute the corresponding joint acceleration vector, $\ddot{\mathbf{q}}_{\text{des}}$, using a geometric transformation. Then, we compute the joint torque vector, τ, that achieves $\ddot{\mathbf{q}}_{\text{des}}$, using an inverse dynamics transformation [3].

The inverse dynamics computation is of the form

$$\mathbf{H}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau} \tag{10.12}$$

where $\mathbf{H}(\mathbf{q})$ is a matrix of inertial terms, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is a matrix of velocity-related terms, and $\mathbf{g}(\mathbf{q})$ is a vector of gravitational terms. Hence, for a particular joint state $\left[\mathbf{q}^T, \dot{\mathbf{q}}^T\right]$, Eq. (10.12) represents a linear relation between $\boldsymbol{\tau}$ and $\ddot{\mathbf{q}}$. Note also that because $\boldsymbol{\tau}$ and $\ddot{\mathbf{q}}$ are both 12-element vectors (corresponding to the 12 actuators), $\mathbf{H}(\mathbf{q})$ is a 12×12 square matrix, so Eq. (10.12) is fully constrained.

In order to obtain a relation between $\ddot{\mathbf{y}}_{\text{des}}$ and $\ddot{\mathbf{q}}_{\text{des}}$, we differentiate Eq. (10.11) twice to obtain

$$\dot{\mathbf{y}} = \frac{\partial \mathbf{h}}{\partial \mathbf{q}}\dot{\mathbf{q}} = \mathbf{J}\dot{\mathbf{q}} \tag{10.13}$$

$$\ddot{\mathbf{y}} = \mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}} = \mathbf{J}\ddot{\mathbf{q}} + \Psi \tag{10.14}$$

where $\mathbf{J}$ is the Jacobian matrix. The matrix $\mathbf{J}$ and the vector Ψ are functions of joint state. Therefore, for a particular joint state $\left[\mathbf{q}^T, \dot{\mathbf{q}}^T\right]$, Eq. (10.14) represents a

linear relation between $\ddot{\mathbf{q}}$ and $\ddot{\mathbf{y}}$. Note also that because $\mathbf{q}$ and $\mathbf{y}$ are both 12-element vectors, Eq. (10.14) represents a fully constrained system.

To achieve the linearization shown in Fig. 10.11, we combine Eqs. (10.12) and (10.14):

$$\begin{bmatrix} \mathbf{I}_{12\times12} & \mathbf{0}_{12\times12} & \mathbf{0}_{12\times12} \\ \mathbf{I}_{12\times12} & -\mathbf{J} & \mathbf{0}_{12\times12} \\ \mathbf{0}_{12\times12} & \mathbf{H} & -\mathbf{I}_{12\times12} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{y}} \\ \ddot{\mathbf{q}} \\ \boldsymbol{\tau} \end{bmatrix} = \begin{bmatrix} \ddot{\mathbf{y}}_{\text{des}} \\ \Psi \\ -\mathbf{C} \end{bmatrix} \tag{10.15}$$

Note that this is a fully constrained, linear system.

10.3.4 Multivariable Optimal Controller

The linearization of the square system represented by Eq. (10.15) is subverted if inequality constraints are introduced, and these constraints become active; the system represented by Eq. (10.15) becomes over-constrained in this case. Inequality constraints are used to represent actuation limits. An important constraint of this type is the requirement to keep the stance foot flat on the ground during single support; while balancing on one leg it is undesirable for the stance foot to roll, particularly on its lateral edge. This particular constraint is accomplished by requiring the ZMP to be inside the edge of the support envelope. Note that this constraint is distinct from the physical constraint that the ZMP not be outside the support base. If the ZMP is on the edge of the support envelope, the foot will begin to roll [16]. Hence, in order to avoid foot roll, we employ linear inequality constraints to keep the ZMP inside the edge of the support envelope.

For the humanoid model, the ZMP is given by expanding Eq. (10.4), or

$$x_{\text{ZMP}} = \frac{\sum_{i=2}^{7} m_i r_{xi} (\ddot{r}_{zi} + g) - \sum_{i=2}^{7} m_i r_{zi} \ddot{r}_{xi} - \sum_{i=2}^{7} \tau_{yi}}{\sum_{i=2}^{7} m_i (\ddot{r}_{zi} + g)} \tag{10.16}$$

$$y_{\text{ZMP}} = \frac{\sum_{i=2}^{7} m_i r_{yi} (\ddot{r}_{zi} + g) - \sum_{i=2}^{7} m_i r_{zi} \ddot{r}_{yi} + \sum_{i=2}^{7} \tau_{xi}}{\sum_{i=2}^{7} m_i (\ddot{r}_{zi} + g)} \tag{10.17}$$

$$\boldsymbol{\tau}_{xi} = \mathbf{I}_{Gi} \dot{\boldsymbol{\omega}}_{xi} \tag{10.18}$$

$$\boldsymbol{\tau}_{yi} = \mathbf{I}_{Gi} \dot{\boldsymbol{\omega}}_{yi} \tag{10.19}$$

where i is the segment index, r_{xi}, r_{yi}, and r_{zi} denote the CM position of segment i, $\mathbf{I}_{Gi}$ is the inertia matrix of segment i, and $\boldsymbol{\omega}_{xi}$ and $\boldsymbol{\omega}_{yi}$ are the angular velocities of segment i about the x and y axes, respectively. The moments, $\boldsymbol{\tau}_{xi}$ and $\boldsymbol{\tau}_{yi}$, are about the segment i CM in the x and y axes, respectively.

Equation (10.19) is transformed into a set of linear inequality constraints by replacing x_{ZMP} and y_{ZMP} with min and max terms, reflecting the bounds, so that these become constants:

$$\mathbf{H}_r \ddot{\mathbf{y}}_r \leq \mathbf{K}_r \tag{10.20}$$

where

$$\mathbf{H}_r = \begin{bmatrix} -\mathbf{m}^T \bullet \mathbf{r}_z^T & \mathbf{0}_{1\times 6} & \left(-\mathbf{x}_{\max} + \mathbf{m}^T \bullet \mathbf{r}_x^T\right) & \mathbf{0}_{1\times 6} & -\mathbf{I}_y \\ \mathbf{m}^T \bullet \mathbf{r}_z^T & \mathbf{0}_{1\times 6} & \left(\mathbf{x}_{\min} - \mathbf{m}^T \bullet \mathbf{r}_x^T\right) & \mathbf{0}_{1\times 6} & \mathbf{I}_y \\ \mathbf{0}_{1\times 6} & -\mathbf{m}^T \bullet \mathbf{r}_z^T & \left(-\mathbf{y}_{\max} + \mathbf{m}^T \bullet \mathbf{r}_y^T\right) & \mathbf{I}_x & \mathbf{0}_{1\times 6} \\ \mathbf{0}_{1\times 6} & \mathbf{m}^T \bullet \mathbf{r}_z^T & \left(\mathbf{y}_{\min} - \mathbf{m}^T \bullet \mathbf{r}_y^T\right) & -\mathbf{I}_x & \mathbf{0}_{1\times 6} \end{bmatrix} \tag{10.21}$$

$$\ddot{\mathbf{y}}_r = \begin{bmatrix} \ddot{\mathbf{r}}_x^T & \ddot{\mathbf{r}}_y^T & \ddot{\mathbf{r}}_z^T & \dot{\boldsymbol{\omega}}_x^T & \dot{\boldsymbol{\omega}}_y^T \end{bmatrix}^T \tag{10.22}$$

$$\mathbf{K}_r = \begin{bmatrix} g\left(m_{\text{tot}} x_{\text{zmpmax}} - \mathbf{m}^T \bullet \mathbf{r}_x^T\right)^T \\ -g\left(m_{\text{tot}} x_{\text{zmpmin}} - \mathbf{m}^T \bullet \mathbf{r}_x^T\right)^T \\ g\left(m_{\text{tot}} y_{\text{zmpmax}} - \mathbf{m}^T \bullet \mathbf{r}_y^T\right)^T \\ -g\left(m_{\text{tot}} y_{\text{zmpmin}} - \mathbf{m}^T \bullet \mathbf{r}_y^T\right)^T \end{bmatrix} \tag{10.23}$$

$$\mathbf{x}_{\max} = x_{\text{zmpmax}} \mathbf{m}^T \tag{10.24}$$

$$\mathbf{x}_{\min} = x_{\text{zmpmin}} \mathbf{m}^T \tag{10.25}$$

$$\mathbf{y}_{\max} = y_{\text{zmpmax}} \mathbf{m}^T \tag{10.26}$$

$$\mathbf{y}_{\min} = y_{\text{zmpmin}} \mathbf{m}^T \tag{10.27}$$

where $\mathbf{m}$ is a six-element vector of masses for segments 2–7, $\mathbf{r}_x$, $\mathbf{r}_y$, and $\mathbf{r}_z$ are six-element vectors of the segments' CM x, y, and z positions, $\mathbf{I}_x$ and $\mathbf{I}_y$ are six-element vectors of the segments' inertias about the x and y axes, and x_{zmpmin}, x_{zmpmax}, y_{zmpmin}, and y_{zmpmax} are the ZMP limits. The operator $\bullet$ represents element-wise multiplication.

Now, $\mathbf{r}_x$, $\mathbf{r}_y$, $\mathbf{r}_z$, $\mathbf{I}_x$ and $\mathbf{I}_y$ are kinematic functions of $\mathbf{q}$, and $\mathbf{m}$ is a constant. Therefore, Eq. (10.20) is linear with respect to the current joint state. Furthermore, $\ddot{\mathbf{y}}_r$ is related to the joint acceleration vector through a linear function similar to Eq. (10.15):

$$\ddot{\mathbf{y}}_r = \mathbf{J}_r \ddot{\mathbf{q}} + \Psi_r \tag{10.28}$$

Therefore, if we combine the inequality constraints of Eq. (10.28) with (10.20), we have an overall system that is either fully constrained or over constrained (because $\ddot{\mathbf{y}}_r$ is fully dependent on $\ddot{\mathbf{q}}$, it does not add any flexibility). If none of the constraints

in Eq. (10.20) are active, the system is fully constrained. However, if one of these constraints is active, the system becomes over constrained, and there is no feasible solution. Consequently, if the controller does not take the constraints in Eq. (10.20) into consideration, it could generate values for $\ddot{\mathbf{y}}_{\text{des}}$ that are infeasible.

One way to avoid this type of infeasibility is to use "slack" variables that provide flexibility to the overall system. Thus, the controller output, $\ddot{\mathbf{y}}_{\text{contout}}$, is given by

$$\ddot{\mathbf{y}}_{\text{des}} = \ddot{\mathbf{y}}_{\text{contout}} + \ddot{\mathbf{y}}_{\text{slack}} \tag{10.29}$$

where $\ddot{\mathbf{y}}_{\text{slack}}$ is the vector of slack variables. The goal of the overall control system is then to minimize $\ddot{\mathbf{y}}_{\text{slack}}$, taking into account the relative importance of each element.

This minimization is accomplished by formulating the control problem as a quadratic program (QP), and then using a QP optimizer to solve it. The relative importance of the slack variables is expressed in the cost function for the QP. This causes the optimizer to prioritize goals by first minimizing the slack variables for the most important outputs, and therefore, setting $\ddot{\mathbf{y}}_{\text{contout}}$ to be as close as possible to $\ddot{\mathbf{y}}_{\text{des}}$ for these outputs. For example, slack variables associated with the CM position output are given higher cost than those associated with trunk and swing leg orientation. The slack variable costs were determined empirically. Their precise value is not crucial; as long as the slack costs for CM position are higher than the slack costs for the other outputs, desired behavior is achieved.

The QP formulation is under-constrained, due to the use of the slacks. The QP optimizer explores the null space of the formulation, choosing the solution that minimizes the cost function. This cost function is of the form $\mathbf{w} \cdot \ddot{\mathbf{y}}_{\text{slack}}$ where $\mathbf{w}$ is a vector of weights reflecting the importance of minimizing the associated slack variable [11].

10.4 QSP and Problem Specification

We seek to guide a bipedal mechanism so that it accomplishes a particular motion task, such as walking at a specified speed, region, walking on a set of irregularly placed stones, or walking to a soccer ball in time to kick it. Motion tasks are specified by a QSP, which is executed by a *Model-based Executive* [9, 14]. The executive uses a *Plant Model* combined with current state estimates to generate control inputs. The flexibility provided by state and temporal constraints in the QSP allows the executive to consider multiple possible state and control input sequences, and to choose the most appropriate one given the situation. Sections 10.4.1 and 10.4 formally define the Plant Model and QSP representations and Sect. 10.4.3 defines the problem solved by the executive, based on these inputs.

10.4.1 Plant Model

We assume that the plant can be modeled as a set of subsystems whose dynamics are linear and decoupled, within specified discrete *modes*. The linearization and decoupling are provided by the DVMC, described previously, which is a component of the Model-based Executive. Thus, the Plant Model provides the Model-based Executive an abstraction of the actual system, which is easier to control. Each discrete mode, for example, single support or double support, has its own set of linearized dynamics. Such a plant model, which incorporates all discrete modes and associated continuous linearized dynamics is called a hybrid (discrete/continuous) model. We now define a plant model more formally.

Definition 1 (Plant Model). A Plant Model is a *Hybrid Concurrent Constraint Automaton* (HCCA) [8], which consists of a set of hybrid automata. Each automaton is defined by the tuple $A_a = \langle m_a, L_a, T_a \rangle$, where m_a is the discrete mode of the automaton, L_a maps each mode to a *Linearized Subsystem* that defines the continuous dynamic behavior of the mode, and T_a is a set of transition functions. A Linearized Subsystem is a tuple $\langle \mathbf{x}, \mathbf{u}, \mathbf{A}, \mathbf{B}, \mathbf{c_c} \rangle$, where $\mathbf{x} \in \Re^n$ is the state vector, $\mathbf{u} \in \Re^m$ is the input vector of the subsystem, and $\mathbf{A} \in \Re^{n \times n}$, $\mathbf{B} \in \Re^{m \times n}$ are matrices that represent the plant dynamics according to $\dot{\mathbf{x}} = \mathbf{Ax} + \mathbf{Bu}$. Additionally, $\mathbf{c_c}$ is a set of actuation constraints of the form $\mathbf{H_c} \left[\mathbf{x} \;\; \mathbf{u} \right]^T \leq \mathbf{K_c}$. Given a current mode assignment and guard condition g_a, each transition function $\tau_a (m_a, g_a)$ specifies a target mode that the automaton will transition into, if the guard is satisfied. A guard condition is associated with a Linearized Subsystem, and is represented by a set of (convex) linear algebraic equality and inequality constraints over the state vector of the Linearized Subsystem: $\mathbf{f}(\mathbf{x}) = \mathbf{c_1}$, $\mathbf{g}(\mathbf{x}) \leq \mathbf{c_2}$ where $\mathbf{c_1}$ and $\mathbf{c_2}$ are vectors of constants.

One requirement for successful plan execution is that the state trajectory satisfies the dynamics and actuation constraints of the linearized subsystems, and that it corresponds to valid mode transitions of the automata in the plant. We call such a trajectory a *Plant-feasible Trajectory*. We develop this concept by first defining a *Mode Feasible Trajectory*, for a particular mode of a particular automaton in the HCCA, and then generalizing.

Definition 2 (Mode Feasible Trajectory). Given a Plant Model, an automaton, A in the model, and a mode, M, for the automaton, imply a particular Linearized Subsystem $L = L(A, M)$. We call a state and input trajectory, $\langle \mathbf{x}(t), \mathbf{u}(t) \rangle$, a *Mode Feasible Trajectory* with respect to M if it satisfies the dynamics and actuation constraints of L, as specified in Definition 1.

We now utilize this definition as a basis for defining feasible trajectories for a mode sequence in an automaton, and for automata in a plant.

Definition 10.1 (Automaton Feasible Trajectory). Given a Plant Model, and a particular automaton, A in the model, suppose we have a sequence of trajectories

$\{T_0, T_1, \ldots, T_n\}$, where $T_i =< x_i(t), u_i(t) >, t \in [ts_i, tf_i]$. Suppose, further, that each trajectory, T_i, in the sequence is a Mode Feasible Trajectory with respect to a mode M_i in A. We call the sequence of trajectories an *Automaton Feasible Trajectory* if for every trajectory, T_i, in the sequence, the final continuous state, $x_i(tf_i)$, in the trajectory satisfies the guard condition for transition to mode M_{i+1}, and $x_i(tf_i) = x_{i+1}(ts_{i+1})$.

Definition 10.2 (Plant Feasible Trajectory). Given a Plant Model, a *Plant Feasible Trajectory* is a set of Automaton Feasible Trajectories, one for each automaton in the plant, where the start and finish times of all Automaton Feasible Trajectories in the set are the same.

10.4.2 Qualitative State Plan

In this sub-section, we begin with an informal description of a QSP, and provide an example of such a plan. We follow this with a formal definition of a QSP, and in the following sub-section, a formal definition of the problem solved by the Model-based Executive.

A QSP provides a loose, flexible specification of desired performance, in terms of state space regions and temporal ranges. This flexibility may be exploited, for example, to improve optimality or to adapt to disturbances (improve robustness).

Reaching a goal location may require the biped to take a sequence of steps. Such steps represent transitions through a sequence of qualitatively different states, defined by which feet are in contact with the ground. Thus, a stepping sequence consists of alternating between double support phases, where both feet are on the ground, and single support phases, where one foot (the stance foot) is in contact with the ground, and the other foot (the swing foot) is taking the step. These phases represent qualitatively different system states, with correspondingly different behaviors.

We formalize the concept of a *Qualitative State* as a set of constraints on state and temporal behavior. For example, a Qualitative State may contain constraints on which feet of a legged robot are on the ground, and may include constraints on the position of each foot. It may also include state constraints on quantities like center of mass, and temporal constraints specifying time ranges by which the state goals must be achieved. Thus, a qualitative state is a loose, partial specification of desired behavior for a specific maneuver, like taking a step.

For example, a plan for a biped divides the walking cycle into a sequence of Qualitative States representing single and double support gait phases. Such a plan is shown in Fig. 10.12. In this plan, the first Qualitative State represents double support with the left foot in front, the second, left single support, the third, double support with the right foot in front, and the fourth, right single support. The fifth Qualitative State repeats the first, but is one gait cycle forward.

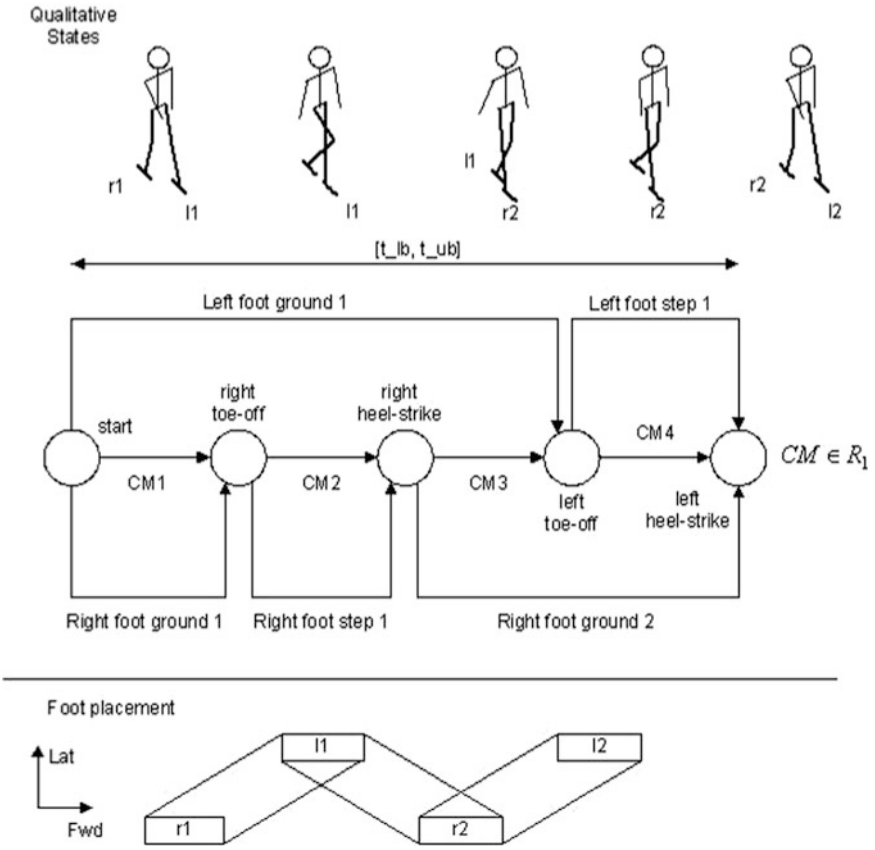


Fig. 10.12 Example QSP for walking gait cycle. *Circles* represent events, and *horizontal arrows* between events represent activities. For example, the activity "left foot ground 1" indicates that the left foot is on the ground from the event "start" to the event "left toe off". The activity "left foot step 1" indicates that the left foot is stepping (system is in right single support) from the event "left toe off" to the event "left heel strike". Similar activities for the right foot indicate when the right foot is on the ground, and when it is stepping. Activities may have associated state space constraints, such as the goal region constraint CM $\in r_1$, which specifies a goal for CM (center of mass) position and velocity. Foot placement constraints are indicated at the *bottom*; for example, rectangle r_1 represents constraints on the first right foot position on the ground, and rectangle l_1 on the first left foot position. The *lines between the rectangles* define the polygon of support when in double support

The QSP in Fig. 10.12 has a temporal constraint between the start and finish events. This constraint specifies a lower and upper bound, $[l, u]$, on the time between these events. It is a constraint on the time to complete the gait cycle, and thus, can be used to specify walking speed.

In addition to temporal constraints, QSPs include state constraints that specify valid initial, operating, and goal regions for an activity. If an initial region is specified for an activity, then the trajectory must be within this initial region, in order

for the activity to begin. If an operating region is specified, then the trajectory must stay within this region for the duration of the activity. If a goal region is specified, then the trajectory must be within this region in order for the activity to end. In Fig. 10.12, the goal region constraint CM $\in r_1$ represents the requirement that the CM trajectory must be in region r_1 for the CM movement activity to finish successfully.

We now provide formal definitions for *Events* and *Activities*, and then use these components to define a QSP.

Definition 10.3 (Event). An event, e, represents a point in time. For a schedule, T, the specific time of e is given by $T(e)$.

Definition 10.4 (Activity). An activity is a tuple $\langle e_s, e_f, R_{\text{input}}, R_{op}, R_{\text{init}}, R_{\text{goal}}, s_i \rangle$, where e_s is an event representing the start of the activity, e_f is an event representing its finish, s_i is a Linearized Subsystem associated with the activity, R_{input} is a set of constraints on the inputs, $\mathbf{u}$, of s_i, R_{op} is a set of operational constraints on the state, $\mathbf{x}$, of s_i that must hold for the duration of the activity, R_{init} is a set of constraints on the state that must hold for the activity to begin, and R_{goal} is a set of constraints on the state that must hold for the activity to finish. The state constraints, R_{op}, R_{init}, and R_{goal}, are each of the form $\mathbf{Hx} \leq \mathbf{K}$, where $\mathbf{H} \in \Re^{q \times n}$ and $\mathbf{K} \in \Re^{q \times 1}$, and q is the number of linear inequalities in the set. The input constraints, R_{input}, are of the form $\mathbf{H}\left[\mathbf{x}^T\ \mathbf{u}^T\right]^T \leq \mathbf{K}$.

Definition 10.5 (QSP). A QSP is a tuple $\langle E, A, C \rangle$, where E is a set of Events, A is a set of Activities, and C is a set of externally imposed temporal constraints on the start and finish times of the activities. For example, the QSP shown in Fig. 10.12 has five Events ("start," "right toe-off," "right heel-strike," "left toe-off," "left heel-strike"), nine activities ("Left foot ground 1," "Left foot step 1," "CM1," "CM2," "CM3," "CM4," "Right foot ground 1," "Right foot step 1," "Right foot ground 2"), and one temporal constraint.

Definition 10.6 (Temporal Constraint). A temporal constraint is a tuple $\langle e_1, e_2, l, u \rangle$, where e_1 and e_2 are events, and l and u represent lower and upper bounds on the time between these events. Thus, $l \in \Re \cup \{-\infty\}$, and $u \in \Re \cup \{\infty\}$, such that $l \leq t(e_2) - t(e_1) \leq u$. In the QSP of Fig. 10.12, the temporal constraint restricts the time between the start and finish events. Events are used to represent start and finish times of an activity.

10.4.3 Plan Execution: The Problem Solved by the Model-Based Executive

Having formally defined a Plant and a QSP, we are now in a position to define the problem solved by the Model-based Executive in terms of a successful execution of a QSP. Successful execution can be expressed in terms of satisfaction of the individual activities in the QSP, and a consistent schedule, which combined, define satisfaction of a QSP.

Definition 10.7 (Schedule and Consistent Schedule). Given a QSP, Q, a *Schedule*, T, is an assignment of a specific time to each Event in Q. T is consistent with Q if it satisfies all Temporal Constraints in Q, that is, for each Temporal Constraint, $c \in C(Q)$, then a schedule assigns $t(e_1(c)) = T_1$, $t(e_2(c)) = T_2$ such that $l(c) \leq T_2 - T_1 \leq u(c)$, where e_1, e_2, l, u are the elements of c from Definition 10.8.

Definition 10.8 (Satisfaction of an Activity). Given an activity, a (Definition 10.6), with associated Linearized Subsystem s_i, a plant-feasible trajectory $\langle \mathbf{x}(t), \mathbf{t}(t) \rangle$ for s_i, and a schedule T, then a is satisfied by $\mathbf{x}(t)$ and T if the following conditions hold:

(1) $\mathbf{x}(t)$ must satisfy the initial and goal region state constraints of a. Let $t_s = T(e_s(a))$ be the start time of a under schedule T, and $t_f = T\left(e_f(a)\right)$ be the finish time. Then, $\mathbf{x}(t)$ satisfies the initial and goal region constraints if $\mathbf{x}(t_s) \in R_{\text{init}}(a)$ and if $\mathbf{x}\left(t_f\right) \in R_{\text{goal}}(a)$.
(2) $\mathbf{x}(t)$ must satisfy the operating state constraints of a. That is, it must be the case that $\mathbf{x}(t) \in R_{op}(a)\ \forall t : t_s \leq t \leq t_f$.

Definition 10.9 (Satisfaction of a QSP). Given a QSP, Q, plant-feasible trajectory $\langle \mathbf{X}(t), \mathbf{U}(t) \rangle$, and a schedule, T, then Q is satisfied by $\langle \mathbf{X}(t), \mathbf{U}(t), T \rangle$ if T is consistent with Q (Definition 10.7), and $\langle \mathbf{X}(t), \mathbf{U}(t), T \rangle$ satisfies all activities in Q (Definition 10.8).

We now formally define the problem solved by the Model-Based Executive.

Definition 10.10 (Problem Solved by the Model-Based Executive). Given a QSP, Q, and a Plant Model, M, the Model-Based Executive must find a plant-feasible trajectory and schedule that satisfy the QSP (Definition 10.9), and then execute that trajectory and schedule. If no such trajectory and schedule exist, the executive will abort and indicate a plan infeasibility error. If a disturbance occurs during execution, then the Model-Based Executive must find a new plant-feasible trajectory and schedule, and continue execution. If no such trajectory and schedule exist, the executive will abort and indicate a plan infeasibility error.

The next section describes the Model-based Executive, and how it solves this problem.

10.5 Task Executive

The Task Executive (Model-based Executive) is responsible for attempting to execute a QSP, according to Definition 10.10. It does this using a two-part approach. The first part is an off-line component in which the QSP is compiled into a form that can be executed more efficiently. The second part is an on-line component that performs this execution.

10.5.1 Plan Compilation

In order to reduce runtime computational load, we construct, at compile time, a *Qualitative Control Plan* (QCP), which uses *Flow Tubes* to represent all trajectories that satisfy the QSP and the plant dynamics (see also [10]). Using the QCP, the executive achieves efficiency by selecting an appropriate trajectory, within each flow tube, that begins at the current system state. We first define the QCP, and present theorems that define conditions under which the problem is solvable by the Model-Based Executive. We then describe the algorithm for compiling a QCP, given a QSP and Plant Model.

10.5.1.1 Qualitative Control Plan

A key concept in plan feasibility for a hybrid system is *Temporal Feasibility* of individual activities in the plan. Temporal feasibility implies that the set of plant feasible trajectories includes ones that go from R_{init} to R_{goal} over the entire duration range $[l, u]$. More specifically, if a trajectory starting anywhere in the initial region, R_{init}, of the activity and ending somewhere in the goal region, R_{goal}, at any duration d such that $l \leq d \leq u$, is Plant Feasible, then the activity is temporally feasible in the duration range. This implies that the actuation limits imposed by the plant, and the operating constraints of the activity allow for actuation commands that can be used to control the linearized subsystem to the goal region from the initial region, at any duration in the duration range.

We now introduce the concept of a control policy for an activity, and use this, along with the previous definition, to define a *Temporally Controllable Activity*. A control policy maps a state, $\mathbf{x}$, associated with an activity's plant, to an actuation command $\mathbf{u}$ for the plant. A *Valid Control Policy* must provide the mapping for all states (all $\mathbf{x}$) that satisfy the operating constraints of the plant and the activity.

Definition 10.11 (Temporally Controllable Activity). Given an activity and associated plant, and given a Valid Control Policy, P, for the activity, the activity is *Temporally Controllable* by P in the duration range $[l, u]$ if the activity is Temporally Feasible in this range, and if all trajectories for the activity are consistent with (generated by) P. A trajectory is consistent with, or generated by P if for every state $x(k)$ in the trajectory, the subsequent state $x(k + 1)$ results from applying P to $x(k)$.

We next use the concept of a Temporally Controllable Activity to state the conditions under which a QSP can be satisfied. Given a Plant Model and a QSP, suppose that each activity a_i in the QSP is Temporally Controllable over the duration range $[l_i, u_i]$. Let N be the Simple Temporal Network formed by combining the temporal constraints explicitly specified in the QSP, with the duration range temporal constraints $[l_i, u_i]$. If N is dispatchable [14], then a trajectory and schedule exist that satisfy the QSP (see Definition 10.11). Under these conditions, the plan

is feasible with respect to the plant and control policy. This extends the notion of *dispatchability*, first introduced by Muscettola for discrete activity systems [14] , to general hybrid system.

In order to support efficient execution, the flow tube representation must allow the dispatcher to: (1) quickly determine whether a feasible state trajectory exists from the current state, and (2) if such a trajectory exists, what the control commands should be (based on a control policy) that achieve the trajectory. In order to leverage the advantages of dispatchable representations for discrete activity systems [4, 14], we require that the QCP temporal constraints be represented in minimum dispatchable form. This should include the temporal constraints specified in the QSP, and those implied by the dynamic limitations of the plant.

A variety of approaches are possible for the Flow Tube implementation. These include explicit sets (bundles) of trajectories with associated control policies, and discrete time sequences of polytope cross sections that define the feasible state space and associated control policies [2]. Regardless of the implementation, the Flow Tube representation must support in its API the following three functions.

Given a current plant state, $\mathbf{x}$, a goal region to achieve, $\mathbf{R}_g$, and an allowed duration range, $[l, u]$, the function

$[\mathbf{u}, d] = \textbf{ComputeControlAction}\left(\mathbf{x}, \mathbf{R}_g, [l, u]\right)$

must determine whether a feasible trajectory exists that will reach the goal from the current state within the allowed duration range. If so, it returns the next control action, $\mathbf{u}$, consistent with moving the state along the trajectory, along with a prediction d of the remaining duration until the goal is achieved. If no feasible trajectory exists, **ComputeControlAction** returns an error indicating that this is the case. The goal region is represented as a convex polytope: $\mathbf{R}_g = \mathbf{Hx} \leq \mathbf{K}$, where $\mathbf{x} \in \Re^n$ is the state vector of the plant. This function is used by the Model-based Executive to generate commands to the biped, at each control step.

The function

$[l, u] = \textbf{ComputeControllableDuration}\left(\mathbf{x}, \mathbf{R}_g\right)$

computes the range of feasible (controllable) durations in which the state can be moved from the current state to the goal set. If no such duration range exists, **ComputeControllableDuration** returns an error indicating that this is the case. This function is used by the Model-based Executive to schedule activities consistently at runtime.

Given an initial region in state space, $\mathbf{R}_i$, the function

$[l, u] = \textbf{ComputeControllableDuration}\left(\mathbf{R}_i, \mathbf{R}_g\right)$

computes the range of feasible (controllable) durations in which the state can be moved from any state in the initial set to the goal set. If no such duration range exists, **ComputeControllableDuration** returns an error indicating that this is the case. This function is used by the Model-based Executive at compile time to generate QCPs that can be scheduled at runtime.

Now, consider two QSP activities, A_1 and A_2, that share the same linearized subsystem, s_i. Suppose that the finish event of A_1 is the start event of A_2. We call A_2 the *Successor Activity* of A_1. Consider feasible trajectories for A_1 and A_2 as shown in Fig. 10.13.

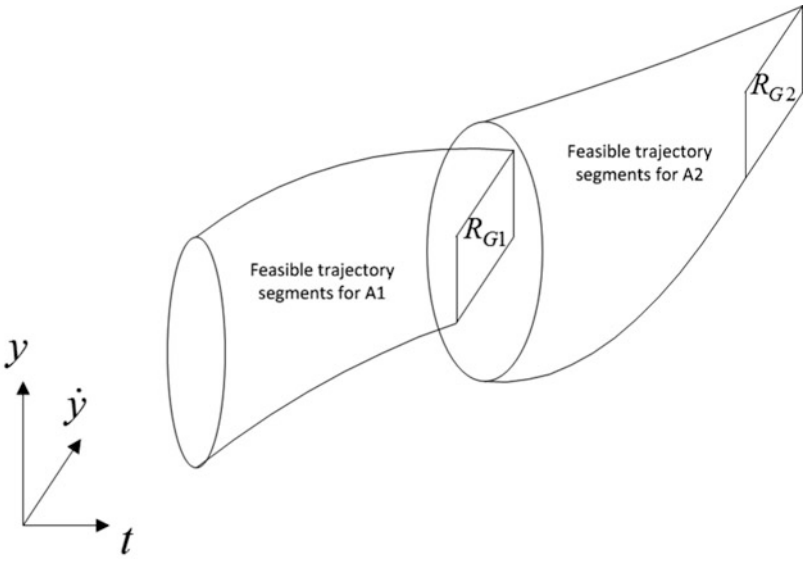


Fig. 10.13 Feasible trajectory segments for A_1 and A_2

Because A_2 is the successor of A_1, any feasible trajectory segment for A_1 must be part of a trajectory that has a feasible trajectory segment for A_2. Therefore, it is a requirement that the goal region for the flow tube for A_1 be a subset of the initial cross section of the flow tube for A_2.

We now formally define a QCP in terms of *Control Activities*, and what it means for a QCP to be executed successfully. A QCP has a structure similar to that of a QSP, but augments this with flow tube cross-sections representing feasible state trajectories and corresponding control policies. A control activity includes the information of the corresponding activity in the QSP, augmented with flow tubes specifying the activity's feasible state trajectories and corresponding control policies.

Definition 10.12. A *Control Activity* is a tuple $< A, F >$, where A is an activity (Definition 10.6), and F is a corresponding Flow Tube that implements the **ComputeControlAction** and **ComputeControllableDuration** functions.

Definition 10.13. A QCP is a triple $< E, A_c, C_t >$, where E is a set of events (Definition 10.5), C_t is a set of temporal constraints on the events (Definition 10.8), and A_c is a set of control activities (Definition 10.12). Each event is either a start event or finish event of a Control Activity.

Having specified the structure of a QCP, we now specify properties of a valid QCP. We begin with a Lemma that specifies requirements for activity succession.

Lemma 10.1. *Given a control activity, A_2, with predecessor activity A_1, for any specified duration range $[l_{\text{des}}, u_{\text{des}}]$, if*

$[l, u] = \textbf{\textit{ComputeControllableDuration}}\left(R_{\text{goal}}(A_1) \cap R_{\text{init}}(A_2), R_{\text{goal}}(A_2)\right)$
returns an error (no duration, $[l, u]$), then no valid trajectories exist for the activity. Further, if a duration, $[l, u]$ exists, but $[l, u] \cap [l_{\text{des}}, u_{\text{des}}]$ is empty, then no valid trajectories exist for the activity in the desired duration range $[l_{\text{des}}, u_{\text{des}}]$.

Note that the elimination of feasible durations for activities due to this Lemma will result in a tightening of the temporal bounds for such activities, beyond the [l, u] bounds specified in the QSP. In the extreme case, there are no feasible durations for the activity at all, leading to the following Lemma.

Lemma 10.2. *If there are no feasible durations according to Lemma 10.1 within the $[l, u]$ bounds specified for the control activity in the QCP, then there are no feasible trajectories for the activity, or for the QSP as a whole. Conversely, if a feasible duration exists, then a feasible trajectory exists for the activity that satisfies all initial, goal, operational, and actuation constraints.*

The tightening of temporal constraints due to Lemma 10.1 can cause a QCP to become infeasible, even if the individual activities are all feasible according to Lemma 10.2. This is expressed in the following theorem.

Theorem 10.1. *If a minimum dispatchable graph based on the temporal constraints specified in a QCP (Definitions 10.7 and 10.8), and possibly tightened according to Lemma 10.1, has a negative loop, then the QCP is infeasible.*

Proof. A minimum dispatchable graph represents a network of temporal constraints. If this graph has a negative loop, then there is an inconsistency in the temporal constraints [14]. If the minimum dispatchable graph is based on the temporal constraints explicitly specified for the QSP, as well as the additional temporal constraints implied by Lemmas 1, then a negative loop indicates an inconsistency in the overall set of temporal constraints, and the QCP is infeasible. □

A temporal inconsistency may result if the explicitly specified temporal constraints are inconsistent, or if they are inconsistent with the overall set of temporal constraints due to Lemma 10.1. The concepts presented here are useful for recognizing cases where the problem formulation, as represented by the QSP and Plant Model, is infeasible.

10.5.1.2 Plan Compilation Algorithm

The purpose of the plan compiler is to generate a QCP from a QSP. The main compilation steps are shown in Algorithm 1. The algorithm iterates over each activity in the QSP, calling ComputeFlowTubeForActivity. If the result is a flow tube with no valid trajectories, then the algorithm stops and indicates an error; the QSP is infeasible. If the flow tube is non-empty (has valid trajectories), then these trajectories are feasible in terms of plant dynamics, but not necessarily other

Algorithm 1: `CompileQSP`

Input: A QSP, Q, a plant model, M
Output: A QCP

```
1 foreach activity in the QSP do
2     flow tube ← ComputeFlowTubeForActivity(activity);
3     if !ValidTrajectories(flow tube) then
4         Error (Activity goal, operational, and temporal constraints are incompatible with
          plant dynamics and actuation constraints.)
5     PruneInfeasibleTrajectories (flow tube, activity, QSP);
6     UpdateFeasibleDuration (flow tube, activity);
7 STN ← ComputeMinimumDispatchableGraph(QSP);
8 if STNInfeasible(STN) then
9     Error (Temporal constraints, and plant dynamics and actuation constraints are
      inconsistent.)
```

aspects of the QSP. Therefore, the algorithm calls PruneInfeasibleTrajectories, which performs the intersection of initial region, predecessor goal region, and durations specified in Lemma 10.1. The consistency of temporal constraints is checked using a minimum dispatchable graph algorithm [14].

10.5.2 Plan Execution

To execute a QCP, the Dispatcher (online component of the Task Executive) must successfully execute each control activity. The dispatcher accomplishes this by, in real time, monitoring plant state, and generating plant control inputs based on the appropriate QCP control policy for the current state and time. In this way, the dispatcher indirectly schedules start and finish events so that they are consistent with the temporal constraints of the QCP. This is a key difference between the dispatcher described here, and those of discrete activity execution systems [14], in which event times are set directly by the dispatcher.

The Dispatcher performs three key functions in executing a control activity: initialization, monitoring, and transition. During initialization, the dispatcher chooses a goal duration for the control activity that is consistent with its execution window, and computes an initial control input. This control input is consistent with an optimal trajectory that will reach the activity's goal region in the chosen duration, if there are no disturbances.

After initializing an activity, the dispatcher begins monitoring its execution by obtaining an updated state estimate at each time increment, and checking whether the state is within the flow tube. If this is not the case, then a disturbance has occurred, and the dispatcher must determine the type of disturbance, and react accordingly.

As part of the monitoring function, the dispatcher also checks whether the state trajectory has achieved the activity's goal region in an acceptable time. If this is the case, it checks whether the activity's end event has occurred. This involves checking if the state trajectories of other activities whose completion must be synchronized are in their respective goal regions. If all completion conditions for a control activity are satisfied, the dispatcher switches to the transition function. If the control activity has a successor, the transition function invokes the initialization function for this new activity. As part of this transition, the dispatcher notes the time of the transition event and propagates this through the temporal constraints.

For each executing activity, the Dispatcher maintains a *target completion time*. The concept of a target completion time represents a key distinction between this system, and discrete activity execution systems [14]. This concept is needed here because activity completion is controlled indirectly, by applying control inputs. Thus, activity completion event times cannot simply be set, but rather occur as a consequence of the plant dynamics. If there are disturbances, the target completion time may have to be adjusted.

The dispatcher chooses a target activity completion time, and then selects an appropriate control policy that is predicted to complete the activity at the desired time. Because multiple activities are typically executing in parallel, achieving a desired event execution time requires synchronization of these activities. The Dispatcher uses a data structure called the *Event Horizon* to provide a mechanism for ensuring this consistency.

Definition 10.14 (Event Horizon). An Event Horizon is a set of Event Paths, each of which is a list of events and associated target execution windows. Hence, each element of an Event Path is a tuple $\langle e, l, u \rangle$ where e is the event, and l and u represent lower and upper bounds on times for the event.

The Event Paths represent events whose target execution windows have to be propagated in order to properly set target completion times for current activities in the Runtime Activity State. This propagation is a special kind of tightening of execution windows, distinct from the execution window tightening that is performed when events occur. This special propagation is necessary in order to ensure that target activity completion times are temporally consistent. This is a unique feature of our dispatcher; it is not used in discrete activity execution systems.

The algorithm for determining the event horizon involves starting at the target events of current activities, and searching back along outgoing negative arcs in the minimum dispatchable graph until executed events are reached. Consider the example QSP shown in Fig. 10.14. Activity a11 is for linearized sub-system 1, and activities a21, a22, and a23 are for linearized sub-system 2. Suppose that event ev1 has just occurred, and activities a11 and a21 are about to start executing. The dispatcher must choose target completion times for each activity. However, it cannot just choose target completion times that fit inside the event execution windows that were propagated when ev1 occurred; the Event Horizon must be taken into account. For example, suppose that the duration constraints on a11 are [3, 15], and the duration constraints on a21, a22, and a23 are [1, 5]. If event ev1 occurs at time

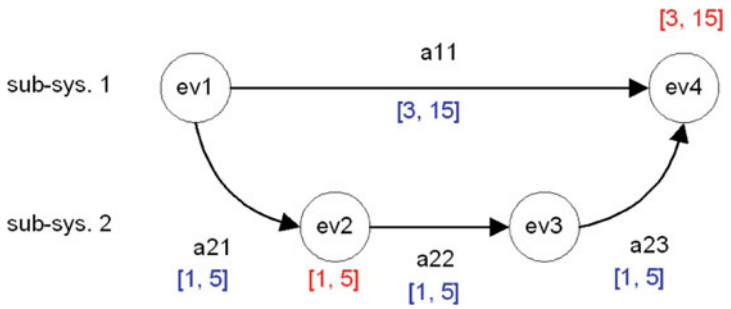
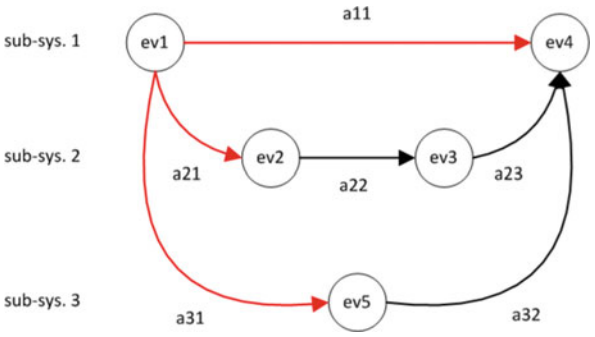


Fig. 10.14 Example event horizon. *Circles* represent events, and *horizontal arrows* between events represent activities

Fig. 10.15 Example Event Horizon. Circles represent events, and horizontal arrows between events represent activities.



0, then the execution window for ev4 is [3, 15], and the execution window for event ev2 is [1, 5]. If the dispatcher were to base its decisions about target completion times solely on these execution windows, it could choose a target execution time of 4 for a21, and 4 for a11. This would cause a future temporal infeasibility, because a22 and a23 would have to be executed in 0 time, which violates their duration constraints.

To solve this problem, the dispatcher considers the event horizon, which, in this case, is ev1, ev2, ev3, ev4. In this case, if the dispatcher chooses duration range midpoints as target durations, then the target completion times for a21, a22, and a23 are 3, 6, and 9, respectively, and the target completion time for a11 is 9 as well.

Algorithm 2 shows the top-level dispatch loop of the executive. This algorithm is based on the one for discrete activity execution systems [14], but has some key extensions, which are highlighted.

InitializeActivities sets target execution times for the initial activities in the QCP. UpdateCurrentActivities iterates over each currently executing activity, checking if the event that has just occurred, ExecutableEvent?, is the finish event for the activity. If this is the case, it transitions the activity to the subsequent activity for the plant.

Algorithm 2: `Executive Dispatcher algorithm`

Input: A Qualitative Control Plan, QCP, and a plant model, P
Output: Actuation commands, u

```
1  t = 0 ;                                          // Initialize current time.
2  InitializeActivities(QCP, P);
3  ExecutableEvent? = StartEvent(QCP);
4  EnabledEvents = { ExecutableEvent? };
5  InitializeExecutionWindows ();
6  while EnabledEvents not empty do
7      if ExecutableEvent? exists then
8          ExecuteEvent (ExecutableEvent?) ;
9          PropagateExecutionWindows (QCP, ExecutableEvent?) ;
10         EnabledEvents = UpdateEnabledEvents (QCP, ExecutableEvent?) ;
11         UpdateCurrentActivities(ExecutableEvent?, QCP) ;
12         EventHorizon = UpdateEventHorizon(EventHorizon, ExecutableEvent?,
           QCP) ;
13         UpdateTargetExecutionTimes(EventHorizon, QCP) ;
14     UpdateCurrentState(P) ;
15     UpdateControlInputs(P) ;
16     t = t + dt ;                                  // Increment current time.
17     ExecutableEvent? = EventOccurred(t, EnabledEvents);
```

UpdateEventHorizon is used to update the event horizon after an event has occurred. To perform the update, the algorithm first removes any paths from the event horizon that contain the executed event. The algorithm then iterates over each currently executing activity, obtaining the target event for the activity. If the target event is not already in the event horizon, a search back from this event is started. This depth-first search proceeds back from the target event along negative out-going arcs in the minimum dispatchable graph. The search proceeds back along events that have not been executed. When an event that has been executed is encountered, the search branch stops, and search proceeds along the next branch. When there are no more branches, the path is added to the event horizon. For Fig. 10.14, this algorithm computes an event horizon $\{\{ev1, ev2, ev3, ev4\}\}$.

UpdateTargetExecutionTimes uses the event horizon to decide target completion times for activities. The algorithm first initializes target execution windows of all events in the event horizon to be identical to the execution windows computed by InitializeExecutionWindows and PropagateExecutionWindows. It then iterates over every path in the event horizon, retrieving the activity corresponding to the beginning of the path. This is always a currently executing activity. The algorithm sets the target completion time for the activity to be the midpoint of the execution window, and then propagates this decision to future events in the path. This can result in a tightening of target execution windows for these events.

The function UpdateCurrentState updates the estimate of plant state. The function UpdateControlInputs iterates over each currently executing activity, and computes new plant control inputs by accessing flow tube control policies based on

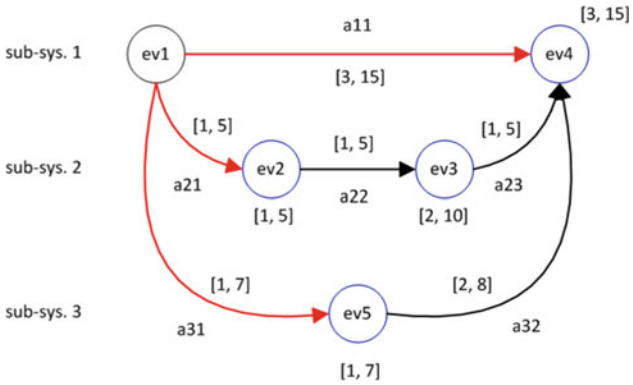


Fig. 10.16 Example Event Horizon. Circles represent events, and horizontal arrows between events represent activities.

the remaining execution time for the activity. The function EventOccurred checks whether an event has occurred, and if so, returns this as an executable event, whose occurrence must be propagated in the next iteration of the main loop in DispatchQCP.

10.6 Experiments

This section describes two types of experiments. The first type involves balancing on one leg. This type of experiment is used to evaluate the DVMC controller. The second type involves more general walking tasks. This type of experiment is used to evaluate the Model-based Executive as a whole, which the DVMC controller is a part of.

10.6.1 Single Support Leg Balance Experiments

A series of tests was performed to test the DVMC controller's ability to restore balance after a disturbance. This series of tests was performed with the humanoid model in single support. Initial conditions were such that the ground projection of the CM was outside the support polygon, and all velocities were set to zero. For such initial conditions, the CM cannot be stabilized by stance ankle torques alone without the foot rolling and the model going unstable. Simple reference trajectories consisting of single, time invariant setpoints were selected for the

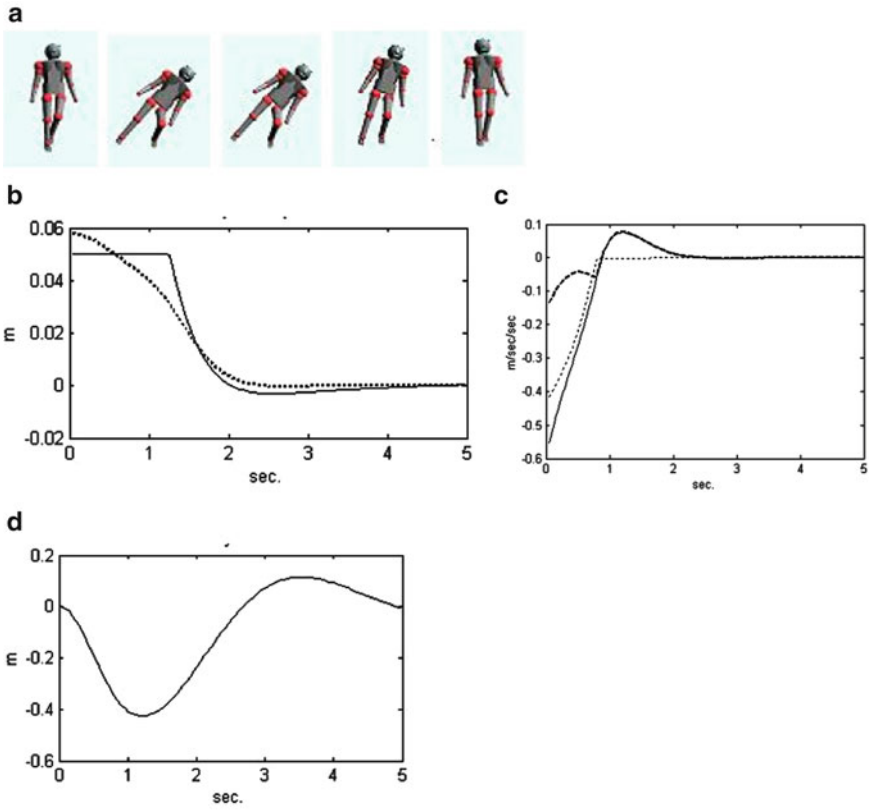


Fig. 10.17 Lateral disturbance recovery. In (**a**), several frames of the model are shown, starting from the maximally displaced CM posture (*left most image*) to the final static equilibrium posture (*right most image*). From the perspective of the model, the right leg is the swing leg and the left the stance leg. In (**b**), the lateral direction CM (*dotted line*) and the ZMP (*solid line*) are plotted versus time. In (**c**), the desired CM acceleration (*solid line*), the actual CM acceleration (*heavy dashed line*), and the slack value (*dotted line*) are plotted, showing the stabilization of the model's CM. Finally the body roll is plotted in (**d**), showing the corrective measures taken by the controller

controller. These setpoints specified the desired equilibrium positions and velocities of the model's COM and swing leg foot. Because the desired final equilibrium posture was to stand on one leg assuming a static pose, all setpoint velocities were set to zero.

Figure 10.15 shows the system's recovery from an initial displacement in the lateral (positive y) direction. From the model's perspective, the left most edge of the foot support polygon is at 0.05 m. As is shown in B, the ZMP remains within the foot support polygon, while the laterally displaced CM position begins outside the stance foot, but is brought quickly to zero by the controller. Part C shows the desired, actual, and slack values for the lateral CM acceleration in the DVMC optimal controller. Note how the slack goes to zero quickly, due to its high penalty. Part D shows the

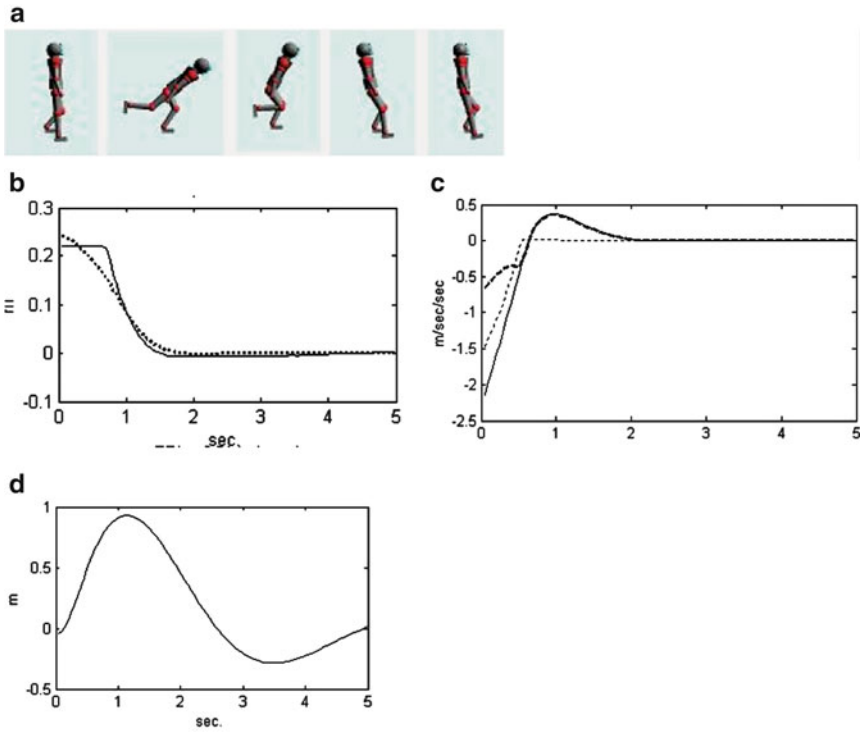


Fig. 10.18 Forward disturbance recovery. In (**a**), several frames of the model are shown, starting from the maximally displaced CM posture (*left most image*) to the final static equilibrium posture (*right most image*). From the perspective of the model, the right leg is the swing leg, left is stance leg. In (**b**), the forward direction CM (*dotted line*) and the ZMP (*solid line*) are plotted. In (**c**), the desired CM acceleration (*solid line*), the actual CM acceleration (*heavy dashed line*), and the slack value (*dotted line*) are plotted, showing the stabilization of the model's CM. Finally, in (**d**), body pitch is plotted

roll angle of the body. Because roll angle is less tightly controlled (penalty on slack variable is less than for CM position), the angle converges, but more slowly than the lateral CM position.

Figure 10.17 shows the system's recovery from a forward initial displacement. The front most edge of the foot support polygon is at 0.22 m. As is shown in B, the ZMP remains within the foot support polygon, while the forward CM position begins outside the foot, but is brought quickly to zero by the controller. Part C shows the desired, actual, and slack values for forward COM acceleration. Note how the slack goes to zero quickly, due to its high penalty. Part D shows the pitch angle of the body. Pitch converges, but more slowly than forward CM position because it is less tightly controlled (Fig. 10.18).

Figure 10.17 shows the system's recovery from a combined forward and lateral displacement,

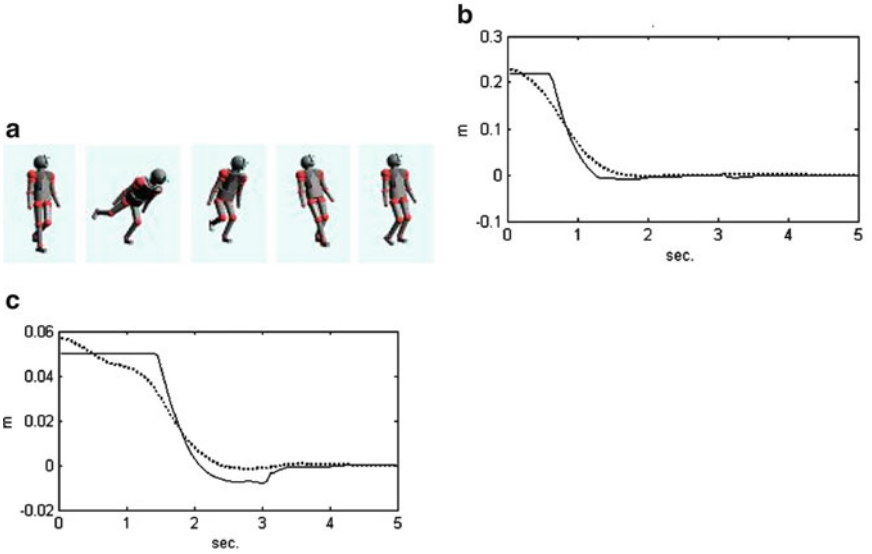


Fig. 10.19 Forward and lateral disturbance recovery. In (**a**), several frames of the model are shown, starting from the maximally displaced CM posture (*left most image*) to the equilibrium posture. In (**b**), the forward direction CM (*dotted line*) and the ZMP (*solid line*) are plotted. (**c**) shows lateral COM and ZMP

The results show that the controller makes appropriate use of non-contact limbs and stance leg ankle torques to stabilize the system. The non-contact limbs are used in two ways: to shift the ZMP, and to shift the CM. Consider, for example, the experiment shown in Fig. 10.17. From the model's perspective, the model stands on its left foot, leaning to the left (positive y direction). If the controller were to take no action, it would tip further to the left and fall down. Due to the action of the controller, the upper body leans further to the left, and the swing leg swings out to the right. Both of these actions correspond, initially, to a negative angular acceleration about the x axis.

The negative angular acceleration about the x axis allows a linear acceleration of the CM to the right (in the negative y direction) while not requiring the ZMP to shift further to the left (positive y direction). This is important since, as shown in Fig. 10.17, the ZMP begins up against the left-most edge of the foot support polygon. As the CM approaches the desired position, the ZMP moves away from the edge and towards the center of the foot support polygon. At this point, the swing leg and body are able to return to their nominal neutral positions.

The lateral acceleration of the swing leg to the right (negative y direction) is also beneficial in that it moves part of the model's mass to the right, and so, helps move the CM in the right direction. The net effect of the swing leg and body movements is an overall angular acceleration at the ankle joint that, together with the action of the stance ankle torque, moves the CM back to the center of the foot support polygon.

The extreme case of non-contact limb movement occurs when the support polygon becomes very small, as is the case for a tight-rope walker. A tight-rope walker's support polygon is very narrow, and therefore, little stance ankle torque can be exerted. Lateral forces by the foot against the tight-rope move the CM, but also create torques of the CM about the contact point. This must be countered by spin angular accelerations (angular accelerations about the CM), so that overall angular momentum is conserved. The spin angular accelerations are generated by movement of the non-contact limbs. Thus, a tight-rope walker extends his arms, and moves his arms, body, and non-contact leg to generate appropriate spin angular accelerations.

An important feature of the controller is that the coordinated behavior of the stance leg and non-contact limbs is not controlled explicitly, but rather, emerges indirectly from a high-level specification of desired behavior. This specification is given in terms of setpoints and PD gains for the CM, body orientation, and swing leg control outputs, in terms of constraints such as the one on the ZMP, and in terms of penalties for slacks and torques in the optimization cost function.

Another important feature of the controller is that, due to its extended range of operation, it can reject significant disturbances more easily than simpler controllers. This feature also means that reference trajectories for the new controller need not be as detailed as those for simpler controllers. The reference "trajectories" for the above tests were single, time invariant setpoints for CM, body orientation, and swing leg outputs. Simpler controllers require more detailed reference trajectories, with more waypoints as a function of time. This extra level of detail puts significant computational burden on the motion planning component of an integrated motion planning and control system. The motion planner has to be executed more frequently, when there are disturbances, and it must produce more detailed reference trajectories.

10.6.2 Biped Walking Tasks

We now present test results of execution of a variety of QCPs for bipedal walking with foot placement and temporal constraints, and with disturbances. Test results for nominal walking at different speeds are provided in [9].

To perform these tests, we used a high-fidelity, 20 degree-of-freedom humanoid simulation to represent the plant being controlled [9, 11]. This simulation accurately models gravity, ground reaction forces and joint torques, and the resulting link acceleration dynamics. In particular, just as with a real biped robot (or a human), this simulated humanoid will fall if inappropriate control commands are provided.

The hybrid plant model (Definition 1) has four modes: left foot single support, double support right foot in front, right single support, double support left foot in front. The mode transition guard conditions are based on toe-off and heel-strike events. The linearized plant abstraction, for each mode, is provided by a feedback linearizing controller [9, 11]. This produces linearized decoupled models for the forward, lateral, and vertical center of mass (CM) components, and stepping foot components. Thus, performing walking tasks involves synchronization of the 3

CM and the 3 stepping foot components when in single support, and the 3 CM components when in double support. For example, when the mode is left foot single support, 6 parallel activities are in the Runtime Activity State, all with right heel strike as the finish event. Each activity must be in its goal region when right heel strike occurs in order for execution to proceed successfully. The event horizon in this case is simple since the parallel activities all share the same finish event.

10.6.2.1 Irregular Foot Placement

Figure 10.20 shows dynamic walking, but with an irregular stepping pattern, which is necessary due to the blocks the biped is walking on. These blocks move slowly, so the timing of foot placement, as well as the positioning is important. At this speed, the biped can't just balance statically on each block. Instead, the moderately fast speed requires dynamic balancing and coordination of the center of mass trajectory. Figure 10.21 shows the CM trajectory and foot placements for this test. The dynamic nature is indicated by the fact that the CM trajectory barely touches the foot placement polygons, and in one case, is 0.1 m away. This indicates that the system is not statically stable in this pose, and is relying on the subsequent foot placement sequence to maintain balance.

10.6.2.2 Lateral Push Disturbances

A biped is especially sensitive to lateral push disturbances when in single support, due to the limited support base provided by one foot. In particular, a biped is most sensitive to lateral push disturbances when there are foot placement constraints, and when the push disturbance results in acceleration towards the outer edge of the stance foot.

Figure 10.22 shows recovery from a lateral push disturbance, while walking on a balance beam. The push occurs from the right side of the biped during left single support. Thus, the push results in an acceleration of the CM to the biped's left. Because foot placement is constrained by the narrowness of the balance beam, compensation by stepping is not an option. Instead, the system compensates for the disturbance by exerting a restoring torque at the ankle. This torque has a significant actuation limit; the lateral center of pressure must not get too close to the outer edge of the foot, or the foot will roll. Because the foot is relatively narrow, this presents a severe actuation limit. Additional (but also limited) compensation is accomplished through the angular movement of the torso and right leg, as shown in the third frame of the sequence [9, 11]. In particular, as shown in Fig. 10.22, the torso rotates clockwise, from the viewer's perspective, which induces a counterclockwise rotation of the stance leg, which, in turn, engenders an acceleration of the biped's CM toward the biped's right.

Due to joint acceleration limits, there is a limit to the angular acceleration that can be produced by the torso and the right leg. Therefore, recovery of lateral balance

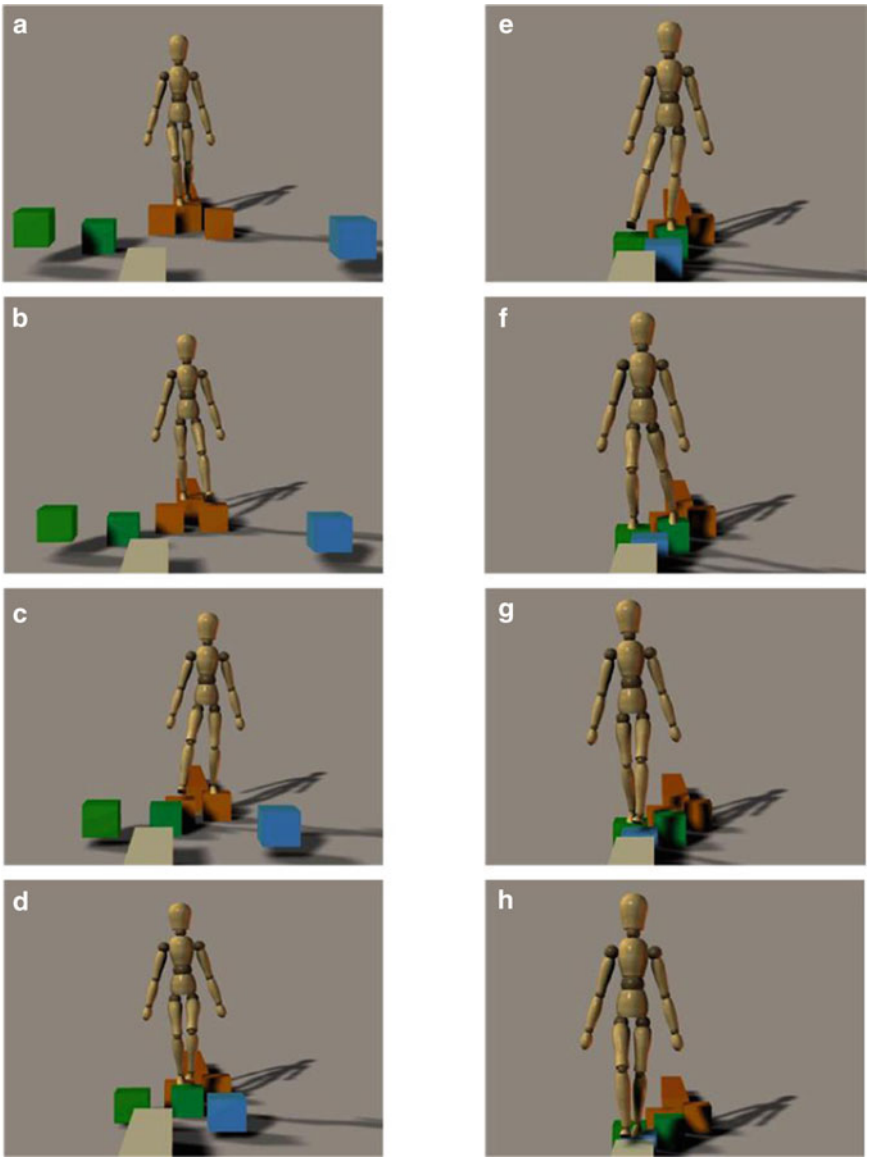


Fig. 10.20 Walking by stepping on slowly moving blocks: (**a**) biped starts on long, narrow path; (**b**) steps with left foot onto the *brown* block; (**c**) steps with right foot onto the other *brown* block; (**d**) steps with left foot onto the *green* block; (**e**, **f**) steps with right foot onto the other *green* block; (**g**) steps with left foot onto *blue* block; (**h**) finished

takes some time; the right leg is out for a significantly longer time (about 2 s) than it would be if it were just taking a normal step. This means that the forward center of

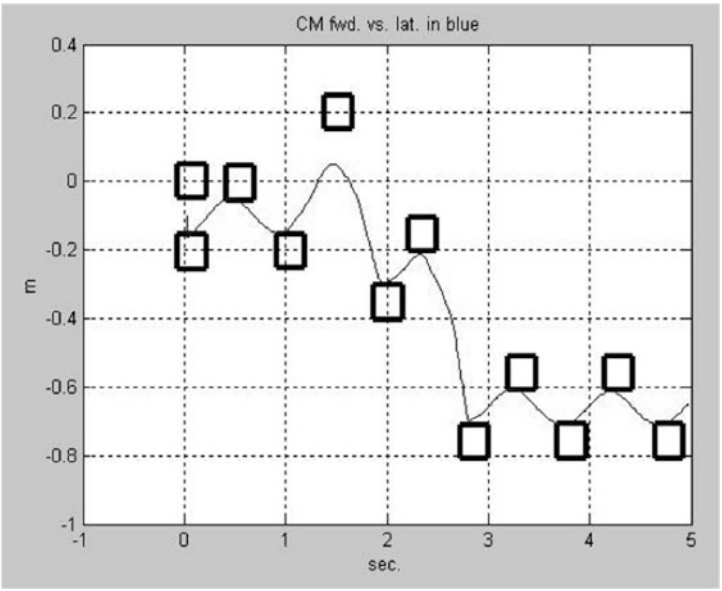


Fig. 10.21 CM trajectory and foot placements for irregular stepping task

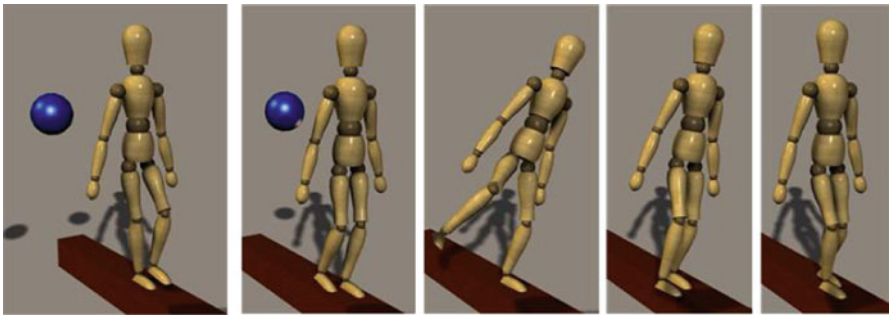


Fig. 10.22 Recovery from lateral push while walking on a balance beam

mass velocity must be reduced while the lateral compensation movement is taking place. This forward velocity reduction must be accomplished by the left (stance) foot alone. Due to support base limitations, there is a limit on the force that can be applied in this way, and therefore a limit to the negative forward acceleration that can be produced. Thus, the biped must be walking relatively slowly, in the first place, for this sort of maneuver to work at all. If this is the case, then forward movement of the CM can be slowed while the right leg is out, and then sped up again after the lateral compensation maneuver is completed. Thus, the forward CM position and the forward stepping position remain synchronized. This is one reason why people tend to walk slowly on tightropes or balance beams.

10.6.2.3 Kicking a Soccer Ball

The problem of moving a biped to kick a soccer ball (Fig. 10.2) requires synchronization of the forward and lateral components of the CM with the step movement, and with the movement of the kicking foot. Figure 10.23a, b show flow tubes and nominal trajectories for the forward and lateral CM components. The flow tubes correspond to CM movement for taking three steps before kicking a soccer ball. Figure 10.23c, d show flow tube cross sections (in the position-velocity plane) for the first activity. The intersection of the initial state with the cross sections determines the duration (temporal) controllability. If the initial state intersects all cross sections, as shown in the figure, then the executive is free to choose any duration in the range covered by the cross sections. This is important for adjusting timing of task completion when kicking a moving soccer ball. Disturbances from the nominal trajectory may restrict the controllable duration range. For example, suppose the system is subjected to a lateral push disturbance at the beginning of the task. This may cause the initial state to deviate from the nominal initial state. If the initial state intersects all cross sections for forward movement, but only a subset for lateral movement, then the overall temporal controllability is limited by the latter set of cross sections.

10.7 Discussion

10.7.1 Scientific Contributions

The techniques described in this chapter extend previously developed temporally flexible execution systems for discrete activity plans to work with hybrid discrete/continuous systems such as bipedal walking mechanisms. This is achieved by first developing a representation for temporally and spatially flexible tasks for hybrid systems, called a QSP, then developing a plan compiler that transforms the QSP to a QCP, which is easier to execute, and then developing a plan dispatcher that executes the QCP. The QCP produced by the plan compiler represents the set of feasible trajectories in an easily executable form. The dispatcher is based on ones developed for discrete activity systems [14], but extends these to allow for indirect scheduling of events through control actions applied to a dynamic system, thus allowing the approach to be used for hybrid systems.

An important attractive property of this approach is that it clearly represents the boundaries between feasible and infeasible states and times with respect to successful plan execution. This allows the system to detect that a plan will fail, sooner rather than later.

It is interesting that traditional control theory does not explicitly address the issue of temporal flexibility. Traditional control theory deals with two basic kinds of problems: stabilization, and trajectory following [18]. Stabilization is an infinite-time

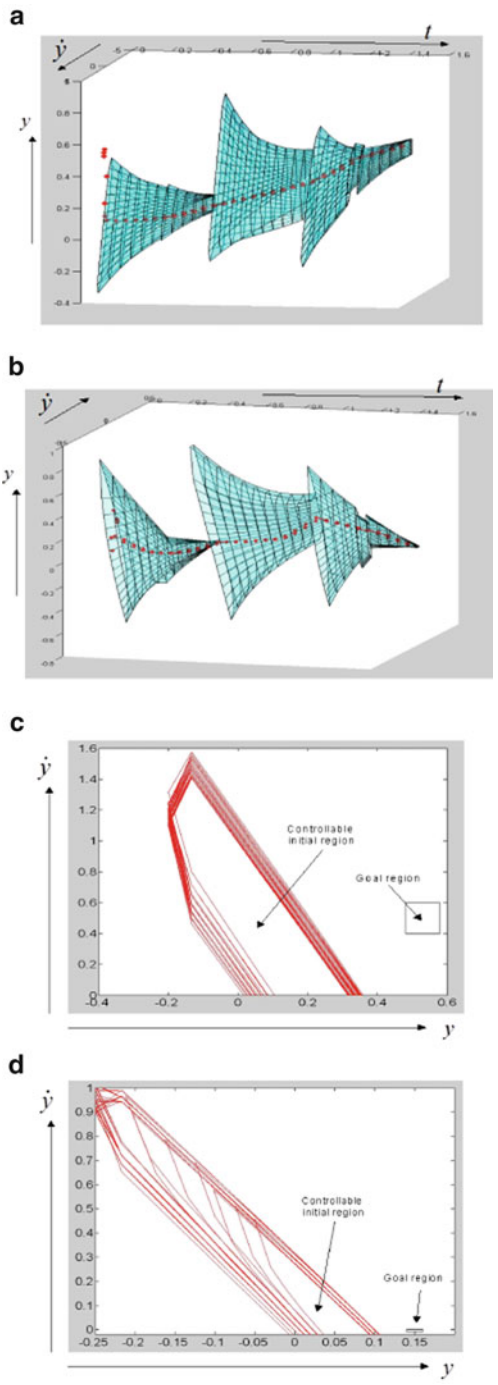


Fig. 10.23 Flow tubes for CM corresponding to taking three steps before kicking a soccer ball: (**a**) flow tubes and nominal trajectory for forward CM component; (**b**) flow tubes and nominal trajectory for lateral CM component; (**c**) flow tube cross sections corresponding to different durations, superimposed, for forward CM component, first step; (**d**) flow tube cross sections for lateral CM component, first step

concept; a system is stable if it converges to an equilibrium point at some time in the future, possibly, infinity. Thus, stabilization has infinite temporal flexibility. Reference trajectory following, on the other hand, has no temporal flexibility. If the control system tracks the reference trajectory exactly, it will reach a goal state at a specific time. We believe that this is a significant omission. Temporal flexibility exists in most task specifications, and should be taken advantage of to achieve robust and efficient plan execution.

To summarize, the system described here takes full advantage of plan specification flexibility, both temporal and spatial, in order to maximize robustness to disturbances. Key contributions are: (1) a plan specification that represents task flexibility; (2) a DVMC that decomposes a complex nonlinear system into a set of loosely coupled linear systems; (3) a plan compiler that transforms the plan, using the abstraction provided by the DVMC, into a form that can be easily executed (QCP); and (4) a model-based executive that uses the flexibility in the QCP to reject disturbances, and to detect when a disturbance is so severe that the plan will fail.

10.7.2 Applications and Impact

The approach described here is intended for inherently under-actuated systems, such as bipeds or aerial vehicles, where there are more degrees of freedom to be controlled than actuators to control them. However, even systems that are fully actuated, such as most robot manipulators, can have actuation limits that become relevant for demanding tasks. For example, moving a robot manipulator at high speeds so that it can perform tasks quickly exposes the velocity and acceleration limits of the joints. Such limits are potentially in conflict with temporal constraints imposed on tasks by the user. Thus, the techniques described here are potentially applicable to any physical system that has velocity and/or acceleration actuation limits, and temporal constraints associated with tasks it is to perform.

References

1. Arakawa, T., Fukuda, T.: Natural motion generation of biped locomotion robot using hierarchical trajectory generation method consisting of ga, ep layers. In: Proceedings of 1997 IEEE International Conference on Robotics and Automation, vol. 1, pp. 211–216. IEEE, New York (1997)
2. Bemporad, A., Morari, M., Dua, V., Pistikopoulos, E.: The explicit linear quadratic regulator for constrained systems. Automatica **38**(1), 3–20 (2002)
3. Craig, J.J.: Introduction to Robotics: Mechanics and Control. Pearson/Prentice Hall, Upper Saddle River (2005)
4. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artif. Intell. **49**(1), 61–95 (1991)
5. Formal'skii, A.: An inverted pendulum on a fixed and a moving base. J. Appl. Math. Mech. **70**(1), 56–64 (2006)

6. Goswami, A.: Postural stability of biped robots and the foot-rotation indicator (fri) point. Int. J. Robot. Res. **18**(6), 523–533 (1999)
7. Hirai, K.: Current and future perspective of honda humamoid robot. In: Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'97), vol. 2, pp. 500–508. IEEE, New York (1997)
8. Hofbaur, M.W., Williams, B.C.: Hybrid estimation of complex systems. IEEE Trans. Syst. Man Cybern. Part B Cybern. **34**(5), 2178–2191 (2004)
9. Hofmann, A.: Robust execution of bipedal walking tasks from biomechanical principles. Ph.D. thesis, Massachusetts Institute of Technology (2006)
10. Hofmann, A., Williams, B.: Exploiting Spatial and temporal flexibility for plan execution of hybrid, under-actuated systems. In: AAAI 2006 (2006)
11. Hofmann, A., Massaquoi, S., Popovic, M., Herr, H.: A sliding controller for bipedal balancing using integrated movement of contact and non-contact limbs. In: Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004), vol. 2, pp. 1952–1959. IEEE, New York (2004)
12. Kajita, S., Matsumoto, O., Saigo, M.: Real-time 3d walking pattern generation for a biped robot with telescopic legs. In: Proceedings 2001 ICRA IEEE International Conference on Robotics and Automation, vol.3, pp. 2299–2306. IEEE, New York (2001)
13. Léauté, T., Williams, B.C.: Coordinating agile systems through the model-based execution of temporal plans. In: Proceedings of the National Conference on Artificial Intelligence, vol. 20, p. 114. AAAI Press/MIT Press, Menlo Park/Cambridge (2005)
14. Muscettola, N., Morris, P., Tsamardinos, I.: Reformulating temporal plans for efficient execution. In: In Principles of Knowledge Representation and Reasoning, Citeseer (1998)
15. Nishiwaki, K., Kagami, S., Kuniyoshi, Y., Inaba, M., Inoue, H.: Online generation of humanoid walking motion based on a fast generation method of motion pattern that follows desired ZMP. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 2684–2689. IEEE, New York (2002)
16. Popovic, M.B., Goswami, A., Herr, H.: Ground reference points in legged locomotion: definitions, biological trajectories and control implications. Int. J. Robot. Res. **24**(12), 1013–1032 (2005)
17. Pratt, J., Dilworth, P., Pratt, G.: Virtual model control of a bipedal walking robot. In: Proceedings of the 1997 IEEE International Conference on Robotics and Automation, vol. 1, pp. 193–198. IEEE, New York (1997)
18. Slotine, J.J.E., Li, W., et al.: Applied Nonlinear Control, vol. 199. Prentice Hall, New Jersey (1991)
19. Sugihara, T., Nakamura, Y., Inoue, H.: Real-time humanoid motion generation through ZMP manipulation based on inverted pendulum control. In: Proceedings of ICRA'02 IEEE International Conference on Robotics and Automation, vol. 2, pp. 1404–1409. IEEE, New York (2002)
20. Vukobratović, M., Borovac, B.: Zero-moment point—thirty five years of its life. Int. J. Humanoid Robot. **1**(01), 157–173 (2004)
21. Vukobratovic, M., Juricic, D.: Contribution to the synthesis of biped gait. IEEE Trans. Biomed. Eng. **16**(1), 1–6 (1969)
22. Vukobratović, M., Stepanenko, J.: On the stability of anthropomorphic systems. Math. Biosci. **15**(1), 1–37 (1972)
23. Williams, B.C.: The use of continuity in a qualitative physics. In: AAAI, pp. 350–354 (1984)
24. Williams, B.C., Nayak, P.P.: A reactive planner for a model-based executive. In: IJCAI, Citeseer, vol. 97, pp. 1178–1185 (1997)
25. Yokoi, K., Kanehiro, F., Kaneko, K., Fujiwara, K., Kajita, S., Hirukawa, H.: A honda humanoid robot controlled by aist software. In: Proceedings of the IEEE-RAS International Conference on Humanoid Robots, pp. 259–264 (2001)