TASK AND MOTION PLANNING WITH BEHAVIOR TREES FOR LOCOMOTION AND MANIPULATION

A Thesis
Presented to
The Academic Faculty

By

Nathan Boyd

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Mechanical Engineering
Department of Mechanical Engineering

Georgia Institute of Technology

May 2022

TASK AND MOTION PLANNING WITH BEHAVIOR TREES FOR LOCOMOTION AND MANIPULATION

Thesis committee:

Dr. Ye Zhao
Department of Mechanical Engineering
Georgia Institute of Technology

Dr. Seth Hutchinson
School of Interactive Computing
Georgia Institute of Technology

Dr. Stephen Balakirsky
School of Interactive Computing
Georgia Tech Research Institute

Dr. Jun Ueda
Department of Mechanical Engineering
Georgia Institute of Technology

Date approved: May 22, 2022

I’m not super. Any talents I have, I worked for – it seems a long time since I thought of myself as a hero.

Oliver Queen

For the friends and family who have supported me throughout my life.

ACKNOWLEDGMENTS

This thesis is the result of almost 2 years of study. Through the many long nights, I have been lucky enough to work with many people to whom I am very grateful. The results of this paper would not have been possible without their support, collaboration, or guidance.

First and foremost, I would like to express my gratitude to my advisors, for their ongoing guidance during the length of my thesis work. Their support as a source of knowledge and resources have been indispensable for the completion of my work. In addition, their passion for the study of robotics has been an ongoing inspiration to my work.

Zhaoyuan Gu, for being an invaluable partner and friend since beginning at the Georgia Institute of Technology. For his great knowledge, enthusiasm, and experience in legged locomotion for developing state-of-the-art systems.

Achintya Mohan, Christopher Lindbeck, Collin Avidano, William Freidank, Iaian Mackeith, Ben Blacklock, and Brad Greer for their commitment to research as undergraduates supporting the work in this thesis. Without them this thesis would not have been possible.

Luke Drnach, Jonas Warnke, Abdulaziz Shamsah, Ziyi Zhao, Yunhai Han, and Max Asselmeier for their friendship as office mates since the very beginning. As well as the many wonderful discussions about life and research through both the joys and sorrow of the M.S path.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Robust and flexible behavior generation for robot task execution remains a difficult problem due to the wide variety of constraints, disturbances, and uncertainties present in the environment. Behavior Trees (BTs) have emerged as a powerful Control Architecture for reliable and flexible autonomous action execution that is capable of authoring complicated logic. More specifically, BTs can execute high level actions based on a tree of composed primitive policies to accomplish a task. This thesis examines the design of two unified robot frameworks that integrate BTs as a robust middleware between high-level decision making as well as low-level motion planning and control. Firstly, a framework for legged locomotion with disturbance rejection is proposed, which incorporates model based trajectory optimization for motion primitive generation and a hybrid dynamic tracking controller for bipedal stability. Each node of the BT is integrated as a single walking step that can be sequentially combined for multi-step locomotion plans with robustness to disturbances. A high level decision maker, Linear Temporal Logic with reactive synthesis, is then used to automatically generate scalable actions with the understanding of potential disturbances. Secondly, a manipulation framework was designed that incorporates a variety of motion primitives with a supporting perception framework for solving generic manipulation problems. At the task planning level, the ROS2 PlanSys2 architecture was used to generate a plan based on BTs and solve multi-step plans. Assembly and disassembly tasks are specifically demonstrated using pick, place, alignment, locking, unlocking, insertion, removal, and many other action primitives for the maintenance of industrial and aerospace environments.

CHAPTER 1
INTRODUCTION

Robots are increasingly being applied to real-life scenarios to provide physical, economic, and societal benefits if deployed correctly. In particular, robots have found a large amount of success in solving dull, dirty, and dangerous tasks that humans would otherwise not want to do. Not only does this benefit workers, it also provides novel solutions for businesses to optimize maintenance and logistics problems. Unfortunately, many of these tasks require well trained manipulation skills or the ability to navigate unstructured terrains. Anthropomorphic robot arms have been used in industry for decades to solve simple manipulation tasks, but it hasn't been until recently with the advent of intelligent perception systems robots could perceive a generic environment and translate it into an action. Additionally, recent advances in the control of legged robots has vastly expanded the ability of robots to explore difficult unstructured environments. However, many of these complex systems require large and flexible frameworks to design appropriate task, perception, planning, and control problems.

At the highest level, a robot must be capable of reasoning about its environment to dictate lower level commands to itself. These tasks can often be decomposed symbolically into a set of steps based on the logic connections of each step. For example, if a robot wants to wash a dish in the dishwasher, it must make sure the washer is open, the rack is pulled out, the dish is unobstructed, and the dish is free of excess residue. However, this plan does not consider the large variety of errors and uncertainties that can arise in this task. Slippery or stuck plates, stuck racks, unidentifiable handles, overly constrained spaces for grasping, and sink water could all cause the robot to fail the task. In the context of dynamic legged locomotion, the task of maintaining balance is critical to avoid a fall. Furthermore, footstep locations may need to be determined based on a series of logical choices. However, legged

robots are expected to operate in highly unstructured environments, where uncertainty and error is prevalent. It is clear that control architectures need to be robust to errors that are guaranteed to arise at a symbolic level. Plans must be malleable and change based on the actual state of the world. AI Planning has traditionally been used to explore this process of solving planning and scheduling problems. However, many AI Planning techniques remain fragile to deviations from the nominal plan. In addition, it is sometimes the fault of a lower-level module like control that causes an error.

Indeed, motion planning and control are essential for the robust execution of tasks. Humans are quick to re-plan if an initial motion plan or control strategy fails. For a robot, this is often analogous to adjusting constraints, gains, or biases to execute the correct action. Recognizing these failures and then knowing which parameters to tune remains a difficult problem. Task and Motion Planning attempts to more tightly couple the control and symbolic aspects of a robotic architecture. By using mutual constraints between the task planning and motion planning, a longer horizon robust motion plan can be generated. The task planning computes the symbolic actions (discrete) and motion planning generates geometric policies (continuous) for control. Unfortunately, these plans are often difficult to solve in a real-time. As a result, they are often incapable of re-planning in an online fashion to handle disturbances or deviations from the original plan.

However, the separation between Task and Motion Planning does not need to be a rigid one. More flexible control framework abstractions can be integrated between the Task and Motion Planning levels to give developers the flexibility to design recovery strategies. Behavior architectures, such as Finite State Machines, Behavior Compositions, and more recently Behavior Trees can be used to help expand the robustness of a robotic platform. In particular, Behavior Trees have shown immense success in the video game industry to design AI agent behaviors and actions that engage users. Many of these strengths transfer to robotic systems and can be leveraged for robust execution.

Recently, robots have demonstrated advanced motion planning, navigation, vision, and

control methods to complete difficult tasks in the real world. For example, the DARPA Subterranian challenge recently showed impressive multi-robot navigation capabilities to search underground and industrial sites for key objects [1]. The challenge stresses that system should be able to encounter a wide variety of problems while operating in the field. Similarly, the Amazon Picking Challenge [2] showed that robot arms could provide generalized solutions to warehouse pick-place tasks. The challenges emphasize the need for a system framework that can handle a variety of different world states with novel planning, perception, and control techniques.

1.1 Thesis Outline

This work proposes two novel robot frameworks that integrate Behavior Trees as the core tool for robust task execution. The first framework shows the integration of Behavior Trees with a bipedal legged robot, Cassie, for disturbance rejection. Where Behavior Trees are automatically generated from Linear Temporal Logic Reactive Synthesis specifications for robust action execution. However, legged robots are difficult to stabilize and require a large amount of low-level motion planning and control to work in tandem with the higher level decision making. Consequently, this work also shows extensive controller, motion planning, and state estimation development to achieve fast bipedal walking. The second framework developed shows the integration of Behavior Trees with a manipulation system execution action sequences with the Planning Domain Definition Language (PDDL). For each PDDL action, a Behavior Tree is used to robustly execute a manipulation task. Furthermore, a database is used to allow continuous and online updates to the knowledge schema of PDDL to handle uncertainties. A perception pipeline is also shown for generating sparse keypoints representing actionable points on objects. These keypoints can then be used as constraints for manipulation problems.

CHAPTER 2

BACKGROUND

2.1 Robust Behavior Authoring with Behavior Trees

This chapter examines the use of Behavior Trees (BTs) as a tool for authoring robust control structures. BTs are one of the more popular architectures to arise recently, especially in computer graphics and gaming [3]. Despite it's success in the graphics community, it must be compared to other common control authoring frameworks that have been traditionally used in robotics. Each control framework has a task execution model associated with it, meaning the sequence of executions and triggers for set of tasks can be represented by a formal model. As a result, a reasonable comparison between approaches can be constructed based on the expected model performance.

2.1.1 Common Robot Control Behavior Architectures

Many trade-offs must be considered to choose an appropriate control architecture for executing resiliant and robust robot behaviors for a task. To this end, many modern control architectures allow for code re-usability, modular designs, closed loop execution, readability, and analysis. Closed-loop execution refers to a robots' ability to execute a sequence of actions in a close-loop fashion. Note that planning can still be conducted offline and open-loop, but the execution should maintain constant feedback. Furthermore, closed-loop execution justifies the potential of synthesizing control architectures, where task planners or machine learning techniques can automatically generate action orders. Principles of re-usability also arise for large codebases and long term projects, where the scaling of a complex system is dictated by the reuse of smaller subsystems. Modular designs thus further the notion of re-usable code by subdividing a system into smaller independent modules.

The rate at which a system can scale is also largely dictated by the time to integrate a new functionality. To this end, human readability/usability refers to the structure of a code-base or tool to minimize developer expertise for a task. An architecture is analyzable if it maintains the ability to extract both qualitative and quantitative properties from the system, such as efficiency and reliability. These aforementioned properties are essential to control architectures and come in a variety of forms.

Finite State Machines

Finite State Machines (FSMs) are one of the most frequently used mathematical models of computation. A FSM constructs a system such that it can only be in one of a finite number of states at a given moment. Provided an event, a FSM could *transition* to another state in response [4]. The FSM in Figure 2.1 demonstrates this property by transitioning between multiple states *exploring*, *tracking*, etc. in order to pick up an object based on the completion of actions and the object state.

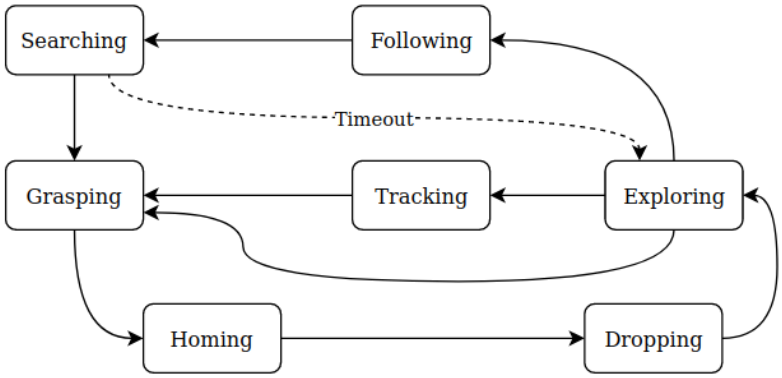

Figure 2.1: Graphical representation of a FSM for a robot that searches a room picking up objects.

FSMs are widely used for a variety of reasons. Most notably, their ease of implementation allows them to be easily implemented in any sequential programming language. In addition, their structure is intuitive and can easily be interpreted by developers to author

new actions.

However, FSM's struggle to scale in an efficient and maintainable way. As the number of states increase, so do the number of transitions between them. Eventually, the transitions become difficult to modify and interpret for both the designer and machine. Adding or removing states also requires the re-evaluation of all the transitions and internal states of the global FSM, making it hard to modify and susceptible to human error.

Hierarchical Finite State Machines

An extension to the classical FSM is the Hierarchical Finite State Machine (HFSM), which eases some of the scaling and re-usability issues associated with FSMs [5]. HFSMs allow for the creation of *super states*, which consist of two or more states in a FSM. Transitions between super states can then be formed to reduce and scope the transitions of the sub-states.

Organizing super states thus allows for so-called *behavior inheritance*, where sub-states can inherit properties from the super state. For example, if a robot is carrying a tool and in the super state *Use Screwdriver*, there could be a sub-state *Align* which requires the robot to align its end-effector in-axis with the objective using one hand. However, a similar super state *Use Powerdrill* could be defined that has the sub-state *Align* requiring the end-effector to be offset by a specified amount and use two hands.

Despite the modularity improvements of HFSMs, they still do not address the issues of maintaining large code behavior structures. Long action sequences still require manual transitions that can backtrack. In addition, HFSM hierarchy is entirely user defined and not always clear.

Behavior Composition

Behavior Composition expands the domain of a controller by composing multiple asymptotically stable controllers. Provided that the goal of each controller is within the region

of attraction of the next controller, the overall basin of attraction for the system can be expanded. This process of layering controller funnels is repeated over local regions of state space until the state enters the domain of a terminal controller that drives the state to a final goal [6]. The most common representation of this process can be seen as a series of Lyapunov functions, where the bottom of each controller funnel feeds into the next (Figure Figure 2.2). Instead of modeling state-independent global dynamics, behavior compositions model state-dependent local dynamics where the regions of state space are handled by models that know how to coax the system into another funnel.

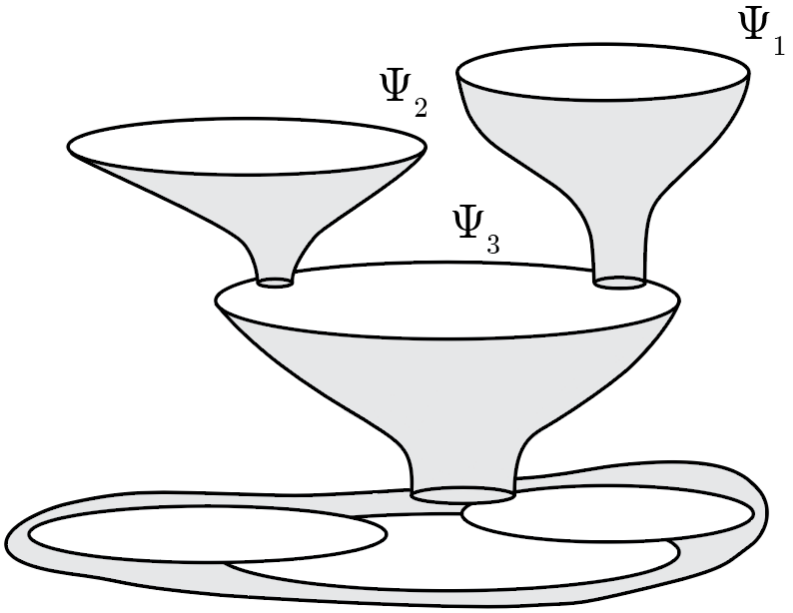

Figure 2.2: A simple visualization of a Composition with three controllers. Each controller has a Lyapunov function that is active if its state is outside the domain of a lower funnel.

Since Behavior Composition is grounded in stability theory, it is rather easy to analyze the effects of sub-tasks on each other. These sub-tasks can also be developed independently and strictly bounded, allowing for modularity between applications. Unfortunately the mathematical notation does come at the cost of inflexbile code and N-dimensional state spaces that are difficult to understand/visualize.

2.1.2 Behavior Trees

Behavior Trees (BTs) were first developed and standardized in the video game industry as an alternative to FSMs. In particular, BTs were used in the control structure of NPCs [3]. Large codebases meant that modularity was a key property for the game industry to enable code reuse, incremental design, and testing. BTs provided an alternative view of FSMs that focused on modularity and hierarchy.

A BT is formally a directed tree where leaf nodes are called *execution nodes* and internal nodes are called *control nodes*. These nodes maintain a standard *parent* and *child* terminology, where the root is a node with no parent and all other nodes only have one parent. Control nodes, on the other hand, can have one or more child below them. A BT begins from the root and generates *ticks*, which get propagated to the child nodes at a stated frequency . Ticks are a signal that specify the execution of a node. For each tick, the child will return a *running* status if its executing a process and a *success* or *failure* depending on the termination of the process. Ticks traverse a tree in a Depth First Search fashion and thus traverse from the left to the right. As a result, a modular tree-structure design provides hierarchical deliberation with both plan extensions and refinement.

While there are only two types of notes (*execution* and *control*), each has multiple node categories. Execution nodes are defined as either *action* or *condition* nodes. Control nodes are defined as either a *sequence*, *fallback*, *parallel*, or *decorator* node. Many implementations of behavior trees expend these four basic control nodes, but we will focus on the classical implementation.

Action: The action nodes executes a command (e.g a motion plan) when it receives ticks. *success* is returned if the action completes successfully or *failure* if the action fails. Ongoing actions return *running*.

Condition: The condition node checks a proposition like an ”if” statement. For example the *GraspOk* condition in Figure 2.4. A condition node returns *success* or *failure* based on if the condition is met. Since conditions are binary (true/false) they do not return

a *running* status.

Sequence: The sequence node is represented by a "⟶" as shown in Figure 2.3 and propagates ticks to its leaves from the left to the right until it finds a leaf that returns either *running* or *failure*, then it returns *running* or *failure* respectively to the parent. However, a sequence node will only returns success if all its children return *success*. If a leaf returns *failure* or *running*, the sequence node will not propagate ticks to the following leaf node.

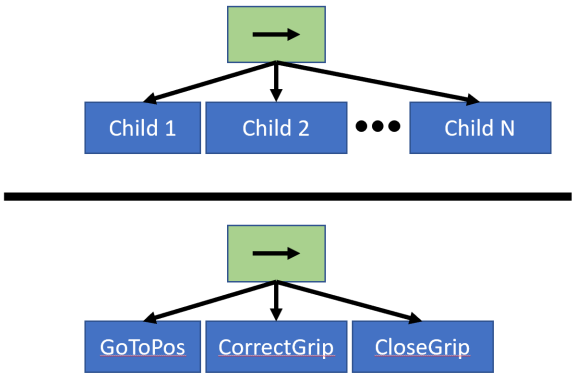

Figure 2.3: (Top) A sequence node (green) with multiple action leaves (blue). (Bottom) This example would execute the following operations in order: 1) go to a desired position 2) make sure the robot gripper is in the correct position 3) close the robot gripper.

Fallback: The fallback node is commonly represented by a "?" as shown in Figure 2.4 and is the core of robust behavior trees. Fallbacks propagate ticks to its leaves from left to right until it finds a leaf that returns either *success* or *running*, which it then echos to its parent. Importantly, a fallback will only return failure if all its leaves return a *failure*. However, if a leaf returns *success* or *running*, the fallback node will not propagate ticks to the following leaf node.

Parallel: The parallel node is represented by a "⇉" as shown in Figure 2.5 and propagates ticks to all its leaves. It returns *success* if K leaves return *success*, but returns failure if $N - K + 1$ leaves return *failure*. Otherwise the parallel node returns *running*. Given that N represents the number of leaves and $K < N$ is threshold designed by the developer.

Decorator: The decorator control node only has a single leaf, which is manipulated to return a modified status and be ticked according to a developer-specified rules. For ex-

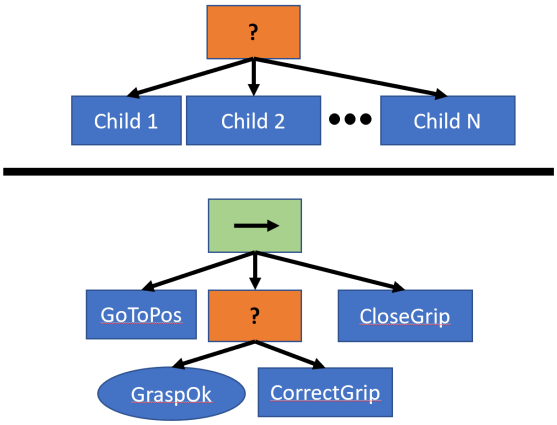

Figure 2.4: (Top) A fallback node (orange) with multiple action leaves (blue). (Bottom) This example would execute the following operations in order: 1) go to a desired position 2) make sure the robot gripper is in the correct position and **only proceed if the grasp condition is valid** 3) close the robot gripper.

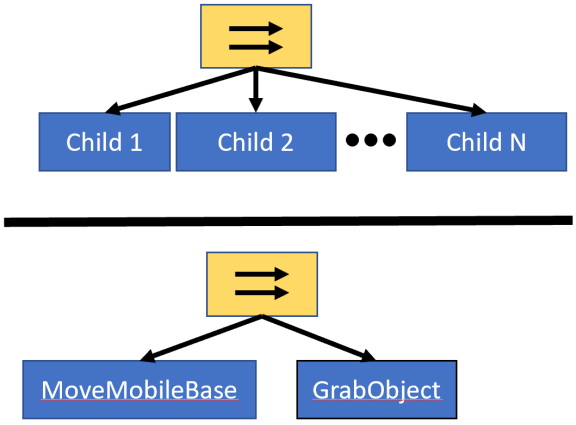

Figure 2.5: (Top) A parallel node (yellow) with multiple action leaves (blue). (Bottom) This example would move both the mobile base and try to grab an object "simultaneously" since the ticks propagate to both of the actions.

ample, an *invert* decorator inverts the *success/failure* status of the leaf. Another common decorator is the *Repeat* node, which only lets its leaf fail M times to succeed before returning *failure* without ticking the leaf. Alternatively, a *RepeatSeconds* decorator allows the leaf run for T seconds to succeed before returning *failure* without ticking the leaf.

2.2 Trajectory Optimization

Optimization has become an essential tool as robots become increasingly reliant on numerical solutions for both online and offline motion planning and control. Most optimization problems are the numerical solution to a problem that minimize a general cost function. The cost dictates the performance of the system and must adhere to a wide variety of constraints. Given a linear or non-linear dynamical system $\dot{x} = f(t, x, u)$, a trajectory optimization (TO) problem solves for a locally optimal trajectory. The resulting solution to the TO problem will provide the state and/or control $x(t), u(t)$ over the entire trajectory. For example, the path a robot end-effector or center of mass follows over time would be the state *trajectory* whereas the actuator torques to create the trajectory is the control. If a trajectory adheres to all specified system constraints it is considered *feasible*. A wide variety of constraints can be considered, such as the system dynamics, collision avoidance, and other boundary conditions. A trajectory is said to be *optimal* if it is the considered to be the best among feasible trajectories. Where the best trajectory is defined based on the cost function of the optimization [7].

TO problems come in multiple forms [7, 8, 9], but this introduction will focus on differentiable continuous-time dynamical systems. To determine the "best" trajectory based on cost function minimization (Equation 2.1), the TO problem will adjust its *decision variables* to find an optimal solution. Total time, torque, acceleration, pose error, etc. are all common costs for trajectory optimization problems.

$$\underset{x(t),u(t)}{\arg\min} \quad \mathcal{L}(x(t), u(t), t) \tag{2.1}$$

As mentioned previously, cost function minimization can be subject to to different constraints. First, dynamic constraints specify how control effort propagates the system state over time (Equation 2.2). For example, a spring-mass motion is constrained by Hooke's law. For many cases, the system dynamics can be simplified to template models to decrease

solve time for a TO problem.

$$\dot{x}(t) = f(x(t), u(t), t) \tag{2.2}$$

Path constraints (Equation 2.3) restrict a trajectory geometrically and is one of widest constraint categories. For example, reachibility or collision constraints are common constraints for constraining the kinematics of a robot to a specific configuration space through time.

$$h(x(t), u(t)) \leq 0 \tag{2.3}$$

Boundary constraints (Equation 2.4) ensure the starting and terminal conditions are correct. Simply stated, they guarantee the solution of the end goal is feasible and reachable. In Equation 2.4, t_0, t_f represent the initial and final time in the optimization.

$$g(x(t_0), x(t_f), t_0, t_f) \tag{2.4}$$

Lastly, the system state (Equation 2.5) and control effort (Equation 2.6) can both be bounded by upper and lower boundary constraints. For example, joint positions and efforts could be bounded by known robot lower (x_{lower}, u_{lower}) and upper (x_{upper}, u_{upper}) state and control limits of the actuators.

$$x_{lower} \leq x(t) \leq x_{upper} \tag{2.5}$$

$$u_{lower} \leq u(t) \leq u_{upper} \tag{2.6}$$

2.2.1 Trajectory Optimization Problem Formulations

While there are a variety of ways to construct a TO problem, the two major methods implemented are *direct* method and the *indirect* method [10]. The primary difference between

the direct and indirect method is the way dynamics are incorporated and what decision variables are selected. The indirect method only uses control inputs, such as torque, as decision variables and does not directly include the system state. Instead, the dynamics state is forward propagated based on the initial state and the control inputs to estimate the entire trajectory of the system. Unfortunately this makes the terminal state heavily dependent on the estimation of the initial state and good tracking control. On the other hand, direct methods incorporate both the state and control as decision variables. As a result, this method can incorporate a large number of decision variables at the cost of having a multiple optimization problems. Since the direct method incorporates the state as a decision variable, it is not sensitive to the initial state of the system like the indirect method. While indirect methods tend to perform better for real-time applications, the performance of both methods is predominantly dependent on the type of problem being solved.

For many problems, the optimization can be decomposed into a set of much smaller problems in a process known as *transcription*. Transcription separates the dynamics of the problem into a finite set of nodes called *knot points*. The continuous time dynamics between knots can be approximated by polynomial splines in a process called *collocation*. Furthermore, constraints are added to the problem to ensure continuity between adjacent polynomials. Each polynomial coefficient and the expected control must then be defined as decision variables for the optimization to solve [11]. As a result, more knot points increase the accuracy of the approximations, but at the cost of computation time. Higher order splines can also be used to increase the accuracy, but this also increases the number of decision variables and computation time. Alternatively, the dynamics can be forward propagated explicitly using integration schemes, such as Runga-Kutta, in a process known as *shooting* similar to indirect methods.

2.2.2 Non-Linear Programming

The formulation of the TO problem dictates what tools and techniques can be used. The structure of the cost function and constraints in particular dictate whether the optimization is considered non-linear. A general formulation for a Non-Linear Program (NLP) where the cost function, inequality constraints, and equalities can all be non-linear functions is shown in Equation 2.7. Where x is the set of decision variables, $\mathcal{L}(x)$ is the cost function, $f_i(x)$ is the i^{th} inequality constraint, and $h_i(x)$ is the i^{th} equality constraint.

$$\begin{aligned} \min_{x} \quad & \mathcal{L}(x) \\ \text{s.t.} \quad & f_i(x) < 0, \quad i = 1, ..., m \\ & h_i(x) = 0, \quad i = 1, ..., n \end{aligned} \tag{2.7}$$

Many commercial NLP solvers exist to solve these problems, such as IPOPT [12], SNOPT [13], and KNITRO [14]. However, most NLP's will only find a local minima instead of the global minimumum for the problem. Additionally, there are no guarantees that a NLP will converge to a valid solution in the first place. Numerical conditioning thus becomes imperative for solving TO problems quickly and accurately in robotics.

2.2.3 Convex Optimization and Quadratic Programming

Mathematical simplifications can also be made to the dynamics and constraints to decrease the computation time of a TO problem. In particular, a convex optimization problem requires that the cost function (Equation 2.7) and inequality constraints are convex functions. The equality constraints should also be affine functions. Compared to a NLP, convex problems typically get solved much more efficiently and are also guaranteed to converge to the global minimum. Software like CVX [15] is readily available to solve convex problems of all varieties.

Quadratic Programs (QP) can be formulated to further simplify the problem and im-

prove the solve times. QP's are a particular type of convex program that take the form of Equation 2.8 where the inequalities are affine and the cost is a quadratic function. There are a multitude of commercial solvers such as Gurobi [16], OSQP [17], and qpOASES [18] that can solve a QP efficiently.

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} x^T R x + c^T x \\ \text{s.t.} \quad & Ax \leq b \end{aligned} \tag{2.8}$$

Additional optimization problem structure can be exploited to help to reduce the time it takes to solve the problem. Costs or constraints in block diagonal, upper triangular, or diagonal matrices can be solved much faster with the use of sparse matrix libraries. Additionally, using previous problem solutions as an initial guess that is close to the optimal solution will also reduce the solve times. This process is known as *warm starting* a problem and it is frequently used in robotics since many control problems require the sequential solving of optimization problems based on feedback [19].

2.3 Dynamic Modeling of Robots

Many optimization problems for multi-body robots rely on accurate kinematic and dynamic models for the system. Systems can modeled with extreme accuracy by accounting for non-linearities such as friction in the joints and the compliance of linkage systems. However, this system identification is very challenging and will often results in optimization problems that can't be solved in a reasonable amount of time. As a result, simplifications to a model are made to provide tractable optimizations problems. These reduced models are called *template models*. Legged robots, in particular use template models to represent the system as a floating mass or inverted pendulum. Control engineering tradeoffs must be made to capture enough of the full dynamics to successfully control the robot.

The interaction of rigid bodies with each other is a well studied subject. A robot arm

or leg can be seen as a chain of coupled rigid bodies with additive properties $[q_1, q_2, ...]$. Additionally, for mobile robots (such as legged robots), a *floating base* model of the robot can be used to represent the moving body as a spatial pose vector $q_b \in \mathcal{R}^6$ with three rotational and linear components. In many cases, position of the floating base is the same as the center of mass for the robot. The complete vector for a generic robot system can be expressed as $q = [q_b, q_1, ..., q_n]^T$, which represents the current pose of the robot base and its joints. The full rigid body dynamic representation of the system can thus be represented by Equation 2.9 [20].

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = J(q)^T\lambda + S\tau \tag{2.9}$$

Where $M \in \mathcal{R}^{n\times n}$, $C \in \mathcal{R}^{n\times n}$ and $G \in \mathcal{R}^n$ are the Mass, Coriolis, the Gravitational matrices. $J \in \mathcal{R}^{mxn}$ represent the jacobian of the multi-body system under the influence of external contact wrenches $\lambda \in \mathcal{R}^m$. The matrix $S^{n\times n}$ is a selection matrix that only selectes commands for actuated joints. Often times, the full-body dynamics of Equation 2.9 can be further simplified by constraining certain joints, such as a pinned point contact dynamic model that assumes no yaw motion at a contact point. A more thorough discussion of rigid body dynamics and its computational algorithms can be found in [21].

CHAPTER 3

TASK, MOTION PLANNING, AND CONTROL FOR BIPEDAL LOCOMOTION

3.1 Introduction

Push recovery is one of the most essential capabilities for legged locomotion. Falling can be considered a critical failure, so implemented methods should be backed by formal guarantees to ensure consistent and reliable locomotion. To this end, this chapter examines a reactive task and motion planning framework resilient to external perturbations [22]. Rejecting perturbations has been widely studied, but the nature of bipedal locomotion incorporates difficult motion planning and control problems for stability. For example, adversarial disturbances and aggressive turning can lead to negative lateral step width (i.e., crossed-leg scenarios) with unstable, underactuated, motions and self-collision risks. In addition to being a computationally difficult motion planning problem, most methods neglect the task planning layer in the context of disturbances. Thus a planning and decision-making framework that closely ties linear-temporal-logic-based reactive synthesis with trajectory optimization incorporating the robot's full-body dynamics, kinematics, and leg collision avoidance constraints was explored. BTs are integrated between the high-level discrete symbolic decision-making and the low-level continuous motion planning to handle disturbance since the Linear Temporal Logic (LTL) decision maker can only contextualize disturbances during apex points in locomotion.

Push recovery for bipedal locomotion has been extensively studied at the motion planning level in previous works and has been inspired by human locomotion biomechanics [23, 24]. Multiple strategies such as ankle, hip, and foot placement methods are proposed to handle external perturbations [25, 26, 27]. However, many of these push recovery strategies have difficulty guaranteeing leg self-collision avoidance because they employ template

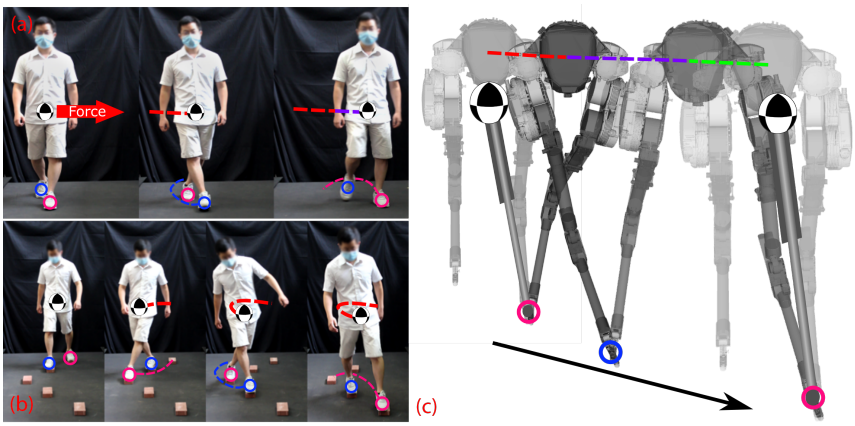

Figure 3.1: a) A Human is forced to cross legs to recover from an external disturbance based on its foot placement. b) The Human is constrained to the stepping stones and must execute a crossed-leg motion plan. c) An illustration of the biepdal robot cassie executing a crossed-leg motion. A Prismatic Inverted Pendulum model is superimposed to show the underlying template model.

models such as centroidal momentum models or inverted pendulum. Since template models do not completely model the kinematics of a legged robot, it makes solving full-leg kinematic constraints difficult. However, it is a strong assumption to state that a robot will never be in close contact with itself in highly dynamic locomotion. Liu et al. [28] demonstrated a control framework that considers self-collision under various disturbances, but only considers a single step and not more difficult multi-step or non-periodic recoveries. Reactive approaches for high dimensional robots have used distance metrics to generate repulsive motions, but this leads to motion plan discrepancies and tracking errors [29, 30, 31]. ”Behavior” or ”Gait” libraries have also been used to generate neutrally stable real-time motion plans in unstructured or constrained environments [32, 33]. However, all these methods fail to also address replanning at the higher, symbolic, level when a disturbance occurs.

Reactive execution is crucial in the context of high-level task planning to account for environmental changes at runtime. The task planning layer is often slow to update plans or only able to consider discrete logic. Temporal-logic-based reactive synthesis [34, 35, 36] has been explored to find strategies that generate formally-guaranteed safe motions with

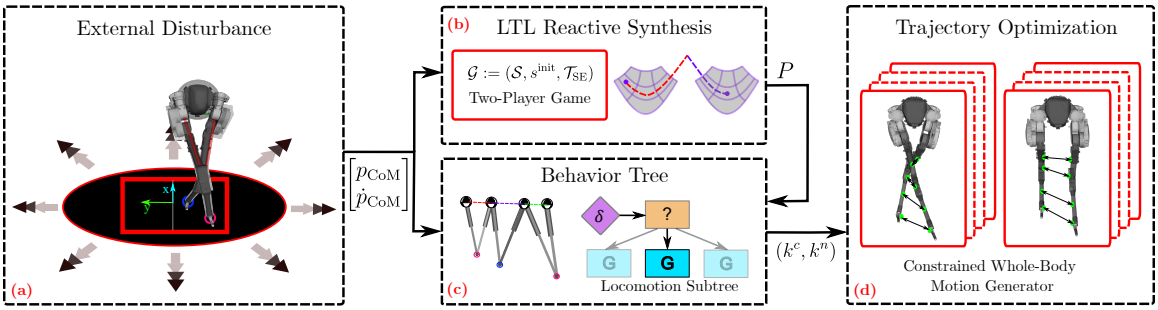

Figure 3.2: Block diagram of the proposed framework. a) Experiments of Cassie disturbed during stable walking; b) The high-level task planner synthesis, employing an LTL two-player game; c) The BTs act as a middle layer that reactively execute subtrees based on real-time environmental disturbances; d) A whole-body motion planner is used to generate feasible motions and refine LTL specifications ψ. The high-level task planner and the phase-space planner are integrated in an *online* fashion as shown by the solid black arrows.

provably correct robot actions, even subject to disturbances. However, reactive synthesis has been under-explored for dynamic locomotion until recent years. LTL [37, 38, 39] has been used to synthesize reactive locomotion navigation plans over various terrains and with obstacles. However, the feasibility of executing synthesized task plans on high degree-of-freedom legged robots when subjected to perturbations remains fairly unexplored. To address this challenge of perturbed task and motion planning for legged robots, this work leveraged trajectory optimization (TO) to verify the synthesized task feasibility.

Due to the hierarchical nature of control frameworks, the motion and symbolic planners often have fragmented understandings of the system state and environment between continuous and discrete modes. Previous methods have explored controller synthesis that can sanction a finite sequence of assumed environmental errors [40], online synthesis of imperfect local strategies that are further patched with lower-level controller [41], and robust automata that are designed with unmodeled disturbances [42]. Unfortunately the discrete symbolic nature of LTL planning also has limitations for real world applications. Since plans exist symbolically, unexpected disturbances at runtime (such as pushes) between discrete plan points or unmodeled errors can dramatically change the execution symbolic plan [43]. This can result in expensive recalculation processes or invisible disturbances at the LTL level.

Behavior Trees (BTs), as mentioned in the beginning of this paper, have been widely explored to author and execute autonomous plans while tracking environmental changes [44, 45]. Their reactive and modular structure can authorize multiple behavioral plans and achieve fault-tolerant task executions [3, 46]. [47] previously devised finite state machine (FSM) controllers for unexpected terrain height variation, but relied on large handmade state machines. Intuitively speaking, BTs can be viewed as a more scalable and acyclic version of FSM for complex behavior execution. It was recently shown that LTL-based reactive synthesis [37] could generate a reactive TAMP with robust reachability analysis for dynamic maneuvers and disturbance rejection. However, it only accounts for perturbations applied at specific apex keyframe instances. These formal methods still have not shown formal safety to perturbations at any locomotion phase. Naturally, this means BTs could be used to handle continuous environmental perturbations by designing actions in a fallback-based structure online to amend the errors in the synthesized discrete automaton.

3.1.1 Contribution

This study addresses the push recovery problem for legged robots subject to external perturbations that can happen anytime. In particular, this work focuses on the coupling of the high-level LTL with the low-level motion planning for reactivity at both layers. A combined TAMP framework composed of hierarchical planning layers operating at different temporal and spatial scales is proposed (Figure 3.2). The LTL planning was tightly coupled with the dynamics of a bipedal robot to design safety-guaranteed decisions at keyframe states. As a result, these plans include Center of Mass (CoM) state or foot placements, in response to the keyframe perturbations. When perturbations occur at non-keyframe instants, analytical Riemannian manifolds are used to recalculate a new keyframe transition online for the current walking step. BTs are integrated to allow updated keyframes to be any continuous value within the allowable range, instead of a finite set of discrete values quantified in the LTL-based planner. Full-body legged motions are generated using kinodynamic-aware TO

for non-periodic multi-step locomotion with self-collision constraints.

3.2 Additional Background

3.2.1 Phase-space Planning

First and foremost, I must further introduce the mathematical concept of reduced-order "template" models for legged robots. The CoM position $p_{\mathrm{CoM}} = (x_{\mathrm{CoM}}, y_{\mathrm{CoM}}, z_{\mathrm{CoM}})^T$ is composed of the sagittal, lateral, and vertical positions in the reference frame of the robot. Likewise, the CoM velocity is defined as $\dot{p}_{\mathrm{CoM}} = v_{\mathrm{CoM}} = (v^x, v^y, v^z)$. Following traditional locomotion models, the apex state of the system is the state at the maximum position p_{apex} in a gait. During bipedal locomotion, the apex state appears once per footstep and is not strictly dependent on periodic motion, thus capturing the complexities of walking gaits. The apex position can be denoted as $p_{\mathrm{apex}} = (x_{\mathrm{apex}}, y_{\mathrm{apex}}, z_{\mathrm{apex}})^T$ with foot placement $p_{\mathrm{foot}} = (p^x_{\mathrm{foot}}, p^y_{\mathrm{foot}}, p^z_{\mathrm{foot}})^T$ and relative CoM height h_{apex}.

The Prismatic Inverted Pendulum Model (PIPM) has been widely used in literature [48]. The PIPM model extends the simple linear inverted pendulum by allowing for a variety of heights in the locomotion plan. Using it's system dynamics, the CoM trajectories can get generated according to the the phase-space plan [49]. For completeness, we summarize the derivation from [48] to justify the analytical solutions used in section 3.3. Consider a single contact case, the centroidal momentum dynamics can be calculated using a simple balance of forces and moments:

$$(p_{\mathrm{CoM}} - p_{\mathrm{foot}}) \times (f_{\mathrm{CoM}} + mg) = -\tau_{\mathrm{CoM}} \tag{3.1}$$

where the forces f_{CoM} and angular moments τ_{CoM} of the modeled virtual flywheel act on the CoM under the gravitational vector g. A nominal locomotion plan maintains $\tau_{\mathrm{CoM}} = 0$, which allows us to formulate the dynamics for a i^{th} walking step walking plan

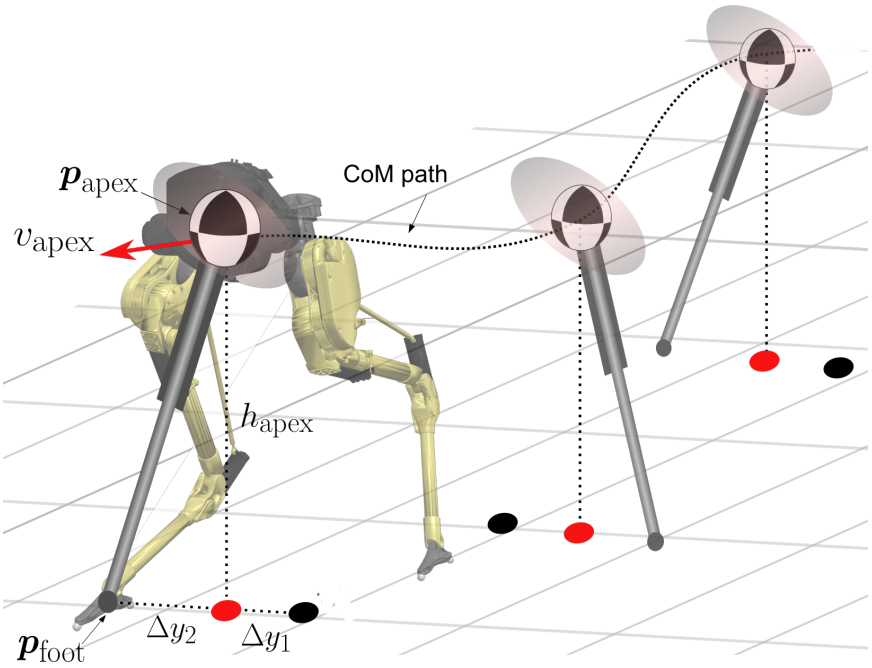

Figure 3.3: A 3D prismatic inverted pendulum template model of the Cassie robot. The lateral distance between the CoM and the feet is expressed as Δy. A negative lateral step width $\Delta y < 0$ implies that the robot is in a crossed-leg scenario.

$$\ddot{p}_{\text{CoM,i}} = \Phi(p_{\text{CoM,i}}, u_{\text{i}}) = \begin{bmatrix} \omega^2_{\text{asym,i}}(x_{\text{CoM}} - p^x_{\text{foot,i}}) \\ \omega^2_{\text{asym,i}}(y_{\text{CoM}} - p^y_{\text{foot,i}}) \\ a\omega^2_{\text{asym,i}}(x_{\text{CoM}} - p^x_{\text{foot,i}}) \end{bmatrix} \tag{3.2}$$

where the asymptotic slope $\omega_{\text{asym}} = \sqrt{g/h_{\text{apex}}}$. The control input $u_i = (\omega_{\text{asym,i}}, p_{\text{foot,i}})$ is calculated based on the position of the feet. h denotes the height of the CoM from the contact points in stance. Furthermore, the template model becomes linear with an analytical solution if the CoM is constrained within the piece-wise linear surface $h_s = a(x_{\text{CoM}} - p^x_{\text{foot}}) + h_{\text{apex}}$ with slope a. The resulting analytical solutions are then used to derive the analytical equations in section 3.3 based on [48].

3.2.2 Linear Temporal Logic Reactive Synthesis

To formally guarantee locomotion task completion under environmental disturbances, our method must use General Reactivity of Rank 1 (GR(1)) [50]. The GR(1) formula is a

fragment of LTL that allows reactive synthesis algorithms to encode a large variety of specifications and provide correct-by-construction guarantees [35, 34, 51].

LTL specifications are temporal and logical relations can be used to represents tasks. The transition system is a discrete description of the system dynamics and environment. A GR(1) formulation supports atomic propositions (APs) that can be True ($\varphi \vee \neg\varphi$) or False ($\neg$True). In addition, these formulas can use logical symbols of negation ($\neg$), disjunction ($\vee$), and conjunction ($\wedge$). Temporal operators such as "next" ($\bigcirc$), "eventually" ($\Diamond$), and "always" ($\Box$) are used as propositional logic extensions. The semantics of LTL can be further explored in [52].

3.3 Planning Methods

In this section, the mid-level motion planning and high-level symbolic decision-making components of the framework (Figure 3.2) are described. The hierarchical framework is composed of (i) LTL-level reactive synthesis for handling perturbations at apex keyframes, (ii) a BT interface that is generated for the robust execution of one walking step (OWS) between keyframe instances, (iii) whole-body motion primitive generation for kinodynamic TO to avoid self-collision.

3.3.1 Keyframe-based Non-periodic Locomotion

To formulate a multi-step bipedal robot walking, the entire trajectory is split into multiple OWS phases that start and end at a keyframe state [53]. The discrete keyframe transition pair (k^i, k^{i+1}) can be used to describe the i^{th} OWS cycle. Each keyframe contains the apex CoM state (sagittal and lateral), as well as the index of the stance foot (subsection 3.3.2). Each OWS has a keyframe transition and maintains a CoM trajectory according to the locomotion dynamics. Forward and backward walking in the sagittal plane uses numerical integration to solve the contact switching time of OWS. Where t_1 and t_2 represent first-half and second-half of the OWS phases, respectively. The next lateral keyframe state can then

be determined using the next sagittal keyframe. In particular, the next lateral keyframe can be calculated by matching the t_1 and t_2 timing constraint, due to the simultaneous contact switch in both directions. Consequently, the lateral keyframe transition and the lateral CoM state $(p_{\text{switch,l}}, \dot{p}_{\text{switch,l}})$ can be calculated at the contact switch instance. Finally, the next lateral foot placement can be computed with the following analytical solution:

$$p_{\text{foot,l}} = p_{\text{switch,l}} + \frac{(e^{2\omega_{\text{asym}}t_2} - 1)\dot{p}_{\text{switch,l}}}{(e^{2\omega_{\text{asym}}t_2} + 1)\omega_{\text{asym}}} \tag{3.3}$$

where w_{asym} and p are defined from Equation 3.2. The subscript l indicates that the foot placement is for the lateral space. As mentioned previously, the concise derivations can be referred to in the work of [53].

Keyframe-base walking allows for non-periodic walking, which is more accommodating to rough terrain and disturbances compared to periodic walking. Thus a keyframe decision-maker is necessary for real-time feasible and safe keyframe transitions.

3.3.2 LTL Specifications for Push Recovery

Given the LTL specification ψ and transition system $\mathcal{T}_{\text{SE}}$, the reactive synthesis problem attempts to find a winning strategy such that the execution path satisfies ψ [39]. Provided that the specification is realizable, the automation will be generated and provide transitions for any environmental actions within the constraint.

Definition 3.3.1 (Riemannian partition)**.** *The transition system discretizes the continuous robot state space (i.e., robot's CoM phase-space near the apex state) into Riemannian partitions defined as:*

$$\mathcal{R} := \mathcal{R}_{\text{position}} \times \mathcal{R}_{\text{velocity}} = \{r_{p,n}, r_{p,z}, r_{p,p}\} \times \{r_{v,z}, r_{v,s}, r_{v,m}, r_{v,f}\}$$

where the relative position of the CoM with respect to the contact feet is represented by the elements in $\mathcal{R}_{\text{position}}$ and CoM apex velocity (zero, slow, medium, fast) is defined as

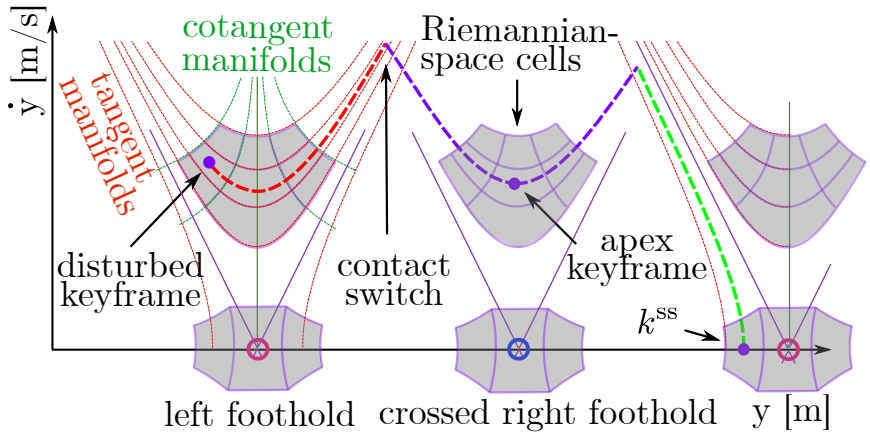

Figure 3.4: An illustration of a phase-space Riemannian partition and lateral keyframe transition for a two-step crossed-leg disturbance recovery.

$\mathcal{R}_{\text{velocity}}$. *To minimize computational complexity, the sagittal and lateral phase-spaces were discretized into 12 Riemannian partitions.*

a disturbed keyframe state $(r_{p,n}, r_{v,m})$ is shown in Figure 3.4, which represents a negative step position and medium velocity to reject the disturbance. A keyframe state whose CoM velocity is zero in sagittal axis is noted as $(r_v)_s = (r_{v,z})_s$. The analytical manifolds of CoM dynamics from the Prismatic Inverted Pendulum Model (PIPM) are used to derive the Riemannian partitions. More details will be introduced in subsection 3.3.5.

Definition 3.3.2 (Locomotion keyframe)**.** *A keyframe $\mathcal{K}$ is defined as a system apex state composed of the sagittal partition $\mathcal{R}_s$, the lateral partition $\mathcal{R}_l$, as well as the foot stance index set $\mathcal{F}_{\text{st}} = \{\textit{left}, \textit{right}\}$ which is used to identify crossed-leg motions or wider lateral step strategies based on the gait.*

$$\mathcal{K} := \mathcal{R}_s \times \mathcal{R}_l \times \mathcal{F}_{\text{st}}.$$

In order to decide the next keyframe state k^n, the system takes an actions $a_{\text{sys}} \in \mathcal{A}_{\text{sys}} \subseteq \mathcal{R}_s \times \mathcal{R}_l \times \mathcal{I} \times \mathcal{W}$. In addition, $\mathcal{I} = \{\textsf{small}, \textsf{medium}, \textsf{large}\}$ and $\mathcal{W} = \{\textsf{small}, \textsf{medium}, \textsf{large}\}$ represent the step length and width. Lastly, $l \in \mathcal{I}$ and $w \in \mathcal{W}$ are the nominal distances between the current and the next foot placements projected on sagittal and lateral axis. Note

that $\mathcal{I}, \mathcal{W}$ represent the nominal global distances between footholds while $\mathcal{R}_s, \mathcal{R}_l$ represent the relative apex states for the CoM in nominal foot frame.

The environment state is represented by a perturbation set $p_{\text{env}} \in \mathcal{P}_{\text{env}} := \mathcal{R}_s \times \mathcal{R}_l \cup \{\emptyset\}$ that pushes the system to the center of a Riemannian cell. In the current implementation, the task planner assumes that the environment will only make perturbations at keyframe instances. Perturbations are seen as near instantaneous actions that cause CoM position and velocity jumps after applying an external force to the robot's pelvis frame. The environment can also choose to do nothing, i.e., $p_{\text{env}} = \emptyset$. Together, the robot action $\mathcal{A}_{\text{sys}}$ and environment action $\mathcal{P}_{\text{env}}$ are used to determine the next apex keyframe state $k^n = \mathcal{T}_{\text{SE}}(k^c, a_{\text{sys}}, p_{\text{env}})$. The combination of these actions are used to define part of the automaton state $\mathcal{S}$.

Definition 3.3.3 (Steady state keyframe)**.** *To further characterize the locomotion of the system, a special set of keyframes are defined as steady state keyframes $k^{\text{ss}} \in \mathcal{K}^{\text{ss}}$ during perturbation-free walking.*

$$\mathcal{K}^{\text{ss}} = \{k^{\text{ss}} | k^{\text{ss}} = \big((r_{p,z}, r_{v,\cdot})_s, (r_{p,\cdot}, r_{v,z})_l, f_{\text{st}}\big)\}$$

where $(r_{p,z}, r_{v,\cdot})_s$ means that the sagittal CoM apex position is above the nominal foot placement and can take any allowable sagittal velocities, while $(r_{p,\cdot}, r_{v,z})_l$ means that the lateral CoM apex position can take any values and the apex velocity has to be zero $r_{v,z}$ (see Figure 3.4).

The system assumes a steady state initial condition $k^{\text{ss}}_{\text{sys}} = ((r_{p,z}, r_{v,m})_s, (r_{p,z}, r_{v,z})_l, \mathsf{right})$. to define the initial state:

$$\begin{aligned} s^{\text{init}} &= (k^{\text{init}}, a^{\text{init}}_{\text{sys}}, p^{\text{init}}_{\text{env}}) \\ &= \big(k^{\text{ss}}, ((r_{p,z}, r_{v,m})_s, (r_{p,z}, r_{v,z})_l), \emptyset\big) \end{aligned}$$

By default, the robot will choose to maintain stable walking unless there is a disturbance

from the environment. Given a perturbation, the keyframe state is required to return to a steady state trajectory within two steps:

$$\square\big(k = \neg k^{\text{ss}} \Rightarrow (\bigcirc k = k^{\text{ss}}) \vee (\bigcirc \bigcirc k = k^{\text{ss}})\big)$$

Two steps are used to consider disturbances in either direction during OWS. If the left foot is in stance and the robot get pushed aggressively to the left, the only feasible motion to stabilize involves a crossed-leg (negative step length) motion. The feasibility of these transitions must be verified by the low-level full-body TO (subsection 3.3.6). However, infeasible task transitions should be removed due to the systems full-body kinematic and dynamic constraints. In this way, the LTL defines a set of TO-refined task specifications. After the TO refinement, all full-body-dynamics-feasible transitions are known offline and encode TO-refined specifications:

$$\begin{aligned}\square\big(k = ((r_{p,z}, r_{v,m})_s, (r_{p,z}, r_{v,m})_l, \mathsf{right}) \Rightarrow \\ a = ((r_{p,z}, r_{v,m})_s, (r_{p,z}, r_{v,s})_l, \mathsf{small}, \mathsf{small}) \\ \ldots \\ \vee a = ((r_{p,z}, r_{v,m})_s, (r_{p,z}, r_{v,m})_l, \mathsf{small}, \mathsf{medium})\big)\end{aligned}$$

Recovering to a steady state k^{ss} in the presence of perturbations requires the next keyframe k^n to minimize the sagittal apex deviation from its expected value and decrease the lateral apex velocity. For example, an instantaneous medium apex velocity would require the next apex velocity to be either medium, small or zero: $\square\big(r_v = r_{v,m} \Rightarrow (\bigcirc r_v = r_{v,m} \vee r_{v,s} \vee r_{v,z})\big)$. In addition, a smaller step width $w \in \mathcal{W}$ will be chosen instead of a larger one: $\square\big((w = \mathsf{small} \vee w = \mathsf{large}) \Rightarrow (\bigcirc w = \mathsf{small})\big)$.To prevent the recovery motion from being interrupted, the robot also assumes the environment perturbation happens at most once per two steps: $\square\big(p_{\text{env}} = \neg\emptyset \Rightarrow (\bigcirc p_{\text{env}} = \emptyset)\big)$

3.3.3 Task Planner Synthesis

Provided the LTL specifications above, the task planner models the interplay between the robot system and the environment as a two-player game. The keyframe transition game structure is constructed in the form of a tuple $\mathcal{G} := (\mathcal{S}, s^{\text{init}}, \mathcal{T}_{\text{SE}})$ with:

- $\mathcal{S} = \mathcal{K} \times \mathcal{A}_{\text{sys}} \times \mathcal{P}_{\text{env}}$ represents the possible automaton state of the transition system,
- $s^{\text{init}} = (k^{\text{init}}, a_{\text{sys}}^{\text{init}}, p_{\text{env}}^{\text{init}})$ is the initial automaton state
- $\mathcal{T}_{\text{SE}} \subseteq \mathcal{S} \times \mathcal{S}$ is a transition describing the possible moves that the robot system can make with the antagonist environment.

Disturbances toward the stance leg (see Figure 3.1) will require the foot placement of the swing leg to move closer to or cross the stance foot. In such extreme cases, a minimum of two steps is required to recover which poses a self-collision challenge for making safe decisions. These dynamics-informed decisions make use of the TO-refined transition specifications to guarantee that the task planner makes valid keyframe transitions. By construction, the TO ensures that all constraints on the full-body motion and the keyframe transitions are feasible. The LTL automaton will select the keyframe transition sequence that goes back to steady keyframe state, which represents the apex state during normal forward walking.

The decision maker uses the current estimated system keyframe state k^c at each instant and plans a sequence of transitions until the final state $k^f = k^{\text{ss}}$. The resulting action roll-out produces the action plan $P = \{k^c, \ldots, k^f\}$.

3.3.4 Behavior-Tree-Based Dynamic Replanning

To address continuous perturbations at non-keyframe instants, the framework integrates a perturbation-aware behavior tree (PABT). The BTs allow local modification of the CoM trajectory online and assign a new keyframe transition. For moderate perturbations, the

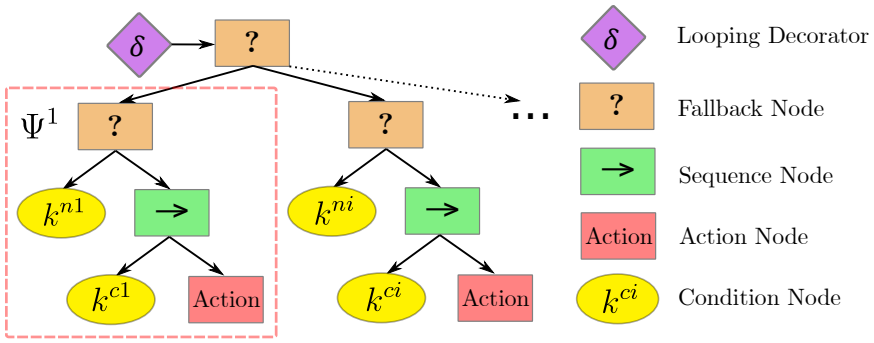

Figure 3.5: An illustration of the PABT structure. The PABT groups a set of locomotion subtrees Ψ^i. Each subtree is a fallback tree that encodes a keyframe transition $(k^{c,i}, k^{n,i})$ and a Riemannian recalculation action.

PABT only complements the previously mentioned reactive synthesis by using the real-time estimated CoM state $(p_{\text{CoM}}, \dot{p}_{\text{CoM}})$ to locally modify the keyframe transitions. In the case of large perturbations, where the perturbation redirects the CoM state to a different Riemannian cell, the PABT recalculates a new transition based on the original keyframe $(k^{c,d}, k^{n,d})$ from the high-level reactive synthesis.

In order to encode this recovery-based planning, the PABT groups a set of locomotion subtrees $\Psi = \bigcup_i \Psi^i$. Each subtree Ψ^i contains a pair of the current-to-next keyframe states $(k^{c,i}, k^{n,i})$. These keyframe pairs are integrated as condition nodes in the locomotion subtrees (Figure 3.5) as pre and post condition requirements. Each locomotion subtree is a fallback BT structure that executes its action nodes when the desired keyframe transition from the high-level matches their condition nodes. More explicitly, the pre-condition and post-condition nodes check if the desired transition $k^{c,d}$ matches with keyframe condition $k^{c,i}$.

After the PABT modification at non-keyframe instances, the desired keyframe transition remains feasible despite the CoM state deviation. This is because the action node A^i can also incorporate a keyframe recalculation procedure. The recovery strategy [53] was used to perform a *Riemannian recalculation*, which replans the motion when the CoM state gets perturbed from the nominal manifold. To ensure the proper constraints are incorporated, the PABT uses a position guard strategy (subsection 3.3.5) to recalculate the keyframe

Algorithm 1: Keyframe Decision Making and PABT Execution

Input: PABT Ψ, Decision-maker DM, current $time$;
Set: status = success;
while *status* == success **do**
  $k^c, time$ = StateEstimation();
  if $time$ == keyframe_instant **then**
    $P = DM(k^c)$;
    for (k^c, k^n) in P **do**
      Ψ^c = LocomotionSubtree(k^c, k^n);
      Ψ.Insert(Ψ^c);
    end
  end
  `/* PABT Riemannian Recalculation */`
  $status$ = Ψ.Tick();
  $(k^c, k^n)'$ = Ψ.GetModifiedTransition();
end
Output: updated PABT Ψ, modified keyframe transition $(k^c, k^n)'$;

state. The modified keyframe state is not guaranteed to be a Riemannian cell center or may end up in a different cell. This shows the recalculation is suitable for continuous perturbation recovery, especially since the synthesized decision-maker only determines the keyframe transition from the Riemannian cell center. As shown in Figure 3.6, the local BT modifications handle the discrepancy between the current (perturbed) CoM state and its Riemannian cell center by recomputing an updated keyframe transition.

The PABT grows as the new action plan P is commanded from the task planner. New subtrees Ψ^c are constructed by the PABT to represent each of the transitions (k^c, k^n) from P. The new subtrees are then inserted as new behavior leaves of the root node. One tick of the PABT thus triggers the corresponding subtree that matches the keyframe conditions. If none of the subtree is able to handle the situation, the PABT will return failure and ask the task planner for a new action plan. The expansion of the PABT and its execution process is illustrated in Alg. algorithm 1.

3.3.5 Riemannian Robustness Margin Design

To quantify the robustness margin of the motion plan, we used the Riemannian distance metric and manifold in [53] to measure the deviation of CoM state from the nominal CoM manifolds in phase-space. Since locomotion operates on in periodic motions, the Riemannian metric discretizes the phase-space with tangent and cotangent locomotion manifolds. This captures the distance error in a discretized curved space instead of a naïve Euclidean-type discretization. Both the tangent and cotangent manifolds comply with the PIPM locomotion dynamics and thus provide an intuitive trajectory recalculation representation for CoM deviations. As shown in Figure 3.4, 12 Riemannian cells in the sagittal and lateral directions partition the CoM state near the stance foothold. The top 9 cells capture non-zero velocities and the bottom 3 cells represent states with zero velocities. Since the apex CoM state is of particular interest for planning, the top 9 cells are centered around the nominal sagittal apex state during walking. The robust margin is represented by the size of each cell, which indicates the amount of deviation the system can handle before switching to an adjacent cell.

Furthermore, the position guard strategy is used to recalculate the next CoM apex state. Assuming the CoM state jumps to $(p'_{\mathrm{CoM}}, \dot{p}'_{\mathrm{CoM}})$ on a new tangent manifold σ', the next CoM apex state is recalculated based on:

$$(p_{\mathrm{apex}}, \dot{p}_{\mathrm{apex}}) = (p_{\mathrm{foot}}, \sqrt{\frac{\dot{p}'^2_{\mathrm{CoM}} \pm \sqrt{\dot{p}'^4_{\mathrm{CoM}} - 4\omega^2_{\mathrm{asym}}\sigma'}}{2}}) \tag{3.4}$$

where $(p_{\mathrm{apex}}, \dot{p}_{\mathrm{apex}})$ corresponds to the next keyframe k^n in the LTL plan. A set of motion primitive then interpolates a full-body motion that connects the current CoM state to the updated next keyframe.

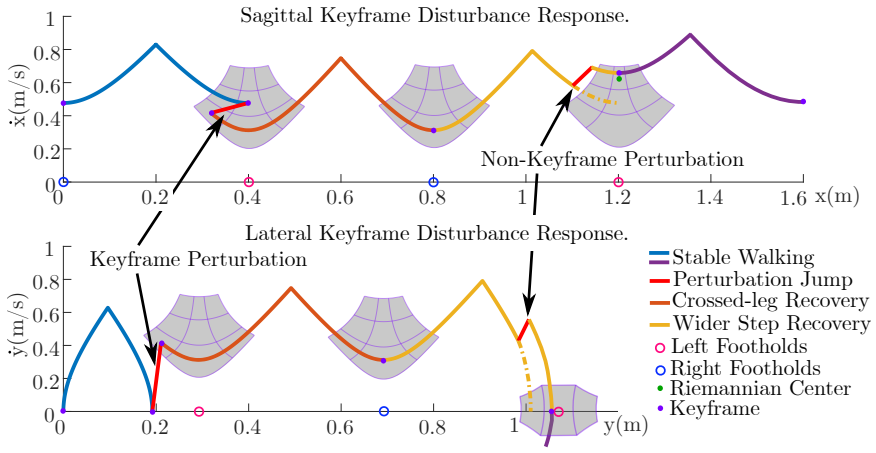

Figure 3.6: Lateral and sagittal responses to diagonal disturbances at keyframe and non-keyframe instants while walking at 0.5 m/s apex velocity. Each color represents a single step generated by the LTL-BT.

3.3.6 Collision-Aware Kinodynamic Trajectory Optimization

While the task planner and PABTs generate keyframe transitions robust to perturbations, they do not consider the full set of constraints acting on the robot. However, mapping the keyframe transitions to whole-body trajectories in real-time often poses a challenge due to computational difficulties. To address this, TO was used to create a set of motion primitives offline in a gait library. Each motion primitive represents OWS trajectories that can be sequentially composed for multi-step walking. The initial state of the motion primitives was varied to also include recovery transitions. The TO generates reference motions that satisfy the physical constraints while minimizing the trajectory cost [54, 55, 56]. Critically, the TO incorporates self-collision constraints and keyframe boundary constraints to guarantee that the optimal full-body motion is feasible. As mentioned previously, the TO is formulation is also used as a verification to check the feasibility of high-level keyframe transitions. The

nonlinear program (NLP) of TO is formulated as:

$$
\begin{aligned}
\underset{X}{\arg\min} \quad & \sum_{j=1}^{D} \sum_{i=0}^{N_j} \Omega_j \cdot \mathcal{L}_j(u_i^j) && (3.5)\\
\text{s.t.} \quad & M_j(q_i^j)\ddot{q}_i + C_j(q_i^j, \dot{q}_i^j) + G_j(q_i^j) = u_i, && \text{(dynamics)}\\
& \dot{q}_0^{j+1} = \Delta_j(\dot{q}_{N_j}^j), && \text{(reset map)}\\
& \lambda_{c,z} \geq 0,\ |\lambda_{c,xy}| \leq \mu\lambda_{c,z}, && \text{(friction)}\\
& E_j^{\text{kin}}(q_i^j) \leq 0, && \text{(kinematics)}\\
& E_j^{\text{col}}(q_i^j) \leq 0, && \text{(self-collision)}\\
& E_j^{\text{key}}(q_i^j, \dot{q}_i^j) = 0 && \text{(keyframe boundary)}
\end{aligned}
$$

where the OWS domain includes $D = 2$ continuous single stance phases and a single velocity reset map. Each stance contains N_j knots, which represents the state-control pair $n_i^j = (x_i^j, u_i^j)$ at the i^{th} instant. The full state vector is $x = [q_b; \dot{q}]$ denoted by the robots floating base q_b and joint states q. The NLP solves the optimal state-control trajectory $X^* = \{n_i^{j*}\}$ by minimizing the pseudo energy cost $\mathcal{L}_j = ||u_i^j||^2$ with weights Ω_j while enforcing the physical constraints of the robot.

The resulting trajectory was shaped by the physical constraints of the system. The system dynamics constraint was enforced between knot points using Hermite-Simpson collocation. M, C, and G denote the Mass, Coriolis, and Gravity matrices of the robot's rigid body dynamics section 2.3. $\dot{q}_0^{j+1} = \Delta_j(\dot{q}_{N_j}^j)$ is a jump map that connects the velocity jump between two consecutive stance modes. Additionally, the a linearized friction cone is used to bound the horizontal contact forces $\lambda_{c,xy}$. The kinematic constraints $E_j^{\text{kin}}(q_i^j)$ ensure that the joint angles, foot positions, and CoM trajectories are bounded. D geometric point pairs (g_l^d, g_r^d) on two legs are selected as self-collision constraints (see Figure 3.2d). The signed distances l^d are evaluated at each geometric point pair using forward kinematics $FK_{g^d}(q_i)$ for all $d \in D$, $i \in N_j$. The minimally allowed distances are thus denoted as $l_{\min}^d$. While

heuristic, the finite set of geometric points is enough to constraint safe trajectories for the swing leg.

$$l^d(q_i) = FK_{g_l^d}(q_i) - FK_{g_r^d}(q_i), \tag{3.6}$$

$$E_j^{\text{col}}(q_i) = ||l_{\min}^d||_2^2 - ||l^d(q_i)||_2^2. \tag{3.7}$$

Lastly, the keyframe transition (k^c, k^n) commanded from the task planner is enforced as a boundary condition for the apex CoM position, velocity, and foot position. However, online motion planning for the footstep planning was found to be more stable based on the angular momentum of the system.

3.4 Tracking Controller Design

The tracking controller is the lowest layer of the control hierarchy for a legged system. While many formulations exist, its objective is to accurately track desired reference trajectories while maintaining robust to errors. For legged robots, these errors appear primarily in the multi-body system dynamics and contact with the environment. When interacting with the environment, it is critical that the robots maintains compliant to account for modeling errors. Recall that the motion planning methods presented previously make a flat world assumption, meaning that any uncertainties in the terrain must be stabilized using the tracking controller.

3.4.1 Feedback Linearized Controller

Initially, the system used a simple feedback linearized proportional-derivative controller with virtual constraints to track reference trajectories. The estimated CoM velocity was used to interpolate gaits from offline generated motion primitive library. Each motion primitive represented a two-step whole-body trajectory. The resulting whole-body trajectory from the motion primitive library was transformed into the desired control variables

$\gamma_d(\tau, q_{\text{pitch}}, q_{\text{roll}}, \alpha)$. Since γ_d represented with virtual constraints (Table 3.1), they needed to be transformed back to the joint reference frames using forward and inverse kinematics. The corrected joint angles were then regulated by a simple PD controller and static gravity compensation.

Following the bipedal control method from [33], the virtual state for the stance and swing leg lengths, swing orientation, swing foot pitch angle, and torso orientation were regulated (Table 3.1).

Table 3.1: Virtual Control Variables with right leg in stance and left leg in swing.

Virtual Constraints Name	Variables in γ_d
Torso Roll	q_{roll}
Stance Hip Yaw	$q_{2\text{st}}$
Torso Pitch	q_p
Stance Leg Length	q_{LLst}
Swing Leg Roll	q_{LRsw}
Swing Hip Yaw	$q_{2\text{sw}}$
Swing Leg Pitch	q_{LPsw}
Swing Leg Length	q_{LLsw}
Swing Foot Pitch	q_{FPsw}

A simple PD control law can then be formulated around the desired virtual states. The following $\widetilde{\gamma}_0$ is ordered such that the first four actuators refer to the stance leg, followed by the swing actuators.

$$\widetilde{\gamma}_0(q) = [q_{\text{roll}}, q_{2\text{R}}, q_{\text{pitch}}, q_{4\text{R}}, q_{1\text{L}}, q_{2\text{L}}, q_{3\text{L}}, q_{4\text{L}}, q_{7\text{L}}] \tag{3.8}$$

$$\widetilde{e} = \widetilde{\gamma}_0(q) - \widetilde{\gamma}_d(\cdot) \tag{3.9}$$

The PD control law, with diagonal proportional and derivative gain matrices $k_{\text{P,fbl}}, k_{\text{D,fbl}} \in \mathbb{R}^{9x9}$, was used to scale the output of the system.

$$u_v = -k_{\text{P,fbl}}\widetilde{e} - k_{\text{D,fbl}}\dot{\widetilde{e}} \tag{3.10}$$

The virtual constraints in $\widetilde{\gamma}_0(q)$ and control u_v were order such that they correspond to the first four actuators of the stance leg and the five actuators on the swing leg. Since the assumed torque on the stance foot is zero, so the virtual constraints are approximately zeroed.

The output of the PD controller will drive the error between the desired CoM and swing trajectories to zero. Simple PD control schemes for multi-body systems has been thoroughly studied in recent years while amassing numerous robustness properties.

3.5 Results

To demonstrate the robustness of the proposed methods, the system have tested various scenarios in simulation using Matlab Simulink and hardware with a bipedal robot, Cassie. The Fast Robot Optimization and Simulation Toolkit (FROST)[55] to solve for the kinematics and dynamics functions of the robot and formulate the direct collocation NLP. Two closed-lop linkage systems are integrated into the leg of Cassie, making it difficult to solve the constrained rigid-body dynamics numerically online. Thus the system dynamics functions generated and solved analytically offline. The NLP solver IPOPT[12] solved the TO problems detailed in Eq. Equation 3.5. The entire framework, together with a virtual constraint controller[33], executed continuously at a rate of 2kHz online. Impulse forces acting on the body were measured through near discontinuous changes in CoM velocity. As for the task planner, the SLUGS reactive synthesis toolbox[57] was used to design LTL specifications with APs and synthesize the keyframe-based automaton.

An ideal tracking controller will perfectly track the periodic CoM profile with clear apex states. The PD controller from subsection 3.4.1 was run on hardware at an average steady state walking speed of 0.4, 0.5, and 0.6 m/s (Figure 3.7). Furthermore, Figure 3.8 shows the acceleration of the robot over multiple steps to a reference steady state walking speed of 0.35 m/s. Thorough hardware testing demonstrated that the method was capable of smoothly transitioning between a standing and walking state with a max steady state walk-

ing speed of 0.7 m/s. The peak-to-valley amplitude of the CoM velocity varies between 0.07 to 0.1 m/s depending on the desired walking speed according to the PSP plan.

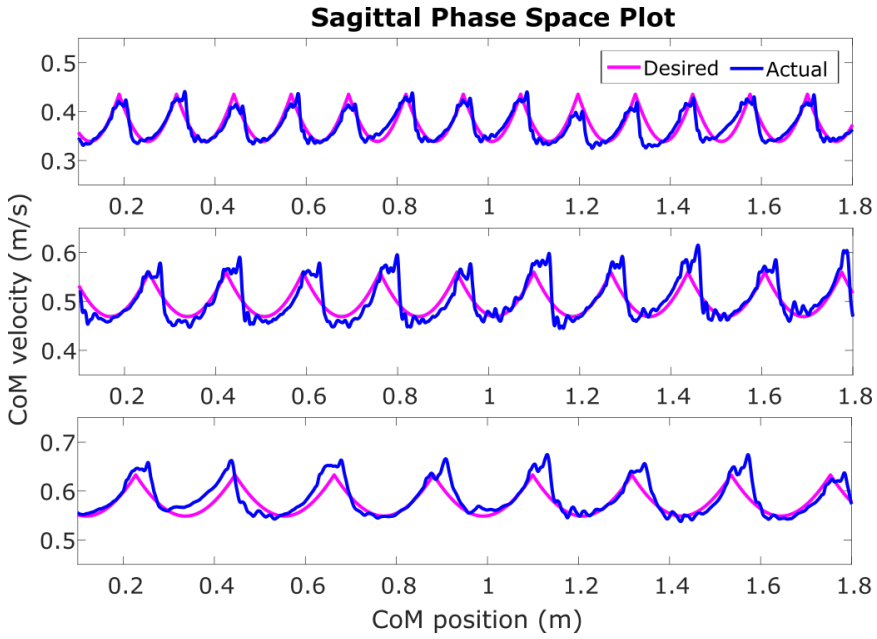

Figure 3.7: PSP trajectory compared to Cassie's actual trajectory.

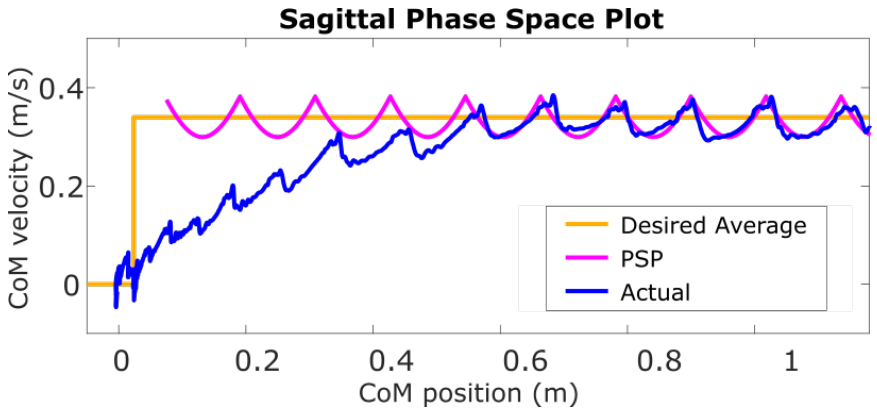

Figure 3.8: Desired average speed of the PSP trajectory compared to Cassie's actual velocity. Step response of Cassie being commanded to walk at 0.35m/s.

For crossed-leg experimentation, 9 riemannian partitions were used with non-zero apex velocities for $\mathcal{R}_s^c$, $\mathcal{R}_l^c$ and $\mathcal{R}_s^n$, respectively. For each (r_s^c, r_l^c, r_s^n) pair, the phase-space planning found the next allowable r_l^n. The resulting Riemannian abstraction thus provided $9 \times 9 \times 9 = 729$ possible crossed-leg transitions prior to the full-body TO. The stopping

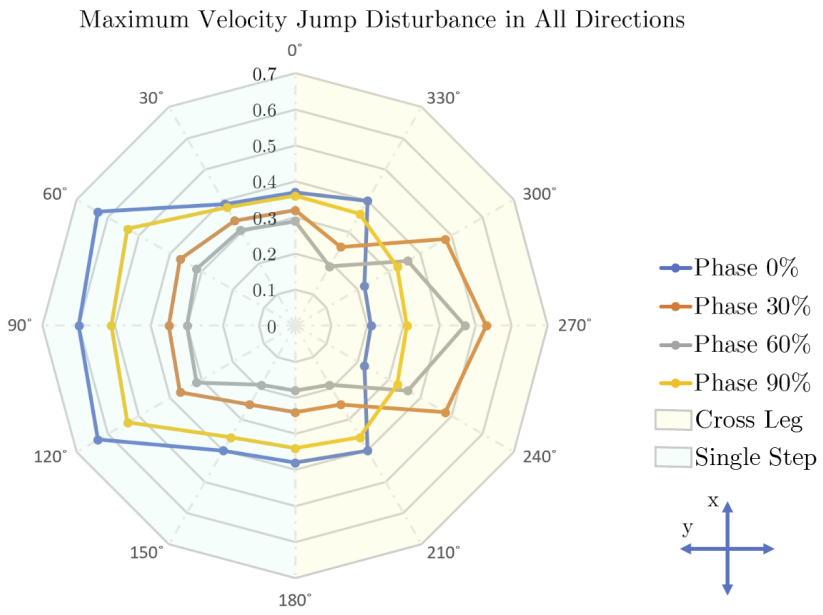

Figure 3.9: Maximum allowable velocity change exerted on the CoM for a single step at 30° increments. Values on the left half resulted in single wider step recoveries and values on the right half require crossed-leg maneuvers.

criteria for the constraints and variable bound violation of the TO were set on the order of 10^{-3}. It was found that 4 pairs of collision points gave safe solutions for self-collision avoidance. Three of the points were located at the middle of shin, tarsus, and toe links. The fourth pair was located at the edge of Cassie's heel spring, as shown in Figure 3.2(d). However future motions may require up to 6 or 8 pairs to increase the density of avoidance constraints. Alternatively, a more formal mesh-based collision method like [58] could be used to certify the motions at the cost of more computation. In total, it took 250 minutes to generate 520 feasible transitions for crossed-leg motions and 5 minutes for 9 wider step recovery motion. Hardware experiments were conducted on the CAREN [59] testing platform to obtain exact disturbance results.

The feasible transitions were evaluated and automatically generated specifications into structured SLUGS files. Where each specification encoded feastible keyframe transitions for the high-level decision making. For steady state walking and wider step recovery scenarios, the lateral r_l^s and r_l^n were the bottom three zero'd Riemannian partitions.

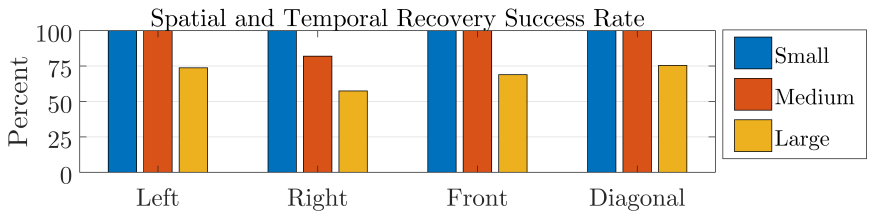

Figure 3.10: Success rate of the recovery motion when a disturbance happens anytime during OWS at multiple directions. Three disturbances were used with a) small 0.1 m/s b) medium 0.2 m/s and c) large 0.3 m/s disturbances

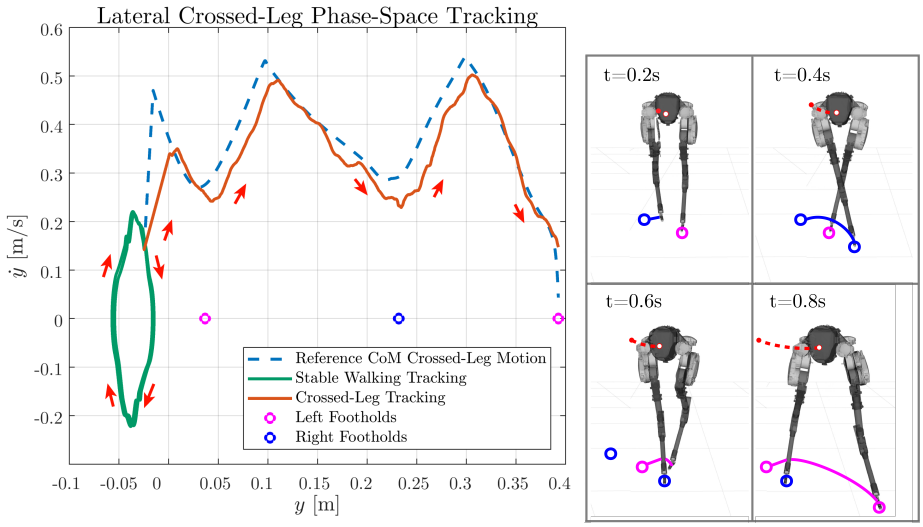

Figure 3.11: Tracking controller results for Cassie executing a 0.4 m/s laterally disturbed leg crossing maneuver from a 0.5 m/s stable forward walking.

The performance of our framework was evaluated through multiple push recovery studies. Pushing tasks were of particular interest since they demand fast responses from the system and often result in large discrepancies between high-level task planners and low-level motion planners. Perturbations were applied once every other walking step and could occur at any phase of OWS. As shown in Figure 3.6, the system was capable of composing multiple OWS trajectories together according to the reactive synthesis plan. This discrete composition of single steps can be seen as somewhat similar to Model Predictive Control approaches, where the system control is forward simulated over a horizon. Cassie was firstly disturbed to the non-apex velocity $(0.4, 0.4)$ m/s at a keyframe instant. Since the perturbation occured at the keframe instant, the keyframe decision-maker planned a

two-step recovery strategy with one crossed-leg step and one wider step to return to a steady state k^{ss}. Disturbances at non-keyframe states required the robot to recalculate a new CoM trajectory to an updated keyframe state. The PABT locally modified the desired keyframe transition and allowed the transitions to begin and terminate in non-Riemannian-cell-centers. The reactive synthesis could still update the keyframe transitions as long as the CoM state was inside the Riemannian robustness bound (the grey areas in Figure 3.6). The PABT thus succeeded in preserving the notion of continuous replanning as opposed to only considering a finite set of discrete keyframe transitions like traditional LTL.

In Figure 3.9,maximum impulse velocity changes that the system could recover from were compared in 12 directions during OWS. The perturbation was measured by velocity change instead of force impulse because the CoM state is measurable, whereas the external forces are difficult to quantify in general. The robot walked sagittally (positive x direction) with an apex velocity of 0.5 m/s. After getting perturbed, the system was allowed to recover using up to two steps. In theory, longer horizon multi-step recovery strategies can be designed to potentially allow the robot to handle larger velocity impulses. However, this study only adopted the two-step recovery strategy because it is the minimum number of steps required for a crossed-leg. Note that the arguement for longer, $N > 2$ step recovery and its validity for disturbance rejection is still disputed. When the push direction was lateral left (positive y), the robot would take a wider step to come back at k^{ss}; otherwise, when the push direction was lateral right, the robot needs to adopt the crossed-leg maneuvers. The perturbations were applied at 4 different phases, with phase $\phi = 0\%$ and 90% closer to keyframe states (boundary phases), and $\phi = 30\%$ and 60% closer to the contact switch phase (50%). The result shows that the phases close to keyframes were better at absorbing large left perturbations. Closer to the contact switch phase, the right side pushes were handled better due to the increased lateral velocity halfway through the step.

What's more, the recovery success rate with 100 trials in 4 directions was tested (Figure 3.11). Diagonal disturbances were applied at 45° from the front to the right. For each

trial, the robot disturbed with 3 instantaneous velocity jumps of $0.1, 0.2, 0.3$ m/s. The perturbations for each trial were evenly spaced ($\phi = 1\%$) for the entire phase duration. It can be seen that failures primarily occurred at the point of maximum velocity for the stance phase (right stance: $\phi \leq 10\%$ and $\phi \geq 90\%$, left stance: $40\% \leq \phi \leq 60\%$). Similar trends were seen in the maximum velocity disturbances (Figure 3.9).

Finally, the tracking performance for the system was evaluated for ± 0.4 m/s lateral disturbances while the left leg was in stance, during a stable walking with 0.5 m/s apex velocity. The positive disturbance forced a two-step crossed-leg recovery due to the stance feet (Figure 3.11) and has a RMS tracking error of 0.0084 m and 0.0593 m/s. For negative lateral disturbances, the system stabilized within 1 wide step, similar to methods like capture point [25] with a RMS tracking error of 0.0039 m and 0.0363 m/s in lateral phase-space.

CHAPTER 4
STATE ESTIMATION AND MOMENTUM CONTROL FOR BIPEDAL LOCOMOTION

4.1 Introduction

Examining the results of Chapter 3, it is evident that the robot was not capable of walking at normal human walking speeds. Since the robot was not capable of walking faster than 0.75m/s in the sagittal direction and also barely able to handle disturbances of half that value. Through extensive hardware testing it was found that the performance limitations of the methods proposed in Chapter 3 were due to a combination of factors at the motion planning and control layers of the framework. First, it should be stated that bipedal locomotion stability is heavily dependent on footstep location [60]. Consequently, it was found that the desired footstep locations determined at the motion planning level were not dramatic enough to fully reject larger disturbances. Even if sufficient footsteps were being planned, the simple feedback-linearized PD controller was not able to respond fast enough to reach the desired location. Lastly, the kinematic estimation of Cassie's CoM state was not accurate enough to respond to disturbances and hybrid impacts at faster speeds.

To get the best performance on hardware with the bipedal robot Cassie, additional development effort was put into the implementation of a floating base Extended Kalman Filter (EKF), Angular Momentum based Linear Inverted Pendulum (ALIP) footstep planner, and passivity-based hybrid dynamic tracking controller. However, the LTL, BT, and Gait Library components of the original framework (Figure 3.2) remained the same. The Extended Kalman Filter provided accurate kinematic estimates for the floating body of Cassie by combining its multi-body kinematics, contact, and inertial proprioceptive sensor information. The ALIP determines robust swing foot positions to accurately reject disturbances.

Footstep prediction was also conducted prior to the collision avoidance trajectory optimization, which meant the robot was still able to prevent self-collision for crossed-leg motions. Tracking these improved footstep locations was also essential to ensure the stability of the system. The feedback-linearized tracking controller was replaced with a hybrid dynamic passivity-based tracking controller that incorporated the natural dynamics of the robot for better performance. Each of these modules were then tied together to create a cohesive hierarchical legged framework with extensive hardware testing (Figure 4.1).

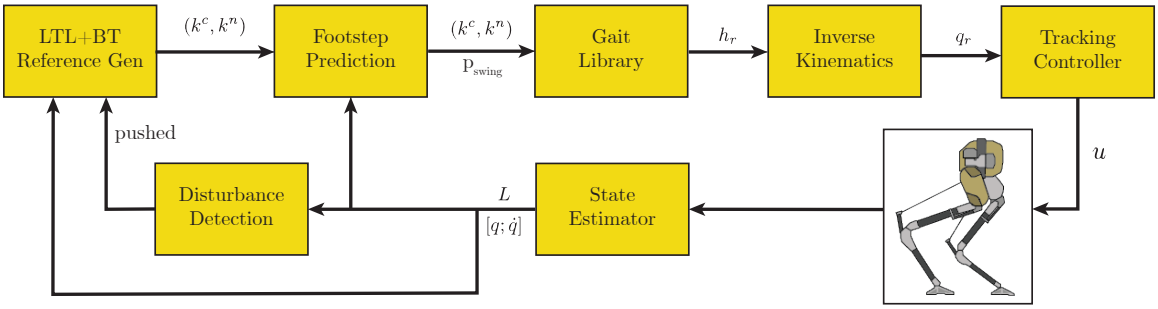

Figure 4.1: Extended system framework that includes the LTL+Behavior Tree task planner with the Extended Kalman Filter and Passivity Controller. The passivity controller could be replaced with either of the controllers seen in section 3.4

4.2 Angular Momentum Based Footstep Planning

Momentum is a core component of locomotion because it encapsulates both the mass and velocity properties of a system. Previously, CoM position and velocity were used as variables for representing the status of how well the system was balancing. Instead, balance could be represented based on the momentum of the system. To this end, the momentum-based footstep planning and control architecture from [61] was implemented using a linear inverted pendulum model (LIPM). The resulting Angular Momentum based Linear Inverted Pendulum (ALIP) control model showed many beneficial properties by representing it in a different reference frame, despite being a template model.

The LIPM model makes many assumptions as a template-model to best estimate the dynamics of the system while maintaining linear control properties. The model assumes a

constant height h, but also maintains a constant center of mass m under the influence of gravity g. Instead of choosing sagittal CoM position and velocity $(x_{\mathrm{CoM}}, \dot{x}_{\mathrm{CoM}})$ as a state, the system state was augmented to use the CoM and y-component angular momentum in the frame of the contact foot (x_c, L_c^y). Where the angular momentum in the contact frame can be represented as $L = (L_c^x, L_c^y, L_c^z)$. In addition, the kinematics of the CoM in the frame of the contact foot are represented as $p_c = (x_c, y_c, z_c)$ for position and $\dot{p}_c = v_c = (v_c^x, v_c^y, v_c^z)$ for velocity. The resulting angular momentum transform, while approximate, has many beneficial properties. First and foremost, momentum incorporates both the mass and velocity during a step to better predict the dynamics of the system. Secondly, momentum about a contact point is invariant to the effects of impacts at that point. This constant momentum property removes the need for the jump map prevalent in velocity models. Removing the need for a jump map dramatically improves the prediction of the system state through contact and mitigates aggressive tracking responses [61].

The LIPM dynamics about the foot contact point can be calculated using the simple dynamic equation:

$$\begin{bmatrix} \dot{x_c} \\ \dot{L_c^y} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{mh} \\ mg & 0 \end{bmatrix} \begin{bmatrix} x_c \\ L_c^y \end{bmatrix} \tag{4.1}$$

From the current state, we can predict the future state at time $\tilde{t}$ before foot contact switch.

$$\begin{bmatrix} x_c(\tilde{t}) \\ L_c^y(\tilde{t}) \end{bmatrix} = \begin{bmatrix} cosh(\omega_l((\tilde{t}-t)) & \frac{1}{mh\omega_l} sinh(\omega_l((\tilde{t}-t)) \\ mh\omega_l sinh(\omega_l((\tilde{t}-t)) & cosh(\omega_l((\tilde{t}-t)) \end{bmatrix} \begin{bmatrix} x_c(t) \\ L_c^y(t) \end{bmatrix} \tag{4.2}$$

where $\omega_l = \sqrt{g/h}$ (note that this is not strictly the apex state), t is the current time and $t \leq \tilde{t} \leq T$, and T is the expected switch contact time. Furthermore, T_k^- and T_k^+ represent the left and right evaluations of the discrete contact instant. The next contact time is represented as T_{k+1}. The prediction for the end of the next step can then used to calculate the next stable footstep location p_{foot}. The dynamics from Equation 4.2 can then be used to estimate the momentum at any time in the horizon. Provided the CoM height

is constant and the ground is flat, the angular momentum at the next contact will be equal to the angular momentum about the current stance leg. As a result, the simple transfer law can be stated through contact:

$$L_c^y(T_k^+) = L_c^y(T_k^-) \tag{4.3}$$

At the beginning of the next step, the angular momentum can be estimated by the angular momentum and position of the center mass relative to the swing foot based on the dynamics (Equation 4.2) and transfer law (Equation 4.3):

$$\begin{aligned} L_c^y(T_{k+1}^-) &= mh\omega_l sinh(\omega_l T) p_{\text{foot}}^x(T_k^-) + cosh(\omega_l T) L_c^y(T_k^-) \\ L_c^y(T_{k+1}^-)(t) &= mh\omega_l sinh(\omega_l T) p_{\text{foot}}^x(T_k^-) + cosh(\omega_l T) L_c^y(t) \end{aligned} \tag{4.4}$$

From here, the longitudinal target step position can be calculated using the closed form solution in Equation 4.6. Where $L_c^{y,d}(T_{k+1}^-)$ can be calculated substituting the desired angular momentum into Equation 4.4. An implementable swing foot expression can then be determined by rearranging the system to include the desired momentum about the next footstep:

$$p_{\text{foot}}(T_k^-)(t) = \frac{L_c^{y,d}(T_{k+1}^-) - cosh(\omega_l T) L_c^y(T_k^-)(t)}{mh\omega_l sinh(\omega_l T)} \tag{4.5}$$

Since angular momentum about the contact point is decoupled between the sagittal and lateral axes, the lateral control can be identical to longitudinal foot placement control.

However, the angular momentum of the next walking step $L^{x,d}(T_{k+1}^-)$ must be known in order to plan footsteps with a non-zero lateral stance width. Assuming an average zero velocity, a simple periodically oscillating LIP model can be sufficient to obtain $L^{x,d}$

$$L_c^{x,d}(T_{k+1}^-) = \pm \frac{1}{2} mhW \frac{\omega_l sinh(\omega_l T)}{1 + cosh(\omega_l T)} \tag{4.6}$$

where W is the desired step width. Note that L is positive or negative depending on the stance leg. The angular momentum in the x-axis can be defined in a similar manner to

Equation 4.1, where $L_c^x = -mgy_c$. Turning is then commanded by assuming a target direction and calculating $L_c^{y,d}$ and $L_c^{x,d}$ in the new desired frame.

Unfortunately for faster walking speeds ($v^x > 1m/s$), the vertical velocity v_{CoM}^z can no longer be assumed to be zero. Thus the vertical velocity in the contact frame v_c^z must be added. Equation 4.6 can be represented for faster speeds with the more general momentum equation:

$$L_c^y(T_k^+) = L_c^y(T_k^-) + mv_c^z(T_k^-)(p_{\text{foot}}^x(T_k^-) - p_{\text{st}}^x(T_k^-)) \tag{4.7}$$

where p_{st} represents the current stance foot position relative to the CoM. As a result, the desired footstep position p_{foot} must be updated to:

$$\begin{aligned} p_{\text{foot}}^x(T_k^-) = \frac{L_c^{y,d}(T_{k+1}^-)}{m(h\omega_l sinh(\omega_l T) - v_{\text{CoM}}^z)cosh(\omega_l T)} - \\ \frac{(L_c^y(T_k^-) + mv_{\text{CoM}}^z(T_k^-)p_{\text{st}}^x(T_k^-))cosh(\omega_l T)}{m(h\omega_l sinh(\omega_l T) - v_{\text{CoM}}^z)cosh(\omega_l T)} \end{aligned} \tag{4.8}$$

Once the analytical foot placement is calculated, its state is used to determine an appropriate full-body trajectory that avoids collision (subsection 3.3.6). The resulting combined motion planning framework preserves the notion of momentum based stability from the LIPM model while maintaining safe trajectory generation for a highliy nonlinear multi-body system.

4.3 Passivity-Based Hybrid Dynamic Controller

While the previous PD controller functions well for low speeds, it does not incorporate the natural dynamics of the multi-body system. Following the method implemented in [61], the passivity-based controller adapted from [62] was applied to a floating base model. First, a more accurate multi-body model of the Cassie robot must be modeled to account for its

dynamics:

$$M(q)\ddot{q} + G(q,\dot{q}) = Bu + J_c(q)^T\tau_c + J_{\text{sp}}^T\tau_{\text{sp}} \tag{4.9}$$

where u is the motor torques, τ_{sp} is the spring torques in Cassie, τ_c is the contact wrench, J_{sp} is the spring jacobian, and J_c is the contact jacobian. The dynamics of system are represented by joint angle q, velocity $\dot{q}$, and accelerations $\ddot{q}$. The Coriolis and Gravity matrices are combined and represented with G and the mass matrix M.

Multiple model simplifications are made to formulate the system's control. A line contact at Cassie's feet during the single support phase do not provide six holonomic constraints and leaves the foot roll as a free degree of freedom (Equation Equation 4.11). In addition, the springs are assumed to be rigid once firmly in contact (Equation Equation 4.10), which provides two constraints per leg. Cumulatively, these simplifications reduce the 20 DoF floating base model to 11 DoF.

$$J_{\text{sp}}\ddot{q} = 0 \tag{4.10}$$

$$J_c(q)\ddot{q} + \dot{J}_c(q)\dot{q} = 0 \tag{4.11}$$

As a result, the full dynamic model for Cassie while in single stance support can be formulated as the following:

$$\underbrace{\begin{bmatrix} M & -J_{\text{sp}}^T & -J_c^T \\ J_{\text{sp}} & 0 & 0 \\ J_c & 0 & 0 \end{bmatrix}}_{\tilde{M}} \underbrace{\begin{bmatrix} \ddot{q} \\ \tau_{\text{sp}} \\ \tau_c \end{bmatrix}}_{\tilde{f}} + \underbrace{\begin{bmatrix} G \\ 0 \\ \dot{J}_c\dot{q} \end{bmatrix}}_{\tilde{G}} = \underbrace{\begin{bmatrix} B \\ 0 \\ 0 \end{bmatrix}}_{\tilde{B}} \tag{4.12}$$

The coordinates can be decomposed into both controllable (actuated), q_a, and uncontrollable, q_u, components $q = [q_a, q_u]^T$. An uncontrolled state vector $\tilde{f}_u = [q_u, \tau_{\text{sp}}, \tau_c]^T$ can

then be used to partition Equation Equation 4.12 into two component equations:

$$\begin{bmatrix} \tilde{M}_{11} & \tilde{M}_{12} \\ \tilde{M}_{21} & \tilde{M}_{22} \end{bmatrix} \begin{bmatrix} \ddot{q}_a \\ \tilde{f}_u \end{bmatrix} + \begin{bmatrix} \tilde{G}_1 \\ \tilde{G}_2 \end{bmatrix} = \begin{bmatrix} \tilde{B}_1 \\ \tilde{B}_2 \end{bmatrix} u \tag{4.13}$$

Solving the above equation, we eliminate the $\tilde{f}_u$ and obtain the following set of partial passive-dynamic equations dictated by the controllable components:

$$\bar{M}\ddot{q}_a + \bar{G} = \bar{B}u \tag{4.14}$$

where

$$\begin{aligned} \bar{M} &= \tilde{M}_{11} - \tilde{M}_{12}\tilde{M}_{22}^{-1}\tilde{M}_{21} \\ \bar{G} &= \tilde{G}_1 - \tilde{M}_{12}\tilde{M}_{22}^{-1}\tilde{G}_2 \\ \bar{B} &= \tilde{B}_1 - \tilde{M}_{12}\tilde{M}_{22}^{-1}\tilde{B}_2 \end{aligned} \tag{4.15}$$

The Passivity-based controller can then be defined with error dynamics $e := q_a - q_r$ based on the actuated joint reference q_r such that

$$\bar{M}\ddot{e} + (\bar{C} + k_{\mathrm{D,pc}})\dot{e} + k_{\mathrm{P,pc}}e = 0 \tag{4.16}$$

where $\bar{M}$ is the augmented Mass matrix and $\bar{C}$ is the augmented Coriolis matrix in $\bar{G}$ such that $\dot{\bar{M}} = \bar{C} + \bar{C}^T$. The PD control gains $k_{\mathrm{P,pc}}, k_{\mathrm{D,pc}}$ and provide compliant tracking to unexpected terrain variations. Reformulating Equation 4.14 with Equation Equation 4.17 provides the following torque command law:

$$u_a = \bar{B}^{-1}(\bar{M}\ddot{q}_r + \bar{G}) - \bar{B}^{-1}(k_{\mathrm{P,pc}}e + (\bar{C} + k_{\mathrm{D,pc}})\dot{e}) \tag{4.17}$$

where u_a is the vector of joint torques. The resulting passivity-based controller incorporates more of the robots natural dynamics. Choosing diagonal gain matrices will also

approximately decouple the tracking errors for Cassie.

4.4 State Estimation

The accurate and real-time acquisition of position, orientation, and velocity of its CoM with respect to local and global frame is essential to stable robot localization, path planning, navigation and controls. A simple approach to estimate CoM position and velocity is to use forward kinematics and foot contact information. Forward kinematics can be used to calculate the relative position and velocity of the CoM with respect to contact foot. This allows for stable control and stabilization within the local frame. However, using the forward kinematics alone poses several issues in estimating Cassie's pose in global frame, which are crucial for motion planning. In particular, the orientation estimate would be subject to significant error from joint measurement errors and the method would fail to estimate any aerial motion. Using an Inertial Measurement Unit (IMU) provides direct orientation and acceleration data, but is subject to sensor drift. For the case of Cassie, a VectorNav VN-100 9-DoF IMU is used for obtaining gyro and accelerometer readings in a global reference frame. The internal state estimator provided by VectorNav, while effective, will still drift and provide noisy estimates due to the highly dynamic and impact driven nature of locomotion. Furthermore, this error in state measurement will accumulate and provide worse estimates for the state of the robot. Simple dead reckoning approaches are thus too inaccurate for bipedal kinematic estimation. Consequently, the errors prevalent in the IMU and kinematic estimates require an additional filter to extract the proper motion profile.

In particular, Hartley et al.'s Contact-aided Invariant Extended Kalman Filtering for legged robot state estimation was implemented because it demonstrated that a Right-Invariant Extended Kalman Filter (RIEKF) formulation was superior to the more commonly used Quaternion Extended Kalman Filter (QEKF) [63]. Both the kinematics of the robot and the IMU are used to help estimate its CoM relative to it's contact points and maintain an

accurate global state estimate. However, the RIEKF approach reformulated the entire state as a single matrix Lie group instead of a decoupled state to formulate the error dynamics that take advantage of symmetry and geometry. The RIEKF also demonstrates a faster convergence rate to the correct estimation and better performance in real-time.

A full derivation of the RIEKF algorithm can be found in [63], but the core of the algorithm is restated here for completeness. The state matrix X_t was defined using state variables R_t, p_t, v_t and d_t representing the time varying IMU orientation, position, velocity, and relative contact points in the world frame respectively. The IMU body frame is not identical to the CoM frame and requires a small transform to correctly represent the CoM frame. $\tilde{\omega}_t$ and $\tilde{a}_t$ represent the IMU angular velocity and acceleration measurements and $\tilde{v}_t$ represents the measured velocity with slippage affected by Gaussian white noise w_t^g, w_t^a, w_t^v respectively. The biased IMU parameters θ_t include angular velocity b_t^ω and acceleration b_t^a, which slowly vary with time according to the standard Brownian motion model such that $\dot{b}_t^\omega = w_t^{b\omega}$ and $\dot{b}_t^a = w_t^{ba}$. h_R denotes the measured orientation of the contact frame with respect to the IMU frame based on joint encoder measurements. The resulting dynamics $\frac{d}{dt}X$ satisfies the group affine property for log-linear error dynamics.

$$X_t \triangleq \begin{bmatrix} R_t & v_t & p_t & d_t \\ 0_{1,3} & 1 & 0 & 0 \\ 0_{1,3} & 0 & 1 & 0 \\ 0_{1,3} & 0 & 0 & 1 \end{bmatrix}, \tag{4.18}$$

$$\frac{d}{dt}X_t = \begin{bmatrix} R_t(\tilde{\omega}_t)_\times & R_t\tilde{a}_t + g & v_t & 0_{3,1} \\ 0_{1,3} & 1 & 0 & 0 \\ 0_{1,3} & 0 & 1 & 0 \\ 0_{1,3} & 0 & 0 & 1 \end{bmatrix} - X_t \begin{bmatrix} (w_t^g)_\times & w_t^a & 0_{3,1} & h_R w_t^v \\ 0_{1,3} & 1 & 0 & 0 \\ 0_{1,3} & 0 & 1 & 0 \\ 0_{1,3} & 0 & 0 & 1 \end{bmatrix} \tag{4.19}$$

$$\frac{d}{dt}X \triangleq f(X_t) - X_t w_t \tag{4.20}$$

The deterministic system dynamics $f(X_t)$ satisfies a group affine property to make the error dynamics right-invariant. Based on this affine property, the log-linear right-invariant error dynamics can be linearized on the exponential map of the Lie group [63]. The error dynamics of the robot is calculated based on the estimated dynamics with bias parameters $f(\bar{X}_t, \bar{\theta}_t)$. Where $\bar{R}_t$, $\bar{p}_t$, $\bar{v}_t$ and $\bar{d}_t$ are the kinematic state estimate and the IMU measurement estimates are calculated with $\bar{\omega}_t \triangleq \tilde{\omega}_t - b_t^{\omega}$ and $\bar{a}_t \triangleq \tilde{a}_t - b_t^{a}$ based on the bias terms, thus make the dynamics parameter dependent.

$$f(\bar{X}_t, \bar{\theta}_t) = \begin{bmatrix} \bar{R}_t(\bar{\omega}_t)_\times & \bar{R}_t\bar{a}_t + g & \bar{v}_t & 0_{3,1} \\ 0_{1,3} & 0 & 0 & 0 \\ 0_{1,3} & 0 & 0 & 0 \\ 0_{1,3} & 0 & 0 & 0 \end{bmatrix} \tag{4.21}$$

The continuous right-invariant error dynamics is defined as A_t and calculated as in [63] with covariance matrix $\bar{Q}_t$ based on the white noises of the IMU gyro and accelerometer. The estimated state tuple is predicted with the set of differential equations $\frac{d}{dt}(\bar{X}_t, \bar{\theta}_t) = (f_u(\bar{X}_t, \bar{\theta}_t), 0_{6,1})$. $Adj_{\bar{X}_t}$ represents the matrix adjoint of the estimated state variables from Equation 4.18. The noise vector $w_t \triangleq [w_t^{\omega}, w_t^{a}, 0_{3,1}, h_R w_t^{v}, w_t^{b\omega}, w_t^{ba}]$ is augemented to also include the bias terms.

$$A_t = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -\bar{R}_t & 0 \\ (g)_\times & 0 & 0 & 0 & 0 & -(\bar{v}_t)_\times \bar{R}_t & -\bar{R}_t \\ 0 & I & 0 & 0 & 0 & -(\bar{p}_t)_\times \bar{R}_t & 0 \\ 0 & 0 & 0 & 0 & 0 & -(\bar{d}_t)_\times \bar{R}_t & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.22}$$

$$Q = \begin{bmatrix} Adj_{\tilde{X}_t} & 0_{12\times 6} \\ 0_{6\times 12} & I_6 \end{bmatrix} Cov(w_t) \begin{bmatrix} Adj_{\tilde{X}_t} & 0_{12\times 6} \\ 0_{6\times 12} & I_6 \end{bmatrix} \tag{4.23}$$

During the *prediction stage*, the state estimate, $\bar{X}_t$ is propagated based on the system dynamics (Equation 4.22) and the covariance matrix (Equation 4.23). The covariance matrix P_t is updated using the Ricatti equation (Equation 4.24) according to [64].

$$\frac{d}{dt}P_t = A_t P_t + P_t A_t^T + \bar{Q}_t \tag{4.24}$$

The linear correction stage of the RIEKF can then be completed using the kalman gains K_t, estimated forward kinematics to the foot h_p, IMU biases $\bar{\theta}$, and contact jacobian J_c. The measurement Y_t, output H_t, and noise $\bar{N}_t$ matrices are first calculated:

$$Y_t^T = \begin{bmatrix} h_p^T & 0 & 1 & -1 \end{bmatrix} \tag{4.25}$$

$$H_t = \begin{bmatrix} 0 & 0 & -I & I & 0 & 0 \end{bmatrix} \tag{4.26}$$

$$\bar{N}_t = \bar{R}_t J_c Cov(w_t^\alpha) J_c^T \bar{R}_t^T \tag{4.27}$$

The gain K_t is composed of the dynamics gain K_t^ξ and IMU bias gain K_t^γ and computed as follows:

$$S_t = H_t P_t H_t^T + \bar{N}_t \tag{4.28}$$

$$K_t = \begin{bmatrix} K_t^\xi \\ K_t^\gamma \end{bmatrix} = P_t H_t^T S_t^{-1} \tag{4.29}$$

An auxiliary selection matrix $\Pi \triangleq [I \quad 0_{3,3}]$ was introduced to then select the appropriate state variables when correcting the system as in [65]. The corrected state tuple and

covariance are then corrected through the update equations:

$$\begin{bmatrix} \bar{X}_i^+ \\ \theta_i^+ \end{bmatrix} = \begin{bmatrix} exp(K_t^\xi \Pi \bar{X}_t Y_t)\bar{X}_t \\ \bar{\theta}_t + K_t^\gamma \Pi \bar{X}_t Y_t) \end{bmatrix} \tag{4.30}$$

$$P^+ = (I - K_t H_t)P_t(I - K_t H_t)^T + K_t \bar{N}_t K_t^T \tag{4.31}$$

The RIEKF will try to estimate the kinematics of the robot floating base and prevent IMU drift that will cause positioning error. This state estimator is purely proprioceptive because it only assumes inertial, contact, and encoder measurements as inputs to the model. While variants of the RIEKF do exist that include perception for Simultaneous Localization and Mapping (SLAM), it was not necessary and would require much more system development with a supporting computer to process the camera or LIDAR inputs [63]. SLAM is more relevant when discussing navigation and mapping problems rather than disturbance rejection.

4.5 Results

4.5.1 RIEKF Estimation Performance

The primary objective of the RIEKF was to to estimate the correct orientation, position, and velocity of the CoM in a global reference frame. The IMU is located in the robot's torso and provides measurements at 800Hz. Additionally, Cassie provided joint angle measurements from the encoders at 2kHz. Contact was detected by measuring the deflection in the pair of springs on each leg to estimate a force. To verify the method, Cassie was run on a treadmill at a constant sagittal velocity of 0.45 m/s for 47 seconds. The resulting RIEKF results were then compared to the unfiltered VectorNav IMU results.

First, the estimated orientation needed to correct for sensor drift in the IMU and incorrect measurements due to the hybrid nature of contact. The IMU already has a low gyro and accelerometer bias noises of 5 deg/hr and 0.04 mg due to its internal orientation esti-

mation [11]. While CAssie is standing, the pure orientation measurements from the IMU remain accurate, but this stability doesn't hold when the robot begins walking. The impact driven nature of locomotion caused large errors in the pitch and roll axis using just the unfiltered VectorNav estimator. By incorporating the full state of the system, the RIEKF was able to quickly converge to the ground truth orientation in both simulation and hardware Figure 4.2.

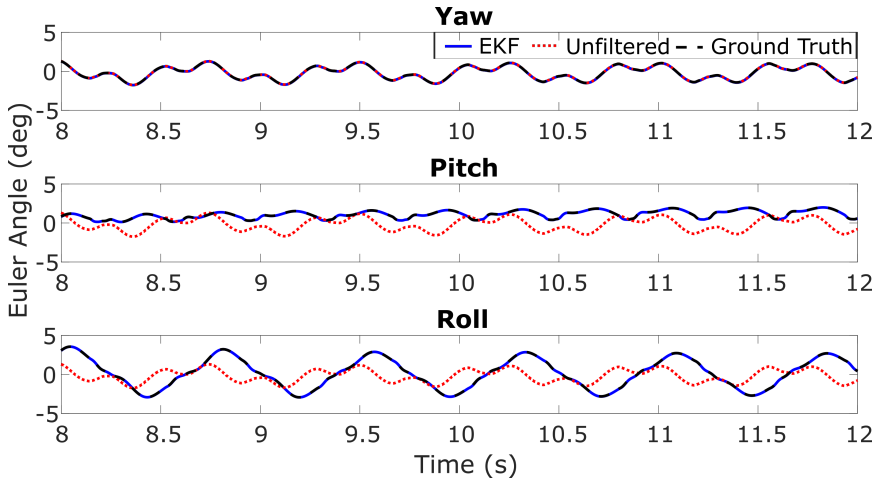

Figure 4.2: The yaw, pitch, and roll angle tracking error was addressed to track the ground truth.

The nature of contact causes instantaneous velocity jumps in the floating base of a legged robot. As a point of comparison, the noisy velocity profile was filtered out by the first-order filter that was incapable of recovering the correct velocity profile. Figure 4.3 clearly shows that the first-order method over-filtered the noisy contact dynamics. By applying RIEKF, the velocity profile was able to observe the desired phase-space profile to a much higher resolution. The unfiltered IMU data from the VectorNav clearly detects the moments of impacts, but the signal is so noisy it is difficult to construct a control signal from.

Lastly, velocity estimates from the unfiltered kinematics and first order filter were used to estimate Cassie's position in the world frame using dead reckoning. The RIEKF would estimate it's position in the world frame internally Figure 4.4. In particular, The xy-world

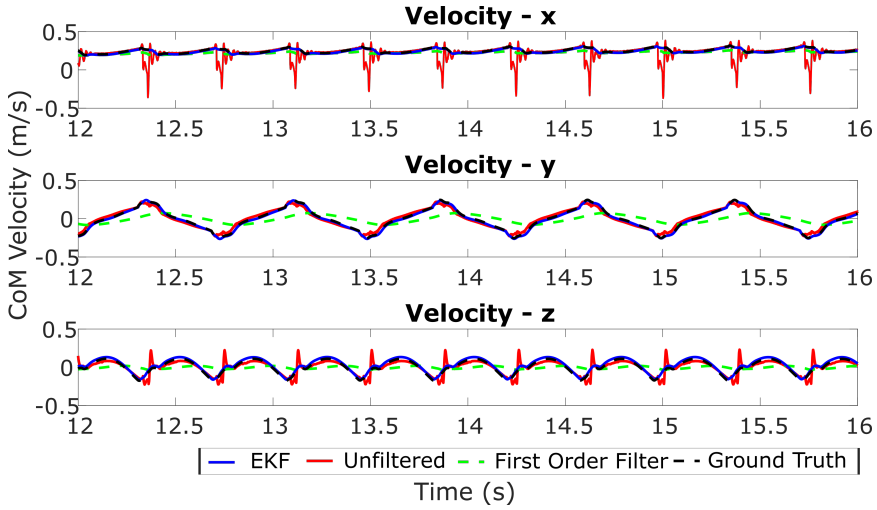

Figure 4.3: The velocity comparison between noisy unfiltered measurement, first order filter, RIEKF, and ground truth.

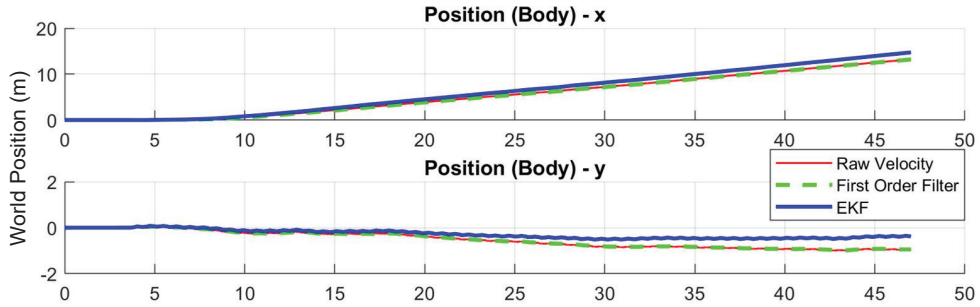

Figure 4.4: Position estimation in the world frame based on the kinematics of the robot. The raw velocity and first order filtered velocity estimate of the IMU were used to dead reckon a position. The trivial estimation results are compared to the RIEKF position, which correctly tracks the position of the robot in the world coordinate frame.

frame is of the most interest since it is subject to the most estimation drift. This is largely due to the fact that the IMU z-axis is inline with the gravity vector. After the designated time, it can be seen the RIEKF corrects the resulting bias and drift in the position estimation to maintain the correct position estimate. Meanwhile the unfiltered and first-order filter approaches accumulated over 1m of drift in both the x-axis and y-axis.

4.5.2 Comparison of PD and Partial Feedback Linearization with ALIP Footstep Placement

The Passivity-based controller (section 4.3) and ALIP footstep planner (section 4.2) allowed for significant improvement in the performance of the system. The footstep prediction from the ALIP was used to generate stable footsteps, which were then used as constraints for the gait library interpolation. Once a safe gait policy gets calculated, the passivity controller tracks the desired swing and stance trajectories. Both the ALIP and gait library interpolation were executed online at 2kHz.

It can be seen in Figure 4.5 that the ALIP footstep placement significantly increased the walking speed of the initial feedback-linearized PD controller. The ALIP allowed Cassie to walk in a stable fashion up to 1.5m/s (double its previous speed without ALIP). However, recall that the desired motion profile of the bipedal system in steady state is a periodic trajectory between defined apex keyframes. The velocity of the PD controller did not track the expected apex-to-apex motions in the sagittal plane after 1m/s due to the uncompensated dynamics. However, the passivity-based controller showed far better tracking results and was able to walk up to 2m/s on flat ground while maintaining an apex-to-apex motion profile in the CoM sagittal velocity (Figure 4.5).

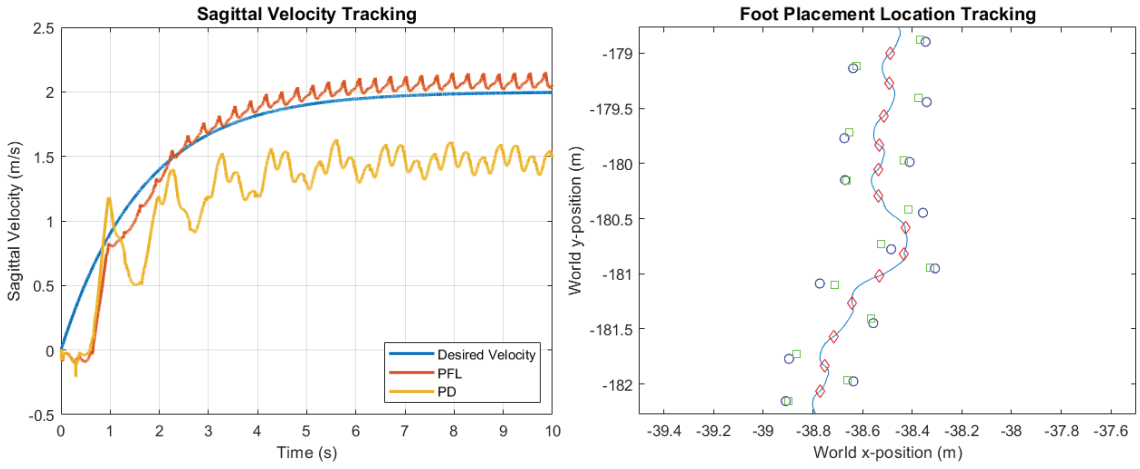

Figure 4.5: (Left) Comparison of the Partial Feedback Linearization and PD Control in the Sagittal axis while tracking a 2m/s desired speed. (Right) Footstep placement and tracking accuracy for multiple steady-state steps and a lateral disturbance

With the passivity controller, Cassie was also able to consistently track the desired footsteps within a few centimeters. Even when disturbed, it can be seen that the footstep swing was still able to accurately track target values through a crossed-leg maneuver Figure 4.5. From experimental testing, footsteps were within 0.2m of the desired location even when disturbed, despite this accuracy is not formally guaranteed.

4.5.3 Validation on the CAREN Testbed

Lastly, Cassie was tested on the CAREN testing platform and subject to a variety of omni-directional disturbances. The robot was tasked with maintaining a 0.7 m/s sagittal walking speed and would receive a 0.7 m/s magnitude disturbance at 30 degree angle increments between 0 and 360 degrees. Disturbances were triggered at $\phi = 50\%$ swing phase during the right leg swing based on the performance shown previously in Figure 3.9. Phases were tracking based on the measured contacts of the robot on the treadmill.

The passivity-based tracking controller and ALIP planning improvement continued to show the best results on hardware. At 0.7 m/s, the system was able to successfully recover from every disturbance and would only fail if it accidentally stepped off the treadmill. Figure 4.6 shows 90 degree increments of the full testing to illustrate it's principal angle disturbance responses. The platform motion would move in the opposite direction that Cassie would measure due to the relative motion between the CoM and the platform. A left perturbation for CAREN thus meant a right wide step response from Cassie. The most extreme cases were a right perturbation causing a crossed-leg maneuver and a backward perturbation that caused Cassie to take a large forward step to stabilize. Unfortunately the CAREN platform was incapable of reaching larger disturbances due to hardware limitation.

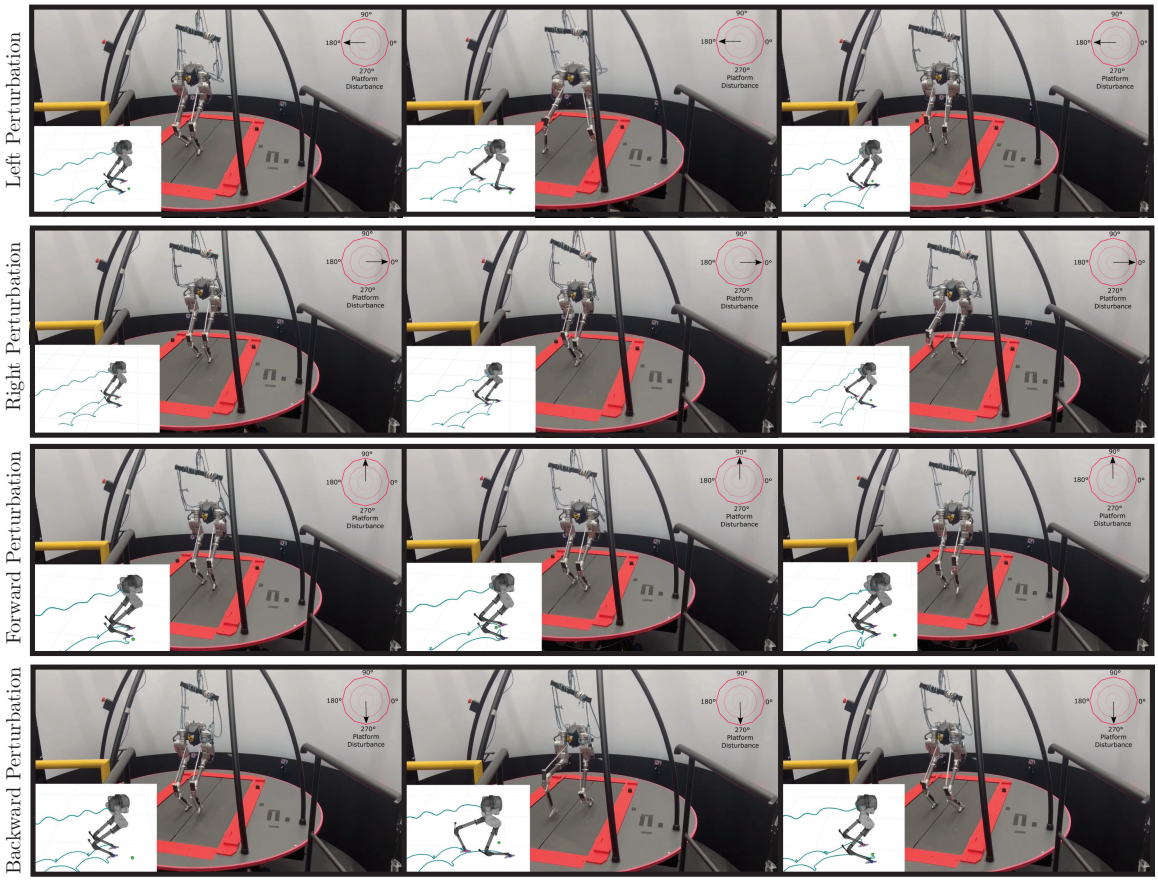

Figure 4.6: Cassie executing a i) wide-step disturbance recovery ii) crossed-leg disturbance recovery iii) forward disturbance recovery iv) backward disturbance recovery to a 0.7 m/s perturbation while walking sagitally at 0.7 m/s.

CHAPTER 5
A ROBUST PLANNING AND MANIPULATION FRAMEWORK FOR ELECTROMECHANICAL TASKS

5.1 Introduction

Long time-horizon task completion that is robust to errors and incorporates a large, heterogeneous, set of motion planning solutions remains a difficult problem. Most task planning frameworks do not possess sufficient knowledge about the environment or task relevant objects to execute a large variety of complicated tasks. Additionally, these frameworks often do not have the functionality to recover from actions that fail or be robust to environment uncertainties.

Assembly tasks are particularly challenging because they often involve a wide variety of objects and tools. To be extensible, manipulation policies and perception algorithms must be capable of generalizing across objects that vary in size, appearance, and topology. Tasking should also be capable of understanding relevant information such as preconditions and effects with manipulated objects. For example, a nut driver affords a robot to fasten/unfasten nuts, and the robot may to control its direction and speed. Provided a complete enough object representation, symbolic decision makers can interpret this knowledge to chain together behaviors and complete a task. However, the execution of these behaviors must still be robust to inevitable skill failures as well as to errors in the perceiving the object or the environment.

The goal for this work was to complete diverse assembly tasks in real-time that are formulated at run-time and are rich with object and environment uncertainty. In particular, we are interested in the assembly of electromechanical systems in industrial or space settings. For example, a space station may be unoccupied for an extended period of time

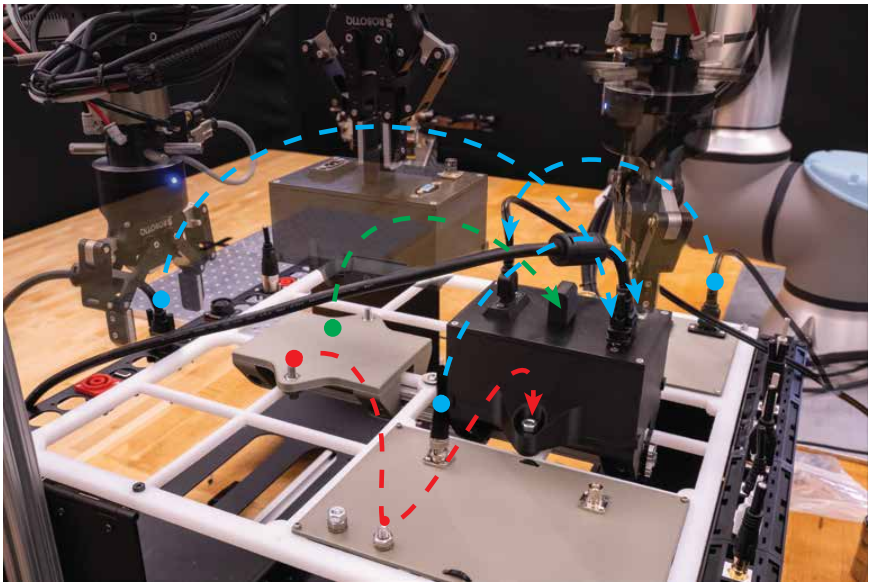

Figure 5.1: Both reactive and deliberative planning strategies are incorporated into the framework for task execution. Here, our system demonstrates the completion of an assembly task.

and requires robots to complete maintenance tasks. Mechanical tasks could include fastening/unfastening bolts, suctioning flat surfaces, and inserting/stacking objects. These mechanical tasks may also require a wide variety of tooling to complete. Electrical objects, such as connectors can be seen as an extension of the classic Peg-In-Hole (PIH) problem [66] with a wide variety of shape, topology, and constraints. The uncertainty of PIH problems for control has been well studied over the years [66]. However, despite this large shape variety, electrical connectors share categorical resemblances with each other and can easily be grouped regardless of cable shape or unique connector features. For example, D-Subminiature (D-Sub) connectors vary by the number of pins, case geometry, etc., but are all represented in the same class.

section 5.3 describes our task planning framework while section 5.4 describes our work with keypoint detection. In section 5.6, the complete system was evaluated in the task of electrical connector insertion using various connector types and subject to state and control errors. An electromechanical repair task was then solved using the complete combined system (Figure 5.1), where the robot must determine a long horizon action plan to solve the

task. Additionally, the task requires the system to flexibly adapt its data online, through the use of a database based on the categorical object representation.

Many existing hierarchical frameworks decompose the planning complexity of a PDDL problem into sub-problems. These systems are preferred in practice [67] [68][69] because they simplify the task sequencing and support code reuse as the system scales. As a result, this hierarchy works naturally with reactive planning schemes, such as Behavior Trees, as a middle layer for task execution [70][71].

Task and Motion Planning (TAMP) [72] [73] reasons about both geometric and symbolic planning constraints. While TAMP methods can solve complex long horizon problems, they are often rigid and incapable of reactive behaviors due to their intractable computational complexity. Furthermore, TAMP problems can fail to abstract to more heterogeneous tasks that involve a wider variety of low-level planning or perception techniques.

Many perception formulations exist for encapsulating object specific information and solving tasks. Most pipelines for pick and place tasks are capable of recognizing a known object and estimating a singular 6-DOF object pose. In general, pose estimation algorithms can be classified as learning-based and geometry-based approaches with supporting annotated datasets [74]. Recently, more object estimators have shown to be very generalizable. Dense descriptors can be used as an intra-category and self-supervised object representation for manipulation at a shape level [75]. While shape representations can be used to solve some tasks, but it is still unclear how these methods extend to partially observed objects or sparser object representations. Instead, one could consider keypoints because they are human specified, offer a sparser representation, and can still generalize with shape variation [76]. Additionally, keypoints are typically used as building blocks for different shape parameterizations, e.g. the polygons in [77] from which a manipulation policy is defined. Consequently, it is fair to say that keypoints offer a consistent object detection method as well as a flexible representation for encapsulating action specific constraints relevant to a task.

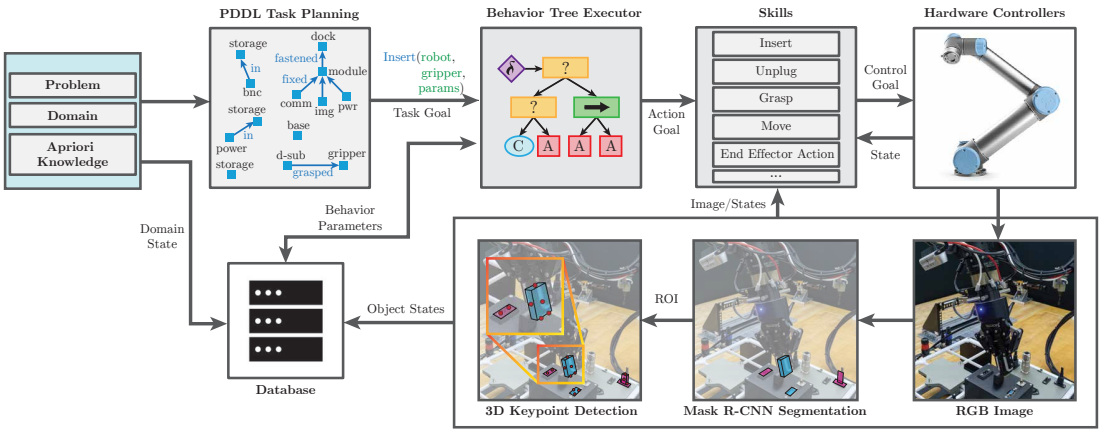

Figure 5.2: System framework with heterogeneous skills and perception pipeline. The database manages logical instances and binding to current metric information. This information was used to characterize the current environment and populate the PDDL domain at runtime. Our perception backbone extracts object position and keypoints to contextualizes objects in the scene for the database. BTs are used to reliably execute PDDL actions by calling on one or more skills using instance based information from the database.

Peg-In-Hole (PIH) problems have been well studied and the subject of control and planning research for decades. At its core, the PIH problem represents a case study in planning and control subject to uncertainty due to the tight tolerances present. Additionally, it has a wide variety of industrial problems and manifests in multiple forms [66]. Contact model-based and contact model-free methods are the most dominant approaches to solve the class of problems. Model-based methods can provide guaranteed results and generalize somewhat well, but often rely on a good compliance controller and heuristic tuning [78][79]. Model-free approaches primarily encapsulate learning based methods, such as Learning from Environment or Demonstration. Moreover, they can generalize with a wide variety of sensor modalities, but can suffer from data inefficiencies and lack of guarantees [80] [81] [82] [83] [84]. Despite the long history of PIH problems, there has been little work that explores action specific variations of the PIH problem, such as locking and unlocking phases that may be required to move the peg.

The goal of this project was to create a architecture based on the Plansys2 framework [70] for robust execution with Behavior Trees. The Plansys2 framework was extended by incorporating a structured XML schema and MongoDB database which conform to the

emerging IEEE RAS P1872-1 Robot Tasking Standard [85] in order to encapsulate more complicated object representations at the symbolic level. Since the logical variables are tied to the database, this backend modification also allows the system to dynamically modify its knowledge base in PDDL using perception and state feedback. Our framework allows for dynamic replanning due to skill/behavior failure as well as unexpected environmental states.

In the work, over a dozen skills were used in our framework, allowing the system to solve a wide variety of tasks in assembly/disassembly. Since the goal was to solve electromechanical problems with an extensive selection of electrical connectors and tools for insertion, special focus is placed on our insert and removal skills. Importantly, the implemented insertion strategy expands the traditional PIH problem to include a locking and unlocking phase. These additional phases are essential for connectors such as BNC and Ethernet to prevent damage or failure.

Coupled with the skills, a keypoint perception pipeline was incorporated to estimate semantic 3D keypoints as a more cohesive object representation. Desired objects are first detected using instance segmentation. Next, a 3D keypoint detection network provides detailed object representations with a set of learned keypoints. These object keypoints are encapsulated in the XML schema representation and are used to parameterize motion planning costs and constraints. By reducing object representation to sparse keypoints, the method also provides focus on task relevant information that can easily be fused with sensor input for grasping, picking, and placing objects. This sparse representation mean that manipulation can be planned using a variety of methods like trajectory optimization or a learned policy.

5.1.1 Contribution

This work thus attempts to develop a system with multiple new components. First, a manipulation pipeline that leverages a diverse set of skills under a Behavior Tree framework.

Where logical planning is provided by a Planning Domain Definition Language (PDDL) [86] planning system and is linked to a back-end database with an XML schema for grounding the symbolic information to motion plans. This allows for compact representations and dynamically changing object data for the symbolic plan. Moreover, the proposed task frame decomposition conforms to the emerging P1872-1 standard and may be seen as a test case of that standard's application. Generic object representations are also made to estimate the object pose using a variety of keypoints to capture shape variation between objects. Task specific keypoints are then used for both symbolic and geometric planning. Lastly, an insertion planner extends the sequential composition pipeline for peg-in-hole problems to locking connectors of varying geometry and lock types.

5.2 Additional Background

PDDL remains one of the most standardized languages for Artificial Intelligence (AI) planning. PDDL attempts to model a *domain* as a set of states using a list of factors and/or objects. These worlds begin with an *initial state* using apriori data and evolve using a set of rules and constraints. *Actions* represent a transition to a different state, but are limited by the constraints of the world. *Objects* define item instances in the world. *Predicates* are facts in the world that can either be True or False. For example, a predicate *box-built ?s - site* in the domain would represent whether a box is assembled (True/False) at a specific site instance. *Goals* specify the terminal state of the world the planner should reach. To organize these logical concepts, PDDL syntax specifies a *domain* and a *problem* file. The domain file establishes the context of the world by specifying the states, predicates, and actions. Consequently this specifies the states, the rules, and the how to move between states. The problem files represents a single instance of a world specified by the domain. Initial states are specified as true or false and a goal state is established for a planning problem. To limit the length of this thesis, we will not cover the exact syntax of these files and the logical constraints, but readers are encouraged to explore [86, 87].

5.3 Task Planning with Behavior Trees and Skills

The proposed framework shown in Figure 5.2 builds upon the recent trend in behavior trees for reliable task planning and execution. However, this work extends the core implementation by integrating it with a backend database and XML schema for structuring object classes and managing dynamic information online.

5.3.1 Task Planning with Behavior Trees

Plansys2 [70] was used to generate a PDDL plan that incorporates BTs as a part of its task planning framework. The PDDL domain and problems are modeled with Behavior Trees to construct a planning graph G_a, which is a directed acyclic graphic of tuples $G_a =< A_a, C_a >$. A_a is defined as a set of actions in the plan corresponding to the nodes of the graph and C_a is the set of directed arcs representing the action execution procedure. The action set A_a is composed of action unit tuples $a_i \in A_a$ of the form $< t_a, \rho_a, R_a, E_a >$. t_{a_i} is the time of the action, ρ_{a_i} is action to be executed, R_{a_i} is the set of predicates that encapsulate the parameters for executing the action, and E_{a_i} is the set of predicates that will be added to the domain knowledge base after successful execution of the action. An existing BT is composed of *skills* regulated by control nodes is then correlated to each action a_i .

Once more, BTS are utilized to organize low-level skills (motion plans, control actions, triggers, etc) into higher-level behaviors. The developer is able to compose and modify behaviors without any robot recompilation due to the XML format of the BT. Pragmatically speaking, this is important for system deployment, since there may be no guarantee of being able to recompile code once it is deployed on a robot. Parameter substitution from the BT into the skills is automatically performed where parameters are typically the names of instances in the environment. Binding of instance name to metric location is then performed by our database during skill execution.

For example, the BT *Attach End Effector(<robot>, <gripper>, <trajectory>, <tool_rack>, <insert>)*, is composed of the skills *Cartesian Move(<tool_rack>)*, *Zero Force Torque()*, *Tool Changer (<constant_open>)*, *Insert (<insert>)*, and *Replay Move (<tool_rack>, <trajectory>)* in a fallback tree structure (section 5.5). This action would insert the end-effector into the specified tool, attache it, and pull it out of its holder using a planned trajectory relative to the tool rack. For the demonstration mentioned in this work, the location of the tool rack was located using an AprilTag and pushed to the database online.

The BT blackboard implementation only passes primitive datatypes for node parameterization. To add schema based data structures to BTs, the blackboard passes ASCII string keys that pull data from the supporting database. This allows the framework to provide more data rich object information throughout the BT as structures. For example, the BT would be provided string *bnc-obj* from PDDL, which is then used to search the database for a connector type structure named *bnc-obj* that would encapsulate electrical connector type, geometry, locking type, etc.

5.3.2 Databases and Schema's for Dynamic Information

A critical part of the overall system is the *knowledge schema*. This XML schema contains a representation of all of the information in the task frame. At the lowest layer, the base schema contains domain specific extensions to represent object data. The base schema is referred to as the *knowledge cell* and contains generic types that may be specialized by other schemas, as well as types that are likely to be utilized across multiple domains. Most notably, these schemas can contain information such as the shape of the object, its pose, and any action specific information. For example, the solid object type is an abstract type that is extended to form any object (an instance) that is represented in the world. An extension of a solid object is the domain specific object instance *electrical connector*, which contains additional information such as *connector type*, if it has *locking/unlocking* logic, etc. These object types can be continuously inherited and combined with other schema

types to construct increasingly complicated object and environment representations that can be referenced in both compiled source code, non-compiled formats (BT XML files) and task planning languages (e.g PDDL).

Predicates are also stored in the *knowledge schema*. The predicates represent the world state as well as preconditions, maintenance conditions, and expected action effects. These predicates build upon any of the structure represented in the schemas. For example, the predicate *(gripper_type ?g - gripper ?f - gripper_function)* is used to encapsulate a structured representations of a gripper object type as well as the entirety of its inhereted functionalities. The parameterizations of this predicate can then be used as part of a goal state, constraint on an action, or an expected result after an action.

A database generator was used to generate three products. First, the stuctured data contained in the XML schema is used to automatically generate C++ Classes. Secondly, all named instances and predicates for logical reasoning were stored in a generated logical database. Lastly, a metric MongoDB database is generated to translate logical names to geometric quantities. Metric data can be populated *a priori* or pushed to the database online, consequently tying symbolic quantities to potentially dynamic information. For example, a *tool_rack* pose that tools use as a reference pose gets identified using fiducials and can be updated at runtime.

5.4 Category-Level Object Representations

Robots should be capable of achieving purposeful manipulation by generating motion policies for multiple different objects within the same category. Many existing pose-estimation methods assume a single SE(3) transform template representation for an object. However, this representation oversimplifies many objects and does not provide object specific information. Keypoint frameworks, on the other hand, are able to generalize SE(3) actions into a superset of points on the object. To this end, the perception system is capable of parameterizing desired keypoints on an object regardless of shape variation or color/texture of the

objects in a given category. Moreover, keypoints can represent costs and constraints for trajectory optimization problems that wish to manipulate a desired object.

Object category-type actions (e.g. locking tab, button) can also be attributed to object keypoints. Furthermore, symbolic logic from the task planning can be linked to the metric object details and be incorporated into predicates or parameters. For example, some electrical connectors have locking mechanisms (e.g ethernet connectors). Consequently, the locking mechanism is defined by a geometric location and symbolic constraint of locked/unlocked that must be pressed to unlock the connector.

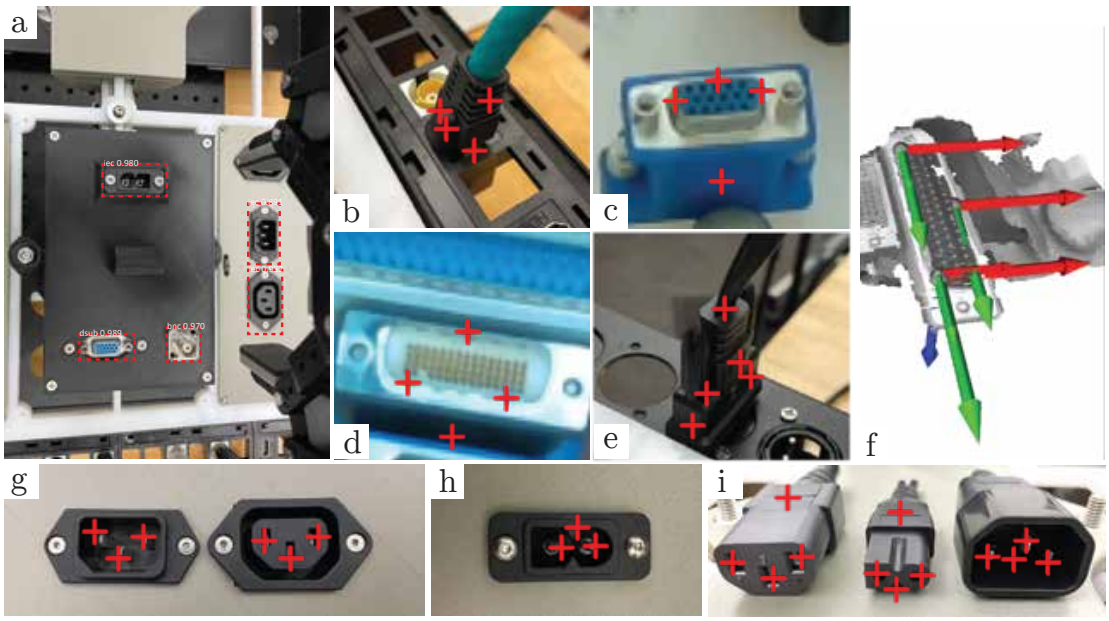

Figure 5.3: (a) Mask R-CNN of connector ports from the wrist camera (b-h) Keypoints detected on various electrical connectors (i) Visualization of depth prediction on a D-sub connector.

Since keypoints are specific to each object instance in a camera frame, each needs to be identified independently in the scene using instance segmentation. To this end, Mask R-CNN [88] instance segmentation was used to estimate bounding boxes around desired objects and to create a mask outline for the respective object instances. Object classification can be used as a lookup in the database for more object rich information, such as the

control parameters or locking type. However, Mask R-CNN does not capture the exact 3D geometry of the object which can lead to the location of a locking tab.

Once an RGB-Depth stream is obtained and object images are segmented by a bounding box, a modified integral network [89] produces a probability heatmap and predicted depth map in the 2D image frame (Figure 5.3). A probability distribution map $f_i(u', v')$ is generated by the nextwork that details the likelihood a keypoint i is to occur at pixel (u', v') provided $\sum_{u',v'} f_i(u', v') = 1$. Expected values of these distributions can be used to recover keypoint i at a pixel coordinate (u_i^*, v_i^*).

$$(u_i^*, v_i^*)^T = \sum_{u',v'} (u' \cdot f_i(u', v'), v' \cdot f_i(u', v'))^T \tag{5.1}$$

The probability distribution maps are then used to generate predicted 3D keypoints using the calibrated camera intrinsics and extrinsics. Estimated z_i depth coordinates for the keypoint are calculated based on the predicted depth at every pixel $d_i(u', v')$.

$$z_i = \sum_{u',v'} d_i(u', v') * f_i(u', v') \tag{5.2}$$

Annotated pixel coordinates and depth for each keypoint from the training images are used to train the network using an integral and heatmap regression loss [89]. A 34 layer Resnet is used as a backbone for the neural network and the dataset generated is detailed in subsection 5.6.2.

The keypoints can be leveraged for a variety of tasks such as grasp planning, visual servoing, and insertion. A set of N position keypoints $r_{\text{kp}} = \{r_{\text{kp,i}}\}_{i=1}^N \in \mathbb{R}^{3\times N}$ and an object pose $p(r_{\text{kp,i}}) \in \mathbb{R}^6$ are defined in the world frame. Common constrained optimization solvers can then be used to generate a kinematic robot action that manipulates the object

with transform $T_a \in SE(3)$

$$\begin{aligned} \min_{T_a \in SE(3)} \quad & \mathcal{L}(T_a, r_{\text{kp,i}}) \\ \text{s.t.} \quad & g(T_a, r_{\text{kp,i}}) = 0 \\ & h(T_a, r_{\text{kp,i}}) <= 0 \end{aligned} \tag{5.3}$$

with flexible constraint choices for $g(\cdot), h(\cdot)$, and cost function $\mathcal{L}(\cdot)$. Since keypoints may not directly align due to perception errors, desired actions are best computed as an optimization problem that minimizes the L2 distance between specified transformed keypoints and their desired targets $r^d_{kp,i}$:

$$\mathcal{L} = ||T_a r_{\text{kp,i}} - r^d_{\text{kp,i}}||^2 \tag{5.4}$$

A wide range of potential constraints for connectors and objects can be used on keypoints for object poses since they are simply kinematic constraints. For the experiments conducted, a point-to-plane constraint was set to make sure objects are aligned with a desired object plane orientation axis v^d_{axis}. An estimated object orientation was calculated using two keypoints along the length of the object $v_{\text{obj\_axis}} = \frac{r_{\text{kp,i}} - r_{\text{kp,j}}}{||r_{\text{kp,i}} - r_{\text{kp,j}}||}$.

$$||1 - < v^d_{\text{axis}}, rot(T_a) v_{\text{obj\_axis}} >||^2 = 0 \tag{5.5}$$

While the mentioned constraints will generate feasible trajectories for pick and place tasks, is worth reiterating that this is a simple transform example. More complicated manipulation policies, such as visual servoing or contact constrained motion plans could also use keypoints as a reference [90]. Additional constraints such as contact or obstacle avoidance could also be used to generate feasible trajectories.

Robot grasping was performed based on the keypoint extraction. The grasp was assumed to be tight and the grasp point non deformable. The keypoints are used to reduce the search space on the object pointcloud to determine the best grasp position. For most con-

nectors, the estimated object pose was used to calculate a desired orientation in the gripper frame based on the cropped bounding box pointcloud data.

5.5 Manipulation Skills

The framework provides a hierarchy that makes use of skills which are programmed onto the robot. These skills are be composed through the use of an XML file into a configurable BT that performs a desired behavior. By composing these BTs, a task may be accomplished. For this work, the implemented skills are listed in Table 5.1. Control nodes in the BT are responsible for making sure a behavior is completed successfully by monitoring skills. Skills range in complexity from trajectory planning to a simple boolean trigger for changing tools with a solenoid.

While some skills have the same goals, their underlying methods can vary based on the task. For example, movement to a point in SE(3) may be accomplished using a *Cartesian Move* which will precisely position the end-effector using only position feedback or through the use of a *Compliant Move* that will incorporate compliance in the movement.

Compliant Control: Interacting with objects necessitates both exteroceptive force feedback as well as pose control. A common approach is impedance control because it indirectly controls contact force through a stiffness. However, impedance control does not directly control force to maintain equilibruim with its environment, which may lead to jamming. Instead, the compliance controller uses a parallel force/position controller [91] that specifies both wrench and pose goals for a desired configuration. The control is formulated as follows:

$$a_x = K_{\mathrm{p,x}}(x^d_{\mathrm{ee}} - x_{\mathrm{ee}}) + K_{\mathrm{d,x}}(\dot{x}^d_{\mathrm{ee}} - \dot{x}_{\mathrm{ee}}) \tag{5.6}$$

$$\hat{\lambda} = G(q) + \lambda_e \tag{5.7}$$

$$a_f = K_{\mathrm{p,f}}(\lambda_d - \hat{\lambda}) + K_{\mathrm{d,f}}(\dot{\lambda}_d - \dot{\hat{\lambda}}) \tag{5.8}$$

$$\pi_{\text{cmd}} = a_x + a_f \tag{5.9}$$

where $G(q) \in \mathbb{R}^6$ is the gravity compensation wrench and $\lambda_d, \lambda_e \in \mathbb{R}^6$ represent the desired and current external wrenches for the task measured at the F/T sensor mounted to the end-effector. $K_{\text{p,x}}, K_{\text{p,f}}$ are the proportional gains for position and force respectively. $K_{\text{d,x}}, K_{\text{d,f}}$ are the derivative gains for position and force components of the compliant controller. Lastly, $x_{\text{ee}}, x_{\text{ee}}^d \in \mathbb{R}^6$ are the current and desired cartesian end-effector pose, which assume rigid contact interactions with the environment. A joint velocity command $\dot{q}_{\text{cmd}}$ is generated based on the inverse kinematics of π_{cmd}. Since the Force/Torque (F/T) sensor is at the end-effector wrist position, and not at the point of gripper contact, the control point varies by grasp. To account for this issue, the contact wrenches can be projected from the object reference frame back to the wrist sensor end-effector frame.

$$\lambda_{\text{obj}} = \begin{bmatrix} R_{\text{obj}}^T & 0 \\ R_{\text{obj}}^T [p_{\text{grasp}} \times] & R_{\text{obj}}^T \end{bmatrix}_{\text{ee}}^{obj} \lambda_e \tag{5.10}$$

where $\lambda_{\text{obj}} \in \mathbb{R}^6$ represents the wrench at an arbitrary point p_{grasp} on a grasped object with relative rotation $R_{\text{obj}} \in \mathbb{R}^{3\times3}$ to the end-effector frame.

Cartesian and Joint Control: Cartesian and Joint movement actions are solved with STOMP [92] and MoveIt! [93] for safe collision-free motions. Replay movement actions execute a parameterized pre-recorded trajectory that can be used for more complicated motion policies such as tool changing. Simple kinematic offsets can be made to offline trajectories, such as changing the starting $T_{0,a}$ or ending $T_{N,a}$ transform to be at a desired pose.

Visual Servoing: ViSP [94] is used for executing Point Based Visual Servoing (PBVS) and Image Based Visual Servoing (IBVS) actions. Given the known coordinate of an object in the view of the camera, such as an AprilTag or identified electrical connector, the visual servoing can be used to track a desired offset pose.

Triggering Actions: Items such as zeroing sensors, switching pneumatics, or push-

ing/pulling data to the database are available. Transforms are added/removed from the database based on perception. In addition, tool changers can use pneumatic solenoids for locking end-effectors in place.

Connector Insertion: Insertion of connectors can be achieved using a variety of methods and is representative of a classic PIH problem. However, electrical connectors are not ideal PIH problems due to their shape variation and locking/unlocking mechanisms. Connector insertion was approached using a model based approach (Figure 5.4)

Connector insertion planning is decomposed into four phases: *meet*, *search*, *embed*, and *lock*. For unplugging connectors, a similar process was used: *unlock* and *unplug*. The meet phase moves to the estimated insertion position and looks for contact with a surface, the search phase executes a search policy to align the peg with the hole, and the embed phase assures that the connector successfully bottoms-out in the hole with the minimum force possible. Like most model based methods, our approach relies on compliant control (section 5.5) to interact with the environment. Each of these planning phases is seen as an independent skill that can be composed into either a BT or a FSM depending on the complexity of the insertion. The FSM (Figure 5.5) is used to manage the transitions between different contact modes based on the estimated contact state. Each stage updates the constraints for the planning required to insert and lock a connector. Since the connector is known from the perception system, the schema representation of object in the database can be used for parameterizing insertion. For example, a BNC connector is insertable based on the PDDL predicates and is constrained as "twist" locking in the database.

Estimating the contact state requires two supporting steps: contact state modeling and classification. Contact states were modeled based on the set of values $\kappa = \{e_\kappa, f_\kappa, d_\kappa\}$, where $e_\kappa \in R^6$ is a set of binary values based on axis constraints for each phase, f_κ is a set of contact force thresholds, and d_κ is the bottom out distance. Contact state classification is determined by measuring the difference between the modeled contact state κ and the observed contact state. Simple online tests verify the constraints of the insertion and

Table 5.1: Generic skills that can be composed in Behavior Trees to reliably complete an action.

Skill Name	**Skill Description**
Cartesian Move	Inverse kinematic planning and tracking to a pose in SE(3)
Compliant Move	Inverse kinematic planning to and tracking to a pose in SE(3) with the compliant controller
Joint Move	Commands M joint configurations $q \in \mathbb{R}^M$
Replay Move	Tracks a premade trajectory in SE(3)
Tool Changer	Tool changer solenoid trigger that allows for end-of-arm-tooling to be replaced
Add/Remove Frame	Adds or removes a transform frame in the world coordinate system
Zero Force Torque	Resets the force torque sensor to avoid drift
Visual Servo	Perform a visual servo move to align with a detected object
Insert/Unplug	Inserts or Unplugs an object
Meet/Search/Embed /Lock/Unlock	Component policies from Insert
Detect Keypoints	Runs Mask R-CNN+Keypoint pipeline
Keypoint Grasp	Calculates a grasp for the desired keypoint object based on pointcloud data
Keypoint Move	Moves the arm based on the desired keypoint transform for an object in hand

propagate the FSM to its next state (Figure 5.5) until the connector is fully constrained according to the model.

A *meet* policy can be simple and is terminated using Eq. Equation 5.11. Provided that the end-effector can orientate itself orthogonal to the plane of the meeting surface, a simple linear motion plan can be executed until termination. Alternatively, if the "hole" is identified in the world, a closed loop PBVS policy would be used until it overcomes a threshold force $f_{\text{meet}} \in \mathbb{R}^3$.

$$v_{\text{dir}} \cdot (R_{\text{ee}} \cdot f) \leq f_{\text{meet}} \tag{5.11}$$

Where v_{dir} is the direction of motion for meeting, R_{ee} is the end-effector rotation, and $f \in \mathbb{R}^3$ is the force sensor reading.

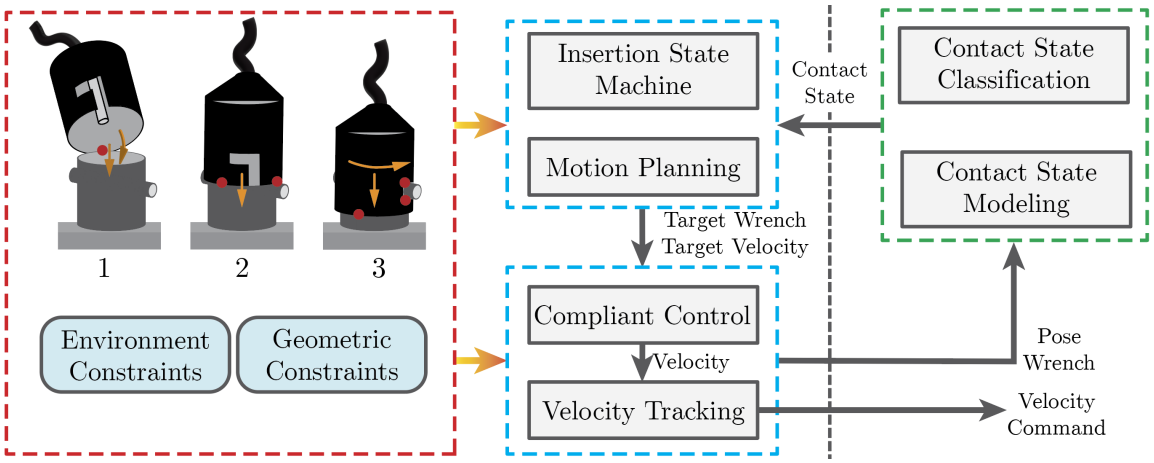

Figure 5.4: Contact strategy for insertion with locking and unlocking phases. A state machine manages transitions between hybrid motion plans based on the contact state and previous states. State machines are used with the intention that each state has the potential to be replaced with a more complex motion planning policy on its own.

Search uses tilt, spiral, or time varying Lissajous trajectories to generate a blind motion plan in $\mathbb{R}^6$. A spiral can be generated from a grasped object with initial position p_0. The robot generates a planar spiral trajectory normal to the directional vector v_{search} with radius $r_{\text{s,i}}$ where $\delta\beta$ and δr_s are discretized step lengths for the angle and radius or the polar motion. I, I_1, I_2 are the identity matrix and its first and second column respectively. Lastly, Q_{rod} is Rodrigues' rotation formula for generating a trajectory.

$$p_{i+1} \stackrel{+}{=} r_{s,i+1} \cdot Q_{\text{rod}} \cdot (I_1 + I_2) \tag{5.12}$$

$$Q_{\text{rod}} = (I + sin(\beta_{i+1}))[v_{\text{search}}\times] + (1 - cos(\beta_{i+1}))[v_{\text{search}}\times]^2 \tag{5.13}$$

$$\beta_{i+1} = \beta_i + \delta\beta, \quad r_{s,i+1} = r_{s,i} + \delta r_s \tag{5.14}$$

An alternative to the spiral is a Lissajous curve for each axis of the end-effector pose:

$$p_i = \eta t^{\sigma} sin(\Omega_f * t + \delta_{\text{off}}) \tag{5.15}$$

Where t is time, η is a unit scaling factor, σ is a time scaling factor, Ω_f is the frequency of oscillation, and δ_{off} is a phase offset.

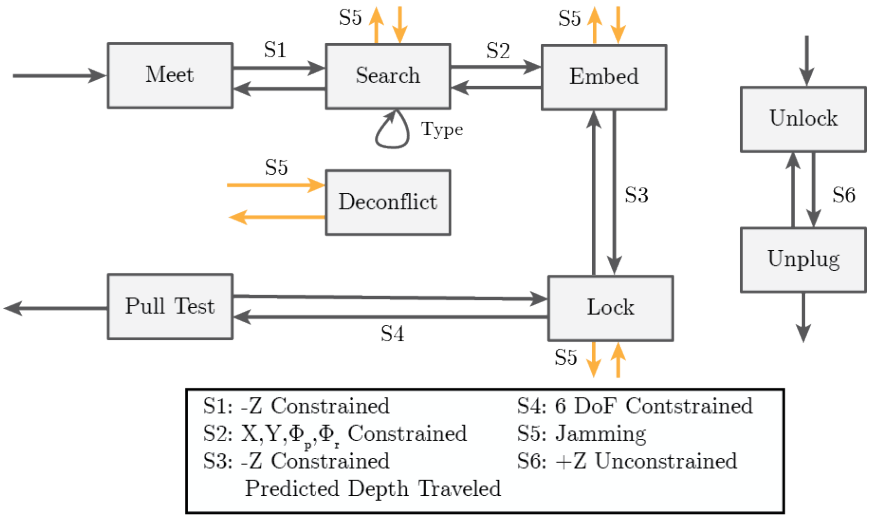

Figure 5.5: The insertion planning state machine manages the stage of the insertion process using the contact classification and modeling. X,Y,Z represent the linear Cartesian components and Φ_r, Φ_p, Φ_y represents the roll, pitch, and yaw components of motion. This FSM only demonstrates one example set of transitions, but is not bound to it. For example, S4 could Φ_y is left unconstrained for round, non-locking, connectors.

Recent works (e.g [83]) have used learned and/or visuo-contact policies for the search and embed phases of the process. These methods are able to be easily integrated into the insertion pipeline for robust insertion since it is containerized into a FSM. But more complex insertion methods were outside the scope of this work.

Embedding provides an increasing force until the end-effector has traveled an estimated distance and a predicted force threshold is surpassed. Since the wrench is directly controlled, the closed-loop parallel force-position controller can actively avoid jamming. If jamming does occur (or more extreme contact situations), a *Deconflict* state commands the opposing currently experienced force, but tries to maintain the same pose until the contact wrench is zeroed. A fully inserted connector can also be subjected to a pull test to verify that it was locked and secured.

Locking mechanisms were modeled as *screw*, *twist*, *tabbed*, *push*, and *null* based locking/unlocking mechanics for the insertion/removal problem. Screwing used a multi-turn trajectory before termination. Twist locks, such as BNC connectors, were similar but only considered trajectories with angles $\alpha_{\text{twist}} < \pi$. Tab locks, such as ethernet connectors,

required the object grasp to squeeze a keypoint to unlock. A keypoint on the tab constrained the predicted grasp such that the the finger remains in contact with the keypoint ($r_{\mathrm{kp,i}} = p_{\mathrm{finger}}$) to unlock the connector while unplugging. Null was used to describe non-locking connectors, such as HDMI, which causes a pass-through to the next state in the FSM. It should be mentioned that some connectors do have multiple locking conditions, such as twist and tab, but these extra complications fall outside the scope of this paper and may require higher degree of freedom grippers.

Table 5.2: Behavior Trees used to execute the desired electromechanical task. Each BT is composed of skills detailed in section 5.5

Behavior Tree Name	**Behavior Tree Description**
Init	Initializes the robot and any world information
Attach/Detach End Effector	Inverse kinematic planning to a cartesian pose
Fasten/Unfasten	Fastens/Unfastens an object based on number of bolts constraining it
Pick/Place	Pick and place unconstrained objects
Insert/Unplug Object	Moves and Inserts or Unplugs an object
Detect Keypoints	Detects keypoints at regions of interest

5.6 Results

To verify the methods and performance of the framework, the perception and insertion pipelines were benchmarked as independent components. The performance of the overall architecture was evaluated in the task of assembling an electromechanical module. Using a single PDDL Domain, the system was able to create a task plan that solved the assembly problem repeatedly while automatically planing trajectories robust to intra-category variations without excessive ad-hoc tuning.

5.6.1 Electromechanical Task Overview

The robot was tasked with assembling an electromechanical module that may represent an industrial module or science experiment on a space station. At a high level, successful

task planning solutions do the following: (1) unfasten the nuts restraining the module to move it to the install location (2) fasten the module to its new location (3) unplug electrical connectors from a storage area to power and communicate with the module, and (4) display a valid output from the module to verify correct installation.

A variety of tooling and sensors are required to complete the desired assembly task. A 6-DOF UR10e robot arm with an internal Force Torque (FT) sensor was used to interact with the environment. Two end-effector tools were used for the demonstration: a 2F-85 Robotiq two finger gripper and a custom nut driver with a magnetic socket. QC-11 pneumatic tool changers were used to quickly attach and detach end-effectors for the arm. An Intel RealSense D435 was used for collecting RGB-Depth images and pointclouds for identifying keypoints. Both the intrinsic and extrinsic parameters for the RGB-Depth camera were calibrated using MoveIt [93].

The PDDL plan was solved using the conventional POPF [95] planner. Since much of the computational task planning complexity was incorporated in the Behavior Tree structure, the solver was able to compute a valid assembly task plan in 0.52 seconds. The assembly task took 10 minutes and 14 seconds to execute with no failures across 10 trials. During that time, for one run, the robot executed 34 behaviors and 148 actions and re-planned two times.

5.6.2 Categorical Object Perception

For the aforementioned task, a small dataset for Mask R-CNN was created to identify the relevant objects in the electrical task. However, more work could be done to further expand the electrical dataset for more generic tasks.

An attempt was made to fully train the keypoint network in simulation, but the method failed to transfer to the real world after much experimentation. Keypoint data collection on hardware was completed using a predefined funnel trajectory that retrieves depth and image data around the desired object(s). Open3D was used to label keypoints in object regions of

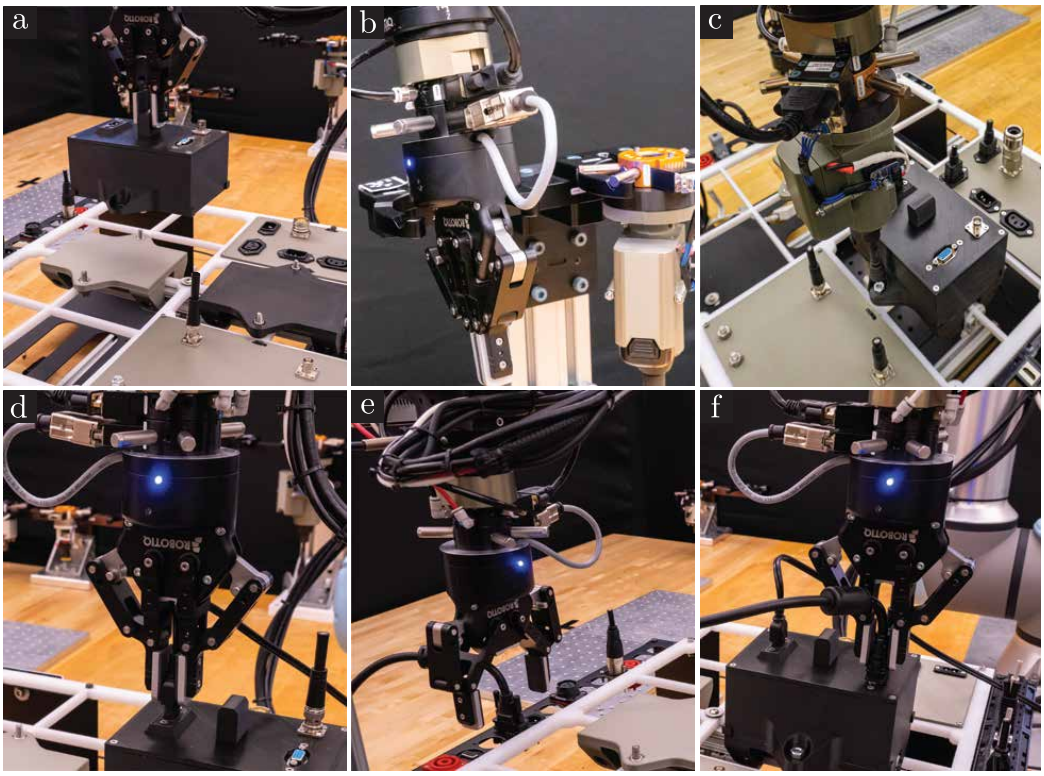

Figure 5.6: Demonstration of a multi-object and multi-tool assembly task. The robot uses a nut driver to unfasten and fastens bolts. A two finger gripper moves the module and inserts each connector. Tool changing was used to swap capabilities. Additional tasks could be added that include a suction gripper.

interest.

We collected our own datasets for the three electrical connector categories in the task: D-Sub, IEC 60320, and BNC connectors. In order to generalize across multiple different connectors, the dataset used 10 different connectors per category. Lastly, the network was trained using a set of 89,000 images for each connector category.

Table 5.3: Experimental results from the keypoint dataset for the different connectors used in the demonstration. Values are the average across the 5 connector variations. Camera was held 20cm from the connector for consistency in pixel accuracy.

Object	Pixel Error Error (px)	Average Depth Error (mm)	Average Keypoint Error (mm)
IEC Connector	4.6	2.6	1.8
D-Sub Connector	3.7	2.4	1.9
BNC Connector	5.1	2.8	2.1

A set of 5 female and male D-Sub connectors and IEC connectors, as listed in Table 5.3, with various pinouts and geometry, were used to test the generalizability of the perception pipeline. Accuracy was important, since connector features are typically only a few millimeters in width or length. For all connectors, the keypoint method was capable of detecting keypoints with millimeter accuracy.

5.6.3 Connector Insertion

To demonstrate the repeatability of the insertion, the system repeated 50 test insertions and removals without the BTs overhead. This would test the success of the insertion skill without the need for higher level fallback help. Table 5.4 shows the success rate and time spent in each phase of a connector insertion. An acceptable search position tolerance for each connector was estimated using the ratio of connector diagonals, where IEC, D-Sub, and BNC could tolerate offsets of 50% their diagonal length. This tolerance was empirically tested using consistently larger diameter randomized distributions. While not rigorously or guaranteed, this 50% boundary of robustness demonstrates the ability of the system to handle deviations in object estimation.

Model based methods typically require more tuning compared to learning methods, but do tend to offer more consistent results. Model based insertion method do heavily rely on a compliance controller to handle model discrepancies. However, it should be reiterated that our skill framework allows developers to easily replace the implemented methods with learned policies.

Table 5.4: Connector insertion success rate and phase times averaged across 50 trials. Timing was cumulative for each insertion, which could repeat phases based on the contact state.

	Meet	Search	Embed	Lock	Unlock
D-Sub	2.5s 100%	6.2s 100%	6.7s 100%	N/A N/A	N/A N/A
IEC	2.2s 100%	4.1s 100%	6.9s 100%	N/A N/A	N/A N/A
BNC	3.1s 100%	11.0s 96%	13.8s 100%	22.4s 96.8%	22.7s 100%

CHAPTER 6

CONCLUSIONS

Robots have been a tremendous asset to mankind in many dull, dirty, and dangerous tasks. Thanks to recent advances in computing and algorithms, there has been more of a push to get robots away from ideal environments with few unknowns or difficulties. There is much to gain by doing this. For example, the United States Postal Service (USPS) managed the transportation of over 6 billion packages in 2018 [96]. A large portion of the cost can be attributed to the last mile of the delivery process, a crucial component to delivery [97]. Legged robots have been proposed as a potential solution to solving this problem because of their ability to climb stairs, step over hazardous objects, and handle the disturbance of packages and people. In addition, NASA is further exploring how robots can operate in space to complete tasks [98]. Deep space habitats are of particular interest because a human crew may not always be present to maintain the habitat. Instead, robots would be needed to complete tasks like checking critical sensors, maintaining experiments, or assembling critical hardware components.

This thesis examined the integration of Behavior Trees into Full-Stack robot control architectures for Task and Motion Planning. Two separate frameworks, one for locomotion and the other manipulation, integrated behavior trees as a core component of their operation. A variety of low-level motion planners and controllers were explored to eventually give Cassie the ability to walk over 1m/s on hardware and upwards of 2m/s in simulation. The footstep planner, tracking controller, and trajectory optimization showed exceptional robustness when foot contact worked as expected. In particular, the ability to execute crossed-leg maneuvers showed a dramatic increase in disturbance rejection. The manipulation framework showed a complete pipeline manipulating a variety of electrical connectors and executing drilling tasks to assemble an electromechanical module. Integrating Behav-

ior Trees allow the arm to do these tasks in a robust way provided in can detect deviation from the original plan. I believe that this work shows the potential for Behavior Trees to be integrated into a wide variety of robotic systems to help them leave structured environments and solve tasks in the real world.

REFERENCES

[1] E. Ackerman, *Qamp;a: Inside darpa's subterranean challenge*, Apr. 2022.

[2] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, N. Fazeli, F. Alet, N. C. Dafle, R. Holladay, I. Morona, P. Q. Nair, D. Green, I. Taylor, W. Liu, T. Funkhouser, and A. Rodriguez, *Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching*, 2017.

[3] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.

[4] A. Church, "Edward f. moore. gedanken-experiments on sequential machines. automata studies, edited by c. e. shannon and j. mccarthy, annals of mathematics studies no. 34, litho-printed, princeton university press, princeton1956, pp. 129–153.," *Journal of Symbolic Logic*, vol. 23, no. 1, pp. 60–60, 1958.

[5] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, 1987.

[6] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999. eprint: https://doi.org/10.1177/02783649922066385.

[7] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017. eprint: https://doi.org/10.1137/16M1062569.

[8] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of Guidance, Control, and Dynamics*, vol. 21, no. 2, pp. 193–207, 1998. eprint: https://doi.org/10.2514/2.4231.

[9] ——, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming, Second Edition*, Second. Society for Industrial and Applied Mathematics, 2010. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898718577.

[10] O. Von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of Operations Research*, vol. 37, pp. 357–373, Dec. 1992.

[11] D. Pardo, L. Moller, M. Neunert, A. W. Winkler, and J. Buchli, "Evaluating direct transcription and nonlinear optimization methods for robot motion planning," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 946–953, Jul. 2016.

[12] A. Wächter and L. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, pp. 25–57, Mar. 2006.

[13] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM Rev.*, vol. 47, no. 1, pp. 99–131, Jan. 2005.

[14] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Knitro: An integrated package for nonlinear optimization," in *Large-Scale Nonlinear Optimization*, G. Di Pillo and M. Roma, Eds. Boston, MA: Springer US, 2006, pp. 35–59, ISBN: 978-0-387-30065-8.

[15] M. Grant and S. Boyd, *CVX: Matlab software for disciplined convex programming, version 2.1*, http://cvxr.com/cvx, Mar. 2014.

[16] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2022.

[17] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[18] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[19] "Fast model predictive control using online optimization," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 6974–6979, 2008, 17th IFAC World Congress.

[20] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007, ISBN: 354023957X.

[21] R. Featherstone, *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2007, ISBN: 0387743146.

[22] Z. Gu, N. Boyd, and Y. Zhao, "Reactive locomotion decision-making and robust motion planning for real-time perturbation recovery," *2022 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[23] B. Stephens, *Push recovery control for force-controlled humanoid robots*. Carnegie Mellon University, 2011.

[24] P.-b. Wieber, "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations," in *IEEE-RAS International Conference on Humanoid Robots*, 2006, pp. 137–142.

[25] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture point: A step toward humanoid push recovery," in *IEEE-RAS international conference on humanoid robots*, IEEE, 2006, pp. 200–207.

[26] S.-J. Yi, B.-T. Zhang, D. Hong, and D. D. Lee, "Online learning of a full body push recovery controller for omnidirectional walking," in *IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2011, pp. 1–6.

[27] M. Shafiee, G. Romualdi, S. Dafarra, F. J. A. Chavez, and D. Pucci, *Online dcm trajectory generation for push recovery of torque-controlled humanoid robots*, 2019. arXiv: 1909.10403 [cs.RO].

[28] C. Liu, J. Ning, K. An, and Q. Chen, "Active balance of humanoid movement based on dynamic task-prior system," *International Journal of Advanced Robotic Systems*, vol. 14, no. 3, p. 1 729 881 417 710 793, 2017.

[29] C. Zhou, C. Fang, X. Wang, Z. Li, and N. Tsagarakis, "A generic optimization-based framework for reactive collision avoidance in bipedal locomotion," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2016, pp. 1026–1033.

[30] A.-C. Hildebrandt, R. Wittmann, D. Wahrmann, A. Ewald, and T. Buschmann, "Real-time 3d collision avoidance for biped robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 4184–4190.

[31] A. Dietrich, T. Wimböck, H. Täubig, A. Albu-Schäffer, and G. Hirzinger, "Extensions to reactive self-collision avoidance for torque and position controlled humanoids," in *IEEE International Conference on Robotics and Automation*, 2011, pp. 3455–3462.

[32] Q. Nguyen, X. Da, J. Grizzle, and K. Sreenath, "Dynamic walking on stepping stones with gait library and control barrier functions," in *WAFR*, 2016.

[33] Y. Gong, R. Hartley, X. Da, A. Hereid, O. Harib, J.-K. Huang, and J. Grizzle, *Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway*, 2018. arXiv: 1809.07279 [cs.RO].

[34] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[35] J. Liu, N. Ozay, U. Topcu, and R. M. Murray, "Synthesis of reactive switching protocols from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 58, no. 7, pp. 1771–1785, 2013.

[36] K. He, A. M. Wells, L. E. Kavraki, and M. Y. Vardi, "Efficient symbolic reactive synthesis for finite-horizon tasks," in *International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8993–8999.

[37] Y. Zhao, Y. Li, L. Sentis, U. Topcu, and J. Liu, *Reactive task and motion planning for robust whole-body dynamic locomotion in constrained environments*, 2018. arXiv: 1811.04333 [cs.RO].

[38] S. Kulgod, W. Chen, J. Huang, Y. Zhao, and N. Atanasov, "Temporal logic guided locomotion planning and control in cluttered environments," in *2020 American Control Conference (ACC)*, 2020, pp. 5425–5432.

[39] J. Warnke, A. Shamsah, Y. Li, and Y. Zhao, "Towards safe locomotion navigation in partially observable environments with uneven terrain," in *IEEE Conference on Decision and Control (CDC)*, 2020, pp. 958–965.

[40] R. Ehlers and U. Topcu, "Resilience to intermittent assumption violations in reactive synthesis," in *International Conference on Hybrid Systems: Computation and Control*, 2014, pp. 203–212.

[41] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 5163–5170.

[42] R. Majumdar, E. Render, and P. Tabuada, "Robust discrete synthesis against unspecified disturbances," in *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, ACM, 2011, pp. 211–220.

[43] K. W. Wong, R. Ehlers, and H. Kress-Gazit, "Correct high-level robot behavior in environments with unexpected events," in *Robotics: Science and Systems*, 2014.

[44] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 5420–5427.

[45] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy, "Reactive task and motion planning under temporal logic specifications," *arXiv preprint arXiv:2103.14464*, 2021.

[46] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," *arXiv preprint arXiv:2005.05842*, 2020.

[47] H.-W. Park, A. Ramezani, and J. W. Grizzle, "A finite-state machine for accommodating unexpected large ground-height variations in bipedal robot walking," *IEEE Transactions on Robotics*, vol. 29, no. 2, pp. 331–345, 2013.

[48] Y. Zhao, B. R. Fernandez, and L. Sentis, "Robust optimal planning and control of non-periodic bipedal locomotion with a centroidal momentum model," *The International Journal of Robotics Research*, vol. 36, no. 11, pp. 1211–1242, 2017.

[49] Y. Zhao and L. Sentis, "A three dimensional foot placement planner for locomotion in very rough terrains," in *IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2012, pp. 726–733.

[50] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Verification, Model Checking, and Abstract Interpretation*, Springer, 2006, pp. 364–380.

[51] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive, high-level robot control," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 65–74, 2011.

[52] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[53] Y. Zhao, B. R. Fernandez, and L. Sentis, "Robust optimal planning and control of non-periodic bipedal locomotion with a centroidal momentum model," *The International Journal of Robotics Research*, vol. 36, no. 11, pp. 1211–1242, 2017. eprint: https://doi.org/10.1177/0278364917730602.

[54] A. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, Jan. 2010.

[55] A. Hereid and A. D. Ames, "Frost: Fast robot optimization and simulation toolkit," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 719–726.

[56] M. Koptev, N. Figueroa, and A. Billard, "Real-time self-collision avoidance in joint space for humanoid robots," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1240–1247, 2021.

[57] V. R. R. Ehlers, *Slugs: Extensible gr(1) synthesis*, Springer, 2016.

[58] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 3859–3866.

[59] B. Isaacson, T. Swanson, and P. Pasquina, "The use of a computer-assisted research environment (caren) for enhancing wounded warrior rehabilitation regimens," *The journal of spinal cord medicine*, vol. 36, pp. 296–299, Jul. 2013.

[60] R. Tedrake, *Underactuated Robotics, Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. 2022.

[61] Y. Gong and J. Grizzle, "One-step ahead prediction of angular momentum about the contact point for control of bipedal locomotion: Validation in a lip-inspired controller," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[62] H. Sadeghian, C. Ott, G. Garofalo, and G. Cheng, "Passivity-based control of underactuated biped robots within hybrid zero dynamics approach," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[63] R. Hartley, M. G. Jadidi, R. M. Eustice, and J. W. Grizzle, "Contact-aided invariant extended kalman filtering for robot state estimation," *CoRR*, vol. abs/1904.09251, 2019. arXiv: 1904.09251.

[64] P. S. Maybeck, *Stochastic models, estimation and control / Peter S. Maybeck*. Academic Press New York, 1979, 3 v. : ISBN: 0124807011 012480702 0124807038.

[65] A. Barrau and S. Bonnabel, "The invariant extended kalman filter as a stable observer," *CoRR*, vol. abs/1410.1465, 2014. arXiv: 1410.1465.

[66] J. Xu, Z. Hou, Z. Liu, and H. Qiao, "Compare contact model-based control and contact model-free learning: A survey of robotic peg-in-hole assembly strategies," *CoRR*, vol. abs/1904.05240, 2019. arXiv: 1904.05240.

[67] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1470–1477.

[68] I. Georgievski and M. Aiello, "An overview of hierarchical task network planning," *CoRR*, vol. abs/1403.7426, 2014. arXiv: 1403.7426.

[69] J. Albus, H.-M. Huang, E. Messina, K. Murphy, M. Juberts, A. Lacaze, S. Balakirsky, M. Shneier, T. Hong, H. Scott, F. Proctor, W. Shackleford, J. Michaloski, A. Wavering, T. Kramer, N. Dagalakis, W. Rippey, K. Stouffer, and S. Legowik, *4d/rcs version 2.0: A reference model architecture for unmanned vehicle systems*, 2002-08-22 00:08:00 2002.

[70] F. Martın, J. Ginés, V. Matellán, and F. J. Rodriguez, "Plansys2: A planning system framework for ROS2," *CoRR*, vol. abs/2107.00376, 2021. arXiv: 2107.00376.

[71] C. Paxton, N. Ratliff, C. Eppner, and D. Fox, "Representing robot task plans as robust logical-dynamical systems," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 5588–5595.

[72] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *CoRR*, vol. abs/2010.01083, 2020. arXiv: 2010.01083.

[73] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15, Buenos Aires, Argentina: AAAI Press, 2015, pp. 1930–1936, ISBN: 9781577357384.

[74] Y. Zhu, J. Tremblay, S. Birchfield, and Y. Zhu, "Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs," *CoRR*, vol. abs/2012.07277, 2020. arXiv: 2012.07277.

[75] P. R. Florence, L. Manuelli, and R. Tedrake, "Dense object nets: Learning dense visual object descriptors by and for robotic manipulation," 2018. arXiv: 1806.08756 [cs.RO].

[76] L. Manuelli, W. Gao, P. Florence, and R. Tedrake, "Kpam: Keypoint affordances for category-level robotic manipulation," 2019. arXiv: 1903.06684 [cs.RO].

[77] D. Rodriguez, C. Cogswell, S.-Y. Koo, and S. Behnke, "Transferring grasping skills to novel instances by latent space non-rigid registration," May 2018.

[78] J. Su, R. Li, H. Qiao, J. Xu, q. Ai, and J. Zhu, "Study on dual peg-in-hole insertion using of constraints formed in the environment," *Industrial Robot: An International Journal*, vol. 44, pp. 00–00, Aug. 2017.

[79] Z. Qin, P. Wang, J. Sun, J. Lu, and H. Qiao, "Precise robotic assembly for large-scale objects based on automatic guidance and alignment," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 6, pp. 1398–1411, 2016.

[80] E. Roberge and V. Duchaine, "Detecting insertion tasks using convolutional neural networks during robot teaching-by-demonstration," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 3210–3216.

[81] D. De Gregorio, R. Zanella, G. Palli, S. Pirozzi, and C. Melchiorri, "Integration of robotic vision and tactile sensing for wire-terminal insertion tasks," *IEEE Transactions on Automation Science and Engineering*, vol. 16, no. 2, pp. 585–598, 2019.

[82] C. H. Kim and J. Seo, "Shallow-depth insertion: Peg in shallow hole through robotic in-hand manipulation," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 383–390, 2019.

[83] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine, "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards," *CoRR*, vol. abs/1906.05841, 2019. arXiv: 1906.05841.

[84] S. Dong, D. K. Jha, D. Romeres, S. Kim, D. Nikovski, and A. Rodriguez, "Tactile-rl for insertion: Generalization to objects of unknown geometry," *CoRR*, vol. abs/2104.01167, 2021. arXiv: 2104.01167.

[85] S. Balakirsky, C. Schlenoff, S. Rama Fiorini, S. Redfield, M. Barreto, H. Nakawala, J. L. Carbonera, L. Soldatova, J. Bermejo-Alonso, F. Maikore, *et al.*, "Towards a robot task ontology standard," in *International Manufacturing Science and Engineering Conference*, American Society of Mechanical Engineers, vol. 50749, 2017, V003T04A049.

[86] M. Ghallab, C. Knoblock, D. Wilkins, A. Barrett, D. Christianson, M. Friedman, C. Kwok, K. Golden, S. Penberthy, D. Smith, Y. Sun, and D. Weld, "Pddl - the planning domain definition language," Aug. 1998.

[87] *Pddl planning wiki*.

[88] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. arXiv: 1703.06870.

[89] X. Sun, B. Xiao, S. Liang, and Y. Wei, "Integral human pose regression," *CoRR*, vol. abs/1711.08229, 2017. arXiv: 1711.08229.

[90] W. Gao and R. Tedrake, "Kpam 2.0: Feedback control for category-level robotic manipulation," *CoRR*, vol. abs/2102.06279, 2021. arXiv: 2102.06279.

[91] S. Chiaverini, B. Siciliano, and L. Villani, "Parallel force/position control schemes with experiments on an industrial robot manipulator," *IFAC Proceedings Volumes*, vol. 29, pp. 25–30, 1996.

[92] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," May 2011, pp. 4569–4574.

[93] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: A moveit! case study," Apr. 2014.

[94] E. Marchand, F. Spindler, and F. Chaumette, "Visp for visual servoing: A generic software platform with a wide class of robot control skills," *IEEE Robotics and Automation Magazine*, vol. 12, no. 4, pp. 40–52, Dec. 2005.

[95] A. Coles, A. Coles, M. Fox, and D. Long, "Forward-chaining partial-order planning," Jan. 2010, pp. 42–49.

[96] USPS, *2019 postal facts - usps*.

[97] M. Joers, J. Schröder, F. Neuhaus, C. Klink, and F. Mann, *Parcel delivery The future of last mile*. McKinsey amp; Company, 2016.

[98] E. Ackerman, *Meet the lunar gateway's robot caretakers*, Apr. 2022.