Coordinated Landing and Mapping with Aerial and Ground Vehicle Teams

by

Yan Ma

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical Engineering

Waterloo, Ontario, Canada, 2012

© Yan Ma 2012

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Micro Umanned Aerial Vehicle (UAV) and Umanned Ground Vehicle (UGV) teams present tremendous opportunities in expanding the range of operations for these vehicles. An effective coordination of these vehicles can take advantage of the strengths of both, while mediate each other's weaknesses. In particular, a micro UAV typically has limited flight time due to its weak payload capacity. To take advantage of the mobility and sensor coverage of a micro UAV in long range, long duration surveillance mission, a UGV can act as a mobile station for recharging or battery swap, and the ability to perform autonomous docking is a prerequisite for such operations. This work presents an approach to coordinate an autonomous docking between a quadrotor UAV and a skid-steered UGV. A joint controller is designed to eliminate the relative position error between the vehicles. The controller is validated in simulations and successful landing is achieved in indoor environment, as well as outdoor settings with standard sensors and real disturbances.

Another goal for this work is to improve the autonomy of UAV-UGV teams in positioning denied environments, a very common scenarios for many robotics applications. In such environments, Simultaneous Mapping and Localization (SLAM) capability is the foundation for all autonomous operations. A successful SLAM algorithm generates maps for path planning and object recognition, while providing localization information for position tracking. This work proposes an SLAM algorithm that is capable of generating high fidelity surface model of the surrounding, while accurately estimating the camera pose in real-time. This algorithm improves on a clear deficiency of its predecessor in its ability to perform dense reconstruction without strict volume limitation, enabling practical deployment of this algorithm on robotic systems.

Acknowledgements

I would like to dedicate this work to my family for supporting me every step of the way. I have to thank my beautiful wife for her love, companionship and patience. To my parents, thank you for your faith in me, and guiding me with everything you could to be the person I am today.

I am also deeply grateful for Prof. Steven Waslander, for giving me this amazing opportunity to work on the bleeding edge of the technology of my passion. This work would not have been where it is without his mentoring and guidance.

I would like to thank Aeryon Labs and Clearpath Robotics for their support and insights. Special thanks to Dr. Mike Peasgood for his knowledge and patience throughout the troubleshooting process for the Scout.

I would also like to thank my colleagues in WAVELab, namely Yassir, Mike, John, P.J., Peiyi, Adeel, Carlos, Arun, Sid and Ryan for the good times! I need to credit Dr. John Daly for being an amazing mentor throughout the landing project.

Finally, I have to thank University of Waterloo for the incredible experiences I have had throughout my university career.

Dedication

This is dedicated to my family, mentors and friends over the years.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Micro Unmanned Aerial Vehicles (UAV) have generated tremendous research interest in the field of mobile robotics with potential applications in building inspection, surveillance and military operations. The recent occurrence of Fukushima nuclear crisis strongly motivates the need for micro UAVs in autonomous exploration of complex, unknown and dangerous environments.

Quadrotors have been one of the most popular UAV research platforms, due to advantages in maneuverability and Vertical Take off and Landing (VTOL) capability. It has been demonstrated that they are capable of aggressive flight maneuvers [3, 4], coordinated formation flight [5] and cooperative construction of physical structures [6]. While impressive, these results are confined within environments with high precision positioning system that provides sub-centimetre accuracy and fast (50-100Hz) update rate. Comparatively, there has been limited results in precise operation of these platforms in outdoor or positioning-denied environments, which are common settings for potential applications of UAV platforms.

An interesting aspect of UAV research is coordinated missions between an unmanned ground vehicle (UGV) and a quadrotor UAV. Such capability greatly expands mission flexibility and range by combining the benefits of both vehicles. UGVs typically have greater payload capacity, and can be equipped with faster computation units and longer battery life, while UAVs provides much better mobility and vantage points for sensor coverage.

A core prerequisite capability for UAV-UGV coordinated mapping is autonomous docking. Due to limited payload capacity, the onboard battery size for a UAV is constrained, leading to a short operation duration (typically $\frac{1}{2}$ hour). For long missions, therefore, there is a need for a system to autonomously coordinate UAV landings for recharging or battery replacement. While successful precise docking has been demonstrated indoors [7], outdoor settings present additional challenges. The quality of outdoor position measurements is reduced in terms of accuracy and update rate. A typical GPS unit provides 3-5m accuracy at 5-10Hz. Such position measurement accuracy results in relative errors of up to 10m between the vehicles, which is far from sufficient for the purpose of autonomous landing. Also, wind gusts introduce time varying disturbances to the position controller, causing errors in position tracking during landing.

For environments where global positioning is denied, Simultaneous Mapping and Localization (SLAM) is a core capability for unmanned system autonomy, and is still a major emphasis of research effort within the robotics community. A successful SLAM algorithm enables autonomous robotic missions in areas that are dangerous or inaccessible to human beings. The objective is to construct a map of the surrounding environment while resolving the vehicle pose estimates with respect to the map. In the absence of position measurements, the map and pose estimates provided by SLAM are the basis for vehicle control, path planning and other aspects of vehicle autonomy.

1.2 Related Work

This work aims to improve on two aspects of UAV-UGV autonomy mentioned in Sec. 1.1: autonomous landing coordination between a UGV and a quadrotor UAV, and real-time SLAM deployable on UAV platforms. This section presents surveys of the current strategies for UAV autonomous landing on a static or mobile platform, multi-agent rendezvous coordination as well as SLAM algorithms.

1.2.1 Autonomous Docking

Much of the research efforts on autonomous docking of aerial vehicles focuses on landing via vision based estimation and control. In [8, 9], computer vision algorithms are developed to track a known pattern on a fixed platform and estimate the vehicle position relative to the landing pad. However, no successful landing experimental result has been reported. Also, the vision algorithms mentioned use a static thresholding to segment pattern, leading to sensitivities to lighting changes.

Saripalli and Sukhatme [10] present an algorithm to fly on Differential-GPS (DGPS) measurements, search for a visual pattern and switch to behaviour based control for landing once the pattern is detected. This algorithm is further extended in [11] to landing on a moving platform, and rely on Kalman filtering for relative positioning and behaviour based control for automated landing. No proof of stability is given; the ground vehicle is assumed to have simple 1D dynamics and the helicopter landing control is based on feeding forward the predicted errors from the simplified model.

In [12], a Infra-Red (IR) camera is mounted on a quadrotor UAV to track IR beacons on a moving platform, and a tracking position controller is implemented to take off from, follow and land on the ground platform. Again, a proof of stability is not provided in this work; the usage of IR sensors is impractical for operations in outdoor settings where IR spectrum light is often saturated in day time. Finally, the system has a very small range of operation. The quadrotor helicopter needs to precisely follow the moving platform to properly track the IR pattern, and such precise maneuvers may not be possible in an outdoor setting where wind disturbance has significant impact.

In [13], a tether is used to relay state information between a helicopter and a moving platform for performing an autonomous landing. The tether greatly limits the range of operations for small UAVs and not suitable for many other applications.

Voos and Bou-Ammar [14] present a novel controller for landing a quadrotor on a moving platform that moves freely at a constant velocity. Landing experiment results is not presented in this work and practical application for this docking strategy is not yet proven.

The landing schemes above all assume that the quadrotor is the only agent attempting to track the ground station and perform landing operations. Another approach is to coordinate a cooperative landing between a moving UGV as landing platform and a UAV. The objective of this problem is to control both vehicles such that the relative position error between the vehicles are reduced sufficiently for landing, which is essentially a multi-agent rendezvous problem.

A possible method to solve the rendezvous problem has been studied in the field of optimal trajectory planning and control. In [15, 16], the rendezvous between a spaceship and an orbiting planet is formulated as a nonlinear trajectory optimization problem. The dynamics of the spaceship and the target planet are modelled, and the continuous trajectories and control inputs of the space ship are discretized in time as optimization variables. The trajectory state variables are related to each other with motion constraints, and the control inputs constrained with upper and lower bounds, forming a standard NonLinear Programming (NLP) optimization problem to solve for a set of spaceship inputs to meet the planet at its predicted location. In cases of non-convex constraints and cost functions, local extrema are present and optimality is not guaranteed.

In [17], a Mixed Integer Linear Programming (MILP) problem is formulated to coordinate flight plans of multiple quadrotor UAVs, and the MILP problem can be solved optimally. Although the original algorithm is not proposed to solve the rendezvous problem, the architecture is generic enough that changing the cost function of the optimization problem can achieve this goal. MILPs are known to be NP-hard [18] and grows exponentially in complexity. Therefore, the MILP approach is typically not applicable to real-time control systems with hard timing constraints.

Rendezvous problem has also been a topic of interest in the field of multi-agent system. Consensus algorithm [19, 20] is a decentralized method to drive the relative distances between N mobile agents asymptotically toward zero. The communication networks between the agents can be modelled as a graph, and the agents, governed by a common control law and sharing the same motion models (generally simple linear models), are guaranteed to reach consensus assuming certain static graph topology. In [21], the algorithm is further extended to time varying graph topology. The decentralized nature of this class of algorithms leads to computation efficiency and implementation simplicity. Furthermore, it provides a method to prove convergence of relative position errors between the agents. However, consensus algorithms for mobile robotic platform with complex nonlinear plants have yet to be thoroughly explored.

1.2.2 SLAM

Bayesian stochastic estimation techniques have been well studied in the context of visual SLAM algorithms [22, 23, 24]. A dynamic motion model is defined for the camera and features are extracted and corresponded between consecutive images to provide measurements dependent on both the feature locations and the current robot pose. Both the pose and the measured feature states are estimated simultaneously through Bayesian inference, often relying on linearization about the current state estimate. As a result, these filtering methods work well for online applications, but slowly accumulate errors which lead to warping and drift in the resulting map. Only by storing all measurement data and incorporating some form of loop closure global optimization can these errors be corrected to some extent [25, 26].

Structure from Motion (SFM) is a well known problem in the field of computer vision that aims to reconstruct 3D structures observed by a sequence of 2D images. Various techniques have been developed to solve the SFM problem. The early eight-point [27] and five-point [28] algorithms solve a system of linear algebraic equations formulated by matched feature pairs from two images taken from two different camera poses. While easy to implement, the solutions are somewhat sensitive to measurement noise and correspondence errors. Bundle-adjustment techniques formulate the relative camera pose estimation as an NLP [25], solving the correspondence problem (finding the same feature in multiple images) for a large number of feature points from images taken from a small number of spatially well separated locations. This method, while well-suited for post processing and robust to noise, is not ideal for real-time applications.

Due to their dependence on point features, both SFM and Bayesian estimation methods are mostly only capable of sparse scene reconstruction, which may not be sufficient for machine perception applications such as object recognition. The recent development of Dense Tracking And Mapping (DTAM) [29] is a monocular SLAM algorithm that uses iterative image alignment to perform dense scene reconstruction and simultaneously estimate camera motion in real-time. While the results are impressive, the tracking and reconstruction performance can be sensitive to the type of camera motion and lighting conditions.

The Microsoft Kinect Red, Green, Blue and Depth (RGB-D) camera offers a low cost range sensor capable of producing dense point cloud measurements, and has generated tremendous interest from the SLAM community. The RGB-D mapping system [30] is an offline algorithm that combines RGB-D point features and the iterative closest-point (ICP) algorithm [31] for image alignment, and therefore remains subject to the sensitivity to illumination changes. A photometric ICP algorithm has been proposed [32] to align RGB-D image data based on correspondences in both geometric and colour similarities. The resulting map for both of these systems, however, is derived by simply stitching noisy depth images taken from the resolved camera pose, leading toin low quality surface reconstruction.

KinectFusion [33] is an algorithm which achieves dense scene reconstruction and ego motion estimation using the Kinect camera in real-time. The method represents the global 3D volume as a discretized Truncated Signed Distance Function (TSDF), from which a stable model of the environment surface can be extracted by ray marching through the TSDF voxel grid to identify the zero level set. An ICP algorithm, based on perspective projection association, is used to track camera pose by aligning the raw depth image with the current global surface model. The computation of the TSDF function and each of the ICP iterations is highly parallelizable, and real-time performance is achieved by the use of a Graphic Processing Unit (GPU). The resulting algorithm has been demonstrated to provide detailed 3D maps by accurately estimating 6-DOF camera motion and fusing depth images. However, the algorithm requires the entire map to be stored as a 3D voxel grid in the GPU global memory. The inefficient volume representation and limited memory resource prevent the algorithm from being used on a large scale environment, which is a common requirement for mobile robotics applications.

In a similar effort to the WAVELab KinectFusion extension described in this thesis, the recent Kintinuous project [34] removes the KinectFusion spatial limitation by representing the cubic volume about the camera as a cyclical buffer of TSDF volume slices. As a slice exit the volume, it is processed to extract resulting surface points from the dense reconstruction. The algorithm also replaces the KinectFusion ICP algorithm with the FOVIS visual odometry algorithm [35] for camera pose estimation, allowing the camera pose to be tracked in scenes with planar structures. Finally, it incorporates the visual loop closure detection with incremental Smoothing and Mapping (iSAM) [36] pose graph optimization to ensure global map consistency. The surface slices are not reused for mapping once it exits the capture volume, discarding the advantage of having a global map to match against in the original KinectFusion. Furthermore, the positioning of the TSDF volume cube centered around the camera implies that more than of cube outside of the camera fields of view, resulting in inefficient allocation of GPU memory.

In [37], LIDAR point clouds are aggregated in a 3D probabilistic occupancy grid and dense surface reconstruction is achieved through rasterization in a CPU cluster, a similar approach to the KinectFusion algorithm with different volume representation and rendering technique. The method overcomes the GPU memory limitation by decomposing the large, global voxel grid into multi-axis aligned volumetric tiles. The tiles are loaded into the GPU and processed individually to extract the reconstructed surface, and surface points from all tiles are combined to form the global surface model. The system is intended for post processing purpose, and does not provide online position estimation needed for autonomous robotics exploration in unknown environments. It does, however, inspire the extension to KinectFusion algorithm discussed in this thesis.

1.3 Research Approach and Contribution

The goal of this work is to address the two current limitations of coordinated missions with UGV-UAV teams: autonomous docking between a UGV and a quadrotor, and the development of a real-time SLAM algorithm for both vehicles.

Chapter 2 presents the system developed to autonomously coordinate docking operation between a skid-steered UGV and a quadrotor. The novelty of our system lies in the fact that both vehicles are actively actuating to minimize their relative position error. Previous research efforts assume that the ground vehicle does not have knowledge of quadrotor states, and that tracking is all performed by the quadrotor, leading to difficulty in proving stability for the overall system. By establishing a communication link between the vehicles, state information of each vehicle is shared, and a provably stable controller is developed to coordinate a rendezvous. The simulation and experimental results for successful landings are also captured in this chapter. We first validated theories developed in simulations. Using standard sensor suites for both vehicles, we successfully coordinated a landing in outdoor setting between a quadrotor UAV and a skid-steered UGV.

An algorithm to perform dense reconstruction and localization using a RGB-D camera is described in Chapter 3. With a low-cost point cloud sensor, the algorithm is capable of detail reconstruction of the environment while accurately localizing itself in real-time. Due to hardware memory limitation, this algorithm is limited in operation volume. An extension is presented to expand the operation space of this algorithm without sacrificing map quality, while maintaining real-time performance.

In Chapter 4, we present the experimental results for the extended SLAM algorithm. Real-time 3D reconstruction of a large environment is shown to demonstrate the effectiveness of the algorithm in building detailed surface model without volume constraint. The algorithm is also tested against published RGB-D camera datasets with camera pose groundtruth information to validate its localization performance. We prove that our algorithm is successful in delivering solid localization accuracy with real-time computation efficiency.

Finally, Chapter 5 provides a conclusion of the work presented in this thesis, as well as a discussion for future improvements on the developed systems.

The main results which will be presented in this thesis are:

- a novel rendezvous controller between a UGV and a quadrotor UAV
- Simulation and Experiment validation for the controller
- Verification for outdoor coordinated autonomous docking
- a SLAM algorithm using RGB-D sensor capable of generating large high-fidelity map and real-time localization
- Experimental validation for the SLAM algorithm

With these advances in autonomous rendezvous and 3D dense scene reconstruction, the goal of autonomous aerial and ground vehicle team mapping mission is two steps closer to practical deployment.

Chapter 2

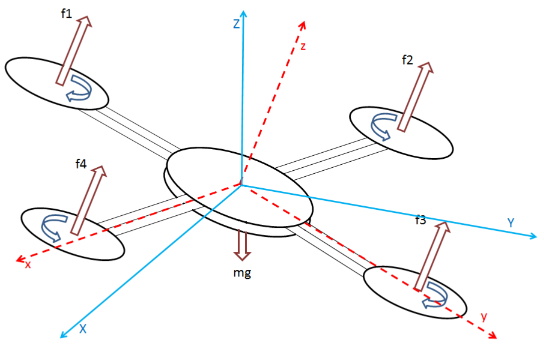
Autonomous Docking

In this chapter, an autonomous docking controller is designed to enable quadrotor vehicles to land on ground vehicles, with both vehicles aiding in the process in a coordinated manner. Feedback linearization nonlinear control technique is applied to a quadrotor UAV and a skid-steered UGV based on known models, and a provably stable linear controller is designed to coordinate rendezvous of the vehicles. Simulation results of the coordinated controller are presented to verify the stability of the controller. Using the Aeryon Scout quadrotor and the Clearpath Robotics Husky skid-steered UGV, successful landings are experimentally demonstrated in both indoor and outdoor settings. This work is an extension of the controller we presented in [38] with modifications to the quadrotor control strategy. Unlike the orignal work, we only feedback linearize the position portion of the quadrotor model, leading to ease of implementation and, therefore, achieving successful experimental results.

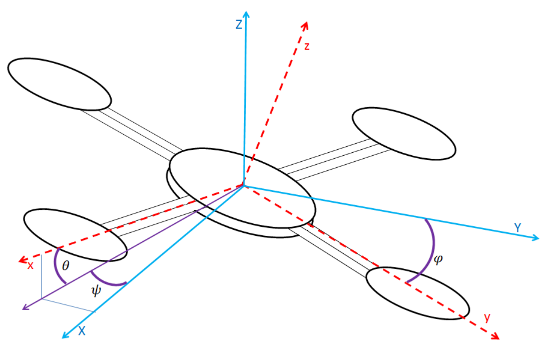
2.1 System Models

2.1.1 Quadrotor Model

Quadrotor UAVs have four rotors arranged in a cross configuration. The angular speed of each rotor can be independently controlled to produce different thrusts. Let $\Omega = (\Omega_1, \Omega_2, \Omega_3, \Omega_4)$ be the angular velocity of rotors as enumerated in Fig. 2.1(a), the thrust



(a) Freebody diagram



(b) Coordinate frames

Figure 2.1: Forces and coordinate frames of a quadrotor

generated by the i^{th} rotor, f_i, is given by $f_i = b\Omega_i^2$ and the reaction torque is obtained via $t_i = df_i$, where $b, d \in \mathbb{R}$ are the aerodynamic constants of the rotors.

The roll and pitch moments about the corresponding axes can be achieved by varying the difference in rotor speeds and, hence, the thrusts for opposite pairs of rotors; the sum of all four rotor thrusts provides acceleration in altitude, and the net torque from all four rotors results in a yaw moment.

We use the quadrotor model presented in [39] for controller design in a East-North-Up (ENU) frame of reference. Denote

$$p_q = [\ X_q \quad \dot{X}_q \quad Y_q \quad \dot{Y}_q \quad Z_q \quad \dot{Z}_q \quad \phi_q \quad \dot{\phi}_q \quad \theta_q \quad \dot{\theta}_q \quad \psi_q \quad \dot{\psi}_q\ ]$$

the state vector of the quadrotor dynamic model, where $\zeta = [X_q, Y_q, Z_q]$ is the quadrotor position coordinate vector in the inertial frame, ϕ_q, θ_q and ψ_q are Euler angles with respect to North-East-Down (NED) inertial frame axes, and $g = 9.81\text{m/s}$ is the gravitational constant, as illustrated by Fig. 2.1(b). We also denote

$$U = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -l & 0 & l & 0 \\ 0 & l & 0 & -l \\ d & -d & d & -d \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \tag{2.1}$$

the control input vector, where U_1 is the sum of thrusts of all four rotors, U_2, U_3 and U_4 the angular accelerations induced by the rotors about x, y and z body frame axes respectively,

l is the distance from a motor to the center of mass (COM) of the quadrotor. Denote $\omega = [p, q, r]^T$ the body angular rates, m the weight, $I \in \mathbb{R}^{3x3}$ the inertial matrix of the following form

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} \tag{2.2}$$

and $I_x, I_y, I_z \in \mathbb{R}$ are the inertia about x, y and z body axes. Also denote $R \in \mathbb{SO}_3$ the rotation matrix from the body frame to the global frame, and

$$R = \begin{bmatrix} c\phi_q c\psi_q & -c\phi_q s\psi_q + s\phi_q s\theta_q c\psi_q & s\theta_q s\psi_q + c\phi_q s\theta_q c\psi_q \\ c\theta_q c\psi_q & c\phi_q s\psi_q + s\phi_q s\theta_q s\psi_q & -s\theta_q s\psi_q + c\phi_q s\theta_q s\psi_q \\ -s\theta_q & s\phi_q c\theta_q & c\phi_q c\theta_q \end{bmatrix} \tag{2.3}$$

where $c\phi_q = \cos\phi_q$ and $s\phi_q = \sin\phi_q$, and the similar notation applies to θ_q and ψ_q.

The Newton-Euler equations for a quadrotor helicoptor is given by

$$\begin{bmatrix} m\ddot{\zeta} \\ I\dot{\omega} + \omega \times I\omega \end{bmatrix} = \begin{bmatrix} F \\ \tau \end{bmatrix} \tag{2.4}$$

where $F = [F_X, F_Y, F_Z]^T \in \mathbb{R}^3$ is a the sum of all external force vectors acting on the COM of the quadrotor in the global frame, $\tau = [\tau_x, \tau_y, \tau_z] \in \mathbb{R}^3$ is the sum of external torques applied on the body frame. Ignoring aerodynamic drag forces and other disturbances, the only external forces applied on the quadrotor air frame is the gravity and the total thrust U_1, giving

$$F = U_1 R e_3 - mge_3 \tag{2.5}$$

The gyrocopic torque introduced by of the rotors is given by $\tau_g = I_r \Omega_r$, where $\Omega_r = \sum \Omega_i$ is the sum of rotor speeds that introduces a gyroscopic torque affecting the vehicle. The sum of external torque, τ, is the combination of the thrust induced torque vector, $\tau_t = [U_2, U_3, U_4]^T$ and the gyroscopic torque vector, τ_g,

$$\tau = \tau_g + \tau_t \tag{2.6}$$

Combining (2.4), (2.5) and (2.6), the full quadrotor dynamic model derived using Newton formalism is the following,

$$\begin{bmatrix} \ddot{X}_q \\ \ddot{Y}_q \\ \ddot{Z}_q \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} u_x \frac{1}{m} U_1 \\ u_y \frac{1}{m} U_1 \\ (\cos\phi_q \cos\theta_q)\frac{U_1}{m} - g \\ qr\frac{I_y - I_z}{I_x} + q\frac{I_r}{I_x}\Omega_r + \frac{l}{I_x}U_2 \\ pr\frac{I_z - I_x}{I_y} + p\frac{I_r}{I_y}\Omega_r + \frac{l}{I_y}U_3 \\ pq\frac{I_x - I_y}{I_z} + \frac{l}{I_z}U_4 \end{bmatrix} \tag{2.7}$$

where

$$u_x = \cos\phi_q \sin\theta_q \cos\psi_q + \sin\phi_q \sin\psi_q \tag{2.8}$$

$$u_y = \cos\phi_q \sin\theta_q \sin\psi_q - \sin\phi_q \cos\psi_q \tag{2.9}$$

The transformation matrix, M, between Euler angular velocities and pqr rates is given by,

$$\begin{bmatrix} \dot{\phi}_q \\ \dot{\theta}_q \\ \dot{\psi}_q \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \sin\phi_q \tan\theta_q & -\cos\phi_q \tan\theta_q \\ 0 & \cos\phi_q & -\sin\phi_q \\ 0 & \frac{\sin\phi_q}{\cos\theta_q} & \frac{\cos\phi_q}{\cos\theta_q} \end{bmatrix}}_{M} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.10}$$

$$\Rightarrow \begin{bmatrix} \ddot{\phi}_q \\ \ddot{\theta}_q \\ \ddot{\psi}_q \end{bmatrix} = M \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} \tag{2.11}$$

At hover position, we have $[\phi_q, \theta_q]^T \approx \mathbf{0}$, leading to $M \approx I^{3\times3}$. Therefore, we can make the following approximations,

$$[\dot{\phi}_q, \dot{\theta}_q, \dot{\psi}_q]^T \approx [p, q, r]^T$$
$$[\ddot{\phi}_q, \ddot{\theta}_q, \ddot{\psi}_q]^T \approx [\dot{p}, \dot{q}, \dot{r}]^T$$

This gives the state space representation $\dot{p}_q = f(p_q, U)$ of the plant,

$$f(p_q, U) = \begin{bmatrix} \dot{X}_q \\ u_x \frac{U_1}{m} \\ \dot{Y}_q \\ u_y \frac{U_1}{m} \\ \dot{Z}_q \\ (\cos\phi_q \cos\theta_q)\frac{U_1}{m} - g \\ \dot{\phi}_q \\ \dot{\theta}_q \dot{\psi}_q \frac{I_y - I_z}{I_x} + \dot{\theta}_q \frac{I_r}{I_x}\Omega_r + \frac{l}{I_x} U_2 \\ \dot{\theta}_q \\ \dot{\phi}_q \dot{\psi}_q \frac{I_z - I_x}{I_y} + \dot{\phi}_q \frac{I_r}{I_x}\Omega_r + \frac{l}{I_y} U_3 \\ \dot{\psi}_q \\ \dot{\theta}_q \dot{\phi}_q \frac{I_x - I_y}{I_z} + \frac{l}{I_z} U_4 \end{bmatrix} \tag{2.12}$$

Finally, we define the position subplant as $f_{pos}(p_g, U) = [\ddot{X}_q, \ddot{Y}_q]^T$ and altitude subplant as $f_z(p_g, U) = \ddot{Z}_q$,

2.1.2 Skid-steered UGV Model

A 2-D planar kinematic model of the skid-steered ground vehicle based on work in [40] is used for the Husky UGV, and the coordinate frames are illustrated in Fig. 2.2. The Husky platform expects linear and yaw angular velocity inputs, and a kinematic model is deemed appropriate. Let the UGV state $p_g = [X_g, Y_g, \psi_g]^T$, where X_g and Y_g are the global position coordinates of the vehicle centre of mass (COM) and ψ_g is the yaw angle. The vehicle kinematic model is given by,

$$\underbrace{\begin{bmatrix} \dot{X}_g \\ \dot{Y}_g \\ \dot{\psi}_g \end{bmatrix}}_{\dot{p_g}} = \underbrace{\begin{bmatrix} \cos\psi_g & x_{ICR}\sin\psi_g \\ \sin\psi_g & -x_{ICR}\cos\psi_g \\ 0 & 1 \end{bmatrix}}_{S(p_g)} \underbrace{\begin{bmatrix} v_x \\ \omega_g \end{bmatrix}}_{v_c} \tag{2.13}$$

where $v_c = [v_x, \omega_g]^T$ represents the body frame velocity inputs with v_x being the forward velocity and ω_g the angular velocity. Furthermore, x_{ICR} is the projection of the Instantaneous Centre of Rotation (ICR) of the vehicle into the x axis of the body-fixed frame, and

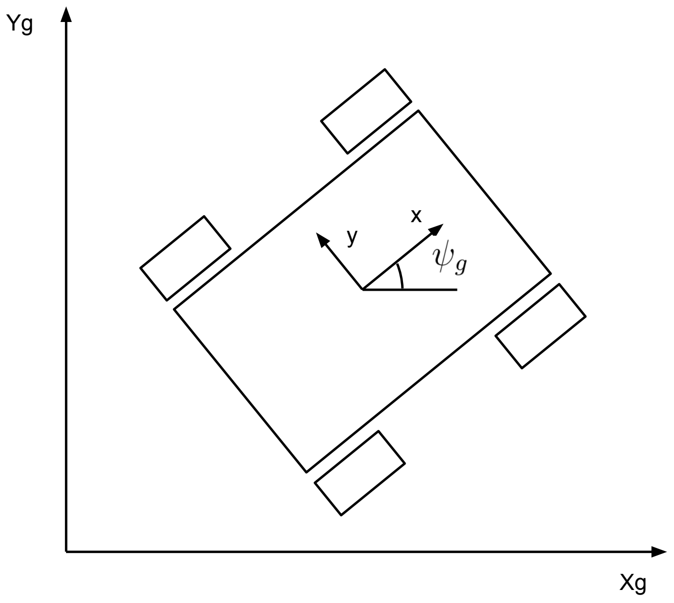


Figure 2.2: Coordinate frames for the skid-steered UGV kinematic model

a graphical illustration for x_{ICR} is given by Fig. 2.3. This projection relates the angular velocity, ω_g, to the y component of body frame velocity, v_y, via $\omega_g = -v_y/x_{ICR}$ [41]. It is the local x projection of the point about which the vehicle rotates at any given instant. In practice, due to the skid steering, the location of the ICR will move [41]. However, in this work a constant approximation of the x_{ICR} is used, denoted x_0. This approximation implies the assumption that the ratio between foward velocity, v_x, and lateral velocity, v_y, is constant and the ICR is always in the same location for a given steering direction. While we ignore this assumption in the controller design for the UGV in Sec. 2.2.2, it can be demonstrated in simulation that the error introduced is reasonably inconsequential. In fact, it has been shown in [40] that using a constant non-zero x_0 in the controller imposes a motion constraint on the vehicle that limits lateral skidding, which might lead to loss of motion stability.

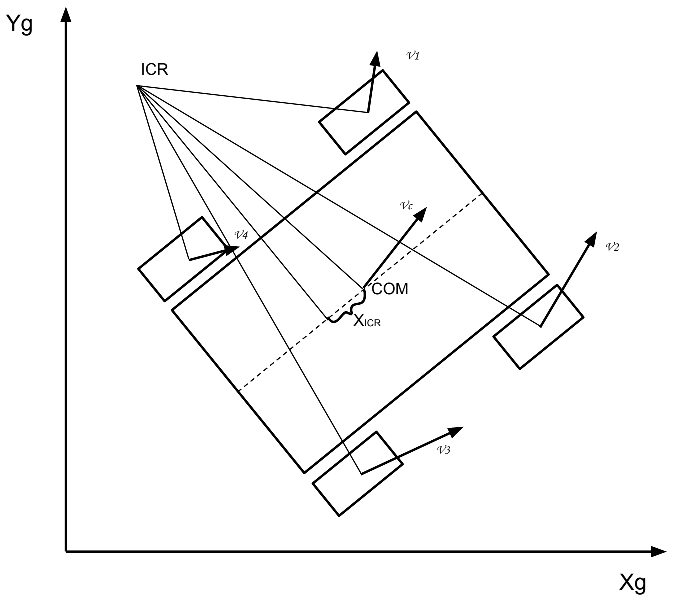


Figure 2.3: Body, wheel velocity relationships and the resulting x_{ICR}

2.2 Controller Design

2.2.1 Quadrotor Feedback Linearization

This section entails the derivation for the feedback linearization of the position subplant of the quadrotor model, $f_p os$, presented in [39]. We firstly start with the altitude subplant, f_z, as follows

$$f_z(p_q, U_1) = \frac{\cos\phi_q \cos\theta_q}{m} U_1 - g \tag{2.14}$$

Using the following control law for the total thrust input v_z

$$U_1 = \frac{m}{\cos\phi_q \cos\theta_q}(v_z + g) \tag{2.15}$$

we can feedback linearize the model into the following form $\forall \phi_q, \theta_q \in (-\frac{\pi}{2}, \frac{\pi}{2})$

$$f_z(p_q, U_1') = \ddot{Z}_g = v_z \tag{2.16}$$

where the set $(-\frac{\pi}{2}, \frac{\pi}{2})$ ensure that $\cos\phi_q \cos\theta_q$ is nonzero and divisible.

After computing total thrust U_1 from (2.15), we can use it to compute θ_q and ϕ_q required to regulate x and y positions. It is known from (2.8) and (2.12) that

$$\begin{aligned}\begin{bmatrix}\ddot{X}_g \\ \ddot{Y}_g\end{bmatrix} &= \begin{bmatrix}u_x \\ u_y\end{bmatrix}\frac{U_1}{m} \\ &= \begin{bmatrix}\cos\phi_q \sin\theta_q \cos\psi_q + \sin\phi_q \sin\psi_q \\ \cos\phi_q \sin\theta_q \sin\psi_q - \sin\phi_q \cos\psi_q\end{bmatrix}\frac{U_1}{m}\end{aligned} \tag{2.17}$$

Using small angle approximation we can compute that, we convert (2.17) into the following

$$\begin{aligned}\begin{bmatrix}\ddot{X}_g \\ \ddot{Y}_g\end{bmatrix} &= \begin{bmatrix}\theta_q \cos\psi_q + \phi_q \sin\psi_q \\ \theta_q \sin\psi_q - \phi_q \cos\psi_q\end{bmatrix}\frac{U_1}{m} \\ &= \begin{bmatrix}\cos\psi_q & \sin\psi_q \\ \sin\psi_q & -\cos\psi_q\end{bmatrix}\begin{bmatrix}\theta_q \\ \phi_q\end{bmatrix}\frac{U_1}{m}\end{aligned}$$

We choose the virtual input vector $v_q \in \mathbb{R}^2$ such that

$$\begin{aligned}\begin{bmatrix}\theta_q \\ \phi_q\end{bmatrix} &= \begin{bmatrix}\cos\psi_q & \sin\psi_q \\ \sin\psi_q & -\cos\psi_q\end{bmatrix}\begin{bmatrix}v_{q,1} \\ v_{q,2}\end{bmatrix}\frac{m}{U_1} \qquad (2.18) \\ \Rightarrow \begin{bmatrix}\ddot{X}_q \\ \ddot{Y}_q\end{bmatrix} &= \begin{bmatrix}v_{q,1} \\ v_{q,2}\end{bmatrix}\end{aligned}$$

Using these virtual inputs, we feedback linearize the quadrotor position plant as a linear double integrator in x, y and z directions. When v_q is computed, we derive the desired attitude commands, denoted as ϕ_d and θ_d, using (2.18) in the place of ϕ_q and θ_q. Relying on time scale separation of the attitude and position control loops, and we use a PID controller for the attitude inner loop control to track ϕ_d and θ_d.

2.2.2 UGV Feedback Linearization

To design a controller for the skid-steered UGV model mentioned in Sec. 2.1.2, input-output feedback linearization is used to derive a linear plant in the UGV inertial positions, X_g and Y_g based on the work in [40, 38]. Choosing X_g and Y_g as outputs, the input-output dynamics of the system is as follows,

$$\begin{bmatrix}\dot{X}_g \\ \dot{Y}_g\end{bmatrix} = \begin{bmatrix}\cos\psi_g & x_0\sin\psi_g \\ \sin\psi_g & -x_0\cos\psi_g\end{bmatrix}\begin{bmatrix}v_x \\ \omega_g\end{bmatrix} \tag{2.19}$$

The feedback linearizing controller for this set of dynamics is determined to be,

$$\begin{bmatrix} v_x \\ \omega_g \end{bmatrix} = \begin{bmatrix} \cos\psi_g & \sin\psi_g \\ \frac{\sin\psi_g}{x_0} & -\frac{\cos\psi_g}{x_0} \end{bmatrix} \begin{bmatrix} v_{g,1} \\ v_{g,2} \end{bmatrix} \tag{2.20}$$

Applying this controller to (2.19) yields $[\dot{X}_g, \dot{Y}_g]^T = [v_{g,1}, v_{g,2}]^T$.

2.2.3 Joint State Rendezvous Control

This section presents a control approach proposed by our work in [38] to coordinate a rendezvous between the quadrotor and the ground vehicle with a slight modification. The original work employs a full feedback linearization of the quadrotor model, which results in quintuple integrators in X, Y and Z directions. This leads to difficulties in practical implementation of the controller due to the requirement of the 5th order time derivative of the states, which is not measurable and noisy to estimate. We, therefore, choose to feedback linearize the inertial position portion of the quadrotor model into double integrators in its inertial position. The primary goal of this controller is to drive the relative X-Y position error of the two vehicles to zero. If only position errors are stabilized, the vehicles may continue to drive together at some unknown velocity, resulting in unpredictable behaviours. Hence, the decision was made to ensure that the velocities of both vehicles are also driven to zero. Once landing is successfully performed, the ground vehicle can carry on with the rest of the mission while transporting the quadrotor.

The feedback linearization controllers mentioned in Sec. 2.2.1 and Sec. 2.2.2 result in linear plants for both of the vehicles. The closed feedback linearized model for the quadrotor position plant $[\ddot{X}_q, \ddot{Y}_q]^T = v_q$ and $[\dot{X}_g, \dot{Y}_g]^T = v_g$ for the UGV. The goal is to ensure that $X_q - X_g = X_q - Y_g = 0$ and $\dot{X}_q = \dot{Y}_q = 0$. To stabilize inertial velocities for both vehicle to zero, we only need to ensure that the quadrotor velocity and the relative position of the vehicles converges to the equilibrium. In developing the controller, since the plants for X and Y are decoupled and symmetric, we only need to design a control law for X and apply the same control to Y. It is possible to extend this controller to the Z state as well. However, it is much safer to command the quadrotor in Z only once one is certain that the X and Y positions of the vehicles are close to each other.

Defining $e \in \mathbb{R}^2$, the relative position error of the vehicles, as,

$$e = \begin{bmatrix} X_q \\ Y_q \end{bmatrix} - \begin{bmatrix} X_g \\ Y_g \end{bmatrix} \tag{2.21}$$

the state vector of the dynamics to be stabilized may be expressed as $[e_1, \dot{X}_q]^T$. The dynamics of the outputs are given by,

$$\begin{bmatrix} \dot{e}_1 \\ \ddot{X}_q \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ \dot{X}_q \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v_{g,1} \\ v_{q,1} \end{bmatrix}$$

evidently, rank of the controllability matrix for the dynamics above is 2. The system is, therefore, completely controllable and can be stabilized with the following control law,

$$\begin{bmatrix} v_{g,1} \\ v_{q,1} \end{bmatrix} = -K \begin{bmatrix} e_1 \\ \dot{X}_q \end{bmatrix} \tag{2.22}$$

where $K \in R^{2\times 2}$ can be selected using standard linear state feedback control techniques such as pole placement or LQR design.

2.3 Experimental Platforms

The autonomous docking experiments are performed using a Aeryon Scout quadrotor and a Clearpath Husky skid-steered UGV.

The Aeryon Scout is equipped with a U-Blox GPS unit capable of outputting raw satellite information for Real-Time Kinematic (RTK) GPS fix acquisition. It also carries an on-board Inertial Measurement Unit (IMU) that includes a 3-axis gyroscope and accelerometer, as well as a magnetometer. The measurements from these sensors are fused with a complementary filter to estimate its roll, pitch and yaw for attitude and position regulation at 100Hz. The quadrotor is designed to have the capability of maintaining position in 70km/h winds, which is a desirable feature for our outdoor experiments. Modeling the quadrotor position subplant as a linear double integrator, we designed a Kalman filter to estimate the position vehicle states, including inertial frame quadrotor position and velocity required by the joint state feedback rendezvous controller in Sec. 2.2.3. Fig. 2.4 is a picture of the Aeryon quadrotor platform.

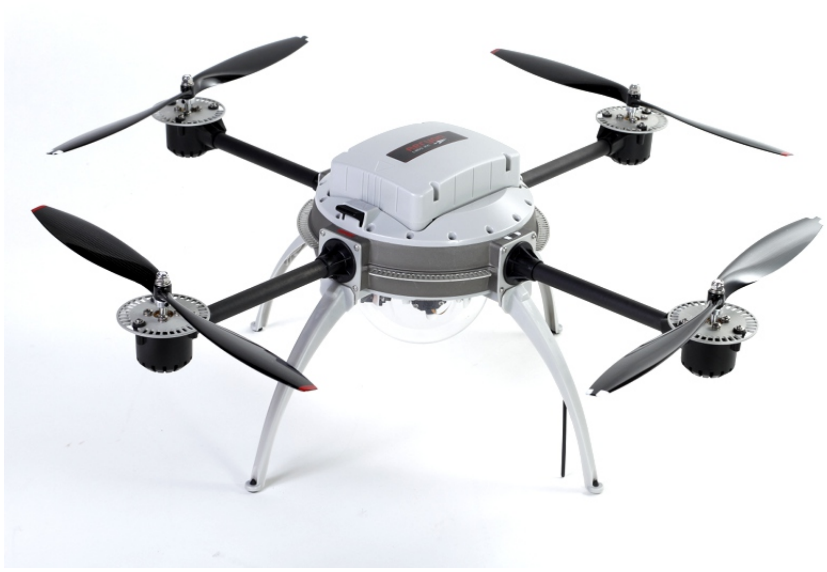


Figure 2.4: Aeryon Scout [1]

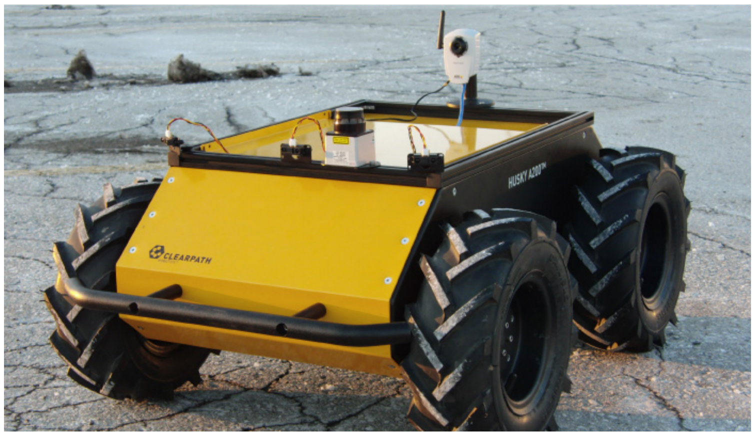


Figure 2.5: Clearpath Huskty [2]

The Clearpath Robotics A200 Husky, as shown in Fig. 2.5, also carries a U-Blox unit for RTK GPS. It is equipped with a digital compass for heading information, as well as an optical encoder on each side of the differential drive for odometry measurements and velocity control. The UGV can be controlled by requesting linear and angular velocities, and is an ideal option for the kinematic control scheme we developed.

Indoor landing experiments are performed within a OptiTrack positioning system. IR reflectors are mounted on both the vehicles, and six IR cameras are positioned around the test environment to capture the reflectors and triangulate their 3-D positions. The system provides 50Hz of position and yaw measurements with sub-centimetre, sub-degree accuracy.

For outdoor relative position feedback, an RTK GPS computation engine is also implemented to provide accurate relative position error between the quadrotor and the UGV. C/C++ software is written to interface with the U-Blox GPS to extract raw satellite information. A User Datagram Protocol (UDP) communication socket is established between the vehicles over WiFi, and raw satellite packets are sent from the UGV to the quadrotor, where the RTK fix computation takes place. The relative position error $[e_1, e_2]$ computed from the RTK fix along with inertial quadrotor velocities $[\dot{X}_q, \dot{Y}_q, \dot{Z}_q]$ are subsequently relayed back to the Husky as required by the UGV portion of the joint rendezvous controller.

We propose to control the height of the quadrotor UAV using the relative altitude error from the RTK GPS. Conventional systems use a SOund Navigation And Ranging (SONAR) sensor to measure the distance from the UAV to the closest surface within the SONAR measurement volume. This leads to discrete jumps in height measurements as the quadrotor flies over the landing platform of the UGV, resulting in a sudden increase in thrust as the height controller tries to overcome the change in altitude measurement that results. Such behaviors cause disturbances in position control during the critical landing stage when the UAV is within the proximity of the landing platform. The RTK relative altitude information, on the other hand, provides much more consistent height measurements, and is deemed more suitable for this application.

2.4 Simulation Results

The controller described in Sec. 2.2 is now verified in simulations. The joint controller is used to control the inertial X and Y positions of both vehicles, while the altitude of the quadrotor is controlled using the feedback linearizing tracking controller.

The effects of wind gusts and sensor noise are included in the simulations. Varying wind speed is simulated using the Dryden wind gust model, and affects the quadrotor through the resulting aerodynamic drag force, F_d, given the following model,

$$F_d = \frac{1}{2}\rho v^2 C_d A \tag{2.23}$$

where v is the relative speed between the wind and the quadrotor, the coefficient of drag for the quadrotor, C_d, was chosen as 0.5, and the reference area of the vehicle, A, was computed as 0.2027 m^2 assuming a circle with a radius of 25 cm. Zero mean Gaussian noise is added to all measurements. The standard deviation of the noise added to position measurements is 10 cm. The orientation measurements are corrupted with noise that has a standard deviation of 5 degrees, and orientation rate measurements with noise having a standard deviation of 0.1 degrees/s.

In the UGV model, x_0 is set to 3.7 cm, and the same value is used in the UGV controller. The initial state is $[X_g, Y_g, \psi_g]^T = [0, 1, 45]^T$, where the linear units are in metres and the angular units in degrees. For the quadrotor, the full nonlinear dynamic model in (2.7) is used to test the robustness of this controller against remaining nonlinearities. The mass, m, was set to 1 kg, I_r is set to 0.01, and $I_x = I_y = I_z = 1$. Since this work is not focused on system identification of quadrotor parameters, values used for the system parameters are not necessarily realistic, and the re-tuning of the controller is expected when testing on real platforms.

A PD controller is used for attitude inner loop control, and the proportional gain is chosen as $k_p = 10$ and derivative gain as $k_d = 30$ for ϕ_q, θ_q and ψ_q. The gains for the joint controller (2.22) were chosen as $K_1 = 0.04$, $K_2 = 0.2$. Tracking in Z_q is also achieved using a PD controller to control the double integrator altitude subplant (2.16) as a result of feedback linearization in (2.15), with the proportional gain $k_{z,p} = 0.04$ and derivative gain as $k_{z,d} = 0.2$. To ensure that the reference trajectory for Z_q is a gradual descent for safe landing operation, the reference signal for Z_q was passed through a first

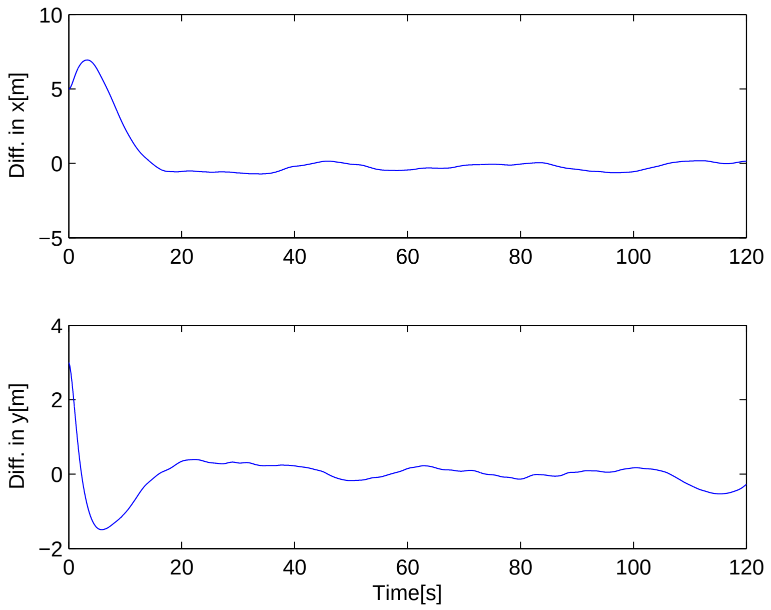


Figure 2.6: Difference between the quadrotor and ground vehicle trajectories in simulation

order low-pass filter with poles at $s = -2$. The initial pose of the quadrotor was set to $[X_q, Y_q, Z_q, \phi_q, \theta_q, \psi_q]^T = [5, 4, 5, 10, 10, 10]^T$, where the linear units are in meters and the angular units in degrees. The initial linear and angular velocities are set to zero for both vehicles. The simulations were implemented using a first order Runge-Kutta solver with a fixed sample period of 0.01 seconds.

The position states of the quadrotor are shown for this simulation in Fig. 2.8. After 120 seconds, the X and Y states have converged. The Z_q state remains stable and executes the commanded landing trajectory from 5m to 1m after 25 seconds. This demonstrate the robustness of the proposed controller against sensor noises and wind disturbances.

2.5 Experimental Results

Using the experimental setup described in Sec. 2.3, landing experiments were performed in both indoor and outdoor environments.

A successful indoor landing experiment within the OptiTrack positioning environment

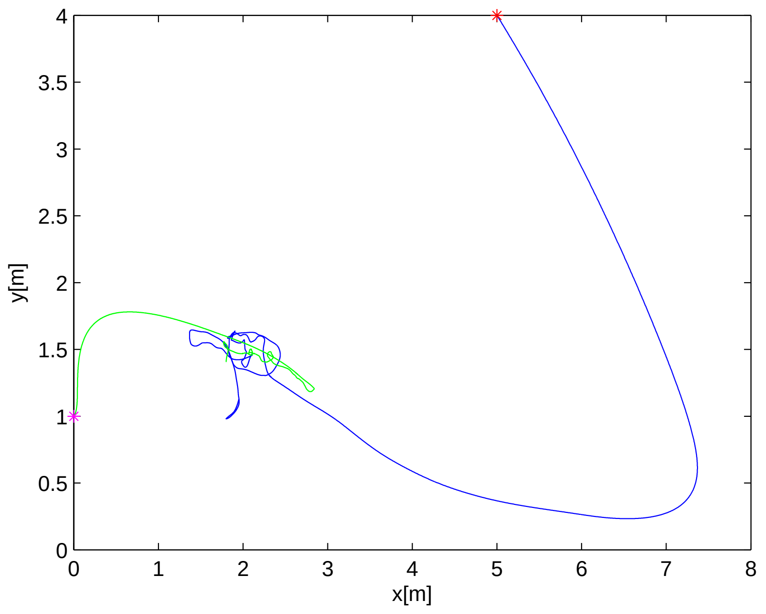


Figure 2.7: Simulated trajectories of the quadrotor (blue) and ground vehicle (green)

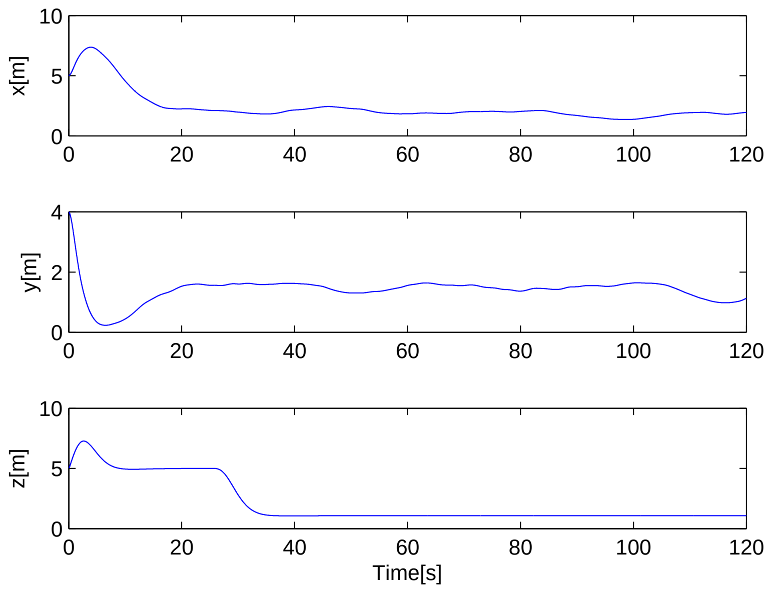


Figure 2.8: Position of the quadrotor for the simulation

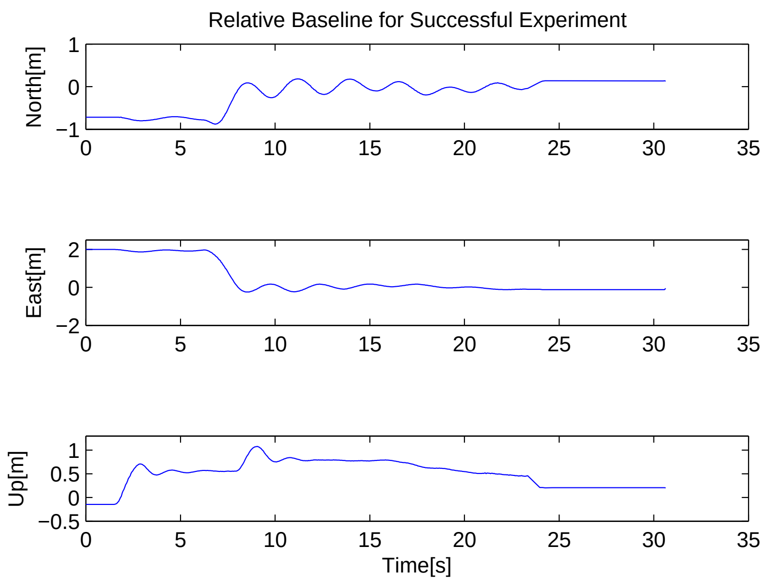


Figure 2.9: Relative error during a landing from RTK baseline measurements

is recorded. As shown in the top-down view of the vehicle trajectories in Fig. 2.9, both the ground vehicle and the quadrotor drive toward each other to eliminate the relative position error. The error quickly decays toward zero and oscillate about the equilibrium as observed in Fig. 2.10. As the relative position error becomes small enough at 24 second mark of the experiment, a landing command is triggered, and the quadrotor landed on the UGV landing pad shown by the pink square. The oscillation may be caused by the transient in quadrotor roll and pitch to meet the requested angles by the position controller, which introduces unmodelled phase lag into the control system.

Successful outdoor landings are performed in the presence of wind disturbances and sensor noises. Integral efforts are added to the quadrotor portion of the joint controller to overcome the wind disturbances. The control inputs on the quadrotor are saturated to prevent exit from the region of attraction due to large position error.

The RTK baseline measurements from a successful outdoor landing in East-North-Up (ENU) are presented in Fig. 2.11. As shown, the relative error between the vehicles quickly decreases within the first 10 seconds of the experiments, and oscillates about the origin due to the varying wind disturbance at 15km/h and sensor noises in GPU and IMU

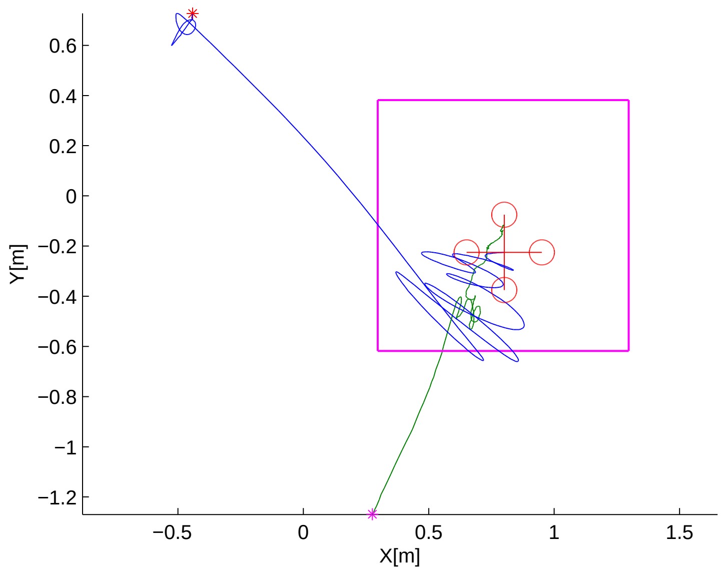


Figure 2.10: Trajectories taken by the quadrotor (blue) and ground vehicle (green) during an indoor landing experiment

measurements. The altitude subplot in Fig. 2.12 indicates oscillation in height control due to the impact of ground effect. At the 10s mark, as the quadrotor X-Y relative position error approaches zero, the quadrotor is above the ground vehicle, as indicated by the steady difference in altitude measured by the the sonar and GPS sensors. Large variations in the sonar measurement are then visible at 25, 30, 38, 52 and 59 seconds. The result is significant increase in the thrust produced by the quadrotor when over the ground vehicle due to ground effect, and this leads to oscillation in the GPS altitude control. This unmodelled disturbance also leads to extra horizontal acceleration at nonzero ϕ_q and θ_q, also contributing to the observed oscillation in X-Y plane. Such oscillations continue until the relative errors and quadrotor horizontal velocities are small enough for safe docking, and a landing command is triggered at 50s mark. Videos are also captured for a successful indoor landing at `http://tinyurl.com/d93jgwd` and an outdoor landing at `http://tinyurl.com/98rpz7b`.

One advantage of joint controller docking strategy is its cooperative nature. Fig. 2.13 contains a 2D plot of the trajectories of both vehicles during the outdoor docking experiment. The UGV trajectory is derived from the NED coordinate measurements from

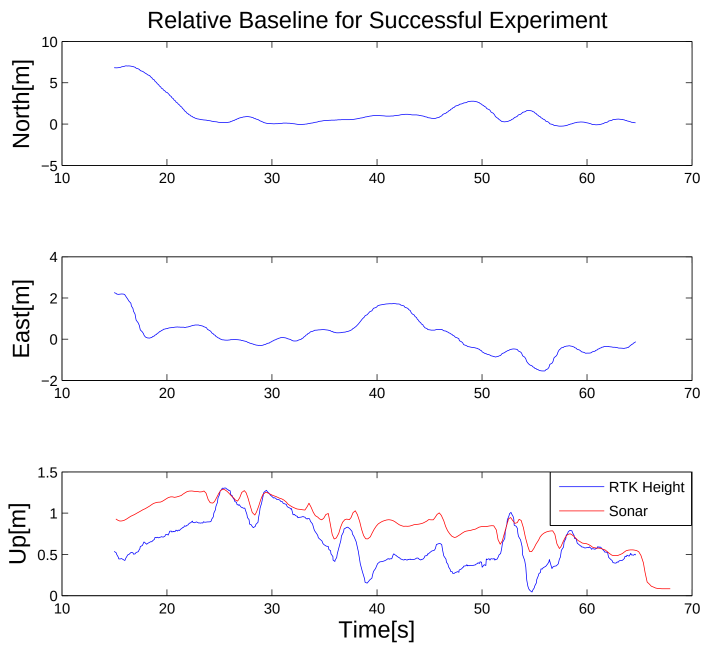


Figure 2.11: Relative error during a landing from RTK baseline measurements

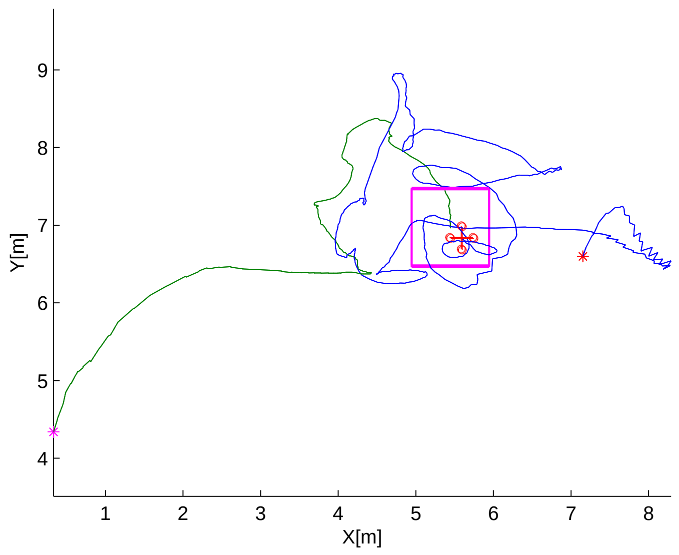
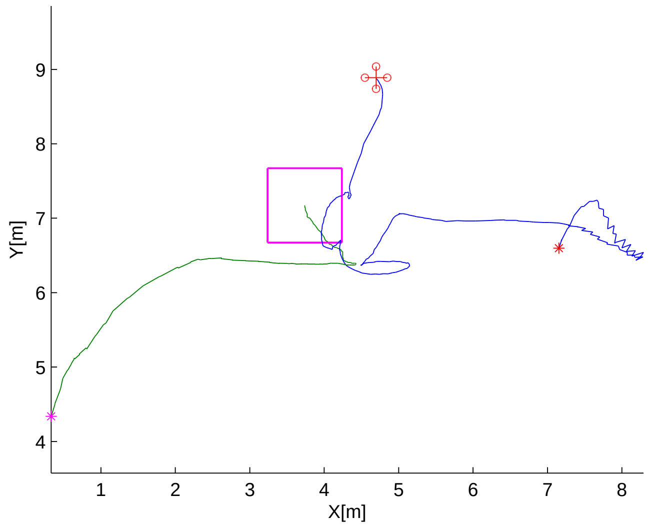
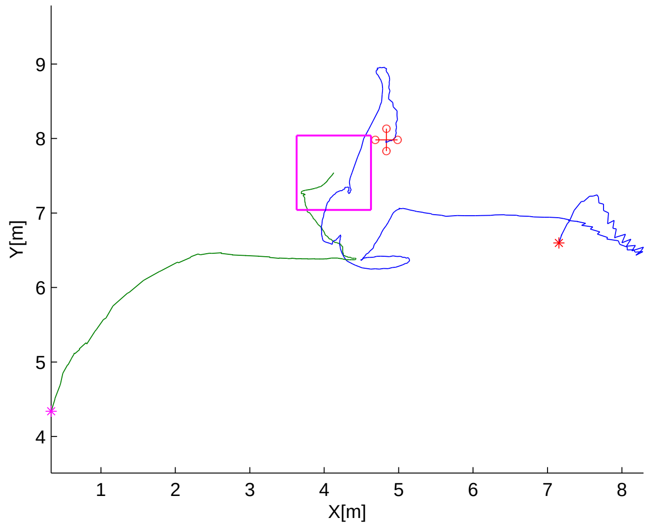


Figure 2.12: Trajectories taken by the quadrotor (blue) and ground vehicle (green) during an outdoor landing experiment



(a) Large relative error introduced by wind at the 42s mark



(b) At the 45s mark, both vehicles are actively attempting to eliminate relative error

Figure 2.13: Trajectories taken by the quadrotor (blue) and ground vehicle (green)

the GPS, while the quadrotor trajectory is extracted by offsetting each measurement of the UGV trajectory with the RTK baseline measurements at the same time step. In Fig. 2.13(a), a spike in position error is introduced when the quadrotor is driven away by a wind disturbance, while the trajectories shown in Fig. 2.13(b) indicate that both vehicles are actively trying to converge to each other and eliminate relative position error, effectively utilizing all available control inputs of the UGV-UAV joint system.

We have successfully demonstrated autonomous coordinated landing of a small UAV on a ground vehicle, in the presence of wind and despite difficulty with ground effect. This paves the way for multi-vehicle teams to be applied to persistent surveillance problems, and allows short lived UAVs to operate indefinitely with recharge or swap on moving support platforms.

Chapter 3

Dense Reconstruction and Localization with RGB-D Camera

While landing is a core capability for long range, long duration UGV-UAV coordination in outdoor setting, mapping and localization is essential for safe operations for both vehicles in unknown environment. With this purpose in mind, the spatially extended KinectFusion algorithm is developed using the Microsoft Kinect RGB-D camera to perform dense reconstruction of the physical environment while localizing the camera pose with respect to the constructed map.

The original KinectFusion algorithm [33] stores the global 3-D map in a voxel grid containing values of a discretized truncated signed distance function (TSDF), which is a mathematical tool to encode the locations of vertices in the surface model of obstacles. The TSDF grid is used to aggregate depth measurements into a global map through weighted averaging, effectively removing raw depth image noise from the resulting surface reconstruction. From a given camera pose, a stable surface model of the environment can be extracted by performing ray tracing in the TSDF voxel grid to locate the surface vertex. While TSDF leads to high fidelity surface reconstruction, it is memory inefficient, limiting the original KinectFusion algorithm in small work space. In this work, the TSDF voxel grid is divided into smaller, multi-axis aligned grid of TSDF tiles, and a swapping system between Graphics Processing Unit (GPU) memory and the hard-drive is implemented to enable large scale maps to be constructed.

Another issue with the original KinectFusion algorithm lies in its inability to track camera position when viewing environment that lacks geometric textures. To address this issue, a visual odometry algorithm is developed in house to extract and track features in subsequent images with the optical flow method. The flow vectors are processed with the RANdom SAmple Consensus (RANSAC) algorithm [42] to estimate the relative camera motion, while rejecting outlying measurements. Using the visual odometry camera motion estimate as an initial solution, the modified point-to-plane iterative closest point (ICP) algorithm [31] from KinectFusion is used to refine camera pose solution by aligning the raw depth image with the current global surface model when geometric texture is present. This hybrid approach combines color and shape information for camera pose estimation, allowing robust localization in real-time.

The algorithm consists of the following main steps.

1. **New Measurement Processing**: The raw Kinect data is processed to extract a vertex and a surface normal at each pixel.

2. **Camera Pose Estimation**: The new vertex image is aligned with the global surface model using an ICP algorithm, generating a rigid transformation matrix to describe the camera pose.

3. **Memory Management**: The active and inactive map regions are identified based on the current camera pose and sensor measurement volume, and appropriate memory slices are swapped between hard drive and GPU memory.

4. **TSDF Update**: The TSDF voxel grid is updated with a weighted average of the new depth map and the existing voxel grid values, based on the estimated camera pose.

5. **Surface Prediction**: Ray marching is performed at each image pixel to find the zero crossing of the TSDF voxel grid, to update the global surface model matched by a ICP algorithm for the next iteration.

Using these five steps, the proposed algorithm leads to a rich description of the environment while provide accurate localization information in real-time, further enhancing the autonomy of UAV-UGV team in positioning denied environment.

3.1 Preliminaries

The intrinsic characteristics of the Kinect depth camera are captured by the camera calibration matrix $K \in \mathbb{R}^{3\times 3}$, and $\mathcal{U} \subset \mathbb{R}^2$ is the set of pixel locations in the image. A ray $r \in \mathbb{R}^3$ from pixel location $u \in \mathcal{U}$ can be computed via $r = K^{-1}\dot{u}$, where $\dot{u} = [u, 1]^T$ is the homogeneous vector of u. A point p_c in the camera coordinate frame can be projected onto the image plane via $u = \pi(Kp_c)$, where $\pi : \mathbb{R}^3 \to \mathbb{R}^2$ transforms a point, $p_c = [x, y, z]^T \in \mathbb{R}^3$ to a point on the image plane, $u = [\frac{x}{z}, \frac{y}{z}] \in \mathbb{R}^2$. Denote p_i as the i^{th} element of a point, $p \in \mathbb{R}^3$, where $i \in \{1, 2, 3\}$.

The inertial camera 6DOF pose at time step k is encoded within the rigid transformation matrix from camera coordinate frame c to global coordinate frame g,

$$T^g_{c,k} = \begin{bmatrix} R^g_{c,k} & t^g_{c,k} \end{bmatrix}$$

where $R^g_{c,k} \in \mathbb{SO}_3$ is a rotation matrix and $t^g_{c,k} \in \mathbb{R}^3$ is the translation vector. The global location $p_g \in \mathbb{R}^3$ of a point $p_c \in \mathbb{R}^3$ in the camera frame is computed via $p_g = T^g_{c,k}\dot{p}_c$. The homogeneous form of $T^g_{c,k}$ is given by,

$$\dot{T}^g_{c,k} = \begin{bmatrix} R^g_{c,k} & t^g_{c,k} \\ \mathbf{0} & 1 \end{bmatrix}$$

and let $w(\dot{T}^g_{c,k}) = T^g_{c,k}$

Each new depth image at time step k is denoted D_k, while the vertex map and surface normal in coordinate frame j are denoted as $V_{j,k}$ and $N_{j,k}$. Denote $D_k(u) \in \mathbb{R}$ as the depth value of the point recorded in image pixel $u \in \mathcal{U}$. Similarly, denote $V_{j,k}(u) \in \mathbb{R}^3$ as the surface vertex point projected on image pixel u, and $N_{j,k}(u) \in \mathbb{R}^3$ the surface normal vector for $V_{j,k}(u)$. We define an image feature as $f = [u_f, d_f]^T$, where $u_f \in \mathcal{U}$ is the image pixel location and d_f is the depth value associated with this pixel. A feature vector $F_k = f_1, f_2, ..., f_n$ is a set of n image features extracted from an RGB-D image I_k at time step k.

A Truncated Signed Distance Function (TSDF) voxel grid consists into smaller chunks of TSDF volumetric tiles. Let C_k denote the active set of TSDF tiles, or those tiles that reside in the GPU memory at the k^{th} iteration; the set A_k contains the list of memory slots available for loading from the file system; and the proximity set, B_k, indicates the set of TSDF tiles whose volume center positions are within a certain constant radius of the camera position, $t^g_{c,k}$. The function $n : X \to \mathbb{Z}^+$ returns the number of element in set X.

3.1.1 Optical Flow

Optical flow is a well-studied technique for estimating relative motion between image frames. Given two subsequent images in time, I_{k-1} and I_k, feature set F_{k-1} is a set of point feature extracted from I_{k-1}. Provided with a squared window size of w pixels, denote the tracking region for each feature, $f \in F_{k-1}$, at pixel location u_f as $W_f = \{u \in \mathcal{U} | u_1 \in [u_{f,1} - w, u_{f,1} + w], u_2 \in [u_{f,2} - w, u_{f,2} + w]\}$. The objective of optical flow problem is to estimate the relative motion, $d \in R^2$, that image patch W_f about each $f \in F_{k-1}$ experience between I_{k-1} and I_k, also known as the optical flow of W_f.

The following assumptions are made for optical flow algorithm to function:

Assumption 3.1. *All pixels within W_f experience the same displacement vector $d \in \mathbb{R}^2$.*

Assumption 3.2. *The overall image intensity of pixels within W_f stay constant spatially and temporally, i.e.*

$$I_{k-1}(u) = I_k(u + d), \quad \forall\, u \in W_f \tag{3.1}$$

Assumption 3.3. *The relative motions and time difference between consecutive images are small.*

Assumption 3.4. *All pixels within the window, W_f, for each feature, $f \in F_{k-1}$, in I_{k-1} are also viewable in the subsequent image I_k*

Assumption. 3.2 provides the basis for formulating the optical flow solution as an optimization problem. Given the small relative spatial and temporal difference between two subsequent image frames in Assumption. 3.3, we can perform Taylor series expansion on $I_k(u + d)$ as follows,

$$I_k(u + d) = I_{k-1}(u) + \bigtriangledown I(u)d + I_t(u)\Delta t \tag{3.2}$$

where $\bigtriangledown I(u) = [I_x(u), I_y(u)] \in \mathbb{R}^2$ is the spatial gradient at pixel u, typically calculated by filtering the image with edge detection gradient kernels; and I_t is the derivative of image intensity over time at u and Δt is the time difference between two image frames. We substitute 3.2 into 3.1 to derive the following,

$$\begin{aligned} I_{k-1}(u) &= I_{k-1}(u) + \bigtriangledown I(u)d + I_t(u)\Delta t & (3.3) \\ 0 &= \bigtriangledown I(u)d + I_t(u)\Delta t & (3.4) \end{aligned}$$

We can use (3.4) to formulate a least square problem,

$$J = \sum_{u \in W_f} \| \bigtriangledown I(u)d + I_t(u)\Delta t\|_2^2 \tag{3.5}$$

and optimal solution d^* for J is given by

$$\hat{d}^* = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}^{-1} \begin{bmatrix} \sum I_x I_t \Delta t \\ \sum I_y I_t \Delta t \end{bmatrix} \tag{3.6}$$

where $\hat{d}^*$ is the estimated optical flow vector experienced by image patch W_f.

Using the process illustrated above, we can estimate the optical flow of the image patch about every feature $f \in F_{k-1}$ to generate their corresponded feature f' in I_k via $f' = [u_f + \hat{d}^*, D_k(u_f + \hat{d}^*)]^T$. This results in a vector of corresponded RGB-D feature pairs, which can be used to estimate 6 Degree Of Freedom (DOF) relative motion the camera experiences between subsequent images, I_k and I_{k_1}.

The issue with the optical flow approach is that its derivation is based on the strong assumptions of spatial and temporal locality, and it suffers from erroneous flow estimation when these assumptions are violated. As a result, a method is needed to detect and reject erroneous optical flow vectors the relative 6DOF motion solution. The RANSAC algorithm, as described in Sec. 3.1.2 can be used for this purpose.

3.1.2 RANSAC Algorithm

The RANSAC algorithm [42] is a probabilistic method for fitting a mathematical model on a set of measurements. It differs from the regular least square fitting scheme in that the RANSAC algorithm fundamentally assumes the existence of outliers in the measurements. Least square model fitting includes all samples within the solution for the model, and results in optimal solutions when the noise in measurements is sampled from a zero-mean Gaussian distribution. The presence of outliers, however, injects large error into the quadratic cost evaluation that dramatically degrades the resulting solution.

To mediate the impact of outliers, RANSAC iteratively selects a smaller cluster of samples from the measurement set, and a model is fit on the cluster of samples. This model is subsequently evaluated by checking if other remaining samples agree with the model, and store the model with the highest number of inliers. This process is repeated for a certain number of iterations, and the final model is generated from the inliers of the best model.

Let $\mathbb{M}$ the measurement space that contains all permutations of a generic measurement type, $\mathbb{S}$ the model space that includes all models generated from any set of measurements $M \in \mathbb{M}^n$. We define $\alpha : \mathbb{M}^n \to \mathbb{M}^m$ as a function that takes in a set of measurements and returns a random subset of the input with size m where $n, m \in \mathbb{R}$ and $n \geq m$; the function $\beta : \mathbb{M} \to \mathbb{S}$ returns a model in $\mathbb{S}$ derived from a measurement set of size m; finally, $\gamma : \mathbb{M} \times \mathbb{S} \to \mathbb{R}$ take a measurement and a model as inputs, and returns 1 if the measurement is considered an inlier of the measurement set described by the model, 0 if otherwise. Given these definitions, Algorithm. 1 illustrates the operations of RANSAC.

The RANSAC algorithm has been applied extensively to fitting a rigid transformation model for visual odometry, which refers to the estimation of relative camera motion between two image frames. The common approach is to use a either feature tracking technique (optical flow) or feature matching (extracting point features and matching them across two image frames) techniques to generate a set of corresponded feature pairs. The correspondences are subsequently filtered using the RANSAC algorithm to reject the outliers, and a final model is computed from the remaining feature pairs.

3.1.3 Rigid Transformation Model Fitting

The section describe a method to calculate the relative motion experienced by the camera between two subsequent image frames, given two corresponded set of RGB-D feature set F_{k-1} and F_k. Firstly, a RGB-D feature f can be converted to a 3-D point p via $p = d_f K^{-1} \dot{u}_f$, and denote P_K the 3-D point set generated from features from F_k.

The rigid transformation model fitting technique from [43] is used in the RANSAC visual odometry algorithm in place of the function $\beta : \mathbb{M} \to \mathbb{S}$ described in Sec. 3.1.2.

Algorithm 1 RANSAC Algorithm

```
$M \in \mathbb{M}^n$
$b \leftarrow \emptyset$, $b \in \mathbb{S}$
$b^* \leftarrow \emptyset$, $b^* \in \mathbb{S}$
$c \leftarrow 0$, $c \in \mathbb{R}$
$c^* \leftarrow 0$, $c^* \in \mathbb{R}$
for $i = 1 : K$ do
  $c \leftarrow 0$
  $M_r \leftarrow \alpha(M)$
  $s \leftarrow \beta(M_r)$
  for $j = 1 : n$ do
    $c \leftarrow c + \gamma(C_j, s)$
  end for
  if $c > c*$ then
    $c^* \leftarrow c$
    $b^* \leftarrow b$
  end if
end for
```

Given two sets of 3-D points $P_{k-1} = \{p_i \in \mathbb{R}^3 | i \in [1, N] \in \mathbb{N}\}$ and $P_k = \{p'_i \in \mathbb{R}^3 | i \in [1, N] \in \mathbb{N}\}$, and we arrange the points such that p_i is corresponded with p'_i. Assuming that the correspondence is correct, there exists a rigid transformation $T = [R|t]$ where R is the rotation matrix and t is the translation, such that

$$p'_i = Rp_i + t \tag{3.7}$$

Let $p = \frac{1}{N}\sum p_i$ and $p' = \frac{1}{N}\sum p'_i$ denote the centroids of P_{k-1} and P_k, we let $q_i = p_i - p$, $q'_i = p'_i - p'$ and $H = \sum_{i=0}^{N} q'_i q_i$. It is proven in [43] that the optimal estimated rotation $\hat{R}$ can be obtained by performing singular value decomposition (SVD) on H, i.e.

$$H = U\Lambda V^T \tag{3.8}$$

where $U, V \in \mathbb{R}^{3\times 3}$ are unitary matrices containing the left and right singular vectors of H, and Λ is a diagonal matrix containing the singular values. The optimal rotation is constructed via $\hat{R}$ via $\hat{R} = VU^T$. Finally, we derived the estimated translation vector $\hat{t}$ from $\hat{t} = p' - \hat{R}p$. Note that it is also possible to formuate the relative transformation solution as an iterative least square optimization problem [44], but this algorithm provides a non-iterative closed form solution, delivering better computation efficiency while minimizing the cost function of the least square problem.

3.1.4 Kinect RGB-D Camera

The Kinect camera, captured by Fig. 3.1 consists of a RGB camera, infra-red (IR) camera and a IR projector. To extract depth information, a IR pattern is projected onto the environment, and images from the IR camera are processed for correspondence of the constellation. Since the extrinsic characteristics between the projector and the IR camera are fixed and pre-calibrated, the depth to each point in the pattern can be computed through ray triangulation. Through hardware acceleration, the Kinect camera is capable of outputting 640×480 RGB-D images at 30 frame-per-second (fps), providing a dense 3-D point cloud with speed and quality for \$150. Despite all these benefits, the Kinect camera has a limited range of 8m. Due to the usage of IR imaging, the sensor is only operatinal

Figure 3.1: Kinect RGB-D Camera

in indoor environment. Finally, the resulting depth image contains has difficulty resolving depth of around the edges of an object, due to occlusion of the IR pattern from the IR camera viewing angle.

3.1.5 CUDA Toolchain

General-purpose computing on graphics processing units (GPGPU) refers to the incorporation of GPUs to perform computations other than graphic processing. The interest in GPGPU originates from the inherent advantage of GPU hardware architecture to perform parallel computations. To put it in perspective, a high-end Intel i7 CPU has 4 hyper-threaded cores at clock speed of 2GHz or more, while a consumer-grade NVIDIA graphics card has up to 1000 CUDA cores with 700MHz clock that can all run concurrently, leading to the efficiency of GPUs in processing large number of independent, simple calculations. GPGPU is particularly suitable for image processing applications, which typically involve performing the same computation on every image pixel. In [45], the author demonstrated up to 200x speed-up in processing speed for CUDA implementation of numerous image processing techniques over a single-core, sequential implementation.

CUDA is a set of proprietary tools and libraries that allows software developers to perform parallel computation using NVIDIA GPUs. Developers only need to write software for a single unit of computation - commonly referred to as a kernel - and specify how many kernels are required to launch. The GPU driver takes in this information and distributes the computation tasks to the GPU cores for executions, abstracting task scheduling from developers. An example is shown in Listing. 3.1 to demonstrate the advantage of parallel computation.

```
__global__ memCopy(float *A, float *B) {
    int i = blockId*blockDim.x + threadId.x;
    A[i] = B[i];
}
```

Listing 3.1: CUDA Code Example: Sample Kernel

The `memCopy` function is a sample CUDA kernel that copies the value of every element in a float array to a corresponding element of another array of the same size. Typically this is achieved by using a for-loop that sequentially writes to each array element, and the execution time is the sum of every write operation. In the above parallel implementation, the execution time is compressed to one of a single unit of computation as long as the number of elements is less than the number of GPU cores.

Overhead is introduced, however, in steps required to launch the kernel and retrieve computation results as illustrated in Listing. 3.2. Since the computer Random Access Memory (RAM) is physically separated from the GPU memory, memory transfer operations between RAM and GPU memory are necessary to download the RAM content to proceed with the `memCopy` in the GPU, as well as collecting the results from it. Such memory transfer operations are expensive, bottlenecked by communication bandwidth of the motherboard, and are to be avoided as much as possible.

```
int main(int argc, char** argv) {
    float* Acpu = (float*)malloc(N*sizeof(float));
    float* Bcpu = (float*)malloc(N*sizeof(float));
    float* Agpu = NULL;
    int threads = N;

    // allocate GPU global memory
    cudaMalloc(&Agpu, N*sizeof(float));
    cudaMalloc(&Bgpu, N*sizeof(float));

    // transfer the content of A array to GPU
    cudaMemcpy(AGpu, ACpu, N*sizeof(float),
        cudaMemcpyHostToDevice);
    // launch kernel
    memCopy<<<1, N>>>(Agpu, Bgpu);

    // retrieve results
    cudaMemcpy(Bcpu, Bgpu, N*sizeof(float),
        cudaMemcpyDeviceToHost);
}
```

Listing 3.2: CUDA Code Example: Kernel Launch

One method to mediate memory transfer overhead is stream processing. A stream in CUDA is a queue for sequential GPU tasks, including memory transfers and kernel launches. Kernel launches on different streams are executed in parallel, while memory transfer requests are sequentially serviced by the Direct Memory Access (DMA) peripheral built into the GPU. The DMA device performs asynchronous memory transfer without using clock cycles of the processors (GPU and CPU). Fig. 3.2(a) demonstrates the advantage of the stream processing technique over synchronous handling of GPU tasks. As shown in Fig. 3.2(b), DMA memory transfers allow the kernel execution and memory transfer to be overlapped in time, hiding the memory transfer overhead. The stream processing technique is extensively utilized in the extended KinectFusion algorithm as illustrated in Sec. 3.2.3.

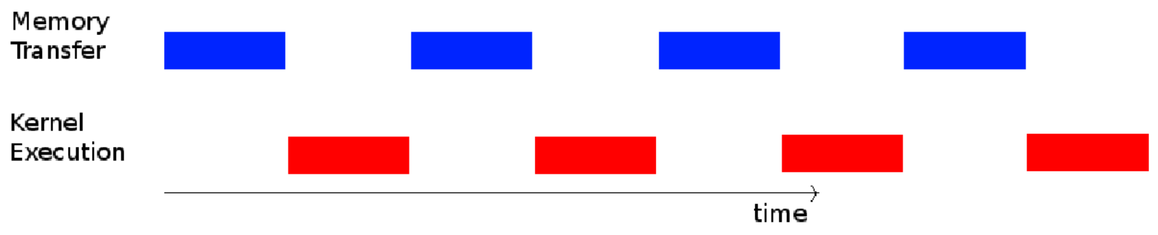
3.2 Spatially Extended KinectFusion Algorithm

This section describe the mathematical formulation and software implementation details of each of the five components of the spatially extended KinectFusion algorithm.

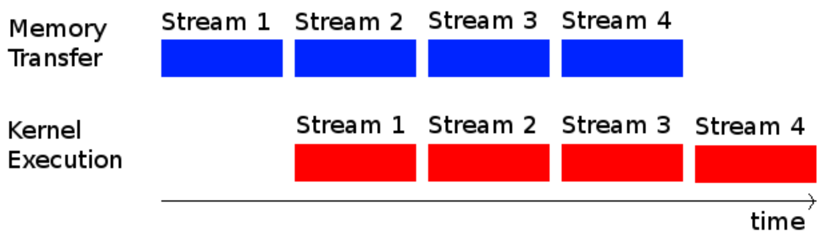
3.2.1 New Measurement Processing

The raw surface model is generated in the same manner described in KinectFusion. Upon receiving a new depth image, D_k, the value of the new camera frame vertex map $V_{c,k}$ at pixel u is given by the following equation,

$$V_{c,k}(u) = D_k(u)K\dot{u} \tag{3.9}$$



(a) Synchronous Processing



(b) Asynchronous Stream Processing

Figure 3.2: Timing diagrams for synchronous and asynchronous stream processing

The camera frame surface normal $N_{c,k}$ can be generated at each pixel $u \in \mathcal{U}$ with a cross product of two surface tangent vectors,

$$\begin{aligned} \delta V_i &= V_{c,k}(i+1,j) - V_{c,k}(i,j) \\ \delta V_j &= V_{c,k}(i,j+1) - V_{c,k}(i,j) \\ N_{c,k}(u) &= v(\delta V_i \times \delta V_j) \end{aligned} \tag{3.10}$$

where $v : \mathbb{R}^3 \to \mathbb{R}^3$ is the normalization operator, $v(p) = p/\|p\|_2$. The global frame vertex map, $V_{g,k}$, is computed by the transformation $V_{g,k} = T^g_{c,k}\dot{V}_{c,k}(u)$ applied to each vertex, and the global frame surface normal is derived with the rotation $N_{g,k} = R^g_{c,k}N_{c,k}(u)$. The computations of vertex and surface normals are trivially parallelizable, and are performed on the GPU.

3.2.2 Camera Pose Estimation

The camera pose estimation component of the algorithm aims to track the camera pose with respect to the global map. We first extract relative rigid transformation using visual odometry technique with on optical flow based correspondence of RGB-D features in subsequent image frames. In the second step of the camera estimation, we seed the KinectFusion iterative closest point (ICP) algorithm implemented in GPU with the visual odometry solution, and refine it further with the geometric shape information in the scene.

Given two consecutive images I_{k-1} and I_k, Shi-Tomasi features are extracted from I_{k-1} to generate feature set F_{k-1}, and track them in I_k using optical flow to generate a second set of features F_k. The feature correspondences are further filtered using RANSAC algorithm to reject outlying feature pairs. A final relative transformation solution between step k and $k-1$, denoted as T_v, is derived using the transformation model fitting method mentioned in Sec. 3.1.3 on the filtered set of 3-D points features pairs.

Fig. 3.3 demonstrates the robustness of the RANSAC visual odometry algorithm. The red features indicates outlying correspondence while green features are included in the relative camera motion solutions. As shown, the erroneous feature correspondences are successfully discarded to derive the clean relative transformation solutions between two consecutive camera pose.

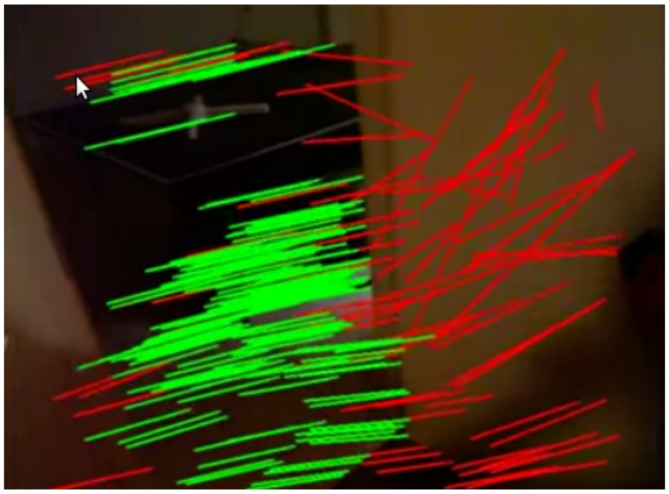


Figure 3.3: RANSAC based visual odometry

In KinectFusion, an ICP algorithm based on projective association is used to optimize the camera pose estimation further, and is introduced as follows. The objective of this step is to find the camera transformation matrix, $T_{c,k}^{g}$, that minimizes the function $J : SE(3) \to R^{+}$, the error sum of squares between the new global frame vertex map, $V_{g,k}$, and the predicted vertex map, $\hat{V}_{g,k}$, from the previous surface prediction step (details of this step is described in Sec. 3.2.5). $J(T^g_{c,k})$ is given by the following equation,

$$\begin{aligned} J(T^g_{c,k}) &= \sum_{\substack{u \in \mathcal{U} \\ \Omega(u)=1}} E(u)^2 && (3.11) \\ &= \sum_{\substack{u \in \mathcal{U} \\ \Omega(u)=1}} \|(T^g_{c,k}\dot{V}_{c,k}(u) - \hat{V}_{g,k}(\hat{u}))^T \hat{N}_{g,k}(\hat{u})\|_2^2 && (3.12) \end{aligned}$$

where $E(u)$ is denoted as the reprojection error, $\hat{N}_{g,k}$ is the predicted surface normal, $\hat{u} = \pi\left(Kw((\dot{T}^g_{c,k-1})^{-1})T^g_{c,k}\dot{V}_{c,k}(u)\right)$ is the predicted pixel location of the new vertex in the previous camera frame at step $k-1$. Furthermore, $\Omega(u) \in \{0, 1\}$ is a binary function that indicates if the vertices $V_{g,k}(u)$ and $\hat{V}_{g,k}(\hat{u})$ are corresponded. For $\Omega(u) \in \{0, 1\}$ to return 1, the depth image D_k must contain a valid value at pixel u; the distance between the vertices, $\|V_{g,k}(u) - \hat{V}_{g,k}(\hat{u})\|$, has to be smaller than a tunable threshold, $\varepsilon_d \in \mathbb{R}^+$, for rejecting outliers from the ICP optimization; finally, the angle between the surface normal vectors of both vertices, given by $acos(N_{g,k}(u)^T \hat{N}_{g,k}(\hat{u}))$, needs to be within another threshold, $\varepsilon_\theta \in \mathbb{R}^+$. Compiling these criteria, $\Omega(u)$ can be defined as the following,

$$\Omega(u) = 1 \iff \begin{cases} D_k(u) & \neq \emptyset \\ v(V_{g,k}(u) - \hat{V}_{g,k}(\hat{u})) & \leq \varepsilon_d \\ acos(N_{g,k}(u)^T \hat{N}_{g,k}(\hat{u})) & \leq \varepsilon_\theta \end{cases}$$

To find the minimum of J, an iterative optimization scheme is used. Let T^z denote the iterative solution of $T^g_{c,k}$ at ICP iteration z, where T^0 is initialized with $T_v\dot{T}^g_{c,k-1}$. Also let T^z_{inc} denote the linearized (with small angle approximation) incremental rigid transformation between iteration z and $z-1$,

$$T^z_{inc} = [R^z \mid t^z] = \begin{bmatrix} 1 & \alpha & -\gamma & t_x \\ -\alpha & 1 & \beta & t_y \\ \gamma & -\beta & 1 & t_z \end{bmatrix}$$

We can obtain the update step of each ICP iteration $T^z = \dot{T}^z_{inc}T^{z-1}$. Denoting $\tilde{V}_{g,k} = T^{z-1}\dot{V}_{c,k}(u) = [v_1, v_2, v_3]^T$, the reprojection error function $E(u)$ from (3.12) is reconstructed with T^z as follows,

$$\begin{aligned} E(T^z) &= [T^z \dot{V}_{c,k}(u) - \hat{V}_{g,k}(\hat{u})]^T \hat{N}_{g,k}(\hat{u}) \\ &= [T^z_{inc} \dot{\tilde{V}}_{g,k}(u) - \hat{V}_{g,k}(\hat{u})]^T \hat{N}_{g,k}(\hat{u}) \\ &= [G(u)x + \tilde{V}_{g,k}(u) - \hat{V}_{g,k}(\hat{u})]^T \hat{N}_{g,k}(\hat{u}) \end{aligned} \tag{3.13}$$

where

$$G(u) = \begin{bmatrix} 0 & -v_3 & v_2 & 1 & 0 & 0 \\ v_3 & 0 & -v_1 & 0 & 1 & 0 \\ -v_2 & v_1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$x = \begin{bmatrix} \beta & \gamma & \alpha & t_x & t_y & t_z \end{bmatrix} \in \mathbb{R}^6$$

The optimal solution x^* of the modified cost function, J, is then given by

$$x^* = \left(\sum A^T A\right)^{-1} \left(\sum A^T b\right)$$

where

$$A = \hat{N}_{g,k}(\hat{u})^T G(u)$$

$$b = \hat{N}_{g,k}(\hat{u})^T (\hat{V}_{g,k}(\hat{u}) - \tilde{V}_{g,k})$$

$A^T A$ and $A^T b$ are calculated in parallel for each corresponded pixel u within the GPU software. A GPU tree reduction algorithm [46] is implemented to perform accelerated summation across the entire image. The solution x is obtained by solving the linear system (3.14) on the CPU with SVD method described in Sec. 3.1.3.

The linear system is ill-conditioned when the geometric structure of the scene is predominately planar, resulting in erroneous ICP solutions. To address this issue, the condition number from the diagonal matrix from the SVD solution is checked. Let Λ denote the diagonal matrix from the SVD solution to the linear system. Λ has the following structure,

$$\Lambda = \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_6 \end{bmatrix}$$

and the condition number c is given by $c = \frac{\lambda_1}{\lambda_6}$. If c is larger than a certain threshold T_{svd}, the transformation solution from the visual odometry algorithm, given by $T_v T^g_{c,k-1}$, will be accepted without further ICP refinement as the ICP solution is likely to fail.

3.2.3 GPU Memory Management

The original KinectFusion operates within a single TSDF voxel grid stored in the GPU global memory, and the operation space of the algorithm is upper bounded by the limited memory size. A possible solution is to split the operation space voxel grid into smaller volumetric tiles.

Each TSDF tile s is assigned a descriptor, $d_s = [a_g, p_v, f]^T \in \mathbb{R}^3$, which contains the following information:

- a_g: GPU memory address
- p_v: volume center position
- f: file ID

Upon initialization, a kd-tree is constructed from a set of tile descriptors ($k = 2$ in our implementation). Global GPU memory is initially allocated for a tunable number of tiles that make up the active tile set, denoted C_0, and $n(C_0) > n(C_k), \forall k > 0$ to allow extra buffer for loading operations.

The pseudo-code of the memory swapping process is described in Algorithm 2, where `memcpyAsync`$(a, b, stream_s)$ is the command for asynchronous memory copy from source address, b, to destination address, a, executed by CUDA stream $stream_s$; `TSDFUpdate`$(a_s, stream_s)$ is the kernel launch command to invoke the `calcTSDF` function in Algorithm 3 on every voxel within tile s that resides in GPU memory address a_s, and the kernel launch request is again handled by the CUDA stream $stream_s$.

Line 2-7 of Algorithm 2 describes the unloading operations. Denote $R_l(T^g_{c,k}) \subset \mathbb{R}^3$ as the loading region fixed about the body frame origin of the camera pose $T^g_{c,k}$, and $R_u(T^g_{c,k}) \subset \mathbb{R}^3$ the unloading region. For the k$^{\text{th}}$ iteration, each tile $s \in C_{k-1}$ is checked to see if it is in $R_u(T^g_{c,k})$. If so, the memory content within s is transferred from the GPU memory to a corresponding file on the hard drive, and the index of this memory slot is inserted into the previous set of available memory addresses, A_{k-1}, resulting in intermediate set, A'_k. Also, s is removed from previous active set C_{k-1} to construct intermediate C'_k.

The loading operation is captured by Line 8-21 of Algorithm 2. After every unloading stream execution is completed, loading operations are commenced. A search is performed for all tiles with their volume centers located within a ball of radius upper bounded by the GPU memory size and the maximum sensor range, returning proximity set, B_k. A tile $s \in B_k$ that falls within loading region, $R_l(T^g_{c,k})$, is loaded into the GPU memory if there is a memory slot address left in A'_k. The newly loaded GPU memory slot index is removed from A'_k to construct A_k and assigned to the tile descriptor of s. Such a radius search for B_k in a kd-tree is efficient with complexity $\mathcal{O}(\log n)$.

Each memory loading operation is performed using the CUDA asynchronous stream processing API [47]. A stream is assigned to perform an asynchronous memory transfer to the GPU using the GPU direct memory transfer (DMA) unit, and the TSDF update kernel launch request is sent to the same stream to be processed after the loading is completed. The advantage of stream processing is that the execution of kernel launches and asynchronous transfers from different streams are concurrent. This allows multiple memory transfers followed by kernel launches (i.e. multiple tiles loaded into the GPU memory and updated right after) to be processed more efficiently.

Launching CUDA kernels via streams also enables concurrency of the kernel executions in different streams, a beneficial feature for TSDF updates in a multi-tile map. The original KinectFusion has one large volume in a continuous block of memory, allowing linear conversion from each CUDA kernel thread index to their corresponding voxel memory address. The multi-tile system instead has multiple blocks of continuous memory that are not necessarily continuous with each other. Therefore, one KinectFusion TSDF update kernel can only concurrently update voxels within an individual tile, while multiple kernels still need to be launched to cover all active tiles in C_k. Stream kernel launching allows further parallelization on a tile basis. The GPU execution is blocked until all concurrent TSDF update kernels are executed, ensuring that all tiles will be ready for surface prediction step.

A spatial hysteresis, $R_h(T^g_{c,k})$, is also introduced between $R_l(T^g_{c,k})$ and $R_u(T^g_{c,k})$, and defined as follows:

$$R_h(T^g_{c,k}) = \overline{R_l(T^g_{c,k}) \cup R_u(T^g_{c,k})}$$

The purpose of this feature is to prevent excessive loading and unloading requests as a tile moves back-and-forth across the region boundaries due to small changes in the camera pose estimate. A 2D example of the hysteresis is illustrated in Fig. 3.4. For the 3-D case, the loading region is defined as the camera frustum volume, and the hysteresis is a band of volume around the loading volume with a certain thickness. The selection for a hysteresis thickness is a trade-off between the amount of GPU memory overhead and the chance of triggering a memory transfer. Increasing the thickness allows more tiles in the hysteresis region to simply idle in the GPU memory, requiring more slots to be allocated to accommodate loading requests. Decreasing thickness leads to less tolerance for back-and-forth camera motion before transfer requests are triggered, which is taxing on the realtime performance of the algorithm. In our implementation, the thickness value is empirically tuned, and the investigation of more intelligent selection remains an area for future work.

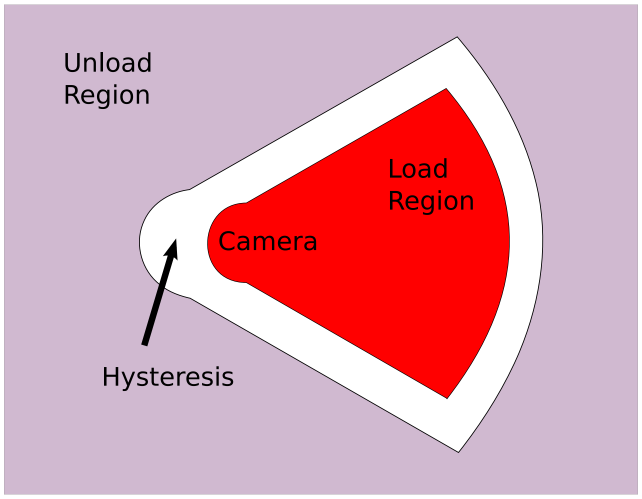


Figure 3.4: Hysteresis between loading and unloading regions

3.2.4 TSDF Update

KinectFusion aggregates the depth measurements into an online global surface representation using TSDF [33]. Signed distance function (SDF) [48] is a mathematical representation

Algorithm 2 `memSwap`$(A_{k-1}, C_{k-1}, T^g_{c,k})$

```
 1: $A'_k \leftarrow A_{k-1}$
 2: $C'_k \leftarrow C_{k-1}$
 3: **for all** $s \in C_{k-1} \cap R_u(T^g_{c,k})$ **do**
 4:     memcpyAsync$(d_{s,3}, d_{s,1}, stream_s)$
 5:     $A'_k \leftarrow A_{k-1} \cup d_{s,1}$
 6:     $C'_k \leftarrow C'_k \setminus s$
 7: **end for**
 8: **for all** $s \in B_k \cap R_l(T^g_{c,k})$ **do**
 9:     **if** $d_{s,1} \neq \emptyset$ **then**
10:         TSDFUpdateKernel$(d_{s,1}, stream_s)$
11:         continue
12:     **end if**
13:     $d_{s,1} \leftarrow a$ where $a \in A'_k$
14:     $A'_k \leftarrow A'_k \setminus a$
15:     **if** $d_{s,1} = \emptyset$ **then**
16:         continue
17:     **end if**
18:     $C'_k \leftarrow C'_k \cup s$
19:     memcpyAsync$(d_{s,1}, d_{s,3}, stream_s)$
20:     TSDFUpdateKernel$(d_{s,1}, stream_s)$
21: **end for**
22: $A_k \leftarrow A'_k$
23: $C_k \leftarrow C'_k$
```

of an set boundary, which in this case describes the environment surface. Any point in front of the surface is assigned a positive value, while any point behind the surface is negative. The zero crossing of the SDF is therefore where the surface lies.

KinectFusion uses a truncated, discrete approximation of the full SDF [49, 33]. A 3-D volume is discretized into voxel grids stored in the global memory of the GPU, and each voxel contains a 32-bit structure $S = [F, W]^T \in \mathbb{R}^2$, where F is the TSDF value, and W is the weight that describes the current confidence in this value. Given some maximum truncation distance μ, the new TSDF for a point $p_g \in \mathbb{R}^3$ in the global coordinate frame is calculated by projecting p_g onto some pixel, $u \in \mathcal{U}$, and calculating the difference between the depth of p_g relative to the camera and $D_k(u)$. The difference is truncated and normalized by μ to derive F. A point p_g with truncated TSDF value F ($F = 1.0$ or $F = -1.0$) are labeled as unseen if $F < 0$, and empty space if $F \geq 0$. The reason for truncation is that the TSDF value approximates the true SDF well only when the voxel is close to the isosurface. Furthermore, this truncation labeling strategy only assigns unique meaningful values to a small band of volume around the surface, potentially allowing room for effective compression.

The new TSDF value of each voxel is the weighted average between the new and the existing values. The weighted averaging over time offers robustness against noise in the resulting global surface model. A unit weight is chosen for new values for simplicity of implementation, but alternative weight selection schemes can be used. The weight value is truncated by parameter W_{max} not only to avoid variable register overflow, but also to allow map adaptation to dynamically changing environments. The algorithm is detailed in Algorithm 3. where the function `nearestPixel`(u) finds the nearest neighboring integer pixel coordinates to the projected pixel location, u; *Empty* is a flag defined to label empty space with a TSDF value of 1.0, and *Unseen* defined to label unseen space with a TSDF value of -1.0. Note that u can be outside of the camera frustum and depth value at u, $D_k(u)$, can be an invalid measurement, hence the need to check for validity of u as done in Line 3-5.

Since we only invoke Algorithm 3 for each point in the active set R_l, which has a fixed volume, the worst-case runtime is constant for the voxel grid update step. Each TSDF calculation can be handled independently, and so parallelization is again trivial in the GPU.

Algorithm 3 `calcTSDF`$(p_g, T^g_{c,k})$

```
1: $p_c \leftarrow T^g_{c,k} p_g$
2: u $\leftarrow$ `nearestPixel`$(\pi\,(K p_c))$
3: **if** $u \notin \mathcal{U}$ **or** $p_{c,3} \leq 0$ **or** $D_k(u) = \emptyset$ **then**
4:     **return**
5: **end if**
6: $d \leftarrow p_{c,3}$ - $D_k(u)$
7: **if** $d < 0$ **then**
8:     $tsdf \leftarrow \min(Empty,\, \frac{-d}{\mu})$
9: **else**
10:    $tsdf \leftarrow \max(Unseen,\, \frac{-d}{\mu})$
11: **end if**
12: $[F, W]^T \leftarrow S_{p_g}$
13: **if** $W = 0$ **then**
14:    $S_{p_g} \leftarrow [tsdf, 1]^T$
15: **else**
16:    $F \leftarrow \frac{tsdf + F \times W}{W+1}$
17:    $W \leftarrow \min(W_{max},\, W + 1)$
18:    $S_{p_g} \leftarrow [F, W]^T$
19: **end if**
```

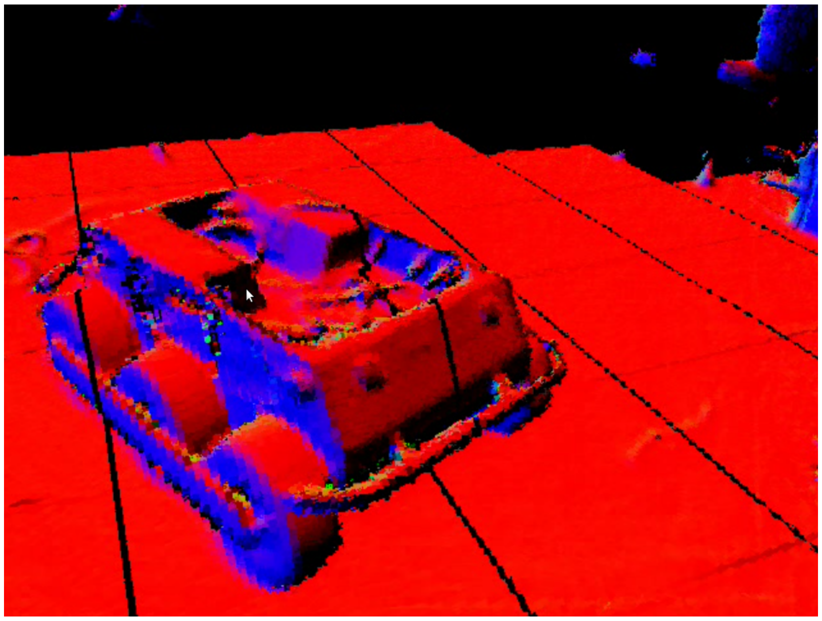
3.2.5 Surface Prediction

The predicted vertex map $\hat{V}_{g,k+1}$ and surface normal $\hat{N}_{g,k+1}$ need to be updated for ICP alignment in the next step of the main loop using raycasting through the updated TSDF voxel grid as in the original algorithm.

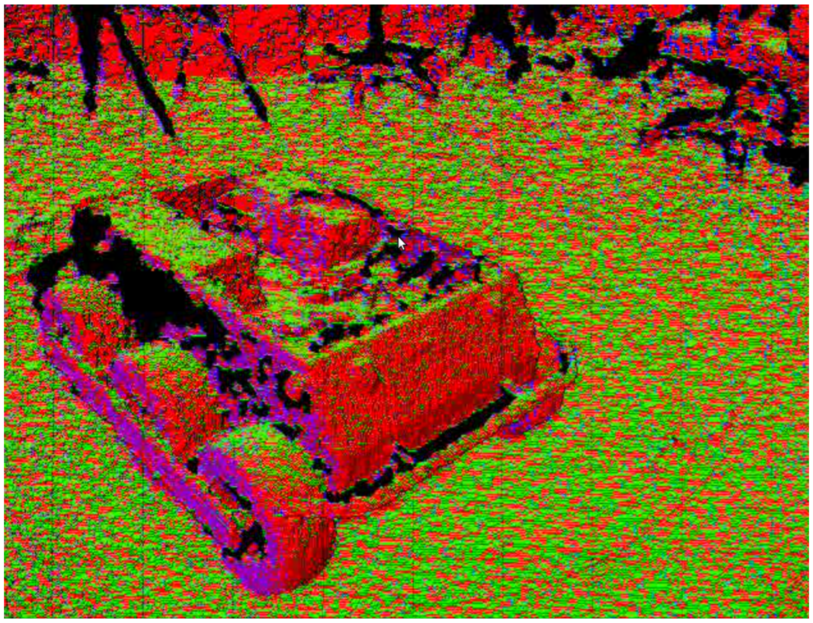
Given the current camera pose $T^g_{c,k}$, a ray $r(u) \in \mathbb{R}^3$ for each pixel $u \in \mathcal{U}$ is given by $r(u) = R^g_{c,k} K^{-1} \dot{u}$. To generate the vertex map $\hat{V}_{g,k+1}(u)$, we march along each $r(u)$ within the TSDF voxel grid until we find the first negative TSDF value. To adapt the original KinectFusion ray tracing algorithm to operate in multiple TSDF tiles, the global coordinates of $p_g \in \mathbb{R}^3$ visited by a ray are mapped to the index of the corresponding tile descriptor to find the memory slot to which p_g belongs. The index of the TSDF element corresponding to p_g within this tile is then computed to access the element content. Fig. 3.5 demonstrates the quality of surface reconstruction relative to the surface normal map created from raw Kinect depth data. Some spacing is inserted between map tiles to visualize the boundaries, which are visible as black borders in the image in Fig. 3.5(a). To more accurately estimate the intersection of the ray with the isosurface, trilinear interpolation is used. The global surface normal, $\hat{N}_{g,k+1}(u)$, corresponding to the vertex $\hat{V}_{g,k+1}(u)$, can be calculated by finite differencing the TSDF values of the neighboring voxels about the vertex point [33].

$$\hat{N}_{g,k+1}(u) = v\left(\left[\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z}\right]^T\right)$$

The TSDF representation provides efficiency to the raycasting process. Instead of performing binary ray marching where every voxel along the ray is visited, step size $\bigtriangledown t \leq \mu$ is taken until the first positive TSDF value less than 1 is encountered (such an encounter is guaranteed), at which point binary ray marching takes place. Significant speed-up is achieved using this ray skipping strategy.



(a) Reconstruction from Raycasting



(b) Raw surface normal

Figure 3.5: Raycasting vs Raw Surface Model

Chapter 4

Extended KinectFusion Experimental Results

To demonstrate the operation of the algorithm, we use a Kinect camera to map the Waterloo Autonomous Vehicle Laboratory (WAVELab) area - a 10×10×3m^3 space - with a centimeter voxel resolution. Each TSDF tile has a volume of 100×100×400 voxels and uses 35MB of memory; 1GB of global memory is allocated on the GPU for 29 TSDF tiles. Given a hard drive with a moderate size of 500GB, a 100×100×4m^3 area can be covered.

The current algorithm initiates a 2D square grid of TSDF tiles fixed on a certain height, assuming the operation space does not have much height variation (extension to a 3D TSDF tile grid is trivial). This allows the tile descriptors to be stored in a matrix, providing direct access of tiles about the camera position to find the proximity set, B_k, without the need for a kd-tree radius search. An alternative is to dynamically grow the tile kd-tree during runtime. New slices can be added in the directions required by robotic exploration, and the active and inactive sets are identified using an efficient radius search within the kd-tree. The algorithm is implemented using NVIDIA CUDA architecture, achieving real-time performance at 13Hz on a computer equipped with a Intel i7 2630 CPU and a NVIDIA GTX 590 GPU. The frame rate can be further improved by enforcing coalesced global memory access in the GPU software [47].

A video of this experiment is available at `http://tinyurl.com/7ehyhqw`. The captured Kinect depth data images are processed at 13Hz, and resulting video frames are encoded at 30fps. As demonstrated, tiles of the global map are being swapped in and out of

GPU memory for successful update and rendering with minimum impact on the tracking performance.

To extract the resulting map, each file of the corresponding tile is checked to find voxels with TSDF value $F \leq F_{max} \leq 1$, where F_{max} is a tunable threshold. The global vertex and surface normal of these voxels are extracted and stored in a point cloud. Fig. 4.1 shows a birds-eye view of the resulting surface model of the WAVELab. As demonstrated by the surface normal image presented in Fig. 4.1(b), the aggregated surface model extracted from TSDF tile grid is dense with well-defined surface normals, achieving detailed reconstruction of the environment. The resulting global map appears to show minimal warping without explicit loop closure. This indicates accurate camera pose tracking, as a result of performing ICP with respect to the stable global surface model rather than with the previous frame. These two desirable qualities from the original KinectFusion method are retained, while the operation space is expanded from 7m^3 [33] to 240m^3 despite maintaining 1cm voxel resolution.

To demonstrate the accuracy of localization of the algorithm, it is tested against a depth image dataset with indoor positioning ground truth from the Technical University Munich (TUM) computer vision group and numerically evaluated using their RGB-D SLAM benchmarking toolkit [50]. The Kinect RGB-D video sequence captures a camera moving in a half circle trajectory in X and Y while following a zig-zag motion in Z with a length of 17.044m. Furthermore, a person moves into the camera FOV during the video sequence, introducing dynamic motion into the scene. The video of this dataset can be found at `http://vision.in.tum.de/data/datasets/rgbd-dataset/download`, and the name of the dataset is $freiburg2\_desk\_with\_person$.

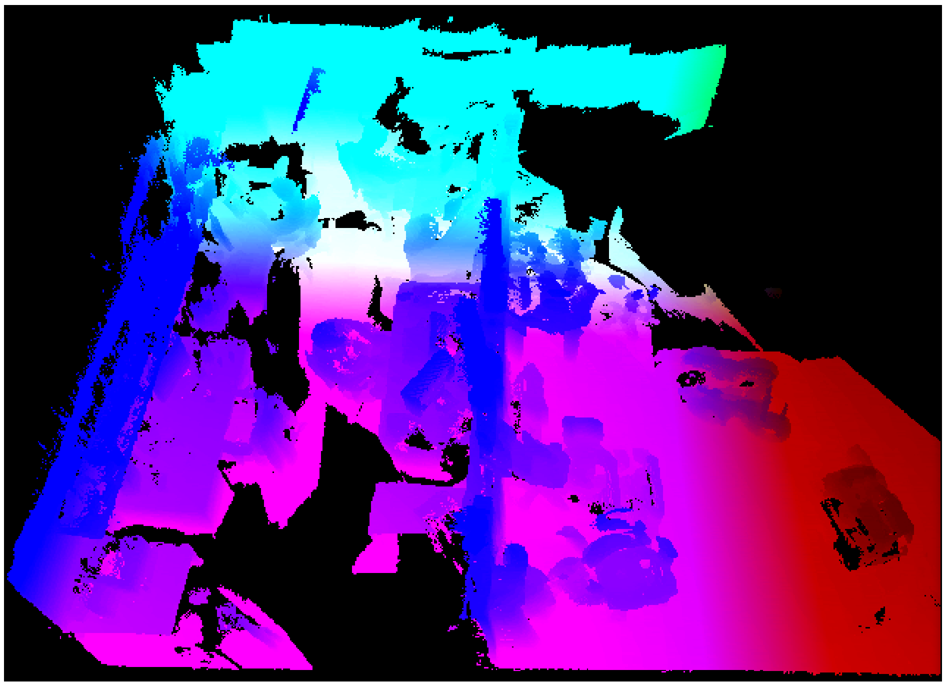
Fig. 4.2 captures the camera trajectory groundtruth as well as the motion estimation during the experiment. To calculate the Absolute Trajectory Error (ATE), the evaluation tool first associates each estimated pose with a groundtruth pose measurement using time stamps; it subsequently aligns the trajectory using Singular Value Decomposition (SVD) to find a relative transformation between the estimated and true trajectories that minimizes the sum of errors for all associated poses; once the trajectories are aligned with the resulting transformation, the ATE is calculated by taking the euclidean difference between each each corresponding point of the aligned trajectories. As shown, the algorithm is able to estimate the camera motion with a Root-Mean-Square (RMS) ATE of 9.88cm over the 17m long camera trajectory, without any explicit loop closure mechanism. It also demonstrates its

Dataset Name	Length	Avg. Angular Velocity	Avg. Trans. Velocity	Translation RMSE	Rotation RMSE
FR1 xyz	7.11m	8.920deg/s	0.24m/s	3.5068cm	3.247638°
FR1 rpy	1.66m	50.15deg/s	0.06m/s	3.3157cm	2.555076°
FR1 desk	9.26m	23.33deg/s	0.41m/s	6.4690cm	4.048444°
FR1 desk2	10.16m	24.86deg/s	0.43m/s	5.5079cm	4.066718°
FR2 deskwp	17.04m	21.32deg/s	0.32m/s	9.2322cm	3.745877°

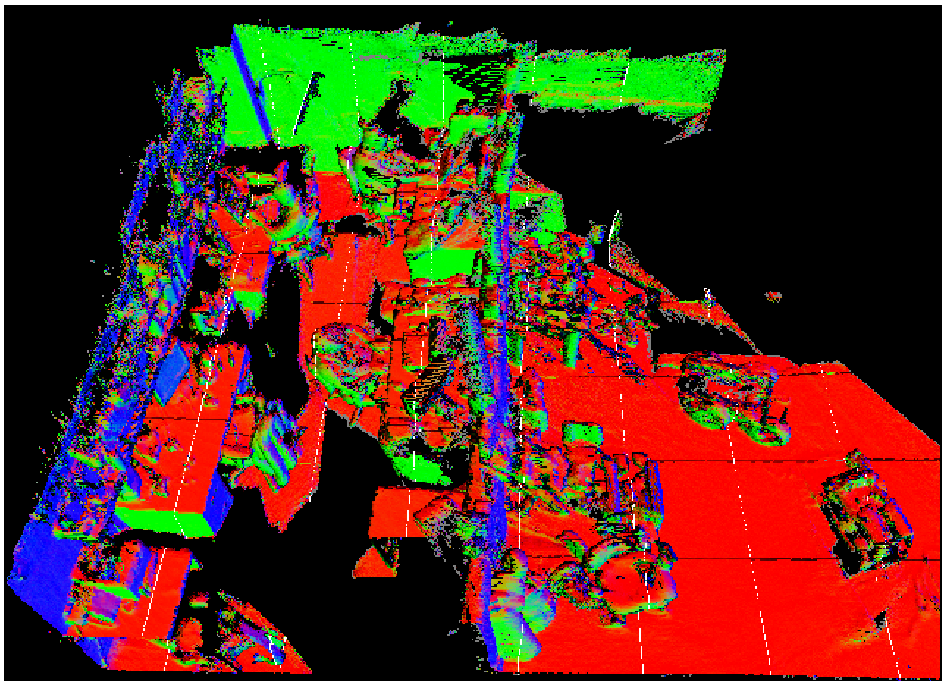
Table 4.1: Localization results against TUM RGB-D datasets

robustness against outliers introduced by the dynamic motion, effectively ignoring the time varying surface measurements from the person, while maintaining tracking of the camera pose. The algorithm is also tested against other TUM RGB-D datasets and the results are captured in Table. 4.1. Comparing against results described in [50], our algorithm is comparable with the state-of-the-art in terms of localization accuracy, while surpassing it in computation efficiency and reconstruction quality.

The weakness of the algorithm is exposed when a loop is formed in the camera trajectory and a previously mapped area is revisited. The position estimate with accumulated drift results in a surface prediction from an incorrect perspective when viewing the revisited area. The deviation between the surface prediction and the new measurements, when sufficiently large, leads to difficulties for the ICP pose optimization to converge to the correct solution, causing errorneous integration of depth information into the TSDF grid and degradation of the surface model. A vicious cycle is then triggered, in which a failure to estimate the camera location diminishes the map quality, and subsequently further decreases the localization accuracy. This indicates a need for a method to detect localization failure and a mechanism for relocalization.

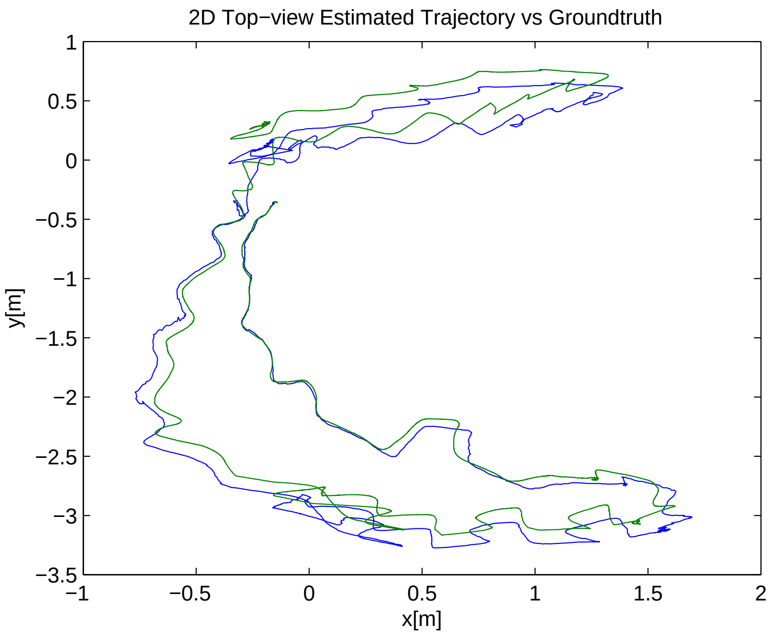


(a) Color mapped image of the map point cloud

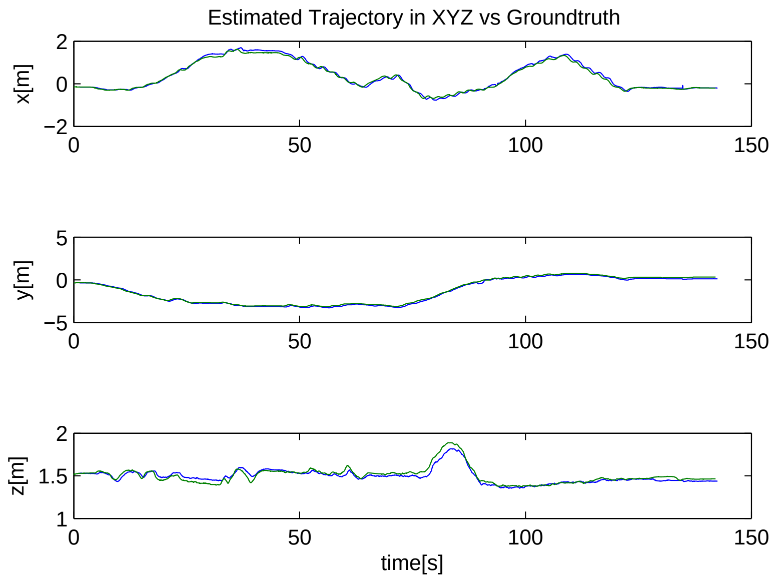


(b) Surface normal of the map point cloud

Figure 4.1: Surface reconstruction from raycasting



(a) 2D top view of the estimated trajectory (blue) vs the groundtruth (green)



(b) The estimated trajectory (blue) vs the groundtruth (green)

Figure 4.2: Estimated trajectory with 9.23cm RMS ATE

Chapter 5

Conclusion and Recommendations

Quadrotor UAVs have shown tremendous potential in applications such as unmanned exploration, surveillance, building inspection and disaster relief. Given sufficiently fast and accurate positioning measurements, quadrotors have demonstrated excellent mobility and controllability. To enable practical deployment of quadrotor platforms, it is essential to develop technologies that allow operations in outdoor settings or positioning denied regions. This work aims to develop methods to improve quadrotor UAV autonomy in these settings.

One of the main limitations of electrically powered micro UAVs is that they have limited payload, constraining the size of their onboard battery source and, therefore, their operating range and flight duration. This limitation motivates the search for a strategy to autonomously land the quadrotor on a ground station for battery recharge or swapping. The idea can be further expanded with the introduction of an autonomous mobile platform that introduces more mission flexibility by combining the advantages of both vehicles. UGVs have strong payload and power capacity, allowing them to equip heavy-duty, high-precision sensors and more powerful computation units. Micro UAVs, on the other hand, provides better mobility and sensor coverage, enabling access to regions with undrivable terrain.

A system is presented to coordinate an autonomous landing between a quadrotor UAV on a skid-steered UGV. A provably stable, joint-state decentralized controller is designed to drive the relative position error between the vehicles to zero, and a landing sequence is triggered when the relative error is below a certain threshold. The controller is validated in simulations to be stable, and using a standard sensor suite on commercially available UAV and UGV platforms, a successful landing in both indoor and outdoor settings was demonstrated.

The system can be further improved by modeling the aerodynamic effects such as wind disturbance and ground effects. During the landing coordination experiments, the combination of these unmodelled disturbances lead to difficulties in relative error convergence of the vehicles. It will be a very interesting improvement to the current scheme to introduce online estimation of wind and ground effect, and to actively cancel these disturbances within the controller.

This work also aims to enable further quadrotor autonomy in a positioning-denied environment. The KinectFusion algorithm [33] is implemented in house to perform dense surface reconstruction and simultaneous localization with a lowcost RGB-D camera. We further extended the algorithm by developing a memory swapping system that removes the volume constraint imposed by GPU memory upperbound, allowing the algorithm to operate in large environment without sacrificing surface reconstruction quality. A RANSAC visual odometry algorithm is integrated into the system that eliminates camera pose estimate divergence when viewing planar geometric structures in the scene. To demonstrate the effectiveness of the resulting system, we mapped a volume that far exceeds the KinectFusion upper volume limit, generating detailed and smooth surface model of the environment. The algorithm is also tested on published datasets with motion groundtruth, demonstrating comparable accuracy to the TUM RGB-D SLAM system without explicit use of loop closure, while surpassing it in terms of real-time performance and the quality of scene reconstruction.

A weakness of the algorithm lies in its inefficient map representation. While the 3D TSDF voxel grid allows a high degree of parallelization, it is expensive in its memory usage. An area for improvement could be an integration of a compression algorithm that further reduces the size of the voxel grid before its transfer to the hard-drive, expanding the mapping volume allowed by the storage amount of hard drives. One potential method for more efficient representation of the voxel grid is octree compression [51]. The entire mapping volume is assigned to the parent node of the Octree, and is divided into eight sub-regions, each further divisible up to the point that the desired volumetric resolution is obtained. In the TSDF map representation, only a thin crust of voxels within the truncation distance from the surface contain meaningful values, while the others contain truncated values (1 or -1) to represent empty space. Therefore, we can represent the large sub-regions of voxels with the same truncated value as a single node in the Octree, while keeping fine resolution in sub-region near the surface.

Another problem with the our dense reconstruction algorithm is the lack of a relocalization mechanism. When we revisit the same area after a loop is formed in the camera trajectory, the pose estimate can accumulate enough drifts that lead to erroneous surface prediction that the camera estimation algorithm cannot localize against. While the detection of pose estimation failure can be achieved by checking the resulting number of outliers from ICP, we have yet to find an effective method to recover our position after failures. A potential solution is proposed in the recent GPSlam system [52]. The authors introduce a simple state machine to handle pose estimation failure, relocalization and map morphing to maintain global map consistency in an occupancy grid. The algorithm starts in exploration mode, aggregating depth measurements into the occupancy grid in a similiar manner to the KinectFusion algorithm. When a failure is detected, the algorithm creates a new map for measurement aggregation, and operates in relocalization mode where the algorithm extracts visual features and matches them to previously recorded features to relocalize its pose. Once the camera position is recovered, the separate maps are merged into a global map based on the corrected poses. Due to the similarity between an occupancy grid and a TSDF grid, this mechanism is very transferable to the KinectFusion architecture.

In conclusion, this work presents two advancements in the autonomy of coordinated UGV and UAV teams. With the coordinated landing strategy, the UGV-UAV vehicle teams will soon be able to execute autonomous missions with long range and flight duration in outdoor settings. In positioning denied environments, we can now generate a detailed 3D reconstruction of the physical environments, providing important information for other aspects of robotic intelligence such as path planning and object recognition. The applications enabled by these technologies are quickly becoming viable and will very likely have a tremendous impact on humanity.

References

[1] Aeryon Labs, Inc. http://www.aeryon.com/(current Jul 2011). ix, 19

[2] Clearpath Robotics, Inc. http://www.clearpathrobotics.com/images/husky/husky_5_(700x400).jpg(Aug 2012). ix, 19

[3] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012. 1

[4] H. Huang, G. Hoffmann, S. Waslander, and C. Tomlin, "Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering," in *IEEE International Conference on Robotics and Automation.*, pp. 3277–3282, 2009. 1

[5] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 3, pp. 56–65, 2010. 1

[6] Q. Lindsey, D. Mellinger, and V. Kumar, "Construction of cubic structures with quadrotor teams," in *Proceedings of Robotics: Science and Systems*, (Los Angeles, CA, USA), June 2011. 1

[7] J. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *Control Systems Magazine, IEEE*, vol. 28, no. 2, pp. 51–64, 2008. 2

[8] S. Lange, N. Sünderhauf, and P. Protzel, "Autonomous landing for a multirotor uav using vision," in *SIMPAR 2008 Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots*, pp. 482–491, 2008. 3

[9] C. Sharp, O. Shakernia, and S. Sastry, “A vision system for landing an unmanned aerial vehicle,” in *Proceedings of IEEE International Conference on Robotics and Automation.*, vol. 2, pp. 1720–1727, 2001. 3

[10] S. Saripalli, J. Montgomery, and G. Sukhatme, “Visually guided landing of an unmanned aerial vehicle,” *IEEE Transactions on Robotics and Automation*, vol. 19, no. 3, pp. 371–380, 2003. 3

[11] S. Saripalli and G. Sukhatme, “Landing on a moving target using an autonomous helicopter,” in *Field and Service Robotics*, pp. 277–286, Springer, 2006. 3

[12] K. Wenzel, A. Masselli, and A. Zell, “Automatic take off, tracking and landing of a miniature uav on a moving carrier vehicle,” *Journal of intelligent & robotic systems*, vol. 61, no. 1, pp. 221–238, 2011. 3

[13] S. Oh, K. Pathak, S. Agrawal, H. Pota, and M. Garrett, “Autonomous helicopter landing on a moving platform using a tether,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, (Orlando, FL), pp. 3960–3965, 2005. 3

[14] H. Voos and H. Bou-Ammar, “Nonlinear tracking and landing controller for quadrotor aerial robots,” in *Proceedings of IEEE International Conference on Control Applications*, pp. 2136–2141, 2010. 3

[15] G. Whiffen and J. Sims, “Application of a novel optimal control algorithm to low-thrust trajectory optimization,” *AAS/AIAA Space Flight Mechanics Meeting*, 2001. 4

[16] J. Sims and S. Flanagan, “Preliminary design of low-thrust interplanetary missions,” *AAS/AIAA Astrodynamic Specialist Conference*, 1999. 4

[17] A. Richards, J. Bellingham, M. Tillerson, and J. How, “Coordination and control of multiple uavs,” in *Proceedings of AIAA guidance, navigation, and control conference, Monterey, CA*, 2002. 4

[18] B. Donald, P. Xavier, J. Canny, and J. Reif, “Kinodynamic motion planning,” *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993. 4

[19] J. Cortés, S. Martinez, and F. Bullo, "Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions," *Automatic Control, IEEE Transactions on*, vol. 51, no. 8, pp. 1289–1298, 2006. 4

[20] J. Lin, A. Morse, and B. Anderson, "The multi-agent rendezvous problem," in *Proceedings of Decision and Control.*, vol. 2, pp. 1508–1513, 2003. 4

[21] H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita, "Distributed memoryless point convergence algorithm for mobile robots with limited visibility," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 5, pp. 818–828, 1999. 4

[22] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. Munich, "The vSLAM algorithm for robust localization and mapping," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 24–29, 2005. 5

[23] J. Civera, A. Davison, and J. Montiel, "Dimensionless monocular SLAM," *Pattern Recognition and Image Analysis*, vol. 4478, pp. 412–419, 2007. 5

[24] A. Davison, I. Reid, N. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007. 5

[25] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski, "Bundle adjustment in the large," in *Proceedings of the 11th European conference on Computer vision: Part II*, (Berlin, Heidelberg), pp. 29–42, Springer-Verlag, 2010. 5

[26] S. Thrun and M. Montemerlo, "The GraphSLAM algorithm with applications to large-scale mapping of urban structures," *International Journal on Robotics Research*, vol. 25, no. 5/6, pp. 403–430, 2005. 5

[27] H. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms, MA Fischler and O. Firschein, eds*, pp. 61–62, 1987. 5

[28] J. Philip, "A non-iterative algorithm for determining all essential matrices corresponding to five point pairs," *The Photogrammetric Record*, vol. 15, no. 88, pp. 589–599, 1996. 5

[29] R. Newcombe, S. Lovegrove, and A. Davison, “DTAM: Dense tracking and mapping in real-time,” in *Proc. of the Intl. Conf. on Computer Vision, Barcelona, Spain*, vol. 1, 2011. 5

[30] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “RGB-D mapping: Using depth cameras for dense 3d modeling of indoor environments,” in *Proceedings of the 12th International Symposium on Experimental Robotics*, 2010. 5

[31] P. Besl and N. McKay, “A method for registration of 3-D shapes,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 14, no. 2, pp. 239–256, 1992. 6, 30

[32] D. Neumann, F. Lugauer, S. Bauer, J. Wasza, and J. Hornegger, “Real-time RGB-D mapping and 3-D modeling on the GPU using the random ball cover data structure,” in *Proceedings of IEEE International Conference on Computer Vision Workshops*, pp. 1161–1167, 2011. 6

[33] R. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon, “KinectFusion: Real-time dense surface mapping and tracking,” in *Proceedings of IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136, 2011. 6, 29, 46, 48, 50, 53, 58

[34] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. Leonard, “Kintinuous: Spatially extended KinectFusion,” in *Proceedings of RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Jul 2012. 6

[35] A. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, “Visual odometry and mapping for autonomous flight using an RGB-D camera,” in *Proceedings of Int. Symposium on Robotics Research*, (Flagstaff, Arizona, USA), 2011. 6

[36] M. Kaess, A. Ranganathan, and F. Dellaert, “iSAM: Incremental smoothing and mapping,” *IEEE Trans. on Robotics*, vol. 24, pp. 1365–1378, Dec. 2008. 6

[37] C. Holenstein, R. Zlot, and M. Bosse, “Watertight surface reconstruction of caves from 3D laser data,” in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3830–3837, IEEE, 2011. 7

[38] J. Daly, Y. Ma, and S. Waslander, "Coordinated landing of a quadrotor on a skid-steered ground vehicle in the presence of time delays," in *Proceedings of International Conference on Intelligent Robots and Systems*, pp. 4961 –4966, sept. 2011. 9, 16, 17

[39] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2247 – 2252, april 2005. 10, 15

[40] D. Pazderski, K. Kozłowski, and W. Dixon, "Tracking and regulation control of a skid steering vehicle," in *American Nuclear Society Tenth International Topical Meeting on Robotics and Remote Systems*, (Gainesville, Florida), pp. 369–376, 2004. 13, 14, 16

[41] K. Kozłowski and D. Pazderski, "Modeling and control of a 4-wheel skid-steering mobile robot," *Int. J. Appl. Math. Comput. Sci*, vol. 14, no. 4, pp. 477–496, 2004. 14

[42] M. Fischler and R. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981. 30, 33

[43] K. Arun, T. Huang, and S. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 5, pp. 698–700, 1987. 34, 36

[44] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991. 36

[45] Z. Yang, Y. Zhu, and Y. Pu, "Parallel image processing based on CUDA," vol. 3, pp. 198–201, 2008. 37

[46] M. Harris, S. Sengupta, and J. Owens, "Parallel prefix sum (scan) with CUDA," *GPU Gems*, vol. 3, no. 39, pp. 851–876, 2007. 43

[47] NVIDIA, "CUDA C Best Practices Guide," vol. 3, 2010. http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf. 45, 52

[48] S. Osher and R. Fedkiw, *Level set methods and dynamic implicit surfaces*, vol. 153. Springer Verlag, 2003. 46

[49] B. Curless and M. Levoy, “A volumetric method for building complex models from range images,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 303–312, ACM, 1996. 48

[50] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, “An evaluation of the RGB-D slam system,” in *Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1691 –1696, may 2012. 53, 54

[51] J. Zhang and C. Owen, “Octree-based animated geometry compression,” *Computers & Graphics*, vol. 31, no. 3, pp. 463–479, 2007. 58

[52] G. S. Katrin Pirker, Matthias Rther and H. Bischof, “GPSlam: Marrying sparse geometric and dense probabilistic visual mapping,” in *Proceedings of the British Machine Vision Conference*, pp. 115.1–115.12, BMVA Press, 2011. 59