Perception-driven optimal motion planning under resource constraints

by

Thomas Sayre-McCord

Submitted to the Joint Program in Applied Ocean Science & Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

and the

WOODS HOLE OCEANOGRAPHIC INSTITUTION

February 2019

Author ..... Signature redacted .

Joint Program in Applied Ocean Science & Engineering
Massachusetts Institute of Technology
& Woods Hole Oceanographic Institution
October 12, 2018

Certified by ..... Signature redacted

Sertac Karaman
Associate Professor of Aeronautics and Astronautics
Massachusetts Institute of Technology
Thesis Supervisor

Accepted by ..... Signature redacted .....

Nicolas Hadjiconstantinou
Chairman, Committee for Graduate Students
Massachusetts Institute of Technology

Accepted by ..... Signature redacted .....

Henrik Schmidt
Chairman, Joint Committee for Applied Ocean Science & Engineering
Massachusetts Institute of Technology
Woods Hole Oceanographic Institution

Perception-driven optimal motion planning under resource constraints

by

Thomas Sayre-McCord

Submitted to the Joint Program in Applied Ocean Science & Engineering
Massachusetts Institute of Technology
& Woods Hole Oceanographic Institution
on October 12, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Over the past few years there has been a new wave of interest in fully autonomous robots operating in the real world, with applications from autonomous driving to search and rescue. These robots are expected to operate at high speeds in unknown, unstructured environments using only onboard sensing and computation, presenting significant challenges for high performance autonomous navigation.

To enable research in these challenging scenarios, the first part of this thesis focuses on the development of a custom high-performance research UAV capable of high speed autonomous flight using only vision and inertial sensors. This research platform was used to develop state-of-the-art onboard visual inertial state estimation at high speeds in challenging scenarios such as flying through window gaps. While this platform is capable of high performance state estimation and control, its capabilities in unknown environments are severely limited by the computational costs of running traditional vision-based mapping and motion planning algorithms on an embedded platform.

Motivated by these challenges, the second part of this thesis presents an algorithmic approach to the problem of motion planning in an unknown environment when the computational costs of mapping all available sensor data is prohibitively high. The algorithm is built around a tree of dynamically feasible and free space optimal trajectories to the goal state in configuration space. As the algorithm progresses it iteratively switches between processing new sensor data and locally updating the search tree. We show that the algorithm produces globally optimal motion plans, matching the optimal solution for the case with the full (unprocessed) sensor data, while only processing a subset of the data. The mapping and motion planning algorithm is demonstrated on a number of test systems, with a particular focus on a six-dimensional thrust limited model of a quadrotor.

Thesis Supervisor: Sertac Karaman
Title: Associate Professor of Aeronautics and Astronautics
Massachusetts Institute of Technology

Acknowledgments

I would like to express my appreciation to several key people who have supported and taught me over the past five years. First of all, I want to express my heartfelt thank you to my PhD advisor Sertac Karaman. He has been a constant source of inspiration and advice, and I feel very privileged to have gotten to work with him. Thank you for the freedom to work on what excited me for better or worse, and the knowledge, advice, and resources to make those ideas so much better. Your knowledge and energy across such a range of fields has been inspiring, and your ability to convey that knowledge on to me has been so valuable. The hours spent discussing topics across robotics, and how we could address them, were some of the most fun and educational parts of my PhD. I would also like to thank my Masters advisor and committee member Hanu Singh. Thank you for bringing me into the world of robotics; for teaching me so much about not only the theory of robotics but also the practice. Thank you for being a tireless advocate for me, for always having your door open, and for going well out of your way to help me however you can both professionally and personally. I am greatly indebted to John Leonard, who provided me my first "home" at MIT and never ceased being a source of support and advice for the rest of my PhD. Thank you for never once saying "no" to me, for always having the time no matter how busy you were, and for being able to say just the right things in our meetings to put me on the right path. Finally, I would like to thank Luca Carlone for the hours spent with me helping to debug and teaching me along the way. You taught me immeasurable amounts not only about robotics, but also about how to be a good engineer and a good scientist. I can not thank you enough for the skills you gave me and as a model of mentorship that I will look to for the rest of my life.

I was very privileged to be a part of three excellent research communities over the course of my PhD. I want to thank the Deep Submergence Lab for teaching so much about building and deploying robots, the Marine Robotics Group for the innumerable conversations over coffee and many late nights, and the AgileDrones group for the laughter through bad times (basement labs and broken drones) and good (anywhere with windows, but still broken drones). To all of you, thank you for the knowledge and the friendship, without all of you my PhD experience could never have been the same. At WHOI, I particularly want to thank Karen Schwamb and Judy Fenwick for always having the answers and the care for whatever I needed, Kevin Manganini for good times and great help getting the Jetyak running, and

John Bailey for his immense knowledge, help, stories, and general good times whether we are on land or at sea. At MIT, thank you to Jin for always helping with my many many requests, to Dehann and Pedro for your friendship from the very first day we started, and to Winter for making our research better and our time more fun from the beginnings of my time on the drone project.

Outside of MIT, I can't express enough my gratitude to my family and friends. I feel to have my parents, Happy and Geoff, who have always provided their endless support, advice, and encouragement. They have always believed in me, encouraged me, and have been ready to help at a moments notice. I also want to thank my brother, Rob, and sister-in-law Claire, for their advice, for being role models in how to approach life, and as a perpetual outlet for a fun weekend. The biggest thank you goes to Narges, who has been the constant source of all things good in my PhD. Thank you for the endless love, support, advice, and jokes (even the ones at my expense). You have made everything over the last four years amazing; providing the fun and the relaxation when I was not working, and the knowledge, advice, and support when I was. I would also like to express my appreciation to my friends across Boston, both inside MIT and out. Thank you for always being there for me, for the good memories, the good fun, and for making my time in Boston so wonderful. Thank you to all of the remarkable people in my life; I feel very fortunate to have all of you by my side.

The work reported in this dissertation was supported in part by NVIDIA, MIT Lincoln Laboratory, MIT-WHOI Joint Program, and JD.com. Their support is gratefully acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years there has been a huge increase in interest and development of fully autonomous robots in the real world. Unlike the robots that are currently prevalent in structured applications such as manufacturing and wherehouses, this new wave of robots are working in diverse unknown environments using only the sensors and computation available to them onboard. The potential use cases are numerous, including self-driving cars [83], consumer products[1], defense applications[2], search and rescue [102], site inspection [73], marine surveillance[3], and agriculture [17].

Many factors have combined to bring about these changes, but foremost among them has been new, more powerful embedded computers such as the NVIDIA TX modules[4], improved and lower cost sensors[5], and advances in algorithms particularly centered around the boom in learning based methods. Despite the increase in spending on research and development, very few truly autonomous robots are acting on their own in the real world, with notable exceptions such as the Skydio R1[6]. Many open problems remain in robustness and capabilities, particularly in bringing demanding algorithms to real time constraints and embedded platforms.

This thesis is primarily concerned with the problem of autonomous robots working in unknown cluttered environments with a high performance to computation ratio, meaning that the capabilities of the platform will largely be dominated by the available computation.

[1] https://www.skydio.com
[2] https://www.delftdynamics.nl/
[3] http://sailbuoy.no/
[4] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/
[5] https://velodynelidar.com/vlp-16.html
[6] https://www.skydio.com

This type of platform is perpetually on the forefront of development in robotics, as in almost all use cases there is an interest in higher performance and in cheaper computation platforms. The inherent limitations in the problem force a mix of adapting existing algorithms to work in resource constrained environments and looking for new, computationally aware solutions. This thesis focuses primarily in the second area: what problems can be solved by building algorithms that take a full-robot view of the navigation problem.

Classically the problem of robot navigation can be broken down into five "pillars": system (what is the robot), state estimation (where is the robot), mapping (where is everything else), motion planning (where should the robot go), and control (how does the robot get there). Traditionally these problems are treated as isolated and cascading processes – system determines sensing and actuation, state estimation informs mapping, mapping informs motion planning, and motion planning informs control. This breakdown allows for the isolation of specific sub-problems for rapid and deep development, however, it can fail to take advantage of the inherent coupling of the sub-problems. Various works have looked at bridging the gap between the pillars. The most notable and ubiquitous of these methods is Simultaneous Localization and Mapping (SLAM) [59, 58] merging state estimation and mapping into a single seamless process. Numerous other examples exist in literature, particularly in recent years, which mix motion planning and state estimation [9, 12, 78, 105, 20] and motion planning and control [70, 32].

A different design path is enabled by bringing multiple sub-modules together. For example, in a purely visual state estimation context, research would primarily be focused on questions such as "how can state estimation be robust in cases of few visual features," a combined approach allows for questions such as "how can I perform motion planning to never have few visual features." While the single topic approach has numerous benefits, both in the depth that can be taken in a problem, and in the general applicability (the same algorithm for a car, UAV, or phone), it leaves open gaps in integration that can give substantial increases in performance, and provide more insight into the full navigation problem.

This thesis primarily focuses on the problem of optimal motion planning in unknown environments, and particularly situations where the "mapping" phase of operation is a non-negligible cost. Two such example scenarios are described below.

Visual UAV navigation. For a vision based UAV, a common sensor modality due to the light weight and low power requirements of vision sensors, mapping is done using

either stereo matching across a pair of cameras [82] or structure from motion with a single camera [64]. Stereo matching tends to be the less computationally intensive process, yet it still runs at only 30 Hz for a single VGA (640x480) stereo pair on an NVIDIA TX1, a modern embedded processor [36]. With a required safety boundary for how far a UAV can travel between map updates, the speed of a UAV can be severely limited. For example a safety boundary of 10 cm traveled between "re-map" events requires that an autonomous UAV be limited to ~3 m/s, making such a system ineffective for time critical applications. This speed limit presents a dramatic decrease in UAV capability, as systems have been demonstrated that can perform state estimation [25], control [99], and motion planning [77, 87] at significantly higher rates. In this scenario, we have also accepted data from only a single stereo pair with its limited field of view. To perform omni-directional mapping, a set of six stereo pairs like those found on the Skydio R1 would be required, causing an even further slow down of flight. The underlying problem, to be addressed in this thesis, is that there is simply more data to process than computation to process it. The key element to increase performance is that the processing of stereo data is directly related to the amount of volume to map, so if areas can be marked as "unimportant" through motion planning that data may go unprocessed.

Intent prediction for self driving cars. One of the major challenges facing autonomous driving is the problem of intent prediction [93], *i.e.* determining the probabilistic future motion of other actors in the environment. It is on the resulting time varying "intent" map that motion planning must be performed, however, deploying the most advanced prediction detectors on every actor in the environment can be prohibitively expensive. Again, this presents a scenario where there is too much data (the full raw description of every actor in the environment) to process, forcing either slower or more conservative performance. By bringing motion planning into the process an increased awareness of what elements of the environment are important can be found, *i.e.* which elements of the environment warrant expensive intent prediction, and for which can cheap methods be used. A key element of this process is that it is not a single shot prediction, the result of one prediction process can inform the need for another.

The computational challenges inherent in mapping have long been recognized, and various approaches have been taken to mitigate them. The most obvious approach is to change sensor modalities to a less computationally intensive sensor such as a LIDAR; for example

Mohta *et. al.* [75] use visual data for state estimation but LIDAR data for mapping. Even by switching sensor types the authors are forced to map only in a locally dense area due to high computation costs. The other clear approach to reducing the costs of mapping is heuristic decisions made *a priori* about which areas, and to what level, mapping should occur. This can include mapping only local areas densely, or mapping only the areas in the general regions a robot hopes to move into. These techniques can be effective in reducing computation, and keep the mapping process relatively isolated, but the require a pre-determined limitation on accuracy and operation envelopes.

Other techniques have been developed that move further down the path to process data in a way that is less accurate as a traditional map, but still useful for motion planning. Methods include using a "pushbroom stereo" method to assume movement of a vehicle and generate a dense local map by only processing a single disparity level and integrating through time [5], "NanoMap" which forms a history of single pose local maps and checks candidate trajectories against the local maps rather than performing expensive map merger actions [23], and Tunable Stereo which uses a mixture of sparse depth estimation and meshing to give a fast and arbitrarily dense representation of the world [84]. While each of these can provide benefits to speed at some price of accuracy, they are still fundamentally purely mapping procedures and are agnostic to the goals of the system and the motion planning process. In contrast, and of particular interest to this work, Ghosh and Biswas [30] demonstrate that by using motion planning to drive mapping; in this case by performing local stereo matching during graph expansion in motion planning, significantly less mapping (stereo processing) can be performed.

Due to the computational challenges involved in mapping, and the inherent limitations in heuristic approaches, this thesis proposes a new way of looking at the robot navigation problem. Rather than a two step processes where sensor data is converted into a map, and then a motion plan is determined based on the map, the new process is the joint problem of moving from sensor data into a motion plan, skipping the unnecessary intermediate step of a full map. By viewing the problem jointly, a single optimization problem can be solved (the optimal motion planning problem) using methods that account for the computational costs of both motion planning and mapping. Viewed through the lens of the example problems described above, this means that stereo data need only be processed in the volumes of interest for the motion planner. A key point is that this is not a single step decision, as the results

of processing certain sensor data will inform what new data should be processed next. A similar framework can be used for the intent prediction problem – a single cheap detector can be used, and intent prediction can be iteratively refined as various agents become involved during the motion planning problem. Unlike reactive methods, the resulting problem is a motion planning one in the traditional sense, except that the internal process takes into account both the costs of mapping and motion planning. By taking a "map aware" approach to the motion planning problem, certain properties of optimal motion planning become apparent allowing for efficient motion planning search as the map is filled in.

The theory and algorithms described in this thesis for mapping and motion planning are widely applicable, however, in this thesis a particular focus is placed on aerial robotics. Aerial robotics provides an excellent test platform both due to the interest in its applications to time critical problems such as search and rescue, and because of the realistic constraints it provides. Aerial platforms are constrained in computation and power due to weight limitations, they provide tight real-time constraints due to the lack of a safe state (such as parking) in which they can stay until state estimation or motion planning algorithms complete, and they provide an almost arbitrarily complex and high dimensional system for planning and control.

As part of moving forward into these high speed, high computation aerial platforms, we found that the systems and development environments required for advancing new high performance algorithms did not exist. Part of the work for this thesis has been developing an end-to-end system for agile aerial vehicle development including hardware, electronics, software, visual state estimation, and a photorealistic development environment for rapid prototyping.

This thesis is organized as follows: Chapter 2 describes the UAV research platform, photorealistic development environment, and experiments with each, Chapter 3 lays out the joint mapping and motion planning optimization problem, an algorithm to solve the problem, and proofs of the optimality of the algorithm, Chapter 4 applies the mapping and motion planning algorithm to several test systems, particularly a coupled double integrator UAV model, and Chapter 5 describes properties of the algorithm and optimization problem, and discusses future work.

Chapter 2

UAV System and Onboard State Estimation

Unmanned Aerial Vehicle (UAV) systems have been demonstrated in many domains, ranging from agriculture to consumer utilization [17]. Although fully autonomous UAVs have long been available, their capabilities still do not match the operational speeds that can be easily achieved by minimally-trained UAV operators. Various algorithmic components, such as control, planning, or perception for agile flight have been demonstrated in isolation using off-the-shelf components, however, a system that integrates control and perception algorithms developed from the ground up has not yet been designed, developed, and demonstrated. An important opportunity that may help close the gap is the emergence of powerful embedded supercomputers that can process high-rate, high-resolution exteroceptive sensory data in an efficient manner. This sensor data is essential to enable situational awareness and accurate state estimation for closed-loop agile control at high speeds. Developing the algorithms required for fully autonomous high speed flight, however, is a major challenge due to the lack of two critical components: *(i)* powerful research platforms equipped with high-rate electronics and massively-parallel embedded computers and *(ii)* safe development environments that can help propel algorithm and software development.

In this chapter, we describe a powerful UAV system that is capable of high-speed agile navigation in GPS-denied environments using control closed on visual-inertial state estimation. For this purpose, we develop: *(i)* state-of-the-art mechanical and electronics hardware that integrate a powerful embedded supercomputer, the NVIDIA Jetson Tegra X1, with an

inertial measurement unit and a camera, centered around the design of a custom NVIDIA Jetson carrier board; *(ii)* a unique virtual-reality UAV development environment, which we call *FlightGoggles*, that allows us to simulate camera images photorealistically while the UAV is in flight, providing the integration of simulated visual data with real inertial measurements; *(iii)* state-of-the-art visual-inertial state estimation algorithms that give an accurate state estimate for closed-loop navigation in complex environments, such as through doors and windows, in a robust and repeatable manner. The development and integration of these three systems are the main contributions of the present chapter.

To the best of our knowledge, in this chapter we describe one of the most capable NVIDIA Jetson carrier boards designed specifically for the development of high-speed agile flight by integrating key peripherals such as high-resolution, high-rate cameras and precision inertial measurement units.

Furthermore, we present the idea of a "virtual-reality" UAV development environment. While using simulation systems for UAV development has attracted a vast amount of attention, especially very recently with the introduction of AirSim by Microsoft [95], our system utilizes a motion capture environment to photorealistically render camera images which are then fed back to the UAV for active decision making and closed-loop flight control. In this system, the physics are real, the inertial measurements are real, but all exteroceptive measurements are simulated photorealistically in *real time* with the help of powerful desktop GPUs. We emphasize that this system does *not* simulate physics and interoceptive measurements, and is designed for use with a live robot. Hence, this system can be used for applications involving complex physics, *e.g.*, when aerodynamic effects are dominant, and complex electromechanical effects dominate propulsion forces. Because only a single element of the system is simulated, this environment allows us to rapidly develop agile UAVs and move into field deployments in a safe and scalable manner.

Finally, we develop on-board visual-inertial navigation algorithms that integrate monocular camera images and inertial measurements to estimate the vehicle's state in real time on-board the drone for closed loop control. The visual-inertial odometry algorithm presented in this paper validates the idea of utilizing synthetic camera images generated in real time together with real inertial measurements in closed-loop flight. The image simulation system is used to develop the visual-inertial algorithms in a challenging scenario of flying through a window gap, which is subsequently verified in laboratory experiments with an on-board

camera.

Much of the work presented in this chapter was published in the International Robotics and Automation Conference in 2018 [91].

2.1 Related Work

UAV Systems. Multi-rotor UAVs for research in vision based algorithms have primarily used off-the-shelf flight platforms and/or autopilot systems that have been modified with additional sensors and computation power. The most popular off-the-shelf platforms have been the AscTec Hummingbird [41, 1], the AscTec Pelican [96], and the Parrot AR Drone [18]. A few custom platforms have also been built, typically augmented with an off-the-shelf autopilot board [68, 21, 19]. Due to being lightweight and low-power, the most popular sensor package for UAVs is a single camera (either forward or downward facing) and an IMU, although lightweight 2D laser scanners [1], stereo camera pairs [1, 97], and an RGB-D sensor [67, 68] have also been used. The recent DARPA Fast Lightweight Autonomy challenge[1] has brought an increase in attention and funding to high speed UAV flight in unknown environments. Mohta *et. al.* have built a combined visual (state estimation) and laser (mapping) platform demonstrating high speed flight in unknown environments [75]. The platform we present in this paper is built from the ground-up to house high-end inertial measurement units, high-rate high-resolution cameras, and state-of-the-art embedded CPU/GPU computing systems.

Synthetic Environments for Robotics. There has been a variety of work on the use of synthetic data sets and simulation in robotics and more generally computer vision. Synthetically generated data sets, such as those in [90, 29], have become of particular interest as the need for large labeled data sets for deep learning has become prevalent. Of particular note to the work presented here is the method of Richter *et. al.* in [89] which uses pre-built video games to generate semantically mapped synthetic data sets. Kavena *et. al.* use photorealistic renderings to evaluate the performance of different feature descriptors under a variety of camera conditions [45]. Handa *et. al.* provide a synthetic data set for the verification of SLAM algorithms against a known 3D model and trajectory [33]. In robotics, Gazebo [51] is the ubiquitous full simulation environment, with specific applications to

[1] https://www.darpa.mil/program/fast-lightweight-autonomy

UAVs in RotorS [28], which is studied in depth in [74]. Of primary relevance to this work is Microsoft Research's release of a developing project, AirSim, an Unreal Engine based simulation environment for UAVs [95]. AirSim is a plug-in to Unreal Engine providing a rendered viewpoint of a simulated (or possibly real) UAV location in the Unreal world. Early releases have an eye toward being able to generate large data sets for deep learning based off a simulated UAV model. To the best of our knowledge, the proposed system for UAV development is the first system that allows the UAV computer to use synthetic exteroceptive sensor data along with real interoceptive sensor data, both streaming in real time, while the UAV is in flight experiencing real physics.

Visual-Inertial Navigation. The literature on visual-inertial navigation is vast, including approaches based on filtering, e.g., [53, 37], fixed-lag smoothing, e.g., [76, 60], and full smoothing [10, 24, 26, 79]. We refer the reader to the survey by Forster *et. al.* [26] for a comprehensive review. As the computational power that can be carried on a flying platform has increased, some visual-inertial navigation algorithms have begun to be run in real time on UAVs. Early implementations such as [18, 6, 103] focused on extending the full SLAM system of Klein and Murray (PTAM) [48] to work on aerial vehicles. Because PTAM was originally designed as a single camera solution for small workspaces, subsequent works primarily focus on application to large workspaces without computational costs growing too high, and using IMU information to correct for the scale drift that is inherent in monocular vision only solutions. More recent approaches have included using a cascading estimate of orientation and position with a low rate stereo camera [97], replacing the PTAM visual SLAM system with the semi-direct approach SVO [25] augmented by an IMU [19], using an off-the-shelf pose estimate from an RGB-D sensor (Google Tango) [68], low energy applications [104], a factor graph based approach similar to our own [63], and a high speed multicamera SVO approach [75].

2.2 Hardware

A UAV test platform and development environment was built for the testing of on-board autonomous navigation algorithms while performing agile maneuvers. To integrate the electronics on the UAV, a custom carrier board for the NVIDIA TX1 module was designed, providing the interfaces necessary for sensing and control, while minimizing size and weight.

A mechanical frame was designed and built around this carrier board (see Fig. 2-1a). The UAV is fully controlled by an on-board NVIDIA TX1 module with a modular software framework, enabling rapid testing of new algorithms and sensors. A real-time visual simulation environment runs using a motion capture system and the Unity game engine, allowing for rapid prototyping of visual algorithms.

UAV Mechanics. The mechanical layout of the UAV consists of a series of three stacked plates carrying the power, control, and sensors (see Fig. 2-1a). At the bottom is the power plate carrying the electronic speed controllers (ESCs), power distribution board, batteries, and the four quadrotor arms with motors and propellers. The middle board is the TX1 module carrier board, and the top plate serves as a utility plate for mounting the camera(s), external IMU, NAZE flight controller (safety mechanism only), and the WiFi antennas. To reduce the vibration on the sensors, the bottom "dirty" plate is separated from the top two "clean" plates by four mechanical dampeners. For maximum agility, the motors are placed as close as possible to the center of the board.

UAV Electronics. The *Penguin* Carrier Board (Fig. 2-1b) carried by *Penguin* was designed in-house to integrate the TX1 module with the rest of the vehicle. The board is designed to minimize size and weight while providing seamless integration of the essential capabilities. Elements such as an extra microcontroller (Atmel328P MCU) to control the ESCs and high speed data lanes for IMU and camera data were integrated to provide autonomous flight. A single USB 3.0 Point Grey Flea3 monochrome camera with a resolution of 1024x1280 and an external Xsens MTi-3 IMU provide the visual and inertial sensor package for the board. The Point Grey camera uses a Sunex DSL219 fisheye lens; to avoid the high distortion at the edges of the lens only the center of the image was used for VIO algorithms, leading to an effective resolution of 512x640.

2.3 Software Framework

The UAV is controlled through an on-board software setup that provides complete end-to-end operation of the UAV from raw sensor data to the signals sent to each ESC. The system uses Lightweight Communications and Marshaling (LCM) [42] for communication between on-board modules, giving a lightweight and flexible framework. Each on-board module (controller, VIO estimation, motor control) remains agnostic to its data source, allowing for

Symbol	Property	Value
m	mass	$1.05\ kg$
I_x	inertia around x axis	$4.9 \times 10^{-3}\ kg\,m^2$
I_y	inertia around y axis	$4.9 \times 10^{-3}\ kg\,m^2$
I_z	inertia around z axis	$6.9 \times 10^{-3}\ kg\,m^2$
d	torque coefficient	$2.6 \times 10^{-8}\ kg\,m^2$
b	thrust coefficient	$1.89 \times 10^{-6}\ kg\,m$
d	thrust lever w.r.t x and y	$0.158\ m$

Table 2.1: Measured model parameters of the *Penguin* UAV

easy switching between methods and data sources (e.g. moving from motion capture to VIO state estimation). All processing occurs on-board the CPU and GPU of the TX1.

2.4 Control

A backstepping controller based on the work of Bouabdallah and Siegwart [8] was implemented to perform trajectory tracking on the UAV. The controller uses an outer loop position controller and an inner loop orientation controller.

Position controller. The total thrust, U_z, is defined by an altitude controller according to:

$$U_z = \frac{m}{\cos\phi\cos\theta}\left(e_z + g - \alpha_z\left(e_{\dot{z}} + \alpha_z e_z\right) - \alpha_{\dot{z}} e_{\dot{z}}\right) \tag{2.1}$$

where ϕ (roll), θ (pitch), ψ (yaw) are the Euler angles that rotate XYZ (global frame) to xyz (body frame), $e_\eta = \eta^d - \eta$ and $e_{\dot{\eta}} = \dot{\eta} - \dot{\eta}^d - \alpha_\eta\left(\eta^d - \eta\right)$ are offsets from the desired value η^d of variable η, and α_η are control parameters. The x and y positions are controlled by adjusting the associated projections of U_z onto the XY axes, that is, $u_X U_z$ and $u_Y U_z$. The desired values of u_X and u_Y are specified by another set of backstepping controllers:

$$\begin{aligned} u_X^d &= (m/U_z)\left(e_x - \alpha_x\left(e_{\dot{x}} + \alpha_x e_x\right) - \alpha_{\dot{x}} e_{\dot{x}}\right) \\ u_Y^d &= (m/U_z)\left(e_y - \alpha_y\left(e_{\dot{y}} + \alpha_y e_y\right) - \alpha_{\dot{y}} e_{\dot{y}}\right) \end{aligned} \tag{2.2}$$

Orientation controller. Since the systems thrust is assumed to be directly along the z axis of the body, u_X^d and u_Y^d prescribe desired pitch and roll for the inner loop:

$$\phi^d = -\sin^{-1}\left(u_Y^d \cos\psi - u_X^d \sin\psi\right) \tag{2.3}$$

$$\theta^d = \sin^{-1}\left(\frac{u_X^d \cos\psi + u_Y^d \sin\psi}{\sqrt{1-\left(u_Y^d \cos\psi - u_X^d \sin\psi\right)^2}}\right) \tag{2.4}$$

These values, alongside the desired yaw inputs, are then controlled by specifying the torques in their associated directions:

$$U_\phi = \frac{I_x}{l}\left(e_\phi - \frac{(I_y - I_z)}{I_x}\dot{\theta}\dot{\psi} - \alpha_\phi\left(e_{\dot{\phi}} + \alpha_\phi e_\phi\right) - \alpha_{\dot{\phi}} e_{\dot{\phi}}\right) \tag{2.5}$$

$$U_\theta = \frac{I_y}{l}\left(e_\theta - \frac{(I_z - I_x)}{I_y}\dot{\phi}\dot{\psi} - \alpha_\theta\left(e_{\dot{\theta}} + \alpha_\theta e_\theta\right) - \alpha_{\dot{\theta}} e_{\dot{\theta}}\right) \tag{2.6}$$

$$U_\psi = \frac{I_z}{l}\left(e_\psi - \frac{(I_x - I_y)}{I_z}\dot{\phi}\dot{\theta} - \alpha_\psi\left(e_{\dot{\psi}} + \alpha_\psi e_\psi\right) - \alpha_{\dot{\psi}} e_{\dot{\psi}}\right) \tag{2.7}$$

Finally the desired forces and torques are transformed into appropriate propeller speeds using the thrust and torque coefficients assuming a quadratic relationship with motor speed.

2.5 State Estimation

The on-board state estimation system uses an inner loop visual-inertial Extended Kalman Filter (EKF) for high rate control, with an outer loop fixed lag smoother Visual-Inertial Odometry (VIO) algorithm providing accuracy and robustness. The VIO algorithm estimates the motion of a device from visual and inertial cues. Our VIO approach is based on the work by Forster *et. al.* [26] with modifications made to allow for real time state estimation on the limited computation of a TX1 module. In the following, we discuss the different components of our VIO pipeline, made up of the low-level signal processing (*vision and IMU front-end*), the inference engine used for accuracy (*estimation back-end*), and the high-rate visual-inertial filter used in the control loop (*visual-inertial EKF*).

The Vision Front-end. Our vision front-end includes *feature detection*, *tracking*, and *geometric verification*. The state estimation system uses a keyframe based scheme where computationally intensive tasks (feature detection, MAP estimation, geometric verification) only occur at keyframes, while computationally cheap tasks (feature tracking, EKF estimation) occur at full camera frame rate. A camera frame is declared to be a keyframe if one of

three situations occurs: after a maximum amount of time has elapsed, after the smoother has finished processing the previous keyframes, or if the number of tracked features drops below a threshold. The feature detector, triggered at each keyframe, extracts Shi-Tomasi corners [98]. Between keyframes, given the pixel locations of the features in the $(k-1)$-th frame, we use the Lucas-Kanade feature tracking method for finding the location of these features in the k-th frame. We use OpenCV's GPU implementations for these tasks. To restrict the computational complexity of the optimization problem for real tiem application on embedded systems, features are restricted in the length of time they may be tracked for.

Verification of the tracked features is performed using 2-pt RANSAC [50] (implemented in OpenGV [49]) to determine the largest set of tracked features that could be described by a rigid body transformation given the rotation estimated from Euler integration of the on-board gyroscope. Individual tracked features are converted from raw pixel locations to physical direction measurements through the camera calibration, which is performed with the OCamCalib Toolbox [92] for our on-board camera, and a known pinhole model for the Unity generated images.

The IMU Front-end. The IMU front-end is responsible for the preintegration of IMU measurements, which amounts to compressing a set of IMU measurements collected between two consecutive keyframes into a single preintegrated measurement and its corresponding covariance matrix. Preintegration decouples the IMU measurement from the keyframe states that it links, allowing for those states to be updated in the MAP estimation without performing the computationally intensive task of reintegrating the IMU.

The on-manifold preintegrated IMU model used in this system was proposed and described in detail in [26], and is described briefly below for clarity. Let us denote the accelerometer and gyroscope measurements acquired at time k by $a_k \in \mathbb{R}^3$ and $\omega_k \in \mathbb{R}^3$, and denote their respective biases at time k by $\tilde{b}_k^a \in \mathbb{R}^3$ and $\tilde{b}_k^g \in \mathbb{R}^3$. We wish to determine the relation between the state, $\mathbf{x}$, of the UAV at two consecutive keyframes, where the state is made up of the attitude R, position p, velocity v and IMU biases b^a, b^g.

Considering two consecutive keyframes at time i and j, the IMU preintegration performs integration of the IMU measurements (a_k, ω_k) for all sampling times $k = i, \ldots, j$, with time spacing Δt, to produce a relative rotation $\Delta\tilde{R}_{ij}$, a psuedo-relative velocity $\Delta\tilde{v}_{ij}$, and a

psuedo-relative position $\Delta\tilde{p}_{ij}$ in the local frame at time i, as shown below [26]:

$$\begin{aligned}
\Delta\tilde{R}_{ij} &= \prod_{k=i}^{j-1} \mathrm{Exp}((\omega_k - \tilde{b}_k^g)\Delta t) \\
\Delta\tilde{v}_{ij} &= \sum_{k=i}^{j-1} \Delta\tilde{R}_{ik}(a_k - \tilde{b}_k^a)\Delta t \\
\Delta\tilde{p}_{ij} &= \sum_{k=i}^{j-1} \left[\Delta\tilde{v}_{ik}\Delta t + \frac{1}{2}\Delta\tilde{R}_{ij}(a_k - \tilde{b}_k^a)\Delta t^2\right],
\end{aligned} \tag{2.8}$$

These same values can be computed directly as functions of the keyframe states and noise values $\delta\phi_{ij}, \delta v_{ij}, \delta p_{ij} \in \mathbb{R}^3$ in Eqn. 2.9. The decoupling of measurement integration and keyframe states significantly saves computation by allowing for adjusting state estimates during optimization without reintegrating IMU measurements.

$$\begin{aligned}
\Delta R_{ij} &= R_i^\mathsf{T} R_j \,\mathrm{Exp}(\delta\phi_{ij}) \\
\Delta v_{ij} &= R_i^\mathsf{T}(v_j - v_i - g\Delta t_{ij}) + \delta v_{ij} \\
\Delta p_{ij} &= R_i^\mathsf{T}(p_j - p_i - v_i\Delta t_{ij} - \frac{1}{2}g\Delta t_{ij}^2) + \delta p_{ij}
\end{aligned} \tag{2.9}$$

The Maximum a Posteriori (MAP) Back-end. The optimization back-end performs fixed-lag smoothing and computes the MAP estimate of the most recent keyframe states within a given time window, using the measurements produced by the front-end. Of critical importance for real time flight applications is that the state estimator is not only accurate and fast, but that it is *consistently* so. Even if average accuracy and timing values for the system are low, a single long delay in computation can potentially cause the UAV to crash. To ensure more consistent computation times, the MAP system is restricted in the data in receives both in time, and in number. Only a fixed number of vision measurements are used, for a defined number of keyframes into the history. Older measurements are marginalized out of the factor graph. In addition, a pure odometry system is used, sacrificing global accuracy but removing the high, and variable, computation costs of loop closures.

The *vision measurements* (produced by the vision front-end) are the pixel observations of a landmark in a keyframe. More specifically, each measurement u_{im} generated by the vision front-end represents the projection of landmark $l_m \in \mathbb{R}^3$ onto the keyframe at time i. The

vision measurements are included in the MAP problem as structureless vision factors which treat the unknown landmark location l_m as a direct function of the measurements of the landmark and the state estimate, rather than as an unknown variable in the MAP estimation [26]. The *IMU measurements* (produced by the IMU front-end) are the preintegrated measurements $(\Delta\tilde{R}_{ij}, \Delta\tilde{v}_{ij}, \Delta\tilde{p}_{ij})$ in (2.8).

The MAP estimator is a nonlinear least squares optimization problem, whose minimum is the MAP estimate:

$$\begin{aligned}\mathbf{x}^* = \arg\min_{\mathbf{x}} & \sum_{(i,j)\in\mathcal{F}} \|r_{\text{IMU}}(\mathbf{x}_i, \mathbf{x}_j, \Delta\tilde{R}_{ij}, \Delta\tilde{v}_{ij}, \Delta\tilde{p}_{ij})\|^2 + \\ & \sum_{m\in\mathcal{L}} \sum_{i\in\mathcal{F}_m} \|r_{\text{CAM}}(\mathbf{x}_i, u_{im})\|^2 + \|r_{\text{PRIOR}}(\mathbf{x})\|^2\end{aligned} \tag{2.10}$$

where the elements of Eqn. (2.10) are the negative log-likelihood of the IMU measurements, vision measurements, and the priors, respectively. In Eqn (2.10), $\mathcal{F}$ is the set of consecutive keyframes indices, $\mathcal{F}_m$ is the set of keyframes in which landmark m has been observed, and $\mathcal{L}$ is the set of landmarks observed during the time horizon. The functions $r_{\text{IMU}}(\cdot)$, $r_{\text{CAM}}(\cdot)$, $r_{\text{PRIOR}}(\cdot)$ are often called *residual errors* in that they quantify the mismatch between a given state estimate and the available measurements and priors. The optimization problem in Eqn. (2.10) is solved using the incremental smoothing algorithm iSAM2 [44] implemented in the GTSAM 4.0 toolbox [15].

Visual-Inertial Extended Kalman Filter (EKF) Estimator. Because of the computation limitations on-board the UAV, the MAP estimation back-end runs with a keyframe rate of only 3-10 Hz, which is insufficient for use in closed loop control. Unlike standard techniques which use IMU data only to bridge the gaps in MAP estimates, we take advantage of the frame rate camera data in a decoupled visual-inertial EKF. The EKF follows the standard two step EKF process with a prediction step provided by IMU integration and an update step provided by comparing frame rate feature measurements against the estimated 3D landmark locations generated by the MAP estimator. The IMU state prediction is given by Euler integration of the acceleration and angular velocity measurements, with the unknown bias terms $\tilde{b}_k^a$ and $\tilde{b}_k^g$ updated at each keyframe by the MAP estimator.

The camera update step runs at the frame rate of the camera (rather than the keyframe rate used for the MAP estimate) using the error between tracked pixel locations u_m of the m^{th} landmark and the reprojection of the MAP estimated 3D location $\tilde{l}_m$ onto the camera.

Because the EKF uses tracked features from every frame in its update step and the outlier rejection method (RANSAC) only occurs on keyframes, we perform a fast outlier rejection by excluding measurements for which the reprojection error is above a threshold.

By using the IMU bias and the 3D landmark locations from the MAP estimate, the EKF maintains similar accuracy to the MAP estimate, while still running at a rate suitable for closed-loop control. Furthermore, by using the visual data between keyframes from the EKF estimate, a more accurate initial guess for the pose is used in the MAP smoother. By creating a decoupled system between the EKF and the MAP smoother, the state estimation system is more robust to fluctuations in accuracy and speed of the MAP smoother, allowing for a smooth accurate signal for controlling the vehicle. Large jumps in the MAP estimate which can be hazardous for control (even when improving the accuracy of the state estimate) are naturally smoothed out by the filter.

The full estimation system, from processing IMU and camera data in the IMU/Vision front-ends, to a high rate state estimate from the EKF is shown in Figure 2-2.

2.6 Photorealistic sensor simulation in the Loop (PiL)

Algorithm development for UAVs faces a natural challenge that many of the locations that we want to deploy UAVs (cities, forests, and other obstacle rich landscapes) are difficult and dangerous locations to develop algorithms. To counteract this, we have developed a simulation system that allows for real dynamics, inertial measurements, and closed loop control, while simulating the exteroceptive sensors that are primarily effected by the surrounding environment.

Simulation of exteroceptive data (in this case imagery) is performed in the Unity game engine via a ground station computer featuring an NVIDIA TitanX GPU. The simulation of imagery is performed by creating an environment in Unity that contains a virtual world for the UAV, and one or more camera objects which are attached to a TCP socket. Over TCP, the various parameters of the camera may be set, most importantly the camera pose can be set in real time based on the motion capture position of the UAV. For each pose of the camera received, the Unity camera object returns a timestamped image of the virtual reality environment as it would be seen from that pose (see Figure 2-3). Because of the networking limitations of sending full images wirelessly to the UAV, for our VIO state

estimation experiments using simulated imagery the vision front-end is executed on the ground station computer at a speed that the on-board GPU can execute, and only feature data is sent wirelessly to the UAV for state estimation. The total delay in receiving visual data on the UAV (rendering and wireless transmission) is primarily Gaussian around $37 \pm 8\ ms$ with 1.3% outliers above two standard deviations due to wireless network dropouts. For comparison, the time from image acquisition to processed data with our live camera is $15 \pm 5\ ms$.

While our image simulation system may be used for traditional Hardware in the Loop (HiL) simulations (simulated dynamics and inertial measurements with real decision making) or with logged data from a real UAV (real dynamics and inertial measurements, predetermined decisions), it was implemented with the intention of running in real time while the UAV is in the air for Photorealisitic (exteroceptive) sensor simulation in the Loop (PiL) (real dynamics and inertial measurements, online decision making). By running all systems in real time the PiL system comes as close as possible to a real camera running on-board the UAV, allowing visual algorithms to be used in the decision making loop.

2.7 Experiments

Experimental Setup. Experiments were performed in an approximately rectangular $6\,m \times 4\,m$ environment. A set of 6 OptiTrack Prime 17W cameras provide a ground truth pose estimate in the enclosed area, running at 120Hz, which is used for photorealistic camera image generation. Three sets of experiments were performed:

1. Visual state estimation and control in a baseline scenario involving an indoor environment

2. Visual state estimation and control in a challenging scenario involving flying through a window

3. Camera parameter sweep to investigate estimation accuracy for various camera parameters

The first two experiments were conducted both in simulated environments (using FlightGoggles) and in real environments, whereas the last experiment was performed purely taking advantage of the simulated environments. Each experiment has two phases; first, a take-off

phase where the UAV flies under motion capture using position references provided by the operator, and second; the experimental phase where the UAV flies with the VIO state estimate in the loop and executes a predefined periodic pattern until a low battery warning occurs. The take-off period serves to both stabilize the VIO state estimate with visual features (a natural restriction of a monocular method) and to initialize the integrator for the controller. Due to the small available flight space and long flight times the UAV will eventually drift into a wall if given a fixed reference trajectory in the global coordinate frame. To keep the UAV within the flight cage drift is corrected by shifting the global desired trajectory to match the visual-inertial odometry (VIO) local frame after each loop. This mimics the behavior that would occur if the UAV were generating trajectories based on its available local map to navigate a room.

Visual Navigation in Open Space. In total 42 experiments were performed in open space, 21 using the on-board camera on the UAV and 21 using our photorealistic image generation system to simulate a camera in real time. In all 21 experiments using the on-board camera and in 19 out of 21 experiments using the simulated camera the UAV traced out the desired trajectory with the VIO state estimate in the control loop for the full life of the battery (2-3 minutes). In the two simulated experiments that had to be ended early, WiFi network dropouts caused visual data to not reach the UAV, and the experiment was ended for safety. The reference trajectory flown for these experiments is an oval of length 2.8 m and width 1.6 m, with a period of $3.5 - 3.8$ s, for an average speed of ~ 2 m/s and a maximum speed of ~ 3 m/s on the long sides of the oval.

The estimation error as a function of distance traveled for all 42 experiments is shown in Figure 2-6. Since this system has no loop closures the initial estimation error during take-off cannot be recovered, resulting in the higher error percentages at the beginning of the flight when little distance has been traveled. Once the UAV starts flying its trajectory the estimation error remains below 1% (1 cm error for every 1 m flown) in all experiments. Note that the tracking of features is intentionally limited to 3 seconds, both to maintain low computation costs and to better mimic flying through an ever changing environment where no features can be seen continuously. The VIO state estimate was continuously in the control loop without assistance from motion capture for all 42 flights, demonstrating a stable and accurate state estimate.

Visual Navigation through a Window. Our second set of experiments involves fly-

ing through a window ($0.90\,m \times 0.60\,m$, approximately twice the size of the UAV) with the VIO state estimate in the loop. Flying through windows presents a challenging problem for monocular VIO systems with forward facing cameras, as the visual element of the VIO system relies on motion to triangulate features. When flying through a window the only visual data linking state estimates on one side of the window to the other are those seen through the window, for which there is little tangential motion making triangulation inaccurate.

To the best of our knowledge the only two demonstrations of on-board, vision-based navigation through window openings come from Loianno *et. al.* [69] and Falanga *et. al.* [21]. In both cases the focus was on trajectory generation and control under uncertainty, and state estimation was only maintained for a single traversal of the window before landing. A key concern is the ability to continue flight after passing through a window; therefore we sought to repeat the baseline experiment with a window in the path of the oval trajectory, although at a slower speed (average speeds of ~1.7 m/s and maximum speed of ~2.3 m/s). At each loop a simulated window detection occurred to set a new flight trajectory through the window, however, this window detection was not used for state estimation.

Our photorealistic sensor simulation system provides the platform to develop our algorithms for window navigation. Developing algorithms with a physical window hazards numerous crashes as the system is developed, and the development of the system without a window or without visual navigation algorithms in the loop does not provide an accurate description of the performance of the UAV powered by visual navigation algorithms. Instead, our development environment allows for the visual effect of flying through a window, real dynamics, and real inertial measurements, without crashing on failure. We show the pipeline for developing with photorealistic image simulation in Figure 2-5, first developing and tuning the system with virtual imagery and then performing the same task in the real world. Once the system was successful with virtual imagery, it was a simple change of camera source and camera parameters to perform the same task in the real world.

A total of 10 flights were performed with a simulated camera, constituting 361 traversals of the window, with 3 traversals resulting in a "crash" with the virtual window (crashes detected from the motion capture position of the UAV). The estimation error across the flights is shown in Figure 2-6.

Through experimentation with simulated imagery we found that a high keyframe rate with less feature data is necessary to both consistently bridge the gap created by flying

through the window, and to quickly re-establish an accurate state estimate on the other side of the window. Based on these lessons, we performed the same experiment with a real window and the on-board camera. A total of 8 flights were performed, constituting 119 traversals of the window with 6 crashes/pilot take overs due to estimation divergence. Post processing of data revealed that the lower success rate when using a real camera was due to a combination of noisier visual data than provided by the simulation system, and the additional computational load on the on-board computer of image acquisition slowing down the optimization rate. The noisier visual data can come from a combination of lower quality features in the world, motion blur of the live camera, and imperfections in the estimated camera model.

Camera Parameter Tests. In addition to allowing for testing in a variety of visual environments, our development environment also allows for rapidly evaluating sensor properties and configurations. For instance, we took a 70 second pre-recorded flight of our UAV flying an oval trajectory under motion capture and tested the VIO's performance in real-time using real-world IMU measurements against a set of camera parameters spanning Field of View, Camera Resolution, and Frame Rate. See Figure 2-7 for a selection of results. These measurements are not meant as a declaration of the best camera to use for visual-inertial navigation, but rather to show the capabilities of the system for rapid system prototyping to fit new challenges.

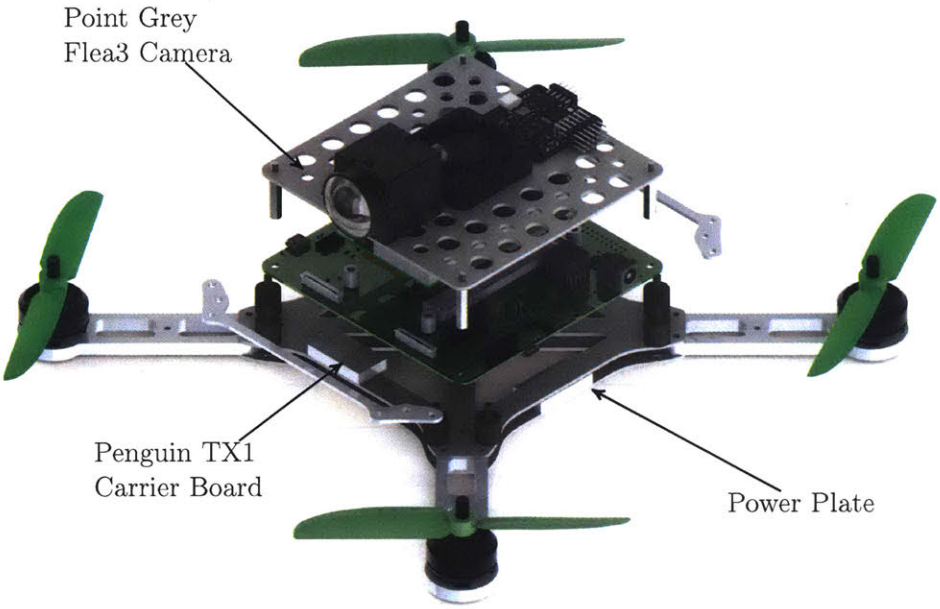
The ease of experimenting with this simulation system makes testing out sensor configurations in new flight scenarios easy and cost effective. While we have focused on a few parameters of the camera sensor itself, a wide range of other effects such as camera blur, scene lighting, feature richness, and accuracy of the camera model can easily be investigated.

2.8 Conclusion

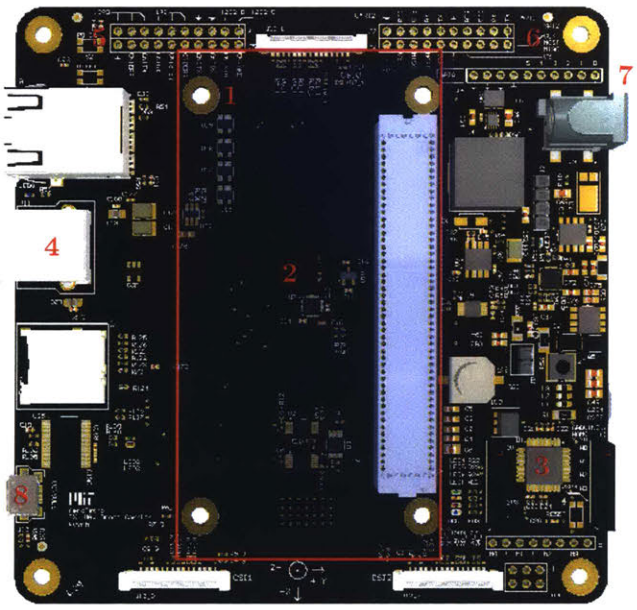
The work described here demonstrates the capabilities of our fully integrated drone platform, its onboard visual-inertial odometry system, and a novel real time visual simulation environment. The full platform provides the essential components for further development of high speed vision based algorithms, with the required hardware, processing power and onboard state estimation performance. The combined drone and visual simulation system provides a platform for rapid development of vision based algorithms in increasingly complex

scenarios.

The results shown here demonstrate both that the simulated imagery is viable as an alternative to real onboard imagery for algorithm development, and that developing first in a simulated world and then moving to the real world can be a seamless process. This architecture, combined with the controller from [99], has been used in [4] to collect a large scale publically available dataset of high speed UAV flight containing high rate imagery, onboard inertial data, and precise motor speeds.



(a) Exploded view of *Penguin* quadrotor platform used in VIO experiments



1. NVIDIA TX1
2. MPU-9250 IMU
3. Arduino MCU
4. USB 3.0 Host
5. CAN/I2C Port
6. UART/SPI Port
7. GPIO Port
8. Serial Debug

(b) *Penguin* Carrier Board

Figure 2-1: Mechanical and electronic designs for our agile quadrotor platform *Penguin*. The quadrotor platform is CNC routed out of Garolite G10 laminate for a strong and lightweight housing of the drone electronics. Custom carrier boards were designed in house to provide all of the essential elements for flight (TX module, IMU, camera(s), motor control) while minimizing the weight and footprint of the electronics.

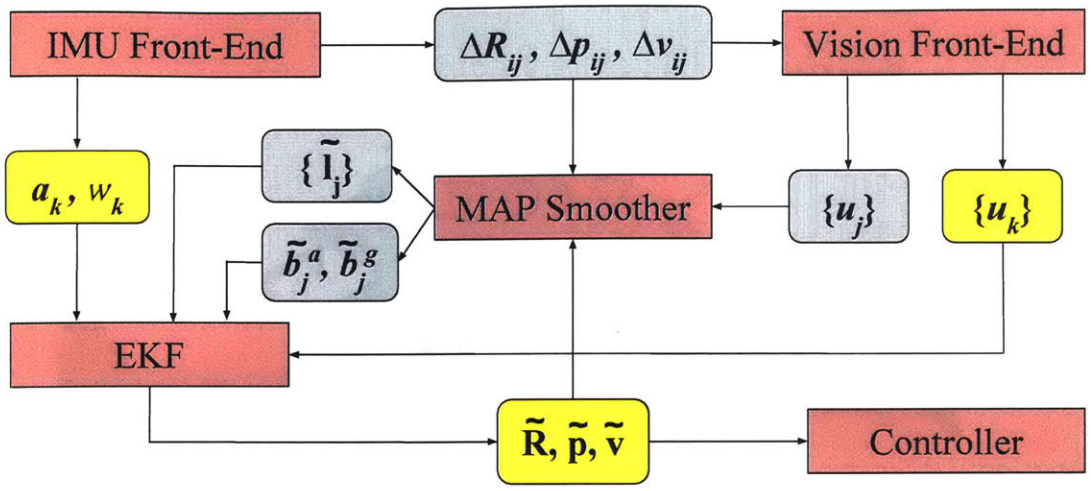


Figure 2-2: Diagram of the full VIO state estimation system. Gray items are transmitted at keyframe rate (3-10 Hz) and yellow items are transmitted at frame rate (60 Hz). Subscripts i and j denote subsequent keyframes, while subscript k denotes the current frame.

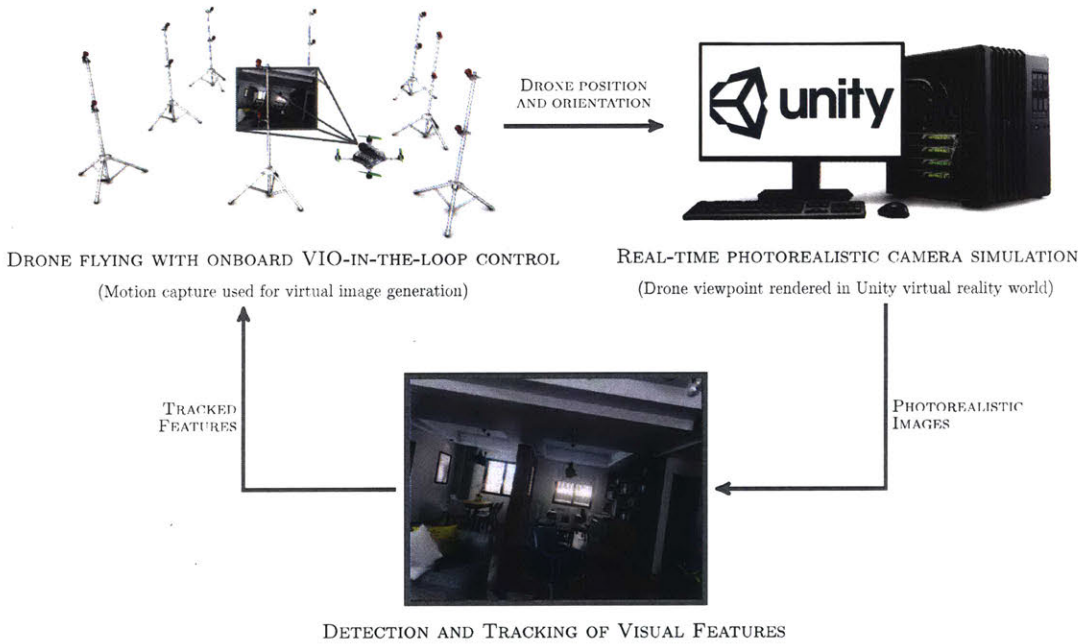
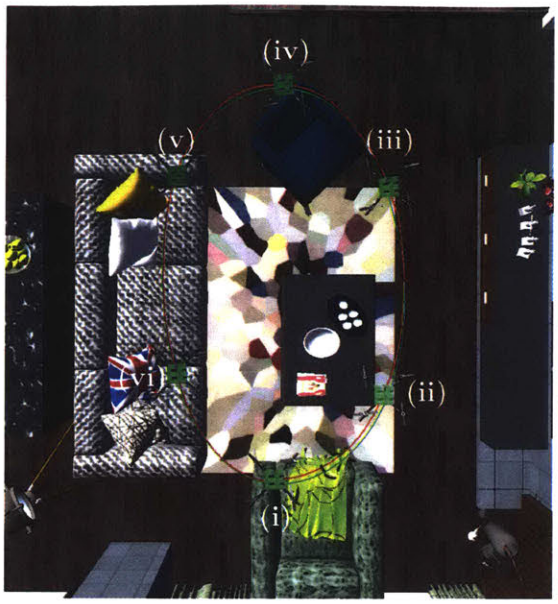
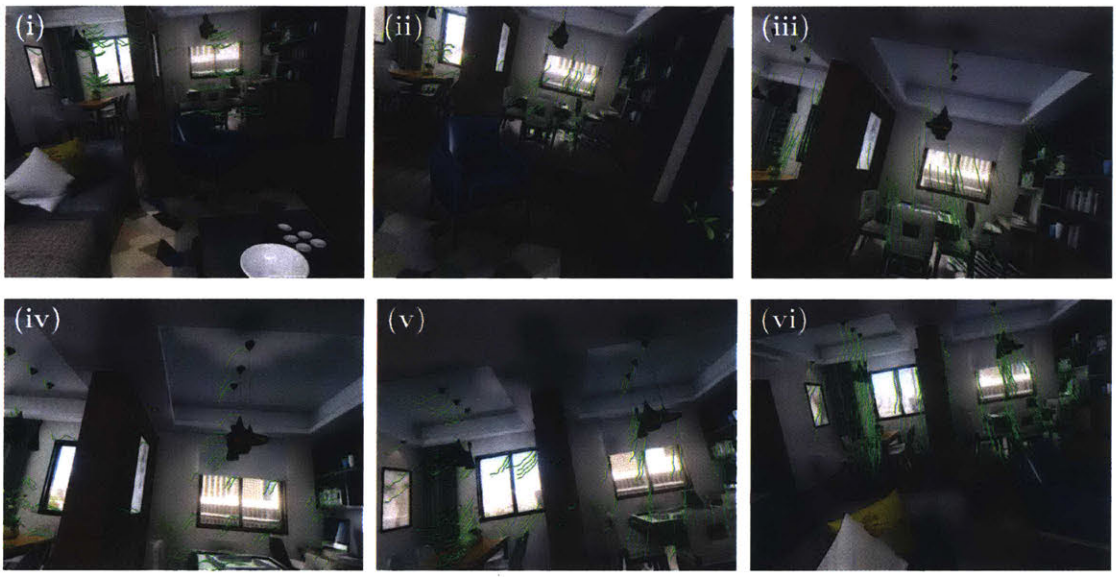


Figure 2-3: To enable algorithmic work in a wide range of visual conditions we have developed a system to replace the UAV's on-board camera with images from a virtual environment. While the UAV is in flight (left) the motion capture pose estimate of the UAV is sent to the Unity game engine running on a TitanX GPU (right) which can generate the corresponding photorealistic image (bottom) for that pose from a virtual world which is processed and transmitted to the UAV in real time. The system runs fully in real time as if the sensors were on the UAV, allowing experiments and decision making in adverse conditions such as obstacle rich environments or in environments that are difficult to access such as cities.



(a) Trajectory from a single oval of VIO-in-the-loop flight overlaid on the 3D model used for photorealistic image generation. The green line shows the position of the drone as recorded from motion capture while the red line shows the VIO state estimate of position used for control.

(b) Top view of the drone flying in the lab during the experiment, the pose of the drone for the six images images displayed below is emphasized.



(c) Six photorealistic images generated and streamed to the drone in real time for VIO state estimation during a VIO experiment around an oval trajectory. Green lines show 0.3 seconds of history for visual features detected and tracked by the vision front end. Our simulation system generates photorealistic images at 60 Hz based on motion capture pose estimate, which are processed by the VIO system, and used in-the-loop for controlling the drone's flight.

Figure 2-4: Visualization of a VIO experiment using simulated imagery showing the (a) true and estimated trajectories of the drone, (b) top view of the drone's flight, and (c) six images generated during flight with the tracked features shown.

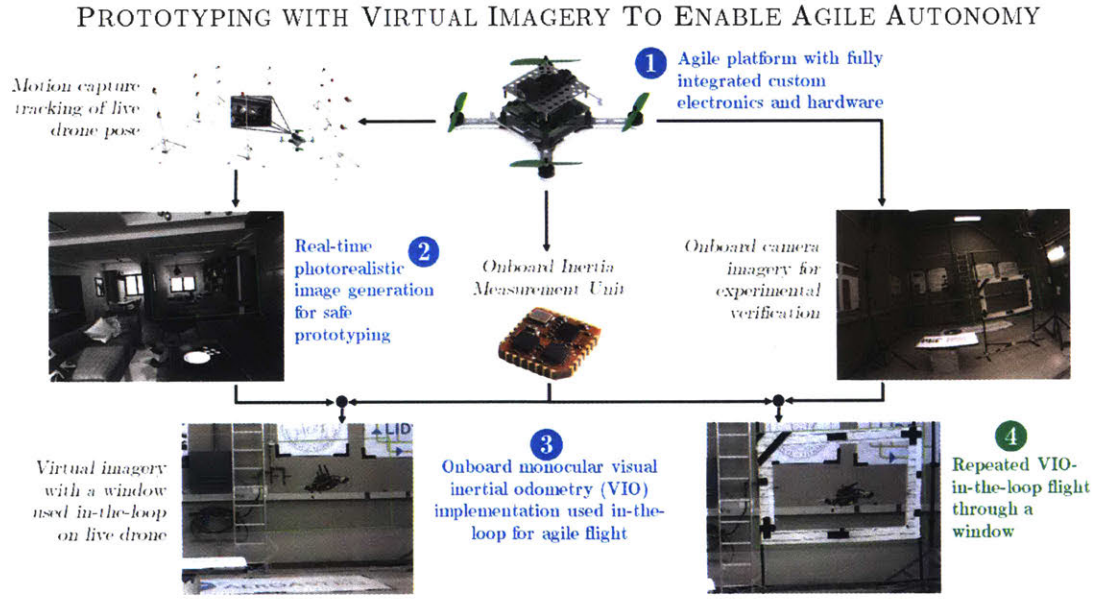


Figure 2-5: An example development pipeline using photorealistic imagery in the loop. While the UAV flies in a motion capture room, photorealistic imagery is sent to the UAV to be fused with onboard sensors (IMU) and used in the control loop. Once the algorithms have been developed, it is a simple switch to use onboard camera imagery. No other changes are required as the UAV is already in flight with autonomy algorithms in the control loop. In this case, the challenging scenario was flying through a window gap repeatedly with a forward facing camera. The ability to "hit" the virtual window at no cost during development significantly decreased both the cost and time of development.

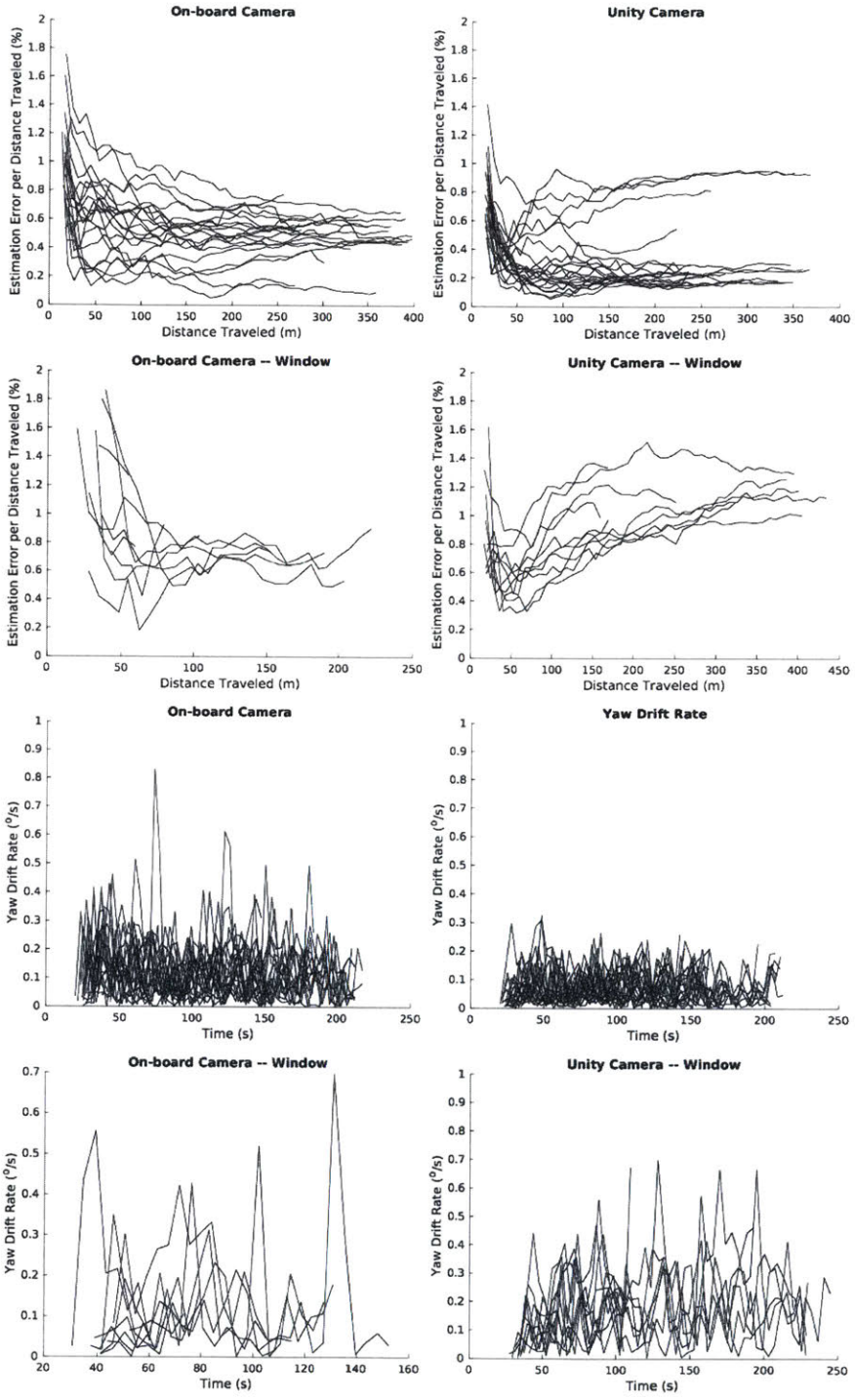


Figure 2-6: Error in VIO state estimate of the UAV's position as a percentage of the distance traveled by the UAV, and rate of error in the VIO estimate of yaw. Flights flown with the on-board camera are shown on the left while flights flown using camera images rendered in Unity are shown on the right; a total of 21 flights were flown with each style of camera without a window, and 8 and 10 flights with the real and simulated cameras respectively were flown through a window.

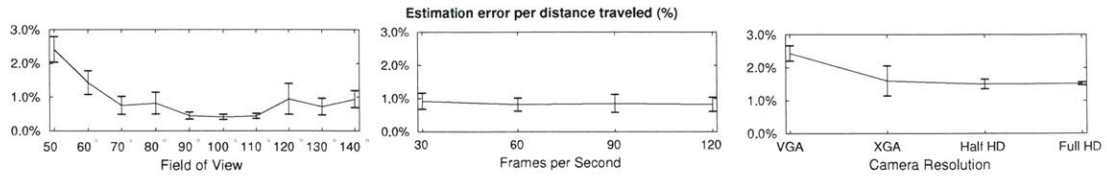


Figure 2-7: Data from camera parameter testing using real-time photorealistic image simulation generated from logged flight data to assess VIO estimation performance for different camera types. Eight trials were performed for each sensor type. All trials were run in real-time using our simulation pipeline and an attached Jetson TX1. FOV trials were conducted with XGA (1024x768) resolution at 60 FPS. FPS trials were run at VGA (640x480) resolution and 80° field of view. Camera resolution trials were conducted at 50 FPS and 60° FOV.

(a) Unity generated image with window to fly through in the upper right corner.

(b) Image of UAV flying through a physical window using VIO in the loop based on its on-board camera and IMU.

Figure 2-8: Images from VIO experiments, showing an image from a virtual on-board camera (left) and of our UAV flying through a window gap under VIO control (right)

Chapter 3

Perception-driven motion planning

Both motion planning and mapping are fundamental problems of robotics. The problem of mapping is to create an accurate representation of obstacles around a robot based on sensory measurements, and the problem of motion planning is to find a dynamically-feasible trajectory around these obstacles. These two problems are intimately linked. A high-quality motion plan often requires working with an accurate high-resolution map, which requires extensive processing of large amounts of sensory data.

Unfortunately, both the mapping problem and the motion planning problem can require significant computational resources. The computational constraints are even more pronounced for small vehicles attempting to traverse complex environments rapidly. Due to the small size of the vehicles, relatively limited computational platforms can be carried on board. Due to the fast operation of the vehicles, there is little time that can be devoted to computation. In particular, typical mapping methods using camera data, *e.g.*, stereo reconstruction, structure from motion, and learning based techniques, are all computationally burdensome, and their computation scales directly with the amount of area they need to map.

Due to the computation effort devoted to mapping there has been increasing interest in methods that attempt to minimize the processing of sensory data. For instance, the pushbroom stereo method [5] avoids full stereo depth computation for each stereo pair by integrating over time, and the NanoMap method [23] maintains data in a sensor frame to avoid explicitly integrating it into a map.

In this chapter, we consider a joint mapping-and-planning problem, in which sensory

data is processed for mapping only when it is necessary for planning, with the aim of minimizing the computational costs (for both mapping and planning), while maintaining the completeness and optimality guarantees of motion planning. The algorithms that solve this problem are well suited for online settings that require small vehicles to rapidly traverse complex environments that are unknown *a priori*, but revealed in an online manner.

In many implementations robot navigation systems, the coupling between motion planning and the obstacle map is through a search graph, *e.g.* [54, 39]. The nodes of the graph consist of a set of robot states with the edges representing collision-free, dynamically-feasible trajectories between these states. There is a vast literature on the *construction* of this discrete graph from the continuous robot dynamics and its environment. Common algorithms include regularized discretizations of the state space into state lattices, roadmap methods (*e.g.*, PRM [47]), and tree methods (*e.g.*, RRT [56] or RRT* [46]). There is also a large literature on algorithms that focus on the *search* of an existing graph. A* [34] is the *de-facto* standard search method with numerous adaptations to provide properties such as planning on dynamic graphs (*e.g.*, D* Lite [52]), and heuristically accelerated planning (*e.g.*, ARA* [61]).

The algorithm proposed here lies in the area of graph construction, starting from a single edge from origin to goal and an empty map, and iteratively switching between checking the validity of the solution (mapping), and updating the graph structure to account for newly processed sensor data (planning). The graph is structured as a tree of "Problems" where each problem is an sub-motion planning problem from some intermediate state in configuration space to the goal state.

Our approach incrementally grows a sparse tree by taking advantage of two provable properties of the problem, specifically that (1) the constrained optimal trajectory will be made of free space trajectories joined at the boundaries of obstacles and (2) adding obstacles to the problem will never decrease the optimal cost of the motion planning problem. By not constraining the trajectories to a pre-determined form, we are also able to handle systems with differential constraints, provide a naturally multi-resolution representation of the state space, and create plan trees that can be efficiently queried, while minimizing the mapping required. The incremental growth of the plan tree starts from a best case, obstacle free solution to the motion planning problem, checks it against available sensor data, and in the event of discovering a new obstacle adds new elements to the plan tree to avoid the obstacle.

This process is repeated until an optimal solution is reached. By mapping along only the current best solution we can perform "edge optimal graph search" following the model of Dellin and Srinivasa [16] to perform minimal mapping on the way to finding the optimal solution. These techniques are a unification of the general lazy techniques used in literature [7].

This chapter is organized as follows. Section 3.1 lays out the related work in the field of graph construction and minimal sensing. Section 3.2 lays out the notation that will be used in this chapter. Section 3.3 describes the coupled mapping-and-motion planning problem we seek to solve. Section 3.4 describes the principles behind the algorithm and the algorithm itself. Section 3.5 provides the proofs surrounding the algorithm's properties. Section 3.6 describes ways of extending the motion planning algorithm to work in high dimensional spaces, with dynamic systems, and as an anytime system.

3.1 Related Work

A typical setup for the robotic motion planning problem combines a pre-built map of the environment and a plan graph of nodes (robot states) and edges (robot trajectories). In common implementations the plan graph uses the map for validity checks of nodes and edges, but does not reference it for the structure of the plan graph [57]. Many methods have been developed, however, that more tightly couple the planning and mapping processes to achieve lower cost trajectories or computationally more efficient planners. For example, visibility graphs place nodes at the vertices of polygonal obstacles providing exactly optimal solutions for 2D holonomic robots [2]. In sampling based methods, the obstacle map can be used to inform node sampling strategies such as increased placement near obstacle boundaries for navigating cluttered environments and narrow cooridors [3].

Several methods exist which use the result of collision checks executed during search to adapt the structure of the plan graph. The any angle planning variants Theta* [80], Lazy Theta* [81], and Incremental Phi* [62] start with a grid structure for the initial search but add virtual diagonal edges to shorten the path where permissible on the obstacle map. Of close relation to this work, several planners have been proposed which initially start searching a simple plan graph, and incrementally increase the complexity of the problem based on the result of collision checks. For example, Wagner and Choset [101] propose initializing an

n-robot planning problem as n 1-robot sub-problems. When two sub-problems produce collisions between robots, they are combined and re-solved, thereby locally increasing the complexity of the problem but eliminating the collision. This process is repeated until no collisions remain. Similar concepts have been proposed for other navigation scenarios. Shah *et. al.* [94] initialize with a low resolution grid representation of the state space and adaptively increase its resolution during search. Gochev *et. al.* [31] propose a method that incrementally increases the dimensionality of the state space for complex robots when low dimensional representations cause collisions.

Another field of work has looked at reducing computation by minimizing the number of collision checks that must be carried out during motion planning. These methods, often called "lazy" search techniques, typically focus on robotic arms [13] but apply to any graph search problem. Dellin and Srinivasa [16] show that several existing algorithms for "lazy" search are actually specific instances of a more general algorithm for minimizing the number of collision checks.

Recent work has considered bringing together robot mapping and motion planning. Pryor *et. al.* [86] use motion planning to determine which areas of the map around a humanoid robot to resolve from sensor data. Ghosh and Biswas [30] show significant reductions in the matching of stereo pairs for a ground robot by directly connecting the checking of disparity matches to the expansion of the plan graph. These methods provide a strong basis for the benefits of creating joint mapping and planning processes.

3.2 Notation

The overall motion planning problem is to find a *feasible* motion between two robot states, formalized as follows. The *configuration space* of the robot is defined as $\mathcal{X} \subseteq \mathbb{R}^d$, and is the set of all possible *states*, written $\mathbf{x}$, of the robot. The *map*, *i.e.* invalid configurations of the robot, is denoted $\mathcal{M} \subset \mathcal{X}$, which is assumed to be an open set. The *free space* is defined as $\mathcal{X}_{free} := cl(\mathcal{X} \backslash \mathcal{M})$, where $cl(\cdot)$ is the closure operator.

The dynamics governing the transition between robot states is represented by an ordinary differential equation of the following form:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{3.1}$$

where $\mathbf{u} \in \mathcal{U}$ is the control input. A *dynamically feasible trajectory* from a starting state $\mathbf{x}_{start} \in \mathcal{X}$ to a final state $\mathbf{x}_{goal} \in \mathcal{X}$ is a mapping ${}_{start}\sigma_{goal} : [0, \tau) \to \mathcal{X}$ such that ${}_{start}\sigma_{goal}(0) = \mathbf{x}_{start}$, ${}_{start}\sigma_{goal}(\tau) = \mathbf{x}_{goal}$, and ${}_{start}\sigma_{goal}(t)$ satisfies Eqn. (3.1) for all $t \in [0, \tau)$. A *valid trajectory* from a starting state $\mathbf{x}_{start} \in \mathcal{X}$ to a final state $\mathbf{x}_{goal} \in \mathcal{X}$ against a map $\mathcal{M}$ is a dynamically feasible trajectory ${}_{start}\sigma_{goal}$ such that ${}_{start}\sigma_{goal}(t) \in \mathcal{X}_{free}$ for all $t \in [0, \tau)$. Validity is a map specific term, and we will denote trajectories that are valid on a map $\mathcal{M}$ by ${}^{\mathcal{M}}_{start}\sigma_{goal}$.

We denote the set of all dynamically feasible trajectories from $\mathbf{x}_{start}$ to $\mathbf{x}_{goal}$ as ${}_{start}\Sigma_{goal}$, and the set of all valid trajectories on map $\mathcal{M}$ by ${}^{\mathcal{M}}_{start}\Sigma_{goal}$.

We define a *trajectory cost function* to be a mapping $\mathcal{C}(\cdot) : \sigma \to \mathbb{R}_{\geq 0} \cup \infty$ that assigns a non-negative cost to a trajectory. We define this cost function for a set of trajectories to be the minimum cost of any trajectory in the set, *i.e.* $\mathcal{C}(\Sigma) := \min_{\sigma \in \Sigma} \mathcal{C}(\sigma)$.

We will define the addition of two trajectories as the concatenation of them in time:

$$({}_a\sigma_b + {}_b\sigma_c)(t) = \begin{cases} {}_a\sigma_b(t) & \text{for all } t \in [0, \tau_{ab}); \\ {}_b\sigma_c(t - \tau_{ab}) & \text{for all } t \in [\tau_{ab}, \tau_{ab} + \tau_{bc}), \end{cases} \tag{3.2}$$

which gives a new trajectory ${}_a\sigma_c : [0, \tau_{ab} + \tau_{bc}) \to \mathcal{X}$.

3.3 Problem Statement

The general optimal motion planning problem can be stated as follows:

Problem 1 (Optimal Motion Planning). *Given an optimal motion planning problem instance* $(\mathbf{x}_{start}, \mathbf{x}_{goal}, \mathcal{M}, \mathcal{C}(\cdot))$, *find a valid trajectory* ${}_{start}\sigma^*_{goal}$ *such that* $\mathcal{C}\left({}_{start}\sigma^*_{goal}\right) = \min_{\sigma \in {}^{\mathcal{M}}_{start}\Sigma_{goal}} \mathcal{C}(\sigma)$, *and return failure if no such trajectory exists.*

For this work, we wish to consider a wider problem where the pre-built map $\mathcal{M}$ does not exist, and instead a set of sensor data $\mathcal{S}$ is available with some non-trivial incremental function $f_{map}(\mathcal{S}) \to \mathcal{M}_{\mathcal{S}}$ that can be used to generate a map. We can modify Problem 1 accordingly to generate the new problem:

Problem 2 (Optimal Motion Planning). *Given an optimal motion planning problem instance* $(\mathbf{x}_{start}, \mathbf{x}_{goal}, \mathcal{S}, \mathcal{C}(\cdot))$, *find a valid trajectory* ${}_{start}\sigma^*_{goal}$ *such that* $\mathcal{C}\left({}_{start}\sigma^*_{goal}\right) = \min_{\sigma \in {}^{\mathcal{M}_{\mathcal{S}}}_{start}\Sigma_{goal}} \mathcal{C}(\sigma)$, *and return failure if no such trajectory exists.*

The movement from a map to a sensor opens questions for the motion planning problem, namely how to handle parts of the world for which information is not available within the sensor data. For this work we leave that decision to the function f_{map}, requiring it to generate a full map from any amount of sensor data, and instead focus on the problem of motion planning against the (assumed complete) map $\mathcal{M}_{\mathcal{S}}$.

A naive solution to Problem 2, e.g. generate the map with f_{map} and then solve the motion planning problem will be computationally expensive in two places. First, as noted in the problem definition, f_{map} is a non-trivial function and generating a full map from $\mathcal{S}$ requires the maximum number of evaluations of f_{map}. Second, solving the motion planning problem is related to the size of the search space ${}^{\mathcal{M}_{\mathcal{S}}}_{start}\sigma_{goal}$. Approaches to solving the second problem make up decades of motion planning literature with the assumption of a pre-existing map. For the first problem, specifically for the problem of graphical search, there is a set of literature that is focused on reducing the number of validity checks required for motion planning. These techniques, generally known as "lazy" methods, were first proposed by Bohlin and Kavraki [7]. These methods are typically used for systems with complex collision checks such as robot arms, however, the conceptual approach is equally valid for processing perception data. In recent years, Dellin and Srinivasa have created a coherent picture for these lazy methods, which they denote LazySP [16]. In this work, we will build off the principles of these lazy methods to feed collision checking back into the motion planning problem to simultaneously reduce the computational costs of both the map generation and the motion planning problem.

To properly characterize the desired problem, let us define three subsets of the full solution space Σ. Σ_{out}: elements of the solution space that are not currently in the motion planning problem, Σ_{in}: elements of the solution space that are currently in the motion planning problem *i.e.* the search graph, and Σ_{sensed}: elements of the solution space for which validity against the sensor data/map have been evaluated. These sets have the properties:

$$\Sigma \supseteq \Sigma_{in} \supseteq \Sigma_{sensed} \tag{3.3}$$

$$\Sigma_{out} = \Sigma \setminus \Sigma_{in} \tag{3.4}$$

Generally speaking we can say that the computational requirements of mapping will be related to $|\Sigma_{sensed}|$ and the computational requirements of motion planning will be related

to $|\Sigma_{in}|$. From this, we can state a new, computationally aware, motion planning problem.

Problem 3 (Perception Driven Optimal Motion Planning). *Given an optimal motion planning problem instance* $(\mathbf{x}_{start}, \mathbf{x}_{goal}, \mathcal{S}, \mathcal{C}(\cdot))$, *find a valid trajectory* ${}_{start}\sigma^*_{goal}$ *such that* $\mathcal{C}\left({}_{start}\sigma^*_{goal}\right) = \min_{\sigma \in \mathcal{M}_{\mathcal{S}}\Sigma_{goal}} \mathcal{C}(\sigma)$, *while minimizing some cost mixing function* $g(|\Sigma_{in}|, |\Sigma_{sensed}|)$.

For the theorems put forward in this work we also require two assumptions about the nature of the system. These assumptions are maintained throughout the following sections, but will be referenced when their existence is key.

Assumption 4. *The dynamics of the robot, Eqn.* (3.1), *are such that all candidate optimal solutions from Pontryagin's Minimum Principle [85] can be found for generic boundary conditions* $\mathbf{x}(0) = \mathbf{x}_0$ *and* $\mathbf{x}(\tau) = \mathbf{x}_f$. *We will denote this candidate set by* ${}^{\emptyset}_{0}\Sigma'_{f}$.

This assumption is effectively a requirement that we have the capability of finding all local minima of the system given boundary conditions in a *obstacle free* world. While this is a strict requirement for the proofs that follow, Section 4.2 demonstrates using an approximation to solve the boundary value problem while still producing a highly effective, though sub-optimal, system.

Assumption 5. *The cost function* $\mathcal{C}(\cdot)$ *obeys the triangle inequality on all* $\sigma \in \Sigma$, i.e..

$$\mathcal{C}({}_a\sigma_c) \leq \mathcal{C}({}_a\sigma_b) + \mathcal{C}({}_b\sigma_c), \text{ for all } \mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c \in \mathcal{X} \tag{3.5}$$

This provides a relatively generic assumption for the standard motion planning problem, however, it can be restrictive for situations with a history based cost function such as localization accuracy [9].

3.4 Algorithm

In this section we will describe the mapping and planning algorithm, first describing its properties (to be proved in Section 3.5), then the conceptual approach, and finally the algorithm itself.

The algorithm described in the following sections, Algorithm 1, will have the following properties:

Theorem 6 (Cost Optimality of Algorithm 1). *Given an optimal motion planning problem instance* $(\mathbf{x}_s, \mathbf{x}_g, \mathcal{S}, \mathcal{C}(\cdot))$, *the solution produced by Algorithm 1,* ${}_s\sigma_g^*$, *will have the property* $\mathcal{C}\left({}_s\sigma_g^*\right) = \min_{\sigma \in {}^{\mathcal{M}_{\mathcal{S}}}{}_s\Sigma_g} \mathcal{C}(\sigma)$, *where* ${}^{\mathcal{M}_{\mathcal{S}}}{}_s\Sigma_g$ *are all elements of* ${}_s\Sigma_g$ *that are valid on the map generated by fully processing* $\mathcal{S}$.

Theorem 7 (Completeness of Algorithm 1). *If there is a valid solution to the optimal motion planning problem instance* $(\mathbf{x}_s, \mathbf{x}_g, \mathcal{S}, \mathcal{C}(\cdot))$, *then Algorithm 1 will return a solution, and if there is no valid solution then Algorithm 1 will return failure in finite time.*

Theorem 8 (Sensing Optimality of Algorithm 1). *There is no alternative algorithm that can be guaranteed to require less sensing than Algorithm 1,* i.e. *result in* Σ_{sensed} *being smaller, while maintaining the guarantees of Theorem 6 and Theorem 7.*

Theorem 9 (Graph Optimality of Algorithm 1). *For any sub-problem* ${}_a\mathcal{P}_g$ *in Algorithm 1, the submap* $\mathcal{M}_a$ *that the problem uses is optimally compact, meaning that it can be made no smaller while maintaining the guarantees of Theorem 6 and Theorem 7.*

Theorem 9 references the optimal compactness of sub-maps in the algorithm. This is used as a proxy for the compactness of the tree search itself, as additional edges are required to account for new obstacles in a map, causing an increase in $|\Sigma_{in}|$, however, the direct optimal compactness of $|\Sigma_{in}|$ is not shown here as additional conditions can be placed on which trajectories must be included in the tree beyond those conditions included in Algorithm 1.

As previously stated, our goal is to solve an *optimal* motion planning problem, meaning that we wish to find a *cost optimal* trajectory that is valid on the underlying world map. Algorithm 1 is centered around two principles: (1) that any optimal trajectory can be built from long, free space optimal trajectories that are joined at boundaries of configuration space (Theorem 10), and (2) solving the optimal motion planning problem for a map that is a sub-map of the true map will result in a solution with cost less than or equal to the true cost (Theorem 12).

Based off of the first of these principles, we will treat the optimal motion planning problem as a decision tree rooted at $\mathbf{x}_s$, where every node in the tree is either a state on the boundary of configuration space or is the goal state $\mathbf{x}_g$, and every edge connecting two nodes is the set of free space optimal trajectories that join those nodes. Every leaf in the tree that ends in $\mathbf{x}_g$ is therefore a valid solution to the motion planning problem. If the tree

is *complete*, then every node in the tree is connected to every other node on $\mathcal{B}(\mathcal{M}) \cup \mathbf{x}_g$, and the minimum cost leaf that ends in $\mathbf{x}_g$ is also the optimal motion plan. Such a tree can be structured as a dense graph and rather trivially be proven to be optimal and complete through Theorem 10, however, it suffers two problems: first, for $N = |\mathcal{B}(\mathcal{M}) \cup \mathbf{x}_s \cup \mathbf{x}_g|$, the graph will have N nodes each with a branching factor of N, making the graph extremely expensive to form and to search for any reasonably sized problem. Second, the graph requires full previous knowledge of $\mathcal{B}(\mathcal{M})$ requiring many expensive sensing evaluations.

Instead, we will look at the overall motion planning problem from $\mathbf{x}_s$ to $\mathbf{x}_g$ as set of N sub-problems from $\mathbf{x}_i \in \mathcal{B}(\mathcal{M}) \cup \mathbf{x}_s \cup \mathbf{x}_g$ to $\mathbf{x}_g$. We will denote such a sub-problem by ${}_i\mathcal{P}_g$. Any given problem can then be broken down into moving from the current state to any other state, and then solving the subsequent motion planning problem. This creates a tree structure, with nodes denoting intermediate sub-problems and edges denoting dynamically feasible trajectories within a sub-problem. A full solution to this tree reverts to the dense graph described above, however, we will use the second principle described above to create a *admissible heuristic* for the cost to solve any given problem. This heuristic (the solution to the sub-problem on a sub-map) drives which sub-problem to investigate and increase the heuristic for. Once the *admissible heuristic* becomes an *optimal heuristic*, meaning that it matches the true solution cost, then the optimal solution to the full problem is found. This sub-map approach fits well with the desire to sense as little of the environment as possible.

To carry out such a heuristic search, each sub-problem, labeled ${}_a\mathcal{P}_g$ will have the following characteristics:

1. A *map*, $\mathcal{M}_a$ of obstacles which have been included in the sub-motion planning problem

2. An ordered set of *sub-maps*, $m_a(i)$ which show the growth of $\mathcal{M}_a$. These sub-maps are used to allow other problems to minimally add another problem. The state of a problem at the addition of $m_a(i)$ will be denoted ${}_a\mathcal{P}_g(i)$.

3. A set of *children*, which are triplets $\left({}_a\sigma_i, {}_i\mathcal{P}_g, {}_az_i\right)$ denoting possible solutions to the problem ${}_a\mathcal{P}_g$ made up of moving to the intermediate state $\mathbf{x}_i$ with ${}_a\sigma_i$, and subsequently solving the problem ${}_i\mathcal{P}_g({}_az_i)$. The value ${}_az_i$ marks the sequence of submaps of ${}_i\mathcal{P}_g$ that ${}_a\mathcal{P}_g$ is aware of. A higher ${}_az_i$ brings the heuristic of ${}_i\mathcal{P}_g$ closer to optimal, but requires more complexity within ${}_a\mathcal{P}_g$. These children are maintained in a priority queue, with priority matching the cost described below. The top priority child is the current best

solution to the motion planning problem.

4. A cost function for a problem $\mathcal{C}\left({}_a\mathcal{P}_g\right)$ given by the top child $\left({}_a\sigma_i,\ {}_i\mathcal{P}_g,\ {}_az_i\right)$, where $\mathcal{C}\left({}_a\mathcal{P}_g\right) = \mathcal{C}\left({}_a\sigma_i\right) + \mathcal{C}\left({}_i\mathcal{P}_g({}_az_i)\right)$

The algorithm has two primary functions, the first is **MakeConsistent** which updates the tree until the current minimal element of the tree is free on all already sensed data $\mathcal{M}_s$. This is a process of incrementally updating sub-problems until their heuristics lie above the optimal cost on $\mathcal{M}_s$. The second function is **Sense** which checks the current best solution against all available sensor data. If the current best solution is valid on Sense, then it is also optimal. If not, the tree is updated for the new map information, and another round of **MakeConsistent** is called.

Result: $\sigma^* = \arg\min_{\sigma \in \mathcal{M}_{S\Sigma_g}} \mathcal{C}(\sigma)$

```
1  Algorithm SparseShortestPath(x_s, x_g)
       /* Initialize the problem                                  */
2      free = False
3      Initialize(_sP_g)
4      while not free do
5          consistent = False
           /* Update the graph given currently processed data     */
6          while not consistent do
7              M_new = MakeConsistent(_sP_g)
8              consistent = IsEmpty(M_new)
           /* Check the best solution against all data            */
9          M_new = Sense(_sP_g)
10         free = IsEmpty(M_new)
       /* Return the unblocked solution                           */
11     return Solve(_sP_g)
```

Algorithm 1: Sparse Graph Algorithm

3.4.1 Algorithm Walkthrough

To better understand Algorithm 1 a visual walkthrough of the algorithm is provided in Table 3.1. For ease of visualization, a 2D holonomic robot moving through line obstacles is used as an example. For a more complex example, please refer to Section 3.6.1. At each iteration of the algorithm (a row in Table 3.1) the currently effected paths in the motion planning problem are shown in the center column, while the search tree is shown in the right column.

```
1  Procedure MakeConsistent(ₐ𝒫_g)
       /* Already reached goal                                          */
2      if x_a = x_g then
3          return ∅
       /* Find the best solution available to ₐ𝒫_g                     */
4      (ₐσ_b, ₆𝒫_g, ₐz_b) = Pop(ₐ𝒫_g)
5      M_new = SenseTrajectory(ₐσ_b, M_s)
6      if IsEmpty(M_new) then
           /* Make ₆𝒫_g up to date in ₐ𝒫_g                              */
7          M_new = UpdateProblem(₆𝒫_g, ₐz_b)
8          Push(ₐ𝒫_g, (ₐσ_b, ₆𝒫_g, ₐz_b))
9      if IsEmpty(M_new) then
           /* ₐ𝒫_g is consistent, move to ₆𝒫_g                          */
10         M_new = MakeConsistent(₆𝒫_g)
11         Push(ₐ𝒫_g, (ₐσ_b, ₆𝒫_g, ₐz_b))
12     AddObstacles(ₐ𝒫_g, M_new)
13     return M_new
14 Procedure Sense(ₐ𝒫_g)
15     if x_a = x_g then
16         return ∅
17     (ₐσ_b, ₆𝒫_g, ₐz_b) = Pop(ₐ𝒫_g)
18     M_new = SenseTrajectory (ₐσ_b, M_sensor)
19     if IsEmpty(M_new) then
20         M_new = Sense(₆𝒫_g)
       /* Update ₐ𝒫_g for any new obstacles found                       */
21     AddObstacles(ₐ𝒫_g, M_new)
22     return M_new
```

Algorithm 2: Sparse Graph Algorithm (Part 2)

Each node in the search tree is a *Problem* in the algorithm, with the node colored to correspond with the physical locations in the center column. As the tree grows new sub-problems are added to effected parts of the tree. Note that at any given step the tree can only be expanded in problems that were part of the current best solution (highlighted in blue).

There are two primary elements of the algorithm that are visualized in this walkthrough:

1. Finding the current best solution (branch) of the tree, as shown in steps (1), (3), (6), and (9). As the search space is an already sorted tree this is a very cheap operation.

2. Checking the current best solution against all available sensor data, steps (2), (4), (5),

1 **Procedure** `Initialize`$({}_a\mathcal{P}_g)$
  `/* Initialize the problem with an empty map */`
2 $\mathcal{M}_a = \emptyset$
3 `Push`$({}_a\mathcal{P}_g, \left({}^{\emptyset}_{a}\Sigma'_g, {}_g\mathcal{P}_g, 0\right))$
4 $z_a = 0$
5 ${}_ah_g(0) = \mathcal{C}\left({}^{\emptyset}_{a}\Sigma'_g\right)$
6 $m_a(0) = \emptyset$
7 **Procedure** `Push`$({}_a\mathcal{P}_g, \left({}_a\sigma_b, {}_b\mathcal{P}_g, {}_az_b\right))$
8 priority $= \mathcal{C}\left({}_a\sigma_b\right) + {}_bh_g({}_az_b)$
9 ${}_a\mathcal{P}_g \rightarrow$ queue() $\rightarrow$ push($\left({}_a\sigma_b, {}_b\mathcal{P}_g, {}_az_b\right)$, priority)
10 **Procedure** `Top`$({}_a\mathcal{P}_g)$
11 **return** ${}_a\mathcal{P}_g \rightarrow$ *queue()* $\rightarrow$ *top()*
12 **Procedure** `Pop`$({}_a\mathcal{P}_g)$
13 bestChild = `Top`$({}_a\mathcal{P}_g)$
14 ${}_a\mathcal{P}_g \rightarrow$ queue() $\rightarrow$ pop()
15 **return** *bestChild*
16 **Procedure** `UpdateProblem`$({}_b\mathcal{P}_g, {}_az_b)$
17 **if** ${}_az_b = z_b$ **then**
18 **return** $\emptyset$
19 ${}_az_b = {}_az_b + 1$
20 **return** $m_b({}_az_b)$
21 **Procedure** `AddObstacles`$({}_a\mathcal{P}_g, \mathcal{M}_{new})$
22 **if** $\mathcal{M}_{new} \notin \mathcal{M}_a$ **then**
23 $\mathcal{M}_a = \mathcal{M}_a \cup \mathcal{M}_{new}$
24 **for** $\mathbf{x}_i \in \mathcal{B}(\mathcal{M}_{new})$ **do**
25 **if** `IsNew`$({}_i\mathcal{P}_g)$ **then**
26 `Initialize`$({}_i\mathcal{P}_g)$
27 `Push`$({}_a\mathcal{P}_g, \left({}^{\emptyset}_{a}\Sigma'_i, {}_i\mathcal{P}_g, 0\right))$
28 $z_a = z_a + 1$
29 $m_a(z_a) = \mathcal{M}_{new}$
30 $({}_a\sigma_b, {}_b\mathcal{P}_g, {}_az_b) =$ `Top`$({}_a\mathcal{P}_g)$
31 ${}_ah_g(z_a) = \mathcal{C}\left({}_a\sigma_g\right) + {}_bh_g({}_az_b)$
32 **Procedure** `Solve`$({}_a\mathcal{P}_g)$
33 **if** $\mathbf{x}_a = \mathbf{x}_g$ **then**
34 **return**
35 $({}_a\sigma_b, {}_b\mathcal{P}_g, {}_az_b) =$ `Top`$({}_a\mathcal{P}_g)$
36 **return** ${}_a\sigma_b +$ `Solve`$({}_b\mathcal{P}_g)$

Algorithm 3: Sparse Graph Algorithm (Part 3)

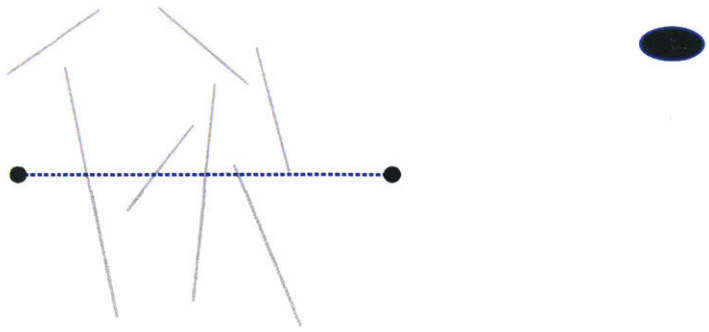
(7), (8). This operation only effects the sub-problems directly involved in the current best solution.

The third major component of the algorithm, the use of "inconsistent" sub-problems to restrict growth of the sub-problem is never directly used in this demonstration, however, it can be seen in the final tree as part of ${}_5\mathcal{P}_g$.

The most important takeaway from the algorithm is the sparsity of the search tree, which can be seen in three places. First, we can see that for the first 8 steps of the algorithm, the left half of the search tree (corresponding to moving below the first obstacle) remains a single sub-problem with awareness of a single obstacle, while the right half (corresponding to moving above the first obstacle) has become aware of three obstacles and added many more sub-problems. We are able to do this because the "best case scenario" for moving below the first obstacle has a higher cost than going above the obstacle with a more complex map. At step (9), however, the heuristic cost of the right hand side of the tree has been increased to the point where exploring the left hand side of the tree becomes necessary.

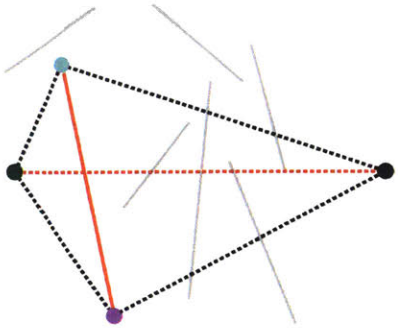
The second important element can be seen in the final search tree in step (11). Here we see that the sub-problem on the right hand side of the tree corresponding to $\mathbf{x}_5$ is aware of the third obstacle and therefore has sub-problems to $\mathbf{x}_6$ and $\mathbf{x}_7$, while the same sub-problem on the left hand side of the tree has no sub-problems. This corresponds to the "consistency" for the problem, where on the right hand side the problem is fully expanded, and ${}_3z_5 = 1$, while on the left hand side the problem is not and ${}_2z_5 = 0$. By not expanding ${}_5\mathcal{P}_g$ fully on the left hand side, we avoid the need to add sub-problems from $\mathbf{x}_2$ to $\mathbf{x}_6$ and $\mathbf{x}_7$.

Finally, we can see that only 3 out of 7 obstacles in the world directly effected the optimal solution, and the extra 4 never entered the search space, further decreasing the size of the search tree.

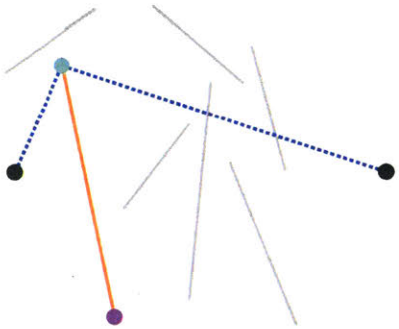


(1) Initialize the tree (Line 3) with ${}_0\mathcal{P}_g$, *i.e.* the best case no obstacle scenario

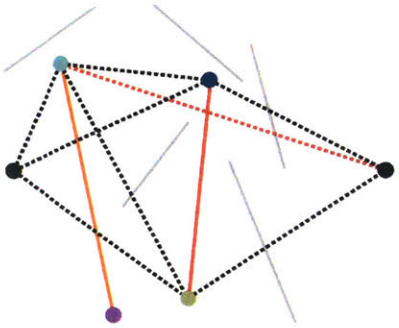
(2) Sense the blocking obstacle and add new candidate sub-problems that avoid the blocked obstacle (Line 12)



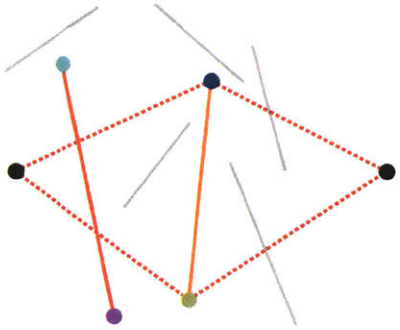
(3) Select the new best branch of the tree, ${}_0\mathcal{P}_g \rightarrow {}_3\mathcal{P}_g$

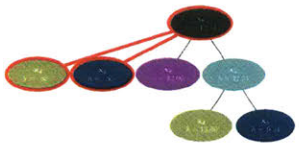


(4) Sense the blocking obstacle, and add new candidate sub-problems that avoid the blocked obstacle to the effected sub-problems ${}_0\mathcal{P}_g$ and ${}_0\mathcal{P}_3$ (Line 12). Note that the unused problem ${}_2\mathcal{P}_g$ is unaffected.

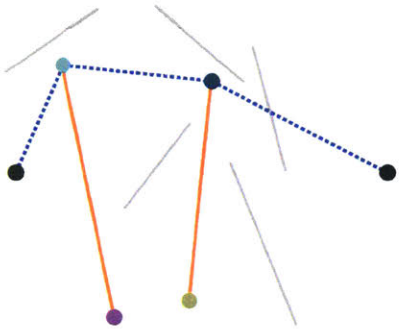


(5) Candidate top solutions ${}_0\mathcal{P}_g \rightarrow {}_4\mathcal{P}_g$ and ${}_0\mathcal{P}_g \rightarrow {}_5\mathcal{P}_g$ are blocked by already sensed obstacles, so they are removed (Line 8).

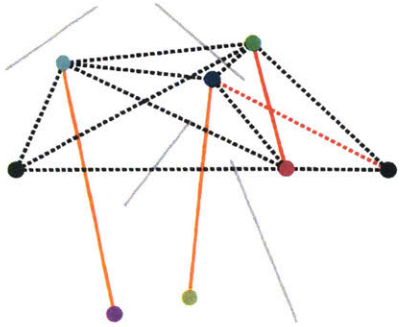


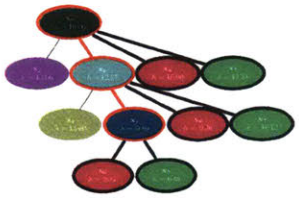


(6) Select the new best branch of the tree, ${}_0\mathcal{P}_g \rightarrow {}_3\mathcal{P}_g \rightarrow {}_5\mathcal{P}_g$

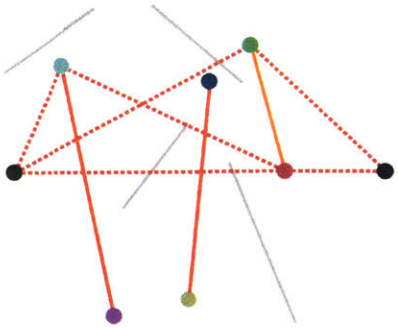


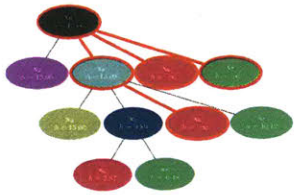
(7) Sense the blocking obstacle, and add new candidate sub-problems that avoid the blocked obstacle to the effected sub-problems ${}_0\mathcal{P}_g, {}_0\mathcal{P}_3, {}_0\mathcal{P}_5$ (Line 12)



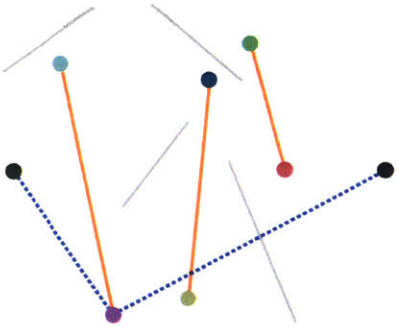


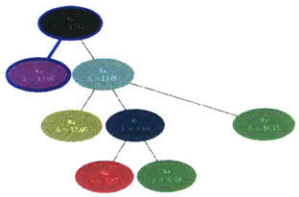
(8) Candidate top solutions ${}_0\mathcal{P}_g \rightarrow {}_6\mathcal{P}_g$, ${}_0\mathcal{P}_g \rightarrow {}_3\mathcal{P}_g \rightarrow {}_6\mathcal{P}_g$, and ${}_0\mathcal{P}_g \rightarrow {}_7\mathcal{P}_g$ are blocked by already sensed obstacles, so they are removed (Line 8).



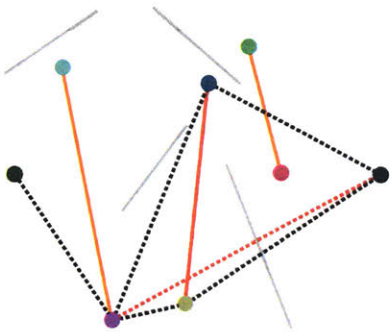


(9) Select the new best branch of the tree, ${}_0\mathcal{P}_g \rightarrow {}_2\mathcal{P}_g$. Note that up until this point in the algorithm, no branches have been expanded on this side of the tree.

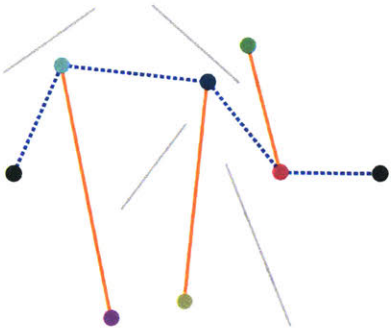




(10) ${}_0\mathcal{P}_g \rightarrow {}_2\mathcal{P}_g$ is found to be blocked against a known obstacle, and new avoiding branches are added to the tree (Line 8). Note that the blocking obstacle has been known since step 3 in the walkthrough, but was not found to be necessary to include in this branch of the tree until this step due to the use of heuristics.



(11) The optimal solution is found and verified (Line 12). Note that the search tree remains unbalanced as only promising areas were explored, and that 4 out of 7 obstacles remain unmapped as they did not effect the optimal solution.



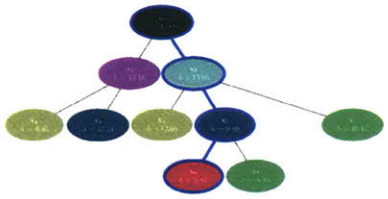


Table 3.1: Walkthrough of Algorithm 1. The left side shows the motion planning environment for a 2D holonomic robot with gray (unmapped) and orange (mapped) lines denoting impassible obstacles, while the right side shows the motion planning tree. Colored dots on the left side correspond with the origin states of problems on the right side.

3.5 Algorithm Analysis

This section will provide the proofs for the optimality and completeness of Algorithm 1, (Theorem 6 - Theorem 9) through a series of Theorems and Lemmas.

As a starting point for the algorithm, we use the *a priori* knowledge of what form an optimal trajectory will take. Unlike many common algorithms (grid search, randomized methods, gradient decent), we will directly take into account our knowledge of the form of the final solution and only search over trajectories that meet this criteria.

Theorem 10 (Structure of optimal trajectories). *Let* ${}^{\mathcal{M}}_{start}\sigma^*_{goal}$ *be a solution to the optimal motion planning problem, Problem 1.* ${}^{\mathcal{M}}_{start}\sigma^*_{goal}$ *must have the form*

$$ {}^{\mathcal{M}}_{start}\sigma^*_{goal} = \sum_{i=0}^{n-1} {}^{i}\hat{\sigma}_{i+1},\ n \geq 1 $$

where

$$ \begin{cases} \mathbf{x}_0 = \mathbf{x}_{start} \\ \mathbf{x}_i \in \mathcal{B}(\mathcal{M}) \quad \textit{for all } i \in [1, n-1] \\ \mathbf{x}_n = \mathbf{x}_{goal} \end{cases} $$

Proof. The proof follows directly from a more constrictive Theorem 25 from Pontryagin's *Mathematical Theory of Optimal Processes* [85].

"Let the optimal trajectory [of Eqn. (3.1)] lie wholly in the closed domain [$\mathcal{X}_{\text{free}}$] and contain a finite number of points of abutment [entrances to the boundary], and let every piece of it that lies on the boundary of G be regular. Then every piece of trajectory in the open kernel of [$\mathcal{X}_{\text{free}}$] (with the possible exception of its ends) satisfies the [minimum] principle; every piece lying on the boundary of [$\mathcal{X}_{\text{free}}$] satisfies MTOP-Theorem 22; and the jump condition (MTOP-Theorem 24) is satisfied at every point of abutment." □

In plain text, this means that the only reason for an optimal trajectory to deviate from

its "locally" optimal trajectory is at the boundary point of configuration space.

Based off this fact, we can break the full motion planning problem down into a two step solution; moving from the origin to one point on the boundary of configuration space and then moving from that boundary point to the goal. This makes the first level of the tree structure used throughout the algorithm.

The next step in the process is to view the subsequent sub-problems of moving from the boundary of configuration space to the goal in the same manner as the original problem. This ends up creating a decision tree of locally minimal trajectories between intermediate states, each of which is on the boundary of configuration space. Based off of Theorem 10 and Lemma 11 we can then state that the minimal path down the resulting tree will be the optimal trajectory.

Lemma 11 (Recursive optimal trajectories). *The optimal motion planning problem, Problem 1, can be broken into two discrete steps: (1) a free space solution to Pontryagin's minimum principle from $\mathbf{x}_{start}$ to some state $\mathbf{x}_i$ in the set $\{\mathcal{B}(\mathcal{M}) \cup \mathbf{x}_g\}$, and (2) solving a secondary optimal motion planning problem from $\mathbf{x}_i$ to $\mathbf{x}_{goal}$ on $\mathcal{M}$. The original optimal motion planning problem can then be solved by taking the minimum of cost across all candidate locations of $\mathbf{x}_i$. This can be written as:*

$$\mathcal{C}\left({}^{\mathcal{M}}_{s}\sigma^{*}_{g}\right) = \min_{\mathbf{x}_i \in \{\mathcal{B}(\mathcal{M}) \cup \mathbf{x}_g\}} \mathcal{C}\left({}^{\emptyset}_{s}\Sigma'_{i}\right) + \mathcal{C}\left({}^{\mathcal{M}}_{i}\sigma^{*}_{g}\right)$$

Proof. This Lemma follows directly from Theorem 10, the optimal trajectory on $\mathcal{M}$ is either a locally optimal trajectory directly from $\mathbf{x}_s$ to $\mathbf{x}_g$, or the optimal solution passes through at least one point bound on the boundary of $\mathcal{M}$. □

As described in Section 3.4, solving for the optimal solution in this full decision tree requires solving a dense graph with N nodes. Instead we will use Theorem 12 to reduce the intermediate states and trajectories that must be considered.

Theorem 12 (Sub-map Super Optimality). *If a map $\mathcal{M}_1$ is a sub-map of another map $\mathcal{M}_2$, meaning that all obstacles within $\mathcal{M}_1$ are also in $\mathcal{M}_2$, then solving the optimal motion planning problem from $\mathbf{x}_a$ to $\mathbf{x}_b$ on $\mathcal{M}_1$ will always find a solution with cost less than or*

equal to the cost of solving the same motion planning problem on $\mathcal{M}_2$, i.e.*:*

$$\begin{aligned} &\textit{If,} && \mathcal{M}_1 \subseteq \mathcal{M}_2, \\ &\textit{then,} && \mathcal{C}\left({}^{\mathcal{M}_1}_{a}\sigma^*_b\right) \leq \mathcal{C}\left({}^{\mathcal{M}_2}_{a}\sigma^*_b\right). \end{aligned} \tag{3.6}$$

Proof. Let ${}^{\mathcal{M}}_{a}\Sigma^{blocked}_{b}$ be the elements of ${}_{a}\Sigma_{b}$ that are invalid due to $\mathcal{M}$, *i.e.* ${}^{\mathcal{M}}_{a}\Sigma^{blocked}_{b} = {}_{a}\Sigma_{b} \setminus {}^{\mathcal{M}}_{a}\Sigma_{b}$. By definition, this means that for all trajectories $\sigma \in {}^{\mathcal{M}_1}_{a}\Sigma^{blocked}_{b}$, there is some time $t \in [0,\tau)$ such that $\sigma(t) \in \mathcal{M}_1$. If $\mathcal{M}_1 \subseteq \mathcal{M}_2$ and $\sigma(t) \in \mathcal{M}_1$, then $\sigma(t) \in \mathcal{M}_2$, therefore $\sigma \in {}^{\mathcal{M}_2}_{a}\Sigma^{blocked}_{b}$. Since all elements of ${}^{\mathcal{M}_1}_{a}\Sigma^{blocked}_{b}$ are in ${}^{\mathcal{M}_2}_{a}\Sigma^{blocked}_{b}$, ${}^{\mathcal{M}_1}_{a}\Sigma^{blocked}_{b} \subseteq {}^{\mathcal{M}_2}_{a}\Sigma^{blocked}_{b}$, therefore:

$${}^{\mathcal{M}_1}_{a}\Sigma_{b} = {}_{a}\Sigma_{b} \setminus {}^{\mathcal{M}_1}_{a}\Sigma^{blocked}_{b} \supseteq {}_{a}\Sigma_{b} \setminus {}^{\mathcal{M}_2}_{a}\Sigma^{blocked}_{b} = {}^{\mathcal{M}_2}_{a}\Sigma_{b} \tag{3.7}$$

Label a minimum cost element of ${}^{\mathcal{M}_2}_{a}\Sigma_{b}$ as ${}^{\mathcal{M}_2}_{a}\sigma^*_b$. From Eqn. (3.7) ${}^{\mathcal{M}_2}_{a}\Sigma_{b} \subseteq {}^{\mathcal{M}_1}_{a}\Sigma_{b}$, therefore ${}^{\mathcal{M}_2}_{a}\sigma^*_b \in {}^{\mathcal{M}_1}_{a}\Sigma_{b}$, so:

$$\mathcal{C}\left({}^{\mathcal{M}_2}_{a}\sigma^*_b\right) \geq \mathcal{C}\left({}^{\mathcal{M}_1}_{a}\Sigma_{b}\right) = \mathcal{C}\left({}^{\mathcal{M}_1}_{a}\sigma^*_b\right) \tag{3.8}$$

□

Theorem 12 is intuitive; adding more obstacles to the environment will never result in a shorter path. Practically, this provides us with a very important property, we may convert a complex optimal motion planning problem on a complex map $\mathcal{M}_2$ into a simple motion planning problem on a simple map $\mathcal{M}_1$ (already processed data), and a complex validity check on $\mathcal{M}_2$ (all data).

Combining Lemma 11 and Theorem 12 gives us a new way to search the tree:

Lemma 13 (Sub-problem super optimality)**.** *An admissible heuristic for the cost to solve the optimal motion planning problem, Problem 1, can be found by breaking the problem into two discrete steps: (1) a free space solution to Pontryagin's minimum principle from* $\mathbf{x}_{start}$ *to some state* $\mathbf{x}_i$ *in the set* $\{\mathcal{B}(\mathcal{M}) \cup \mathbf{x}_g\}$*, and (2) solving a secondary optimal motion planning problem from* $\mathbf{x}_i$ *to* $\mathbf{x}_{goal}$ *on* $\mathcal{M}_i$ *where* $\mathcal{M}_i \subseteq \mathcal{M}$*. The admissible heuristic can be found by taking the minimum of cost across all candidate locations of* $\mathbf{x}_i$*. This can be written as:*

$$\begin{aligned} &\textit{If,} && \mathcal{M}_i \subseteq \mathcal{M}, \textit{ for all } i \in [1, n-1] \\ &\textit{then,} && \mathcal{C}\left({}^{\mathcal{M}}_{s}\sigma^*_g\right) \geq \min_{\mathbf{x}_i \in \{\mathcal{B}(\mathcal{M}) \cup \mathbf{x}_g\}} \mathcal{C}\left({}^{\emptyset}_{s}\Sigma'_i\right) + \mathcal{C}\left({}^{\mathcal{M}_i}_{i}\sigma^*_g\right) \end{aligned} \tag{3.9}$$

Proof. From Lemma 11, there is some "joint state" $\mathbf{x}_j \in \{\mathcal{B}(\mathcal{M}) \cup \mathbf{x}_g\}$ such that $\mathcal{C}\left({}^{\mathcal{M}}_{s}\sigma^*_g\right) = \mathcal{C}\left({}^{\emptyset}_{s}\Sigma'_j\right) + \mathcal{C}\left({}^{\mathcal{M}}_{j}\sigma^*_g\right)$. Using this same joint state $\mathbf{x}_j$ there is an element in the right hand side of Eqn. (3.9) with cost: $\mathcal{C}\left({}^{\emptyset}_{s}\sigma^*_j\right) + \mathcal{C}\left({}^{\mathcal{M}_j}_{j}\sigma^*_g\right)$. From Theorem 12, we know that if $\mathcal{M}_j \subseteq \mathcal{M}$, then $\mathcal{C}\left({}^{\mathcal{M}_j}_{j}\sigma^*_g\right) \leq \mathcal{C}\left({}^{\mathcal{M}}_{j}\sigma^*_g\right)$, therefore:

$$\mathcal{C}\left({}^{\mathcal{M}}_{s}\sigma^*_g\right) = \mathcal{C}\left({}^{\emptyset}_{s}\sigma^*_j\right) + \mathcal{C}\left({}^{\mathcal{M}}_{j}\sigma^*_g\right) \geq \mathcal{C}\left({}^{\emptyset}_{s}\sigma^*_j\right) + \mathcal{C}\left({}^{\mathcal{M}_j}_{j}\sigma^*_g\right) \geq \min_{\mathbf{x}_i \in \{\mathcal{B}(\mathcal{M}) \cup \mathbf{x}_g\}} \mathcal{C}\left({}^{\emptyset}_{s}\sigma^*_i\right) + \mathcal{C}\left({}^{\mathcal{M}_i}_{i}\sigma^*_g\right) \tag{3.10}$$

□

It is important at this point to note that the second element of the right hand side of Eqn. (3.9) has the same form as the left hand side of Eqn. (3.9), and therefore Lemma 13 can be applied recursively.

So far, this section has considered the costs required to solve a given motion planning problem. From this point, we will move to the *Problems* that are used in Algorithm 1. We will describe what it means for a Problem to be *lower bounding* meaning that the cost to solve a given problem at its current state lower bounds the true cost to solve the problem. By being lower bounding, the cost resulting from that Problem is also an admissible heuristic.

Definition 14 (Lower bounding). *A problem, ${}_a\mathcal{P}_g$, in Algorithm 1 is defined to be* lower bounding *if, for $({}_a\sigma_i, {}_i\mathcal{P}_g, {}_az_i) = \texttt{Top}\ ({}_a\mathcal{P}_g)$, $\mathcal{C}\left({}_a\sigma_i\right) + \mathcal{C}\left({}_i\mathcal{P}_g({}_az_i)\right) \leq {}^{\mathcal{M}_{\mathcal{S}}}_{a}\sigma^*_g$. In other words, the assumed cost to move from $\mathbf{x}_a$ to $\mathbf{x}_i$ is less than the true cost, making it an admissible heuristic.*

Lemma 15 (Problem Lower bounding). *A problem ${}_a\mathcal{P}_g$ is guaranteed to be lower bounding if the following four properties hold:*

1. $\mathcal{M}_a \subseteq \mathcal{M}_{\mathcal{S}}$
2. *For all* $\left({}_a\sigma_i, {}_i\mathcal{P}_g, {}_az_i\right) \in {}_a\mathcal{P}_g \rightarrow$ *queue(),* $\cup_{j=0}^{j \leq {}_az_i} m_i(j) = \mathcal{M}_i({}_az_i) \subseteq \mathcal{M}_a$
3. *For all* $\mathbf{x}_i \in \{\mathcal{B}(\mathcal{M}_a) \cup \mathbf{x}_g\}$ *there is a child* $\left({}_a\sigma_i, {}_i\mathcal{P}_g, {}_az_i\right) \in {}_a\mathcal{P}_g \rightarrow$ *queue()*
4. *For all* $\left({}_a\sigma_i, {}_i\mathcal{P}_g, {}_az_i\right) \in {}_a\mathcal{P}_g \rightarrow$ *queue(),* ${}_i\mathcal{P}_g$ *is also lower bounding*

Proof. This lemma is a direct follow on from Lemma 13. From Theorem 12 we have that if condition (1) holds, then:

$$\mathcal{C}\left({}^{\mathcal{M}_{\mathcal{S}}}_{a}\sigma^*_g\right) \geq \mathcal{C}\left({}^{\mathcal{M}_a}_{a}\sigma^*_g\right) \tag{3.11}$$

Combining Lemma 13 with condition (2) and condition (3), we get:

$$\mathcal{C}\left({}^{\mathcal{M}_a}_{a}\sigma^*_g\right) \geq \min_{\mathbf{x}_i \in \{\mathcal{B}(\mathcal{M}_a) \cup \mathbf{x}_g\}} \mathcal{C}\left({}^{\emptyset}_{s}\sigma^*_i\right) + \mathcal{C}\left({}^{\mathcal{M}_i({}_a z_i)}_{i}\sigma^*_g\right) \tag{3.12}$$

Finally, condition (4) gives $\mathcal{C}\left({}_i\mathcal{P}_g({}_a z_i)\right) \leq \mathcal{C}\left({}^{\mathcal{M}_i({}_a z_i)}_{i}\sigma^*_g\right)$, so:

$$\min_{\mathbf{x}_i \in \{\mathcal{B}(\mathcal{M}_a) \cup \mathbf{x}_g\}} \mathcal{C}\left({}^{\emptyset}_{s}\sigma^*_i\right) + \mathcal{C}\left({}^{\mathcal{M}_i({}_a z_i)}_{i}\sigma^*_g\right) \geq \mathcal{C}\left({}_a\mathcal{P}_g\right) \tag{3.13}$$

Combining these equations results in the desired result:

$$\mathcal{C}\left({}^{\mathcal{M}_S}_{a}\sigma^*_g\right) \geq \mathcal{C}\left({}_a\mathcal{P}_g\right) \tag{3.14}$$

□

Lemma 16 (Lower bounding of Algorithm 1). *At every iteration of Algorithm 1, marked by returns to Line 5 (new iteration) or Line 14 (final solution), every problem ${}_a\mathcal{P}_g$ that exists is* lower-bounding.

Proof. We will show this through recursion, showing that at every iteration of Algorithm 1 each problem ${}_a\mathcal{P}_g$ has the four properties described in Lemma 15.

Iteration 0: At iteration 0, the only problem in Algorithm 1 is ${}_s\mathcal{P}_g$, and the only action taken on it is `Initialize(`${}_s\mathcal{P}_g$`)`. The four properties of Lemma 15 are guaranteed by:

1. $\mathcal{M}_s = \emptyset \subseteq \mathcal{M}_{\mathcal{S}}$ (Algorithm 3, Line 2)
2. The only child is $\left({}^{\emptyset}_{s}\Sigma'_g,\ {}_g\mathcal{P}_g,\ 0\right)$. $\mathcal{M}_g(0) = \emptyset \subseteq \emptyset$. (Algorithm 3, Line 3)
3. $\mathcal{B}(\emptyset) \cup \mathbf{x}_g = \mathbf{x}_g$, for which there is a child in the queue. (Algorithm 3, Line 3)
4. $\mathcal{C}\left({}_g\mathcal{P}_g\right) = 0$ by definition, so it is always (tightly) lower bounding

Iteration $\mathbf{k+1}$: Assume that every problem in Algorithm 1 is lower bounding at iteration k; then every problem in Algorithm 1 is still lower bounding at iteration $k+1$. There are two functions that expand the map for an individual problem: `SenseTrajectory()` and `UpdateProblem()`. `SenseTrajectory()` forces an expansion of the map in general, resulting in a call to `AddObstacles()`, whereas `UpdateProblem()` moves along the sequence

of submaps forcing an update due to condition (2). Once `AddObstacles()` is called, the 4 conditions are held according to:

1. All new map elements, $\mathcal{M}_{new}$ are generated from checks against $\mathcal{M}_{\mathcal{S}}$ so they are elements of $\mathcal{M}_{\mathcal{S}}$

2. This condition is only effected if a call to `UpdateProblem()` returned $\mathcal{M}_{new}$, for some problem ${}_i\mathcal{P}_g$. From the assumption that all problems were lower bounding at iteration k, $\mathcal{M}_i({}_az_i - 1) \subseteq \mathcal{M}_a$ and $\mathcal{M}_i({}_az_i) = \mathcal{M}_i({}_az_i - 1) \cup \mathcal{M}_{new}$. From (Algorithm 3, Line 23) $\mathcal{M}_a = \mathcal{M}_a \cup \mathcal{M}_{new} \supseteq \mathcal{M}_i({}_az_i - 1) \cup \mathcal{M}_{new} = \mathcal{M}_i({}_az_i)$

3. Either $\mathcal{M}_{new}$ was in $\mathcal{M}_a$ in which case this condition is already held, or the necessary children are added at (Algorithm 3, Line 27).

4. If `AddObstacles()` is called, it is called recursively for all effected problems, making any child problem also lower bounding.

□

Based off of the Theorems and Lemmas above, we will return to the original algorithm properties from Section 3.4 (restated below) and provide proofs.

Theorem 6 (Cost Optimality of Algorithm 1). *Given an optimal motion planning problem instance* $(\mathbf{x}_s, \mathbf{x}_g, \mathcal{S}, \mathcal{C}(\cdot))$, *the solution produced by Algorithm 1,* ${}_s\sigma_g^*$, *will have the property* $\mathcal{C}\left({}_s\sigma_g^*\right) = \min_{\sigma \in {}^{\mathcal{M}_{\mathcal{S}}}{}_s\Sigma_g} \mathcal{C}(\sigma)$, *where* ${}^{\mathcal{M}_{\mathcal{S}}}{}_s\Sigma_g$ *are all elements of* ${}_s\Sigma_g$ *that are valid on the map generated by fully processing* $\mathcal{S}$.

Proof. From Lemma 16 we know that at every iteration in Algorithm 1, $\mathcal{C}\left({}_s\mathcal{P}_g\right) = \mathcal{C}\left(\text{Solve}({}_s\mathcal{P}_g)\right) \leq {}^{\mathcal{M}}{}_s\sigma_g^*$. If Algorithm 1 returns a solution, then from Line 12 we know that $\text{Solve}({}_s\mathcal{P}_g) \in {}^{\mathcal{M}}{}_s\Sigma_g$, therefore $\mathcal{C}\left(\text{Solve}({}_s\mathcal{P}_g)\right) \geq \min_{\sigma \in {}^{\mathcal{M}}{}_s\Sigma_g} \mathcal{C}(\sigma) = \mathcal{C}\left({}^{\mathcal{M}}{}_s\sigma_g^*\right) \geq \mathcal{C}\left(\text{Solve}({}_s\mathcal{P}_g)\right)$. Therefore, if Algorithm 1 returns a solution, it is also an optimal solution. □

Theorem 7 (Completeness of Algorithm 1). *If there is a valid solution to the optimal motion planning problem instance* $(\mathbf{x}_s, \mathbf{x}_g, \mathcal{S}, \mathcal{C}(\cdot))$, *then Algorithm 1 will return a solution, and if there is no valid solution then Algorithm 1 will return failure in finite time.*

Proof. As stated in the problem, both $\mathcal{M}$ and $\mathcal{B}(\mathcal{M})$ are finite sets, with size $|\mathcal{M}| = N$ and $|\mathcal{B}(\mathcal{M})| = M$. Given this, there are M possible sub-problems in the tree, each of which

has M possible children (including the goal state). There are therefore M^2 possible raw trajectories in the tree. Each sub-problem may also have N possible sub-map expansions, providing MN total sub-maps in the tree.

At every run of MakeConsistent($_s\mathcal{P}_g$), one of three actions will occur:

1. An empty map will be returned, moving the algorithm to Line 12 of Algorithm 1
2. A trajectory will be found blocked at Line 5 of Algorithm 2
3. An index will be incremented at Line 7 of Algorithm 2

In case (1), the algorithm will move to Line 12 of Algorithm 1 where either a trajectory is found to be blocked, or a solution is returned by the algorithm. There are only M^2 possible trajectories in the tree, so this may occur at most M^2 times. Again, case (2) requires a trajectory to be blocked, therefore case (1) and case (2) may occur together a total of M^2 times. Case (3) will result in the advancing of an index along one sub-problem. There are M possible sub-problems, each with N possible submaps, so $\sum_{i=0}^{M-1} z_i \leq NM$. This means that case (3) may occur a maximum of NM times.

Based on these three cases, MakeConsistent($_s\mathcal{P}_g$) may be called a maximum of N^2+NM times before returning a solution. Based off of Theorem 6, if Algorithm 1 returns a solution, then it is an optimal solution, so Algorithm 1 will return a valid (finite cost) solution whenever one is available, and invalid (infinite cost) solution whenever there is no solution.

□

Theorem 9 (Graph Optimality of Algorithm 1). *For any sub-problem $_a\mathcal{P}_g$ in Algorithm 1, the submap $\mathcal{M}_a$ that the problem uses is optimally compact, meaning that it can be made no smaller while maintaining the guarantees of Theorem 6 and Theorem 7.*

Proof. The submap for any given problem, $_a\mathcal{P}_g$ is only increased by the function `AddObstacles()`. `AddObstacles()` is only called if the current best solution to $_a\mathcal{P}_g$ is blocked at `SenseTrajectory()` (the trajectory is directly blocked), `UpdateProblem()` (a sub-problem had previously found a block that had not yet been incorporated into this problem), or `MakeConsistent()` (the sub-problem was effected by one of the two conditions above). Without the addition of the new submap $\mathcal{M}_{new}$, the current best trajectory would remain optimal for all future iterations of the algorithm, despite being invalid on $\mathcal{M}_{\mathcal{S}}$, therefore $\mathcal{M}_{new}$ *must* be included in this problem. □

Theorem 8 (Sensing Optimality of Algorithm 1). *There is no alternative algorithm that can be guaranteed to require less sensing than Algorithm 1,* i.e. *result in Σ_{sensed} being smaller, while maintaining the guarantees of Theorem 6 and Theorem 7.*

Proof. Let $\mathcal{M}_{sensed,i}$ be the map discovered at the i^{th} iteration of Algorithm 1. From Theorem 6 we know that $\mathcal{C}\left(\text{Solve}({}_s\mathcal{P}_g)\right) = \mathcal{C}\left({}^{\mathcal{M}_{sensed,i}}{}_s\sigma_g^*\right)$. At Line 12 of Algorithm 1 there are two discrete options, (i) perform sensing along the current best solution $\text{Solve}({}_s\mathcal{P}_g)$ or (ii) perform sensing elsewhere. If option (ii) is taken, then $\text{Solve}({}_s\mathcal{P}_g)$ remains an option for ${}_s\mathcal{P}_g$, therefore $\mathcal{C}\left(\text{Solve}({}_s\mathcal{P}_g)\right) \leq \mathcal{C}\left({}^{\mathcal{M}_{sensed,i+1}}{}_s\sigma_g^*\right)$. From Theorem 12, we know that $\mathcal{C}\left(\text{Solve}({}_s\mathcal{P}_g)\right) = \mathcal{C}\left({}^{\mathcal{M}_{sensed,i}}{}_s\sigma_g^*\right) \leq \mathcal{C}\left({}^{\mathcal{M}_{sensed,i+1}}{}_s\sigma_g^*\right)$, so $\text{Solve}({}_s\mathcal{P}_g)$ remains the optimal solution. This will be true until option (i) is taken, therefore it is always optimal to perform sensing along the current optimal trajectory. □

3.6 Extensions

The algorithm described in the previous sections, Algorithm 1, addresses the general motion planning problem and provides the foundation for theoretical proofs of the algorithm's global optimality and completeness properties. In the following section, we will describe several extensions of the algorithm for use on robotic systems. Specifically, we look at sensor types and robot dynamics that do not naturally provide discrete obstacles with discrete boundaries (Section 3.6.1), how to update the planner as a dynamic planner to re-use information as the robot moves and new sensor data becomes available (Section 3.6.2), and how to use the planner as an anytime planner in scenarios where computation time for a globally optimal solution may overrun the time available for motion planning (Section 3.6.3).

3.6.1 Configuration Space Boundary Discretization

The algorithms and proofs so far in this chapter have worked in a primarily *map*-centric view, where additional map elements are added to the problem as they are discovered. This map focused approach provides the theoretical basis for the algorithm, and is well suited to scenarios such as the example shown in Table 3.1, where there are discrete obstacles with clear boundaries in configuration space. In more complex real world scenarios, however, this purely map based approach presents several challenges.

First, Algorithm 1 assumes a discrete and identifyable set of obstacles each of which has

a defined boundary in configuration space. For many sensor modalities such as visual stereo processing, however, a given obstacle (*e.g.* a chair) may only be revealed incrementally as more and more sensor data is processed and becomes available. Furthermore, the act of segmenting sensor data into discrete obstacles presents another non-trivial challenge.

Second, while a single *physical* object such as a chair may be clearly identified and seperated from other elements of the map, this is frequently not the case when moving from real space to configuration space. For example, a multi-link robot arm may have inaccessible elements of configuration space that are fully connected, making it difficult to identify and segment out individual obstacles from the map. As we wish to work across general robotic systems and configuration spaces, a move away from physical maps to more general configuration space techniques is required.

Third, for most real robotic systems (effectively all beyond 2D holonimic robots) the states on the boundary of a map $\mathcal{B}(\mathcal{M})$ are not finite, as the boundary surface is continuous and high dimensional. This means that we must make some approximation of the states on the boundary of an obstacle. While such a discretization is possible in a purely object based approach by approximating $\mathcal{B}(\mathcal{M})$, the methods presented below more seamlessly integrate with discretization techniques.

Finally, for higher dimensional systems, the size of the boundary of configuration space for a single obstacle may be a very large, resulting in a high branching factor within the tree. For example, for a 3D double integrator, a single physical obstacle in 3D space requires boundary states for every possible velocity of the system at every boundary (x, y, z) point of the obstacle. Given K physical boundary states to an obstacle, and M possible velocities in each dimension, the total size of the boundary set becomes KM^3.

To combat all of these problems, we change the notion of obstacles from true discrete objects, to "voxels" within a occupancy grid style map. Each voxel in the world is a candidate obstacle, however, unlike traditional grid based methods, all *free* areas of the map remain continuous. In this way we adopt a gridded approximation of *only the boundaries of configuration space* while leaving the remained of configuration space continuous. By creating a small discretization of the configuration space, only a small area of configuration space must be sensed, no distinguishing of full physical objects/objects in configuration space is required, and the number of boundary states for the voxel can be set to 2^N where N is the dimension of the configuration space. As these new voxel obstacles remain a true

"obstacle" in the plan map, the proofs and guarantees provided in Section 3.5 still hold up *to the discretization level* of the configuration space boundary.

Moving to a voxel based approach allows for moving from an obstacle-centric algorithm to a state centric algorithm. Any origin or goal state in the planned trajectory will lie on the configuration space grid, with continuous trajectories stretching through free space. Because of this we may track the states expanded in sub-problems rather than obstacles. To do this we must edit Procedure AddObstacles($_a\mathcal{P}_g$), see Algorithm 4.

1 **Procedure** `AddObstaclesDiscrete`($_a\mathcal{P}_g$, $\mathcal{M}_{new}$)
2 **for** $\mathbf{x}_i \in \mathcal{B}(\mathcal{M}_{new})$ **do**
3 **if** $\mathbf{x}_i \notin {}_a\mathcal{P}_g \rightarrow states()$ **then**
4 ${}_a\mathcal{P}_g \rightarrow states() = {}_a\mathcal{P}_g \rightarrow state() \cup \mathbf{x}_i$
5 **if** `IsNew`($_i\mathcal{P}_g$) **then**
6 `Initialize`($_i\mathcal{P}_g$)
7 `Push`($\left({}_a\sigma_i, {}_i\mathcal{P}_g, 0\right)$)
8 ${}_a\mathcal{P}_g \rightarrow substates() \rightarrow push(\mathcal{B}(\mathcal{M}_{new}))$
9 $({}_a\sigma_b, {}_b\mathcal{P}_g, {}_az_b) =$ `Top`($_a\mathcal{P}_g$)
10 ${}_ah_g(z_a) = \mathcal{C}\left({}_a\sigma_g\right) + {}_bh_g({}_az_b)$

Algorithm 4: Discrete Sparse Graph

3.6.2 Dynamic Properties

While one shot globally optimal motion planning is often of interest, this algorithm was particularly designed for real-time operation of robots as they move. The movement of the robot provides two dynamic elements: first the motion planner must account for the movement of the origin state of the motion planning problem (as the robot moves), and second it must account for new sensor data generated in between planning iterations.

A simple solution to this problem would be to re-run the full global planner from scratch at every planning instance. To prevent the problems that come with "memoryless" planners and a limited sensor range such as repeatedly trying the same dead end, already processed sensor data can be remembered within a global map for future iterations. At the next planning instance, the global map can be used as its own "sensor" determining the validity of previously explored areas in conjunction with the true sensor data.

While the above approach would frequently be viable due to the speed of the motion planning algorithm, it fails to take advantage of the already generated search tree at time

step i, which is still largely applicable at time step $i+1$. To adapt Algorithm 1 to work as a dynamic process, we look to the example of D* Lite [52]. Unlike D* Lite, the algorithm described below is meant only for processes in which unknown or free areas of the world are dynamically changed from free to occupied (*i.e.* edge costs may only increase, never decrease). Such a scenario is well suited to the problem of partially processing sensor data, where the primary addition is newly occupied areas. In the advent that edge costs are required to decrease, a re-building of the tree would be required based off of the assumptions of always maintaining a sub-map up the tree. The primary take away used here from D* Lite is the use of a heuristic offset, k_m to prevent a full resorting of the priority queue.

The structure of the tree maintained by Algorithm 1 is well suited for dynamic re-use as the robot moves because there is only one layer of the tree that is effected by the movement of the robot, the top (${}_s\mathcal{P}_g$). Additionally, as soon as the robot reaches one of the joint states, the top of the tree can be "popped" and we are left with a new complete and consistent tree. In this way the tree can constantly be edited for the movement of the robot and newly available sensor data, while "resetting" the tree at every instance that a joint state is reached. The resetting of the tree prevents a continuous growth in the size of the tree as many motion planning iterations are run on the same tree.

The Dynamic algorithm is described in Algorithm 5, describing both the movement procedure of the robot, and changes to the original algorithm. Specifically, where `SenseTrajectory()`, `Push()`, and `Top()` were called in Algorithm 1, now `D-SenseTrajectory()`, `D-Push()`, and `D-Top()` will be called. The primary additions to Algorithm 1 are the pruning and updating of the tree, and maintaining a heuristic value k_m for trajectories already in the queue. Note that because the origin state has moved in the motion planning problem, all trajectories that were found blocked from the origin must be regenerated from the new state as they may have become free. This regeneration is in contrast to gridded graph based approaches where the robot state typically takes new values only at already existing nodes in the graph.

3.6.3 Anytime Properties

In addition to the Dynamic properties given by Algorithm 5, real time systems are also frequently in need of "Anytime" algorithms, *i.e.* algorithms that can provide a solution on demand. Anytime algorithms allow for control systems to request a control action from the planning module whenever required. Traditionally "Anytime" algorithms are algorithms

Result: $\sigma^* = \arg\min_{\sigma \in \mathcal{M}_{s\Sigma_g}} \mathcal{C}(\sigma)$

Algorithm D-SparseShortestPath($\mathbf{x}_s$, $\mathbf{x}_g$)
  $\mathbf{x}_{last} = \mathbf{x}_s$
  $k_m = 0$
  Initialize($_s\mathcal{P}_g$)
  $_s\sigma_g^* =$ SparseShortestPath($_s\mathcal{P}_g$)
  while $\mathbf{x}_s$ not(=) $\mathbf{x}_g$ **do**
    $\mathbf{x}_s =$ Advance($_s\mathcal{P}_g$)
    /* Sensor data updated */
    $\mathcal{M}_{new} =$ SenseTrajectory($_s\sigma_g^*$, $\mathcal{M}_{\mathcal{S}}$)
    /* Our last best solution wasn't clear, so we have to resolve again */
    if ~IsEmpty($\mathcal{M}_{new}$) **then**
      /* Update cost offset for $_s\mathcal{P}_g$ queue */
      UpdateTree($_s\mathcal{P}_g$)
      $_s\sigma_g^* =$ SparseShortestPath($_s\mathcal{P}_g$)
Procedure Advance($_s\mathcal{P}_g$)
  $(_s\sigma_a, {}_a\mathcal{P}_g, {}_sz_a) =$ Top($_s\mathcal{P}_g$)
  $t = t + \Delta t$
  if $t <$ $_s\tau_a$ **then**
    /* Still in the trajectory, so advance */
    $\mathbf{x}_s = {}_s\sigma_a(t)$
  else
    /* Reached a node point, prune the tree */
    $\mathbf{x}_s = \mathbf{x}_a$
    $\mathbf{x}_{last} = \mathbf{x}_s$
    $k_m = 0$
    $t = 0$
    delete $_s\mathcal{P}_g$
    $_s\mathcal{P}_g = {}_a\mathcal{P}_g$
  return $\mathbf{x}_s$
Procedure UpdateTree($_s\mathcal{P}_g$)
  $k_m = \mathcal{C}\left({}^{\emptyset}_{s}\sigma^*_{last}\right)$
  $t = 0$
  /* Push back in previously blocked paths */
  for $(\mathbf{x}_i, {}_sz_i) \in {}_s\mathcal{P}_g \rightarrow$ *blocked()* **do**
    D-Push($_s\mathcal{P}_g$, $\left({}^{\emptyset}_{s}\Sigma'_g, {}_i\mathcal{P}_g, {}_sz_i\right)$)

Algorithm 5: Dynamic Sparse Shortest Graph

that quickly converge to a valid sub-optimal solution, and then incrementally improve the optimality of the solution with more computation time. Such systems allow for the full

1 **Procedure** D-Push($_a\mathcal{P}_g$, $\left({}_a\sigma_b, {}_b\mathcal{P}_g, {}_az_b\right)$)
2 priority = $\mathcal{C}\left({}_a\sigma_b\right) + {}_bh_g({}_az_b) + k_m$
3 ${}_a\mathcal{P}_g \rightarrow$ queue() $\rightarrow$ push($\left({}_a\sigma_b, {}_b\mathcal{P}_g, {}_az_b\right)$, priority)
4 **Procedure** D-Top($_a\mathcal{P}_g$)
5 $\left({}_a\sigma_b, {}_b\mathcal{P}_g, {}_az_b\right)$ = Top($_a\mathcal{P}_g$)
/* Cycle through queue until we find one with correct origin point */
6 **while** ${}_a\sigma_b(0)$ not(=) $\mathbf{x}_a$ **do**
7 Pop($_a\mathcal{P}_g$)
8 D-Push($_a\mathcal{P}_g$, $\left({}^{\emptyset}_{a}\Sigma'_b, {}_b\mathcal{P}_g, {}_az_b\right)$)
9 $\left({}_a\sigma_b, {}_b\mathcal{P}_g, {}_az_b\right)$ = Top($_a\mathcal{P}_g$)

Algorithm 6: Dynamic Push and Pop

generated motion plan to be executed safely without recomputing a new motion plan. These solutions have the advantage of providing a fully valid solution at any point in time, however, they are somewhat agnostic to the fact that another re-plan will occur in the near future with a dynamic system. Practically, anytime solutions are typically recalculated at a much higher rate as every iteration is used within a control loop. In such a scenario, the true goal of an anytime algorithm should be to provide a solution that is both valid and near optimal *for the next time step.*

In that light we describe Algorithm 7, which tracks the *super-optimal* solution to the motion planning problem that is valid for some time horizon $t_{horizon}$. Unlike a typical anytime solution, these super-optimal solutions are continually *increasing* in cost as more computation time is given, until the planning algorithm reaches the optimal solution and terminates. The guarantee of $t_{horizon}$ provides a safety requirement that the currently generated control actions will be safe, without requiring a check on further actions. We posit, without proof, that the initial step in a *super-optimal* solution will bear a closer resemblance to the initial step in the optimal solution than the initial step in the *sub-optimal* solution generated by traditional anytime algorithms. Empirically, because of the structure of the search tree, early elements of the *super-optimal* solution tend to get settled "earlier" in the motion planning process, and further computation tends to effect later steps more than early steps.

Result: ${}_s\sigma_g^{\sim}$

1 **Algorithm** A-SparseShortestPath($\mathbf{x}_s$, $\mathbf{x}_g$, $t_{horizon}$)

/* Initialize the problem */

2 free = False

3 ${}_s\sigma_g^{\sim} = null$

4 Initialize(${}_s\mathcal{P}_g$)

5 **while** not *free* **do**

6 consistent = False

/* Update the graph given currently processed data */

7 **while** not *consistent* **do**

8 $\mathcal{M}_{new}$ = MakeConsistent(${}_s\mathcal{P}_g$)

9 consistent = IsEmpty($\mathcal{M}_{new}$)

/* If the best solution on current data is valid on all data for $t_{horizon}$ make it the anytime solution */

10 **if** Solve(${}_s\mathcal{P}_g$)$(0, t_{horizon})$ *valid on* $\mathcal{M}_{\mathcal{S}}$ **then**

11 ${}_s\sigma_g^{\sim}$ = Solve(${}_s\mathcal{P}_g$)

/* Check the best solution against all data */

12 $\mathcal{M}_{new}$ = Sense(${}_s\mathcal{P}_g$)

13 free = IsEmpty($\mathcal{M}_{new}$)

/* Return the unblocked solution */

14 **return** Solve(${}_s\mathcal{P}_g$)

Algorithm 7: Anytime Sparse Shortest Graph

3.7 Conclusions

In this chapter a motion planning algorithm is described which couples the motion planning and mapping processes within the robot motion planning pipeline. The motion planning algorithm is built around a tree of locally optimal, continuous trajectories which are joined at the boundaries of configuration space. Incremental mapping is used to drive the growth of the tree, providing a direct coupling between the motion planning and mapping processes: the motion planning algorithm drives mapping using traditional "lazy" planning techniques, while the results of the mapping step are used to drive local expansion of the search space. In this manner, an efficient and dynamics agnostic planning-and-mapping algorithm, Algorithm 1, is formed. This algorithm is shown to be optimal (Theorem 6) and complete (Theorem 7), while minimizing the mapped areas (Theorem 8) and the motion planning search space (Theorem 9).

Chapter 4

Applications

In this chapter, we build off of the theoretical results from Chapter 3 to test Algorithm 1 on several common robotic dynamical systems through Monte Carlo simulations. In the first section, Section 4.1, tests are performed with several canonical low dimensional systems to demonstrate baseline performance, while in Section 4.2, Algorithm 1 is applied to the high dimensional and relatively unsolved problem of optimal quadrotor motion planning.

In both cases, comparisons are made between the proposed algorithm, Algorithm 1, and Lazy D* Lite search [52, 7], comparing the quality of the solution (trajectory cost) and the computation time to get a solution. Lazy D* Lite was chosen for comparison due to its ability to satisfy the joint optimization problem of minimizing sensing and finding an optimal solution. To fully minimize sensing, it is necessary to perform mapping only along the current best solution [16], *e.g.* mapping such that if all checks come up free the motion planning process can terminate. This requirement rules out sampling and optimization based planning methods that use lazy checking *on sample* or *on iteration*, which while limiting sensing, does not fully minimize it. While both Algorithm 1 and Lazy D* are optimal cost algorithms, they are optimal *within the structure of the graph* meaning that different choices of graph structure effect the cost of the resulting trajectory. As Algorithm 1 treats the open elements of configuration space as continuous, while grid based methods use a discrete approximation in the plan graph we can expect lower cost trajectories to result from Algorithm 1.

The section of quadrotor planning contains two parts, first the derivation of a fast and near-optimal method of solving the boundary value problem for a three dimensional coupled

double integrator, and second the application of this solution for 6-DOF globally near-optimal motion planning for a quadrotor. To the best of our knowledge this if the first such system capable of near-optimal motion planning for a coupled double integrator with computation appropriate for real-time use.

4.1 Test Systems

As an initial test on the capabilities of Algorithm 1, we compared its performance on three simple canonical systems: a 2D holonomic robot, a 3D holonomic robot, and a Dubins car. Each of these systems can easily be implemented in grid based Lazy D* Lite for exact comparison between the different motion planners. While improved performance on these simple systems is not the goal of a new motion planner, it provides a baseline of performance that can be further expanded upon for higher dimensional and more complex systems where grid based planning methods struggle.

4.1.1 Simulation Setup

Computational experiments were performed through Monte Carlo simulations in randomly generated obstacle fields in $\mathbb{R}^2$ (Holonomic 2D and Dubins Car) and $\mathbb{R}^3$ (Holonomic 3D). For the 2D holonomic robot and Dubins car, obstacles are impassible line segments of a fixed length and random orientation distributed randomly in $[0, 30]$x$[0, 30] \in \mathbb{R}^2$. For the 3D holonomic robot, obstacles are impassible cubes of fixed side length randomly distributed in $[0, 30]$x$[0, 30]$x$[0, 30] \in \mathbb{R}^3$. The starting location is $\mathbf{x}_s = (5, 5, 5)$ and the goal location was set 20 units away at a random direction in the positive quadrant. The goal location is rounded to the nearest integer to allow for easy integration with grid based methods. For Dubin's cars the initial and final orientations were randomly generated increments of $\pi/2$, again to allow for easy integration with grid based methods. Examples of the setup can be seen in Figures 4-1 and 4-2.

4.1.2 Grid Construction

The gridded graph construction places nodes at regular intervals through the state space with a fixed spatial discretization and, for the Dubins car, angular discretization. The edges between nodes are placed based on a connectivity parameter, which determines to what level

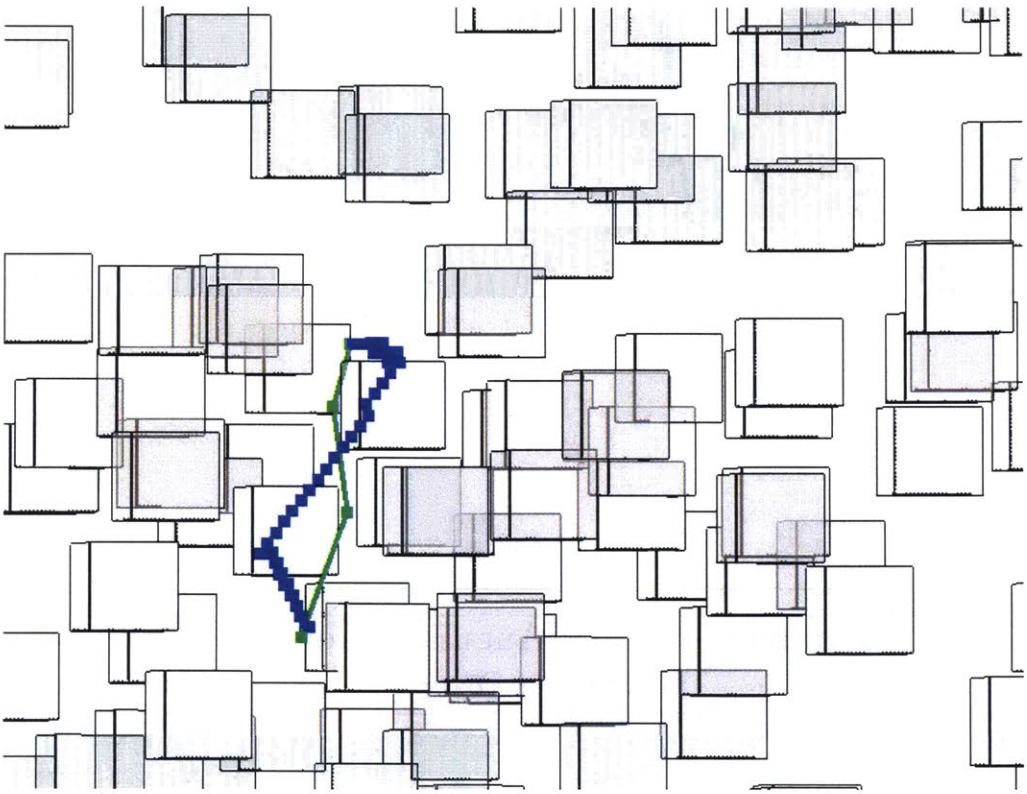
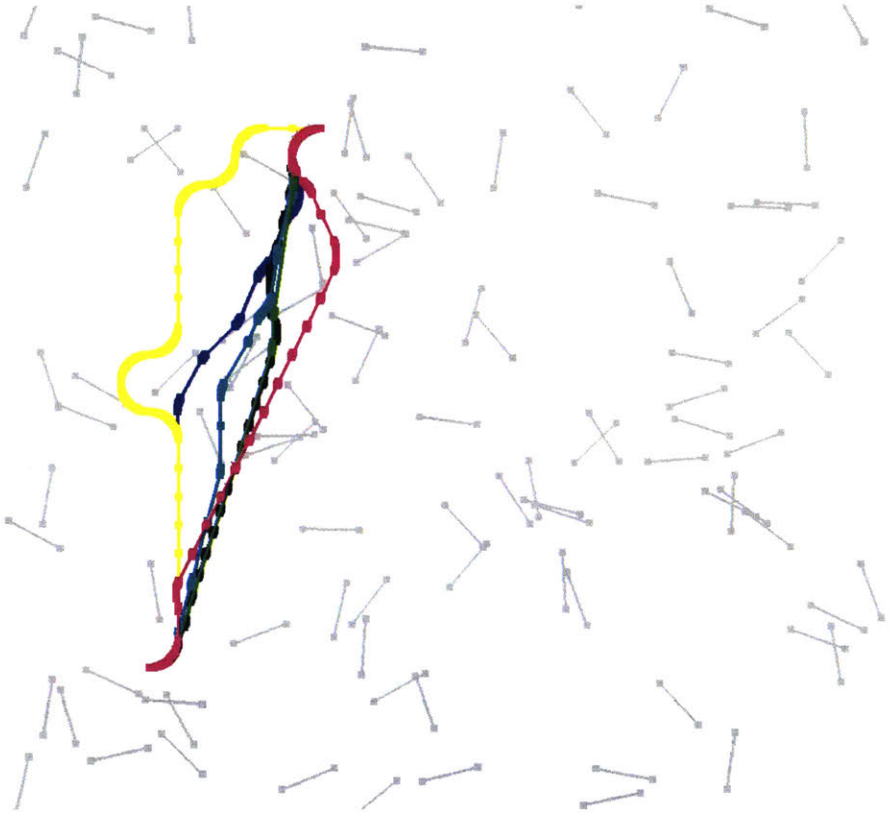
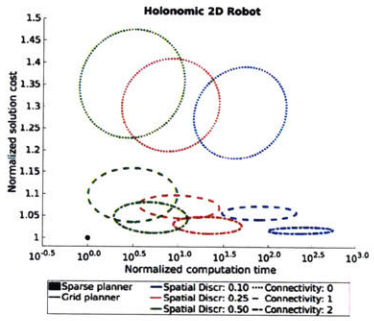


Figure 4-1: View of a 3D simulation with 200 randomly spaced cubes (gray) of length 2. The trajectory generated from a sparse plan graph is shown in green, and from a grid plan graph in blue. Both graphs use a spatial discretization of 0.25, and the grid plan graph uses a connectivity of 1 (26 connected).

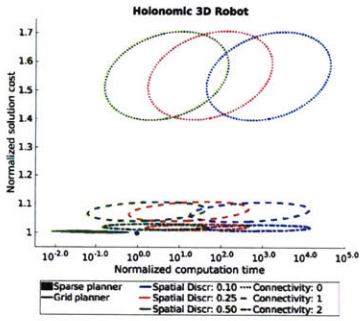


Color	Planner	Grid Discr.	Angular Discr.	Conn.	Cost	Time (ms)
Green	Sparse	–	$\pi/8$	–	**20.71**	**16.9**
Black	Grid	0.25	$\pi/8$	4	20.86	751.3
Cyan	Grid	0.5	$\pi/8$	4	20.88	486.7
Blue	Grid	1.0	$\pi/4$	2	21.43	49.1
Purple	Grid	0.5	$\pi/8$	2	21.86	110.9
Yellow	Grid	1.0	$\pi/2$	0	25.56	25.6

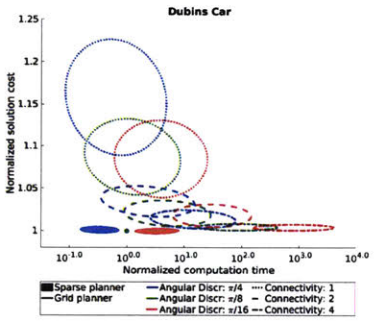
Figure 4-2: Generated trajectories of a Dubins car with turning radius 1, traveling through 100 obstacles of length 2, using 6 different plan graphs. The different graph types, their display color, and the computed trajectory cost and computation time are shown in the table.



(a) Trajectory cost vs. computation time for a holonomic 2D robot moving through 100 obstacles of length 2. Cost and trajectory length for each map is normalized by the values of planning with the sparse plan graph.



(b) Trajectory cost vs. computation time for a holonomic 3D robot moving through 200 obstacles of side length 2. Cost and trajectory length for each map is normalized by the values of planning with the sparse plan graph with spatial discretization of 0.1.



(c) Trajectory cost vs. computation time for a Dubins car moving through 100 obstacles of length 2. Cost and trajectory length for each map is normalized by the values of planning with the sparse plan graph with angular discretization of $\pi/8$.

Figure 4-3: Normalized results for trajectory cost vs. computation time for three robot types on 200 randomly generated maps. Each type of plan graph was run on the same 200 maps for each robot, and the results were normalized to show the relative speed and solution quality of the different graphs. The figures display the 95% boundary of the values for each graph type, with shape fill showing the type of plan graph (sparse or grid), color showing the discretization level, and line style showing the connectivity of the grid graphs.

of adjacent nodes a node is connected to. A connectivity level of "0" connects to adjacent non-diagonal nodes (4 connected in 2D) and a connectivity of n for $n \geq 1$ connects to all nodes within n of that node in *any* dimension. For holonomic robots any path that was a scalar duplicate of another one was trimmed. The Dubins car is connected to all discretizations of angular space within the spatial connectivity, but with a heuristic pruning of high cost (near full turn) trajectory primitives. This was found to significantly speed up computation without hurting the quality of the paths.

Planner	Grid Discretization	Angular Discretization	Connectivity	Path Cost	Plan Time (ms)	Nodes	Edges	Area Sensed
Sparse	–	$\pi/16$	–	22.315	1645	282	4838	436
Sparse	–	$\pi/8$	–	22.328	140	140	1369	418
Sparse	–	$\pi/4$	–	22.357	20	70	383	382
Grid	0.25	$\pi/16$	4	22.400	35871	49318	1101100	528
Grid	0.25	$\pi/8$	4	22.424	2541	25276	278740	513
Grid	0.25	$\pi/4$	4	22.635	258	13704	77655	522
Grid	0.50	$\pi/16$	2	22.710	715	13069	113290	544
Grid	0.50	$\pi/8$	2	22.783	78	6677	27560	546
Grid	0.50	$\pi/4$	2	23.090	29	3954	11733	449
Grid	1.00	$\pi/16$	1	24.214	55	4266	19424	417
Grid	1.00	$\pi/8$	1	25.280	15	2287	5897	403
Grid	1.00	$\pi/4$	1	25.837	7	887	1780	282

Table 4.1: Summary of mean values for planning a trajectory for a Dubins Car with turning radius of 1 amoung 100 obstacles of length 2 (see Figure 4-2). Standard deviations are omitted as individual measurements are map dependent and therefore the values do not follow a normal distribution, see Figure 4-3c for relative distributions.

4.1.3 Implementation

Collision checks in 2D were performed using a simple line intersection check, while collision checks in 3D were performed using ray casting in Octomap [40]. Each collision checker acts as a simulated sensor, moving along the trajectory mapping free or occupied space. Edges for collision checking along the current "best path" are selected starting from the robot's current pose and continuing forward towards the goal. This matches Lazy Weighted A* [13] and the forward edge selector described by Dellin and Srinivasa [16]. As suggested by Dellin and Srinivasa [16] other edge selectors are viable and can have different performance characteristics, however, that is not the focus of this work.

Both the graph created by Algorithm 1 and D* Lite [52] were implemented in C++ using the same custom planning library for collision checking, trajectory generation, and search. As we are only performing a single planning step we only use the lifelong planning

element of D* Lite, with edge updates coming from the "lazy" collision checking rather than movement of the robot.

4.1.4 Results

Comparison data for the three dynamical systems is shown in Figure 4-3, comparing the length of the path generated with the time to generate the path. Since path length and computation time are highly map dependent, the values are normalized by the results from a single method and the same scenario is re-run with multiple planners/settings. As can be seen in the results, while there is a clear trade off between path quality and solution time in the grid based methods, the Algorithm 1 is significantly less sensitive to parameter choice in path length, and provides paths that are both lower in cost and faster to compute.

A summary of the data for the Dubins car experiments is shown in Table 4.1. As expected, the sparse graph construction algorithm created graphs that generated shorter paths, faster, with significantly less nodes and edges. The total area sensed was measured by discretizing the position space into voxels of size 0.2 and marking them as sensed if the mapping process moved through them. Since both methods use the same lazy forward edge selection process we do not see significant differences in the areas sensed.

4.2 Quadrotor UAV: Coupled Double Integrator

Despite the surge in usage and research on quadrotor UAV's, motion planning remains a relatively open problem due to the complex high dimensional dynamics of the system [100]. Much of quadrotor motion planning is based off of the work of Mellinger *et. al.* [72] showing that (simple) quadrotor dynamics are a differentially flat system, meaning that the system can be fully described by a subset of the states and their derivatives. Specifically, this allows for motion planning only in $\{x, y, z, \theta\}$ coordinates, where θ is the yaw of the vehicle. Additionally, Melligner *et. al.* [72] introduced solving the motion planning problem on these flat variables as minimizing the snap of the position variables and second derivative of yaw, which corresponds to minimizing control effort. The snap minimization technique is still one of the most used today, with notable updates in reformulating the problem to be numerically stable for numerous segments [88, 87], and to create a hybrid minimum snap and minimum time problem [11]. While minimum snap trajectories are relatively practical and smooth

for flight, the optimization performed is fundamentally different from the traits generally searched for in optimal motion planning: that the actuator constraints are never violated (this can be verified, but not included in the optimization), and that the trajectories are minimum time. In fact, the minimum snap trajectories assume *pre-determined* segment lengths, possibly with a secondary optimization to find these lengths. Additionally, the minimum snap formulation is relatively limited in its ability to perform obstacle avoidance, with constraints generally limited to linear flight corridors [72].

More recently, several works have followed the approach introduced by Deits and Tedrake of first solving for a viable flight corridor, and then optimizing within it [14]. Deits and Tedrake originally proposed a method of finding an overlapping set of convex volumes (IRIS) and performing a mixed integer convex optimization within them [14, 55]. Various works have followed this model, using different approaches for identifying convex regions to optimize in, waypoints to optimize between, and optimization techniques [65, 66]. While these techniques have also demonstrated effectiveness on real systems, they still prevent a significant deviation from traditional motion planning as they involve a two step process – the first is a dynamics unaware step which sets the bounds for the dynamics aware step. This two step process works well in cases where the dynamics aware path resembles the euclidean shortest path through obstacles, but can be highly sub-optimal in situations such as tight corners which would require significant slow downs for a quadrotor.

The last notable set of techniques used is to treat the differentially flat quadrotor variables as double or triple integrators with bounded velocity, acceleration, and/or jerk. These techniques allow for directly including actuator constraints within the optimization problem and solving the problem as a minimum time motion planning problem. Due to the complexity of the dynamic constraints, however, these techniques typically resort to treating the three dimensions of the quadrotor $\{x, y, z\}$ as independent problems. This allows for simple, and at times analytical, decoupled solutions, but requires artificially limiting the true actuator constraints of the quadrotor to conservative estimates along each axis [35, 77, 65, 66, 22].

In this section, we derive a technique for directly solving the boundary value problem for a *coupled* double integrator using a bang-bang approximation. This boundary value problem takes as inputs the desired initial and final positions ($\mathbf{x}_0$, $\mathbf{x}_f$) and velocities ($\mathbf{v}_0$, $\mathbf{v}_f$), and determines the approximate minimum time (and dynamically feasible) trajectory between them. The resulting solution is sub-optimal due to the bang-bang approximation, however,

we demonstrate numerically that the sub-optimality is within 5% of the true cost in 99.8% of simulations, with computation times on the order of tens of microseconds. This model provides an accurate physical representation of the forces on the quadrotor, while neglecting the inertial properties (it assumes the limited thrust vector can be reoriented arbitrarily fast) and neglecting drag effects which may limit feasible velocities.

With this boundary value solver in hand, we then apply the techniques from Chapter 3 to rapidly find *globally* near-optimal minimum time solutions for the coupled double integrator through obstacle fields. We demonstrate through Monte Carlo simulations that these solutions are significantly lower cost and faster to compute than state of the art decoupled search based techniques.

4.2.1 Coupled Double Integrator Boundary Value Problem

The coupled double integrator is a well known and well studied dynamical system. The basic system can be described as a multidimensional point mass with a limited magnitude of acceleration that may be applied to it, and in this case a fixed acceleration from gravity. Despite the prevalence of this model, there is no known analytical solution to the optimal control problem for a generic boundary value problem when the system has more than one dimension.

In this section we derive a fast numerical procedure to find a dynamically feasible solution that approximates the true optimal solution to the boundary value problem. We base this approximation off the true analytical solution for a one dimensional double integrator, which can be shown to always take the form of a "bang-bang" solution. Based off of this, we force the multidimensional system to take the same form – the boundary value problem will be solved by only taking two discrete control inputs. This technique is guaranteed to find a dynamically feasible solution, while it empirically demonstrates a close approximation of the true optimal solution.

The numerical approximation is based off of the total time T to move the double integrator system from its initial state to its goal state. In the sections that follow, we show that there are analytical upper and lower bounds to the values that T may take for a given boundary value problem, and that given a value T, the required control inputs for a "bang-bang" solution to the boundary value problem can be found analytically (though they may violate control limits). These three analytical elements are combined to find a numerical

solution using a 1D line search over the possible values of T, checking feasibility of control inputs at each value of T. In the (rare) event that no "bang-bang" solution is found within he upper and lower bounds, the analytical upper-bound on T is guaranteed to provide a dynamically feasible solution.

Coupled Double Integrator Dynamics

The state of the system is written $\mathbf{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^T$, with control inputs $\mathbf{u} = [u_x, u_y, u_z]^T$ and system dynamics:

$$\dot{\mathbf{x}} = f_{di}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{0}_{3\times3} \\ \mathbf{I}_{3\times3} \end{bmatrix} \mathbf{u} - \begin{bmatrix} \mathbf{0}_{5\times1} \\ g \end{bmatrix}, \ \|\mathbf{u}\| \leq u_{max} \tag{4.1}$$

The necessary conditions for an optimal coupled double integrator are well known and can be derived using Pontryagin's minimum principle:

$$H(\mathbf{x}, \mathbf{u}, \boldsymbol{\lambda}(t)) = 1 + \boldsymbol{\lambda}(t)^T \left[\dot{x}, \dot{y}, \dot{z}, u_x, u_y, u_z - g\right]^T \tag{4.2}$$

from the adjoint transition equation, $\dot{\boldsymbol{\lambda}}_i(t) = -\frac{\partial H}{\partial x_i}$, we can solve for the forms of the adjoint variables with constant parameters $\mathbf{c}$ [57]:

$$\boldsymbol{\lambda}(t) = [c_1, c_2, c_3, -c_1 t + c_4, -c_2 t + c_5, -c_4 t + c_6]^T \tag{4.3}$$

We can then rewrite the Hamiltonian using the constant terms $\mathbf{c}$:

$$H(\mathbf{x}, \mathbf{u}, \mathbf{c}) = 1 + c_1\dot{x} + c_2\dot{y} + c_3\dot{z} + (c_4 - c_1 t)\, u_x + (c_5 - c_2 t)\, u_y + (c_6 - c_3 t)\,(u_z - g) \tag{4.4}$$

Applying the minimum principle, $H(\mathbf{x}(t), \mathbf{u}^*(t), \boldsymbol{\lambda}(t)) \leq H(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t))$, we can solve for the form of $\mathbf{u}$ in terms of the constants $\mathbf{c}$. For notational simplicity we will replace the constants $\mathbf{c}$ with three new linear time varying values defined as:

$$\begin{aligned} A(t) &= c_4 - c_1 t \\ B(t) &= c_5 - c_2 t \\ C(t) &= c_6 - c_3 t \end{aligned} \tag{4.5}$$

To apply the minimum principle we need to solve:

$$\mathbf{u}^*(t) = \underset{\|\mathbf{u}(t)\| \leq u_{max}}{\arg\min} \; A(t)u_x(t) + B(t)u_y(t) + C(t)(u_z(t) - g) \tag{4.6}$$

Substituting $u_z(t) = \pm\sqrt{u_{max}^2 - u_x^2 - u_y^2}$, ($u_z$ chosen arbitrarily, another element may be selected if $u_z = 0$) gives:

$$\mathbf{u}^*(t) = \underset{\mathbf{u}(t)}{\arg\min}\, A(t)u_x(t) + B(t)u_y(t) + C(t)(\sqrt{u_{max}^2 - u_x^2 - u_y^2} - g) \tag{4.7}$$

which we can find the minimum for by solving $\frac{\partial H}{\partial u_x} = 0$, $\frac{\partial H}{\partial u_x} = 0$:

$$A(t) \mp \frac{C(t)u_x(t)}{\sqrt{u_{max}^2 - u_x^2 - u_y^2}} = 0 \tag{4.8}$$

$$B(t) \mp \frac{C(t)u_y(t)}{\sqrt{u_{max}^2 - u_x^2 - u_y^2}} = 0 \tag{4.9}$$

rearranging, and squaring both sides gives:

$$A^2(t)(u_{max}^2 - u_x^2 - u_y^2) = C^2(t)u_x^2(t) \tag{4.10}$$

$$B^2(t)(u_{max}^2 - u_x^2 - u_y^2) = C^2(t)u_y^2(t) \tag{4.11}$$

which can be solved for $\mathbf{u}^*(t)$:

$$\mathbf{u}^*(t) = \frac{\pm u_{max}}{\sqrt{A^2(t) + B^2(t) + C^2(t)}} \begin{bmatrix} A(t) \\ B(t) \\ C(t) \end{bmatrix} \tag{4.12}$$

For the sake of optimal motion planning we are trying to solve the problem:

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & t^*, \\ \text{subject to} \quad & \dot{\mathbf{x}} = f_{di}(\mathbf{x}, \mathbf{u}), \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & \mathbf{x}(t^*) = \mathbf{x}_f. \end{aligned} \tag{4.13}$$

From Eqn. (4.12) we know the necessary structure of the solution to Eqn. (4.13), and this structure can be single and double integrated to explicitly find the solution to f_{di}. The resulting analytical problem will have 13 free variables, $\mathbf{c}$ (6), integration constants (6), and t^*, with 12 constraint equations from $\mathbf{x}(0) = \mathbf{x}_0$ and $\mathbf{x}(t^*) = \mathbf{x}_f$, however, it can be seen from the structure of Eqn. (4.12) that $\mathbf{c}$ can be scaled by an arbitrary scalar value and produce the same result, leaving 12 free variables and 12 constraint equations. No known analytical technique for directly solving this optimal control problem exists, so rather than directly solving this system we will solve a much simpler approximation of the solution.

Bang-bang approximation

Rather than solving for the exact solution to Eqn. (4.12), we will instead derive a fast-to-compute approximation that still satisfies the constraints of the problem. Our new solution will take a "bang-bang" formulation inspired from the optimal control strategy for a 1D double integrator:

$$\tilde{\mathbf{u}}^*(t) = \begin{cases} \mathbf{u}_1 & \text{for all } t \in [0, t_1); \\ \mathbf{u}_2 & \text{for all } t \in [t_1, t_1 + t_2), \end{cases} \tag{4.14}$$

where $\mathbf{u}_1$ and $\mathbf{u}_2$ are constant control inputs. Though multidimensional double integrators do not share the same bang-bang properties as a one dimensional system, as can be seen in Figure 4-4 the multidimensional system still follows the same pattern of one direction of control effort and then another. This heuristic observation drives the approximation shown below, which is later empirically demonstrated to be valid. Because the control inputs are not time varying, we can easily analytically solve the system dynamics Eqn. (4.1) and set boundary conditions on the position ($\mathbf{x}_0$, $\mathbf{x}_f$) and velocity ($\mathbf{v}_0$, $\mathbf{v}_f$), giving the constrained optimization problem:

$$\begin{aligned} \underset{t_1,t_2,u_1,u_2}{\text{minimize}} \quad & t_1 + t_2 \\ \text{subject to} \quad & t_1 \geq 0,\ t_2 \geq 0 \\ & \|\mathbf{u}_1\| \leq \mathbf{u}_{max},\ \|\mathbf{u}_2\| \leq \mathbf{u}_{max} \\ & \mathbf{x}_f - \mathbf{x}_0 = \mathbf{v}_0(t_1 + t_2) + \frac{1}{2}(\mathbf{u}_1 - \mathbf{g})\left(t_1^2 + 2t_1t_2\right) + \frac{1}{2}(\mathbf{u}_2 - \mathbf{g})\,t_2^2 \\ & \mathbf{v}_f - \mathbf{v}_0 = \mathbf{u}_1t_1 + \mathbf{u}_2t_2 - \mathbf{g}(t_1 + t_2) \end{aligned} \tag{4.15}$$

Assuming that $t_1 > 0$ and $t_2 > 0$ we can invert the boundary conditions to find functions $\mathbf{u}_1(t_1, t_2)$ and $\mathbf{u}_2(t_1, t_2)$:

$$\mathbf{u}_1(t_1, t_2) = \frac{2\left(\mathbf{x}_f - \mathbf{x}_0\right) - (2t_1 + t_2)\mathbf{v}_0 - t_2\mathbf{v}_f}{t_1\left(t_1 + t_2\right)} + \mathbf{g} \tag{4.16}$$

$$\mathbf{u}_2(t_1, t_2) = \frac{-2\left(\mathbf{x}_f - \mathbf{x}_0\right) + t_1\mathbf{v}_0 + \left(t_1 + 2t_2\right)\mathbf{v}_f}{t_2\left(t_1 + t_2\right)} + \mathbf{g} \tag{4.17}$$

We can then perform a substitution of variables, $T = t_1 + t_2$, giving:

$$\mathbf{u}_1(t_1, T) = \frac{2\Delta\mathbf{x} - (T + t_1)\mathbf{v}_0 - (T - t_1)\mathbf{v}_f}{t_1 T} + \mathbf{g} \tag{4.18}$$

$$\mathbf{u}_2(t_1, T) = \frac{-2\Delta\mathbf{x} + t_1\mathbf{v}_0 + (2T - t_1)\mathbf{v}_f}{T(T - t_1)} + \mathbf{g} \tag{4.19}$$

simplifying Eqn. (4.18) further we get:

$$\mathbf{u}_1(t_1, T) = \left(\frac{\mathbf{v}_f - \mathbf{v}_0}{T} + \mathbf{g}\right) + \frac{1}{t_1}\left(\frac{2\Delta\mathbf{x}}{T} - (\mathbf{v}_f + \mathbf{v}_0)\right) \tag{4.20}$$

Making the bang-bang assumption that $\mathbf{u}_1$ will saturate the inputs, *i.e.* $\|\mathbf{u}_1\| = u_{max}$ gives a quadratic equation in t_1:

$$u_{max}^2 = \left\|\left(\frac{\mathbf{v}_f - \mathbf{v}_0}{T} + \mathbf{g}\right) + \frac{1}{t_1}\left(\frac{2\Delta\mathbf{x}}{T} - (\mathbf{v}_f + \mathbf{v}_0)\right)\right\|^2 \tag{4.21}$$

For a given value of T, there are two possible solutions for $t_1(T)$, for which valid solutions must satisfy the conditions $0 \leq t_1(T) \leq T$. A given value of T is a dynamically feasible solution to the bang-bang problem if the condition $\|\mathbf{u}_2(t_1(T), T)\| \leq u_{max}$ is satisfied. In the following sections we will set analytical limits on the possible values of T to allow for a quick line search to find the minimum feasible bang-bang solution.

Lower Bound

We can set a lower bound on the possible time from start to goal, T, by decoupling the dimensions of the double integrator. Each one dimensional problem is then solved analytically as if it had the full control effort of the system available to it. The axis that takes the maximum time is the minimum possible time for the full coupled system. We can therefore

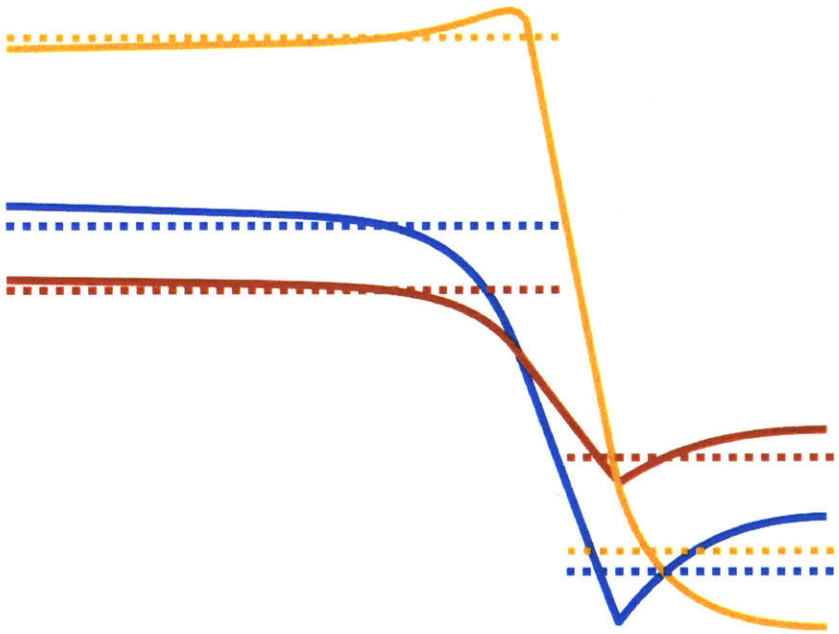


Figure 4-4: Example of acceleration (y-axis) vs. time (x-axis) for a coupled double integrator optimal solution (solid line) and bang-bang approximation (dashed line) for x (blue), y (red), and z (yellow) axes. While the bang-bang solution does not exactly match the optimal solution, it shares the same general structure with negligible additional trajectory time required.

define the lower bound on time by:

$$\begin{aligned} t_{lb} = & \max_{i \in \{x,y,z\}} t_i \\ \text{subject to} \quad & \dot{\mathbf{x}} = f_{di}(\mathbf{x}, \mathbf{u}), \\ & \mathbf{x}_i(0) = \mathbf{x}_{i,0}, \ i \in \{x,y,z\} \\ & \mathbf{x}_i(t_i) = \mathbf{x}_{i,f}, \ i \in \{x,y,z\} \\ & \|u_i\| \leq u_{max}, \ i \in \{x,y,z\} \end{aligned} \tag{4.22}$$

A 1D double integrator is well known to have a single optimal bang-bang solution with one or zero switching points [57]. We can write these equations for a single integrator as:

$$a = \pm u_{max} \tag{4.23}$$

$$\Delta x = v_0(\tau_1 + \tau_2) + \frac{1}{2}\left(a\left(\tau_1^2 - \tau_2^2 + 2\tau_1\tau_2\right) - g\left(\tau_1^2 + \tau_2^2 + 2\tau_1\tau_2\right)\right) \tag{4.24}$$

$$v_f - v_0 = a(\tau_1 - \tau_2) - g(\tau_1 + \tau_2) \tag{4.25}$$

We can solve directly for τ_1 in Eqn. (4.25), $\tau_1 = \frac{(v_f - v_0) + (a+g)\tau_2}{a-g}$ and substitute it into Eqn. (4.24) to get a single quadratic equation:

$$\Delta x = \frac{a^2 + ga}{(a-g)}\tau_2^2 + \frac{2av_f}{(a-g)}\tau_2 + \frac{v_f^2 - v_0^2}{2(a-g)} \tag{4.26}$$

which can be solved with the classic quadratic formula to find:

$$\tau_2 = \frac{-2av_f \pm \sqrt{2a(2\Delta x(a^2 - g^2) + a(v_0^2 + v_f^2) - g(v_f^2 - v_0^2)}}{2a(a+g)} \tag{4.27}$$

Combined with Eqn. (4.23) there are four possible solutions to the problem, of which we can use the smallest real positive solution. The maximum value of $\tau_1 + \tau_2$ for each of the axes of the problem sets a lower bound on the possible values for T in the bang-bang solution.

Upper Bound

We can set an upper bound for the coupled system by finding the analytical solution to a three phase trajectory to move from $\mathbf{x}_0$ to $\mathbf{x}_f$, based off of the method described by Johnson [43]. The first and third segments of the solution are (1) the minimum time solution to

move from $\mathbf{x}_0$ to a state with zero velocity ($\mathbf{x}_{m_0}$) by applying control input $\mathbf{a}_1$, and (3) the minimum time solution to move from a state $\mathbf{x}_{m_1}$ which has zero velocity to the final state $\mathbf{x}_f$ by applying control input $\mathbf{a}_3$. The middle section is then moving from the two stationary states $\mathbf{x}_{m_0}$ to $\mathbf{x}_{m_1}$ in minimum time, which has the bang-bang solution of applying $\mathbf{a}_2$ for half the time and $-\mathbf{a}_2$ for the second half. All three of these phases have simple analytical and ever present solutions, meaning that such a trajectory can *always* be used to move between any two states $\mathbf{x}_0$ and $\mathbf{x}_f$.

The combination of these three easy to solve analytic problems, which fully satisfy the constraints of Eqn. (4.13) provides an upper bound on the possible time between $\mathbf{x}_0$ and $\mathbf{x}_f$. This simple problem can be written as below:

$$\mathbf{v}_0 + (\mathbf{a}_1 - \mathbf{g})\,\tau_1 = \mathbf{0} \tag{4.28}$$

$$(\mathbf{a}_3 - \mathbf{g})\,\tau_3 = \mathbf{v}_f \tag{4.29}$$

$$\mathbf{x}_{m_0} = \mathbf{v}_0\tau_1 + \frac{1}{2}(\mathbf{a}_1 - \mathbf{g})\,\tau_1^2 \tag{4.30}$$

$$\mathbf{x}_f - \mathbf{x}_{m_0} = \frac{1}{2}(\mathbf{a}_3 - \mathbf{g})\,\tau_3^2 \tag{4.31}$$

$$\mathbf{x}_{m_1} - \mathbf{x}_{m_0} = (\mathbf{a}_2 - \mathbf{g})\,\tau_2^2 \tag{4.32}$$

$$\|\mathbf{a}_1\| \leq u_{max},\ \|\mathbf{a}_2\| \leq u_{max},\ \|\mathbf{a}_3\| \leq u_{max} \tag{4.33}$$

Without gravity these problems can easily be solved, however, gravity adds a certain non-linearity to the problem. Because we are looking for an upper bound solution we can further constrain the problem by requiring that gravity be canceled at all times. To do this we can change Eqn. (4.33) to a new set of constraints,

$$\|\mathbf{a}_1'\| \leq u_{max} - \|\mathbf{g}\|,\ \|\mathbf{a}_2'\| \leq u_{max} - \|\mathbf{g}\|,\ \|\mathbf{a}_3'\| \leq u_{max} - \|\mathbf{g}\| \tag{4.34}$$

and let $\mathbf{a}_i = \mathbf{a}_i' + \mathbf{g}$. We can easily see that this representation satisfies Eqn. (4.33), $\|\mathbf{a}_i\| = \|\mathbf{a}_i' + \mathbf{g}\| \leq \|\mathbf{a}_i'\| + \|\mathbf{g}\| \leq u_{max}$, while simplify the equations of motion. These simplified

equations of motion can then be solved analytically to provide the total times and states:

$$\tau_1 = \frac{\|v_0\|}{(u_{max} - \|\mathbf{g}\|)} \tag{4.35}$$

$$\tau_3 = \frac{\|v_f\|}{(u_{max} - \|\mathbf{g}\|)} \tag{4.36}$$

$$a_1 = \frac{-v_0}{\tau_1} \tag{4.37}$$

$$a_3 = \frac{v_f}{\tau_3} \tag{4.38}$$

$$\mathbf{x}_{m_0} = \frac{1}{2} a_1 \tau_1^2 \tag{4.39}$$

$$\mathbf{x}_{m_1} = \Delta x - \frac{1}{2} a_3 \tau_3^2 \tag{4.40}$$

$$\tau_2 = 2\sqrt{\frac{\|\mathbf{x}_{m_1} - \mathbf{x}_{m_0}\|}{(u_{max} - \|\mathbf{g}\|)}} \tag{4.41}$$

$$t_{ub} = \tau_1 + \tau_2 + \tau_3 \tag{4.42}$$

The final solution t_{ub} provides an upper limit to the bang-bang solution time T, as any "bang-bang" solution with a higher total time should instead be replaced by the three phase solution described above.

Solution

Putting together the three elements above, we have analytical techniques to (i) determine a minimum value for total trajectory time T, (ii) determine a maximum value for T, and (iii) validate if the bang-bang solution for a value of T satisfies the dynamic constraints of Eqn. (4.15). Combining these three elements we can perform a binary line search in between the upper and lower bounds, searching for the smallest value of T which satisfies the dynamic constraints.

4.2.2 Numerical Tests

The bang-bang approximation for a double integrator that is described above is guaranteed to return a valid solution (the solution computed by upper bound), however, it can provide no guarantee about the tightness of the solution to the true optimal value. For this purpose, Monte Carlo simulations were used to find the empirical tightness of the solution. Simulations were performed using a thrust-to-weight ratio of 4, meaning that the total available acceleration ($a_{max} = 40\ m/s^2$) is 4 times gravity ($10\ m/s^2$), matching a typical quadro-

tor capability. Tests were performed with $\mathbf{x}_0 = [0,0,0]$, $\mathbf{x}_f \in [0,30] \times [0,30] \times [-30,-30]$, $\|\mathbf{v}_0\| \in [0,30]$, $\|\mathbf{v}_f\| \in [0,30]$ and randomized initial and final velocity directions. Simulations were only performed along the positive axes of x and y because of the symmetry in the problem. Because no analytical solution is available for the coupled double integrator to compare against, a (computationally expensive) numerical shooting method was used as a comparison. The decision variables of the shooting method were the timestep Δt and the accelerations at each timestep $\mathbf{a}_i$, $\mathbf{d} = \{\Delta t, \mathbf{a}_0, \ldots, \mathbf{a}_{N-1}\}$. The optimization was then formulated as follows:

$$\begin{aligned} \underset{d}{\text{minimize}} \quad & \Delta t, \\ \text{subject to} \quad & \|\mathbf{a}_i\| \leq a_{max}, \text{ for all } i \in (0, N-1) \\ & \Delta t \left(N\mathbf{v}_0 + \sum_{i=0}^{N-1} (\mathbf{a}_i - \mathbf{g}) \left(N - i + \frac{1}{2} \right) \right) - \mathbf{x}_f = 0 \\ & \mathbf{v}_0 + \Delta t \sum_{i=0}^{N-1} (\mathbf{a}_i - \mathbf{g}) - \mathbf{v}_f = 0 \end{aligned} \tag{4.43}$$

Simulations were performed with $N = 100$ and initial Δt with three values between the lower bound and upper bound solutions.

The timing results for the bang-bang computation are shown in Figure 4-5, showing approximately $\sim 70\ \mu s$ per trajectory computed, including computation of the lower bound, upper bound, and the binary search. The binary search was stopped once changes in T reached 1% of the current value of T. Comparison in length of the computed trajectories for the bang-bang approximation and the shooting method are shown in Figure 4-6. In 99.8% of cases the bang-bang solution was within 5% or better of the solution found by the shooting method.

4.2.3 Double Integrator Motion Planning

With the boundary value problem solver from above, we now have (approximately) satisfied the assumptions required to use Algorithm 1 for global planning of a double integrator through obstacle fields. Because the boundary value problem is approximate, the global motion planning algorithm can no longer maintain its full optimality and completeness guarantees, however, as demonstrated below, the full system remains highly effective and near optimal.

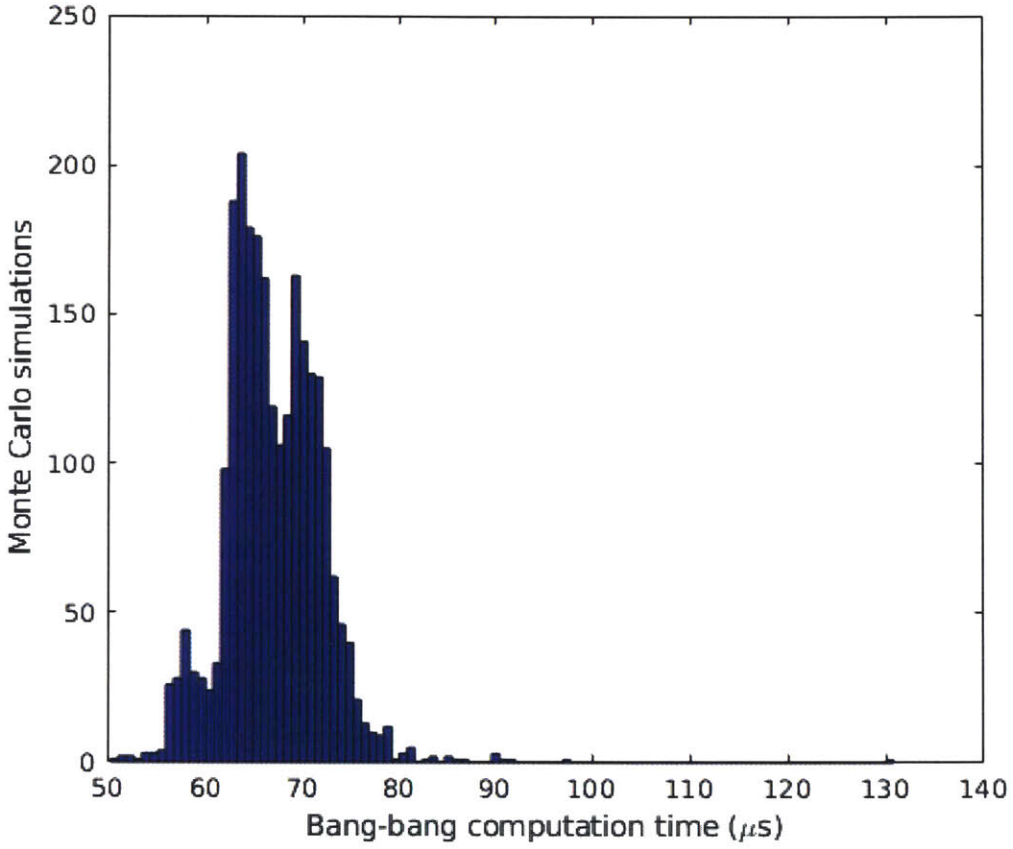


Figure 4-5: Histogram of computation times for double integrator approximation. Timing include computation of the lower bound and upper bound of the solution space, and the binary search for the lowest cost solution.

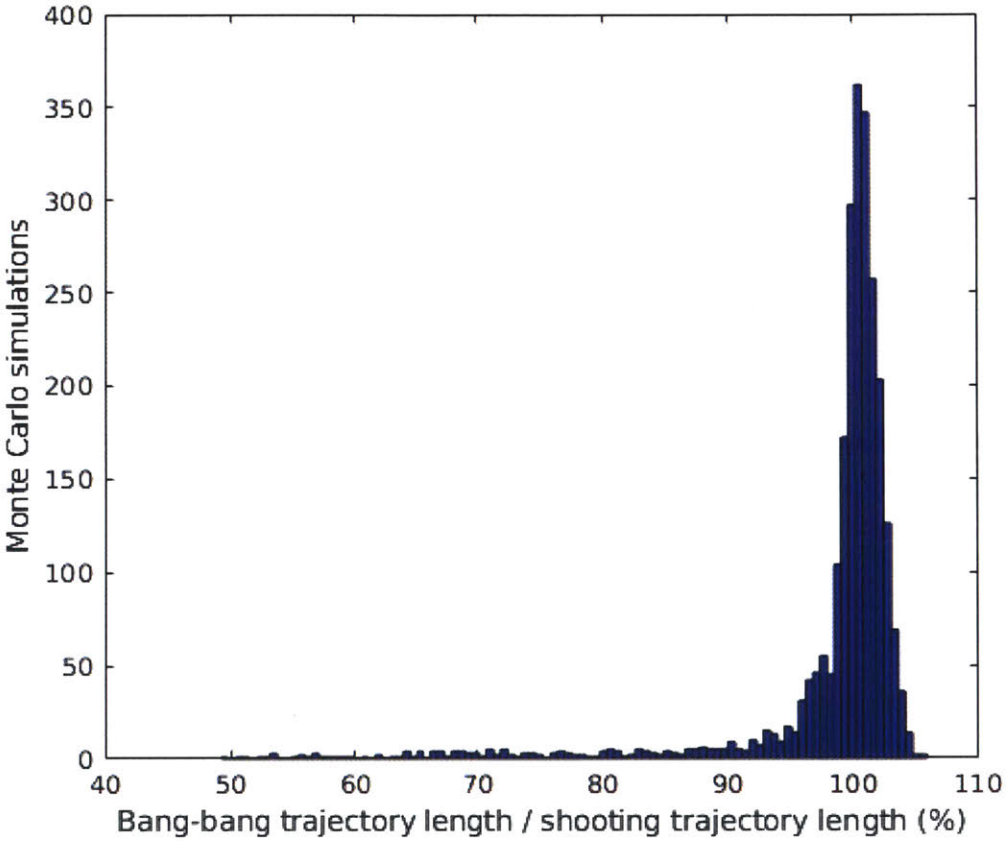


Figure 4-6: Histogram of the ratio of cost of the trajectory found through a bang-bang approximation and the cost found using a shooting method with 100 node points. Histogram does not include 4 out of 2500 samples (0.2%) which had a ratio over 1.2, maximum ratio seen was 1.96

Decoupled Double Integrator

As mentioned previously, there is no good method of true time minimal motion planning for a coupled double integrator existing in literature that is known to this author. An implementation of a coupled double integrator through gridded search techniques is infeasible in nearly all scenarios. Fortunately, the double integrator presents itself well for graph search in a decoupled form, with the total available thrust distributed along each of the axes. It is this decoupled formulation that is used in nearly all works that use a double integrator model, of particular note [66]. While this system technically remains a 6D system, by decoupling the axes the search space becomes closer to 3 2D problems solved simultaneously. This decoupled search technique gives an effective graph structure to search via D* Lite for comparison to Algorithm 1.

Double Integrator Comparison

To test the performance of the complete coupled double integrator motion planner, we perform the same Monte Carlo simulation experiments as carried out in Section 4.1. Comparisons are again made in a randomized obstacle field in $\mathbb{R}^3$, however, because of the velocity dimensions of the system the full configuration space is in $\mathbb{R}^6$. In total, 200 simulations were performed comparing the double integrator solution in Algorithm 1 to the decoupled search method described above, see Figure 4-7 for a snap shot of one of the experiments, and Figure 4-8 for a summary of results.

Because Algorithm 1 only expands the *necessary* components of the 6D configuration space, it is able to solve the more complex fully coupled dynamics with less computation than required for D* Lite to solve the reduced decoupled solution. The benefits of solving with full coupled dynamics can be seen in the trajectory cost, which is 30-40% worse when using the decoupled dynamics.

In addition to the Monte Carlo simulations in the randomly distributed obstacle world, we also performed simulations in real world environments off of recorded data. These real world environments use a base level Octomap representation for "sensing", with queries to parts of the map marking new sensing information. The final result of two such queries are shown in Figure 4-9 and Figure 4-10. In Figure 4-9 a quadrotor is navigating through a window and around a tree to reach the goal point in a more local map, while in Figure 4-10

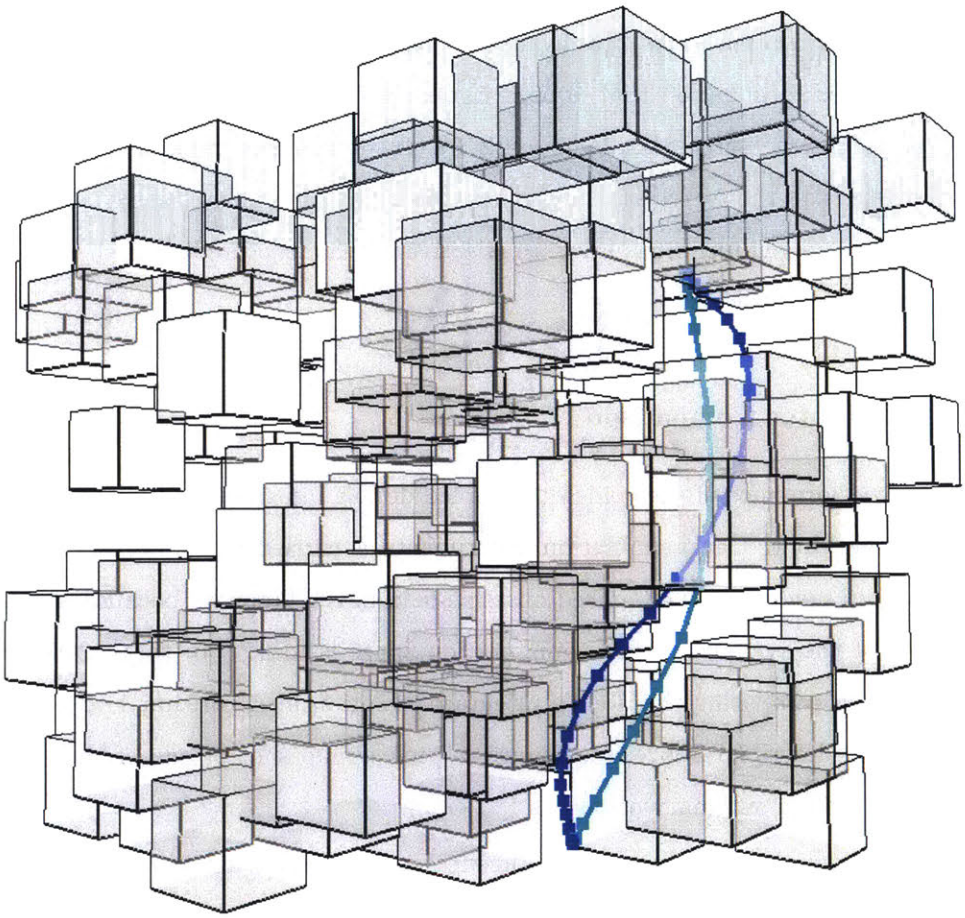


Figure 4-7: Visualization of the Monte Carlo simulations performed for a coupled double integrator model. The solution found using our tree based method is shown in Cyan with an execution time of 1.65 s and a computation time of 141.46 ms, while a grid based search is shown in Blue with an execution time of 2.23 s and a computation time of 973.65 ms. Each planner was run with a discretization of 0.25 m, though note that discretization only occurs for the tree based method along the boundaries of configuration space.

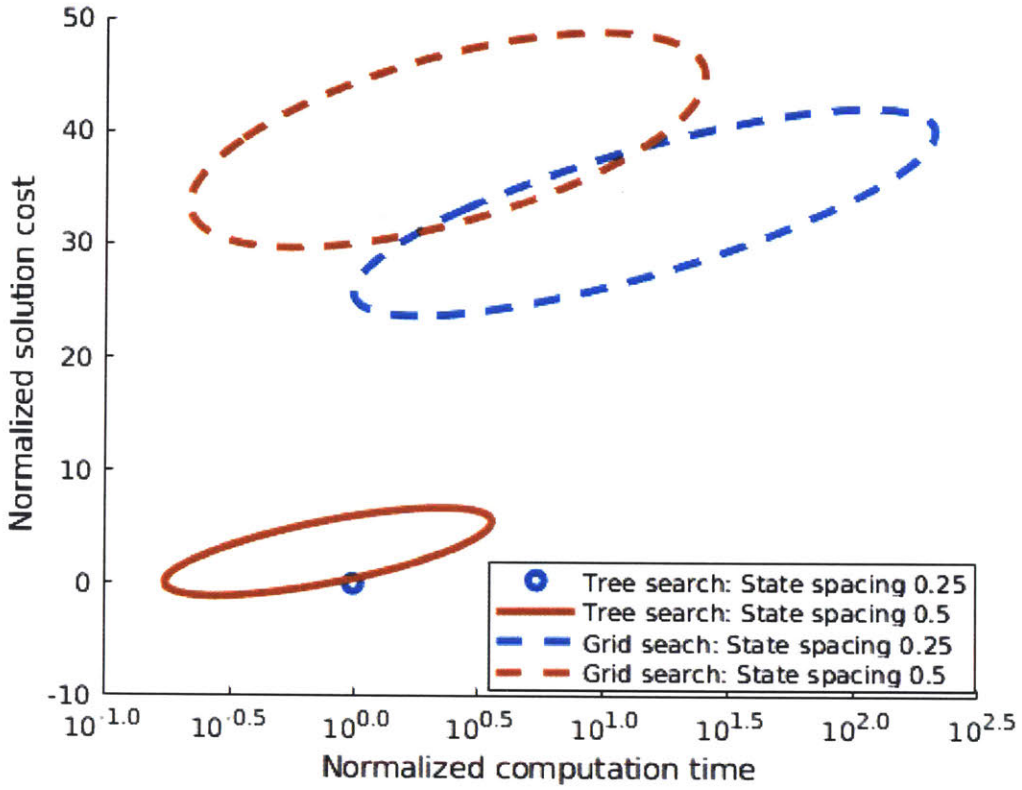


Figure 4-8: Experimental results across 200 Monte Carlo simulations in a randomized world like the one shown in Figure 4-7, with 200 obstacles of length 3.0. Due to the decoupling of the axes as well as the discretized nature of the solution, the grid based method performs 30-40% worse than our tree search, while also finding a slower solution.

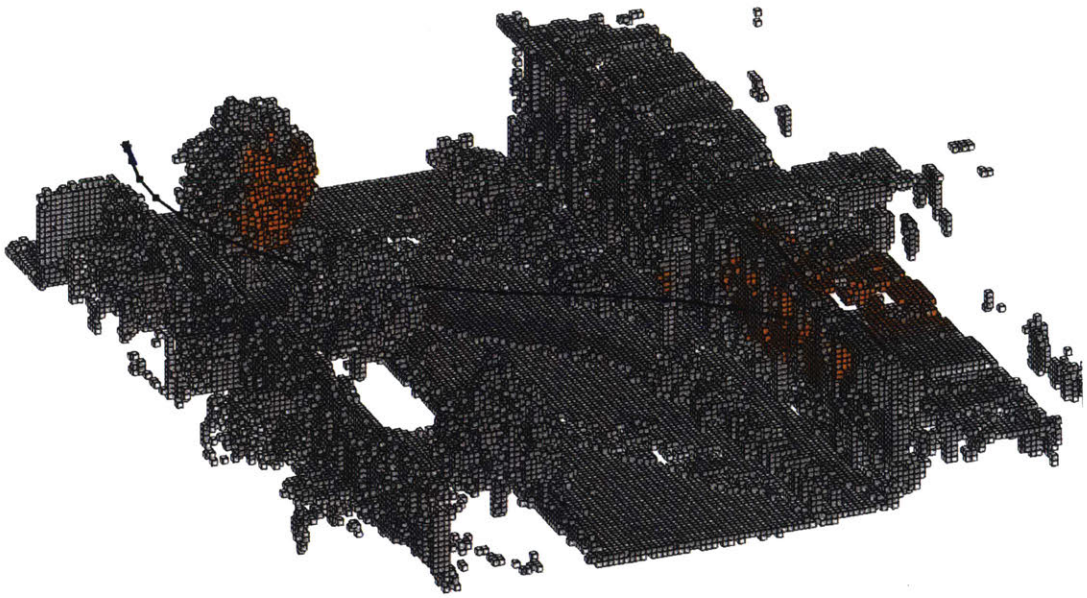


Figure 4-9: Double integrator trajectory computed flying through a window and around a tree in data collected on the University of Freidburg campus†. Gray voxels denote occupied space in the underlying world that was not directly mapped, while orange voxels are mapped areas. Total computation time for this trajectory was 331 ms.

†University of Freidburg campus courtesy of Armin Hornung and Bastian Steder http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/

the quadrotor navigates across the full campus model. The full underlying map is shown in gray (a subset of the model of the University of Friedberg campus), while the directly sensed voxels are shown in orange. In the first example, despite the tight operating windows and the need to explore against a full wall (a challenging scenario for this algorithm), the full computation took 331 ms. In the second for the larger scale problem, Algorithm 1 took 223 ms to calculate and a trajectory length of 3.4 s. For comparison, the same problem took 7521 ms with a trajectory cost of 5.4 s using Lazy D* Lite. Because the algorithm treats unoccupied elements of configuration space as continuous, it seamlessly switches between the larger scale search and the smaller scale search without an increase in complexity or the need for hand tuning adaptive resolution graphs.

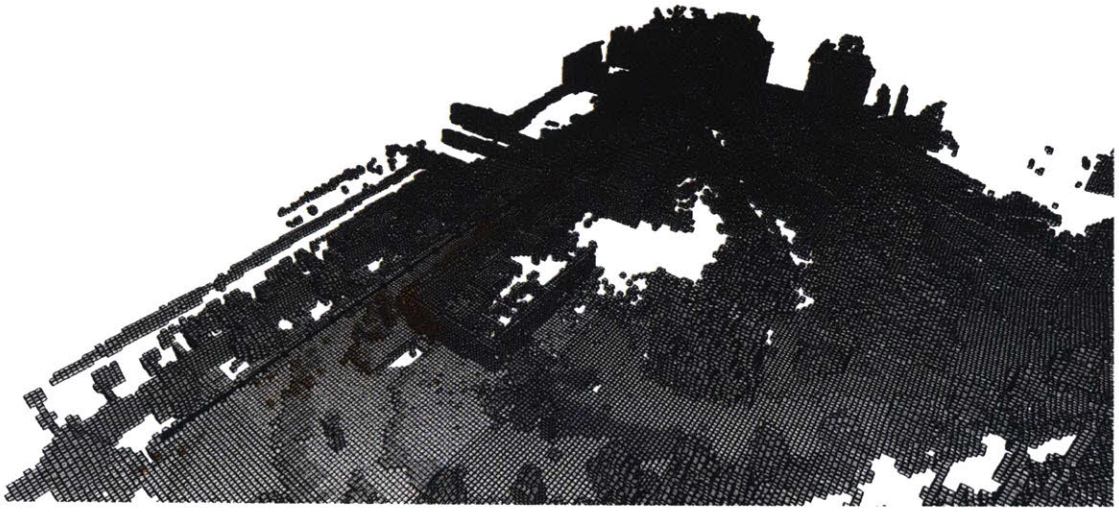


Figure 4-10: Double integrator trajectory computed flying around obstacles through spatial data collected in the University of Freidburg campus[†]. Gray voxels denote occupied space in the underlying world that was not directly mapped, while orange voxels are mapped areas. Total computation time for this trajectory using Algorithm 1 was 223 ms for a trajectory length of 3.4 s. For comparison, the same problem took 7521 ms with a trajectory cost of 5.4 s using Lazy D* Lite.

[†]University of Freidburg campus courtesy of Armin Hornung and Bastian Steder http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/

4.3 Conclusion

In this section we demonstrated the effectiveness of Algorithm 1 on several test systems, comparing its performance in both computation time and trajectory cost against D* Lite. Both Algorithm 1 provide the desired "sensing optimality" through Lazy graph search, but Algorithm 1 sparse structure and continuous representation of configuration space allow it to generate shorter trajectories with less computation time. Of particular note for this work, and for the application of Algorithm 1, we demonstrated the algorithm on a coupled double integrator model of a quadrotor UAV. This model provides a six-dimensional system with non-linear constraints, a challenging and relatively unsolved scenario for standard planning algorithms. Algorithm 1 provides real time performance with near-global optimality in large scale obstacle rich environments. Along the way to solving this motion planning problem, we derived a fast near optimal approximation for solving the obstacle free boundary value problem for a coupled double integrator.

Chapter 5

Future Work and Conclusions

5.1 Future Work

5.1.1 Implementation

While the work presented here is primarily a theoretical and simulation based work, it is designed with the intent of running on-board a real system using the Dynamic (Algorithm 5) and Anytime (Algorithm 7) properties of the algorithm. A key part of implementing the algorithm on a real system is the development of algorithms that are well suited to partial queries of sensor data. For a vision based system we are particularly interested in methods of computing occupancy in small sub-volumes of space in an iterative manner. Basic methods of stereo processing such as local block matching are particularly well suited to this, as computing occupancy of a volume requires a single disparity check. While this would be a clean (and likely effective) initial solution, significant improvements in stereo have been shown with techniques such as semi-global matching [38]. A key component of these techniques, however, is the regularization of all computed disparities across an image. A potentially interesting future work would be to adapt semi-global matching to work in an incremental fashion, regularizing only across currently processed data. In addition to the algorithm presented here, such a system could be used with simpler high speed motion planning methods such as trajectory libraries [27].

5.1.2 Non-physical barriers

One of the notable advantages of working with the incremental mapping technique described here is that it incrementally builds an obstacle map in configuration space during runtime. Though we have primarily looked at obstacles in configuration spaces as physical obstacles, nothing in the methods described requires impassible areas of configuration space to be *physical*. We can use the same techniques to perform incremental discovery of obstacles in configuration space that are non-physical.

Based off of the work in Chapter 2, we are particularly interested in the problem of integrating state estimation and motion planning systems. Some of the most challenging problems in visual state estimation such as maintaining a state estimate through low textured areas and through degenerate motions such as pure rotation are relatively simple problems when coupled with motion planning. Rather than make a complete state estimation system which can handle all scenarios, we can instead create a state estimation system with a clearly defined working area, and perform motion planning that respects those bounds. Several recent works have approached this problem from a trajectory optimization standpoint [105, 20, 71], looking to adapt an existing motion plan to maximize state estimation performance.

A different possible approach would be to look at conditions for state estimation as inoperable areas of configuration space. The state estimation system performs well in a set of scenarios, and does not perform in others (and therefore those states can not be entered safely). This creates a scenario of obstacles which may not be entered in configuration space, similar to those found with typical motion planning. Using the incremental approach described here, these "state estimation obstacles" may be avoided by incrementally expanding the areas around these obstacles until a trajectory is found that satisfies both state estimation requirements and motion planning optimality. What was previously a prohibitively expensive operation, *i.e.* determining a complete map of state-estimation-safe regions, may now be done on the fly.

Based off of the tree structure, it may also be possible to develop algorithms which are history aware. A given state in configuration space may be available given one approach (which for example it triangulates features better), but not available from another.

5.1.3 Complex dynamical systems

This thesis primarily focuses on systems for which the boundary value problem can be solved, or at least approximated as in the case of the coupled double integrator, however, the concepts are generally based upon the ability to solve some dynamics in an obstacle free world. In a similar manner as described in the previous section, some of the constraints on these optimization problems can be relaxed at the expense of adding more "obstacles" to the configuration space. For example, velocity constraints on a double integrator can be incorporated into the problem either in the boundary value problem or in the search by adding virtual obstacles whenever a trajectory surpasses the allowable velocity of the system. An interesting area of future work would be looking at solving highly complex dynamics through a mix of simpler boundary value problems and configuration space constraints.

5.2 Conclusions

In this thesis we show two sides of the high performance navigation problem: an experimental demonstration of a state of the art autonomous UAV platform with fully on-board state estimation and control, and a theoretical description of a combined mapping-and-motion planning system for navigation in unknown environments. The research platform provides a fully integrated and customizable platform for further development, giving access to all components of the system from the high level state estimation to low level motor control and feedback. The combined mapping-and-motion planning system gives a high performance system capable of working in unknown environments where mapping is expensive, and also is demonstrated to work as a practical motion planner on complex high dimensional systems. We demonstrate that taking a holistic view of the robot navigation problem, *e.g.* looking at mapping and motion planning as a joint process opens up new avenues for both. Rather than simply augmenting existing methods we are able to take advantage of fundamental principles of optimal navigation through obstacles to quickly and optimally navigate in complex high dimensional worlds.

Bibliography

[1] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. Stereo vision and laser odometry for autonomous helicopters in GPS-denied indoor environments. In *Intl. Soc. Opt. Eng. (SPIE)*, 2009.

[2] H. Alt and E. Welzl. Visibility graphs and obstacle-avoiding shortest paths. *Zeitschrift für Operations Research*, 32(3-4):145–164, May 1988.

[3] Nancy M Amato, O Burchan Bayazit, and Lucia K Dale. OBPRM: An obstacle-based PRM for 3D workspaces. 1998.

[4] Amado Antonini, Winter Guerra, Varun Murali, Thomas Sayre-McCord, and Sertac Karaman. The Blackbird Dataset: A large-scale dataset for UAV perception in aggressive flight. In *Intl. Sym. on Experimental Robotics (ISER)*, 2018.

[5] Andrew J Barry, Anirudha Majumdar, and Russ Tedrake. Safety verification of reactive controllers for UAV flight in cluttered environments using barrier certificates. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 484–490, 2012.

[6] M. Bloesch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based MAV navigation in unknown and unstructured environments. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 21–28, 2010.

[7] Robert Bohlin and Lydia E Kavraki. Path planning using lazy PRM. In *IEEE Robotics and Automation Letters*, volume 1, pages 521–528. IEEE, 2000.

[8] Samir Bouabdallah and Roland Siegwart. Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2247–2252. IEEE, 2005.

[9] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 723–730. IEEE, 2011.

[10] M. Bryson, M. Johnson-Roberson, and S. Sukkarieh. Airborne smoothing and mapping using vision and inertial sensors. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3143–3148, 2009.

[11] Michael Burri, Helen Oleynikova, Markus W. Achtelik, and Roland Siegwart. Real-time visual-inertial mapping, re-localization and planning onboard MAVs in unknown environments. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 2015-Decem, 2015.

[12] Luca Carlone and Sertac Karaman. Attention and anticipation in fast visual-inertial navigation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3886–3893. IEEE, may 2017.

[13] Benjamin Cohen, Mike Phillips, and Maxim Likhachev. Planning Single-arm Manipulations with N-Arm Robots. In *Robotics: Science and Systems (RSS)*, 2014.

[14] Robin Deits and Russ Tedrake. Efficient Mixed-Integer Planning for UAVs in Cluttered Environments. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2015.

[15] Frank Dellaert. Factor graphs and GTSAM: A hands-on introduction. Technical Report GT-RIM-CP&R-2012-002, Georgia Institute of Technology, September 2012.

[16] Christopher M Dellin and Siddhartha S. Srinivasa. A unifying formalism for shortest path problems with expensive edge evaluations via lazy best-first search over paths with edge selectors. *icaps*, (2295):459–467, 2016.

[17] J. Dong, L. Carlone, G. C. Rains, T. Coolong, and F. Dellaert. 4D mapping of fields using autonomous ground and aerial vehicles. In *2014 ASABE and CSBE/SCGAB Annual International Meeting*, 2014.

[18] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Camera-based navigation of a low-cost quadrocopter. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 320:240, 2012.

[19] Matthias Faessler, Flavio Fontana, Christian Forster, Elias Mueggler, Matia Pizzoli, and Davide Scaramuzza. Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle. *J. of Field Robotics*, 33(4):431–450, jun 2016.

[20] Davide Falanga, Philipp Foehn, Peng Lu, and Davide Scaramuzza. Pampc: Perception-aware model predictive control for quadrotors. *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018.

[21] Davide Falanga, Elias Mueggler, Matthias Faessler, and Davide Scaramuzza. Aggressive Quadrotor Flight through Narrow Gaps with Onboard Sensing and Computing using Active Vision. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, dec 2017.

[22] Dai Feng and B Krogh. Acceleration-constrained time-optimal control in n dimensions. *IEEE transactions on automatic control*, 31(10):955–958, 1986. Discussion of time optimal control for acceleration constrained systems.

[23] Peter R Florence, John Carter, Jake Ware, and Russ Tedrake. NanoMap: Fast, Uncertainty-Aware Proximity Queries with Lazy Search over Local 3D Data. *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2018.

[24] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Robotics: Science and Systems (RSS)*, 2015.

[25] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast Semi-Direct Monocular Visual Odometry. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2014.

[26] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-Manifold Preintegration for Real-Time Visual-Inertial Odometry. *IEEE Trans. Robotics*, pages 1–20, 2016.

[27] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Maneuver-Based Motion Planning for Nonlinear Systems With Symmetries. *IEEE Trans. Robotics*, 21(6), 2005.

[28] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. RotorSâĂŤa modular Gazebo MAV simulator framework. In *Robot Operating System (ROS)*, pages 595–625. Springer Verlag, 2016.

[29] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 4340–4349, 2016.

[30] Sourish Ghosh and Joydeep Biswas. Joint perception and planning for efficient obstacle avoidance using stereo vision. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1026–1031. IEEE, sep 2017.

[31] Kalin Gochev, Alla Safonova, and Maxim Likhachev. Planning with adaptive dimensionality for mobile manipulation. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2944–2951. IEEE, 2012.

[32] Alex A Gorodetsky, Sertac Karaman, and Youssef M Marzouk. Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition. In *Robotics: Science and Systems (RSS)*, 2015.

[33] A. Handa, T. Whelan, J.B. McDonald, and A.J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Hong Kong, 2014.

[34] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[35] Markus Hehn and Raffaello DâĂŹAndrea. Quadrocopter trajectory generation and control. In *IFAC World Congress*, volume 18, pages 1485–1491, 2011.

[36] D Hernandez-Juarez, A Chacón, A Espinosa, D Vázquez, J C Moure, and A M López. Embedded real-time stereo estimation via Semi-Global Matching on the GPU. In *iccs*, 2016.

[37] J.A. Hesch, D.G. Kottas, S.L. Bowman, and S.I. Roumeliotis. Camera-IMU-based localization: Observability analysis and consistency improvement. *Intl. J. of Robotics Research*, 33(1):182–201, 2014.

[38] Heiko Hirschmuller. Stereo Processing by Semi-Global Matching and Mutual Information. *IEEE Trans. Pattern Anal. Machine Intell.*, 30(2):328–341, 2008.

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

[40] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 3:189–206, Apr 2013.

[41] J. How, C. Fraser, K.C. Kulling, L.F. Bertuccelli, O. Toupet, L. Brunet, A. Bachrach, and N. Roy. Increasing autonomy of UAVs. *IEEE Robotics and Automation Magazine*, 16(2), 2009.

[42] Albert S Huang, Edwin Olson, and David C Moore. LCM: Lightweight communications and marshalling. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4057–4062, 2010.

[43] Stewart D Johnson. Computing minimum time paths with bounded acceleration. *arXiv preprint arXiv:1310.5905*, 2013.

[44] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research*, 31:217–236, Feb 2012.

[45] Biliana Kaneva, Antonio Torralba, and William T Freeman. Evaluation of image features using a photorealistic virtual world. In *Intl. Conf. on Computer Vision (ICCV)*, pages 2282–2289. IEEE, 2011.

[46] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Intl. J. of Robotics Research*, 30(7):846–894, May 2011.

[47] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, 1996.

[48] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*, pages 225–234, Nara, Japan, Nov 2007.

[49] L. Kneip, S. Weiss, and R. Siegwart. Deterministic initialization of metric state estimation filters for loosely-coupled monocular vision-inertial systems. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2235–2241, 2011.

[50] Laurent Kneip, Margarita Chli, and Roland A Siegwart. Robust Real-Time Visual Odometry with a Single Camera and an IMU. In *British Machine Vision Conf. (BMVC)*, volume 23, page 1157, 2004.

[51] Nathan Koenig and Andrew Howard. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154. IEEE, 2004.

[52] Sven Koenig and Maxim Likhachev. D* Lite. *AAAI/IAAI*, 15, 2002.

[53] Dimitrios G. Kottas, Joel A. Hesch, Sean L. Bowman, and Stergios I. Roumeliotis. On the consistency of vision-aided inertial navigation. In *Intl. Sym. on Experimental Robotics (ISER)*, 2012.

[54] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.

[55] Benoit Landry, Robin Deits, Peter R Florence, and Russ Tedrake. Aggressive Quadrotor Flight through Cluttered Environments Using Mixed Integer Programming. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1469–1475, 2016.

[56] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[57] Steven M LaValle. *Planning algorithms*. Cambridge University Press, 2006.

[58] John J. Leonard and H.F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1442–1447. IEEE, 1991.

[59] John J. Leonard and Hugh F. Durrant-Whyte. Mobile Robot Localization by Tracking Geometric Beacons. *IEEE Trans. Robot. Automat.*, 7(3):376–382, 1991.

[60] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visual-inertial SLAM using nonlinear optimization. *Intl. J. of Robotics Research*, 2015.

[61] Maxim Likhachev, Geoff Gordon, and Sebastian Thrun. ARAâĂć: Anytime AâĂć with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems*, 16:12, 2004.

[62] Maxim Likhachev, Alex Nash, and Sven Koenig. Incremental Phi*: Incremental Any-Angle Path Planning on Grids. *Intl. Joint Conf. on AI (IJCAI)*, 2009.

[63] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen. Autonomous aerial navigation using monocular visual-inertial fusion. *J. of Field Robotics*, 00:1–29, 2017.

[64] Yi Lin, Fei Gao, Tong Qin, Wenliang Gao, Tianbo Liu, William Wu, Zhenfei Yang, and Shaojie Shen. Autonomous aerial navigation using monocular visual-inertial fusion. *J. of Field Robotics*, jul 2017.

[65] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2872–2879. IEEE, 2017.

[66] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in se (3). *ral*, 3(3), 2018.

[67] G. Loianno, G. Cross, C. Qu, Y. Mulgaonkar, J.A. Hesch, and V. Kumar. Flying smartphones: Automated flight enabled by consumer electronics. *IEEE Robotics and Automation Magazine*, 22(2):24–32, 2015.

[68] G. Loianno and V. Kumar. Vision-based fast navigation of micro aerial vehicles. In *Proc. SPIE, Micro- and Nanotechnology Sensors, Systems, and Applications*, 2016.

[69] Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar. Estimation, Control, and Planning for Aggressive Flight With a Small Quadrotor With a Single Camera and IMU. *IEEE Robotics and Automation Letters*, 2(2):404–411, apr 2017.

[70] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *Intl. J. of Robotics Research*, 36(8):947–982, 2017.

[71] Travis Manderson, Andrew Holliday, and Gregory Dudek. Gaze selection for enhanced visual odometry during navigation. In *Proceedings of the Conference on Computer and Robot Vision*, 2018.

[72] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2520–2525. IEEE, 2011.

[73] Najib Metni and Tarek Hamel. A uav for bridge inspection: Visual servoing control law with orientation limits. *Automation in construction*, 17(1):3–10, 2007.

[74] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar Von Stryk. Comprehensive simulation of quadrotor UAVs using ROS and Gazebo. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 400–411. Springer Verlag, 2012.

[75] Kartik Mohta, Michael Watterson, Yash Mulgaonkar, Sikang Liu, Chao Qu, Anurag Makineni, Kelsey Saulnier, Ke Sun, Alex Zhu, Jeffrey Delmerico, et al. Fast, autonomous flight in gps-denied and cluttered environments. *J. of Field Robotics*, 35(1):101–120, 2018.

[76] A.I. Mourikis and S.I. Roumeliotis. A dual-layer estimator architecture for long-term localization. In *Proc. of the Workshop on Visual Localization for Mobile Platforms at CVPR*, June 2008.

[77] Mark W Mueller, Markus Hehn, and Raffaello D'Andrea. A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Trans. Robotics*, 31(6):1294–1310, 2015.

[78] Mustafa Mukadam, Jing Dong, Frank Dellaert, and Byron Boots. Simultaneous Trajectory Estimation and Planning via Probabilistic Inference. In *Robotics: Science and Systems (RSS)*, 2017.

[79] R. Mur-Artal, J.M.M. Montiel, and J.D. Tardós. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Trans. Robotics*, 31(5):1147–1163, 2015.

[80] Alex Nash, Kenny Daniel, Sven Koenig, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-Angle Path Planning on Grids. In *Nat. Conf. on Artificial Intelligence (AAAI)*, volume 1, pages 1–47, 2007.

[81] Alex Nash, Sven Koenig, and Craig Tovey. Lazy Theta*: Any-Angle Path Planning and Path Length Analysis in 3D. In *Nat. Conf. on Artificial Intelligence (AAAI)*, 2010.

[82] Helen Oleynikova, Dominik Honegger, and Marc Pollefeys. Reactive avoidance using embedded stereo vision for mav flight. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 50–56. IEEE, 2015.

[83] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Trans. Intelligent Vehicles*, 1(1):33–55, 2016.

[84] Sudeep Pillai, Srikumar Ramalingam, and John J. Leonard. High-Performance and Tunable Stereo Reconstruction. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2016.

[85] L. S. Pontryagin. *The mathematical theory of optimal processes.* International series of monographs in pure and applied mathematics: v.55. Oxford, New York, Pergamon Press; [distributed in the Western Hemisphere by Macmillan, New York] 1964., 1964.

[86] Will Pryor, Yu-Chi Lin, and Dmitry Berenson. Integrated affordance detection and humanoid locomotion planning. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 125–132. IEEE, nov 2016.

[87] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Proc. of the Intl. Symp. of Robotics Research (ISRR)*. Springer Verlag, 2013.

[88] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial Trajectory Planning for Quadrotor Flight. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2013.

[89] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for Data: Ground Truth from Computer Games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conf. on Computer Vision (ECCV)*, pages 102–118. Springer International Publishing, Cham, 2016.

[90] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 3234–3243, 2016.

[91] Thomas Sayre-McCord, Winter Guerra, Amado Antonini, Jasper Arneberg, Austin Brown, Guilherme Cavalheiro, Yajun Fang, Alex Gorodetsky, Dave McCoy, Sebastian Quilter, Fabian Riether, Ezra Tal, Yunus Terzioglu, Luca Carlone, and Sertac Karaman. Visual-inertial navigation algorithm development using photorealistic camera simulation in the loop. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2018.

[92] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. A toolbox for easily calibrating omnidirectional cameras. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 5695–5701. IEEE, 2006.

[93] Wilko Schwarting, Javier Alonso-Mora, and Daniela Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):187–210, 2018.

[94] Brual C. Shah, Petr Švec, Ivan R. Bertaska, Armando J. Sinisterra, Wilhelm Klinger, Karl von Ellenrieder, Manhar Dhanak, and Satyandra K. Gupta. Resolution-adaptive risk-aware trajectory planning for surface vehicles operating in congested civilian traffic. *Autonomous Robots*, 40(7):1139–1163, Oct 2016.

[95] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[96] Shaojie Shen, Nathan Michael, and Vijay Kumar. Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft MAVs. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 5303–5310. IEEE, may 2015.

[97] Shaojie Shen, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. Vision-Based State Estimation and Trajectory Control Towards High-Speed Flight with a Quadrotor. In *Robotics: Science and Systems (RSS)*, 2013.

[98] J. Shi and C. Tomasi. Good features to track. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 1994.

[99] Ezra Tal and Sertac Karaman. Precision tracking of aggressive quadrotor trajectories using incremental nonlinear dynamic inversion and differential flatness. In *IEEE Conf. on Decision and Control*. IEEE, 2018.

[100] Sarah Tang and Vijay Kumar. Autonomous flight. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):29–52, 2018.

[101] Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, feb 2015.

[102] Sonia Waharte and Niki Trigoni. Supporting search and rescue operations with uavs. In *International Conference on Emerging Security Technologies (EST)*, pages 142–147. IEEE, 2010.

[103] Stephan Weiss, Markus W Achtelik, Simon Lynen, Michael C Achtelik, Laurent Kneip, Margarita Chli, and Roland Siegwart. Monocular vision for long-term micro aerial vehicle state estimation: A compendium. *J. of Field Robotics*, 30(5):803–831, 2013.

[104] Zhengdong Zhang, Amr Suleiman, Luca Carlone, Vivienne Sze, and Sertac Karaman. Visual-Inertial Odometry on Chip: An Algorithm-and-Hardware Co-design Approach. In *Robotics: Science and Systems (RSS)*, 2017.

[105] Zichao Zhang and Davide Scaramuzza. Perception-aware receding horizon navigation for mavs. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2018.