INTEGRATING PERCEPTION & PLANNING FOR HUMANOID AUTONOMY

Philipp Michel

CMU-RI-TR-08-35

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN ROBOTICS

The Robotics Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213

July 2008

Thesis Committee:
Takeo Kanade, Co-Chair
James Kuffner, Co-Chair
Martial Hebert
Satoshi Kagami

ABSTRACT

Today's agile humanoid robots are testament to the impressive advances in the design of biped mechanisms and control in recent robotics history. The big challenge, however, remains to properly exploit the generality and flexibility of humanoid platforms during fully autonomous operation in obstacle-filled and dynamically changing environments. Increasing effort has thus been focused on the challenges arising for perception and motion planning, as well as the interplay between both, as foundations of humanoid autonomy.

This thesis explores appropriate approaches to perception on humanoids and ways of coupling sensing and planning to generate navigation and manipulation strategies that can be executed reliably. We investigate perception methods employing on- and off-body sensors that are combined with an efficient motion planner to allow the humanoid robot HRP-2 and Honda's ASIMO to traverse complex and unpredictably changing environments. We examine how predictive information about the future state of the world gathered from observation enables navigation in the presence of challenging moving obstacles. We show how programmable graphics hardware can be exploited to create a novel, model-based 3D tracking system able to robustly address the difficulties of real-time sensing specifically encountered on a locomoting humanoid. This thesis argues furthermore that reliability of autonomous operation can be improved by reasoning about perception during the planning process, rather than maintaining the traditional separation of the sensing and planning stages. We use the humanoid robots ARMAR-III and HRP-2 to investigate and validate such planning for perceptive capability in manipulation and navigation scenarios.

While humanoid robots serve as the motivating challenge and application domain for this thesis, much of the resulting work is general in nature and has applications in other areas of robotics and computer vision.

ACKNOWLEDGMENTS

This thesis would have been impossible without the help and support of the Robotics Institute community, friends and family. I would like to express my deep personal gratitude to several people in particular.

First and foremost to my advisors, Takeo Kanade and James Kuffner, for their guidance and support over the past few years and for enabling me to play with some of the most amazing robots on earth as my day job. I thank James for his contagious enthusiasm in pursuing novel research ideas and for fostering great research collaborations. I am grateful to Takeo for his uncanny ability to immediately get to the very core of any idea while always keeping the big picture in mind. His unwavering dedication to his students and his inspirational mentorship, both professional and in more philosophical matters of life, were invaluable. I am deeply honored to have been given the opportunity to work with James and Takeo.

Joel Chestnutt, my partner in research crime, deserves my most personal and honest gratitude. At once collaborator, advisor and friend, Joel is truly one of the smartest, kindest and most positive individuals I'll ever know. Working with him since day one at CMU and in Tokyo, chatting about robots and exploring Japan's culture together has been a pleasure. Thank you for everything, Joel!

I would like to thank my thesis committee members, Martial Hebert and Satoshi Kagami, for their keen insight and productive feedback while completing this thesis. I am thoroughly indebted to Satoshi Kagami and Koichi Nishiwaki for their collaborative efforts, constant support and extraordinary patience throughout my numerous stays at the AIST Digital Human Research Center. I thank Christian Scheurer for his hard work while collaborating on the work with ARMAR described at the beginning of Chapter 7. Dōmo arigatō to Bilge Mutlu for being a fellow ASIMO guy, helping out many times and for the discussions over coffee. Thanks also to Ross Diankov for developing, patiently supporting and helping out with OpenRAVE.

The extensive time I spent researching in Japan would not have been possible without the generous financial support I received from the Japan Society for the Promotion of Science, for which I am extremely grateful. I want to thank NVIDIA Corporation for awarding me an NVIDIA Fellowship and giving me access to some very cool GPUs. Thanks also to Honda R&D Corporation for giving me the opportunity to pursue research on the awesome ASIMO and for supporting it over the years.

Friendship is the only cement that will ever hold the world together. Or at least grad school. I made some great friends during my time at the RI and my life would be infinitely poorer without them. I thank Marek Michalowski, for is true friendship in times good and bad and for always being there in situations sane and not. And for co-founding the Dirty Robot Brew Works. Fred Heger for being the best roommate one could wish for, a good friend and a fellow German a looong way from home. Jenney and Matt McNaughton, for all the food, for rocking out and for being Canadian. Thanks to Geoff Hollinger for 80's night, Dr. Dhawal Nagpal for the pep talks from the other coast and Heather Hendrickson for all the Jello and good company. A massive thanks to my crazy posse in Tokyo—Andy McVitty in particular—for keeping me sane on weekends. Last but certainly not least, I thank I-Ching Ueng for her unwavering moral support in tough times and for keeping me happy and well-fed.

Most of all, I thank my parents, Herbert & Ingrid, without whom I am nothing and to whom I owe everything in this life.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

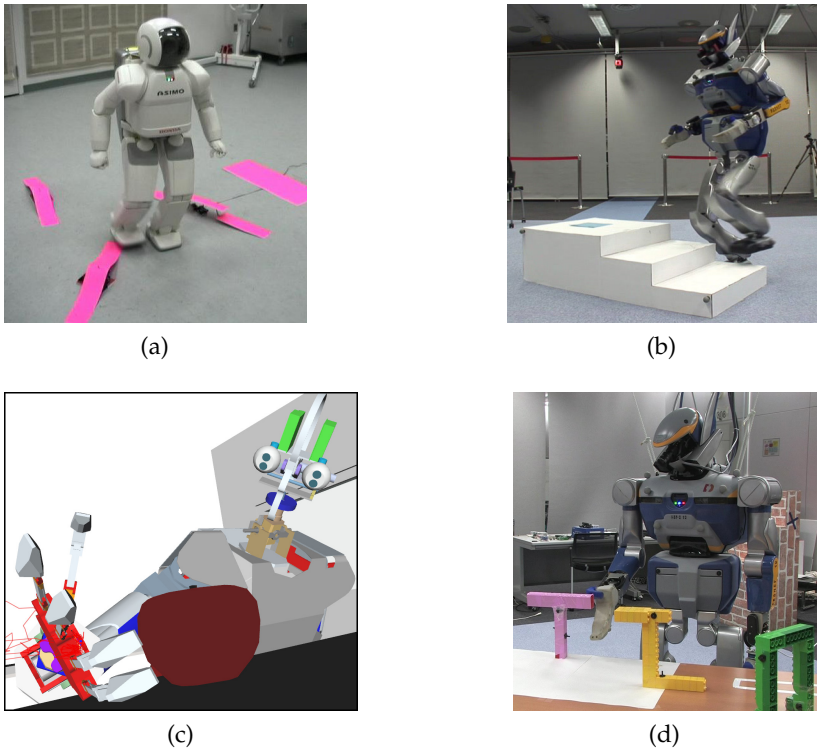
1

INTRODUCTION

Biped humanoid robots present perhaps the most fascinating yet challenging research platform in robotics today, with potential for wide applicability in the human-centric environments we inhabit. While the last decade has witnessed major advances in actuator and mechanism design as well as control that have granted robots such as Honda's ASIMO [75] impressively stable walking behavior, to date humanoids have served as little more than remarkable proof-of-concept platforms demonstrating the feasibility of robotic biped locomotion. The remaining challenges span the gamut of fundamental robotics problems, from low-level, energy-efficient control to natural human-robot interaction.

Until recently, relatively little effort was placed on equipping humanoid robots with the ability to use their newly acquired agility in an autonomous manner to accomplish real-world tasks. To become truly useful agents, humanoids should be able to perform a range of tasks with minimal to no supervision, beginning with core capabilities such as autonomous navigation of their surroundings and manipulation of objects within their environment. In addition to the fundamental control schemes successfully developed recently, this necessitates appropriate approaches to perception and planning to conceptually complete the 'sense-think-act' cycle. As will be discussed, mere transfer of prior perception and planning strategies from more traditional mobile robotics may often fail to meet many of the demands on responsiveness, robustness and accuracy arising from operation on a complex, high degree-of-freedom system such as a humanoid.

This thesis will present a series of perception strategies tailored to humanoid robots and their currently typical operating environments that are coupled with efficient planning methods to accomplish a number of real-world tasks autonomously, focusing specifically on agile navigation in the presence of dynamic obstacles and, to a lesser degree, on manipulation. The resulting implementations on the humanoids HRP-2 and ASIMO, exemplified in Figure 1.1, represent some of the current state of the



(a) (b)

(c) (d)

Figure 1.1: Examples of Humanoid Autonomy. The Honda ASIMO navigating a challenging 'gauntlet' of spinning blade-like obstacles *(a)*. HRP-2 autonomously localizing and climbing a set of stairs *(b)*. A simulation of the ARMAR-III humanoid grasping an apple in a kitchen environment *(c)* and HRP-2 reaching for an object *(d)*, both in a manner optimized for 'sensability' during execution.

art in humanoid autonomy. Our underlying emphasis is on exploiting robot, sensor, hardware, environment, domain and task characteristics to efficiently gather sensing data that is sufficient to reliably accomplish the task at hand, while remaining explicit enough that the humanoid's unique locomotion and manipulation abilities can be appropriately employed.

As part of this effort, we present a novel GPU-accelerated model-based 3D tracking system. While real-time use of this tracker as the perception component during autonomous humanoid operation remains the core focus, the state of the art system developed retains general applicability to a wide range of tracking applications, such as visual servoing problems, matchmoving or augmented reality.

We also make an argument for a novel tight coupling between perception and planning, with the planning stage explicitly taking into account the robot's future ability to sense the environment in every state considered by the planner. Our resulting implementations on the ARMAR-III as well as the HRP-2 robot, shown in Figure 1.1, indicate that such planning for 'future perceptive capability' during manipulation and navigation has the potential to ensure higher task success rates for any task requiring visual feedback during execution than traditional perception-unaware planning.

1.1 THESIS OVERVIEW

This thesis presents our work on integrating perception and planning for humanoid autonomy as follows. In Chapter 2, we describe and formalize the problem of coupling perception, planning and execution in the context of autonomous humanoid operation and identify the key motivating challenges encountered when operating on a locomoting humanoid robot. Chapter 3 discusses related work in the areas of perception and motion planning, as well as their implementation and integration on humanoid robots. An initial approach to a fully integrated perception-planning-action system employing off-body sensing and allowing the Honda ASIMO humanoid to walk in rapidly changing environments is presented in Chapter 4. In Chapter 5, we move perception on-body for the HRP-2 humanoid and combine it with global localization to perform fast environment reconstruction for navigation autonomy. Chapter 6 presents and evaluates the GPU-accelerated 3D tracking system constituting a substantial contribution of this thesis, and presenting an end-to-end solution to on-body perception of robot frameworks used in subsequent chapters. A second major thesis contribution, the concept of planning for future perceptive capability, is first detailed in Chapter 7 and applied to the problem of planning reaching trajectories for manipulation on the ARMAR-III and HRP-2 humanoids. Chapter 8 examines and evaluates the same concept in the context of autonomous navigation planning. Finally, we summarize the contributions of this thesis in Chapter 9 and discuss future directions.

2

PROBLEM DESCRIPTION

We begin by attempting to identify key issues and to unify and characterize the roles of perception and planning in the context of humanoid autonomy. We partially adopt notation from the information space framework as used by LaValle [42] to aid in formalizing the interaction between perception, planning, and execution.

2.1 FORMALIZATION

Consider a task T to be accomplished autonomously by the humanoid as being defined by a series of abstract actions $T = \{u_1, \ldots, u_n\}, u_k \in U$ that, when executed by the robot, yield the desired behavior. Without loss of generality, T might represent the task of navigating to a specific goal location, climbing an obstacle or reaching for a certain object in the environment. We now seek to autonomously discover such a sequence of actions through planning.

While for all problems of interest to us perfect state information is unattainable, let $x_k \in X$ denote the true state state of the robot and the environment at a particular stage k, with x_0 denoting the true initial state. Transitions between states are given by $x_{k+1} = f(x_k, u_k, \theta_k)$, where $\theta \in \Theta(x, u)$ encapsulates the interference of nature (as uncertainty, modeling error, etc.) on the execution of action u in state x. Perception generates sensor observations $y_k = h(x_k, \psi_k)$ that are determined by the sensor model h and affected by nature sensing actions (e.g. noise) $\psi \in \Psi(x)$. The history of such observations can then be used to update an environment description $e_k = \{y_1, y_2, \ldots, y_k\} \in E$. As mentioned, perfect state information is not achievable. Assuming, however, that we are able to constrain our initial condition to at least lie somewhere in a space of potential initial conditions $\mathcal{I}_0$, we can summarize all information available at stage k into a state in information space $\mathcal{I}_k = \{\mathcal{I}_0, (u_1, \ldots, u_{k-1}), (y_1, \ldots, y_k)\}$, consisting of our initial condition space, all actions applied up to this point, and all sensor observations thus far.

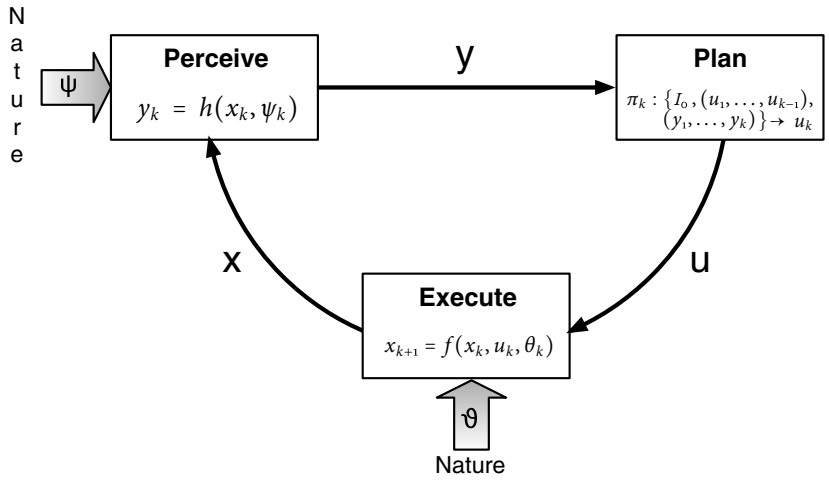


Figure 2.1: The perceive-plan-execute cycle. Perception and planning are coupled through execution. How perception affects planning is often considered, while the effects the planning stage may have on later perception are largely ignored.

The task of the planner is then conceptually to find a information feedback plan $\boldsymbol{\pi} : \mathcal{I} \mapsto U$ that decides an appropriate action to execute in each information state and that, when applied in the initial state, yields a sequence of actions in accordance with the task T to be accomplished. Note that this need not remain fixed throughout execution. In fact, the plan *should* adapt throughout execution via re-planning to changes in the environment and to compensate for execution errors. It is thus more reasonable to consider $\boldsymbol{\pi}$ as a multi-stage feedback plan $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_K\}$ that allows for a different plan π_k to be used at each stage. Figure 2.1 summarizes the relationships presented here.

These definitions allow us to formally derive the interdependence between perception and planning. Consider expanding the equations defining the perception process in stage k:

$$\boxed{y_k} = h(f(x_{k-1}, \boxed{\pi_{k-1}}(\mathcal{I}_{k-1}), \theta_{k-1}), \psi_k) \tag{2.1}$$

It is evident that decisions made previously by the planner, through execution, affect the sensor observations in stage k. Conversely, consider the information taken into account by the planner in stage k:

$$\boxed{\pi_k} : \{\mathcal{I}_0, (u_1, \ldots, u_{k-1}), \boxed{(y_1, \ldots, y_k)}\} \mapsto u_k \tag{2.2}$$

Here, the planning process evidently relies on prior sensor observations in determining the next control input u_k to generate.

The reliance of planning on sensed data, given in Equation 2.2, via cost maps, environment reconstructions, etc. is universally taken for granted. However, the fact that decisions made during planning may significantly affect future iterations of perception, evidenced in Equation 2.1, has received less attention thus far.

2.2 CONSIDERATIONS

Given this cyclic interdependence of the perception and planning processes, together with the use of a biped humanoid as the robotic platform, several important considerations for successful autonomous operation arise. The issues outlined here serve as motivating challenges to the work in this thesis and the approaches outlined in Chapters 4–8.

RESPONSIVENESS When operating on a walking humanoid, the time-frame in which a single cycle of perceive-plan-execute must complete to ensure responsiveness to environment changes or execution errors is very limited. As compute-heavy tasks, perception and planning are particularly affected. Unlike it is the case for wheeled robots, pausing movement for deliberation or to gather sensor readings is usually not an option—perception must operate in real-time and (re-)planning must remain similarly responsive, often iterating once per step cycle. During operation in highly dynamic environments, these constraints on responsiveness suggest that significant benefit may be gained from reasoning predictively about the state of the world during both perception and planning.

ROBUSTNESS The effects of nature (via θ and ψ above) on perception and execution on a high degree-of-freedom system can be particularly severe. Complex kinematics imply that a small change in robot configuration may cause significant changes in the sensor view of the scene that are difficult to predict, and may also give rise to self-occlusion issues. Furthermore, large camera displacement and camera shakiness occur naturally during swift humanoid locomotion. All of these issues should be addressed by the perception system to enable informed decisions to be made in the planning stage.

SCOPE & DESCRIPTIVENESS OF PERCEPTION DATA For perception to provide the planning stage with useful representa-

tions to operate upon, the scope of environment maps must be high-level enough such that motions of meaningful scale (e.g. navigation across a room) can be planned for. The focus is thus on on the generation of 'global', rather than robot-centric environment descriptions, and on the integration of local measurements in a globally consistent manner. Additionally, maps should encode enough information to allow for appropriate use of the humanoid's unique manipulation and stepping capabilities. For instance, presence of a height component may be crucial for maps used during legged locomotion.

ACCURACY Environment information recovered by perception is subject to restrictive error tolerances if it is to be successfully coupled with the robot's walking controller to enable the robot to successfully step onto surfaces or avoid obstacles. When planning stair-climbing motions, for instance, the error in recovering the height of each step may usually not exceed a few millimeters to ensure proper foot-ground contact during execution of the planned leg trajectories.

SAFETY & RELIABILITY Prior approaches to motion planning usually emphasize optimality of generated plans in terms of distance measures such as path length or in terms of resource expenditure in traversing said path. For the current state-of-the-art humanoids and their typical indoor environments, it is perhaps more appropriate to focus on safety and reliability of path execution as the desirables to optimize for. Note how both the perception and the planning process are affected by this prioritization. For instance, a highly optimal path may be of very little use to the robot if it leads through an area of the environment that provides poor sensor operating conditions if sensor feedback is required throughout execution.

MUTUAL AWARENESS Traditionally, perception and planning are conceptualized as mostly separate processes, with planning relying on (at best probabilistic) representations output by perception, but not considering the effects planned actions may have on perception in the future. Likewise, perception often operates on a 'best-effort' basis, attempting to supply the best data possible given the sensors and computation available. Knowledge of the planning process that might be exploited for reliability or efficiency is seldom regarded. It may be beneficial to advocate a tighter coupling between perception and planning, informing

each process about the capabilities and limitations of the other, to effectively yield 'planning-aware perception' and 'perception-aware planning'.

Given the prior problem description and characterization, we posit that a truly successful approach to integrated perception and planning for humanoid autonomy should carefully take these outlined considerations into account and may furthermore benefit significantly by realizing and appropriately exploiting the mutual interdependence of the perception and planning processes.

3

RELATED WORK

Before proceeding to study integrated perception and planning on humanoids in depth, we now consider related work this thesis touches upon and aim to give context for the approaches outlined later in this document. This chapter seeks to give a general overview of related work. More specific references are given throughout the remainder of this thesis as appropriate.

3.1 PERCEPTION-GUIDED HUMANOID AUTONOMY

Most of the literature in autonomous robotic bipedal walking has concentrated on various approaches to reliable, stable gait generation and feedback, while relatively little work has focused on developing global locomotion autonomy for biped robots. Emphasis has primarily been on pre-generating walking trajectories [28, 88, 58], online trajectory generation [59, 60] and dynamic balance [70], without accounting for obstacles.

Obstacle avoidance and local planning based on visual feedback has been studied in humans [66, 67] in order to gain insight into potential robotic implementations. Several previous systems have focused on representing the robot environment as 2D occupancy grids [54] or 2.5D height maps [25], using standard vision techniques as well as stereo [31]. Given knowledge about the environment, many biped navigation approaches thus far have implemented reactive perception-based obstacle avoidance [86, 87, 47, 14]. Such methods either do not account for the global configuration of obstacles between the robot and the desired goal, or do so in a limited way, which makes locomotion in truly cluttered environments challenging. Others accomplish obstacle avoidance using approximate and more traditional path planning techniques [26, 74] that do not fully employ the unique stepping capabilities of bipeds. Planning at a different level of abstraction, such as footsteps [37, 38, 11] has allowed these capabilities to be directly exploited in order to compute efficient global navigation strategies for humanoid robots.

Chapter 6 presents a flexible, fast and reliable approach enabling the humanoid HRP-2 to climb stairs. Several bipeds have successfully accomplished stair climbing previously. Sony's QRIO robot uses stereo to reconstruct stairs and climb them gradually, step-by-step [24]. The Johnnie walking robot has successfully and quickly climbed stairs detected reactively using vision [47, 14]. Honda's ASIMO humanoid [28] first positions itself precisely with respect to a set of stairs equipped with fiducials and then executes a fixed footstep sequence, adjusted according to the contact force with each step, to climb them. The H7 humanoid [61] has also successfully climbed a set of stairs positioned in front of it. Most prior stair-climbing approaches carry some a-priori knowledge of the shape or location of the stairs, or alternatively tend to carefully execute each stepping-up motion in isolation, foregoing continuous execution and smoothness for safety.

3.2 SENSING ON HUMANOIDS

The importance of appropriately descriptive and scoped representations of sensed data recurs as a theme in this thesis. Fully general estimation of the 3D scene geometry and camera egomotion by image feature tracking and structure from motion has been extensively studied [79]. However, its computational complexity has prevented efficient online robotic implementations. Instead, recent research has focused on combining stereo correspondence with robot-centric maps [74] or with localization using dead-reckoning or visual odometry, leading to several successful on-body implementations for humanoids [78, 32]. However, while environment mapping and obstacle detection from onboard stereo processing [25, 62] allow three dimensional obstacles to be localized, stereo implementations often cannot satisfy the delay and accuracy bounds required to have the robot walk near, on, or over obstacles during real-time locomotion.

Chapter 6 studies a GPU-accelerated, model-based tracking system running online on a humanoid robot for autonomous locomotion. As will be shown, such a perception framework addresses and properly satisfies many of the challenges and considerations for operation on a humanoid outlined in Section 2.2, especially with regards to responsiveness, robustness and accuracy of perception. A large body of work exists relating to model-based 3D object tracking and associated visual servoing approaches [45]. Early work by Gennery [22] first focused on tracking objects of known 3D structure, with Lowe [49, 50] pioneering the fitting

of model edges to image edges. Harris' RAPiD tracker [27] first achieved such fitting in real-time, with a range of improvements to the original algorithm having been proposed [1], causing edge- and contour-based tracking systems to maintain significant popularity [17, 18, 39, 13]. Other approaches employ appearance-based methods to perform tracking [30] or view tracking as a combination of wide-baseline matching and bundle adjustment relying on so-called keyframe information gathered offline [80].

There is an ever increasing body of work regarding the use of GPUs for general purpose computation. Several good overview resources exist [68, 65]. Particularly relevant to perception is the work by Fung et. al. [19]. The use of GPUs in application areas related to motion planning, computational geometry and simulation has also significantly increased in recent years [76, 23, 21].

3.3 COUPLING PERCEPTION & PLANNING

The topic of Chapter 7 and Chapter 8, advocating a tighter coupling between perception and planning by reasoning about the former during the latter, relates to the issues of sensor planning [72] and the next-best-view problem [69]. Traditionally, these are often considered independently from the problem of planning the actual robot motions themselves. In light of their natural interdependence outlined previously, such conceptual separation between planning for perception and planning for the task at hand may be somewhat limiting.

In the SLAM community, explicit reasoning about perception during navigation (e.g. in the form of information gain [7]) has resulted in significant progress in robot autonomy. The mobile and multi-robot community has studied the problem of planning with constraints on visibility in a variety of settings, e.g. to maintain line-of-sight communications constraints between a team of exploring robots [33], or to ensure visibility of a moving target [41]. However, scenarios considered are often amenable to purely geometric reasoning about visibility, and make application to high-DOF systems challenging. Coastal navigation techniques [73] seek to maximize the number of landmarks seen along a path to maintain localization. Such approaches tend to be somewhat less general in nature, applying mainly to the task of localization during navigation, and requiring specific ways of statistically computing information gain given a particular

sensor and known environment at hand. Other work seeks to combine navigation and view planning by hallucinating potential look-ahead sensor observations to extend the path planning horizon and achieve a reduction in path length [57]. Also related is work [20] that computes a 'confidence-of-perception' measure to decide where to next perform sensing, effectively tackling the view planning problem.

Chapter 7 considers coupling perception and planning given the scenario of a robot performing manipulation tasks (e.g., reaching and grasping motions) under visual feedback in a free-form environment. Most related to our reasoning is work that explicitly considers visual servo performance during the planning process by augmenting the robot configuration with image-features to arrive at the so-called perceptual control manifold [77]. Limiting assumptions include the use of a static, external camera and the ability to analytically calculate the presence of image features in a particular configuration using forward kinematics.

Work on visual-servo control [29, 40] suggests that errors in the early sensing stages (due to occlusions, calibration error, low resolution, distance from the object of interest, etc.) may have a significant effect on the overall performance of robot motions controlled by visual feedback, and implies that sensing should hence be reasoned about throughout the planning process as well.

4

PERCEPTION-GUIDED HUMANOID NAVIGATION PLANNING

In light of the motivating challenges outlined in Section 2.2, how should one instrument appropriate approaches to humanoid perception and couple them with planning to achieve autonomy? How do we equip humanoids with the necessary perception skills needed to autonomously and reliably navigate, manipulate and interact in everyday human environments? This and the following two chapters focus specifically on perception strategies for tackling the problem of autonomous biped navigation and locomotion in a series of increasingly unrestricted environments and given more difficult tasks to be accomplished. By examining the fully integrated perception-planning-execution systems developed here, we seek to provide insight into possible solutions to the main research challenges given before.

4.1 VISION-GUIDED FOOTSTEP PLANNING FOR DYNAMIC ENVIRONMENTS

Consider a humanoid robot seeking to autonomously navigate to a certain location in the presence of a number of potentially unpredictably moving obstacles that need to be avoided by virtue of exploiting the humanoid's unique ability to step around or over obstacles. Such a scenario reflects some of the basic demands of obstacle-avoidance present in our typical surroundings and requires both accurate and frequently updated information regarding the configuration of the obstacles and the robot location. Additionally, we need an efficient method of generating valid walking paths to the goal that avoid obstacles and are amenable to adjustment should environment changes be detected.

We address this problem specifically for the Honda ASIMO humanoid by combining fast, vision-based sensing with an efficient planner operating at the level of footsteps [54]. An environment map including the robot, goal, and obstacle locations is built in real-time from vision. The planner then computes an optimal sequence of footstep locations within a time-limited planning

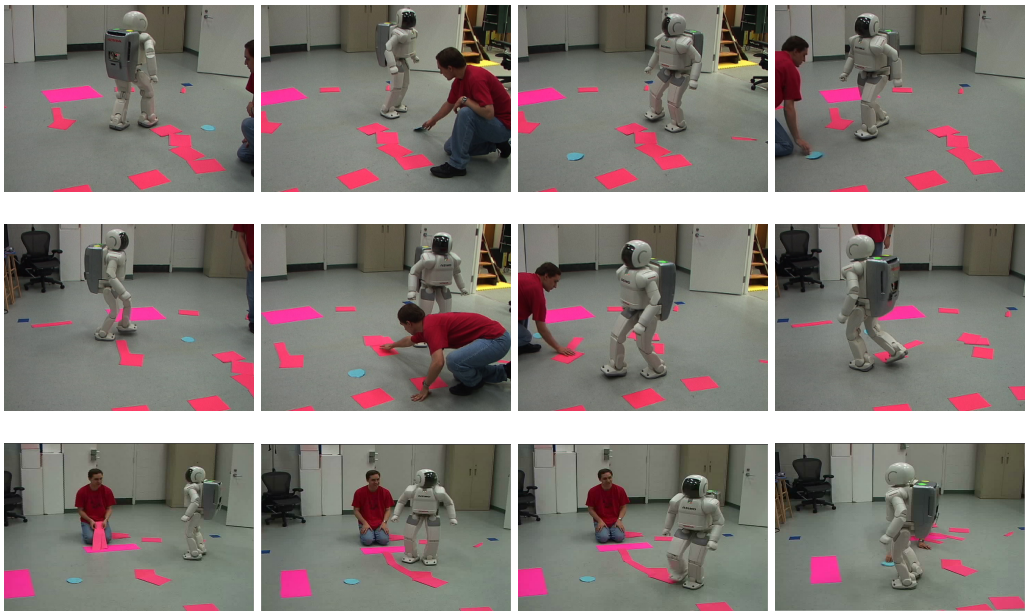


Figure 4.1: ASIMO navigating in an environment with unpredictably moving obstacles (pink, moved by an experimenter) and a changing goal location (light blue).

horizon. The resulting plans are reused and only partially recomputed as the environment changes during the walking sequence.

4.2 SENSING THE ENVIRONMENT

In our scenario, shown in Figure 4.1, a Honda ASIMO humanoid operates in a laboratory room containing a number of planar obstacles placed on a hard, flat floor. We employ an overhead camera to compute the position and orientation of the robot (via two makers attached to ASIMO's 'backpack'), the desired goal location, also indicated by a marker, and all obstacles, idealized by simple colored cardboard markers on the floor. This leveraging of the characteristics of ASIMO's operating environment as well as domain knowledge of the navigation task allows us to directly address several of the considerations of Section 2.2 above. In particular, we can generate appropriately *scoped* global maps of the environment containing all pertinent obstacles. The maps and robot localization are *descriptive* enough to encode all information required to plan footstep sequences including step-over motions. There is no cumulative mapping error or drift, meaning our maps and recovered robot position remain *accu-*

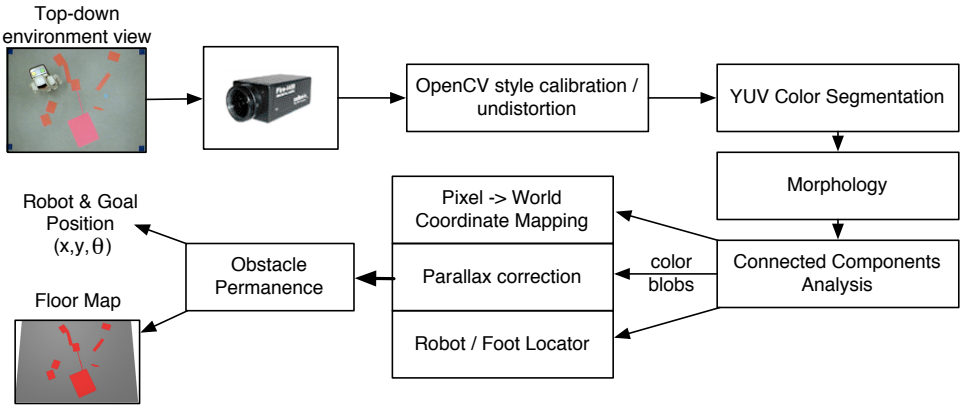


Figure 4.2: Overview of sensing. An environment map of obstacles on the floor and the robot/goal location and orientation are computed from ceiling-based vision.

rate. All vision processing employed operates in real-time on commodity hardware, guaranteeing *responsiveness* of perception.

We perform a step of color segmentation (using thresholds sampled a priori) directly on the YUV stream of images undistorted on the fly as supplied at 25fps by our FireWire camera, which is calibrated once during an off-line phase. Such segmentation [8, 82] has often been successfully employed when speed and accuracy are paramount. After a step of morphological post-processing to eliminate residual noise, we are left with a series of binary masks indicating presence/absence of each marker at each pixel. We group pixels corresponding to the same marker into blobs via a step of connected components labeling. Calculating blob moments then yields the location of each obstacle in image coordinates, area, orientation, and major/minor axes. The flow of vision processing is given in Figure 4.2.

The known geometry of the real-world floor area covered by the camera view (or, alternatively, knowing the camera's distance from the floor) allows us to then convert each blob's pixel coordinates to metric world coordinates simply via scaling. The angle that the line connecting the robot's backpack markers makes with respect to the horizontal gives the orientation of the robot.

4.3 LOCALIZATION & MAPPING

To perform footstep planning, the precise location of the robot's feet at any particular instant in time is required. To determine

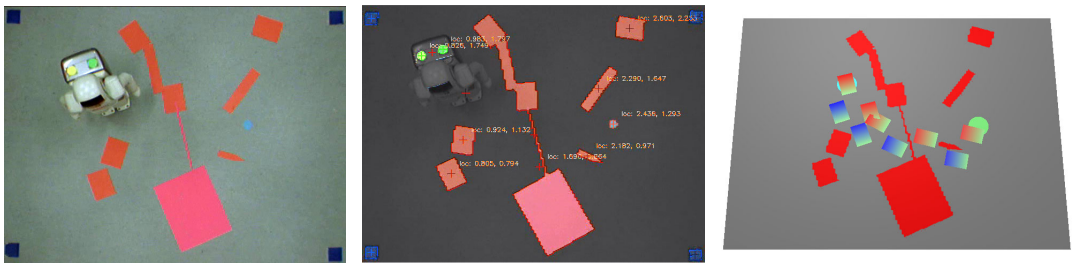


Figure 4.3: Top-down view of the walking environment *(left)*, vision system output *(center)*, and corresponding environment map with current footstep plan generated *(right)*.

the exact foot location, we first query the robot's kinematics in real-time to determine the location of the support leg's heel relative to the backpack markers. This information, together with knowledge of which stepping motion is currently being executed and how far into the motion we currently are allows us to determine the precise foot locations of both feet to within 2cm in world coordinates.

A 2D grid of binary valued cells represents the environment. The value in each cell denotes whether the corresponding terrain is valid (obstacle free) or invalid (totally or partially occupied by an obstacle). The environment map is thus a compact bitmap representation of the free space and obstacles as seen from the ceiling. Figure 4.3 shows a typical ceiling view of the environment, the robot and obstacles tracked by the vision system, and the corresponding map generated.

The use of off-body, global sensing does give rise to a problem of obstacle occlusion. Even if the robot is perfectly centered in the camera view (standing directly below the camera), obstacles close to the feet may be occluded by the robot's own body. The occluded area increases as the robot moves towards the edges of the image. Since obstacles in the occluded area cannot be seen by the camera, they will show up as free space in the environment map used by the footstep planner, causing the robot to potentially step directly onto an obstacle. We overcome this problem by implementing obstacle permanence in a rectangular area bounding the robot in the image. The cells corresponding to the permanence region in the environment map are not updated from the last value they had before entering the area. This reflects the assumption that whatever obstacles were last seen in the area currently being occluded by the robot are still present now. This rectangular approximation to the shape of the occluded region

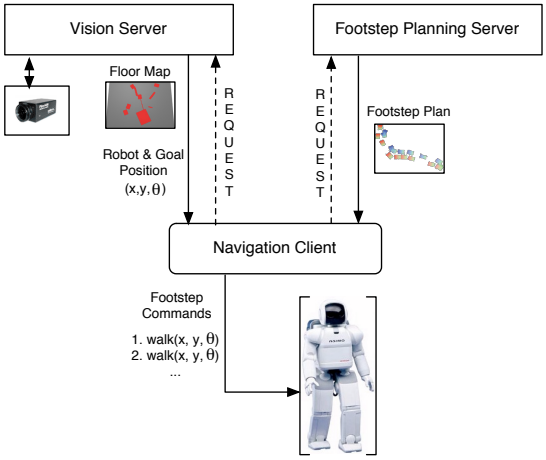


Figure 4.4: ASIMO perception/planning system components. The navigation client gathers updated sensing data from vision, requests for a new footstep plan to be computed and sends the commands required to execute the path to the robot.

works well in our trials. More elaborate approaches involving a top-down projection of a 3D model of the robot in its current state to determine the occluded pixels exactly are a possibility.

4.4 PLANNING & INTEGRATION

Our footstep planner ignores the lower level details of leg movement and control as much as possible, employing a higher-level action set that describes ASIMO's abilities in terms of possible footholds that can be reached in one step from any particular configuration. Full details are given in [54, 11]. During operation, we have ASIMO execute only the first step of the goal directed walking sequence the planner generates, before replanning for the next step, to adjust for possible environmental changes that may have occurred during that step. To increase our planning horizon within our limited planning time, we reuse some of the computation performed for the previous plan, by seeding our footstep search queue with it. At each step cycle, after receiving updated sensing data, we check if the previous plan reaches the goal without obstacle collisions, in which case it can be used without modification. If the previous plan does not reach the goal, the footstep planning resumes from the place where it failed, with all of the steps up to the impasse already in the queue.

Figure 4.4 shows how perception and planning are integrated as a system for ASIMO. A new footstep plan is computed from the current environment map and positioning data every time the robot reports that its stance foot has changed during the walking sequence (indicating completion of a footstep). As soon as an updated plan is available, the next command in the computed stepping sequence is sent to the robot. The change in stance foot is thus the discrete event which triggers replanning based on new environment data and transmission of commands to the robot. The time between footsteps, which averages 0.8s, bounds the planner's time horizon and hence determines the rate at which the perception-planning-action loop runs.

4.5 RESULTS: ASIMO & UNPREDICTABLE OBSTACLES

Figure 4.1 shows ASIMO walking in the laboratory environment while the surrounding obstacles and the goal location are being changed unpredictably by a human. The task of the experimenter is to intentionally block off the currently computed path to the goal, forcing the planner to recompute the path, reusing from previous plans any footsteps leading up to the occlusion. The goal marker is moved every time the robot is about to reach it, forcing ASIMO to turn and walk into the new direction of the goal.

During 10–20 trials lasting up to 10 minutes each, ASIMO continuously traverses the room without stepping on obstacles. We find the accuracy of perception to allow the robot to step within less than 2cm of the obstacles. The vision-guided planner reacts quickly to the dynamic environment, returning a new path within one second of a change in the obstacle configuration or goal position. In many cases a path is computed that causes the robot to either step over parts of an obstacle ("cutting corners") or traversing it altogether if the obstacle is small enough to clear in one step. The biped capabilities of the humanoid are thus properly leveraged by our approach.

4.6 PREDICTING OBSTACLE MOTION FROM OBSERVATION

There are of course improvements to the change-driven replanning outlined here. In environments where obstacles move in a steady manner and changes thus exhibit some degree of pre-

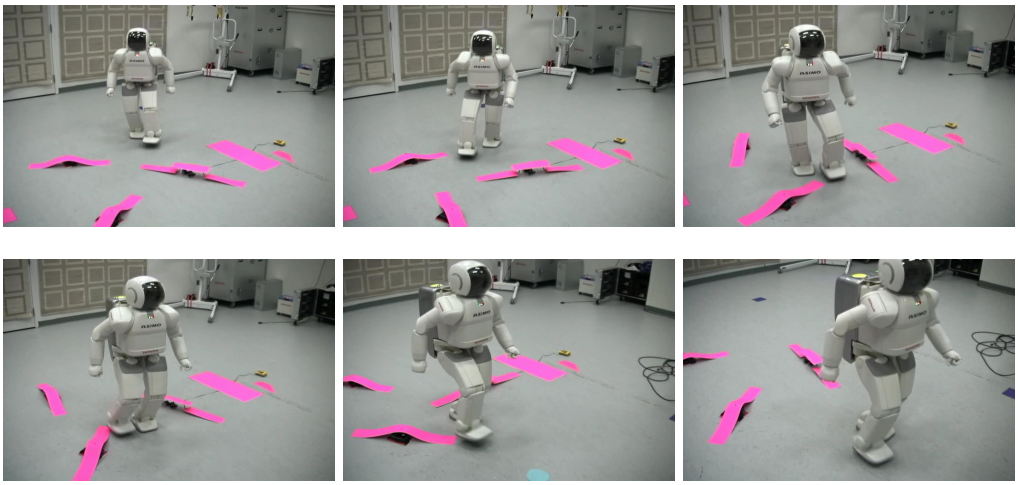


Figure 4.5: The Honda ASIMO autonomously navigating a gauntlet of blade-like obstacles spinning at different velocities.

dictability, the motion trajectories of obstacles can be inferred from observation, allowing for reasoning about not just the *current* state of the world, but also *future* environment configurations.

For the case of ASIMO, we augment the perception process to not only report the configuration of obstacles, but also estimate their velocities. Each obstacle in the environment has its position and orientation tracked using a Kalman filter updated by measurements from vision at approximately 25 frames per second. This allows the linear and rotational velocity components of each obstacle to be estimated with enough precision for the planner to accurately predict their future configuration. The robot is then able to plan through dynamic environments in which the obstacles are constantly moving, but where the motion is not known in advance. Without the velocity information we gather, the robot could still replan every step to account for updated obstacle positions, but every path it planned would quickly be invalidated by the changing state of the world. The estimation of future world conditions allows planning approaches to generate much safer trajectories in dynamic environments.

Figure 4.5 shows ASIMO traversing an environment with several long blade-like obstacles attached to motors such that they spin with a constant velocity. We have varied the number, direction and speed of the spinning obstacles, with the robot able to successfully navigate these gauntlets and reach its goal. The interplay between perception and planning effectively enables ASIMO

to seek 'windows of opportunity' to traverse these challenging scenarios, reflecting reasoning comparable to what humans apply e.g. when walking through revolving doors or when crossing streets with moving traffic.

5

COMBINING LOCAL AND GLOBAL SENSING FOR ONLINE ENVIRONMENT RECONSTRUCTION

As seen in the case ofASIMO, where an off-board camera observed all relevant parts of the environment, the *scope* of the representations generated by perception can be crucial for successful autonomy. To properly plan walking paths to a specific goal location, larger scale global maps are desirable. However, restricting perception to employ only off-board sensing sacrifices robot autonomy and can be susceptible to a range of sensing limitation arising from the fixed scene view, such as occlusions and lack of detail or resolution. On-body perception, however, is inherently limited in range and direction and the robot-centric data it recovers has traditionally only allowed reactive obstacle avoidance and other short-term navigation strategies. Ideally, we would like to employ a variety of sensors present on the humanoid robot and integrate their measurements into a globally consistent representation of the robot environment that can be used for planning autonomous tasks.

To this end, we investigate an approach to environment reconstruction that utilizes both external sensors for global localization, and on-body sensors for detailed local mapping [55]. We use an external optical motion capture system to accurately localize on-board sensors on the humanoid HRP-2. The localization data provided is *accurate*, free of cumulative errors and can be acquired almost instantaneously, ensuring *responsiveness*. Successive 2D views of a calibrated camera and range measurements from a time-of-flight sensor can then be integrated to construct global environment maps in real-time. Accurate external localization allows for a 2D occupancy grid of the floor area to be recovered from the video stream using image warping and segmentation. Similarly, distance measurements from a range sensor can be converted into 2.5D height maps of the robot's surroundings. In both cases, maps are constructed by integrating sensor data accumulated over time as the robot moves through the environment. While each sensor measurement thus only reconstructs a part of the surroundings seen from a partial view of the scene,

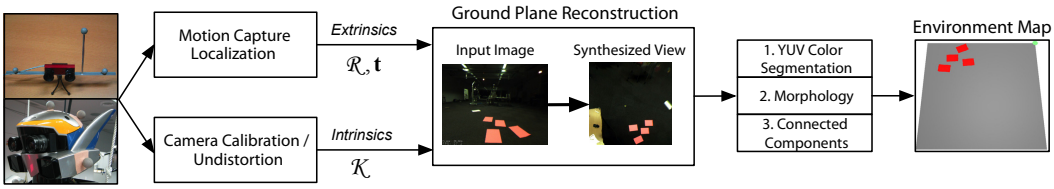


Figure 5.1: Overview of the image-based reconstruction process. An environment map of obstacles on the floor is constructed from a calibrated, moving camera localized using motion capture.

accurate sensor localization allows successive measurements to be correctly placed in a global map of the robot environment, which can then be used for planning. The generated maps exhibit appropriate *descriptiveness* for the navigation task.

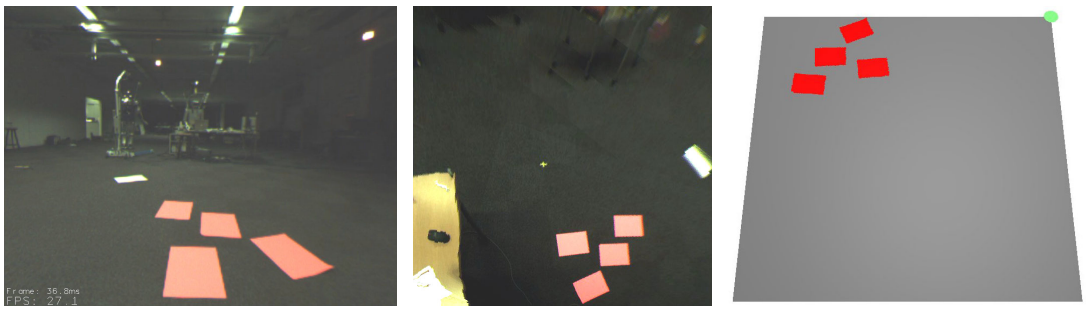
5.1 HOMOGRAPHY-BASED GROUND PLANE RECONSTRUCTION

By outfitting an indoor laboratory room with a modern motion capture system (though any method of acquiring reliable indoor location measurements would work), we are able to accurately recover the location and orientation of a camera in terms of a rigid transform defined by a translation vector $\mathbf{t}$ corresponding to the world coordinates of the camera's optical center as well as a 3×3 rotation matrix $\mathcal{R}$ representing the direction of the optical axis. These extrinsic parameters, when taken together with intrinsics $\mathcal{K}$ recovered during calibration, define the full camera projection matrix $\mathcal{M}$. From this, we can establish a 2D collineation, or homography $\mathcal{H}$, between the floor and the image, allowing incoming camera images to be warped onto the ground plane, effectively yielding a view of the robot surroundings as if seen from a virtual camera mounted overhead and observing the scene in its entirety. A step of obstacle segmentation then produces a 2D occupancy grid of the floor. All processing described is done in real-time or faster on commodity hardware. Figure 5.1 gives an overview of the reconstruction process.

Consider the full 3×4 camera projection matrix $\mathcal{M}$, defined by the 3×3 upper triangular matrix of intrinsics $\mathcal{K}$ and the extrinsics gathered from motion capture

$$\mathcal{M} = \mathcal{K} \begin{bmatrix} \mathcal{R} & \mathbf{t} \end{bmatrix} \tag{5.1}$$

$\mathcal{M}$ uniquely maps a scene point (X, Y, Z) to a point on the image plane (u, v). This mapping is not uniquely invertible, yielding for



(a) (b) (c)

Figure 5.2: Example camera image *(a)*. Synthesized top-down view *(b)* of the ground plane. Corresponding environment map generated *(c)*.

each image point only the equation of a ray in space along which each scene point must lie. Suppose, however, that all points of interest lie in the $Z = 0$ plane (i.e. the 'ground plane'). Such a planarity constraint now allows ground plane points $\mathbf{q} = (X, Y, 1)^T$ in homogeneous coordinates to be related to points $\mathbf{p} = (u, v, 1)^T$ in the image plane via a 3×3 ground-image homography matrix $\mathcal{H}$ as $\mathbf{p} \equiv \mathcal{H}\ \mathbf{q}$. $\mathcal{H}$ can be constructed from the projection matrix $\mathcal{M}$ by considering the full camera projection equation

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ m_1 & m_2 & m_3 & m_4 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.2}$$

and realizing that the constraint $Z = 0$ cancels the contribution of column m_3. $\mathcal{H}$ is thus simply composed of columns m_1, m_2 and m_4, yielding the desired 3×3 planar homography, defined up to scale with 8 degrees of freedom.

The recovered homography matrix is square and hence easily inverted. $\mathcal{H}^{-1}$ can then be used to warp incoming camera images onto the ground plane and thus accumulate an output image, resembling a synthetic top-down view of the floor area. The ground plane image is thus incrementally constructed in real-time as the camera moves through the scene, selectively yielding information about the environment. Updates to the output image proceed by overwriting previously stored data. The recovered view implicitly supports varying level of detail, achieved by moving the camera closer to the scene area of which a higher resolution reconstruction is desired. Figure 5.2a shows a camera image from a typical

sequence and Figure 5.2b the corresponding synthesized floor view. It is worth noting that any scene elements significantly violating the planarity constraint will appear distorted in the reconstruction. Once again employing colored markers to denote obstacles and obstacle segmentation as with ASIMO, we can use the synthesized floor view to generate binary environment maps of an arbitrary grid size for planning. Figure 5.2c gives an example.

5.2 HEIGHT MAPS FROM RANGE SENSOR DATA

Height-based maps, often referred to as 2.5D maps since they cannot represent vertical concavity, are the representation best matched to a humanoid's inherent stepping abilities. The additional depth measurement provided by a range sensor, when combined with accurate external localization using motion capture as before, enables us to cumulatively construct such height maps in real-time. We use a time-of-flight infrared range sensor [64] that is compact and lightweight enough to enable flexible placement on the humanoid's body. Our sensor yields 124×160 distance measurements at 30fps, has a 7.5m operating range and sub-centimeter resolution once calibrated for distance and exposure against a featureless plane at a known distance. We also estimate the sensor's instrinsics during a standard off-line calibration step [89].

Given the calibrated sensor, we are able to convert per-pixel distances (u, v, d) into camera-centric 3D coordinates $\mathbf{P}_C = (X, Y, Z)^T$ of the measured scene points. The extrinsic parameters $\mathcal{R}$ and $\mathbf{t}$ localizing the sensor and recovered using motion capture allow us to construct the 4×4 transform matrix converting between the world frame and the camera frame in homogeneous coordinates as:

$$\mathcal{T} = \begin{bmatrix} & \mathcal{R} & & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

$\mathcal{T}$ is easily inverted and can then be used to reconstruct each measured scene point in world coordinates via

$$\mathbf{P}_W \equiv \mathcal{T}^{-1}\mathbf{P}_C \tag{5.4}$$

The maximum Z value of the measured scene points over a given position on the floor can then be recorded to build 2.5D height

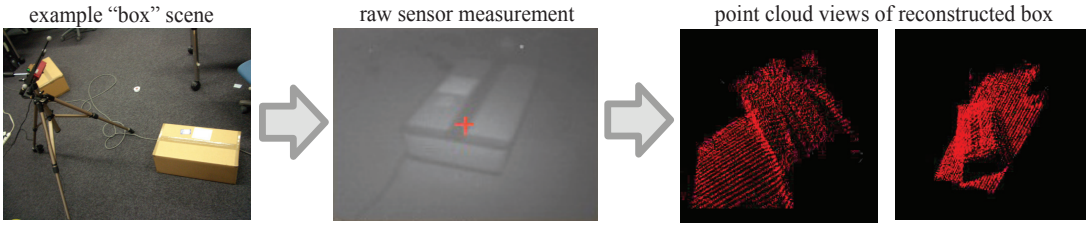


Figure 5.3: The time-of-flight range sensor viewing an obstacle box placed on the floor *(left)*. Raw measurements recorded by the sensor *(center)*. Two views of the 3D point cloud reconstruction of the box *(right)*.

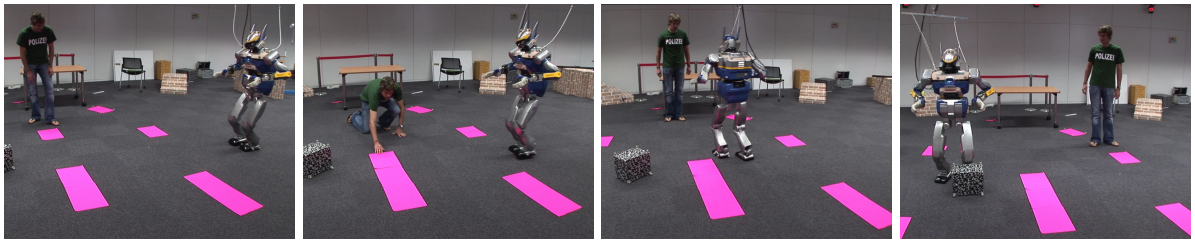
maps of the environment. In practice, we perform some median-filter smoothing to eliminate residual measurement noise. Figure 5.3 shows an example reconstruction.

5.3 RESULTS: HRP-2 NAVIGATION AUTONOMY & INTELLIGENT JOYSTICK

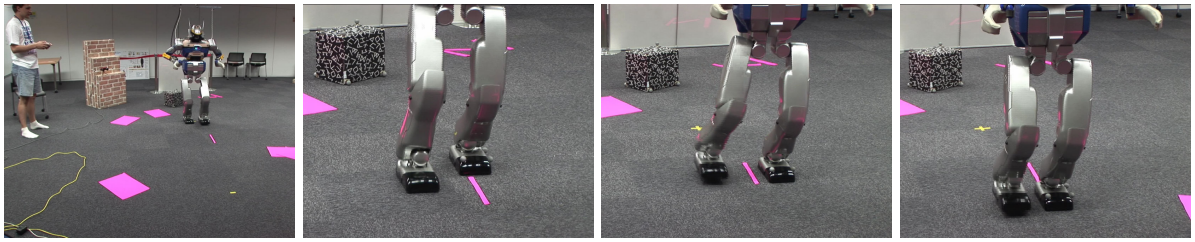
When implemented online on the humanoid HRP-2 and combined with an agile, re-planning capable footstep planner similar to the one outlined for ASIMO, the reconstruction approaches outlined here enable the robot to once again navigate autonomously in the presence of unpredictably moving obstacles, using on-body sensors localized globally.

Figure 5.4a shows HRP-2 walking towards a goal location while the surrounding obstacles are being moved by a human. Our homography-based reconstruction approach using the robot's head-mounted cameras is used. The experimenter is asked to intentionally block off the currently computed path to the goal. The planner quickly adjusts for the sudden occlusion, computing a sequence of footsteps that allows the robot to reach the goal. By having an experimenter additionally reposition the goal location, HRP-2 can traverse the room continuously without stepping on obstacles.

We also combine our 2D environment reconstruction approach with an adaptive joystick-based steering method that allows for intuitive user control of the robot while still selecting safe alternate foot placements in the presence of obstacles [12]. Figure 5.4b shows HRP-2 under such joystick control. Given a simple 'forward' directional command, a detected obstacle is avoided by a



(a)



(b)

Figure 5.4: Examples of online environment reconstruction used during biped locomotion: HRP-2 navigating autonomously towards a goal location (box) with unpredictably moving obstacles *(a)*. Avoiding a thin, elongated obstacle during a joystick controlled walking sequence *(b)*.

sidestepping motion automatically computed using information derived online from the environment map.

Appropriately fusing measurements from the on-body sensors and the global localization sensor can be challenging. In particular, tight temporal synchronization of the sensor data with the positioning information is crucial for accurate reconstruction. There is also the question of representation of data fused from multiple sensors. We currently maintain separate occupancy-grid and height-map representations for use during planning, but hybrid methods may be more compact and informative. Still, the ability to accurately localize on-body sensors gained by equipping the robot's surroundings with a global localization sensor (not unreasonable given the typical indoor operating environments of current humanoids) enables *accurate*, *responsive* and globally *scoped* representations to be generated from local sensing data that can serve as the perception basis for autonomous navigation, allowing the robot to more freely navigate environments of increasing complexity.

6

A GPU-ACCELERATED REAL-TIME 3D TRACKING SYSTEM FOR HUMANOID PERCEPTION

Section 2.2 outlined the key requirements and challenges involved in building perception systems for humanoids and appropriately integrating them with planning to achieve task autonomy: *Responsiveness*, *Robustness*, *Scope & Descriptiveness*, *Accuracy*, *Safety & Reliability* and *Mutual Awareness*. Attempting to optimize for many or all of these at once is a very daunting task conceptually and computationally. The approaches outlined in Chapter 4 and Chapter 5 exploit domain, task, environment and sensor characteristics in order to build approaches to perception that exhibit some of these qualities. Here, we seek to make use of novel hardware to ameliorate some of the computational burden that meeting these goals places on our perception system in the context of enabling a humanoid to climb stairs.

6.1 MOTIVATION

Recall briefly how omnipresent and severe these challenges are to perception on a walking humanoid. The position of obstacles and other objects of interest with respect to the robot needs to be recovered with high accuracy. A localization error of more than a few (2–3) centimeters at a distance of 2–3m is usually fatal for tasks involving stepping on or over obstacles. Objects of interest need to be robustly distinguished in the sensor view from background clutter and localized reliably even during brisk walking at human-like speed. Sensor shakiness as well as swift and abrupt changes in the sensor view as the robot moves its head must be handled appropriately. The outputs of the perception process—localization information and environment representations—must be continuously generated at speeds fast enough to ensure appropriate reaction to changes in the robot's environment.

Many of the classically favored sensors and perception methods in robotics tend to not address these issues satisfactorily. Stereo, while satisfying the responsiveness requirements, often

leads to representations that exhibit too much reconstruction error to be used for challenging locomotions such as stair climbing. This approach also tends to work poorly at even moderate sensor-to-object distances. Accurate laser range finders have until very recently remained too bulky for on-body operation on humanoids. They also require active, controlled sensor movement when mounted to transform their mainly planar distance measurements into 3D representations. While laser-based reconstruction can operate in real time, it may often be unable to precisely recover geometry of objects at moderate to high distances to the sensor due to sparseness of sampling. More traditional, monocular feature-based approaches tend to function well for high level robot localization (e.g. via SLAM), but again fail at precisely recovering the full pose of objects in the robot's environment.

Model-based object tracking approaches, on the other hand, have been shown to tackle these requirements successfully, albeit chiefly in stand-alone scenarios where a fixed camera tracks moving objects or cameras are moved smoothly in a hand-held fashion. When properly implemented, operation proceeds in real-time at frame-rate. The use of an appropriate 3D model for the task at hand ensures accuracy in reconstruction objects of interest. Given an accurate object model, pose reconstruction error can then mainly be attributed to pixel noise in the camera image, which remains within acceptable bounds with current high-resolution CCDs, good quality lenses and proper calibration. However, model-based tracking at frame-rate on the CPU alone tends to easily exhaust available compute resources. This leaves little to no processing resources to deal with ensuring robust operation during fast, unpredictable sensor motion and the environmental complexity typically encountered when dealing with a humanoid robot.

To this end, we accelerate a robust model-based three-dimensional tracking system by programmable graphics hardware to operate online at frame-rate during locomotion of a humanoid robot [52]. The tracker recovers the pose of viewable objects relative to the robot. Leveraging the computational resources of the GPU for perception enables us to increase our tracker's robustness to the significant camera displacement and camera shake typically encountered during humanoid navigation. When combined with a footstep planner and a controller capable of adaptively adjusting the height of swing leg trajectories, our approach allows HRP-2 to reliably and rapidly localize, approach and climb stairs, as well

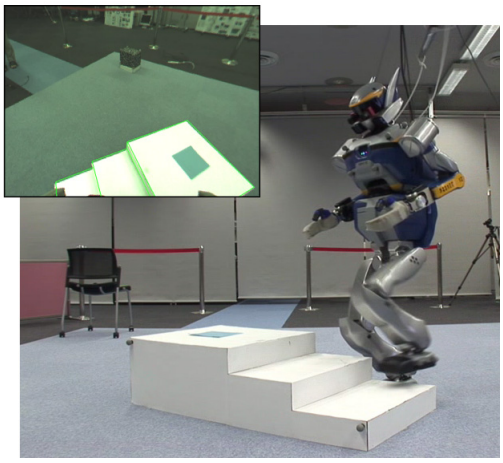


Figure 6.1: The HRP-2 humanoid autonomously climbing a set of stairs. Environment mapping and robot localization is accomplished online using our GPU-accelerated 3D tracker (tracker view inset).

as to avoid obstacles during walking. Figure 6.1 shows HRP-2 executing a climbing motion, with an inset view of the tracker in operation using the robot camera.

We believe that mandating the existence of even a simple 3D model for objects of interest is not an unreasonable requirement considering the well structured indoor operating environments currently typical for humanoids. We *robustly* fit a series of control nodes initialized from the visible model edges of a given object to edge features extracted from the video stream in *real-time*, yielding the full 6DOF pose of the object relative to the camera. The recovered pose, together with the robot kinematics, allows us to *accurately* localize the robot with respect to the object and to generate appropriately *scoped* and *descriptive* height-based maps of the robot environment. These can then be used to plan a sequence of footsteps that, when executed, allow the robot to circumnavigate obstacles and climb stairs with a *speed*, flexibility and *success rate* not achieved before. Note how no external sensor or localization system is required. Perception remains completely on-body. The ultimate result is an end-to-end solution for humanoid perception, operating quickly, accurately and robustly and suitable for use as a localization and environment reconstruction component in a wide range of autonomous humanoid task scenarios.

6.2 MODEL-BASED 3D TRACKING & POSE RECOVERY

Our fundamental approach to monocular model-based 3D tracking closely follows the method proposed by Drummond and Cipolla [16]. We manually initialize and subsequently maintain and recursively update an estimate of the 3×4 matrix of extrinsics $\mathbf{E}$, representing the $\mathrm{SE}(3)$ pose of the tracked object relative to the camera. Together with the 3×3 matrix of intrinsic parameters $\mathbf{K}$, gathered from calibration, it forms the camera projection matrix as $\mathbf{P} = \mathbf{KE}$.

To estimate the relative pose change between two consecutive frames, the object model is projected onto the image plane according to the latest estimate of the pose $\mathbf{E}_t$ and a set of regularly spaced so-called control nodes is initialized along those projected edges. These are used to match the visible projected model edges to edge features extracted from the camera image using a Canny edge detector [9]. The errors in this matching are used to find an update $\Delta\mathbf{E}$ to the extrinsic parameter matrix using robust linear regression. The updated pose of the object is finally calculated as $\mathbf{E}_{t+1} = \mathbf{E}_t \Delta\mathbf{E}$ and the procedure is repeated for the next frame.

Recall that the camera projection matrix takes a point from world coordinates to projective camera coordinates as follows:

$$(u, v, w)^\mathsf{T} = \mathbf{P}\ (x, y, z, 1)^\mathsf{T} \tag{6.1}$$

with pixel coordinates given by $x = u/w$ and $y = v/w$. To recover the rigid transform $\Delta\mathbf{E}$, we consider the six generating motions which comprise it, namely translations in the x, y and z directions and rotations about the x, y and z axes. These generating motions can be represented by the following six matrices, effectively velocity basis matrices:

$$\mathbf{G}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{G}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The pose update $\Delta\mathbf{E}$ can be constructed from these Euclidean generating motions via the exponential map [81] as

$$\Delta\mathbf{E} = \exp\left(\sum_{i=1}^{6} \mu_i \mathbf{G}_i\right) \tag{6.2}$$

The motion vector $\boldsymbol{\mu}$ thus parameterizes $\Delta\mathbf{E}$ in terms of the six generating motions $\mathbf{G}_1$ to $\mathbf{G}_6$. It is $\boldsymbol{\mu}$ that we recover using robust linear regression.

If a particular control node ξ with homogeneous world coordinates $\mathbf{p}^{\xi} = (x, y, z, 1)$ is subjected to the i^{th} generating motion, the resulting motion in projective image coordinates is given by:

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \mathbf{P}\mathbf{G}_i\,\mathbf{p}^{\xi} \tag{6.3}$$

which is given in pixel coordinates by:

$$\mathbf{L}_i^{\xi} = \begin{pmatrix} \tilde{u}' \\ \tilde{v}' \end{pmatrix} = \begin{pmatrix} \frac{u'}{w} - \frac{uw'}{w^2} \\ \frac{v'}{w} - \frac{vw'}{w^2} \end{pmatrix} \tag{6.4}$$

We can project this motion onto the model edge normal $\hat{\mathbf{n}}$ at the control node as:

$$f_i^{\xi} = \mathbf{L}_i^{\xi} \cdot \hat{\mathbf{n}} \tag{6.5}$$

Suppose a 1D edge search along the model edge normal determines that control node ξ is at a distance d^{ξ} from the closest image edge. The individual f_i can conceptually be combined into a Jacobian matrix that determines the effect of each of the i generating motions on the control-node-to-image-edge distance at control node ξ. Considering the set of control nodes in its entirety, we can calculate the motion vector $\boldsymbol{\mu}$ by fitting d^{ξ} to f_i^{ξ} for each control node via the usual least-squares approach:

$$g_i = \sum_{\xi} d^{\xi} f_i^{\xi};\; C_{ij} = \sum_{\xi} f_i^{\xi} f_j^{\xi};\; \mu_i = \sum_{j} C_{ij}^{-1} g_j$$

We can now use the recovered motion vector $\boldsymbol{\mu}$ to reconstruct the inter-frame pose update via the exponential map.

Note that this is essentially the same as finding the solution $\boldsymbol{\mu}$ to the equation

$$\mathbf{J}\boldsymbol{\mu} = \mathbf{d} \tag{6.6}$$

where $\mathbf{d}$ is the vector of collective node-to-image-edge distances for all control points, and the tracking Jacobian $\mathbf{J}$ is defined by:

$$\mathbf{J}_{\xi i} = \frac{\partial d_{\xi}}{\partial \mu_i} \tag{6.7}$$

Algorithm 1: IRLS(D, F)
Perform iteratively reweighted least squares model-edge to image-edge fitting

Data: $D = \{d^{\xi_1}, \ldots, d^{\xi_k}\}$: *measured* edge-normal distances to closest image edge for all k control nodes
$F = \{f^{\xi_1}_{\{1\ldots6\}}, \ldots, f^{\xi_k}_{\{1\ldots6\}}\}$: edge-normal motion resulting from each of the 6 generators for all k control nodes
Result: μ: motion vector parameterizing pose update

μ = OrdinaryLeastSquares(D, F);
while $iteration < max\_iterations$ **do**
  for *each control node* ξ **do**
    $residual^{\xi} = d^{\xi} - \sum_{i=1}^{6} f_i^{\xi} \mu_i$;
  end
  MAD = MedianAbsoluteDeviation(residuals);
  for *each control node* ξ **do**
    $c = \frac{abs(residual^{\xi}) \times 0.6745}{4.685 \times MAD}$;
    $weight^{\xi} = \begin{cases} (1-c^2)^2 & \text{if } c < 1 \\ 0 & \text{otherwise} \end{cases}$
  end
  μ = WeightedLeastSquares(D, F, *weights*);
end

with ξ indexing over control nodes and i over the six generating motions. Note that $\mathbf{J}_{\xi i}$ is thus simply equivalent to f_i^{ξ} defined above. The least-squares solution can thus also be formulated using the pseudo-inverse as

$$\boldsymbol{\mu} = \mathbf{J}^{\dagger}\mathbf{d} \equiv (\mathbf{J}^{T}\mathbf{J})^{-1}\mathbf{J}^{T}\mathbf{d} \tag{6.8}$$

6.3 ROBUST FITTING

The rapidly changing and often cluttered views of the world as observed by the robot camera naturally give rise to outliers (e.g. a control node falsely 'snapping' to a strong background edge in absence of strong model edges), to which regular least squares fitting is quite susceptible. While a variety of robust estimation methods have been proposed to ameliorate this problem, including RANSAC and M-estimators [71], we employ Iteratively Reweighted Least Squares (IRLS) for robust fitting. The residuals of an initial iteration of ordinary least squares with equally weighted measurements are adjusted in accordance with the resulting motion vector estimate $\boldsymbol{\mu}_1$. The measurements are then re-weighted by applying a bisquare function to the residuals from the previous iteration, giving lower weights to points that

do not fit well, essentially in conformance with Tukey's robust estimator:

$$\rho_{tuk}(x) = \begin{cases} \frac{c^2}{6}\left[1-(1-(\frac{x}{c})^2)^3\right] & \text{if } |x| \leqslant c \\ \frac{c^2}{6} & \text{if } |x| > c \end{cases} \tag{6.9}$$

Algorithm 1 details the IRLS process. We iterate the re-weighted fitting a number of times until the residuals change only marginally arriving at the latest estimate for the motion vector, $\boldsymbol{\mu}_n$. This can be used to calculate an *intermediate* estimate for the pose update matrix, $\Delta\tilde{\mathbf{E}}$. Still considering the *same* two frames, we re-project the control nodes using the pose $\mathbf{E}_t\Delta\tilde{\mathbf{E}}$ and re-start the entire inter-frame tracking process. Iterating the pose update process in its entirety several times for a single pair of frames ensures the most accurate and robust model-to-edge fitting. Note that it is only due to the fact that all of the image processing and edge search takes place on the GPU, as will be shown, that we even have the CPU resources to execute the model fitting process in this manner. Using the GPU as a computation platform has thus effectively enabled much of the robustness our approach exhibits.

A 'confidence' measure assessing approximately how well the tracker is operating at a particular point in time can be straightforwardly derived from the weights used during IRLS. Conceptually, we would like the majority of control nodes to get matched to an image edge ('edge presence') and to simultaneously be inliers during the fitting stage, i.e. to have high weights with the overall deviation of residuals from the median being low ('quality of fit'). These quantities can be combined into a single confidence-like measure as:

$$\text{confidence} = \frac{0.9\,MAD}{n \times \max(weights^{\xi})} \sum_{\xi \in matched} weight^{\xi} \in [0,1] \tag{6.10}$$

where n denotes the number of *all* control nodes and *MAD* denotes the median absolute deviation of fitted residuals. Note how the average is taken over all control nodes, while only nodes that are actually matched to an image edge will contribute to the fitting and thus have a non-zero weight (i.e. ξ only iterates over matched control points). A confidence value of 1 would thus reflect the ideal tracking situation where *every* control node is matched to an edge generated from the *actual object* — and thus exhibits a weight of 1. The confidence value is reduced if some control nodes either don't get matched to image edges (e.g. due to occlusion or absence of a strong object edge) or get

matched to background edges as outliers and are thus assigned lower weights. We maintain a separate confidence value for each object tracked and use it together with an empirically determined threshold value to detect when we have lost track of the object (resulting in a drastic drop in the confidence value) or when the object is being fitted to background clutter (resulting in a less drastic, but detectable drop in confidence).

6.4 ROBUST MULTIPLE HYPOTHESIS TRACKING

While the IRLS fitting process outlined above is able to handle even large numbers of outliers, strong background contours present in the scene may give rise to not only numerous, but correlated outliers. To further increase the robustness of the tracker against incorrect snapping to strong misleading background contours, we consider multiple edge hypotheses for each control node during the fitting stage. That is, for each control node ξ, we search along the model edge normal and record distance measurements d_k^ξ to the k closest image edges found, rather than merely attributing a single measurement to each control node. Figure 6.2 illustrates. During the initial fitting step, we take *all* hypotheses extracted for *all* control nodes into account with equal weight. During the subsequent IRLS fitting process, we then compute weights for each hypothesis at every point again by using Tukey's biweight. Now, at each control node, only the residual corresponding to the hypothesis with the *lowest* weight contributes to the fit at each iteration. Effectively, we choose the hypothesis selected by the multivariate version of Tukey's biweight as:

$$\rho^*_{tuk}(x_1, x_2, \ldots, x_n) = \min_i \rho_{tuk}(x_i) \tag{6.11}$$

The more hypotheses are available for each control node, the more likely it is that we will be able to pick the 'correct' match subsequently during fitting. However, the consideration of multiple hypotheses also increases the average per-node computation time spent searching for model-edge to image-edge matches. While this effect is rather minor in our GPU implementation, we restrict our per-node hypotheses to $k = 4$, for ease of GPU-CPU data readback.

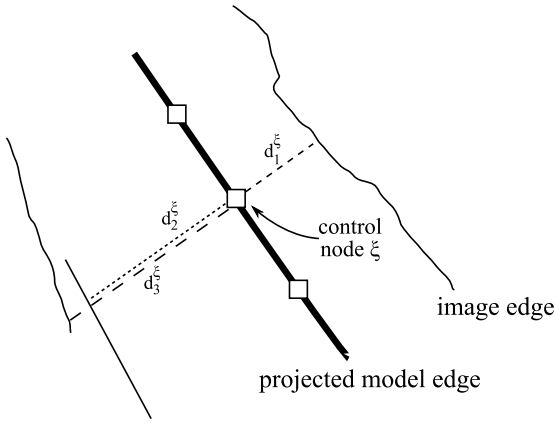
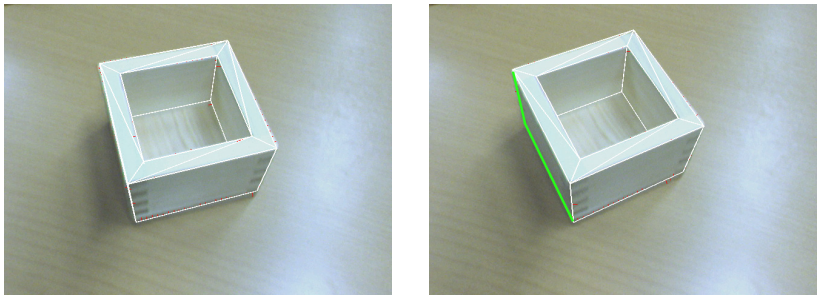


Figure 6.2: For each control node ξ, we maintain multiple hypotheses d_k^ξ indicating which image edge the node should match to.

6.5 FILTERING OBJECT POSE

We combine measurements (i.e. the recovered pose from each inter-frame tracking step, $\mathbf{E}_t$) using a Discrete Extended Kalman Filter [84], in a manner similar to the approach used by Klein [35]. The internal state the filter maintains is determined both by integrating successive, potentially noisy, measurements (during the correction step) and by a model of the system dynamics used to predict the system's behavior (during the prediction step). These steps alternate throughout the lifetime of the filter. The main benefits provided by the filter are twofold: for one, it aids in appropriately integrating successive pose measurements to eliminate jitter and provide 'smoother' pose recovery over time. Perhaps more importantly, the filter's embedded dynamics model provides an estimate of how the object being tracked is expected to move in the near future. We use the filter state after each prediction step to start our inter-frame pose recovery process. This implies that the pose used to project the model and initialize edge search now also incorporates predictive information from the filter's internal dynamic model. We have found this to be especially helpful in situations where the tracked object is rotating and an aspect change introduces new model edges that need to be matched. In this case, the new edges tend to appear at the same location as already existing edges and, without using predictive information from the filter, have a tendency to 'snap' to the already matched edges. The added dynamics serve to 'push' the newly appearing edges to the right location for a successful match. Figure 6.3 illustrates.



(a) (b)

Figure 6.3: View of a sake box being tracked *(a)*. New edges (marked green) appear very close to edges already in view *(b)*. Absent an object motion model as provided by the Kalman filter, the new model edges will tend to snap to image edges already in view.

6.5.1 *State Description & Prediction Step*

The filter maintains a 12 dimensional internal state, representing both pose (6DOF) and velocity (6DOF) of the object being tracked. These are stored as an $SE(3)$ pose matrix, denoted $\mathbf{E}_{K|t}$ here, and velocity 6 vector $\mathbf{v}_{K|t}$ holding translational and rotational velocities, as proposed by Klein [35]. We can thus summarize the state of the filter at a particular time t by:

$$x_t = \{\mathbf{E}_{K|t}, \mathbf{v}_{K|t}\} \tag{6.12}$$

A constant-velocity dynamic model is used during the prediction step to yield the prior estimate of future state after a period of elapsed time δ_t as:

$$x^-_{t+\delta_t} = \{\mathbf{E}_{K|t} \exp(\mathbf{v}_{K|t}\delta_t), \mathbf{v}_{K|t}\} \tag{6.13}$$

where $\exp()$ denotes the exponential map. Being a constant-velocity model, we model acceleration simply as Gaussian process noise (σ^2_p). The a-priori estimate error covariance is projected forward in accordance with the constant-velocity model as:

$$\mathbf{P}^-_{t+\delta t} = \begin{bmatrix} \mathbf{I}_6 & \delta_t \mathbf{I}_6 \\ 0 & \mathbf{I}_6 \end{bmatrix} \mathbf{P}_t \begin{bmatrix} \mathbf{I}_6 & 0 \\ \delta_t \mathbf{I}_6 & \mathbf{I}_6 \end{bmatrix} + \sigma^2_p \begin{bmatrix} 0 & 0 \\ 0 & \mathbf{I}_6 \end{bmatrix} \tag{6.14}$$

6.5.2 *Update Step*

Suppose the tracker has just computed a new pose measurement $\mathbf{E}_M$. We integrate this measurement into the filter during the

correction step. To do so, the relative pose change between the filter's prior state and the just recovered pose (in essence the measurement innovation) is computed as:

$$\mathbf{L} = (\mathbf{E}^{-}_{K|t+1})^{-1}\mathbf{E}_{M} \tag{6.15}$$

and then used to find the posterior pose via:

$$\mathbf{E}_{K|t+1} = \mathbf{E}^{-}_{K|t+1}\exp(\mathbf{K}\ln(\mathbf{L}))) \tag{6.16}$$

$\mathbf{K}$ here is the usual Kalman gain matrix and computed as:

$$\mathbf{K} = \mathbf{P}^{-}\mathbf{H}^{T}(\mathbf{H}\mathbf{P}^{-}\mathbf{H}^{T} + \mathbf{R})^{-1} \tag{6.17}$$

Here, the matrix $\mathbf{H}$ relates the state to the measurement and is set to $\mathbf{H} = [\mathbf{I}_6\, 0]$ (only pose is measured). Finally, the posterior covariance is given by:

$$\mathbf{P} = (\mathbf{I}_6 - \mathbf{K}\mathbf{H})\mathbf{P}^{-} \tag{6.18}$$

This now leaves only the measurement covariance matrix $\mathbf{R}$, the initial state covariance matrix $\mathbf{P}_0$ and the process noise variance σ_p^2 undetermined.

6.5.3 *System Identification*

$\mathbf{P}_0$ is initialized to be the identity matrix and is subsequently updated as illustrated above in Equation 6.14 and Equation 6.18. The process noise variance σ_p^2 essentially quantifies the acceleration, in the camera frame, not explicitly modeled by our constant velocity model. It must be tuned manually and depends on the scale of the object motions we expect to see during operation. As such, we set it differently for close-up tracking situations (e.g. an object on a table) or larger scale tracking scenarios (e.g. obstacles to be avoided by the robot while walking).

To establish the correct measurement covariance matrix $\mathbf{R}$, reflecting the estimated measurement error, first assume that control-node-to-image-edge distance measurements $\mathbf{d}$ extracted from edge search are corrupted by Gaussian noise of 1 pixel variance as $\mathbf{d} = \hat{\mathbf{d}} + \delta$. Assume that, as a result, the motion vector $\boldsymbol{\mu}$ recovered from tracking approximates the true motion vector $\hat{\boldsymbol{\mu}}$, corrupted by noise as $\boldsymbol{\mu} = \hat{\boldsymbol{\mu}} + \boldsymbol{\epsilon}$. The measurement covariance matrix $\mathbf{R}$ can now be derived via expectation as:

$$\mathbf{R} = E[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^{T}] = E[(\boldsymbol{\mu} - \hat{\boldsymbol{\mu}})(\boldsymbol{\mu} - \hat{\boldsymbol{\mu}})^{T}] \tag{6.19}$$

which, recalling Equation 6.6, reduces further to:

$$\begin{aligned}\mathbf{R} &= \mathrm{E}[(\mathbf{J}^{\dagger}\mathbf{d} - \mathbf{J}^{\dagger}\hat{\mathbf{d}})(\mathbf{J}^{\dagger}\mathbf{d} - \mathbf{J}^{\dagger}\hat{\mathbf{d}})^{\mathsf{T}}] \\ &= \mathrm{E}[(\mathbf{J}^{\dagger}\boldsymbol{\delta})(\mathbf{J}^{\dagger}\boldsymbol{\delta})^{\mathsf{T}}] \\ &= \mathbf{J}^{\dagger}\mathrm{E}[\boldsymbol{\delta}\boldsymbol{\delta}^{\mathsf{T}}]\mathbf{J}^{\dagger\mathsf{T}} \\ &= (\mathbf{J}^{\mathsf{T}}\mathbf{J})^{-1}\end{aligned} \tag{6.20}$$

It is thus possible to establish the measurement covariance matrix by way of the tracking Jacobian established previously in Equation 6.7.

6.5.4 *Filtering for Camera Motion*

The above approach assumes the typical tracking scenario involving a stationary camera and a moving object. That is, the process noise variance σ_p^2 is set up to reflect object centric motions and the measurement covariance matrix $\mathbf{R}$ was calculated via the tracking Jacobian $\mathbf{J}$, which defines how each of the six generating motions $\mathbf{G}_1$ to $\mathbf{G}_6$ executed *at the object* affect the edge-to-edge distances. However, these assumptions no longer hold when we are dealing with a moving camera, such as is the case on a walking humanoid.

To deal with a moving camera, we modify the extended Kalman filter to maintain the *inverse* of the object pose as part of its internal state. The filter state is thus initially set to the inverse of the first pose measurement the tracker recovers. Subsequently, suppose the tracker has just computed a new pose measurement $\mathbf{E}_M$, reflecting the object's pose relative to the camera. We will now update the $\mathrm{SE}(3)$ pose maintained by the filter, $\mathbf{E}_{K|t}$, using the inverse of the pose measurement. The measurement innovation (cf. Equation 6.15) is now computed as

$$\mathbf{L} = (\mathbf{E}^{-}_{K|t+1})^{-1}\mathbf{E}_M^{-1} \tag{6.21}$$

thus effectively tracking the camera pose relative to the object. Recall that the filter state after the prediction step, $\mathbf{E}_{K|t+1}$, is used to initialize the inter-frame pose recovery performed by the tracker. However, since the internal state now reflects pose of the camera relative to the object, we must now use its inverse to initialize the tracking process.

To correctly filter pose in a camera-centric manner, the process noise covariance σ_p^2, should also be adjusted. Recall that σ_p^2

essentially quantifies the amount of acceleration, not modeled by our constant-velocity assumption, we expect to encounter during tracking. Especially when the moving camera is mounted on a walking humanoid, σ_p^2 will be somewhat different than when tracking a moving object from a stationary camera, and should be tuned appropriately.

Most importantly, the measurement covariance matrix $\mathbf{R}$ is computed differently when dealing with a moving camera. $\mathbf{R} = (\mathbf{J}^T\mathbf{J})^{-1}$ is now calculated from a modified tracking Jacobian that reflects how each of the generating motions executed *at the camera* affect the edge-to-edge distances. Recall that (cf. Equations 6.7, 6.4, 6.5, 6.3):

$$\mathbf{J}_{\xi i} = \frac{\partial d_\xi}{\partial \mu_i} = f_i^\xi = \mathbf{L}_i^\xi \cdot \hat{\mathbf{n}} \tag{6.22}$$

$$\text{with } \mathbf{L}_i^\xi = \begin{pmatrix} \tilde{u}' \\ \tilde{v}' \end{pmatrix} = \begin{pmatrix} \frac{u'}{w} - \frac{uw'}{w^2} \\ \frac{v'}{w} - \frac{vw'}{w^2} \end{pmatrix}$$

To reflect that the generating motions $\mathbf{G}_1$ to $\mathbf{G}_6$ are now executed at the camera, we now define the projective image coordinates via

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \mathbf{K}\mathbf{G}_i\,\mathbf{E}\mathbf{p}^\xi \tag{6.23}$$

Notice how, compared to Equation 6.3, the coordinates of control point $\mathbf{p}^\xi$ are now first transformed into the camera frame by the pose matrix $\mathbf{E}$. Subsequently, the generating motion $\mathbf{G}_i$ is applied before ultimately projecting the point onto the image plane via the intrinsic matrix $\mathbf{K}$. This allows us to establish an appropriate measurement covariance matrix for scenarios with a moving camera rather than a moving object.

6.6 LEVERAGING THE GPU

All aspects of our 3D tracking system involving image or geometry processing are GPU-accelerated. We have implemented our method using a cascade of fragment programs, shown in Figure 6.4, written using NVIDIA's Cg language [51] and operating on image data stored locally as textures in GPU memory. The latest incoming video frame serves as input to the filter cascade, which ultimately outputs a texture containing the (potentially

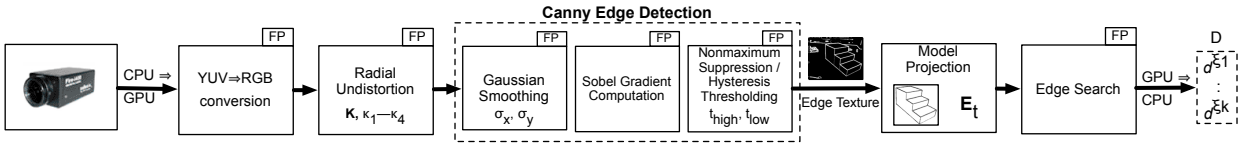


Figure 6.4: GPU fragment program cascade defining flow of image processing, model projection and edge search.

multiple) edge-normal distances d^{ξ_i} for all control nodes on the visible projected model edges of the object. Exploiting the GPU allows us to process the full resolution 1024×768 YUV 4:2:2 video stream as supplied by a Point Grey Flea2 camera at 30 frames per second. Note that merely two CPU-GPU data transfers are performed: one to upload the latest camera image to the GPU initially and one to read back the results of the edge search process.

Color space conversion from the camera-native YUV 4:1:1 to RGB, a costly process on the CPU, proves very amenable to GPU implementation. The same holds true for the per-pixel process of the RGB in accordance with the camera intrinsic parameters f_x, f_y, o_x, o_y (focal lengths and optical center) and the four radial distortion coefficients $\kappa_1, \ldots, \kappa_4$ and to counteract lens distortion. Each of the standard components of the Canny edge detector (gaussian smoothing, gradient computation, non-maximum suppression, hysteresis thresholding) executes sequentially as a fragment program on the GPU, with the defining Canny parameters such as $\sigma_{gauss}, thresh_{high}$ and $thresh_{low}$ staying fully adjustable during tracker operation. All image processing results in a single binary texture indicating presence or absence of an edge at each pixel.

To fit edges rendered using the current pose estimate to image edges, we assume the existence of a simple 3D model of the object of interest in terms of its main salient lines and its faces. Such models are easily generated using CAD or image-based modeling and photogrammetry software. In particular, we use Google SketchUp and its Photo Match feature to quickly generate geometrically accurate textured models from a few photographs. Figure 6.5 gives an example of the model generation process, taking about 15 minutes in this case. We subsequently use the standard OpenInventor / VRML97 format to represent our object models.

Since we are mainly interested in performing edge-matching at boundary edges of the object, we perform a step of mesh sim-

Figure 6.5: Image-based modeling of objects of interest. One of four input images used for modeling the tea carton (*left*). Three sample views of the resulting 3D model (*center*). Hidden-line wireframe version of the model used for tracking (*right*).

plification during a pre-processing stage. Only edges separating two faces that exhibit different face normals are retained. This eliminates a large percentage of the interior edges of each face, which often result from tessellation and do not correspond to edges exhibited by the real object's geometry. Furthermore, we eliminate edges joining two faces if the angle between their corresponding face normals is less than a certain cutoff value (around 20 degrees). Again, such edges often result from approximation of round surfaces (such as cylindrical table legs, for instance) by polygons and often do not correspond to 'true' edges on the object. Finally, for complex objects, we also eliminate edges if their edge length in object space is more than 2.5 standard deviations smaller than the mean edge length of the model. This effectively eliminates the shortest 16% of edges, which often do not contribute any control nodes useful for fitting and are often tessellation artefacts.

We then render our model onto the image plane using the current pose estimate $\mathbf{E}_t$ and the camera intrinsics $\mathbf{K}$, performing depth-buffered hidden line removal efficiently and resulting in a binary texture containing only the visible edges of the model. We initialize a number of control nodes along the model edges, spaced evenly in image coordinates at a user-selectable interval. Unlike choosing control node spacing in object coordinates, this ensures that control nodes do not cluster too tightly on edges that appear short on the rendered model due to perspective projection. Control node information is provided to the edge search fragment program as a single four channel RGBA texture, with the red channel indicating presence/absence, the green and blue channels encoding the x and y components of model

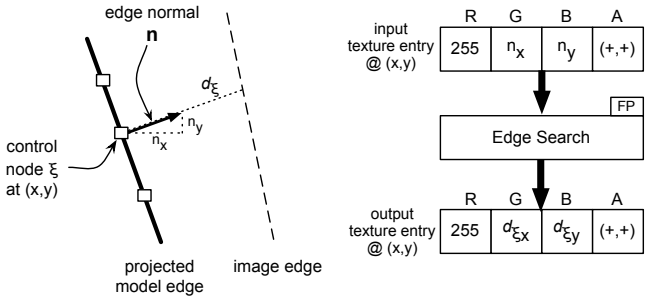
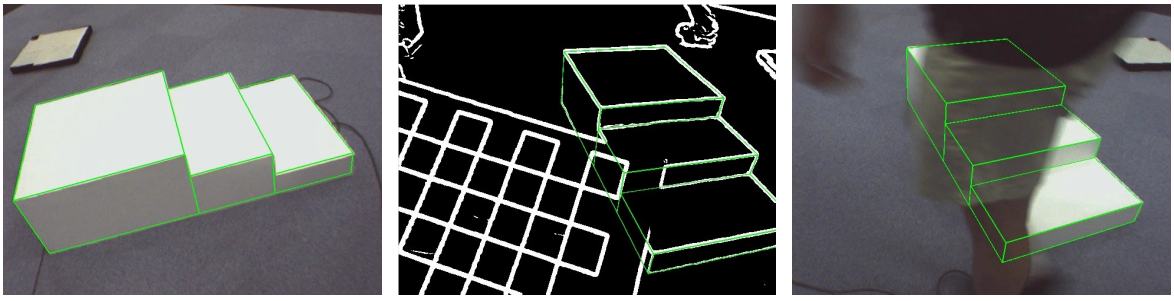


Figure 6.6: Representation of input/output data to and from the fragment program matching control nodes to image edges.

edge normal at the control node, and their signs being integer-encoded in the alpha channel. Figure 6.6 illustrates. The edge search fragment program then steps along the true model edge normal (albeit quantized to pixel coordinates) trying to detect the $k = 4$ closest image edges to the control node in either the positive or negative normal direction. Search is performed up to a certain cutoff distance. If no image edges are found within that distance, the control node is ignored and does not contribute to the solution fit. If search is successful, the results are stored in an output texture a manner similar to the input data format described above, and shown in Figure 6.6. In case multiple image edges were found for a control node at position (x, y), the search results are stored in the output texture at (x, y) and its top, left and right pixel neighbors. While both the search distance and the number of control points search is performed on directly affect the running time of the edge search process, we have not been able to saturate our GPU-implementation even with many hundreds of control nodes and edge search distances spanning more than 50 pixels in either direction of the normal. Figure 6.7a shows a typical view with the tracked model superimposed in green, Figure 6.7b shows a view of the model superimposed on the extracted image edges during object occlusion with a checkerboard. Figure 6.7c shows a view of the tracker during severe occlusion by an experimenter walking in front of the camera.

The GPU's abundant compute resources have also enabled us to handle the tracking of multiple objects present in the scene in a straightforward manner. A separate pose estimate is maintained throughout the tracking process for each object of interest. We then initialize control nodes along each object's model edges and pass a *single* texture containing control node information for *all* objects to the edge search fragment program, with search subsequently proceeding as before. During the fitting stage, search



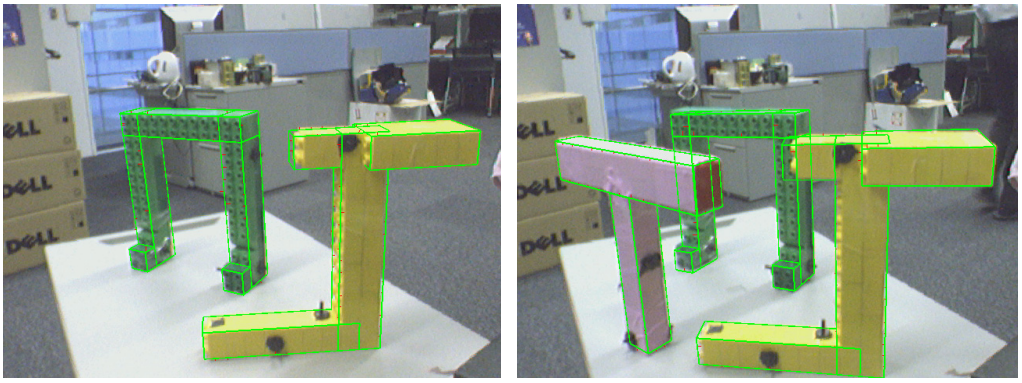
(a) (b) (c)

Figure 6.7: Stairs being tracked in real time *(a)*. View of model-edge to image-edge fitting during occlusion *(b)*. Tracker operation under severe occlusion *(c)*. Views gathered from a handheld camera for illustrative purposes.

results are then associated with their respective objects and fitting proceeds separately for each one. Figure 6.8a gives an example. Since the information pertaining to each control node is encoded in a single pixel value (see Figure 6.6), it is possible for objects that overlap in the scene view to generate control nodes at identical pixel locations. This implies that search results can then potentially be associated with the wrong object during the fitting stage. In practice we have found that, unless object overlap is severe, such wrong associations will simply be treated as outliers by the robust fitting process and will not cause tracking to fail, as Figure 6.8b illustrates. Note also that tracking multiple objects affects only the number of control nodes search is performed on and the number of robust fitting iterations on the CPU side, but has no effect on the number of times we iterate between fitting and search while processing a single pair of frames, currently the most expensive part of the tracking process due to the performance penalty of readbacks from the GPU to main memory. The cost incurred by tracking multiple objects is thus relatively minor.

6.7 RESULTS: TRACKER PERFORMANCE

We assessed the performance of our GPU-accelerated tracker in a number of standalone (off-body) experiments to quantify its operating characteristics, such as speed of component subtasks, accuracy with respect to jitter due to image noise as well as overall accuracy in recovering the object's pose when compared to ground truth.



(a) (b)

Figure 6.8: Multiple objects being tracked against a cluttered background *(a)*. Successful tracking of three objects in the presence of inter-object occlusions *(b)*.

6.7.1 *Speed*

Table 6.1 gives the runtime, broken down by subtask, for a typical tracking iteration resulting in an updated pose. Note how the overall runtime of 32ms allows operation at 30Hz, even when an unnecessarily high number of control points is used (337 in this example). We can essentially process camera images as quickly as our camera can supply them, ensuring responsive on-body operation. Also note that the edge search process is by far the most time-consuming subtask, primarily due to the fact that control points must be initialized from the model edges and that the two main CPU↔GPU memory transfers (uploading the control point locations / normals and retrieving the edge search results) form part of this subtask.

6.7.2 *Stability*

In order to assess the amount of 'jitter' present in the pose measurements recovered by the tracker, a stationary object was tracked from a fixed camera mounted on a tripod. The camera was placed 15cm away from the object. In this scenario, any changes in the recovered pose over time should be attributable to noise in the incoming camera images that cause the model-edge to image-edge distances d^{ξ} to vary from frame to frame. A view of this stationary tracking scenario is shown in Figure 6.9.

Subtask	Runtime
1. Retrieve & undistort camera image	5.8ms
2. Apply Canny edge detector	0.7ms
3. Project model & perform edge search	12ms × 2
4. Compute pose update (fitting)	0.7ms × 2
5. Update Kalman filter	0.1ms
Total	**32ms**

Table 6.1: GPU-accelerated tracker runtime for a single pose recovery iteration, broken down by subtask while tracking the stairs shown in Figure 6.10a. Steps 3 and for are iterated twice here due to iterated fitting & model reprojection. Number of control points used: 337.

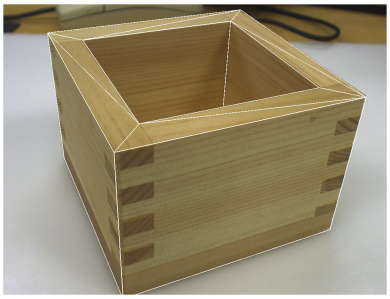
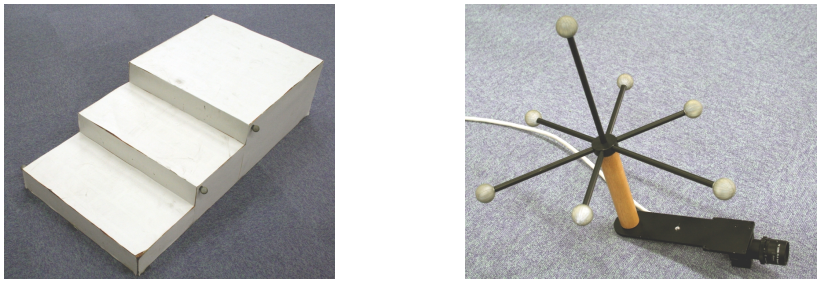


Figure 6.9: View of a sake cup tracked from a distance of 15cm while assessing tracker jitter.



(a) (b)

Figure 6.10: Stairs and camera tracked by motion capture. Stairs equipped with retro-reflective markers *(a)*. Firewire camera mounted on a marker-equipped frame *(b)*.

We measured the standard deviation of the recovered tracker pose, decomposed into translation and rotation, over a run of 110 frames. The apparent Root Mean Square (R.M.S.) of the recovered translation was 0.0313mm. The apparent R.M.S. of the recovered rotation (calculated via quaternion averaging) was 0.0235 degrees. Both of these numbers are in line with results reported previously for CPU-based approached to edge tracking [16], indicating that the use of the GPU likely does not introduce any additional numerical inaccuracy.

6.7.3 *Accuracy of Recovered Pose*

To assess the absolute accuracy of recovered pose measurements when compared to ground truth, the tracker was made to track a set of stairs in an area outfitted with a Motion Analysis 12-camera motion capture system [56]. Both the stairs and the camera were equipped with retro-reflective markers and tracked by motion capture, as seen in Figure 6.10. Motion capture then recovers the 6DOF pose of the camera and the stairs as a transforms ${}^{m}_{c}\mathrm{T}$ and ${}^{m}_{s}\mathrm{T}$, respectively. From this we construct the relative transform between the stairs and the camera as ${}^{s}_{c}\mathrm{T} = ({}^{m}_{s}\mathrm{T})^{-1}\ {}^{m}_{c}\mathrm{T}$, effectively positioning the camera with respect to the stairs. This transform then serves as our 'ground truth' positioning, which we subsequently compare to the *inverse* of the transform recovered by the tracker, $\mathbf{E}^{-1}$. Figure 6.11 shows a visual comparison of these two recovered transforms, positioning the camera with respect to the stairs.

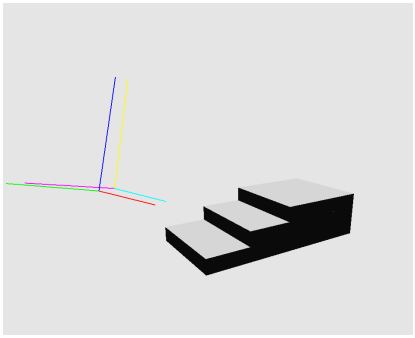


Figure 6.11: Comparative view of the two transforms localizing the camera with respect to the stairs. The transform recovered by motion capture (${}^{s}_{c}\mathrm{T}$) is shown by a frame with red, green and blue axes. The transform recovered via the tracker, $\mathbf{E}^{-1}$, is drawn in cyan, magenta and yellow.

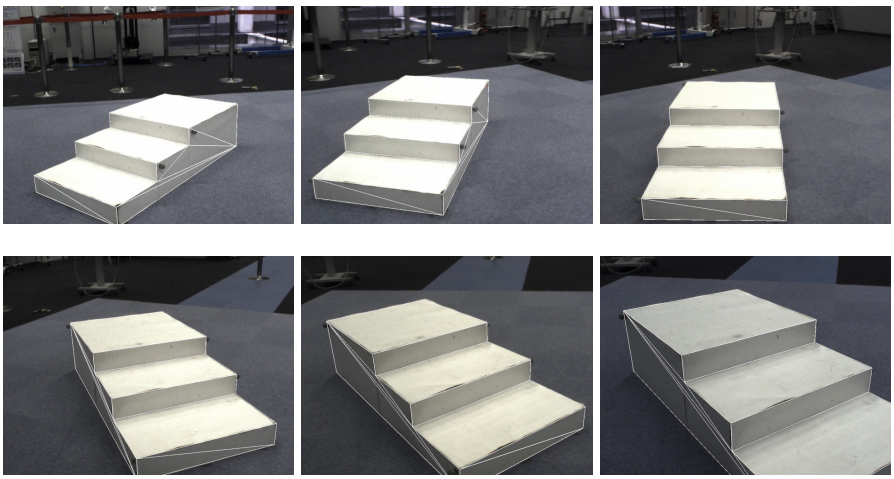


Figure 6.12: Snapshots of tracker operation during the motion sequence used to evaluate accuracy (left-right, top-bottom).

An experimenter then proceeded to move the camera for 23 seconds in a semi-circular fashion around the stairs, varying also the distance to the stairs and the height of the camera off the floor. Figure 6.12 shows selected snapshots of the tracker view from the run.

Throughout the resulting run, pose information from motion capture and the tracker was gathered at evenly spaced, identical points in time, yielding 801 pose samples. Figure 6.13 shows the resulting pose sequence, decomposed into translation along x, y and z, as well as rotation described by roll, pitch and yaw Euler angles. Figure 6.14 plots the 3D trajectory the camera was

moved along by the experimenter, as recovered by the tracker and motion capture.

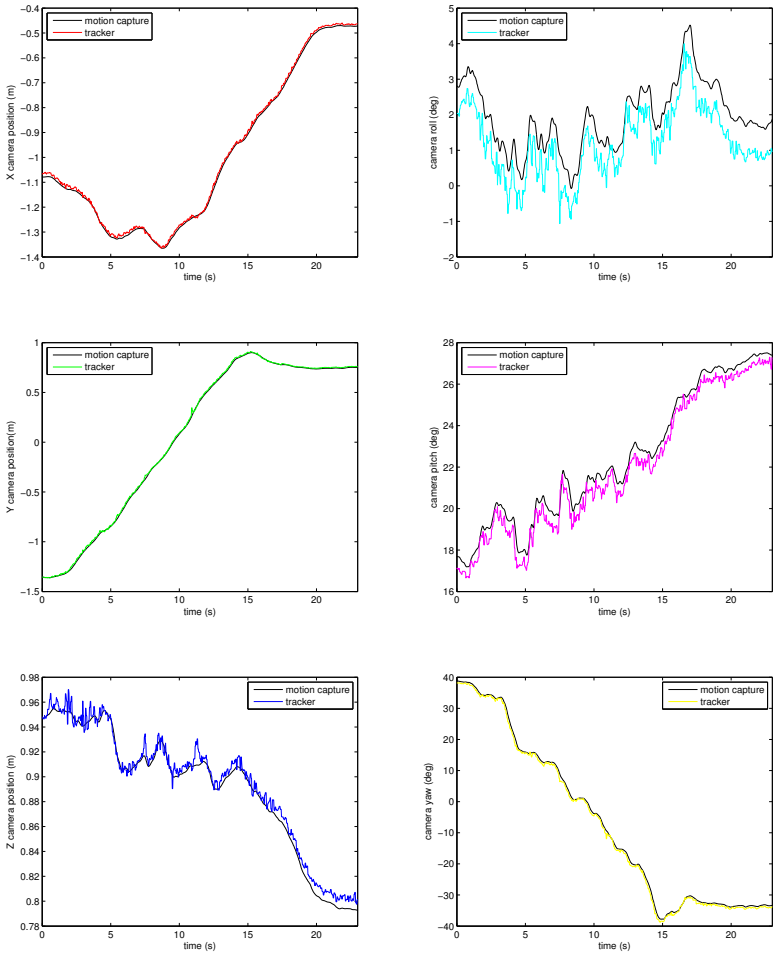


Figure 6.13: Plots of the recovered transforms from the tracker and motion capture, decomposed into translation *(left column)* and rotation *(right column)*.

The smoother shape of the motion capture curves can be attributed to the faster update rate of the system, as well as a built-in filtering process. The average translation error throughout the trajectory along x, y and z was 0.82cm, 0.94cm and 0.52cm, respectively, yielding an overall average translation error of 1.51cm. Average rotation errors for roll, pitch and yaw (calculated through quaternions) were 0.47 degrees, 0.53 degrees

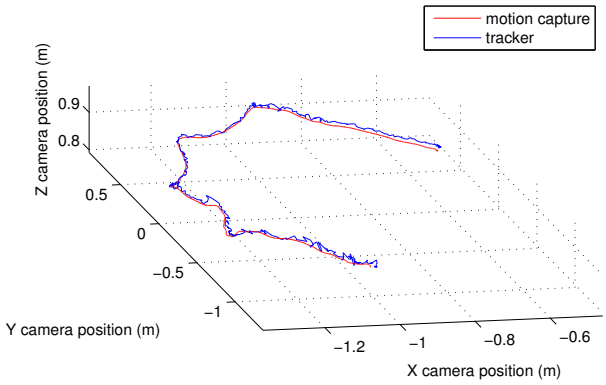


Figure 6.14: 3D plot of the trajectory (with position computed relative to the stairs) the camera was moved along by the experimenter, as recovered by the tracker and motion capture.

and 0.64 degrees, for an overall rotation error of 1.03 degrees. It is worth noting that our use of motion capture coordinates as 'ground truth' implies that some additional, though minor, error may be introduced by calibration errors of the motion capture system itself or errors in the templates used to recover the pose of the camera and stairs from the raw locations of the markers they are outfitted with.

Most importantly, these numbers satisfy the error bounds of the walking controller used on HRP-2 together with the planner's safety margins, where an overall localization error exceeding 2cm becomes problematic.

6.7.4 *Multi-scale Tracking*

The edge- and model-based nature of the tracking systems is implicitly able to handle tracking at multiple scales with the same edge-based representation of the object to be tracked. In other words, the distance of the camera from the object to be tracked is insignificant as long as a sufficient number of model edges are present in the view and can be successfully matched to their corresponding image edges. This implies that tracking can proceed both in situations where the entire model is visible in the image (Figure 6.15a) or when the object is only partially visible (Figure 6.15b,Figure 6.15c). Note that since off-screen model

(a) (b) (c)

Figure 6.15: A door tracked at multiple scales. A single model is used to track the door when it is visible in its entirety *(a)* and when only parts *(b)* or even only a small subset of the model *(c)* are visible in the camera view.

edges do not get rendered (and therefore do not generate control nodes) and because control node spacing is determined in image coordinates, there is no significant performance difference between whole-model and close-up tracking situations. In general, tracking performance is determined less by the number of edges visible in the image than by their relative orientations. A close up tracking situation with edges oriented exclusively in the vertical direction, for instance, is likely to lead to poor performance, since such edges merely 'lock' the object's horizontal direction, but do not yield sufficient information regarding the objects vertical placement. Scenarios where a mix of model edge orientations is encountered are thus preferable.

6.8 AUTOMATIC TRACKER INITIALIZATION

An oft-quoted shortcoming of recursive trackers, which recover relative pose change between subsequent frames rather than treating each frame independently, is the need for a separate, a-priori initialization step to determine the object's absolute pose once, effectively to establish $\mathbf{E}_0$. Many previous approaches tend to employ a stage of manual pose initialization, during which the user roughly establishes the object's translation and rotation from the camera by way of a graphical user interface. Such initialization exhibits high accuracy but is undesirable for robotic applications.

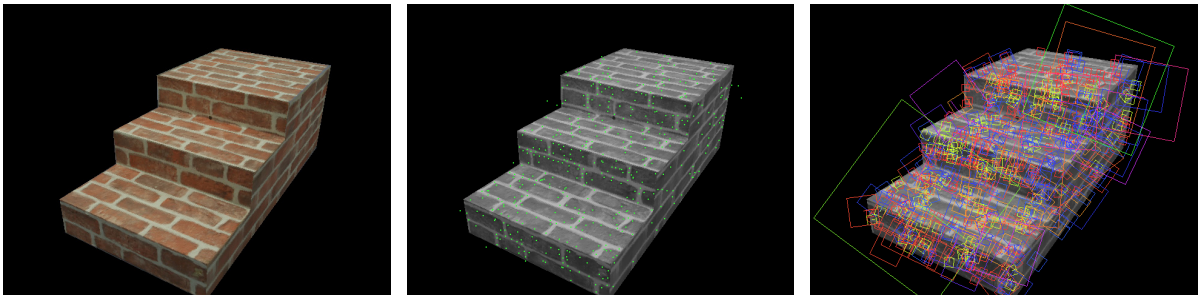
Automatic tracker initialization is essentially a variant of the recognition problem in vision. As such, most automatic ap-

proaches to pose initialization conceptually rely on a library of model images or features extracted from such model images. These are constructed off-line, show the object of interest from a variety of viewpoints and are stored together with associate pose information [6, 46]. The initial scene view is then subjected to the same feature extraction process and the resulting features are then matched to the model views. Depending on the granularity of the model view/pose library, the pose yielded in this manner may then be close enough to the actual pose of the object to successfully initialize the tracker. Alternatively, having established a 2D-2D matching between the input view and a model view and knowing also the 3D coordinates of the matched point in the model view, it is possible to more precisely establish the pose of the object.

We generate representative model views for each object of interest by rendering a textured model of the object from a variety of viewpoints. In the most general setting, this is accomplished by uniformly sampling the view-sphere of camera positions in spherical coordinates (θ, ϕ, r). The generality of this approach is offset by the fact it often generates views that bear little resemblance to the object's likely configuration in the real world (e.g. a cup viewed from the bottom). It is thus often preferable to restrict the configurations model images are generated from or even extend the model generation step by a stage of sample pose generation, where a library of 'likely' poses is generated once via a GUI. We generate between 8 and 16 model views of the object for automatic initialization. The views are rendered using an OpenGL camera with intrinsic parameters set to match those of the actual camera used to supply video to the tracker, $\mathbf{K}$.

6.8.1 *GPU-based SIFT Computation*

We use features based on David Lowe's Scale Invariant Feature Transform [48] to perform matching between incoming camera images and our database of model images, though other fast approaches exist [44]. We chose SIFT features principally due to their relative robustness to lighting and viewpoint changes and their suitability for computation on the GPU. We extract SIFT features from each of the model images. SIFT features extracted from each incoming camera image are then subsequently matched to these. All SIFT feature extraction takes place with subpixel accuracy on the graphics processing unit via a series of Cg shaders, using a modified version of SiftGPU [85]. Keypoint



(a) (b) (c)

Figure 6.16: Example SIFT features extracted from one model image used during tracker initialization. Input model image *(a)*. Location of SIFT keypoints *(b)*. Visualization of SIFT keypoints *(c)*. A total of 508 features were extracted.

Subtask	Runtime
Build DoG pyramid	14ms
Keypoint detection	6ms
Generate list of features	22ms
Compute feature orientations	8ms
Download keypoints from GPU	1ms
Download feature descriptors from GPU	21ms
Misc. initialization / cleanup	9ms
Total	**81ms**

Table 6.2: SiftGPU runtimes broken down by subtask for the model image shown in Figure 6.16a. 508 SIFT features were extracted.

locations and feature descriptors are subsequently downloaded back into main memory from the GPU. Computing SIFT features on the GPU yields very significant speedups compared to even highly optimized CPU implementations. For the sample model image of a set of textured stairs shown in Figure 6.16a, a standard CPU-based implementation of SIFT takes 6.047 seconds to extract features, while extraction on the GPU takes about 0.67 seconds for the initial iteration. Once the startup overhead of setting up GPU memory, generating the Gaussian filters etc. is removed on subsequent iterations, feature extraction completes in roughly 80ms (iterating at around 12Hz) on an NVIDIA 8800GTX GPU, broken down as shown in Table 6.2. We use the canonical 128-element SIFT descriptors with standard parameters to the algorithm as in [48].

6.8.2 *Keypoint Matching*

The matching process attempts to associate features extracted from the input image with features from the model images, seeking to find the model image that most closely matches the input image. It is from this image that we will then recover the initial object pose. In order to perform pose recovery, we require at least 4 non-coplanar points reliably matched between our input camera image and at least one of the model images.

Conceptually we would like to find, for each SIFT keypoint in the input image, the keypoint with minimum Euclidean distance in SIFT descriptor space. However, as outlined by Lowe, simply choosing the closest keypoint does not perform well, since such matching does not assess how discriminative that feature is and can often match keypoints generated from background clutter in the input image to keypoints in the model images. A solution is to keep a match between input keypoint and its nearest model keypoint neighbor only if the *second* nearest neighbor is significantly more distant that the first (where significance is expressed fractionally as $\frac{\mathtt{dist}_1}{\mathtt{dist}_2} < k$, k often around 0.6).

Such matching requires the ability to efficiently and accurately establish the nearest neighbors of points in high-dimensional spaces, which reduces to exhaustive search for finding exact nearest neighbors. We use Best-Bin-First Search [3] to efficiently find approximate nearest neighbors that are accurate with high probability. Best-Bin-First search is a variant on KD-tree search that remains efficient by searching bins in order from their closest distance from the query location and by aborting search after examining a a certain fixed number of nearest-neighbor candidates.

The matches resulting from Best-Bin-First matching can still exhibit a considerable number of outliers due to image noise and background clutter. We thus filter these further to remove outliers by iterating RANSAC on the matches. RANSAC fits a 2D homography to the matches and returns a set of inlier matches from the Best-Bin-First matches that have a high probability of being correct. We allow a maximum pixel error of 5 pixels and iterate RANSAC enough to obtain a 99% confidence. Note that, in reality, the SIFT features used here lie on non-planar surfaces in both the model views and the input view. A 2D homography is thus not theoretically sufficient to describe the transform between model view and input view, but we have found this approximation to work quite well in filtering SIFT matches. Fig-

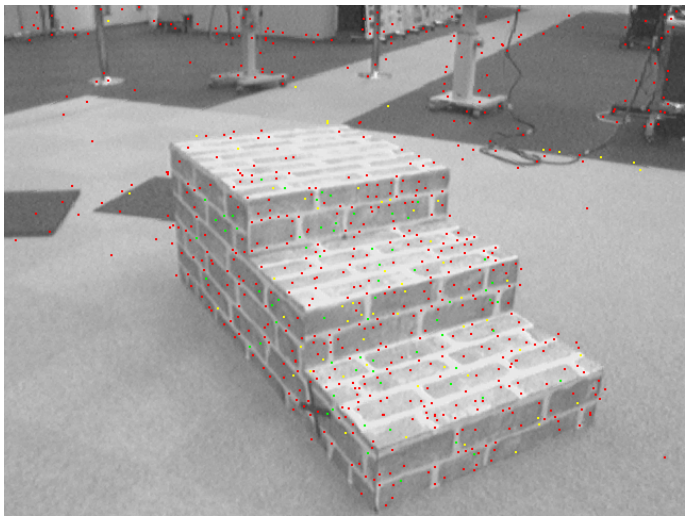


Figure 6.17: SIFT keypoints extracted from camera image while attempting to establish the pose of the stairs in the view. Found keypoints (*red*). Subset of found keypoints matched by Best-Bin-First search (*yellow*) to keypoints from a model image, Subset of matched keypoints determined to be inliers by RANSAC (*green*). Around 40 keypoints are ultimately kept.

ure 6.17 shows SIFT keypoints extracted from an input frame (red), matched through Best-Bin-First search to a model image (yellow) and determined as inliers by RANSAC (green). Note how we are able to successfully match keypoints even though texture on the object is extremely repetitive (in fact, the stairs were simply covered with several repetitions of the same brick pattern printed on an adhesive). Also note that the images used to build our model may be taken under substantially different lighting conditions with a different camera than what we encounter in the input views during operation. Slight to moderate inaccuracies in texture alignment on the model and even model shape inaccuracies are also not a problem (though an accurately shaped model is important for the recursive tracking stage). Despite this, the method outlined here finds reliable matches that can be used for pose recovery.

6.8.3 *Pose Recovery*

The keypoint matching process leaves us with a set of reliable 2D-2D correspondences between the input image and a particular model image (we pick the image exhibiting the most inlier

matches among the database of model images). To establish the full 6DOF pose of the object, however, a set of 2D-2D matches is not enough (at most, a homography between both point sets could be recovered).

Fortunately, given the 3D model of the object of interest, we are able to straightforwardly recover the 3D coordinates of the SIFT keypoints in the model images. We use OpenGL's `gluUnproject()` function to very efficiently determine the 3D object coordinates $(\mathtt{obj}_x, \mathtt{obj}_y, \mathtt{obj}_z)$ of a 2D point with image coordinates (win_x, win_y) using the graphics hardware. To do so, we read back the depth buffer of the rendered model image. For each 2D point, this provides an additional depth coordinate win_z in the range $[0, 1]$. Given additionally the width and height of the model image, a projection matrix $\mathbf{P}$, initialized in accordance with the real camera's intrinsic parameters, and a modelview matrix $\mathbf{M}$ reflecting the object's pose as rendered in the model image, `gluUnproject()` then recovers the 3D point coordinates as

$$\begin{pmatrix} \mathtt{obj}_x \\ \mathtt{obj}_y \\ \mathtt{obj}_z \\ W \end{pmatrix} = (\mathbf{PM})^{-1} \begin{pmatrix} \frac{2win_x}{\mathtt{width}} - 1 \\ \frac{2win_y}{\mathtt{height}} - 1 \\ 2win_z - 1 \\ 1 \end{pmatrix} \tag{6.24}$$

where W is unused.

With our set of 2D-3D correspondences (input-image coordinates to object coordinates) thus established, we now use DeMenthon & Davis' POSIT algorithm [15] to recover the initial object pose $\hat{\mathbf{E}}$. POSIT recovers the pose as a rotation matrix and a translation vector by approximating the perspective projection that generated the 2D points from their 3D coordinates with a scaled orthographic projection (POS: Pose from Orthography and Scaling). This estimate is then used to iteratively (IT) compute better scaled orthographic projections of the keypoints and again applying POS to those, etc. POSIT converges quickly to an accurate pose estimate within a few tens of iterations and is fairly robust to measurement errors or even the odd outlier still present in our 2D-3D correspondences. POSIT implicitly uses a single-focal-length pinhole model, while our calibrated camera may exhibit separate focal lengths for the x and y axes. This is easily ameliorated by adjusting the pixel coordinates of the 2D keypoints input to POSIT a-priori to reflect a single focal length for both dimensions.

The pose recovered by POSIT can then be used to initialize our recursive edge-based tracking process.

6.8.4 *Integration*

To summarize, the automatic tracker initialization process conceptually encompasses the following steps:

A. Generate a set of model views with associated pose from the given 3D object model

B. Extract SIFT keypoints from each model image using SiftGPU, build Best-Bin-First KD-trees for each model image

C. Extract SIFT keypoints from the input camera image

D. Match input keypoints to each model image's keypoints using Best-Bin-First search

E. Filter resulting matching using RANSAC to eliminate outliers

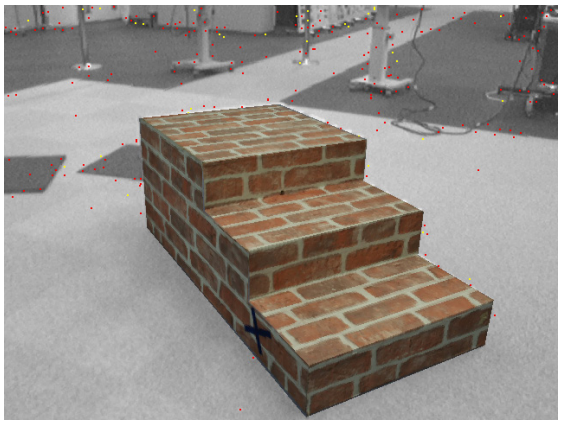
F. Pick the matching yielding the highest number of inliers

G. Recover 3D coordinates of model keypoints

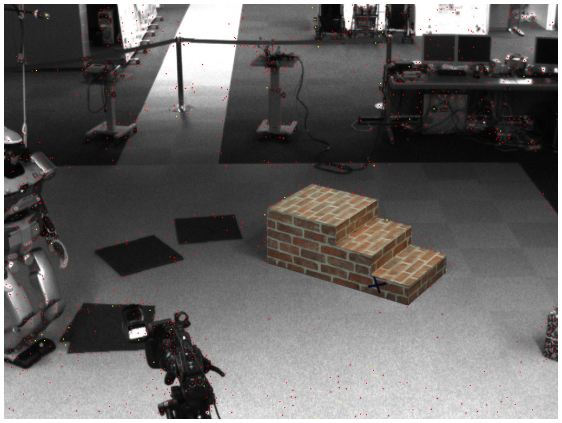
H. Run POSIT on resulting 2D-3D matching to recover pose $\hat{\mathbf{E}}$

The entire process iterates at about 0.5 to 2Hz for a scenario with 8 model images, around 400 SIFT features per model image and around 800 SIFT features in the input image on an Intel Core2Duo CPU at 2.13Ghz with 2Gb of memory and an NVIDIA 8800GTX GPU.

In practice, even absent any scene or camera motion, the SIFT features extracted from two temporally adjacent frames $(i, i+1)$ will not match precisely due to image noise, etc. This may result in the pose $\hat{\mathbf{E}}_i$ recovered for frame i differing slightly from that recovered for frame $i+1$. There is also the possibility that the non-coplanarity requirement for POSIT is violated for a particular frame and only SIFT points lying on the same object face are found. To ameliorate this, we let the initialization process run for several frames, combining the recovered poses by averaging over time, essentially 'smoothing' the pose. This is accomplished through the exponential and logarithm maps defined on the Lie group $SE(3)$ which all poses recovered belong to. If $\hat{\mathbf{E}}_i$ denotes



(a)



(b)

Figure 6.18: Textured model of stairs rendered onto camera view in accordance with recovered pose. Pose recovered from SIFT keypoints shown in Figure 6.17 (*a*). Stairs located in a larger scene (*b*).

the recovered pose at frame i, then the smoothed cumulative pose $\bar{\mathbf{E}}$ is defined by:

$$\begin{aligned} \bar{\mathbf{E}}_1 &= \hat{\mathbf{E}}_1 \\ \bar{\mathbf{E}}_{i+1} &= \exp(\tfrac{1}{2}\ln(\bar{\mathbf{E}}_i) + \tfrac{1}{2}\ln(\hat{\mathbf{E}}_{i+1})) \end{aligned} \tag{6.25}$$

Figure 6.18 illustrates the result of pose recovery for a set of stairs. The recovered pose was used to render the model of the stairs onto the camera image in the appropriate place.

(a)

(b)

Figure 6.19: Views of example augmented reality tracking sequences. 3D cow model (5,804 polygons) positioned on the top step of a set of stairs, localized by the tracking system *(a)*. More complex 3D robot model (46,055 polygons) positioned similarly *(b)*.

6.9 RESULTS: AUGMENTED REALITY

Since our tracker recovers the full 6DOF pose of objects of interest and given that the camera intrinsics have been recovered via an a-priori calibration step, it is very straightforward to superimpose rendered objects onto the video stream from the camera positioned at a fixed or dynamically changing 3D pose to the tracked objects, simply by chaining the (optionally temporally varying) desired transform of the virtual object relative to the physical object with the pose recovered by the tracker.

The resulting synthesized view of the world thus registers virtual elements to the real world in 3D in real-time, resulting in an 'augmented reality' view of the world [2]. The tracking system's low jitter and high accuracy, as detailed in Sections 6.7.2 and 6.7.3 ensure good static and dynamic registration of the overlaid imagery with the camera images. Provided at least one object of interest remains trackable, the GPU-accelerated tracker thus serves as a viable alternative to marker-based augmented reality approaches [34] or the use of custom sensors for determining camera pose. Figure 6.19 shows example augmented reality views, with two virtual objects rendered so as to appear perched atop a set of stairs.

6.10 MAPPING, LOCALIZATION & PLANNING

To perform navigation planning for our robot we need a representation of the environment, with the robot localized within that representation. We can establish a map coordinate system in which the object being tracked is assumed to remain at a fixed position and orientation in map coordinates, given by a transform ${}^{m}_{o}T$. Once we have recovered the pose of the object in camera coordinates (given, say, by a transform ${}^{c}_{o}T$), it is easy to position the camera relative to the object. The pose of the camera in map coordinates is then straightforwardly recovered as ${}^{m}_{c}T = {}^{m}_{o}T\,{}^{o}_{c}T = {}^{m}_{o}T\,({}^{c}_{o}T)^{-1}$, essentially positioning the camera in a consistent coordinate system relative to the object of interest.

For footstep planning, knowing the accurate location of the robot foot at any point during execution is crucial. Fortunately, the robot kinematics can supply us with another transform, ${}^{c}_{f}T$, locating the robot foot relative to the camera at any instant in time. This transform, when chained with ${}^{m}_{c}T$, allows us to recover the position and orientation of the foot in map coordinates.

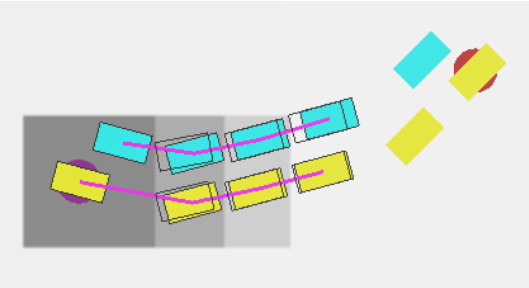


Figure 6.20: Height Map generated for stair climbing with robot and goal position as well as planned footstep path superimposed.

Note that for accurate recovery of the foot location through kinematics, it is paramount that the camera center be precisely positioned with respect to the origin of the head joints (yaw and pitch). For HRP-2, an error in camera pitch of just 1 degree will result in the foot position being off by around 5cm, greatly exceeding the error bounds acceptable for stair climbing. To ensure accurate camera positioning, we recover the relative pose of the foot (outfitted with markers) to a reference object using motion capture during a one-time 'calibration' stage. Subsequently, the same relative transform is recovered using the tracker and robot kinematics. We then adjust the physical camera positioning on the robot head until the error between the two relative transforms is eliminated.

Given the fixed position and orientation of the tracked objects in map coordinates, it is straightforward to generate a height map describing the robot environment. We render our object model using an orthographic projection and depth buffering at the appropriate pose ${}^{m}_{o}T$ and viewed from a camera positioned above and looking straight down onto the scene. A height map can then be generated simply by reading back the depth buffer after rendering and scaling the resultant depths using the values for zNear and zFar used for the orthographic projection. Figure 6.20 shows a sample height map generated for our stair climbing experiments. Due to its object-centric nature, the height map need only be constructed once. Together with the continuously updated robot foot location, this map is used to find a safe path through the environment.

In case the relative configuration of tracked objects changes throughout the walking sequence (as is the case with moving obstacles, for instance), an updated map is generated after each iteration of the pose recovery process by a similar top-down

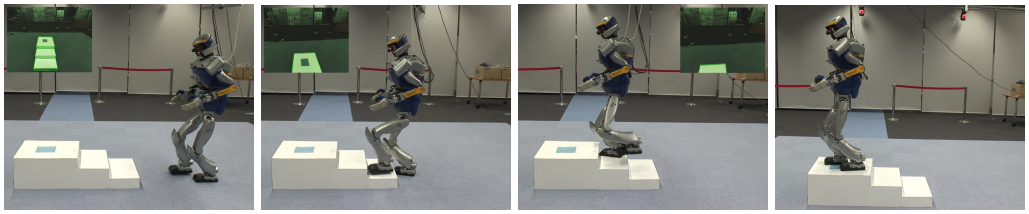
rendering step. We use a map datatype that, much like the case with the simple 2D floor maps used with ASIMO (see Chapter 4), can be parameterized by time to reflect the state of the world at a particular point in the future based on information currently available. In this case, the linear and angular velocities extracted by the tracker using the extended Kalman filter are used to extrapolate the 6DOF poses of all objects of interest into the future before the actual height map is generated by rendering. Such maps could then be used to predict obstacle configurations and could be used for planning navigation sequences in dynamic environments exhibiting moving 3D obstacles to be avoided during walking.

The footstep planner uses the recovered height-maps together with an appropriate action set describing the three-dimensional stepping motions the humanoid can execute to plan autonomous locomotion given a start and goal state. The planner computes knot points for a cubic spline swing leg trajectory which will move the foot smoothly from one foothold to the next while avoiding any obstacles between them. In addition, timings are also calculated for each step so as to slow the walking down when making long steps or when stepping up or down. These adjustments ensure executable trajectories. The path, together with the swing leg trajectories and step timings, is then used by the walking and balance controller to generate a dynamically stable walking motion which can safely take the robot from its current state to the goal location in the environment.

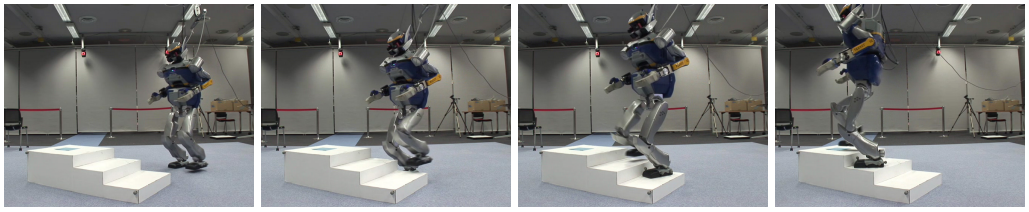
6.11 RESULTS: STAIRCLIMBING FOR HRP-2

Our initial robot experiments combine the GPU-accelerated 3D tracking system, footstep planner and walking and balance controller operating on-line on an HRP-2 humanoid robot. The tracker processes live video supplied by a robot head-mounted camera to an off-board computer, tracking a set of white stairs in the environment at 30fps, which the robot climbs or avoids in our experiments.

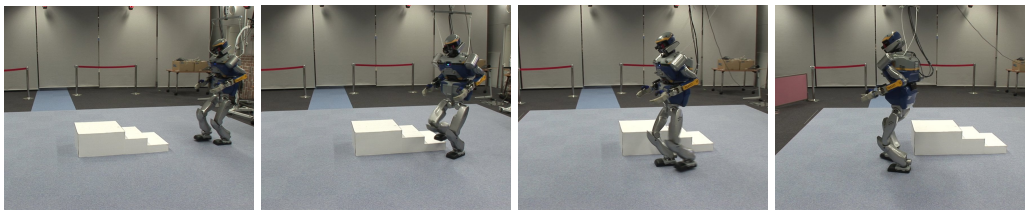
We carry out 15 stair climbing experiments with the robot starting from a wide variety of distances from and orientations relative to the stairs, during 13 of which HRP-2 successfully reaches the top of the stairs. The average length of a successful climbing sequence from the point the robot starts moving is under 8 seconds. Figures 6.21a and 6.21b show HRP-2 successfully approaching



(a)



(b)



(c)

Figure 6.21: Examples of GPU-accelerated tracking used for mapping and localization during humanoid locomotion: HRP-2 autonomously climbing *(a)*, *(b)* and avoiding *(c)* a set of stairs. Insets in *(a)* show tracker view during execution. Stairs are no longer visible from the top step in the rightmost image in sequence *(a)*.

and climbing our set of stairs. Figure 6.21c shows HRP-2 navigating around the same set of stairs. Note that, unlike in several previous approaches to stair climbing, neither planner nor controller have any predefined knowledge about the stair geometry or location, but instead operate only on the map and localization data supplied by the tracker. Also note that our tracker continues to recover the stair pose despite significant camera motion occurring during the walking sequence and even when the stairs have nearly vanished from the camera view, as in the third frame of Figure 6.21a.

Figure 6.22 plots the tracker-reconstructed (x, y, z) camera position in map coordinates during the initial part of a straight-on stair climbing sequence, until the stairs fall outside of the robot's view. Note the increasing y coordinate as the robot approaches

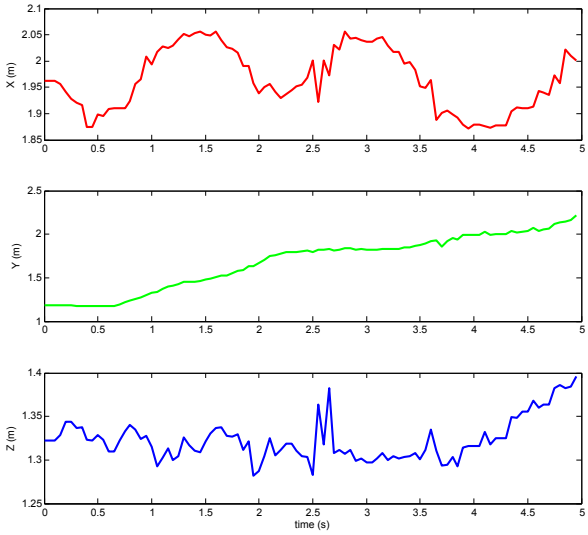


Figure 6.22: Plots of the x, y and z coordinates of the camera in map coordinates during a walking sequence of HRP-2 approaching and starting to climb a set of stairs.

the stairs directly in front of it in map coordinates and the increasing z coordinate toward the end when the robot begins to climb the stairs. The oscillations in x reflect the lateral swaying of the robot as it changes stance foot during walking. The overall lack of smoothness in the curves hints at the shakiness inherent in perception on a rigid, moving humanoid platform. Our tracker handles both of these issues well. The spike in x and z at around 2.5s shows an instance where the object pose is briefly distorted (but later recovered) due to a particularly harsh head motion resulting from a fast sidestepping movement executed by the robot which caused a large amount of motion blur in the camera image.

We believe that increased *robustness*, *accuracy* and *responsiveness* in tracking afforded by leveraging the GPU during perception is crucial in enabling HRP-2 to successfully accomplish complex locomotion tasks such as the stair climbing and obstacle avoidance tasks outlined here with a speed, reliability and flexibility not achieved before. In the following chapters, we will show how to exploit those benefits to achieve successful autonomous operation for more complex navigation scenarios as well as manipulation.

7

PLANNING FOR FUTURE PERCEPTIVE CAPABILITY: MANIPULATION

The process of planning is sometimes informally defined as 'reasoning about the consequences of actions'. Traditionally, the consequence of an action is merely a change in the state of the robotic system, $q \in \mathcal{C}$. The effect of such a change in terms of the robot's *execution capability*, or the possible choice of actions the robot can take in the resulting state, is often taken into account directly in the planning stage via the provided action set describing the robot abilities (e.g. a left stepping action must necessarily matched by a right stepping action in the following state). Often lacking from the planning stage, however, is a similar consideration of the consequences of a chosen action on the robot's subsequent ability to sense the environment. We have dubbed this the *future perceptive capability* of the robot and posit that, given the interdependence of perception and planning as shown in Section 2.1, this measure should explicitly be taken into account as part of the planning process to achieve higher reliability of execution and task success rates. Note that this differs from more traditional approaches involving 'view planning' by virtue of the fact that we do not seek *good sensor observations to aid the planner* but rather *good planning strategies to aid perception* during execution. In this chapter, we use the problem of planning motions as might be executed by a humanoid manipulating autonomously in a kitchen environment as a motivating example for reasoning about perception in the planning stage [53].

7.1 PLANNING HUMANOID REACHING MOTIONS

Consider a robot performing manipulations in a free-form home environment which is not easily purpose-built or instrumented to facilitate specific robot tasks. Unlike in traditional factory settings, it is unreasonable here to assume the precise positioning of all objects of interest. Instead, sensing is used to localize objects and to guide the positioning of the robot manipulator to ensure a successful grasp. Traditionally, a coarse reaching motion that positions the end-effector close to the object is first

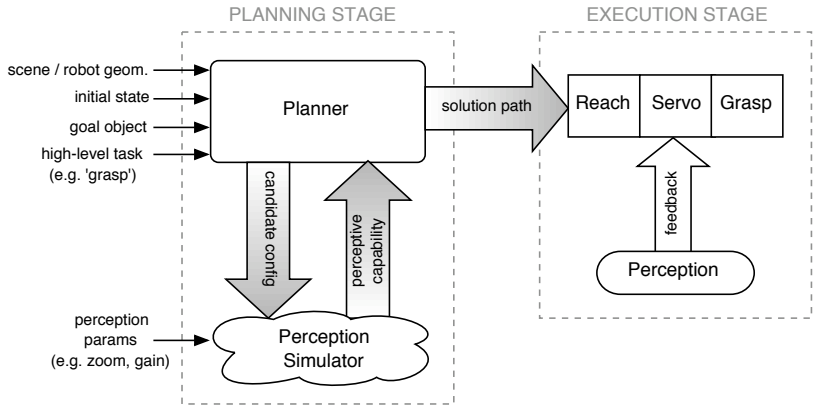


Figure 7.1: Reasoning about perception in the planning stage. A simulation of the perception process is used to compute a metric assessing perceptive capability for each configuration examined during planning.

planned, followed by a stage of fine-grained closed-loop control that incorporates visual feedback to servo the end-effector to a precise position from which the object can be grasped [83].

However, the initial planning stage is almost always oblivious to the need for subsequent visual feedback to successfully complete the grasping motion. It is not uncommon for the planner to generate solutions that preclude the object from being successfully perceived when the visual servo stage is entered, due to environmental or self-occlusions along parts or at the end of the reach motion.

We seek to generate trajectories for the manipulator in a way that takes the robot's 'sensability' of the environment throughout the reaching motion into account. To do so, we augment a sampling-based planner with a simulation of the robot's perception process that, during planning, assesses each candidate robot configuration regarding its perceptive capability, as illustrated in Figure 7.1. Note that this simulated perception process may be of arbitrary complexity, encoding as much relevant information regarding the operational characteristics of the sensor used as necessary given the task at hand. We hypothesize that the resulting plans maximize the likelihood of successful execution by ensuring that sensor information can be gathered at the crucial stages during the motion.

7.2 FORMULATION

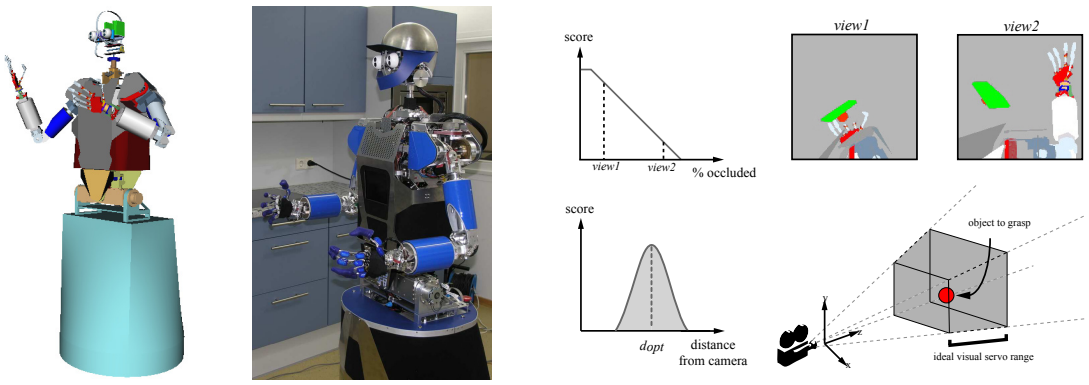
We seek to find a collision-free reaching motion given by a time-indexed path of the manipulator

$$\tau : t \in [0, T] \mapsto \tau(t) \in \mathcal{C}_{free} \tag{7.1}$$

from an initial robot configuration $\tau(0) = q_{init}$ to one of a set of valid goal states $\tau(T) = q_{goal} \in G$ from which an appropriate grasp motion can then be executed. Since a range of solutions $\tau \in \Phi$ usually exist to this under-constrained problem, a best solution τ_{best} is usually selected to optimize a specific metric $\Gamma(\tau)$ (e.g. shortest distance, manipulability, etc.) over the path as $\tau_{best} = \text{argmin}_\tau \Gamma(\tau), \tau \in \Phi$. For simple robotic systems, it may be possible to analytically ensure ensure that each state $q = \tau(t)$ satisfies some geometrically defined binary visibility constraint $q \in \mathcal{C}_{vis}$ (or perhaps only $q_{goal} \in \mathcal{C}_{vis}$ in case visual servoing only takes place in the grasp phase), ensuring a clear line-of-sight between sensor and object. For a high-DOF system in a cluttered environment and able to occlude itself, this is usually not feasible. Also, optimizing sensor placement in a complex system such as a humanoid may force the use of costly full-body motion planning rather than planning for the arm motion alone. Furthermore, how well the perception system can operate in each state may not be adequately captured by a simple line-of-sight constraint (consider e.g. the need for visible texture in a stereo system).

To this end, we introduce a cost metric $\varphi_{PC} : q \in \mathcal{C} \mapsto \varphi_{PC}(q) \in \mathfrak{R}$ that encodes the robot's perceptive capability in configuration q. The calculation of φ_{PC} may amount to anything from a simple line-of-sight constraint to ensure proper sensor placement to a full statistical simulation of the sensor's response given the state of the robot and world. $\varphi_{PC}(\tau(t))$ thus effectively encodes how well the perception system is expected to perform at a certain stage along a planned motion trajectory given the sensing requirements of the task at hand. The planner now seeks to find an optimal reach trajectory that also affords the robot maximal future perceptive capability when executed, as given by

$$\tau_{FPC} = \underset{\tau}{\text{argmax}} \sum_{t=0}^{T} \varphi_{PC}(\tau(t)), \tau \in \Phi \tag{7.2}$$



(a) (b) (c)

Figure 7.2: Rendering of the ARMAR-III humanoid robot *(a)*. The physical ARMAR-III in a kitchen environment *(b)*. The perceptive capability of a configuration is determined by the fraction of the object (here: a red sphere) visible *(c) top* as well as the location of the object relative to the optimal visual servo range *(c) bottom*.

7.3 'SENSABLE' REACHING MOTIONS FOR ARMAR-III

We combine reasoning about perceptive capability with a sampling-based planner built on the Rapidly-Exploring Random Trees (RRT) [43] framework with guiding heuristics that seek to encode our ultimate goal of increased robustness during full pick-up manipulations (reach-servo-grasp). Our robotic platform, the 43-DOF humanoid robot ARMAR-III shown in Figure 7.2a and Figure 7.2b, operates in a full kitchen environment that exhibits significant potential for collisions and occlusions.

For grasping motions executed on ARMAR, most of the task fragility and failure likelihood lies in the final servoing stage. In accordance with our hypothesis, we therefore seek to ensure proper operation of the visual servo system throughout the manipulation to maximize our chance of successful execution. We heuristically define the perceptive capability $h_{PC}(q)$ of a particular configuration to reward an unoccluded view of an object lying inside an optimal operating distance range from the sensor. Figure 7.2c illustrates. $h_{PC}(q)$ is evaluated by simulating the robot's perception system from each state examined during planning. The resulting camera image is then compared to a pre-computed appearance-based template showing the object rendered from the current configuration, but unoccluded. This allows us to quantify the fraction of the object occluded by scene geometry or the

robot itself. More complex methods employing border or texture continuity could also be used to assess occlusion. Further taking into account the optimal servo distance allows us to establish the heuristic perceptive capability $h_{PC} \in [0, 1]$ in configuration q as the weighted sum

$$h_{PC}(q) = w_u p_u(q) + w_{dist} e^{-\frac{1}{2}(d(q)-d_{opt})^2} \tag{7.3}$$

with $p_o(q)$ denoting the fraction of the object *un*-occluded.

We measure progress towards the goal via a heuristic workspace goal function $h_{WS}(q) \mapsto \mathfrak{R}$ assessing the proximity of the end-effector pose to the goal [5], resulting in the following overall cost metric the planner uses to evaluate each node and to sort a priority queue of candidate nodes:

$$l(q) = w_{PC}(1 - h_{PC}) + w_{WS}\, h_{WS} \tag{7.4}$$

During search, we furthermore define a lower threshold on the perceptive capability that each candidate node must meet lest it be discarded from the search. We specifically define a percentage of maximum allowable object occlusion that decreases linearly as the manipulator gets closer to the object, and hence the servo stage, at which point a maximum 8% occlusion is allowed.

The resulting probabilistically complete [43] planner then generates geometrically valid, collision-free paths that also guarantee a minimum perceptive capability along the trajectory.

7.4 RESULTS: ARMAR

A number of experiments carried out on ARMAR-III in simulation, with the robot's binocular vision system approximated by two perspective cameras, support our hypothesis that planning for future perceptive capability results in motion plans that more fully exploit the potential of the robotic platform to execute motions robustly using visual feedback and may increase the chance that the perception system stays operational along the trajectory.

During experiments, ARMAR's task was to grasp a spherical object floating freely in the workspace (experiment *Sphere*), covered on the top by a plate-like obstacle (experiment *CoveredSphere*) and tucked away in a cupboard (experiment *SphereInCupboard*). The experiments were designed to assess the issues and effects of self-occlusion, environment occlusion, appropriate sensor positioning and environment complexity on our approach to planning for

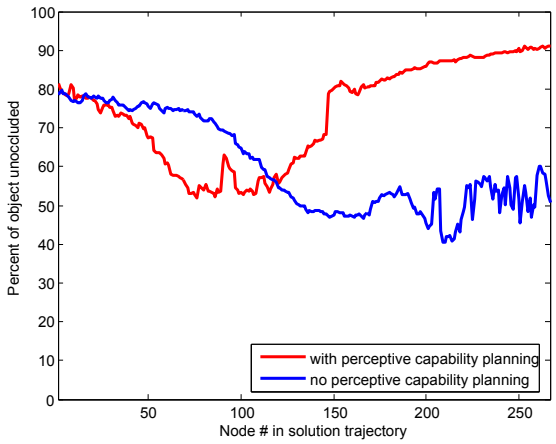


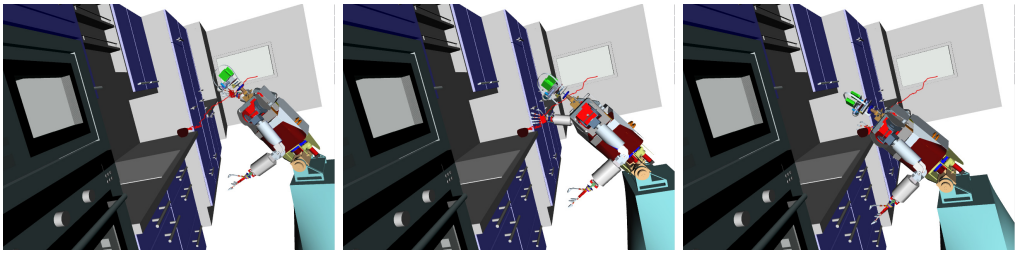
Figure 7.3: Plots of evaluated perceptive capability over a reaching trajectory from experiment *CoveredSphere*, when planning explicitly for it (*red*) and when perceptive capability is not considered in the planning stage (*blue*).

Planned w/ perceptive capability	*Sphere* avg. comp. time (nodes in tree)	*CoveredSphere* avg. comp. time (nodes in tree)	*SphereInCupboard* avg. comp. time (nodes in tree)
Yes	152.9 sec (13,992)	137.0 sec (7,948)	95.9 sec (8,720)
No	66.9 sec (8,307)	8.8 sec (1,726)	50.2 sec (7,886)

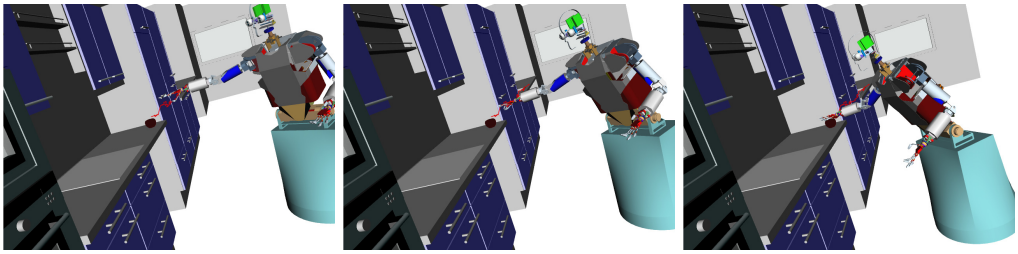
Table 7.1: Average running times and nodes in solution tree for the three experiments *Sphere*, *CoveredSphere* and *SphereInCupboard* with planning for perceptive capability (top row) and without (bottom row).

Planned w/ perceptive capability	*Sphere* avg. occlusion (last 10%)	*CoveredSphere* avg. occlusion (last 10%)	*SphereInCupboard* avg. occlusion (last 10%)
Yes	3.4% (2.8%)	22.3% (10.3%)	5.3% (1.1%)
No	22% (13.0%)	36.7% (43.8%)	19.5% (6.7%)

Table 7.2: Average percentage of object occlusion along the entire reach trajectory when planning with perceptive capability (top row) and without (bottom row). Numbers in parentheses give avg. occlusion over the last 10% of nodes in the solution.



(a)



(b)

Figure 7.4: Key frames from reach trajectories of ARMAR attempting to pick up an apple in a kitchen environment. When planning for future perceptive capability (*a*), the apple is grasped from the side and the head-mounted cameras are appropriately positioned. Without regard to perception (*b*), the robot does not fixate the object and occludes it with the manipulator.

future perceptive capability, and were run 50 times each on a typical office computer. Figure 7.3 and Tables 7.1 and 7.2 summarize.

Note the significantly reduced average occlusion of the object of interest over the solution trajectory when planning with perceptive capability in mind for all three experiments (e.g. from 19.5% to 5.3% for *SphereInCupboard*). Also note that the planner considering perceptive capability exhibits roughly comparable running times and solution sizes across experiments, suggesting that our approach scales well with environment complexity. Figure 7.3 shows percentage occlusion plotted over a typical reaching trajectory for both conditions of the *CoveredSphere* experiment. Note how, when explicitly considered in the planning phase, the measure increases significantly towards the end of the trajectory to meet our increasing minimum acceptable threshold and provides good operating conditions for any subsequent visual servo phase. This does not hold true when not planning for perceptive capability. Figure 7.4 shows two example reach trajectories computed with and without regard to perceptive

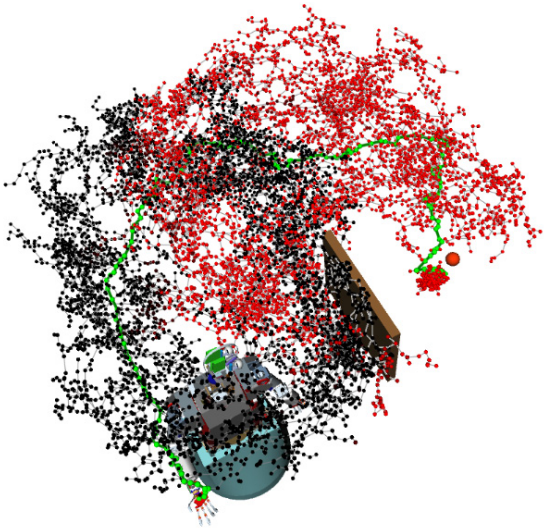


Figure 7.5: Workspace visualization of the generated RRT for experiment *SphereBehindBoard*, in which the robot aims to reach a sphere initially hidden from view behind a board. Small black spheres indicate configurations where the target is occluded, small red spheres denote configurations with satisfied visibility constraints. Green spheres represent the found solution trajectory.

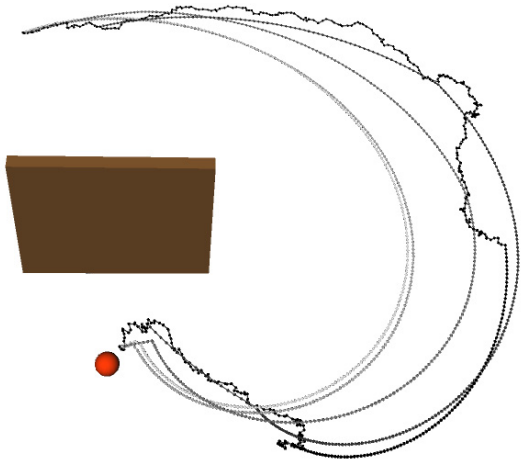


Figure 7.6: Solution trajectory for reaching the target object in experiment *SphereBehindBoard* after increasing iterations of path smoothing.

capability. Figure 7.5 depicts the solution found via RRT search for experiment *SphereBehindBoard*, where a board-like obstacle prevents the target object from being seen by the robot in certain

configurations. Note that the solution path found via the RRT algorithm appears rather jagged, but can easily be smoothed a-priori while remaining collision-free, as shown in Figure 7.6.

7.5 PLANNING REACHING MOTIONS ON HRP-2

We have extened our approach to reasoning about the robot's future perceptive capability during manipulation planning to the HRP-2 humanoid. Here, we integrate our perception-aware planning approach with the hybrid grasp analysis / manipulation planning framework for complex environments as proposed by Berenson et. al [4].

Given a particular object to grasp, this planning framework constructs a grasp-scoring function, incorporating both grasp quality metrics as well as information about the object's immediate surroundings and the robot capabilities. This function is then used to rank a precomputed set of candidate grasps for the object given the scene the object is placed in. The ranked grasps are then validated using collision checking against the environment and tested for reachability using the robot's inverse kinematics. Provided that the resulting arm pose is indeed free of environmental- and self-collision, a bidirectional RRT algorithm [36] is used to find a valid trajectory leading the robot arm from the current pose to the grasping pose. If any of the above steps fail, the next grasp from the ranking is chosen, and the process is repeated.

As described, the planning process remains decoupled from perception, and merely relies on an a-priori perception step to localize the objects to be manipulated with respect to the robot. As motivated previously for ARMAR, the use of perception for a subsequent fine-grained control phase in which the end effector is further adjusted to achieve a successful grasp is likely to yield higher grasp success rates and increased robustness to errors in execution, modeling or the initial perception stage. In light of this, we would like to ensure that perception remains operational throughout the reaching trajectory in its entirety.

More concretely, we employ the GPU-accelerated model-based tracker outlined in Chapter 6 as the perception component used for manipulation on HRP-2. If we do not explicitly reason about perception during the planning stage, the grasp planning framework outlined above often generates trajectories that correctly position the arm for grasping, but exhibit severe object occlusion by the manipulator throughout the reaching phase, as exempli-

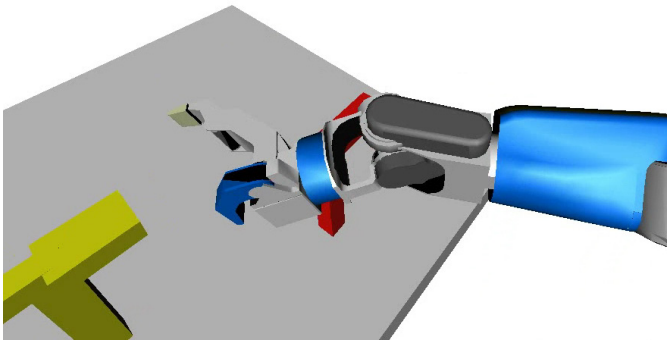


Figure 7.7: Manipulation without reasoning about perception on HRP-2. The robot arm severly occludes the target object (*red*) during the reaching motion, leading to likely tracking failure.

fied in 7.7. By planning for perceptive capability, we aim to ensure that we can continuously track the object of interest throughout the whole reaching motion.

To do so, we again compute a perceptive capability measure from simulation and modify the grasp planning process outlined as follows:

1. Before planning and throughout the reaching trajectory, we appropriately position the sensor in each configuration by orienting the robot's head towards the object to be grasped.
2. Object visibility is assessed for the IK solution returned for each candidate grasp. If too large a percentage of the object remains occluded in this final state of the grasp, the grasp candidate is discarded and the next candidate in the ranking is evaluated.
3. Once a grasp candidate is chosen and we enter the RRT trajectory planning process, perceptive capability is computed for each configuration examined. Configurations not satisfying a minimum perceptive capability threshold are discarded, ensuring a lower bound on the perceptive capability throughout the resulting reaching motion.

Note how in the case of grasping motions on HRP-2, only the arm and the rotational torso DOFs are planned for, implying

that the distance between sensor and object of interest cannot be varied. We thus explicitly set the head yaw and pitch joints to orient the head such that the object of interest lies as close to the center of the camera view in each configuration examined as the robot's joint limits allow. Also note how, once again, we use the percentage of object that is *un*-occluded as a heuristic proxy to the concept of perceptive capability in a particular state and posit that an object that remains unoccluded and fully in-view will remain well-trackable. In our experiments, we set the limit on allowable object occlusion to between 15% and 30%.

7.6 EFFICIENTLY COMPUTING OBJECT OCCLUSION

As with ARMAR, we compute the heuristic perceptive capability by simulating the robot's perception system from each configuration. This is accomplished by rendering the scene using a synthetic camera positioned and oriented to reflect the physical camera on the real robot head. The physical camera's intrinsic parameters, gathered during an a-priori calibration stage, are used to construct an appropriate OpenGL projection matrix for the simulated camera view.

When planning for future perceptive capability for ARMAR, evaluating the percentage of object occlusion in each candidate configuration proved to be by far the most computationally expensive operation, involving explicit comparison of two pixel buffers to count the number of pixels rendered for the current configuration vs. the number of pixels rendered for an unoccluded view of the object alone. In the case of HRP-2, we accelerate this operation significantly by exploiting the GPU.

We make use of the occlusion query mechanism found on modern GPUs [63]. Given a set of primitives to render, the occlusion query mechanism returns an exact number of fragments ('potential pixels') that would successfully pass all the tests of the rendering pipeline (frustum visibility, scissor, alpha, stencil and depth) and would hence actually get drawn on the screen. This mechanism is usually used to determine whole object visibility in complex scenes. The object is replaced by its bounding box and the query mechanism determines whether any part of the box is actually rendered. If the occlusion query mechanism determines that the bounding box is occluded entirely by other geometry in the scene, the object need not be drawn, eliminating unnecessary, potentially expensive rendering and shading operations. This

technique is widely used in graphics engines of current games and even forms the basis of an approach to efficient collision detection [23].

We use occlusion queries to precisely count the number of fragments of the object of interest visible in the robot's simulated camera view, as well as the number of fragments that would be visible were the object rendered alone. This is accomplished as follows:

1. Render only the sub-part of the scenegraph containing the target object by itself. Use occlusion query to record the number of fragments rendered as $\texttt{frag\_count\_alone}$.
2. Clear OpenGL buffers.
3. Render entire scenegraph *without* target object.
4. Render target object by itself into buffers already containing remaining scene geometry. Use occlusion query to record the number of fragments rendered as $\texttt{frag\_count\_with\_env}$.
5. Compute object *un*-occlusion as

$$\texttt{frac\_not\_occluded} = \frac{\texttt{frag\_count\_with\_env}}{\texttt{frag\_count\_alone}}$$

Since fragment counts are computed on the GPU, this process is very efficient, taking only about 1-2ms to complete even for complex scenes, with the only additional cost incurred over a normal rendering operation being the readback required to gather the two fragment counts from the GPU back to main memory (which stalls the rendering pipeline). The use of the occlusion query mechanism to quickly compute object occlusion drastically speeds up the computation of future perceptive capability in each configuration during planning and thus significantly reduces the overhead introduced by considering perceptive capability during the manipulation planning process. The discrepancies between planning times with and without consideration of future perceptive capability are now attributable mainly to the fact that the perception-aware planner discards a larger number of candidate states (due to them failing to meet the minimum perceptive capability requirements) than the perception-unaware planner.

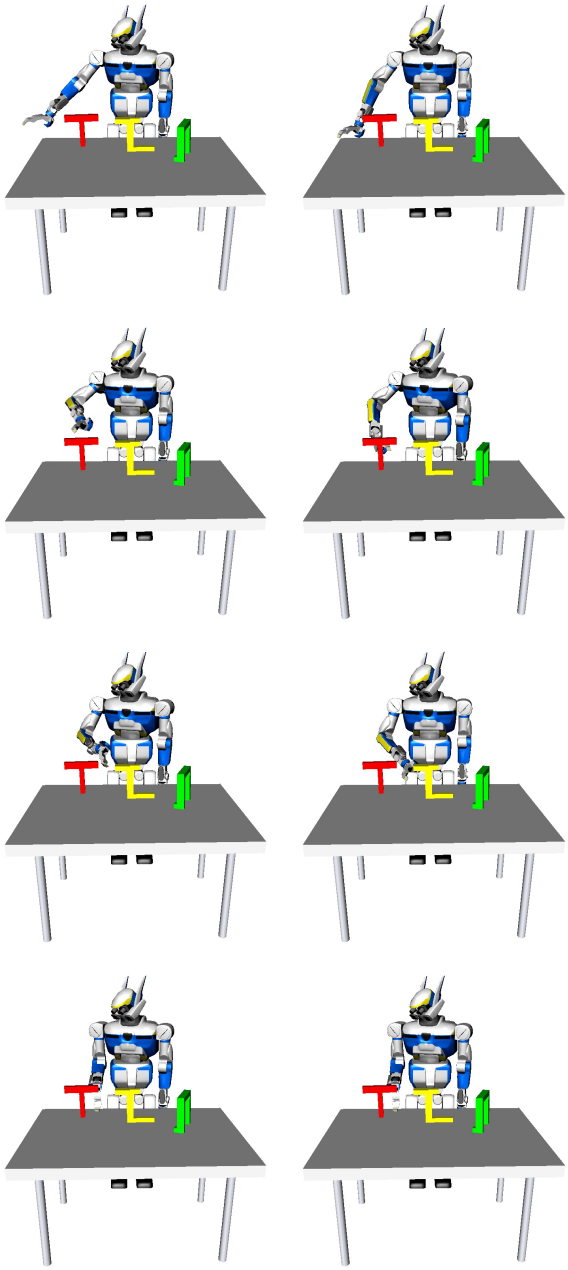


Figure 7.8: The HRP-2 humanoid executing a reaching motion towards an object to be grasped (red 'T'-shaped object). Trajectory resulting from perception-unaware planning (*left column*) and from planning with perceptive capability (*right column*).

7.7 RESULTS: HRP-2

7.7.1 *Simulation*

In our experiments with HRP-2, we sought to evaluate how well our GPU-based tracking system would operate during the

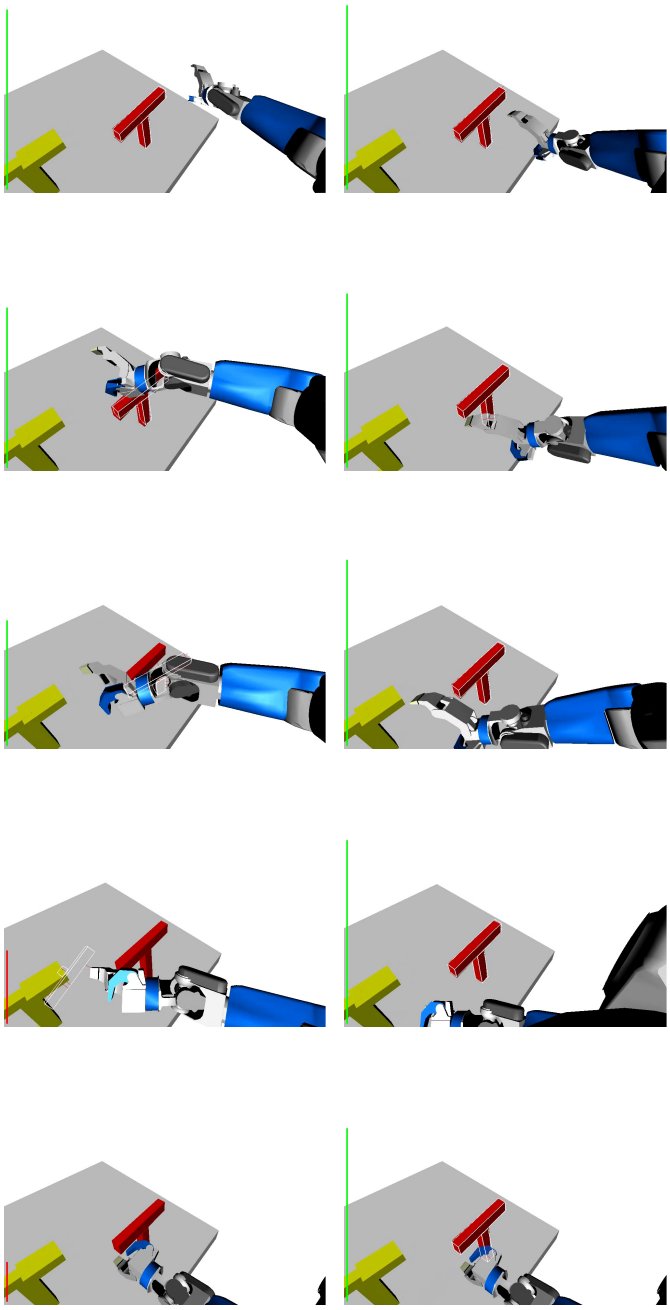


Figure 7.9: Target object tracked from synthetic HRP-2 camera view during reaching trajectories planned without (*left column*) and with (*right column*) perceptive capability. Colored bar on left of each image indicates tracker confidence (green=good, red=bad).

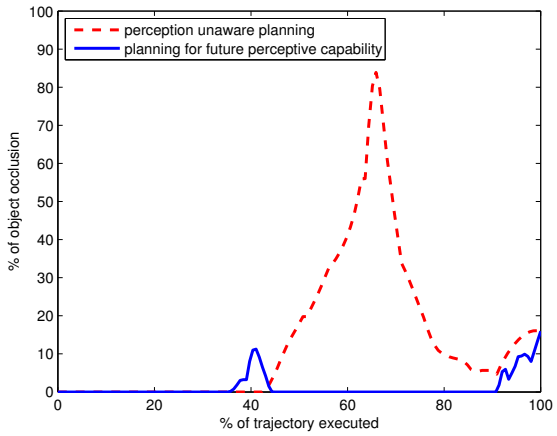


Figure 7.10: Plots of target object occlusion over a reaching trajectory planned without (*red / dashed*) and with perceptive capability (*blue / solid*).

execution of reach trajectories when planned with and without regard to future perceptive capability. To do so, we first tasked a simulated HRP-2 to grasp a red T-shaped object placed on a table in front of the robot among other objects. The executed trajectories were recorded from the point of view of the robot's simulated camera. We then executed the tracker on the resulting video sequences of synthesized scene views, using the same method and parameters as when operating on regular camera images.

Figure 7.8 shows an external view of a typical reaching motion, when planned with (*right column*) and without (*left column*) perceptive capability. Note the different resulting trajectories, despite identical initial and goal configurations for this example.

Figure 7.9 shows the same motions viewed from the robot camera, with the target object tracked using the GPU-based tracker. Note the significantly smaller amount of object occlusion throughout the trajectory when planning for future perceptive capability, as evidenced in the plots shown in Figure 7.10. In the perception-unaware case, up to 80% of the object is occluded and occlusion is so severe that it leads to tracker failure (third picture from top in Figure 7.9). When planning for future perceptive capability, occlusion never exceeds 15% and the target object is tracked successfully throughout the entirety of the reaching motion.

During our experiments the perception-unaware planner generated a typical reaching trajectory after 1–3s, with planning for

future perceptive capability typically taking between 7 and 23 seconds.

7.7.2 *Physical robot*

Ultimately, validation regarding whether or not the concept of planning for future perceptive capability does indeed provide tangible benefits in terms of execution reliability in the reach planning scenario outlined in this chapter can only be gathered on a physical robot platform.

To do so, we had HRP-2 plan reaching trajectories for the two experimental conditions (with and without reasoning about future perceptive capability) using the same experimental setup as in the simulation experiments described above. The GPU-accelerated tracker is used to initially localize the object to be reached for (again, a red T-shaped object made from Lego blocks), with the planner subsequently planning a reaching trajectory toward the object. Again, intrinsic parameters reflecting the actual camera mounted on HRP-2's head were used to generate the simulated views in each configuration the planner examines to determine perceptive capability via occlusion. The resulting trajectories were then executed on the physical HRP-2 robot. We continued the tracking process as the actual reaching motion was carried out, to assess the difference on tracking performance during execution of trajectories from the two experimental conditions.

Figure 7.11 shows HRP-2 carrying out a trajectory planned in a perception-unaware manner (*left column*), along with the corresponding tracker view (*right column*). Note how, just as reflected in the simulation experiments, on the physical robot the large amount of object occlusion that results from planning in a perception-unaware manner means that the tracker fails to track the object throughout the entire trajectory, despite the object remaining in-view (but occluded) throughout. The robot's manipulator simply occludes too large a percentage of the tracked object for perception to remain operational. Any fine-grained control stage potentially coming after the reaching motion has slim chance of success, unless the tracking process is re-initialized.

In contrast, consider the trajectory depicted in Figure 7.12, along with the associated tracker view. When planning using future perceptive capability as done here, object occlusion by the robot's manipulator is significantly less, ensuring that the tracker can remain operational during the entire reaching trajectory. With

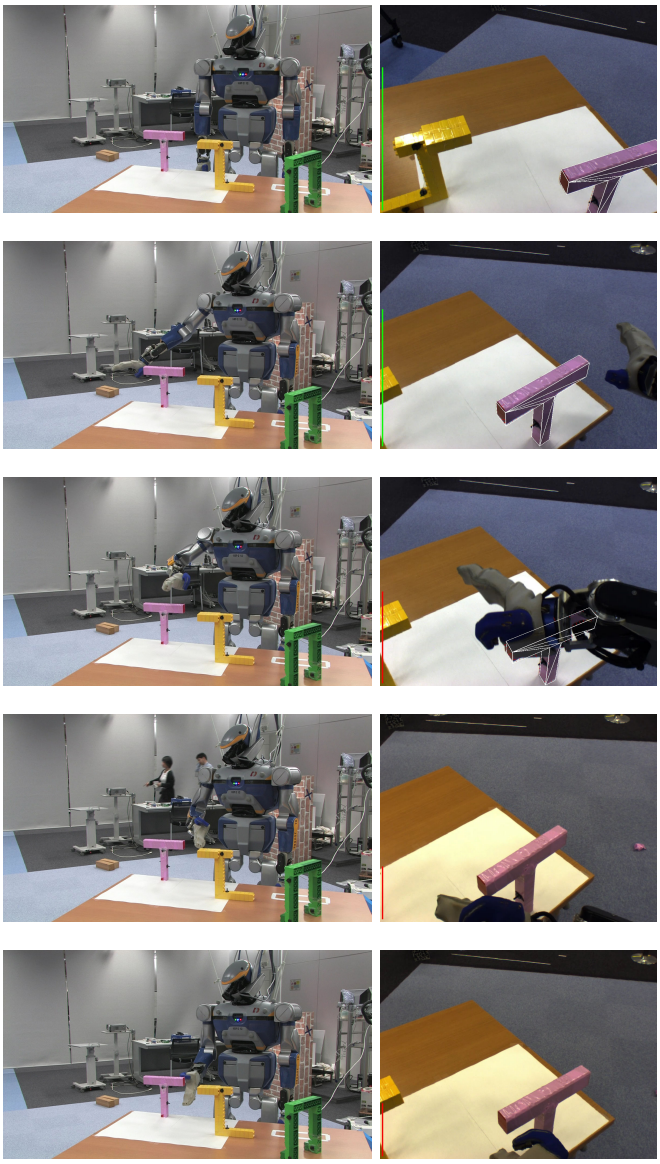


Figure 7.11: The physical HRP-2 humanoid executing a perception-unaware reaching motion towards an object to be grasped. External view (*left column*) and tracker view (*right column*). Note how the tracker loses track shortly after the third frame from the top.

the manipulator now positioned close to the tracked target object, conditions are ideal for any subsequent servoing phase that moves the robot's hand even closer to the grasp point and eventually closes the gripper around the object for a successful grasp.

The trajectories shown in Figure 7.11 and Figure 7.12 exhibit very similar occlusion profiles to those shown in Figure 7.10, with

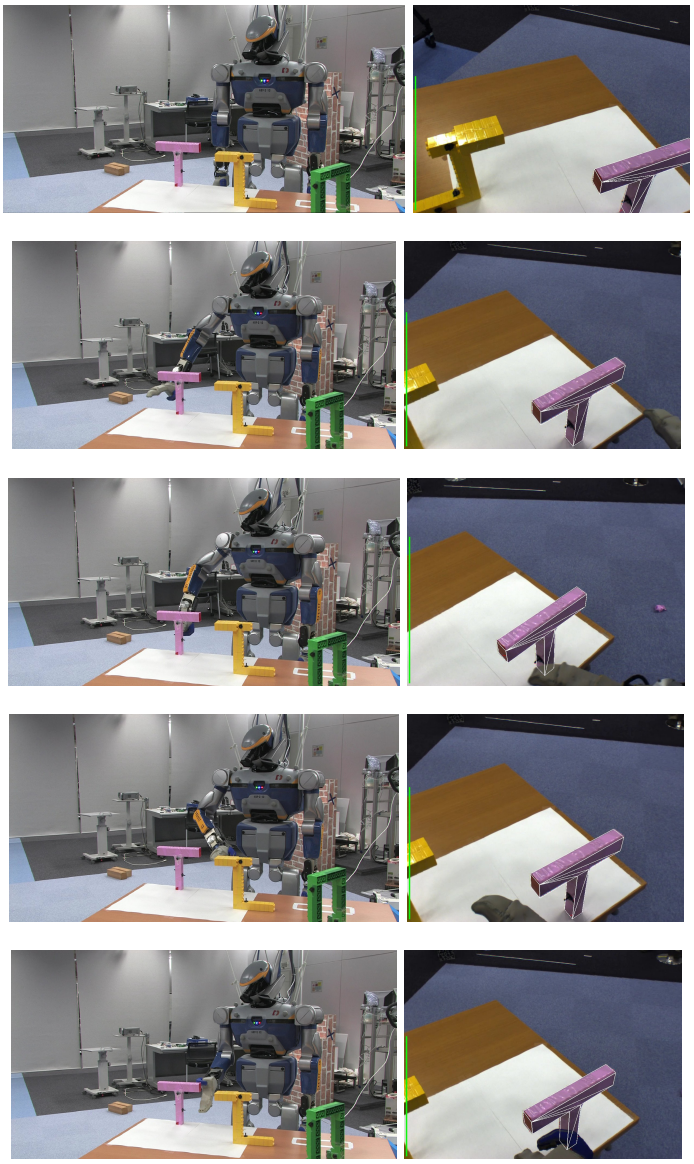


Figure 7.12: The physical HRP-2 humanoid executing a reaching motion planned with perceptive capability. External view (*left column*) and tracker view (*right column*). Tracking continues successfully throughout the entire motion.

perception-aware reaching motions never exceeding 15% object occlusion. Planning times for these experiments were nearly identical to those reported for the simulation experiments above.

7.8 PERCEPTIVE CAPABILITY IN PLANNING: OUTLOOK

The approach to planning for future perceptive capability outlined in this chapter and implemented on ARMAR and HRP-2 has several specific advantages over prior work in this area:

A. It can operate with an arbitrarily complex model of the perception system to arrive at the perceptive capability of each state. In our experiments, we aimed to distill the operating characteristics of our sensor system into measures (here: occlusion and distance from the sensor) that were straightforward to compute and that could proactively be used as good proxies to the robot's 'true' perceptive capability during the planning process. Of course, extending these measures to incorporate an explicit statistical assessment of sensor noise and error during the simulation process would be very desirable.

B. By computing the perceptive capability via simulation, we are able to deal with intricate robot and environment geometry that would otherwise preclude calculation of straightforward visual constraints as a metric of 'sensability'. Hardware-accelerated computation of perceptive capability means that simulating perception remains efficient enough for real-world planning scenarios.

C. Our approach does not require the sensor used to remain stationary. By exploiting sampling-based planning, we are able to compute solution trajectories that involve whole-body motion for complex systems with significant kinematic chains between sensor and manipulator.

Our experiments on HRP-2 show that our initial insights from simulation on ARMAR transferred well to the real robot domain, where the planning stage is preceded by a step of rapid initial object localization and environment reconstruction, yielding an environment model that can then be used for the forward-simulation used during planning to compute the perceptive capability.

The results arising from our reaching experiments on HRP-2 show that reasoning about perceptive capability during planning does indeed provide tangible benefits in terms of execution reliability by ensuring properly operating perception not only when planning manipulations, but also during navigation in complex environments, as shown in the next chapter.

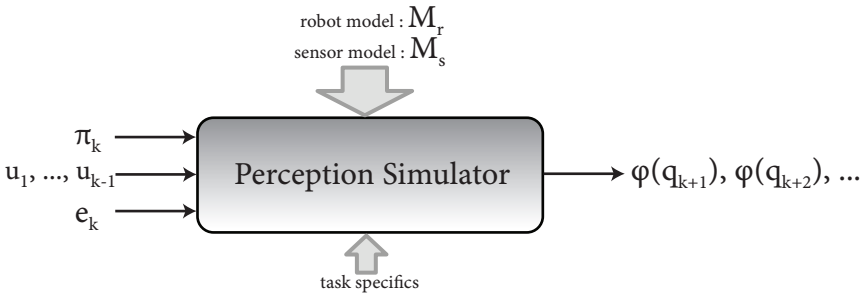


Figure 7.13: More sophisticated calculation of future perceptive capability. Simulated perception takes into account current plan (π_k), robot state via executed actions ($u_1, \ldots, u_{k-1}$) and environment representation (e_k) to calculate perceptive capability values ($\varphi(q_{k+1}), \ldots$), considering robot and sensor models as well as specifics of the task to be accomplished.

Note how in these experiments, 'perception-aware planning' was achieved by computing a single numerical measure via simulation and its subsequent use as another heuristic during the planning process. The dimensionality of the space planned in did not increase as a consequence of planning for future perceptive capability, ensuring that the process as a whole remained efficient. Another option is to shift some of the burden of finding motions that execute well under visual feedback to the search stage of the planning process, by explicitly including degrees of freedom particularly relevant to perception as part of the state space planned in. For instance, on a humanoid platform, the head joints may be explicitly included in the planner's search space, in the hope of leading to a wider variety of configurations that can be assessed for their perceptive capability during the planning process. This, however, leads to the usual planning trade-off of an increased search space and consequently longer planning times in exchange for finer granularity during search.

In the preceding, we have focused on using simulation to generate a single measure of perceptive capability for each state q examined. More abstractly, the 'perception simulator' component of Figure 7.1 may provide further lookahead and consider substantially more task, sensor and environment characteristics in determining appropriate measures of perceptive capability, as Figure 7.13 conceptualizes.

8

PLANNING FOR FUTURE PERCEPTIVE CAPABILITY: NAVIGATION

We extend the idea of explicitly considering the robot's future perceptive capability during the planning stage to the scenario of planning navigation strategies for a humanoid robot. Recall the motivating challenges for perception on a moving humanoid, as outlined in Chapter 2. Even a perception system, such as our GPU-accelerated tracker described in Chapter 6, capable of reconstructing the environment in accordance with the restrictive error tolerances required for complex locomotion tasks and able to do so quickly and robustly can only operate as well as the conditions arising from the robot's planned motions will allow it.

Consider the stair climbing scenario given in Chapter 6. Suppose that information from the perception system is continuously used to localize the robot and map the environment throughout the walking sequence. A perception-unaware planner may easily generate footstep paths that appear desirable from the perspective of its internal step-cost functions and heuristics, but which result in the tracked stairs falling outside the robot's field of view during the walking sequence, as exemplified in Figure 8.1. Such a scenario would likely result in tracking failure that is hard if not impossible to recover from. Perception would then be unable to further localize the robot or map the environment, forcing the planner to rely on the last-known-good map and location estimate and dead reckoning. In this case, it is very likely that the robot positioning error relative to the stairs exceeds the error bounds of 1–2cm deemed acceptable by the controller, resulting in (perhaps catastrophic) task failure if the robot attempts to climb the stairs.

We again draw on the principle of *mutual awareness* between the perception and planning stages from Section 2.2 to argue that such and similar scenarios can be avoided by informing the planning process about the need for subsequent perception during execution. This chapter presents our approach to planning for future perceptive capability in for humanoid navigation. We focus here specifically on the GPU-accelerated perception system and

Figure 8.1: HRP-2 executing a stairclimbing motion planned in a perception-agnostic manner. The stairs move outside the robot's view during locomotion and tracking subsequently fails.

height-based footstep planning method introduced in Chapter 6, with the HRP-2 humanoid serving as the robotic implementation platform.

8.1 CONSIDERATIONS

In defining how to plan for future perceptive capability during navigation, we adapt our problem definition for the manipulation case, given in Chapter 7, slightly. We still seek to find a robot motion $\tau_{FPC} \in \Phi$, leading the robot from an initial state to a goal state, that conceptually maximizes the robot's perceptive capability, given by $\varphi_{PC} : x \in \mathcal{X} \mapsto \varphi_{PC}(x) \in \mathfrak{R}$. However, we now define τ by a discrete set of footholds $\tau = (t_0, t_2, \dots t_n)$ with $t_0 = x_{init}$ and $t_n = x_{goal} \in G$, being the initial and goal states, respectively. Footholds are chosen as part of the footstep planning process given the input information

$$\mathcal{I} = (x_{init}, G, e, \mathcal{A}) \tag{8.1}$$

where e is a representation of the environment and $\mathcal{A}$ defines a set of possible stepping actions $a \in \mathcal{A}$ the robot can execute in any given state x_i to yield the successor state x_{i+1} as

$$x_{i+1} = \mathrm{Succ}(x_i, a, e) \tag{8.2}$$

The desired robot motion maximizing future perceptive capability τ_{FPC} can now be defined (cf. Equation 7.2) as the sequence of footholds $(t_0, t_2, \dots t_n)$ that satisfies

$$\tau_{FPC} = \underset{\tau}{\operatorname{argmax}} \sum_{i=0}^{n} \varphi_{PC}(t_i), t_i \in \tau \tag{8.3}$$

It is worth noting that, especially when compared to the problem of planning reaching motions from Chapter 7, we are dealing with a much lower dimensional state space throughout the planning process, with each foothold nominally defined by its $SE(2)$ position (x, y, θ) on the floor area plus an index variable $\in \{L, R\}$ (indicating the stance foot), with foot height being determined by the environment height at location (x, y). This allows A* search to be used for planning instead of a randomized approach such as RRTs [10]. However, it also implies that many of the robot degrees of freedom crucial for perception (e.g. the torso or neck joints) do not normally form part of the state space we plan in. This is one reason why situations such as the one shown in Figure 8.1 can arise.

Recall that the aim of planning for future perceptive capability is to find *good planning strategies to aid perception*. That is, to increase robustness of perception during operation by making informed decisions during the planning stage. In the case of navigation, how should we define good operating conditions for perception to accomplish the task at hand and how can these be realized through planning? How should we define the heuristics flowing into our perceptive capability measure that appropriately capture the proper operating characteristics of the perception system used on the robot? Of course, we would ideally like to have a perfect description of the sensor used, encompassing all necessary and sufficient conditions that lead to good sensor operation. In practice, several considerations that coarsely describe the majority of relevant sensor characteristics needs to be established.

Consider that, when using the model-based tracking system we employ on HRP-2, we would informally like all objects of interest to the navigation problem to remain "in view and well trackable" throughout the walking sequence. This might mean ensuring that, during the walking sequence, the following hold true:

1. Tracked objects do not leave the camera view in their entirety

2. As much as possible of a tracked object remains inside the camera view & close to the image center
3. Tracked objects remain as unoccluded as possible
4. A good edge response is achieved for tracked objects
5. Camera focus remains adequate for each object's distance from the sensor
6. Background clutter that may lead to false edge matches is minimized
7. ...

Clearly, several of these desirable characteristics lie outside of our direct control (e.g. lighting conditions and foreground/background texture and color will determine the quality of edge responses, cluttered background views may be unavoidable) and certainly outside the realm of influence of the planning process. However, some should be achievable by appropriately augmenting the footstep planning process to reason about perception. In particular, we address points 1–3 above, as follows.

VISIBILITY Whether and how much of an object is visible throughout the walking path is explicitly computed during the planning process and employed as one of the cost metrics the planner uses when evaluating each state. The visibility measure accurately reflects the robot's ability to actively sense the environment from a particular configuration by moving its head, as well as the precise field of view yielded by the particular camera / lens configuration used for sensing on the robot head.

OCCLUSION While in our navigation scenarios, self-occlusion is not a major issue (the robot's upper body remains upright and the arms remain raised, but to the side of the robot, resulting in minimal possible self-occlusion), it is very possible for certain parts of the environment to be occluded by others. If occlusion becomes too severe, the tracking system may lose the object of interest, potentially resulting in task failure (e.g. in the case of stair climbing). To prevent this, we explicitly assess the percentage of occlusion of objects of interest in the environment during planning. This is done by efficiently simulating the robot's perception process from each state examined by the planner using GPU-accelerated rendering. This allows us to determine precisely, for each object, how many pixels are visible or occluded from

a particular robot view. The resulting occlusion percentage can then be used as another cost metric during the footstep planning process to eliminate paths where occlusion would result in poor operating conditions for perception.

VISUAL SERVOING The VISIBILITY criterion above ensures that a tracked object can remain in view. To actually implement this during execution, we employ a fast visual servoing loop that uses the 6DOF pose recovered by the tracker to determine head yaw and tilt angles for HRP-2 that aim the camera at the object of interest during the walking sequence, ensuring it remains within and close to the center of the image.

These serve as viable heuristical proxies to the 'true' perceptive capability given the tracking system used on HRP-2 for our planning scenarios, with the goal of increasing robustness and reliability of perception during execution.

8.2 PLANNING FOR VISIBILITY

Say that the robot is currently at a particular location indicated by the foothold (x_1, y_1, θ_1, L), with the next footstep to be executed leaving the robot at (x_2, y_2, θ_2, R). How do we determine whether a particular target object at location $\mathbf{l} = (l_x, l_y, l_z)$ in world coordinates (usually the object's origin) is visible by the robot? For the class of locomotions we execute on HRP-2, it is known that the torso position and orientation can be constructed straightforwardly from the position and orientation of the two feet (simply by averaging). We also know that, during walking, the camera remains directly in line with the torso if the robot is looking straight ahead, at a height of approximately 1.4m above ground. This yields the position and orientation of the camera in world coordinates as a transform $\mathbf{T}_{cam}$ (in terms of its (x, y, z) position and yaw angle, with pitch and roll set to 0). The relative vector between the camera and the target object is then given by

$$\mathbf{r} = \mathbf{T}_{cam}^{-1}\,\mathbf{l} \tag{8.4}$$

Given this relative vector $\mathbf{r}$, we can now determine the yaw and pitch angles that would need to be set at the robot's head joints

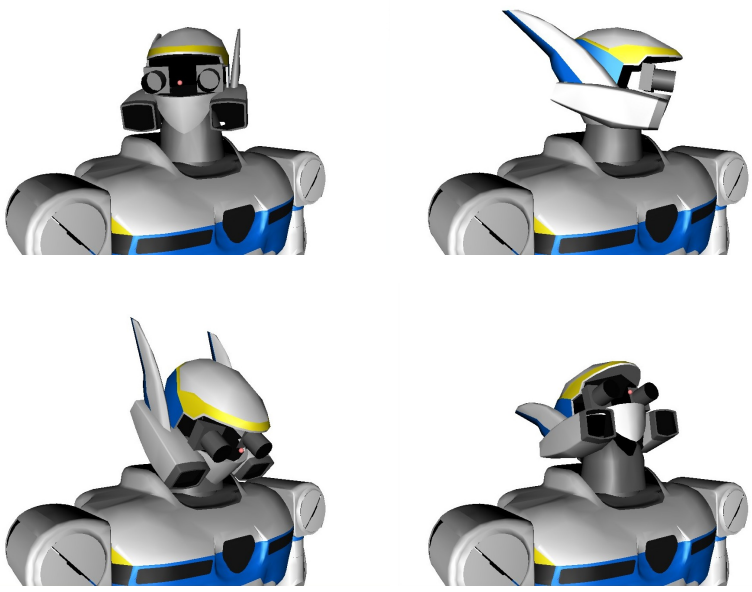


Figure 8.2: Visualization of the absolute head joint limits for the HRP-2 humanoid. Top shows yaw $\sim (-45°, 45°)$, bottom shows pitch $\sim (-28°, 28°)$. A linear gradient $\mathrm{pitch} = 45° - k \times \mathrm{yaw}$ with $k = 0.5952$ (determined by experimentally moving the head through its range of motion and linear fitting) is used on the physical robot when setting values of yaw and pitch in combination to ensure there is no self-collision.

in order to position the target object at the center of the camera view via as

$$\mathrm{yaw}_{head} = \arctan\left(\frac{r_y}{r_x}\right) \tag{8.5}$$

$$\mathrm{pitch}_{head} = \arctan\left(\frac{r_z}{\sqrt{r_y^2 + r_x^2}}\right) \tag{8.6}$$

If yaw_{head} and pitch_{head} are valid angle settings in accordance with the robot's head joint limits and self-collision safety (see Figure 8.2), the target object can be positioned at the very center of the camera view from the current robot configuration. Such a configuration should receive a low visibility cost. If yaw_{head} and pitch_{head} fall outside the limits, we won't be able to center the object in the camera view perfectly and the configuration should receive a higher cost. However, even when centering is not possible, the object might still remain entirely or largely in

view and well trackable. The robot's field-of-view should thus also be taken into account when computing a visibility cost.

We can determine the field of view resulting from the particular camera / lens setup used on HRP-2 from the intrinsic camera parameters resulting from calibration. The field of view in either the horizontal or the vertical direction is simply defined as $fov = 2\arctan(\frac{d/2}{f})$, where d denotes the image (or sensor) size and f the focal length. For the Flea2 camera mounted on HRP-2 (having a 4.8mm $\times$ 3.6mm CCD) with a 4.8mm lens, this yields a 53.13° horizontal by 41.11° vertical field of view.

8.2.1 *Computing Visibility Costs*

We can now define a cost function that takes the head angles yaw_{head} and $pitch_{head}$, required to look at the target object in a particular configuration, and returns a cost encoding the following criteria:

- Configurations where the object can be centered in the view should be assigned a low cost
- The cost should increase exponentially as the object moves away from the center of the image to the border
- Configurations where the object lies outside the view should receive a very high cost

We define this cost function, denoted h_{vis} as

$$h_{vis} = \max(h(yaw_{head}), h(pitch_{head})) \tag{8.7}$$

$$h(\theta) = \begin{cases} 0, & |\theta| < \theta_{max} \\ e^{k(|\theta| - \theta_{max})} - 1, & \theta_{max} \leqslant |\theta| \leqslant \theta_{max} + \frac{\theta_{fov}}{2} \\ c, & |\theta| > \theta_{max} + \frac{\theta_{fov}}{2} \end{cases} \tag{8.8}$$

where c is the high cost when the target is out of view, and k is chosen to make $h(\theta_{max} + \theta_{fov}/2) = c$. θ_{max} and θ_{fov} are the limit and field of view of the joint. Figure 8.3 shows plots of the two functions.

Typical costmaps constructed using the visibility cost function h_{vis} and used by the planner together with the usual stepping costs and other heuristics are illustrated in Figure 8.4 for four

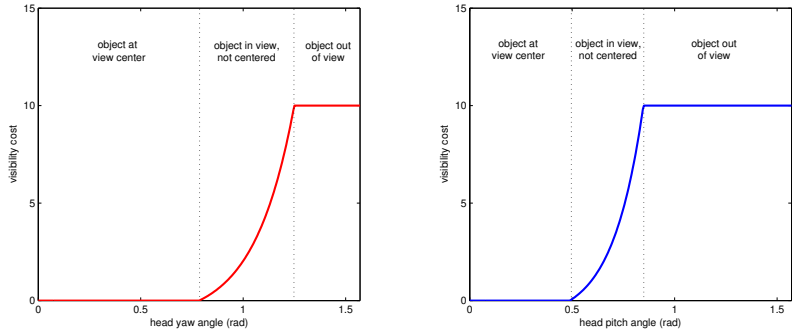


Figure 8.3: The two cost functions h_{yaw} (*left*) and h_{pitch} (*right*), with the x axis denoting the absolute value of the yaw and pitch angles. The costs returned remain at 0 while the robot head can be oriented to center the object in the view. Once this is no longer possible, costs increase exponentially up to a maximum cost value (10 here) as the object proceeds to move towards the edge of the field of view. The maximum cost value continues to be returned if the object lies outside the view.

different sample robot orientations. In these scenarios, the robot attempts to maintain visibility of the stairs located in the center of the map. Note how, as the robot moves closer toward the stairs, the visibility cost increases both due to head pitch ('circular' increase in cost during a full-frontal approach) and head yaw ('diagonal' increase in cost when the stair are positioned to the side of the robot). Note also the gradual increase in cost while the stairs remain in the field of view, but can no longer be centered in it. Visibility costmaps are quick to compute and can further be accelerated by a simple caching scheme, if we limit the allowable torso orientations to a fixed number. Also note that there is no implicit requirement for costmaps to remain static. The object to be kept in view may move during locomotion, resulting in an updated relative position between the object and the robot, from which a new costmap may be generated, to be used for re-planning.

As mentioned in Figure 8.2, in practice we make the head pitch limit depend on the current head yaw angle, to ensure that self collisions are avoided. To do so, we use a linear gradient $limit_{pitch} = 45° - k \times yaw_{head}$ with $k = 0.5952$. When this alternative visibility model is used, the resulting visibility costmaps differ slightly in their appearance, with the cost contribution

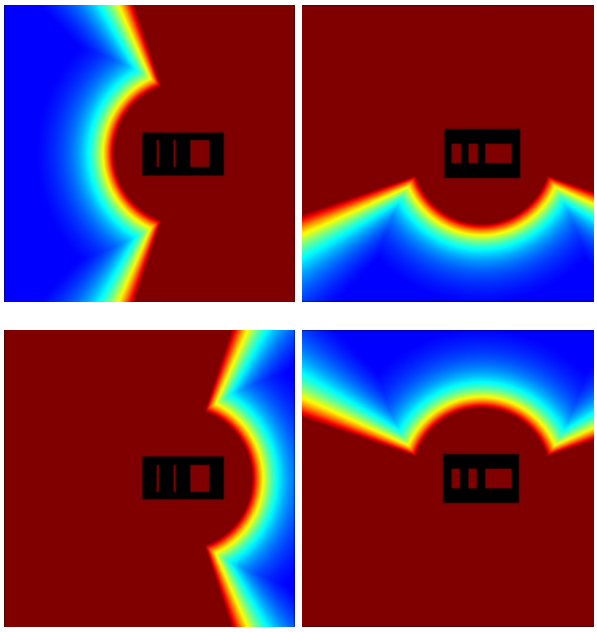


Figure 8.4: Sample visibility costmaps at different robot orientations. The robot attempts to keep the stairs in view. The maps reflect robot orientations of $(0°, 90°, 180°, 270°,)$, clockwise from top left. Blue denotes low cost, red denotes high cost. Maps were constructed with yaw and pitch limits as well as camera field of view values as used on HRP-2.

from pitch and yaw now no longer separable into 'circular' and 'diagonal' components, as Figure 8.5 illustrates.

8.3 SERVOING THE ROBOT HEAD

Even given a planned path that satisfies the visibility criteria, the robot must still actively move its head while walking to ensure that objects of interest remain in view. We accomplish this by using a visual servoing loop that directs the robot's gaze at the object.

To do so, we make use of the translation component $\mathbf{t}$ of the object pose $\mathbf{E} = [\mathbf{R}, \mathbf{t}]$ recovered by the GPU-based tracker at any point in time during the walking sequence. Recall that $\mathbf{E}$ is a transform localizing the tracked object with respect to the camera. Therefore, $\mathbf{t}$ can be straightforwardly used to determine the yaw and pitch head angles, relative to the current head orientation, required to position the object in the center of the view, as shown previously in Equation 8.5 and Equation 8.6.

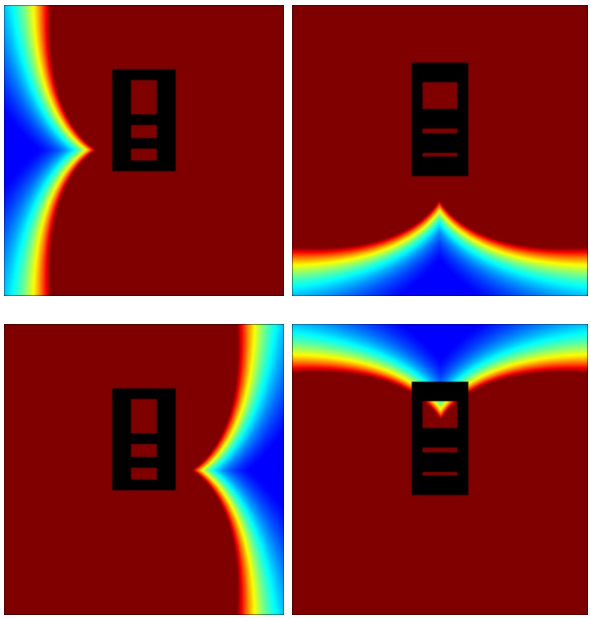


Figure 8.5: Sample visibility costmaps at different robot orientations, with pitch joint limit set from current yaw angle. The robot attempts to keep the stairs in view. The maps reflect robot orientations of $(0°, 90°, 180°, 270°,)$, clockwise from top left.

An exponential control law with velocity and acceleration limits for safety is then used to determine the angles to be commanded to the head motors to servo the head towards the desired orientation. The controller is executed at a rate of roughly 100Hz, with the target yaw and pitch angles refreshed at 30Hz by the tracking process, resulting in smooth servoing of the head towards the target orientation, both during walking and when following a manually moved object, as shown in Figure 8.6. To further increase servoing performance, it is possible to move the control loop into the robot's real-time layer, yielding more executions per second. However, we did not find this to be necessary during any of our experiments.

8.4 RESULTS: PLANNING FOR VISIBILITY

We evaluate our approach to planning for visibility during navigation using two scenarios. In both cases, the GPU-accelerated tracker combined with the visual servoing behavior outlined above is used for perception, with the footstep planner now planning walking paths satisfying the visibility criteria. The tracker generates heightmaps, localizes the robot and defines the target

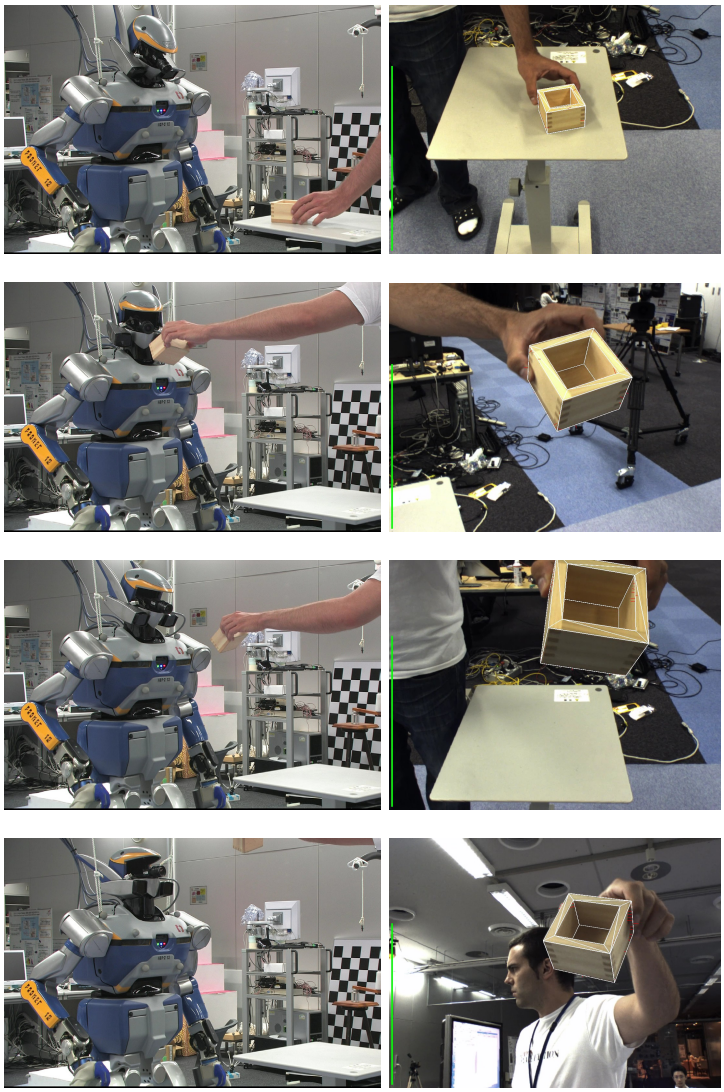


Figure 8.6: The HRP-2 humanoid servoing its head visually to follow a sake cup moved by an experimenter. External view (*left column*) and tracker view (*right column*).

location in the heightmap (corresponding to the target object) that should be kept in view, sending the resulting information to the planner, where the actual visibility cost computation is done during the planning phase. HRP-2's usual walking controller is used to actually execute the motions.

In the first scenario, the robot was commanded to plan a path towards a goal diametrically opposed from it, behind a set of stairs that had to be kept in view throughout the walking sequence. This scenario evaluates how the added visibility constraints shape

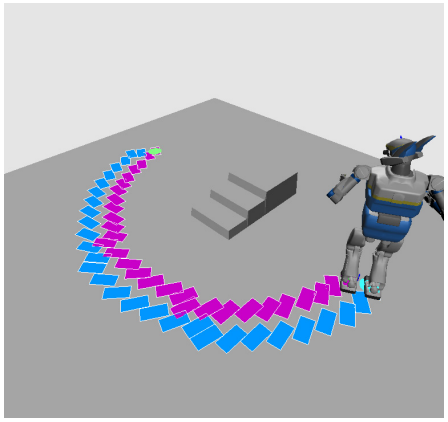


Figure 8.7: Path resulting from planning with visibility reasoning. HRP-2 navigates to a goal location while maintaining the target object (stairs) in view.

the resulting planned footstep paths. In the second scenario, we have the robot navigate to the top of the stairs, which it needs to keep in view. This scenario seeks to improve on the general stairclimbing task to specifically avoid situations such as shown in Figure 8.1.

8.4.1 *Maintaining Obstacles in View*

We executed 5 experiments requiring the robot to keep the stairs in view while walking to its goal, located opposite of the stairs. Figure 8.8 (*left column*) shows the robot executing one such sequence. Figure 8.7 illustrates the corresponding path found by the footstep planner. Note how planning results in a semi-circular walking path around the stairs that keeps the robot facing the object at all times. This corresponds closely to the circular shape of the visibility cost function as shown previously in Figure 8.4. Visibility-unaware planning would have chosen a more direct, shorter route, composed of more forward-stepping motions, but which would have moved the stairs outside of the robot's view, as shown in Figure 8.9

The tracker view corresponding to this walking path is shown in Figure 8.8 (*right column*). The tracker self-initialized and proceeded to track the stairs throughout the sequence. It can be seen that the interplay of planning for visibility and the visual servoing behavior result in the stairs never leaving the robot's view, with the second, central step of the stairs remaining close

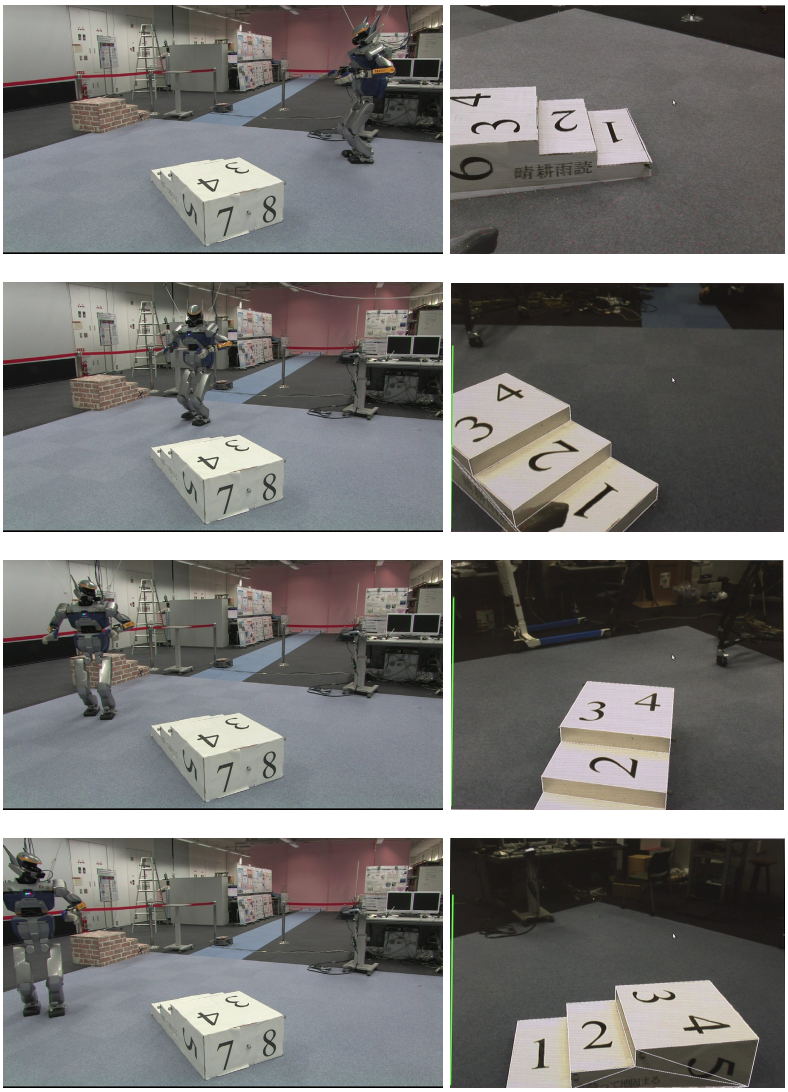


Figure 8.8: The HRP-2 humanoid navigating while maintaining an object in view. External view (*left column*) and tracker view (*right column*). First frame on right shows tracker during initialization phase.

to the image center. The tracker was able to continuously track the object well throughout the sequence, as shown by the green bar representing tracking confidence.

For this experiment, planning took around 7 seconds, generating a plan composed of 50 footsteps. The overall visibility cost along the path recorded was 8.54, yielding an average cost of 0.17 per step of the resultant walking sequence. When ignoring visibility during planning, the resulting path is shorter (17 steps), but exhibits a drastically larger overall visibility cost of 146.97, or

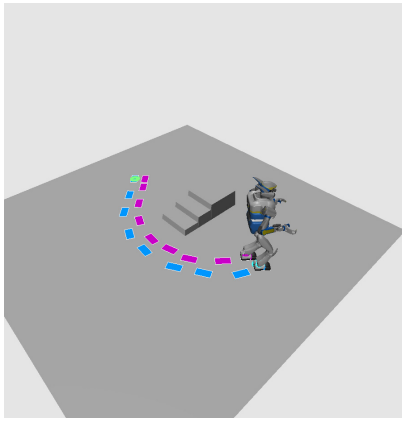


Figure 8.9: Path resulting from perception-unaware planning. HRP-2 navigates to a goal location, but fails to keep the target object (stairs) in view.

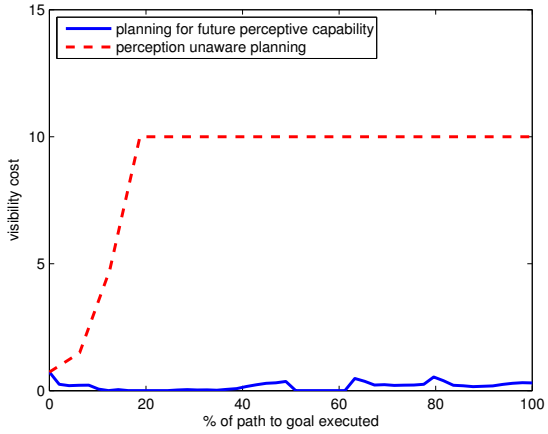


Figure 8.10: Plots of visibility cost for footstep paths around a set of stairs planned without (*red / dashed*) and with future perceptive capability (*blue / solid*).

8.65 per step of the resulting walking sequence. The visibility costs throughout the walking motion for perception-unaware planning and for planning with perceptive capability are plotted in Figure 8.10. Note how the stairs quickly move out of view and remain there (corresponding to the maximum cost of 10) when visibility is ignored.

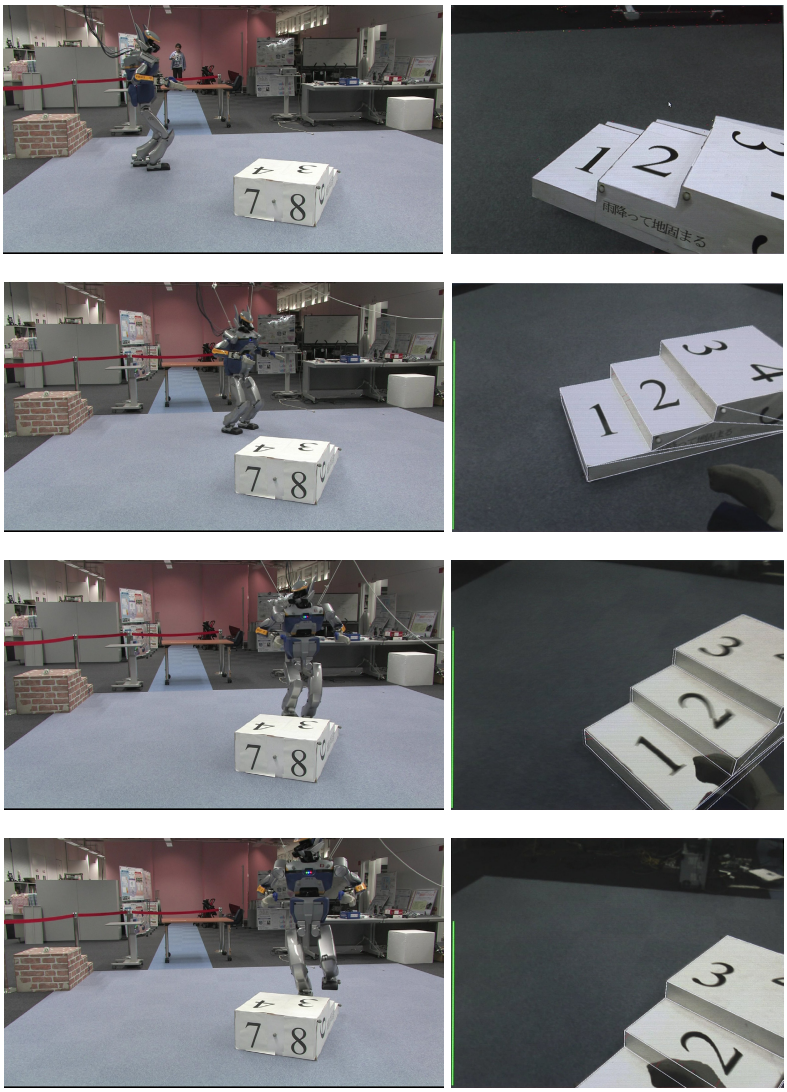


Figure 8.11: The HRP-2 humanoid climbing a set of stairs while maintaining them in view. External view (*left column*) and tracker view (*right column*). First frame on right shows tracker during initialization phase.

8.4.2 *Stairclimbing with Visibility*

We also incorporated visibility into the planning of stairclimbing locomotions. One issue that arises with stair climbing paths is that at some point before the robot reaches the top of the stairs, the stairs *will* move out of view. Given HRP-2's maximum head pitch angle, it is impossible for the robot to look down enough to see the stairs when standing on the topmost step. In fact, it is usually the case that once the robot has locomoted onto the

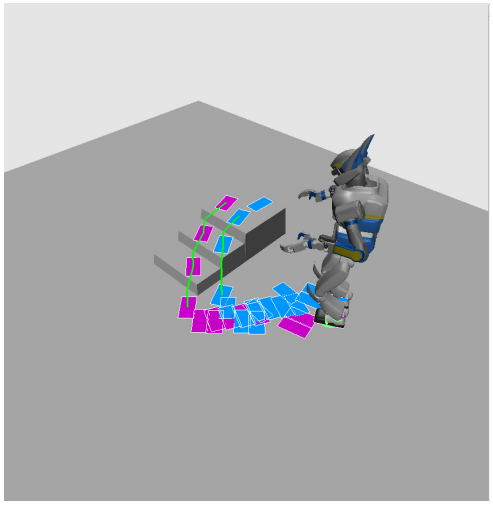


Figure 8.12: Path resulting from planning with visibility reasoning. HRP-2 climbs a set of stairs while maintaining them in view.

first step, the stairs will have moved out of view enough to cause tracker failure. However, the longer the robot can remain tracking the stairs during the approach, the more accurate the localization of the robot feet relative to the stairs will be. Our aim for this experiment, then, is to maintain good visibility of the stairs for as long as possible during the walking sequence, avoiding situations where good visibility could have been achieved, but was not due to the planner's obliviousness to the perception requirements.

A related issue is that visibility costmaps constructed as outlined in Section 8.2.1 will assign a very high cost to step locations actually on the stairs (as can be seen in Figure 8.4), precisely because the robot can no longer see the stairs from those locations. This implies that the planner will try to avoid these during search. To ameliorate this issue, we modify our visibility costmaps to return low cost for locations on the stairs (essentially 'coloring them blue' in Figure 8.4). This reflects the task intuition that, once the robot has committed to climbing the stairs, they are likely to be out of view and attempts to achieve good visibility will be futile anyway.

Figure 8.11 (*left column*) shows an example stair climbing sequence planned with consideration for visibility, along with the corresponding tracker view (*right column*). Figure 8.12 shows the path computed by the planner, executed by a simulated robot. Note how instead of heading straight for the stairs from the initial position, the robot first follows an arc that allows it to keep

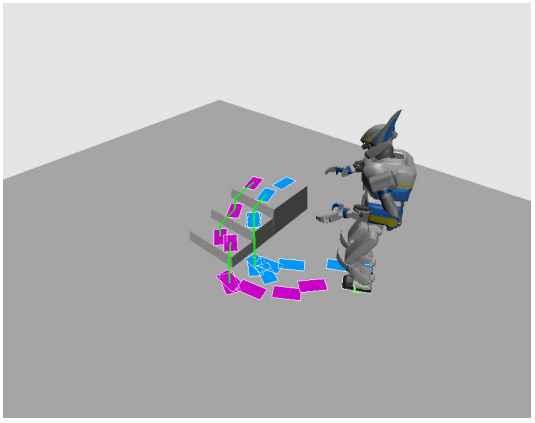


Figure 8.13: Path resulting from planning without visibility reasoning. HRP-2 takes a more direct path toward the stairs, resulting in the stairs moving out of view much sooner.

the stairs in view in accordance with the visibility cost maps, for as long as possible before ultimately committing to climbing the stairs. Compare this to the path resulting from perception-unaware planning shown in Figure 8.13, where a much more direct route is taken, resulting in poorer conditions for perception. From the tracker view of the executed visibility path, it is evident that visibility planning together with our visual servoing approach kept the stairs in-view for a majority of the walking sequence, until they are inevitably lost when actually climbing the stairs.

For this experiment, a 28-step plan was found after 5.09 seconds. The recorded total visibility cost along the resulting path was 1.11, for an average visibility cost of 0.03 per step. Compare this to the path generated by a perception-unaware footstep planner (found after 6.22 seconds—visibility planning generally but not always results in slightly longer planning times), which yields an overall cost of 120.18 for an 20-step plan, resulting in a cost of 6.01 per step. Again, perception-unaware planning generates a shorter path but with drastically larger visibility cost. The visibility costs throughout the walking motion for perception-unaware planning and for planning with perceptive capability are plotted in Figure 8.14. Note how again the stairs quickly move out of view and remain there (corresponding to the maximum cost of 10) when visibility is ignored.

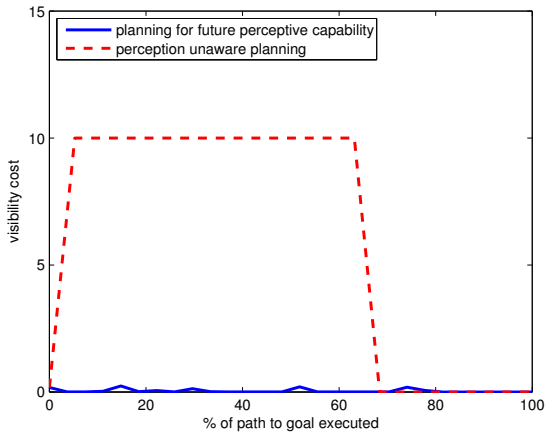


Figure 8.14: Plots of visibility cost for footstep paths climbing a set of stairs planned without (*red / dashed*) and with perceptive capability (*blue / solid*). Note that both plots dip as the robot begins to climb the stairs, where visibility costs have explicitly been set to zero.

8.5 PLANNING TO AVOID OCCLUSION

While explicitly considering object visibility during the planning stage serves to significantly reduce the chance of tracking failures during navigation by keeping the object of interest in-view throughout the walking sequence, this approach remains fundamentally a geometric one (does the object fall within the robot's view cone?), albeit one complicated by the kinematic complexity of a humanoid robot platform. Pure visibility planning, however, fails to take into account another major source of failures for the 3D perception system used on HRP-2: occlusions.

Suppose the robot is tasked with navigating in an environment filled with three-dimensional obstacles while continuing to track a certain landmark object (be it for localization purposes, to eventually grasp it, etc.). Depending on the particular path chosen by planning, it is quite feasible that the resulting walking sequence leads the robot through configurations in which environmental obstacles occlude the view of the landmark object to be tracked, despite steps chosen in a way that keep the landmark object in-view. Visibility-aware planning alone, as outlined above, is unable to detect such circumstances, quite possibly leading to tracking and task failure if the occlusion is severe enough.

To incorporate reasoning about occlusion into the planning process, we employ an approach very similar to the one outlined in Section 7.6. For each footstep state examined by the planner during search, we simulate the robot's perception system by rendering the scene using a synthetic camera set up to reflect intrinsics of the real camera mounted on HRP-2's head. We position this camera at the appropriate height off the floor from the current foot positions, in the same manner as described above for visibility planning. We then orient the camera to point directly at the landmark object the robot should keep tracking. Note that, at this stage, the robot's kinematic ability to orient the camera in this matter is not verified, meaning that configurations where the robot's chest is actually pointing away from the object of interest (which hence cannot be seen) will receive a low occlusion cost as long as no inter-object occlusion occurs from that configuration. During the actual planning phase, visibility costmaps (as described above) are used in addition to occlusion reasoning, ensuring that such configurations receive high cost and are avoided during the walking sequence.

We employ the same hardware-accelerated method of efficiently computing percentage of object occlusion using the OpenGL occlusion query extension as described in detail in Section 7.6, resulting in occlusion costs for each configuration in the interval $[0, 1]$, reflecting object occlusion percentages from 0% to 100%. These costs are then scaled and appropriately weighted during the planning stage so that they can be combined with visibility costs and stepping costs for planning. Occlusion for each configuration is computed quickly, requiring between 0.6ms and 1ms per configuration for the typical scenes we used. To further decrease computation time, we use a caching scheme whereby the occlusion cost computed for a particular configuration is stored in memory and re-used for a different configuration if the two configurations only differ in the robot's torso orientation, but not torso location over the floor, reflecting scenarios where occlusion by other object between the robot and the landmark should be identical.

Figure 8.15 visualizes typical occlusion costmaps generated in this manner. Note the high (red) occlusion cost for configurations in front of the door or when the textured stairs partially occlude the landmark object. Also note the smooth gradients indicating partial occlusion. The maps used to compute occlusion costs are generated by our GPU-accelerated tracker in terms of their constituent objects (e.g. the door and two stairs in this example),

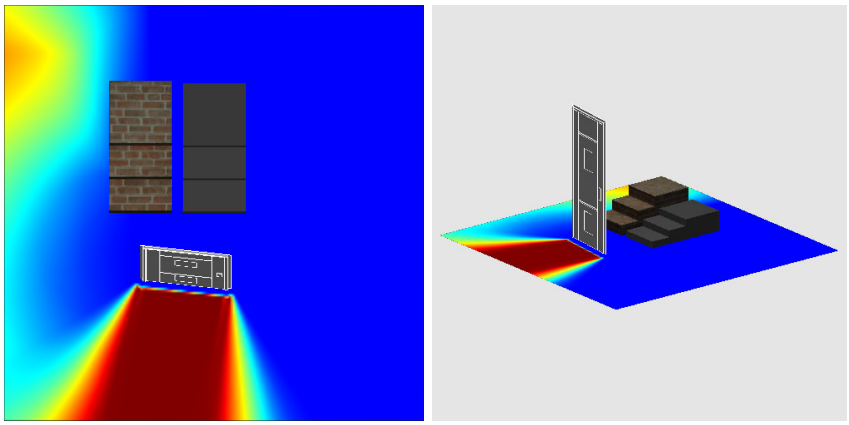


Figure 8.15: Sample occlusion costmaps computed for a scene with two stairs and a door. The grey stairs are the landmark object to be kept in view. High cost regions are colored red, low cost regions blue.

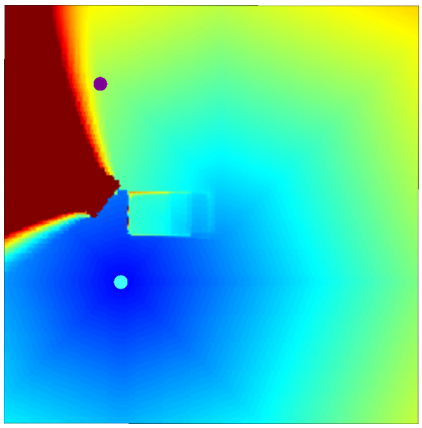


Figure 8.16: Visualization of the heuristic used during occlusion-aware planning. Only the occlusion component of the heuristic (not the obstacle/free-space component) is shown. Initial state: purple. Goal: light blue.

their relative positioning (as 6DOF transforms) and information about which target should remain occlusion-free. This fully 3D map information is then sent to the planner, where the actual occlusion computation takes place during the planning process, leveraging another GPU. The tracker also generates visibility costmaps (composed of a heightmap and a target location—the center of the landmark object—to be kept in view) simply by rendering the 3D scene in a top-down manner and reading back the depth buffer.

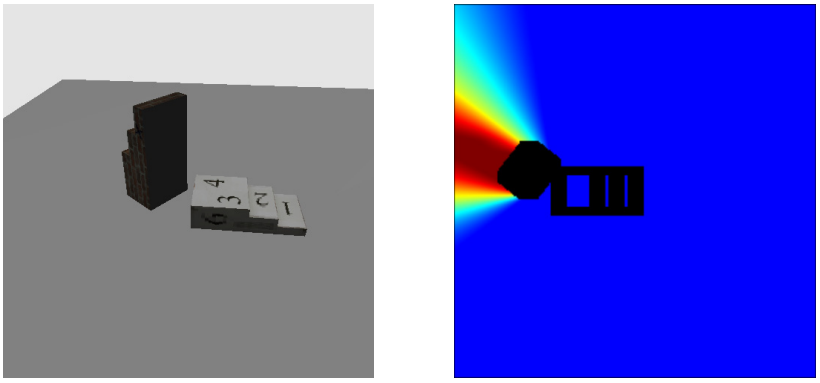


Figure 8.17: Scenario used to evaluate occlusion-aware planning. The textured white stairs are to be kept in-view and unoccluded, with the brick-textured stairs acting as an occluder (*left*). The occlusion cost map corresponding to this scenario (*right*).

Another important implementation aspect is the use of updated heuristics during the planning phase, in addition to the costmaps outlined above. The original footstep planner [10] uses a simple mobile-robot planner planning backwards from the goal to compute the heuristic used for subsequent planning. If this heuristic is used without modification, the planning process will be biased heavily to seek short paths to the goal, without any regard to occlusion, resulting in a planner that takes a long time to compute appropriate paths satisfying the occlusion requirements because its heuristic biases search towards short paths which may often violate the occlusion criteria. To ameliorate this, a modified heuristic is used, which queries the occlusion cost map while planning backwards from the goal, in addition to the heightmap that describes whether a particular location is part of an obstacle or free-space. Note that it remains efficient due to its overall coarseness in resolution and due to the fact that orientation is ignored when constructing heuristics. This latter fact also means that visibility is not explicitly considered when constructing the heuristic. A sample heuristic used in the experiments with HRP-2 is visualized in Figure 8.16. Note its similarity to the actual occlusion cost map for this scenario, as shown in Figure 8.17 (*right*).

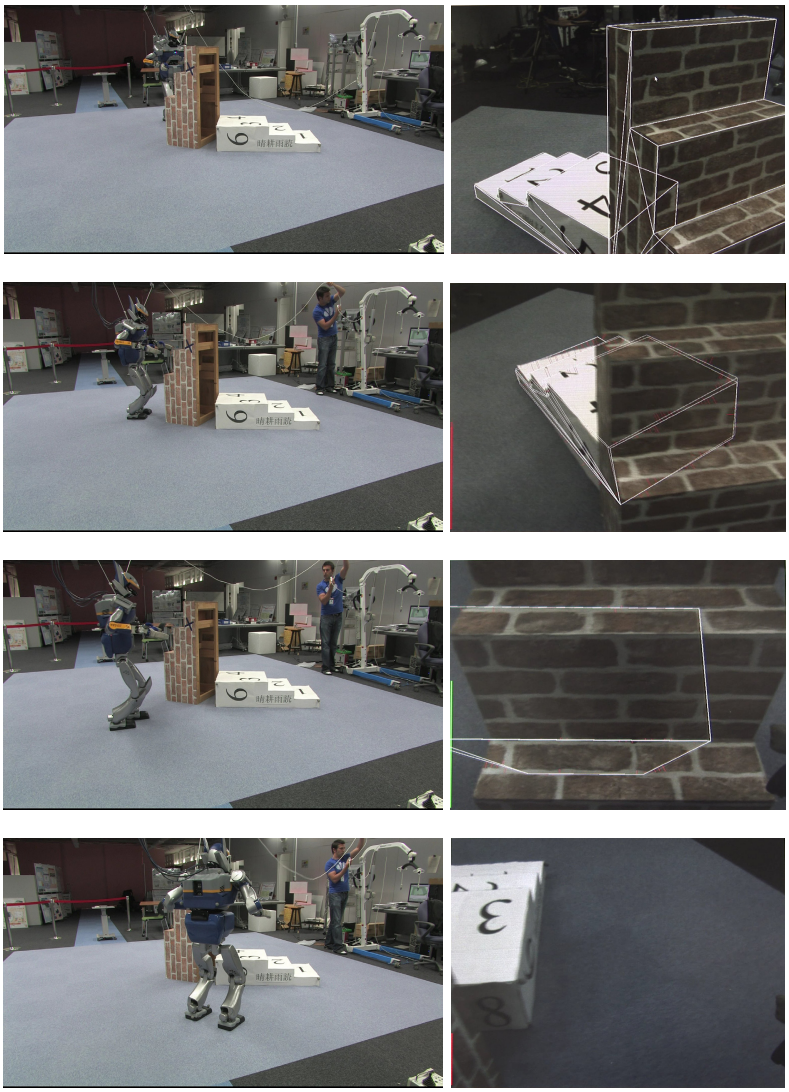


Figure 8.18: The HRP-2 humanoid navigating around an occluder with a path resulting from occlusion-unaware planning. External view (*left column*) and tracker view (*right column*). Tracker loses 'landmark' object (white stairs) shortly after the second frame from the top and does not recover.

8.6 RESULTS: PLANNING TO AVOID OCCLUSION

We evaluate our approach to planning to avoid occlusion during navigation using the following scenario: HRP-2 was asked to navigate to a target location while keeping a set of stairs in-view and free from occlusion. A second set of stairs placed end-on onto the floor (and hence having significant height) served as the occluder. Figure 8.17 shows the experimental setup and the cor-

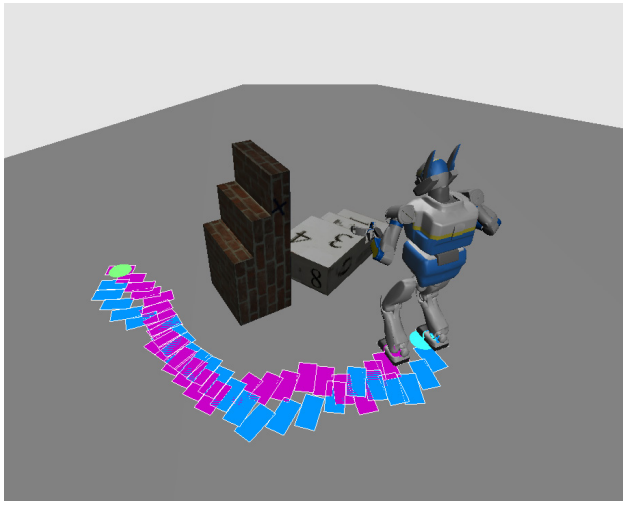


Figure 8.19: Path resulting from planning without future perceptive capability. HRP-2 takes a more direct path toward the goal, resulting in complete occlusion of the object to be tracked.

responding generated occlusion cost map. The GPU-accelerated tracker combined with the visual servoing behavior outlined above is used for perception, with the footstep planner now planning walking paths satisfying the occlusion and visibility criteria. HRP-2's usual walking controller was used once again to actually execute the motions.

The robot was made to navigate from a start location to a goal location located opposite it from the two objects during 3 experiments. This particular setup was chosen since the shortest path to the goal would lead the robot behind the stairs placed end-on on the floor, resulting in large occlusion of the set of 'landmark' stairs. Figure 8.18 shows what happens when occlusion is not reasoned about during planning, with Figure 8.19 showing the corresponding planned path. As expected, the planner chooses the short path to the goal, leading the robot behind the occluder (*left column*). Note how occlusion leads the tracker to lose track very quickly (*right column*). Occlusion of the target object soon becomes total, with no hope of the tracker being able to keep tracking the object, even if the robot keeps the target object in-view (but occluded!) throughout the motion (planning takes visibility into account and the visibility criteria remain valid throughout the path). The path shown here is comprised of 47 steps and was found after 18.90 seconds of planning. The overall occlusion cost over the path is 192.83, yielding a per-step cost of

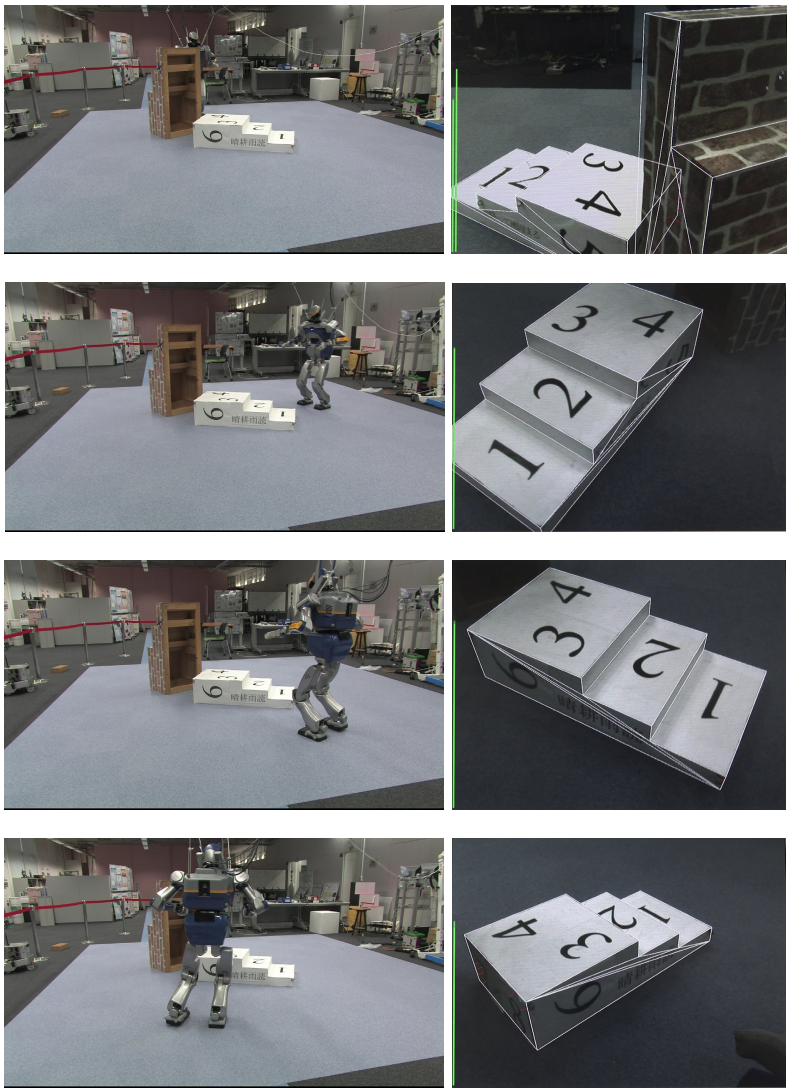


Figure 8.20: The HRP-2 humanoid navigating around an occluder with a path resulting from occlusion-aware planning. External view (*left column*) and tracker view (*right column*). Tracker continues to keep the landmark object (white stairs) in view and tracks them successfully throughout the motion.

4.10. All occlusion costs were scaled to lie in a range between 0 (0% occlusion) and 10 (100% occlusion).

When planning in an occlusion-aware manner, the resulting path is markedly different, leading the robot around the long way around the stairs to the goal in the other direction, as shown in Figure 8.20, with the resulting path illustrated in Figure 8.21. While the resulting path is now significantly longer, the robot's view of the target object remains unoccluded throughout, en-

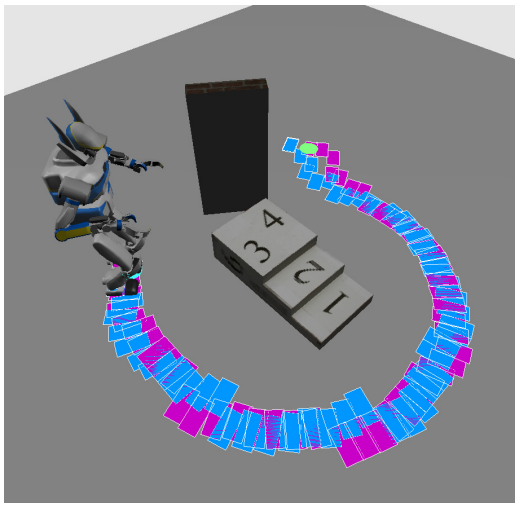


Figure 8.21: Path resulting from occlusion-aware planning. HRP-2 takes a path avoiding the occluder, resulting in an unoccluded view of the target object throughout.

suring that the white stairs are perfectly tracked throughout the entirety of the motion, as evidenced in the right column of Figure 8.20. This path, composed of 91 steps, was found after 33.89 seconds of planning. The overall occlusion cost over the path is 2.06, yielding a per-step occlusion cost of 0.02. Note that this is drastically lower than the costs for the path resulting from occlusion-unaware planning. The occlusion costs during the walking motion for perception-unaware planning and for planning with future perceptive capability are plotted for comparison in Figure 8.22. Note the complete occlusion of the target object by the end-on textured stairs in the perception-unaware condition. Also note how in both cases, the occlusion cost in the latter part of the path is low, as the target object appears unoccluded in both conditions towards the end of the path.

8.7 REPLANNING WITH VISIBILITY

The scenarios described thus far have involved HRP-2 planning navigation paths in a perception-aware manner for environments that remained static during the walking sequence. That is, while the robot's perception system is used to initially localize the robot and generate environment maps and despite perception continuing to track objects of interest and generating updates to said location and maps during the walking sequence, no updated information is used by the planner. Once the initial plan is

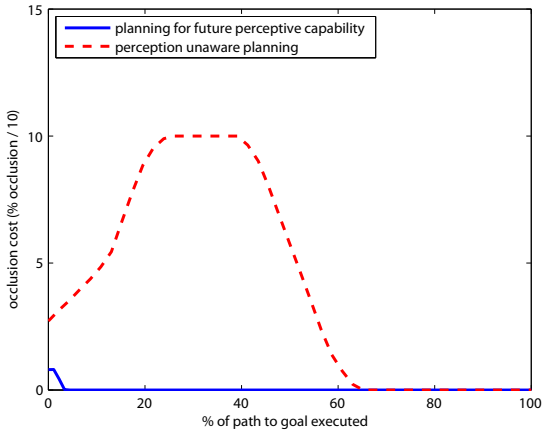


Figure 8.22: Plots of occlusion cost for footstep paths around an occluding obstacle without (*red / dashed*) and with perceptive capability (*blue / solid*). Note the large amount of occlusion (reaching 100%) in the perception-unaware case as the robot steps behind the occluder.

generated, it is not modified in any manner, but simply executed. Several issues arise with this approach. First, even if the initial robot location recovered from perception is highly accurate, the subsequent robot location, updated using dead reckoning from the robot kinematics, is very prone to accumulation of error due to drift. In the case of HRP-2, when comparing the robot's reported foot position generated from perception and kinematics with 'ground-truth' foot positioning recovered using motion capture markers attached to the foot, a positioning error of 10–15cm after around 20 steps is not uncommon. This, however, can have disastrous results in situations, such as the stair climbing scenarios outlined above, where precise foot position recovery is paramount. Secondly, by ignoring perception data throughout the walking sequence, we often tend to miss out on opportunities to perform more accurate localization and mapping than is possible at the beginning of the walking sequence. For instance, when the robot approaches a set of stairs from a distance, the size of the stairs in the camera view will increase as the robot approaches, leading to less pixel error, less jitter and more accurate tracking, and hence better localization and mapping. Perception data gathered during the walking sequence *should* be used.

We sought to introduce the ability for HRP-2 to perform replanning using the perception data acquired by our GPU-accelerated tracker, much like the case in the approaches to perception out-

lined in Chapter 4 and Chapter 5. Furthermore, we would like to perform replanning in a manner that takes the robot's future perceptive capability into account. In the experiments described here, the planner gathers updated robot localization information from the tracker during each replanning cycle and combines it with the robot kinematics to establish the robot foot position, as described previously in Section 6.10. This is done once per step-cycle (which takes around 700ms in the case of HRP-2), when the robot's stance foot changes (i.e. each time a foot touches the ground during walking). It is at this point that tracking error due to robot sway is generally minimized. The planner then plans for a large part of the remainder of the step cycle (around 500ms), with the best partial plan resulting in that time being chosen and subsequent foot trajectories being sent to the controller according to that plan.

The crucial issue of timing, already alluded to in the ASIMO case, becomes even more important here. A non-trivial amount of temporal discrepancy exists between information gathered from perception and from the robot kinematics. The latter process is computationally cheap and takes merely a few milliseconds, while gathering tracking data requires fetching frames from a camera, performing the tracking and pose recovery process itself, generating mapping and localization information from the recovered pose and sending the result over the network to the planner. In other words, considering the true state of the world at time t, the robot kinematics corresponding to that state of the world will be available to the planner at time $t + \Delta t_{kin}$, where Δt_{kin} is on the order of 1–5ms. Tracking data corresponding to the same state of the world will only be available at time $t + \Delta t_{vis}$, where Δt_{vis} is on the order of a few hundred milliseconds. Clearly, since the planner chains the tracker transform with the robot kinematics, timing discrepancies will result in significant errors in foot positioning.

To ameliorate this, the planner keeps a history buffer of camera-to-foot transforms recovered using kinematics. We then establish an offset into this buffer indicating which past transform should be combined with the pose recovered by the tracker by matching peaks and troughs from the x and y translation components of both transforms, resulting from the robot sway during walking. In practice, we have found an offset of around 200ms into the history buffer to work well for our experimental setup, yielding foot positions with at most 3cm of error throughout the walking sequence when compared to motion-capture ground truth.

Finally, the issue of when to stop acquiring newly updated localization / mapping data from perception during the walking sequence needs to be addressed. As mentioned previously, in a stair climbing scenario, the tracker *will* lose track of the stairs eventually as it begins to climb them. At this point, the recovered robot position is no longer valid and dead reckoning should instead be used from that point forward to continue to localize the robot. We use the tracker's built-in confidence measure to decide when to switch to dead-reckoning. If confidence drops beneath a certain threshold (indicating tracking failure is imminent), the tracker stops sending updated localization / mapping data to the planner.

8.8 RESULTS: REPLANNING WITH VISIBILITY

We evaluate replanning using future perceptive capability in two scenarios. For both, the GPU-accelerated tracker combined with the visual servoing behavior is used for perception, with the footstep planner now continuously planning paths satisfying target object visibility criteria as the robot walks. Note that occlusion-based reasoning is not used in this case, as computing occlusion costs (even with the hardware acceleration implemented) tends to be too expensive to yield meaningful partial paths in the 500ms or so the planner is given to plan between steps. The first scenario is a stair climbing task identical in nature to the stair climbing with visibility planning experiments outlined before in Section 8.4.2. Here, we sought to compare the effectiveness of continuously gathering perception data and replanning with the previous stairclimbing experiment.

During the second scenario, HRP-2 was tasked with tracking and following a small wheeled mobile robot around as it moved through the environment. The mobile robot, as tracked by HRP-2's onboard camera, is shown in Figure 8.23. During the experiments, it was moved wirelessly via a joystick by an experimenter. Here, the need for replanning is obvious, as the environment is constantly changing and the footstep plan must be continuously updated in order for HRP-2 to follow the mobile robot.

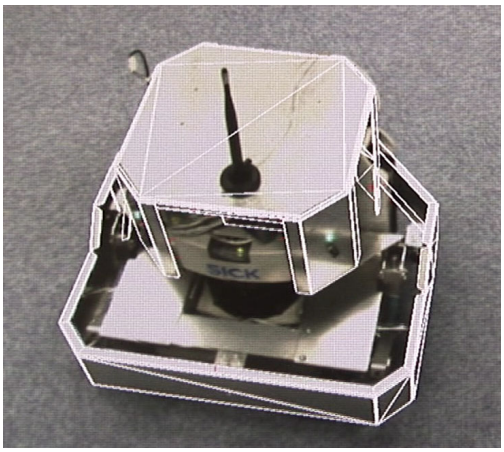


Figure 8.23: The Penguin 2 mobile robot as tracked using the GPU-accelerated tracker from HRP-2's head-mounted camera. The 3D robot model used by tracker is a reduced version of the full-detail model used to design and machine the robot parts.

8.8.1 *Stair Climbing*

Figure 8.24 (*left column*) shows an example stair climbing sequence planned with consideration for visibility, along with the corresponding tracker view (*right column*). Note how the overall appearance of the executed path is similar to the previous stair-climbing experiment shown in Figure 8.11, as expected. In reality, several differences are apparent. The robot initially planned a short partial path towards the stairs, approaching them up to the point where the head tilt limits meant that the stairs might move out of view. At that point, the visibility costs resulted in the subsequently replanned partial paths leading the robot in a circular arc towards the front of the stairs—maintaining them in view throughout—before beginning to climb them. Moreover, the ultimate foot location error once at the top of the stairs was significantly less (approx. 3cm) than in the non-replanning example (around 5–10cm), as the robot location was being updated until much further into the walking sequence.

8.8.2 *Robot Following*

In this experiment, HRP-2 has to navigate to a goal location at a fixed distance and orientation from the mobile robot, relative to which HRP-2 was localized by the tracker. Moving the mobile

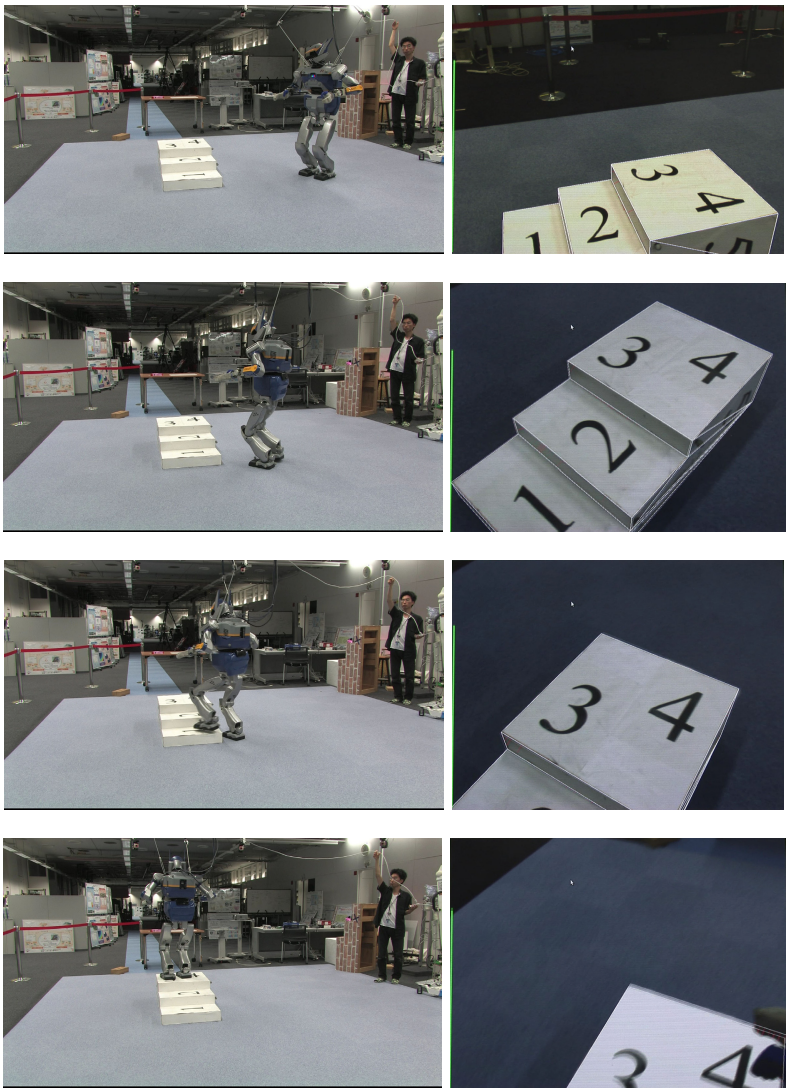


Figure 8.24: The HRP-2 humanoid climbing a set of stairs while maintaining them in view using replanning and continuously updated localization data. External view (*left column*) and tracker view (*right column*).

robot thus also results in a moving goal location, although in practice it is only the relative positioning between HRP-2 and the mobile robot that changes as both the map and the goal location in map coordinates (centered on the mobile robot) remain the same. With replanning, this effectively makes HRP-2 follow the mobile robot around continuously, with visibility planning ensuring that the mobile robot remains in-view and well trackable at all times. Figure 8.25 (*left column*) shows HRP-2 during one such robot-following sequence, along with the corresponding tracker

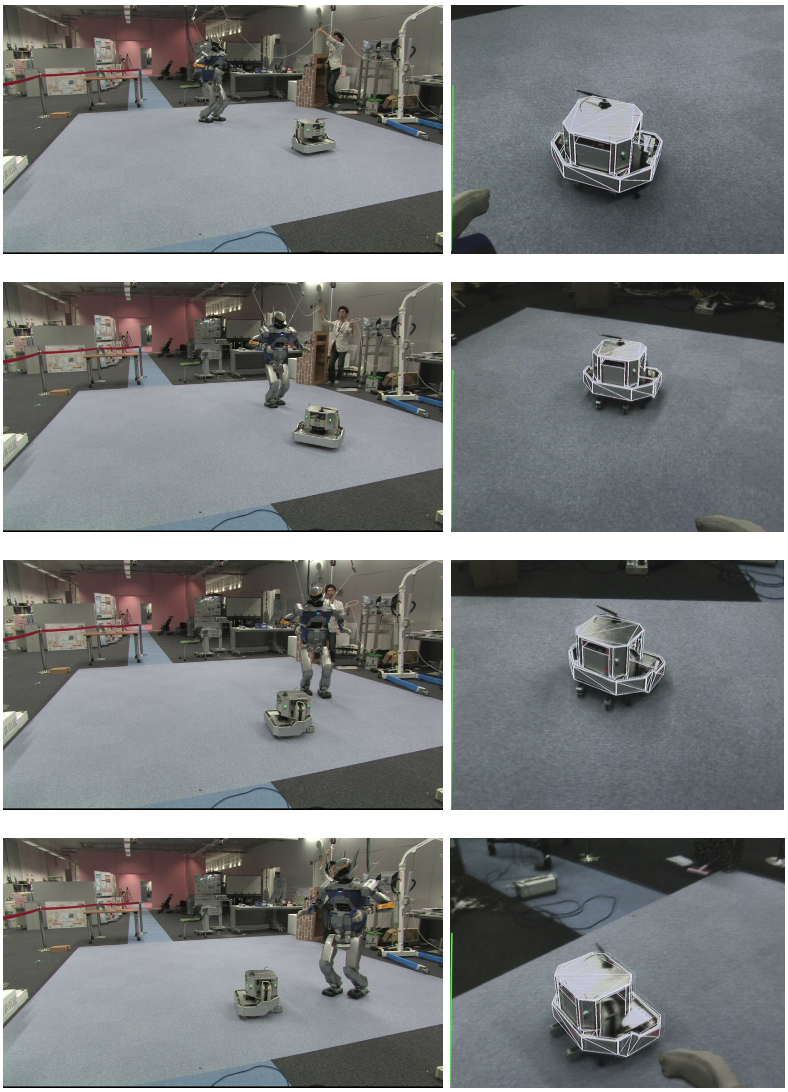


Figure 8.25: HRP-2 following a mobile robot using visibility-aware replanning and continuously updated localization data from perception. External view (*left column*) and tracker view (*right column*).

view (*right column*). Note how several factors contribute to make this quite a challenging scenario for the perception system. The 3D model of the object being tracked is relatively complex, with a large number of distinct edges, resulting in many control points to be used during the fitting stage. Also, the physical mobile robot is fairly small, occupying only a rather small part of the camera view even when HRP-2 stands very close to it. Finally, the robot is made from aluminum, resulting in many reflections and false edges arising from them. It is only the interplay of

efficient, robust tracking, the fact that the mobile robot always remains in-view due to visibility-aware planning and the fast visual servoing loop minimizing motion in the image that enable the mobile robot to be tracked reliably during this follow-me sequence.

8.9 SUMMARY

In this chapter, we detailed how a tighter coupling between perception and planning can be achieved in the context of planning autonomous navigation strategies for humanoid robots, by reasoning about future perceptive capability. Specifically, we enabled HRP-2 to reason about both *visibility* and *occlusion* as they affect the GPU-based 3D perception system used by the robot while planning footstep paths. Our results show that such planning is effective in ensuring that perception remains operational as the robot locomotes and therefore aids safety and reliability by increasing task success rates and minimizes localization & reconstruction error. Our visibility, occlusion and replanning experiments show that reasoning about perceptive capability during planning remains computationally feasible enough to be used for real-world robot tasks.

9

CONCLUSIONS

We now summarize the contributions of this thesis and point out areas of future work.

9.1 CONTRIBUTIONS

Figure 9.1 outlines the nature of the contributions of this thesis. The broad contribution consists of the exploration of a series of widely applicable perception strategies designed to allow humanoid robots to locomote and manipulate autonomously in complicated environments, as evidenced by our results from three distinct humanoid robot platforms. The robot tasks achieved by coupling these perception strategies with efficient planning methods, such as navigation among challenging dynamic obstacles on ASIMO or the stair-climbing and robot-following scenarios on HRP-2, represent some of the current state of the art in humanoid autonomy.

The most sophisticated of the perception approaches presented—the GPU-accelerated 3D tracking system described in Chapter 6—constitutes one of the most advanced model-based monocular trackers available today, offering a speed, robustness and flexibility that make it applicable to a wide range of perception tasks in robotics and vision applications in general. It functions as fully integrated perception system for humanoid robots. It also serves as a prime example of general purpose use of GPU processing to significantly accelerate a computationally intense problem outside of graphics.

This thesis also introduces and thoroughly investigates a framework for reasoning, during the planning stage, about a robot's ability to successfully perform perception when autonomously executing a task. Motivated by the particular challenges of sensing on humanoid platforms, the tighter coupling between perception and planning achieved by reasoning about future perceptive capability provides significant benefit in terms of execution reliability and robustness to a range of desirable humanoid tasks in the areas of navigation and manipulation.

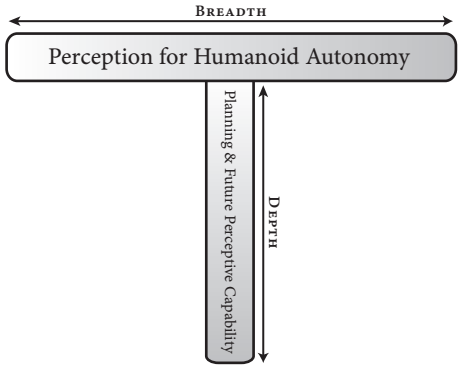


Figure 9.1: Thesis contributions. *Breadth:* A series of perception strategies tailored to humanoid robots coupled with efficient planning methods enabling real-world task autonomy. *Depth:* Investigating the framework of future perceptive capability on humanoids to reason about perception during planning for improved task execution and reliability.

Figure 9.2 visually places the work described in each chapter of this thesis within the context of the sense-plan-execute cycle as implemented on a humanoid robot.

9.2 OUTLOOK

Continued work in integrating perception and planning in the context of humanoid robotics is surely to lead us closer to our vision of capable humanoids able to fulfill meaningful tasks autonomously in the same environments we inhabit. Several avenues of future work related to this thesis come to mind.

GPU computation has been presented as an enabling technology throughout this thesis, being exploited both for perception and motion planning purposes. The increasing viability of GPUs as a general purpose computation platform and big strides in improved ease-of-programming for GPUs mean that new application areas throughout the perceive-plan-execute cycle abound. Fast, physically accurate forward-simulation of the environment, calculations performed during ZMP-based control, high-resolution stereo disparity computations or novel parallelizable or probabilistic motion planning algorithms are just a few areas that might benefit significantly from GPU acceleration.

As touched upon in Section 7.8, the incorporation of perceptive capability into the planning stage presents many opportunities for further investigation. Extending the idea to other

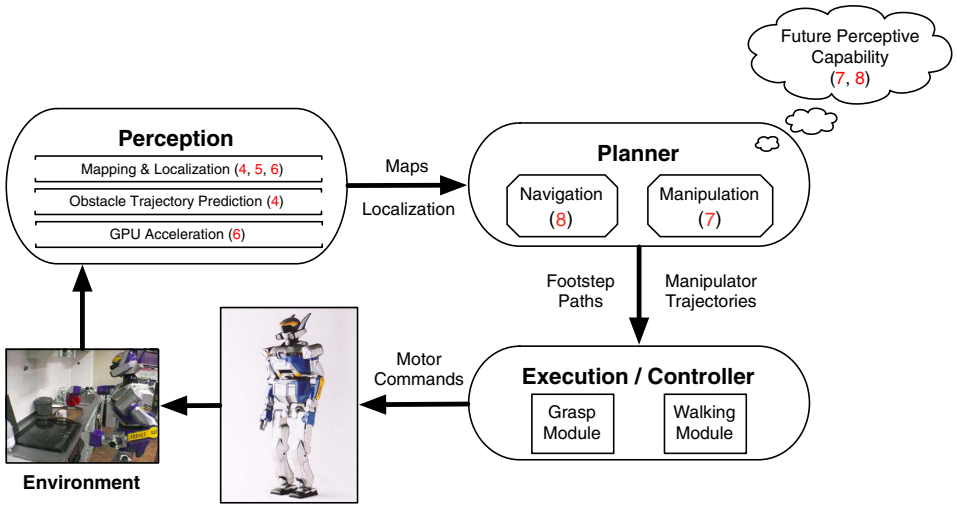


Figure 9.2: Placement of thesis work within the perceive-plan-execute architecture. Red numbers in parentheses indicate chapters providing detail.

kinds of sensors immediately comes to mind. For instance, it is straightforward to envision simulating a laser range-finder to compute perceptive capability (here, distance of objects to the sensor, sweeping speed of the laser rig or even reflective properties of the environment might be used to compute the measure). More realistic, probabilistic sensor models incorporating noise and error components explicitly, the incorporation of multiple sensors and novel application domains of planning for future perceptive capability in robotics could all be areas of continued focus. It would also be very desirable to be able to discover viable sensor models and appropriate heuristic proxies to the 'true' perceptive capability of the sensing system in a particular environment automatically through learning. Perhaps a history of sensed environment configurations gathered during past operation together with assessments of task success rates or even explicitly logged perceptive capability (via recorded perception error or failures) during past operation could be used to build such sensor models automatically. These could then be continuously improved during robot operation.

Recent advances in sensor development mean that an increasing range of accurate sensors compact and power-efficient enough to be mounted on a humanoid now exist. High resolution monocular cameras and stereo rigs, as well as novel laser scanners all remain applicable to different perception scenarios (offering different tradeoffs between speed and accuracy) likely to

be encountered by humanoid robots. Yet, work on employing multiple such sensors and appropriately fusing their measurements during operation on a walking humanoid, as touched upon in Chapter 5, remains limited and constitutes a further direction of research.

Reasoning about future world state during planning was shown in Chapter 4 to allow for navigation among fast-moving, albeit planar obstacles. In this scenario, reactiveness of the entire perceive-plan-execute cycle as well as tight temporal synchronization are absolutely paramount. When using on-body perception on a fast-moving humanoid, reasoning predictively in environments with full spatial geometry and many obstacles or planning with many degrees of freedom—in manipulation scenarios, for instance—these issues are further exacerbated, as touched upon by our replanning experiments in Chapter 8. Further work in accurately and automatically synchronizing perception data with robot joint & force sensor data, predicted environment configurations and planned motions must be done to tackle truly dynamic real-world environments.

BIBLIOGRAPHY

[1] M. Armstrong and A. Zisserman. Robust object tracking. In *Proc. Asian Conference on Computer Vision*, pages 58–61, 1995. (Cited on page 13.)

[2] R. Azuma. A survey of augmented reality. *Presence*, 6(4): 355–385, 1997. (Cited on page 61.)

[3] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7822-4. (Cited on page 55.)

[4] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner. Grasp planning in complex scenes. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids07)*, December 2007. (Cited on page 75.)

[5] D. Bertram, J. J. Kuffner, R. Dillmann, and T. Asfour. An integrated approach to inverse kinematics and path planning for redundant manipulators. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'06)*, pages 1874–1879, May 2006. (Cited on page 71.)

[6] G. Blaskó and P. Fua. Real-time 3d object recognition for automatic tracker initialization. In *ISAR '01: Proceedings of the IEEE and ACM International Symposium on Augmented Reality (ISAR'01)*, Washington, DC, USA, 2001. IEEE Computer Society. (Cited on page 53.)

[7] F. Bourgault, A. Makarenko, S. Williams, B. Grocholsky, and H. Durrant-Whyte. Information-based adaptive robotic exploration. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS'02)*, pages 540–545, Lausanne, Switzerland, 2002. (Cited on page 13.)

[8] James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS2000)*, Japan, October 2000. (Cited on page 17.)

[9] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, November 1986. (Cited on page 32.)

[10] J. Chestnutt. *Navigation Planning for Legged Robots*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2007. (Cited on pages 89 and 107.)

[11] J. Chestnutt, J.J. Kuffner, K. Nishiwaki, and S. Kagami. Planning biped navigation strategies in complex environments. In *Proc. of the IEEE-RAS/RSJ Int. Conf. on Humanoid Robots (Humanoids'03)*, Munich, Germany, October 2003. (Cited on pages 11 and 19.)

[12] J. Chestnutt, P. Michel, K. Nishiwaki, J.J. Kuffner, and S. Kagami. An intelligent joystick for biped control. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, May 2006. (Cited on page 27.)

[13] A. I. Comport, E. Marchand, and F. Chaumette. A real-time tracker for markerless augmented reality. In *ACM/IEEE Int. Symp. on Mixed and Augmented Reality, ISMAR'03*, pages 36–45, Tokyo, Japan, October 2003. (Cited on page 13.)

[14] R. Cupec, O. Lorch, and G. Schmidt. Vision-guided humanoid walking - concepts and experiments. In *Proc. of the 12th Int. Workshop on Robotics in Alpe-Adria-Danube Region (RAAD'03)*, Cassino, Italy, May 2003. (Cited on pages 11 and 12.)

[15] D. DeMenthon and L. Davis. Model-based object pose in 25 lines of code. In *ECCV '92: Proceedings of the Second European Conference on Computer Vision*, pages 335–343, London, UK, 1992. Springer-Verlag. ISBN 3-540-55426-2. (Cited on page 57.)

[16] T. Drummond and R. Cipolla. Real-time tracking of complex structures with on-line camera calibration. In *Proc. of the British Machine Vision Conference (BMVC'99)*, Nottingham, UK, September 1999. (Cited on pages 32 and 48.)

[17] Tom Drummond and Roberto Cipolla. Real-time tracking of multiple articulated structures in multiple views. In *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part II*, pages 20–36, London, UK, 2000. Springer-Verlag. ISBN 3-540-67686-4. (Cited on page 13.)

[18] Tom Drummond and Roberto Cipolla. Real-time visual tracking of complex structures. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):932–946, 2002. (Cited on page 13.)

[19] J. Fung and S. Mann. Openvidia: parallel gpu computer vision. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 849–852, 2005. ISBN 1-59593-044-2. (Cited on page 13.)

[20] J. Gancet and S. Lacroix. Pg2p: A perception-guided path planning approach for long range autonomous navigation in unkown natural environments. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'03)*, Las Vegas, NV, November 2003. (Cited on page 14.)

[21] R. Gayle, P. Segars, M. Lin, and D. Manocha. Path planning for deformable robots in complex environments. In *Proceedings of Robotics: Science and Systems*, pages 225–232, 2005. (Cited on page 13.)

[22] D. B. Gennery. Visual tracking of known three-dimensional objects. *Int. J. Comput. Vision*, 7(3):243–270, 1992. ISSN 0920-5691. (Cited on page 12.)

[23] N. Govindaraju, S. Redon, M. Lin, and D. Manocha. CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In *Graphics Hardware*, pages 25–32, 2003. (Cited on pages 13 and 78.)

[24] J.-S. Gutmann, M. Fukuchi, and M. Fujita. Stair climbing for humanoid robots using stereo vision. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'04)*, pages 1407–1413, Sendai, Japan, September 2004. (Cited on page 12.)

[25] J.-S. Gutmann, M. Fukuchi, and M. Fujita. A floor and obstacle height map for 3D navigation of a humanoid robot. In *Proc. of the Int. Conf. on Robotics and Automation (ICRA'05)*, Barcelona, Spain, April 2005. (Cited on pages 11 and 12.)

[26] J.-S. Gutmann, M. Fukuchi, and M. Fujita. Real-time path planning for humanoid robot navigation. In *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI'05)*, pages 1232–1237, Edinburgh, UK, July 2005. (Cited on page 11.)

[27] C. Harris. Tracking with rigid models. *Active vision*, pages 59–73, 1993. (Cited on page 13.)

[28] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka. The development of Honda humanoid robot. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'98)*, pages 1321–1326, May 1998. (Cited on pages 11 and 12.)

[29] S. A. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Trans. Robotics and Automation*, 12(5):651–670, Oct 1996. (Cited on page 14.)

[30] F. Jurie and M. Dhome. Real time tracking of 3d objects : an efficient and robust approach. *Pattern Recognition*, 35(2): 317–328, 2002. (Cited on page 13.)

[31] S. Kagami, K. Nishiwaki, J.J. Kuffner, K. Okada, M. Inaba, and H. Inoue. Vision-based 2.5D terrain modeling for humanoid locomotion. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, pages 2141–2146, September 2003. (Cited on page 11.)

[32] S. Kagami, Y. Takaoka, Y. Kida, K. Nishiwaki, and T. Kanade. Online dense local 3D world reconstruction from stereo image sequences. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'05)*, pages 2999–3004, August 2005. (Cited on page 12.)

[33] N. Kalra, D. Ferguson, and A. Stentz. Constrained exploration for studies in multirobot coordination. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'06)*, pages 4300–4302, May 2006. (Cited on page 13.)

[34] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Proceedings of the 2nd International Workshop on Augmented Reality (IWAR '99)*, San Francisco, USA, October 1999. (Cited on page 61.)

[35] G. Klein. *Visual Tracking for Augmented Reality*. PhD thesis, University of Cambridge, 2006. (Cited on pages 37 and 38.)

[36] J.J. Kuffner and S.M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'2000)*, pages 995–1001, San Francisco, CA, Apr 2000. (Cited on page 75.)

[37] J.J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep planning among obstacles for biped robots. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'01)*, pages 500–505, 2001. (Cited on page 11.)

[38] J.J. Kuffner, K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue. Online footstep planning for humanoid robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, 2003. (Cited on page 11.)

[39] V. Kyrki and D. Kragic. Integration of model-based and model-free cues for visual object tracking in 3d. *Proc. of the IEEE Int. Conf on Robotics and Automation (ICRA'05)*, pages 1554–1560, April 2005. (Cited on page 13.)

[40] V. Kyrki, D. Kragic, and H. I. Christensen. Measurement errors in visual servoing. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'04)*, pages 1861–1867, New Orleans, LA, April 2004. (Cited on page 14.)

[41] S. LaValle, H. H. González-Baños, C. Becker, and J. Latombe. Motion strategies for maintaining visibility of a moving target. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'97)*, pages 731–736, 1997. (Cited on page 13.)

[42] S. M. LaValle. *Planning Algorithms*. Cambridge University Press (also available at http://msl.cs.uiuc.edu/planning/), 2006. (Cited on page 5.)

[43] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5): 378–400, May 2001. (Cited on pages 70 and 71.)

[44] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *IEEE Trans. Pattern Anal. Mach. Intell (PAMI).*, 28 (9):1465–1479, 2006. ISSN 0162-8828. (Cited on page 53.)

[45] V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects. *Found. Trends. Comput. Graph. Vis.*, 1(1):1–89, 2006. ISSN 1572-2740. (Cited on page 12.)

[46] V. Lepetit, J. Pilet, and P. Fua. Point matching as a classification problem for fast and robust object pose estimation. *Proc. Conf. on Computer Vision and Pattern Recognition (CVPR'02)*, 02:244–250, 2004. ISSN 1063-6919. (Cited on page 53.)

[47] O. Lorch, J. Denk, J. F. Seara, M. Buss, F. Freyberger, and G. Schmidt. ViGWaM - an emulation environment for a vision guided virtual walking machine. In *Proc. of the IEEE Int. Conf. on Humanoid Robotics (Humanoids 2000)*, 2000. (Cited on pages 11 and 12.)

[48] D. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, 2004. ISSN 0920-5691. (Cited on pages 53 and 54.)

[49] D. G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(5):441–450, 1991. (Cited on page 12.)

[50] David G. Lowe. Robust model-based motion tracking through the integration of search and estimation. *Int. J. Comput. Vision*, 8(2):113–122, 1992. ISSN 0920-5691. (Cited on page 12.)

[51] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. Cg: a system for programming graphics hardware in a c-like language. *ACM Trans. Graph.*, 22(3):896–907, 2003. ISSN 0730-0301. (Cited on page 41.)

[52] P. Michel, J. Chestnutt, S. Kagami, K. Nishiwaki, J. Kuffner, and T. Kanade. Gpu-accelerated real-time 3d tracking for humanoid locomotion and stair climbing. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS'07). In Review.*, San Diego, CA, USA, Oct–Nov 2007. (Cited on page 30.)

[53] P. Michel, C. Scheurer, J. Kuffner, N. Vahrenkamp, and R. Dillmann. Planning for robust execution of humanoid motions using future perceptive capability. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS'07). In Review.*, San Diego, CA, USA, Oct–Nov 2007. (Cited on page 67.)

[54] Philipp Michel, Joel Chestnutt, James Kuffner, and Takeo Kanade. Vision-guided humanoid footstep planning for dynamic environments. In *Proc. of the IEEE-RAS/RSJ Int. Conf. on Humanoid Robots (Humanoids'05)*, pages 13–18, December 2005. (Cited on pages 11, 15, and 19.)

[55] Philipp Michel, Joel Chestnutt, Satoshi Kagami, Koichi Nishiwaki, James Kuffner, and Takeo Kanade. Online environment reconstruction for biped navigation. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, Orlando, FL, USA, May 2006. (Cited on page 23.)

[56] Motion Analysis Corporation. Motion Analysis Eagle Digital System. Web: `http://www.motionanalysis.com`. (Cited on page 48.)

[57] Bart Nabbe. *Extending the Path-planning Horizon*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2005. (Cited on page 14.)

[58] K. Nagasaka, M. Inaba, and H. Inoue. Walking pattern generation for a humanoid robot based on optimal gradient method. In *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybernetics*, 1999. (Cited on page 11.)

[59] K. Nishiwaki, T. Sugihara, S. Kagami, M. Inaba, and H. Inoue. Online mixture and connection of basic motions for humanoid walking control by footprint specification. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'01)*, Seoul, Korea, May 2001. (Cited on page 11.)

[60] K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue. Online generation of humanoid walking motion based on a fast generation method of motion pattern that follows desired ZMP. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'02)*, pages 96–101, 2002. (Cited on page 11.)

[61] K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue. Toe joints that enhance bipedal and fullbody motion of humanoid robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'02)*, pages 3105–3110, Washington, DC, USA, May 2002. (Cited on page 12.)

[62] K. Nishiwaki, S.Kagami, J.J. Kuffner, K. Okada, Y. Kuniyoshi, M. Inaba, and H. Inoue. Online humanoid locomotion control using 3D vision information. In *Proc. of the IEEE/RSJ Int. Symposium on Experimental Robotics (ISER'02)*, 99 2002. (Cited on page 12.)

[63] NVIDIA. NVIDIA occlusion query extension. Web: `http://oss.sgi.com/projects/ogl-sample/registry/NV/occlusion_query.txt`. (Cited on page 77.)

[64] Thierry Oggier, Michael Lehmann, Rolf Kaufmann, Matthias Schweizer, Michael Richter, Peter Metzler, Graham Lang, Felix Lustenberger, and Nicolas Blanc. An all-solid-state optical range camera for 3D real-time imaging with sub-centimeter depth resolution (SwissRanger). Available online at: `http://www.swissranger.ch`, 2004. (Cited on page 26.)

[65] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens KrÃ¼ger, Aaron E. Lefohn, and Timothy J. Purcell. A survey of general-purpose computation on graphics

hardware. In *Eurographics 2005, State of the Art Reports*, pages 21–51, Aug 2005. (Cited on page 13.)

[66] A. E. Patla. How is human gait controlled by vision? *Ecological Psychology*, 10:287–302, 1998. (Cited on page 11.)

[67] A.E. Patla, E. Niechwiej, and L. Santos. Local path planning during human locomotion over irregular terrain. In *Proc. of the Int. Symposium on Adaptive Motion in Animals and Machines AMAM2000*, 2000. (Cited on page 11.)

[68] M. Pharr and R. Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, March 2005. ISBN 0321335597. (Cited on page 13.)

[69] R. Pito. A sensor based solution to the next best view problem. In *IAPR Int. Conf. Pattern Recognition, ICPR'96*, pages 941–945, Vienna, Austria, August 1996. (Cited on page 13.)

[70] J. Pratt and G. Pratt. Exploiting natural dynamics in the control of a 3D bipedal walking simulation. In *Proc. of the Int. Conf. on Climbing and Walking Robots (CLAWAR99)*, September 1999. (Cited on page 11.)

[71] P. Preisig and D. Kragic. Robust statistics for 3d object tracking. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'06)*, pages 2403–2408, Orlando, FL, USA, May 2006. (Cited on page 34.)

[72] M. K. Reed and P. K. Allen. Constraint-based sensor planning for scene modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1460–1467, 2000. (Cited on page 13.)

[73] N. Roy and S. Thrun. Coastal navigation with mobile robots. In *Advances in Neural Processing Systems 12*, volume 12, pages 1043–1049, 1999. (Cited on page 13.)

[74] K. Sabe, M. Fukuchi, J.-S. Gutmann, T. Ohashi, K. Kawamoto, and T. Yoshigahara. Obstacle avoidance and path planning for humanoid robots using stereo vision. In *Proc. of the Int. Conf. on Robotics and Automation (ICRA'04)*, New Orleans, LA, April 2004. (Cited on pages 11 and 12.)

[75] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura. The intelligent ASIMO: System

overview and integration. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'02)*, pages 2478–2483, 2002. (Cited on page 1.)

[76] B. Salomon, N. Govindaraju, A. Sud, R. Gayle, M. Lin, and D. Manocha. Accelerating line of sight computations using graphics processing units. In *Proceedings of the 24th Army Science Conference*, 2004. (Cited on page 13.)

[77] H. Sutanto and R. Sharma. Practical motion planning with visual constraints. In *Proc. IEEE Int'l Symp. on Assembly and Task Planning*, pages 237–242, Marina del Rey, CA, August 1997. (Cited on page 14.)

[78] Y. Takaoka, Y. Kida, S. Kagami, H. Mizoguchi, and T. Kanade. 3D map building for a humanoid robot by using visual odometry. In *Proc. of the IEEE Int. Conf. on Systems, Man & Cybernetics (SMC'04)*, pages 4444–4449, October 2004. (Cited on page 12.)

[79] C. Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *Int. J. of Computer Vision*, 9(2):137–154, November 1992. (Cited on page 12.)

[80] L. Vacchetti, V. Lepetit, and P. Fua. Stable real-time 3d tracking using online and offline information. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(10):1385–1391, 2004. (Cited on page 13.)

[81] V. Varadarajan. *Lie Groups, Lie Algebras and Their Representations*. Number 102 in Graduate Texts in Mathematics. Springer-Verlag, 1974. (Cited on page 32.)

[82] Z. Wasik and A. Saffiotti. Robust color segmentation for the robocup domain. In *Proc. of the Int. Conf. on Pattern Recognition (ICPR'02)*, Quebec City, Quebec, CA, 2002. (Cited on page 17.)

[83] L. Weiss, A. C. Sanderson, and C. P. Neuman. Dynamic sensor-based control of robots with visual feedback. *IEEE Journal on Robotics and Automation*, RA-3(5), October 1987. (Cited on page 68.)

[84] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, 1995. (Cited on page 37.)

[85] C. Wu. SiftGPU. Web: http://cs.unc.edu/~ccwu/siftgpu/. (Cited on page 53.)

[86] M. Yagi and V. Lumelsky. Local on-line planning in biped robot locomotion amongst unknown obstacles. *Robotica*, 18 (4):389–402, 2000. ISSN 0263-5747. (Cited on page 11.)

[87] M. Yagi and V. J. Lumelsky. Biped robot locomotion in scenes with unknown obstacles. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'99)*, pages 375–380, Detroit, MI, May 1999. (Cited on page 11.)

[88] J. Yamaguchi, S. Inoue, D. Nishino, and A. Takanishi. Development of a bipedal humanoid robot having antagonistic driven joints and three DOF trunk. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'98)*, pages 96–101, 1998. (Cited on page 11.)

[89] Z. Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Proc. of the Int. Conf. on Computer Vision (ICCV '99)*, pages 666–673, Corfu, Greece, September 1999. (Cited on page 26.)