January 2018

Autonomous Localization Of A Uav In A 3d Cad Model

Akkas Haque

AUTONOMOUS LOCALIZATION OF A UAV IN A 3D CAD MODEL

by

Akkas Uddin Haque

Bachelor of Science, Chittagong University of Engineering and Technology, 2013

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

In partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota

December
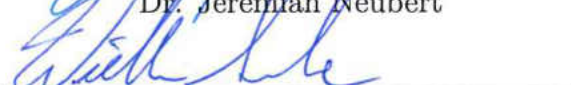
2018

This thesis, submitted by Akkas Uddin Haque in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

_____
Dr. Jeremiah Neubert

_____
Dr. William Semke

_____
Dr. Marcellin Zahui

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies at the University of North Dakota and is hereby approved

_____
Dr. Grant McGimpsey,
Dean of the Graduate School

_____November 28, 2018_____
Date

PERMISSION

Title   Autonomous Localization of a UAV in a 3D CAD Model
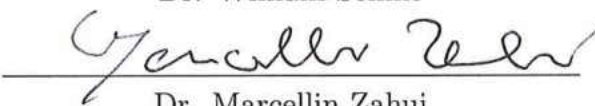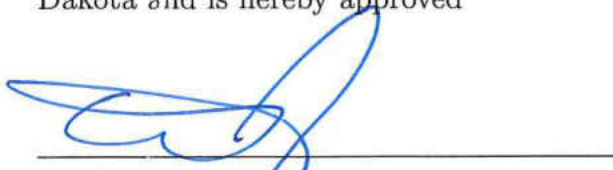
Department   Mechanical Engineering

Degree   Master of Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the chairperson of the department or the dean of the Graduate School. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Akkas Uddin Haque
November 7th 2018

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

To my wife Zeenat Nahar, who has been a constant source of encouragement, and to my parents for getting me to where I am today.

ABSTRACT

This thesis presents a novel method of indoor localization and autonomous navigation of Unmanned Aerial Vehicles(UAVs) within a building, given a prebuilt Computer Aided Design(CAD) model of the building. The proposed system is novel in that it leverages the support of machine learning and traditional computer vision techniques to provide a robust method of localizing and navigating a drone autonomously in indoor and GPS denied environments leveraging preexisting knowledge of the environment. The goal of this work is to devise a method to enable a UAV to deduce its current pose within a CAD model that is fast and accurate while also maintaining efficient use of resources. A 3-Dimensional CAD model of the building to be navigated through is provided as input to the system along with the required goal position. Initially, the UAV has no idea of its location within the building. The system, comprising a stereo camera system and an Inertial Measurement Unit(IMU) as its sensors, then generates a globally consistent map of its surroundings using a Simultaneous Localization and Mapping (SLAM) algorithm. In addition to the map, it also stores spatially correlated 3D features. These 3D features are then used to generate correspondences between the SLAM map and the 3D CAD model. The correspondences are then used to generate a transformation between the SLAM map and the 3D CAD model, thus effectively localizing the UAV in the 3D CAD model. Our method has been tested to successfully localize the UAV in the test building in an average of 15 seconds in the different scenarios tested contingent upon the abun-

dance of target features in the observed data. Due to the absence of a motion capture system, the results have been verified by the placement of tags on the ground at strategic known locations in the building and measuring the error in the projection of the current UAV location on the ground with the tag.

1. INTRODUCTION

There is an estimated cost of $15.8 billion per year due to inadequate interoperability in U.S. capital facilities due to not properly utilizing information technologies [1]. While many industries such as automobile, computer, and aircraft manufacturers have utilized an array of technologies such as automation technology and electronic standards to replace paper documents, the construction industry has not used such technologies as effectively to integrate design, construction, and operational processes, thereby increasing missed opportunities and costs. During the planning phase of a construction project, the construction industry does typically produce a building information model (BIM) from 3D computer aided design (CAD) models of the intended project [2], however current techniques for tracking the progress of the construction project typically are performed using manual measurements that are both time and labor intensive [3].

Many in the construction industry are beginning to use small Unmanned Aerial Vehicles (UAV) to aid in inspections of the as-built structure including both the interior and exterior of the structure [4, 5, 6, 7, 8]. Currently methods using UAVs require a pilot to operate the UAVs to collect pictures and video. Inspection engineers then manually view the pictures and video to detect possible inspections or maintenance problems during the construction process. Using the data collected from the UAVs, methods to create as-built documentation as well as compare to the original design

1

could be introduced to increase the interoperability found in the construction industry.

Collection of data for as-built documentation using UAVs flying autonomously to a goal location would greatly reduce costs. One important part of autonomous flight is the determination of the current location of with respect to the surroundings. This work aims to reduce costs by creating a tool to aid in the autonomous flight of a UAV in the CAD model. More specifically, the work aims to develop a novel system to allow a small UAV to automatically localize itself in a building given only the 3D CAD model, and the images collected via video from an onboard camera. To accomplish this task, the proposed system utilized an open-source Simultaneous Localization and Mapping (SLAM) algorithm, and an open-source Deep Convolutional Neural Network (CNN) to identify key features found in both the 3D CAD model, as well as the fully constructed building (windows, and doors), which are then aligned to perform the registration. A novel and fast orientation invariant descriptor has been developed specifically for this purpose and its efficiency and accuracy measured on a portable system.

To the best of our knowledge, our system is the first to autonomously localize itself within a given 3D CAD model in the context of indoor flight of a UAV.

1.1 Contributions

A novel pipeline is proposed to autonomously localize a UAV given only a CAD model as input. The system would only require the user to provide the CAD model of the building, in which the UAV is to be localized, in the BIM(Building Information

Model) format. The BIM format stores information regarding features like doors and windows which are then used to localize the UAV. A novel 3D feature descriptor is also proposed and developed, that is orientation invariant, and is fast and efficient to compute. The process of computing and matching the descriptors is shown in section 5.1. The performance of the system is evaluated by implementing on a UAV and having it autonomously localize itself in the CAD model. Using the CAD model and the SLAM map as inputs to a path-planning algorithm, waypoints are generated that are then used by the UAV to navigate the goal position within the CAD model. Finally, leveraging a SLAM algorithm that supports relocalization, we ensure the fast relocalization of a UAV in the 3D CAD model once the generated SLAM map has been registered to the 3D CAD model.

1.2 Constraints and Assumptions

The following are the constraints that have been imposed on our use case:

1. **No access to GPS or other active sources of localization :** Since our system was developed for the use of localization within a building, and because GPS systems are unreliable indoors, they were ruled out as a source of localization. This is due to the interference caused by the signals bouncing off walls in addition to the having poor coverage indoors.

2. **No access to Time of Flight, Received Signal Strength Indicator(RSS) and other forms of device based localization:** These systems require the

setup of complex systems in the building and would need to be carefully calibrated to enable efficient localization.

3. **Availability of CAD model with Building Information :** Since we are trying to solve the problem of localizing a UAV given a CAD model, we assume that the UAV will be flown within the building of which we have the CAD model available. We also assume that the said CAD model is a Building Information Model, containing information regarding the features we are interested in. This limitation could be relaxed if we make assumptions about the general size and shape of the features we are interested in and try to deduce the features from information gleaned from the overall structure of the CAD model. For example, windows could be extracted based on the information that most of the windows are only present along the exterior of the building and not the interiors. Similarly doors could be defined as being the access points to different rooms, etc.

4. **Planarity assumption regarding the overall structure of the building:** Since most buildings have walls that are usually planar, the assumption that large numbers of map points that fit certain criteria for being considered parts of planes is used to further refine the SLAM to CAD correspondence. This in turn leads to better localization as the SLAM map-points become more tightly coupled with the CAD model. This assumption could be further extended to include other shapes as well, by estimating the local plane normals using the surrounding map-points. That would however place a restriction on the SLAM

that could be used for the application in that the estimation of local plane normals would need a more densely populated set of map points.

5. **Hardware Constraints :** Since the system will need to be run on a UAV that provides live data to a portable workstation, the only mode of communication would be through a wireless link between the two devices and all the required hardware would have to be on the two devices. The UAV will also need to be able to travel the length of the building and should thus be able to conserve enough power to run for the duration of the flight. This is why, it was decided early on, that the only sensor inputs to the UAV would be a stereo camera and an IMU. The UAV could carry a low powered computer on board to follow waypoints set by the workstation .

2. BACKGROUND

This chapter is devoted to the evaluation of the available algorithms and systems used in the development of our localization system. The first section evaluates the current state of art in indoor localization technologies and their shortcomings. Our system is dependent upon two main components that are crucial to its working: A fast and accurate SLAM algorithm and a Convolutional Neural Network that is capable of running in real-time. The second section assess a few of the state of art SLAM systems that have been considered for our implementation. The SLAM system will be used by the UAV to build a map of its surroundings. The constructed map will also be used by the UAV to perform basic navigation. The third section is devoted to the study of a few of the currently available Convolutional Neural Networks for object detection. This will be used to detect the features necessary for localization.

2.1 Current Research

Indoor localization has been a very hard problem owing to the lack of reliable GPS sensor information in indoor environments. Many different approaches have been made to overcome this problem. A few companies like SnapTrack [9], Atmel and U-blox have developed technologies that enable indoor localization based on specialized devices [10]. However, these technologies tend to rely on wireless sensor

networks, and are thus costly and limited by the fact that a specialized building-wide setup has to be constructed in order for these systems to work [10].

Time of Flight and Received Signal Strength Indicator(RSS) based methods attempt to perform localization by using the known locations of devices. Time of Flight measures the distance between the transmitter and receiver by measuring the time taken for the signal to reach the target. The time of flight value multiplied by the speed of light gives the distance between the transmitter and receiver. The main drawback of this method is that it requires at least three separate transmitters to be able to localize in 3D space as explained in [11]. It would also require the knowledge of the absolute locations of the transmitters. and requires strict synchronization between the reference nodes.

A novel method of localization within a known 3D structure has been explored in [12]. The system extracts lines from the environment using an omni-directional camera and attempts to generate hypothesis for locations by proposing a robust matching algorithm to match the 2D lines from images to 3D line segments in the CAD model. Even though the system brings in a number of innovative features edge segment matching and prior visibility analysis, it requires the robot to be moving only in $SE(2)$ and moving in $SE(3)$ would exponentially increase the search space for the initial localization of the robot. It also requires the availability of a bulky omni directional camera, which is not feasible in our case.

A more recent paper [13] proposes a method of autonomous exploration in an unknown indoor environment. The paper proposes a mode of exploration where it utilizes the semi-dense nature of the map obtained from LSD-SLAM [14] to generate

an octomap [15] that it then uses to locate spaces that the UAV can navigate through. The UAV follows a mode of exploration dubbed as "star discovery" that it uses to populate unknown regions of the space around it.

Our system introduces a novel method of solving the indoor localization problem where we extract features present in images observed using a stereo camera. The features are extracted from the images using a fast and efficient Convolutional Neural Network, which ensures that the detections are accurate and quick to compute. These features are spatially registered to a SLAM map that is built in parallel to the extraction of features. Simultaneous Localization and Mapping(SLAM) is the process of creating a map of the surroundings while also localizing within it. This map is usually made up of feature points that have defining characteristics like corners, lines or intensity gradients. The creation of a SLAM map ensures that the UAV is localized within the generated map. Since we use a stereo camera rig as our sensor, and a SLAM system that uses stereo images, the SLAM map generated has accurate scale. The registration of the extracted features within the SLAM map ensures that the relative distances and orientations of the features are consistent with the relative distances and orientations of features in the CAD model. This information is then used to form correspondences between the two sets of features which are then used to estimate a rigid transformation between the two sets of data. This transformation effectively registers the SALM map with the CAD model, which in turn localizes the UAV within the CAD model.

To the best of our knowledge, our system is the first to autonomously localize itself within a given 3D CAD model in the context of indoor flight of a UAV.

2.2 Simultaneous Localization and Mapping(SLAM)

Simultaneous Localization and Mapping(SLAM) is the process of creating a map of an unknown environment based on sensor inputs while simultaneously tracking the location of the agent within the constructed map [16, 17]. This had traditionally been thought to give rise to a causality dilemma, owing to the fact that both the localization and the mapping components need the other to work. Mapping without localization had been explored in [18, 19] while localization without mapping had been explored in [20, 21]. All of these methods however assumed that either the map or the location was already available. The breakthrough came with the realization that a joint estimation of the location and the map made up of landmarks could be co-estimated if formulated into a single problem if the correlations between the landmarks could somehow be integrated into the solution [16]. A more in-depth analysis of the history and evolution of SLAM algorithms can be found in [16, 22].

For our work, we require a SLAM system that generates a globally consistent and accurate map, can scale well enough to encompass the whole of the interior of a building, and is able to run comfortably on a portable computing platform. The SLAM should also be able to estimate the scale of the environment to a reasonable degree. In the following sections, we evaluate two state-of-art SLAM algorithms and their applicability in our work.

2.2.1 ORBSLAM

ORBSLAM [23] is a graph-based pure visual SLAM that aims to generate an accurate map and 6DOF pose. ORBSLAM uses ORB descriptors (Oriented FAST and Rotated BRIEF) [24]. What this means is that it uses FAST [25] to detect corners in different pyramid levels and then computes the orientation of the corners by computing the intensity centroid [26]. These corners are then used to compute the "rotated BRIEF" descriptor. The BRIEF descriptor [27] is "steered" in the direction of the orientation of the FAST corners. Utilizing FAST as the corner detector and BRIEF as the descriptor, both of which are extremely fast to compute, ORB features result in a very efficient feature matching alternative to the the more expensive and slower SIFT [28] or SURF [29] features.

The system runs three parallel threads that are each responsible for tracking, local-mapping and loop-closing. The tracking thread localizes the camera with every frame by finding feature matches to the local map and minimizing the reprojection error by applying motion-only Bundle Adjustment. Bundle Adjustment is an optimization procedure that is used mainly to obtain consistency in a map by minimizing reprojection errors. The local mapping thread manages the local map and optimizes it performing full bundle adjustment. The loop closing thread detects large loops and corrects that can later drift by performing pose graph optimization.

The scalability, global consistency accurate map and localization provided by OBBSLAM makes it a very good candidate for our work. However, since it is a monocular based system, the scale would have to be estimated from other sources.

2.2.2 Large Scale Direct Monocular SLAM (LSD-SLAM)

Large Scale Direct Monocular SLAM (LSD-SLAM) [14] is a direct or featureless SLAM algorithm designed for monocular cameras. This direct method circumvents the use of keypoints, but instead uses image intensities to estimate the map and camera location. This algorithm provides a semi-dense map by creating a depth map, an inverse depth map and variance of the inverse depth map in the neighborhood of large image intensity gradients from particular key-frames.

LSD-SLAM also seperates the problem into three parts including tracking, depth map estimation, and map optimization. In the tracking section, the current camera pose is estimated from a image with respect to the current keyframe pose by minimizing a variance-normalized photometric error. The depth map estimation section is used to determine if a new image will be selected as a new keyframe, or else refine the current keyframe. Finally map optimization is performed by minimizing an error function based on photometric residual and depth residual which are scaled with the variance of both images. LSD-SLAM provides a more detailed map due to the use of more of the image information and produces a semi-dense map that could potentially provide useful information during for obstacle avoidance and path-planning. This, however comes at a higher computational cost. LSD-SLAM, like ORB-SLAM is also a monocular system and thus requires the estimation of scale from other sources.
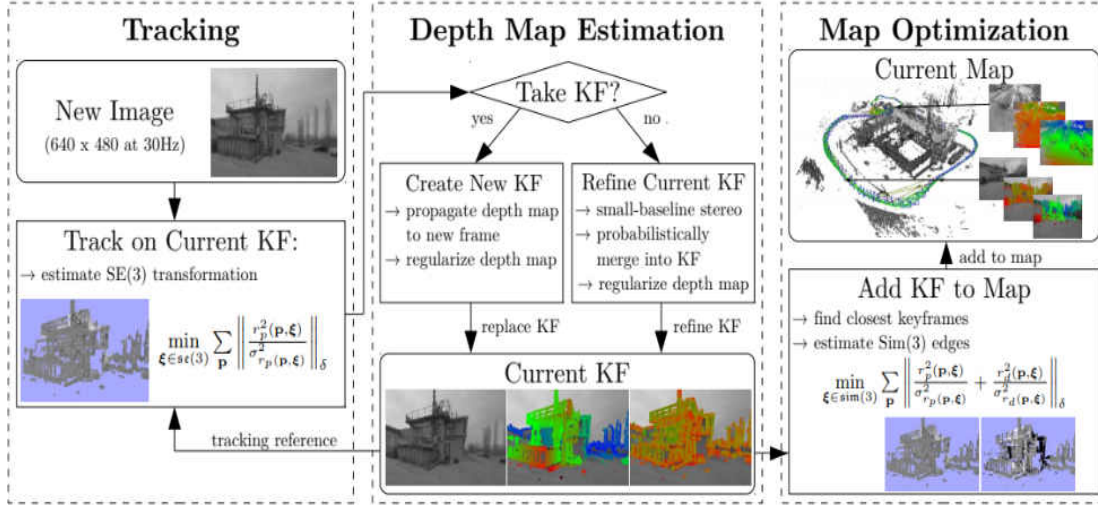
Figure 2.1. LSD process flow. Three parallel threads run to create a SLAM map from direct observations. The tracking thread tracks camera pose based on the current camera keyframe, denoted as KF in the image above. Keyframes in LSD-SLAM are views in the camera view frame that are used to estimate the depth of the surrounding by the computation of the photometric error with the subsequent images. The depth map estimation thread refines the depth from multiple views w.r.t the current keyframe. The map optimization thread optimizes the map by finding the closest keyframes and adding keyframes to the global map. The image is taken from Jakob Engel et al. [14].

2.2.3 ORBSLAM2

ORB-SLAM2 [30], built from the previous ORB-SLAM, was the first open source SLAM system for monocular, stereo and RGB-D cameras. Akin to the original monocular implementation, the system comprises of three main parallel threads: tracking, local-mapping and loop-closing. The contribution of ORB-SLAM2 over ORB-SLAM

was the inclusion of stereo keypoints which provide depth information from just one frame. The stereo observations also enable the weighting of keypoints based on the distances and thus result in a more accurate map than direct methods.
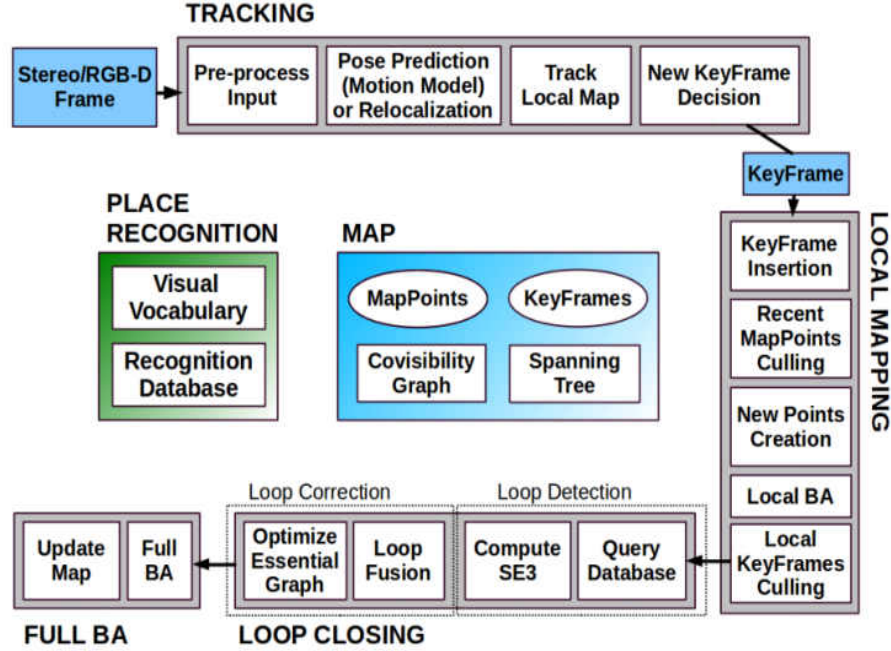
Figure 2.2. ORB-SLAM2 process flow. Three parallel threads are run to produces a globally consistent map. The first one localizes the camera with every frame by matching features with a local map while minimizing a reprojection error. A second thread maintains the local map of 3D features and optimizes it. The last searches for potential loop closures and performs local and global bundle adjustment. The image is taken from Raul Mur-Artal et al. [30]

In addition to these features, it also provides a lightweight mode to localize against a previously constructed map. In this mode, the local-mapping and loop-closing

13

threads are deactivated and the camera is continuously localized by the tracking thread, using relocalization if needed.

Even though it provides a fast and efficient method to estimate the location of the camera sensor and its surroundings, it does lose valuable information as it only tracks feature points. However, this is easily overcome by the computation of 3D points when required as shown in section 4.2.3.

ORB-SLAM2 provides a scalable, globally consistent, accurate and efficient mapping and localizing mechanism and thus meets all our requirements from the SLAM system. Additionally, the relocalization feature provided by this system is a very useful feature that enables the reuse of a previously registered SLAM map to localize within the CAD model without going through the costly process of re-registering or regenerating the SLAM map. Due to these reasons, ORB-SLAM2 was selected as the SLAM algorithm of choice for our work.

2.3   Convolutional Neural Network

Even though Convolutional Neural Networks have been around for almost 20 years, they have only recently gained popularity due to the availability of low-cost and powerful graphics processors that enable the the networks to train in a highly parallel framework. CNNs were first implemented by Yann LeCun in his work involving the classification of different types of numbers with high precision [31]. Interest in the use of CNNs for solving object detection problems was revived by the introduction of AlexNet [9] in 2012 as a contribution in the ImageNet [32] classification competition.

CNNs derive their name from the use of sets of filters that are convolved with the image. The filters are trained to activate specific features in the image. In addition to the convolutional layers, the extracted features are also passed through max-pooling layers which down-sample the image. These layers condense the complex information from the features and enable the drawing of assumptions from different image regions at various scales. The combination of convolutional and max-pooling layers allow for the definition of the abstract form of objects. The network produces a feature map as the output, which provides information about the type and location of features in the image.

For use in our work, we require a CNN that is fast, lightweight, and has the potential to run realtime on a low powered GPU. It should also output bounding boxes predicting the locations of the objects in the image. In the following sections, we evaluate three different state of art networks and discuss their applicability in our work.

2.3.1 Region-based Convolutional Neural Networks (R-CNN)

R-CNN [33] is a region-based proposal network that extracts approximately 2000 region proposals from an input image, which are then fed into a Convolutional Neural Network that extracts features for each of the proposals. Each of the proposed regions are then classified using linear Support Vector Machines(SVM's) that identifies the class the regions belong to. Figure 2.3 shows the different stages in the classification process. R-CNN performed better detections than other algorithms at the time it was introduced. Due to the large size of the network however, the network takes about

20 seconds to detect objects on an image using a desktop GPU [33]. Since we need a system that is real-time, this would not meet our requirements.
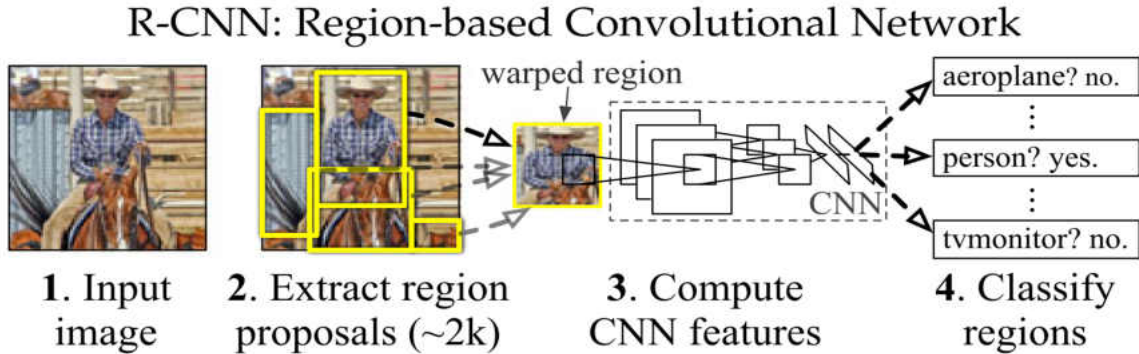
Figure 2.3. R-CNN structure taken from Ross Girshick et al. [33]

Fast R-CNN

Even though R-CNN is able to classify objects well, it is slow in running the detection process as it runs a full forward pass for the 2000 region proposals. Fast R-CNN [34] proposes to increase the processing efficiency by using Spatial Pyramid Pooling Networks(SPPNets). The feature map is computed once for the entire image and is used in classifying the region proposals directly. This speeds up the process of testing images as the whole image is fed into the initial convolutional layer, on which the region specific operations are done as opposed to extracting regions and then feeding the warped region proposals into a network. This speeds up computation in both training and test time. The paper claims a speedup of about 10 times as

compared to the slow R-CNN. This however is still not fast enough to for use in our system.

Faster R-CNN

Fast R-CNN [35] improved the performance of R-CNN at the convolutional network but neglected the slow creation of region proposals. Even though the CNNs were GPU accelerated, the region proposals were still being generated on the CPU. Faster R-CNN solved this problem with the introduction of Region Proposal Networks (RPNs) which were able to reuse the convolutional layers present in the existing CNN. This in turn led to a significant reduction in time taken for the creation of region proposals. The paper claims that it is able to process about 7 fps on a Titan X GPU.

Since our goal is to make make the system available on a portable system, it is still too restrictive.

2.3.2   YOLO

The first iteration of YOLO approached object detection as a regression problem to spatially separated bounding boxes and associated class probabilities [36]. This resulted in an improvement in speed over previous methods as a single neural network predicts both the bounding boxes and class probabilities in one evaluation. As the complete pipeline is a single unified network, end-to-end optimizations directly improve detection performance. Unlike sliding window and region proposal based net-

works, YOLO sees the entire image during training and test time, so it can predict both the classification and the localization in a single pass through the network.

Region based methods on the other hand try to classify the same region several thousand times. YOLO is structured so that it looks at every part of the image only once which is where it got the name. Each image is first split into a 7x7 grid of cells. Each of these cells predict two bounding boxes which have their centers at the centers of the respective cells. Next, the confidence values for each of these bounding boxes are predicted representing the degree of certainty with which the network is able to correctly predict the coverage of an object. This results in a total of 7x7x2=98 boxes with corresponding confidence values. Each of these 98 boxes are then classified into the objects to be detected. This is done by running the classification for every cell. Since all the bounding boxes are centered on the cells, the bounding boxes for the object can be taken to be the same as the bounding boxes predicted for the cells. The output of the network is the set of all bounding boxes and their associated class probabilities. The correct boxes are filtered out by applying a threshold on the class probabilities. The process is shown in figure 2.4.

The paper claims that the network runs at 45 fps using a TitanX GPU. In addition, they discuss a simpler version of the network, tiny-yolo which can be run on smaller GPUs with reduced accuracy.

2.3.3   SSD - Single Shot Multibox Detection

Similar to YOLO, SSD [37] produces a fixed number of detections, the highest ones of which are selected as the valid detections. The initial layers are based on the
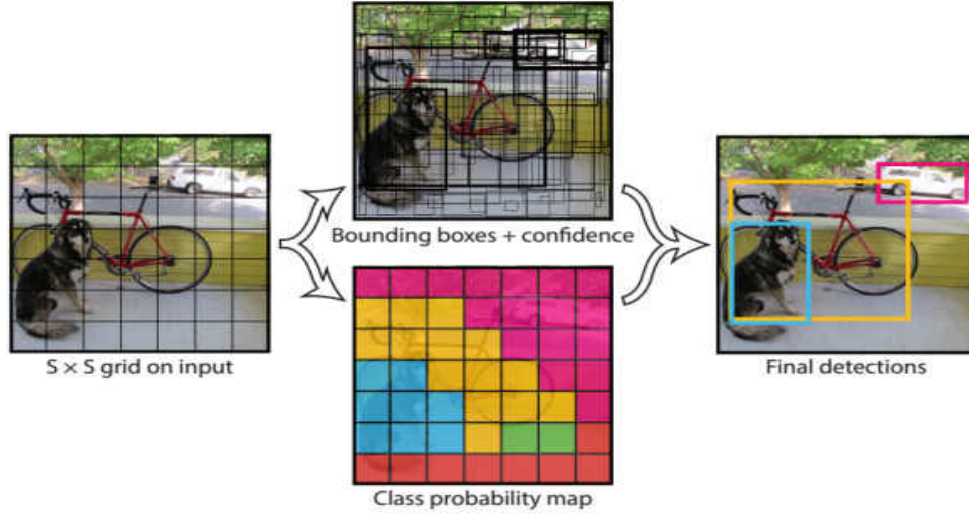
Figure 2.4.   The image is divided into a 7x7 grid from which bounding boxes and class probabilities are predicted. The final output generated after thresholding is shown to the right. Image taken from Joseph Redmon et al. [36]

VGG16 [38] network architecture. A set of convolutional feature layers are added at the end that extract features at different scales. Unlike YOLO, feature maps produced at each layer is used to predict detections and are also used as the input to the next layer.

SSD improves over YOLO by proposing a network that is fully convolutional and is able to detect at different scales. The large number of connections needed in fully connected layers makes them computationally expensive. The use of low cost convolutional layers in SSD enables it to generate a greater number of detections per
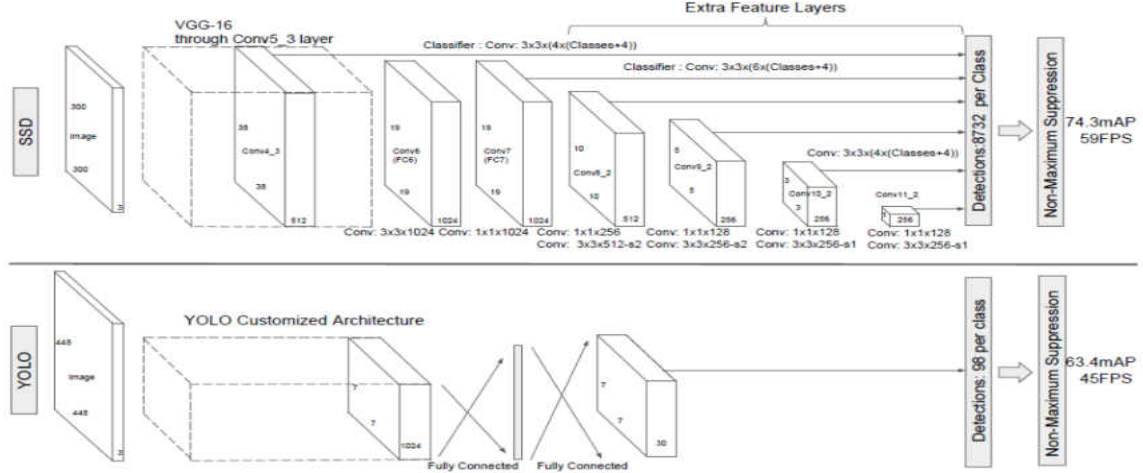
Figure 2.5. Comparison between YOLO and SSD. The upper portion shows the SSD network architecture. The image is passed into a set of convolution layers taken from the VGG16 network which produces 38x38 cells that contain feature vectors of length 512. Bounding boxes are then predicted using each of these cells for objects that fit in the cell approximately. This process is repeated for subsequent convolutional layers that are have smaller number of cells. The initial layers are used to detect smaller objects while the deeper layers combine these features to detect larger objects. The YOLO architecture is shown the lower portion of the image. The fully connected layers impact performance and the absence of scaling give rise to more localization errors [37].

class in each image as compared to YOLO. Figure 2.5 shows that SSD is able output 8732 predictions as opposed to YOLO which is able to output 98 predictions per class. This enables SSD to detect objects of varying sizes better than YOLO.

2.3.4 YOLOv2

YOLOv2 [39] introduced a number of improvements over YOLO that each contributed to an increase in performance of the detector as shown in Figure 2.6. One of the most important improvements is that the network was converted to a fully convolutional network which increased the speed of the network. The 224x224 pretrained classifier used in YOLO was replaced by a classifier of size 448x448 to enable higher resolution in the detections. As shown in Figure 2.6 a number of other features, like batch normalization, use of dimension priors, location prediction, multi-scale detection lead to incremental improvements in the overall accuracy of the detector.

	YOLO								YOLOv2
batch norm		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier			✓	✓	✓	✓	✓	✓	✓
convolutional				✓	✓	✓	✓	✓	✓
anchor boxes				✓	✓				
new network					✓	✓	✓	✓	✓
dimension priors						✓	✓	✓	✓
location prediction						✓	✓	✓	✓
passthrough							✓	✓	✓
multi-scale								✓	✓
hi-res detector									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Figure 2.6. Improvements of YOLOv2 over YOLO. The addition each new feature leads to incremental improvement in the performance of the CNN. Converting to a fully convolutional network did not increase accuracy, but increased the speed of the algorithm

The result of the optimizations and changes to the architecture is a network that is able to run at 67 frames per second on 416x416 size images while maintaining

an mean average precision(mAP) of 76.8 on the VOC 2007 and VOC 2012 training datasets. This corresponds to 45 frames per second on the laptop that was used for the application. Since the network was run wholly on the GPU, the workstation was free to use the CPU for running other parts of the system, like the localizer and the SLAM module.

3. SYSTEM OVERVIEW

Five major steps are involved in enabling the goal of localizing the UAV in a CAD model. The first is a pre-processing step, in which the features in the CAD model are extracted and feature descriptors are computed. During the live operation of the UAV, the system creates a map of the surroundings of the UAV while also simultaneously localizing within this map. This is accomplished using a preexisting SLAM algorithm. Thirdly, known features are extracted from the surroundings using an existing real-time CNN for object detection. These features are spatially registered to the SLAM map. Fourthly, feature descriptors are computed from the extracted features. Finally, feature descriptors from the two sets of data, i.e. the CAD model and the observed data are matched and a rigid transformation is computed using the matched features. Since the UAV is already localized in the SLAM map, the computation of the transformation effectively localizes the UAV in the CAD model.

The hardware system is comprised of two main components: a SLAMdunk module mounted on top of a Parrot Bebop 2 and the ground control station. The SLAMdunk is an integrated kit that uses the NVIDIA Jetson TK1 as the processing unit and has integrated sensors including a stereo camera rig, an IMU, a front-facing ultrasound rig, an inertial measurement unit(IMU), a barometer and a magnetometer. The setup is shown in Figure 3.1.

Figure 3.1. The system setup - A S.L.A.M.dunk mounted on a Parrot Bebop 2 drone

The NVIDIA Tegra K1 has an onboard GPU which has the Kepler architecture with 192 CUDA cores. It also has an ARM cortex A15 CPU and a 2GB x16 Memory with 64-bit width [40]. The SLAMdunk controls the Parrot Bebop 2 by connecting to an access point hosted by the bebop 2. The complete network configuration is given in figure 3.2. The SLAMdunk interacts with the ground control station by connecting via the wifi connection between the Parrot Bebop 2 and the ground control station.

All communication between the SLAMdunk and the ground control is routed via the Parrot Bebop 2 server. This is done to leverage the powerful wifi hardware already available on board the Bebop 2.
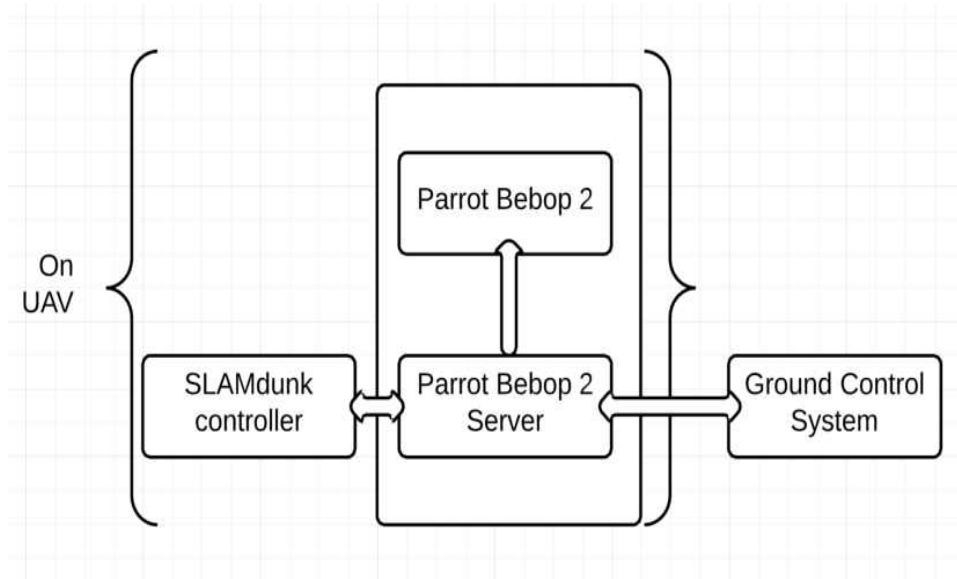
Figure 3.2. Communication setup between Ground Control, SLAMdunk, and Parrot Bebop 2. The Bebop 2 is connected to the ground control station via wifi. It hosts an access point that both the SLAMdunk and the ground control are connected to. The SLAMdunk is connected to the Bebop 2 server via USB. It also uses the USB to send control commands to the Bebop 2.

There are two main modes of operation of the UAV, **exploration** and **localized** mode. During the exploration mode, the UAV operation is based completely on the SLAM map, while the localizer continuously tries to localize the UAV in the SLAM map that is generated on the fly. Once the UAV is localized, the UAV switches to the Localized mode, where it follows a set of way-points to a goal position. The process flow shown in figure 3.3 is the one used in the exploration mode. Before either of these modes are enabled however, there needs to be an extra preprocessing phase done during which the data to be used by the system is prepared.
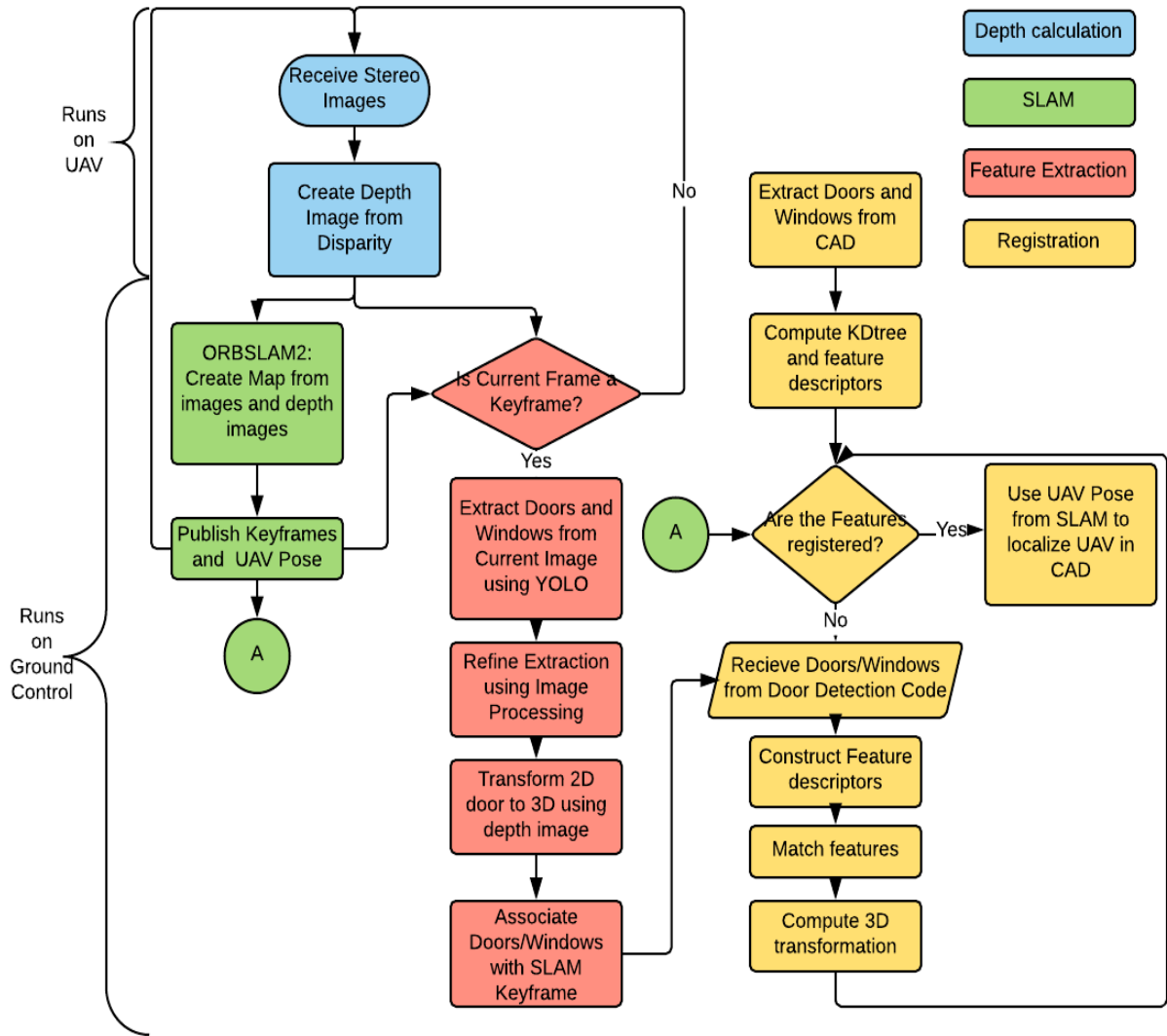
Figure 3.3. Process flow

3.1 Preprocessing

During the preprocessing phase, the 3D CAD model of the building is first loaded onto the program on the ground control system using the Open Asset Import Library(Assimp) [41]. Assimp is a platform independent library used to import different 3D model formats and provides a unified method of accessing them. Being written in C++, it was easily integrated into the pipeline. Since the CAD model is in the modern Building Information Model(BIM) format, it contains information about the features that we later use for the localization: doors and windows(from here on referred to as features).

The features from the CAD model are extracted and the positions of the centroids are stored in a K-Dimensional Tree(k-d tree) for easy retrieval of the nearest neighbors of any given feature. A k-d tree is a data structure that enables easy storage and retrieval of points in a k-dimensional space [42]. The points are stored in the form of a binary tree with every node being a k-dimensional point. A hyperplane that is perpendicular to the axis at a given level is used to split the children along that dimension. This is done in practice by first selecting a root node. Using the first dimension as the axis(for example, the x-axis), the child nodes are split into two groups: nodes having the coordinate value of the first dimension smaller than the root node, go into the left tree and the nodes having greater values go into the right subtree. In the next level, the second dimension is used to divide the nodes and so on. Once all the dimensions are expended, the first dimension/axis is used to divide the data. This process continues in a cycle until all the nodes have been represented. As the elements are represented in the form of a tree structure, retrieving the closest
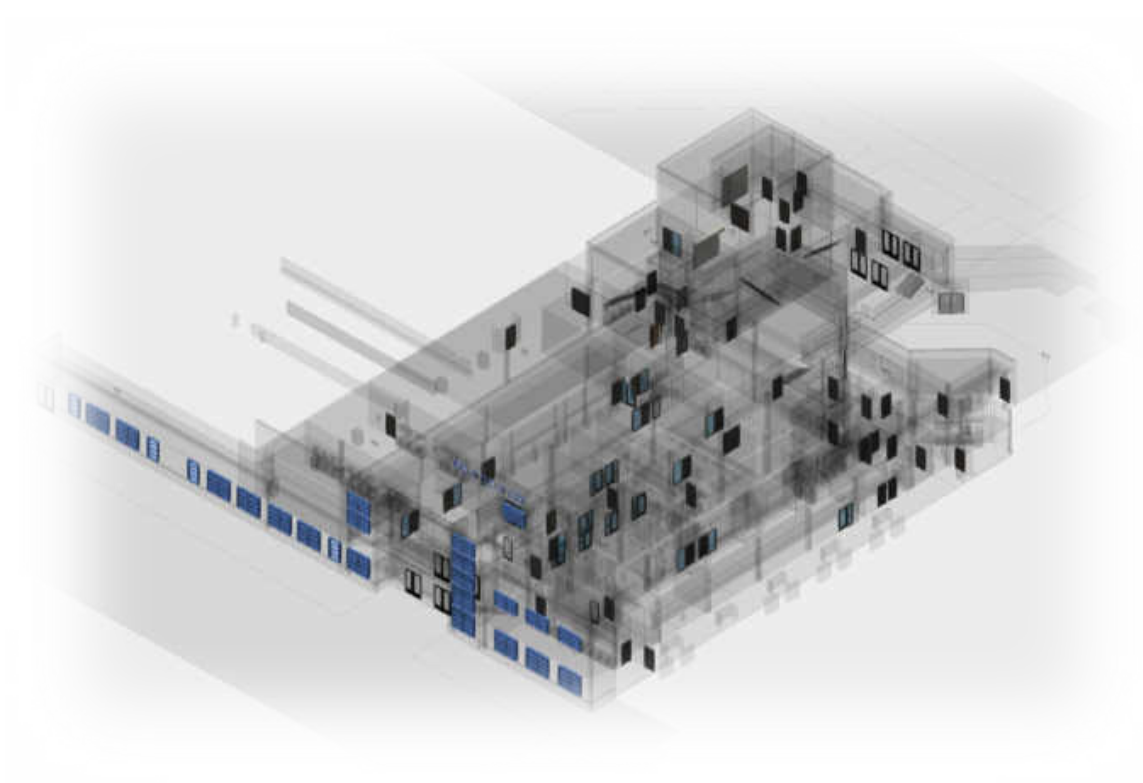
Figure 3.4. The building information model: The required features(doors and windows) are extracted and shown. Doors are depicted in black, windows are shown in blue. All the other components like walls etc are shown in translucent gray.

N nodes is fast and efficient. For our implementation, the dimension of the k-d tree used is 3, to model the 3D structure of the data represented by it. Using a k-d tree speeds up the retrieval of nearest neighbors which will be used later in the pipeline during the coarse registration phase. The construction of a 2D k-d tree is depicted in figure 3.5.
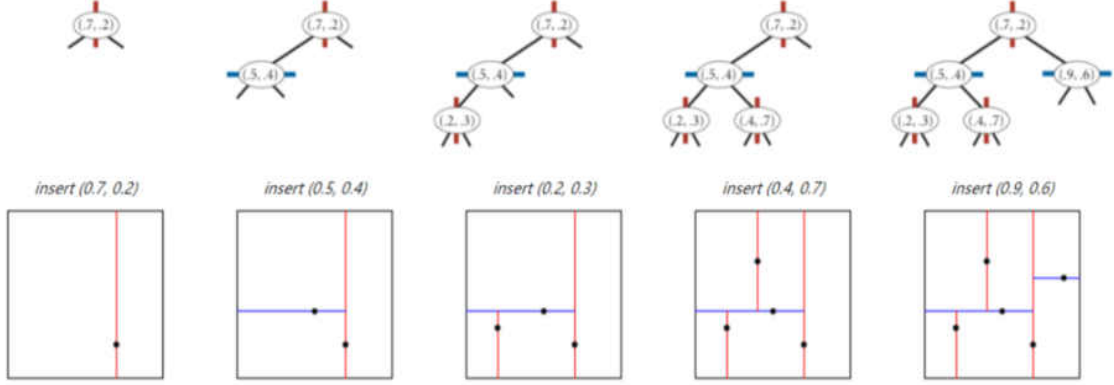
Figure 3.5. Typical K-D tree construction. The algorithm for the construction of the tree is described in section 3.1. The image is taken from [43].

Using the k-d tree, descriptors are computed for each of the features based on the algorithm outlined in algorithm 2. The descriptors are later used to find correspondences between the detected features and the CAD features.

3.2 Exploration mode

The left and right images from the stereo cameras are used to compute depth from disparity using GPU optimized Semi Global Block Matching(SGBM) [44] on-board the SLAMdunk. An explanation of the algorithm can be found in appendix A.1 The left image and the computed depth images are sent via wifi to the ground control station at a rate of 30fps. IMU updates are also sent over to the ground control at

150Hz. All communication between the UAV and the ground control including the images are handled via Robot Operating System(ROS) communication packets.

ROS is a robust messaging system that provides a framework and a set of tools to build robotics applications [45]. It provides a method to enable robust form of communication between the UAV and ground control and takes care of missed or dropped packets along during the transmission. ROS also has a set of tools to visualize point cloud data, lines, objects, and robot pose. It also has plugins that can be used to test path planning and mapping algorithms.

On the ground control, the system divides into two main components : the SLAM module and the localization module. The SLAM module estimates the current UAV pose in a map generated using the images and IMU data. ORBSLAM2 [30] is used as the SLAM algorithm and is used in the RGBD mode, leveraging the left and computed depth images from the UAV. It constructs a map consisting of a sparse set of 3D map points while also estimating the 6DOF current UAV pose in it.

The localizer extracts features(doors and windows) from the images and tries to localize the UAV in the CAD model by first forming correspondences between the detected features and features extracted from the CAD model, which are then used to estimate a rigid transformation in SE(3) between the two coordinate frames(SLAM and CAD model) using a robust transformation estimation algorithm explained in 5.2.

In the localizer, the left images are first passed through a previously trained Convolutional Neural Network(CNN), YOLOv2 [39] which is trained to detect doors and windows. YOLOv2 is a real-time object detection system that is able to simultane-

(a)
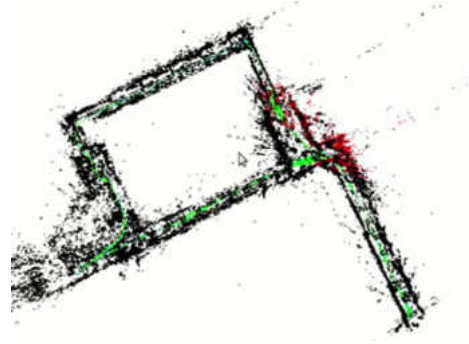
(b)
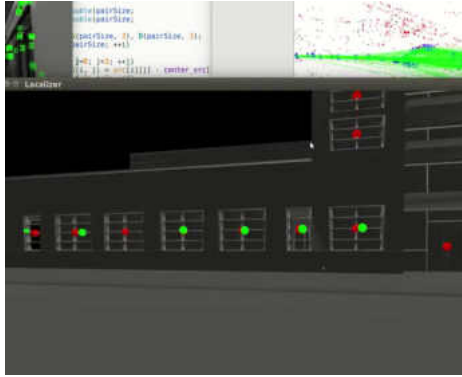
(c)
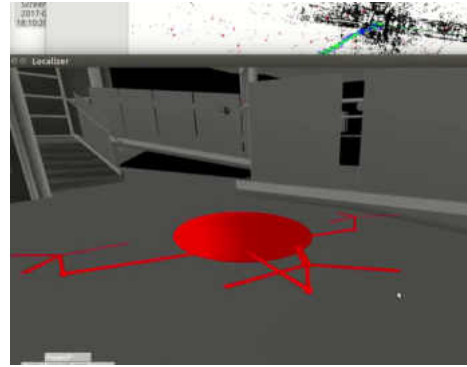
(d)

Figure 3.6. Coordinate Transformation and Registration.The CAD co-ordinate system is shown in (a). The SLAM coordinate system is shown in (b).(c)shows the correspondences found between the SLAM and CAD coordinate systems. The UAV is shown localized in the CAD model in (d).

ously classify and localize objects in images using a single network. YOLOv2 outputs bounding boxes of predictions in the images passed to it. Image processing techniques are applied to the area within the bounding boxes to extract the boundaries of features. These are reprojected into the 3D space by using information from the current camera pose.

These features are then registered to the keyframes in which they were first detected in the SLAM map. This registration enables the accurate positioning of the features in the SLAM map by leveraging the support of bundle adjustment during local and global loop closures.

The detected features are then used to register the SLAM map with the CAD model as described in chapter 5.

3.3   Localized mode

Once the UAV is localized using the localizer module, the UAV begins the process of traveling to the destination. The height of the UAV from the ground level is retrieved using an ultrasound sensor located beneath the Bebop 2 drone. A cross section of the current floor, roughly at the height of flight of the UAV is generated from an Octomap [15] representation of the building. The octomap provides a probabilistic 3D voxel representation of the environment/building that is updated realtime with objects that do not appear in the original CAD model as well. The goal position and the current location of the UAV are then projected onto the 2D occupancy grid that is generated from the cross section generated earlier. An A* search algorithm [46] to plan a path from the current location to the goal position is then run on this

grid map. This generates a path of shortest route between the current location and the goal location. Dividing the floor into a 2D grid reduces the search space and consequently provides an efficient method to compute the shortest distance. Using the path generated, waypoints are sent to the UAV representing the centroids of the cells in the 2D grid which are connected by the computed path. The A* algorithm is illustrated in figure 3.7. Since the UAV does not undergo fast motions, it is sufficient to supply the waypoints as a set of location coordinates with respect to the CAD coordinate system.
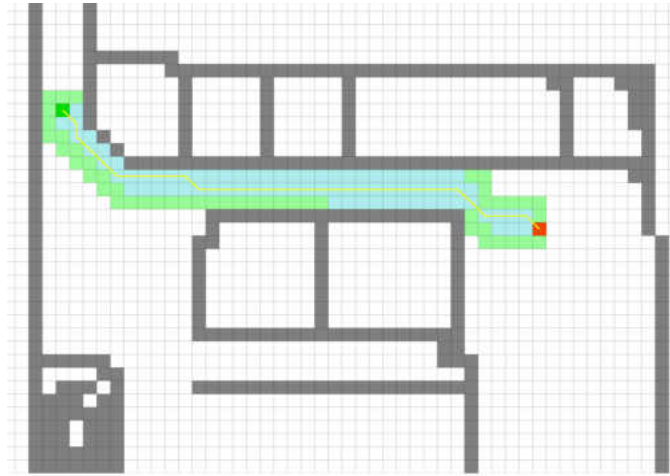
Figure 3.7. A* Algorithm for path finding. A cross section of the floor is used to generate the 2D occupancy grid The dark green cell is the start location and the red cell is the goal location. The gray cells are occupied and the white cells are free. A path is computed by applying the A* algorithm and waypoints are generated using the centroids of the grid cells.

4. FEATURE DETECTION AND EXTRACTION

To register the SLAM map with the 3D CAD model, features have to be detected and one to one correspondences need to be established between detected features and the features extracted from the CAD model. To perform this correspondence, we first create feature descriptors using the features in both the datasets. The descriptors are created so that they are rotation invariant. The process is discussed in greater detail in section 5.1. In this section we will be discussing the methods employed to detect the features from the observed data.

Traditionally, object detection algorithms have focused on extraction of features like HAAR classifiers, HOG, SIFT, SURF etc. which were then passed through a learning algorithm like Support Vector Machines, Random Forests, etc. Deep learning algorithms bypass the feature extraction method completely. We select YOLOv2 for its speed and accuracy in both detecting and localizing the required objects.

Our goal is to devise a method to detect doors and windows reliably and efficiently from observed data and register these with the constructed SLAM map. Using a CNN like YOLOv2 to take care of the object detection ensures a high level of accuracy while maintaining speed. It has also been found to perform much better than filters that are hand-designed to perform the detections. Windows are especially hard to detect using traditional methods, as reflections, objects on other side of the window, prevent successful detections of windows. YOLOv2 performs real-time detections by applying

a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region.

4.1 Data Preparation and Training

4.1.1 Data Collection and Preparation

Short video sequences containing doors and windows from real datasets were used to train the network. The data was collected from a variety of different sources including those from the site of the construction. Care was taken to collect images in different lighting conditions, during night and daytime, and from different distances, and angles. Since the use case is predefined, i.e, it would be used on a UAV that would not undergo extreme motions like flipping etc., the images were collected to mimic only slightly more than the range of motions that the UAV would go through in practice.

4.1.2 Annotation and Data Augmentation

The images were annotated using the LabelImg [47] software. The doors and windows in each image were marked using rectangular boxes and the coordinates were stored in normalized coordinates by expressing the coordinates as a fraction of the length and width of the image.

Slight perturbations in angle were provided to the images during augmentation. This was done by applying a rotation on the input images given by

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

where R_z is a rotation about an axis perpendicular to the xy plane and θ is the angle of rotation. This angle is kept to a maximum of ± 20 degrees in 5 degree increments to simulate the maximum roll of ± 20 degrees from the collected data. Inverse warping was applied to get rid of holes in the warped image. Holes are artifacts of forward warping where pixels in the warped image are not painted in due to the unavailability of one-to-one mapping from the source image to the destination image. The bounding boxes were resized to include the rotated bounding boxes. This was done by detecting the maximum and minimum limits of the bounding box corners after warping and constructing the new bounding box using these limits.

4.1.3 Training

The network selected for our work, YOLOv2 was trained on a desktop with a TitanX GPU for a week from scratch on the recorded and augmented data. A total of 1540 images were collected from short video sequences containing different doors and windows in different lighting conditions to account for variations in the operating environment. A total of 12320 images were generated from these images through augmentation. 80% of the dataset was randomly selected as the training set and 20% was used to validate the detection accuracy. This was done to compute the precision-recall curve for the detections using the 20% set aside. The precision-recall curve

provides information that is used to decide on the cut-off threshold on the detection confidence. This is discussed in more detail in section 6.1. Two classes, one each for the doors and windows were used to classify the detections. In addition, to remove detections that looked like a door or window, a third class was added. This class was not used in the evaluation but led to an increased accuracy of the network.

4.2 Detection Refinement

YOLOv2 provides rectangular bounding boxes of detections in the image space. Each of these bounding boxes correspond to a single detection of either a door or a window. These detections have to be translated into the actual objects in the 3D space. To do this, lines are first detected in each of the areas within the bounding boxes using LSD, a line segment detector [48].

4.2.1 Line Segment Detector(LSD)

LSD is a linear-time Line Segment Detector that claims to provide sub-pixel accurate results and claims to successfully detect lines on images without any parameter tuning [48]. It works by first generating a Gaussian pyramid from the original image, containing N octave levels, by down sampling N-1 times, and blurring at each level by the application of a Gaussian filter. From each layer in the pyramid, lines are then extracted using the various image processing mechanisms like, gradient computation, gradient pseudo-ordering, gradient thresholding, region growing etc. The algorithm is claimed to have an execution time that is proportional to the number of pixels in

the image. In practice, even though the algorithm did perform quite well in detecting lines, it was prone to breaking up lines even after rigorous parameter tuning. These detections were further refined using the methods outlined below.

4.2.2   Refinement

The lines that are closest to the vertical sides of the bounding box that are within a threshold distance from the sides are joined to form a single line provided they meet certain criteria. LSD outputs lines as point pairs denoting the endpoints of the line segments. These endpoints are used to compute the slope of the line segments.

(a)   (b)   (c)
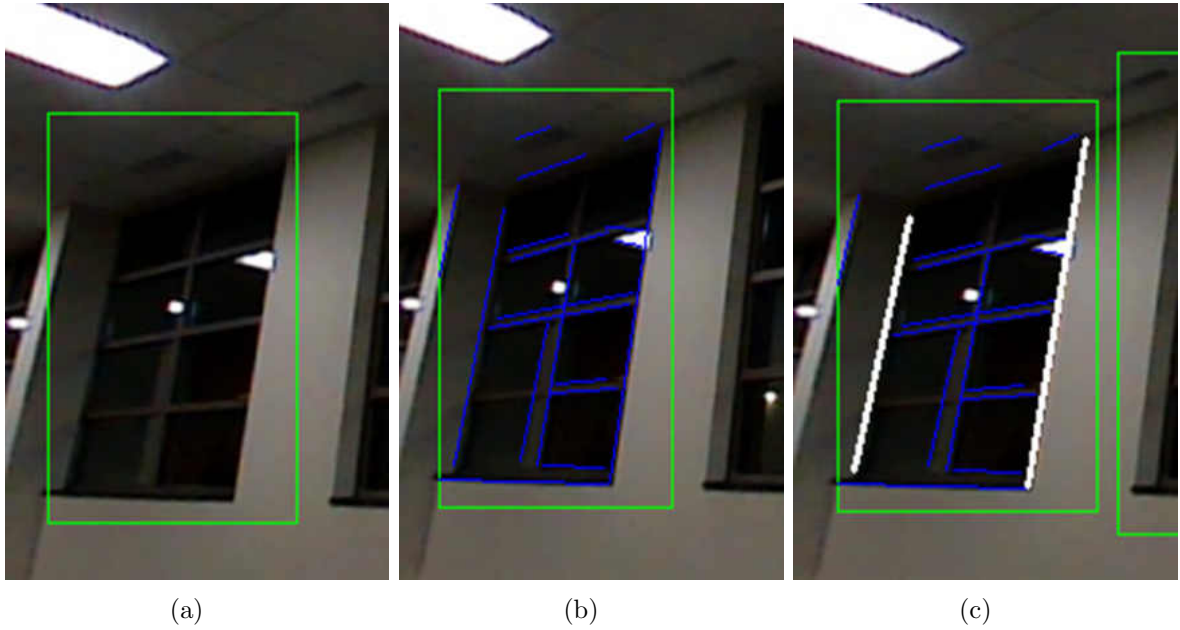
Figure 4.1.   Feature Detection and Refinement. The bounding box from YOLOv2 is shown in (a). Lines detected using LSD Algorithm are shown in (b). Two vertical sides of object are selected and marked in (c).

Line segments that have slope within a threshold angle between them, provided the closest endpoints are within a threshold distance are joined by getting rid of the closest endpoints and using the other two endpoints as the endpoints of the new joined line. The threshold for the angle in our implementation is set to 2 degrees and the distance in pixels between the closest endpoints of the line segments is set to 10 as they seem to have worked best for our system. The longest lines closest to the two vertical sides of the bounding box are then taken to denote the vertical sides of the door or window. This is illustrated in figure 4.1.

(a)   (b)
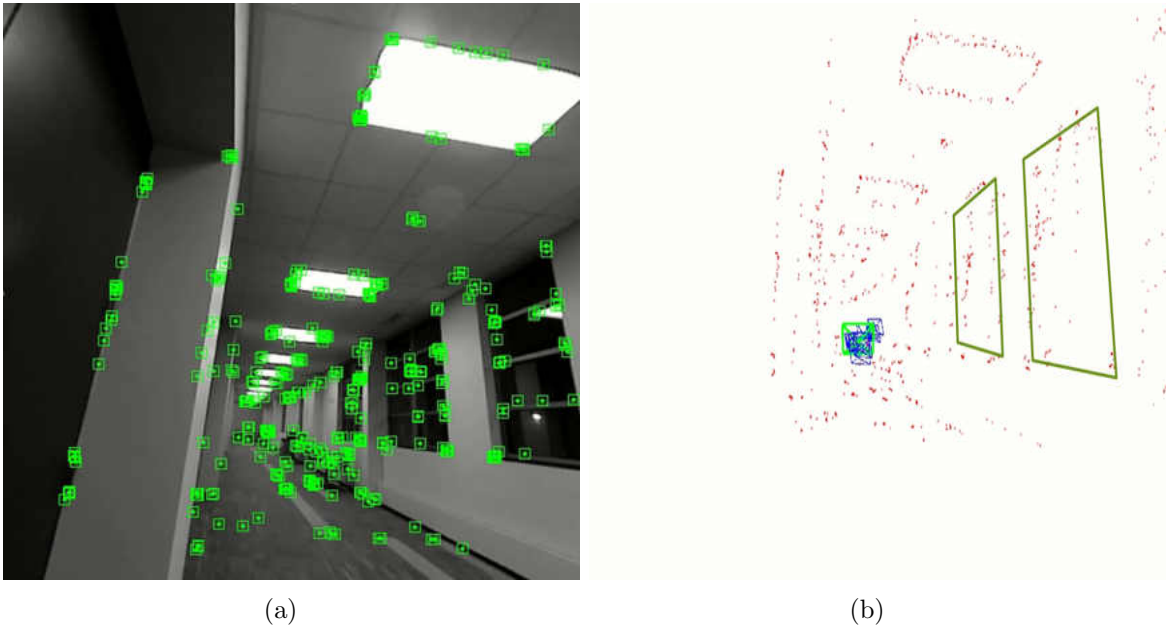
Figure 4.2. Reconstruction and Registration of Feature in 3D. The current frame with tracked ORB features is shown in (a). The detected windows projected into the 3D world coordinates of the SLAM map is shown in (b).

4.2.3   3D Reconstruction

Using the 2D lines detected for the vertical edges of the feature, we project the lines into the 3D space. The stereo image pairs are used to compute the depth from disparity at each point in the image. This computation is highly parallelizeable and is done on board the UAV utilizing the GPU of the Jetson TK1 using the algorithm described in appendix A.1. The left image and the disparity image are used to reconstruct the feature in 3D.

Points are sampled along the 2D lines detected in figure 4.1(c) and using the depth at each point, are projected into the 3D world coordinate system. This is computed by,

$$
\begin{bmatrix} P_{3x} \\ P_{3y} \\ P_{3z} \end{bmatrix} = d \begin{bmatrix} (P_{2x} - c_x)/f_x \\ (P_{2y} - c_y)/f_y \\ 1 \end{bmatrix}, \tag{4.2}
$$

where (P_{3x}, P_{3y}, P_{3z}) are the coordinates of the point in 3D with respect to the current pose of the camera in the world coordinate system(also called view coordinates), (P_{2x}, P_{2y}) the coordinates of the point in 2D image coordinates, (c_x, c_y) the coordinates of the principal point of the camera and (f_x, f_y) the focal length of the camera and d, the computed depth at the point. The sampled points are used to compute the line in 3D using a least-squares line fitting algorithm using orthogonal distance as explained in [49]. Outliers are removed by removing points that are at a threshold distance from the computed line. The line is once again recomputed from the inliers using the orthogonal regression used previously. This two step computa-

tion of the line provides a better estimate and is resistant to outliers in practice. The algorithm used for line fitting is shown in algorithm 1. The endpoints of the line is computed by finding the projection of the two endpoints of the line segment in 3D onto the fitted line.

Algorithm 1: Fitting a line using orthogonal regression as explained in [49]

Result: Descriptor vector, d

```
bool FitOrthogonalLine(int numPoints, Vector<n> points[],
    Vector<n>& origin, Vector<n>& direction)
{
    // Compute the mean of the points.
    Vector<n> mean = Vector<n>::ZERO;
    for (int i = 0; i < numPoints; ++i)
    {
        mean += points[i];
    }
    mean /= numPoints;

    // Compute the covariance matrix of the points.
    Matrix<n,n> C = Matrix<n,n>::ZERO;
    for (int i = 0; i < numPoints; ++i)
    {
        Vector<n> diff = points[i] - mean;
        C += OuterProduct(diff, diff);  // diff * diff^T
    }

    // Compute the eigenvalues and eigenvectors of C, where the eigenvalues are sorted
    // in nondecreasing order (eigenvalues[0] <= eigenvalues[1] <= ...).
    Real eigenvalues[n];
    Vector<n> eigenvectors[n];
    SolveEigensystem(C, eigenvalues, eigenvectors);

    // Set the output information.
    origin = mean;
    direction = eigenvectors[n-1];

    // The fitted line is unique when the maximum eigenvalue has multiplicity 1.
    return eigenvalues[n-2] < eigenvalues[n-1];
}
```

The four points that make up these line segments now represent the feature in 3D in the view coordinates. Each of these features detected are assigned to the keyframe in which they were first detected. As explained earlier, keyframes are frames in a graph

based slam that are tracked across multiple frames and consist of the camera pose at the instant along with the mappoints in the view of that keyframe and also contain the connection to other keyframes in the slam map that share closely correlated map points. Assigning the detected features to the keyframes ensures spatial consistency in the even of a bundle adjustment step occurring before the localization step is complete.

A homogeneous point in 3D is defined by

$$\vec{P} = \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} \tag{4.3}$$

where P_x, P_y, P_z are the three scalar coordinates. A transformation, $T \in SE(3)$ is represented by

$$T = \begin{bmatrix} R & \vec{t} \\ 0 & 1 \end{bmatrix} \tag{4.4}$$

where $R \in SO(3)$ is a rotation matrix and $\vec{t}$ is a translation.

At any point in time, the locations of the feature points in world coordinates can be computed by,

$$\vec{P_w} = T_w^c \vec{P_c}, \tag{4.5}$$

where $\vec{P_w}$ is the homogeneous coordinates of the point in the world coordinate system, $\vec{P_c}$ the homogeneous coordinates of the point in the view coordinate system, and T_w^c the transformation between the camera and the world coordinate system.

For the coarse registration phase that takes place after this step, each of the features computed above are further reduced to their respective centroids in the world coordinate system. The descriptors are then computed as explained in section 5.1.

5. FEATURE REGISTRATION

Registration is the process of transforming a set of data from one coordinate system into another coordinate system. For our system, we need to register the set of features in the SLAM coordinate system to the features in the CAD model. Since the UAV is already localized in the SLAM map, the registration of the SLAM coordinate system with the CAD coordinate system ensures that the two coordinate systems are aligned, thus localizing the UAV in the CAD coordinate system.

Using a SLAM algorithm that utilizes stereo images ensures that the SLAM map generated has an accurate estimate of scale. Since the two sets of data have the same scale, a rigid transformation is sufficient to align the two sets of data. To compute this transformation, point-to-point correspondences need to be found between the two datasets. Once the correspondences are found, A Random Sample Consensus (RANSAC) [50] based approach uses subsets of these correspondences to determine the best set of corresponding features and the required transformation is computed.

To speed up the process of finding corresponding features, a novel method has been devised that encodes each feature into a binary feature descriptor that is orientation and location invariant. The use of a binary descriptor ensures fast descriptor matching through the use of Hamming distance as the distance measure. Computation of Hamming distance can be done in a single XOR operation between the two binary strings. The orientation and location invariance helps in removing the dependence

of the feature descriptors on the coordinate system. The process of computing the descriptors is explained in section 5.1.

5.1 Feature Descriptor

To form matches between the two sets of features, the descriptor computation needs to be decoupled from the coordinate system of each of the points sets. This is done by encoding the distances and angles to the closest 5 features into the feature descriptor. Since the SLAM coordinate system has close to accurate scale, the relative distances and orientations between the closest features stays the same in both coordinate systems. One such example is shown in figure 5.1. As seen in the figure, when the features are reduced to their respective centroids, the relative distances and orientations between the features stay the same.

Figure 5.1(a) shows a portion of the CAD model with a feature whose descriptor is to be calculated. Figure 5.1(b) shows the SLAM map generated from a run, along with the detected features. The corresponding features are labeled in both images. The feature descriptor is computed by first retrieving the closest 5 features from the feature whose descriptor is to be computed. This is done efficiently by the use of a KD-tree to store the centroids of the detected features. Next, using the closest feature as the base, the angles to each of the 4 other features is computed using the formula,

$$\alpha = \arccos\left(\frac{\vec{a} \cdot \vec{b}}{||\vec{a}|| \cdot ||\vec{b}||}\right) \tag{5.1}$$

where $\vec{a}$ is the vector from the current feature to the closest feature, $\vec{b}$ is the vector from the current feature with the feature whose angle is to be calculated and α, the angle between the two vectors.
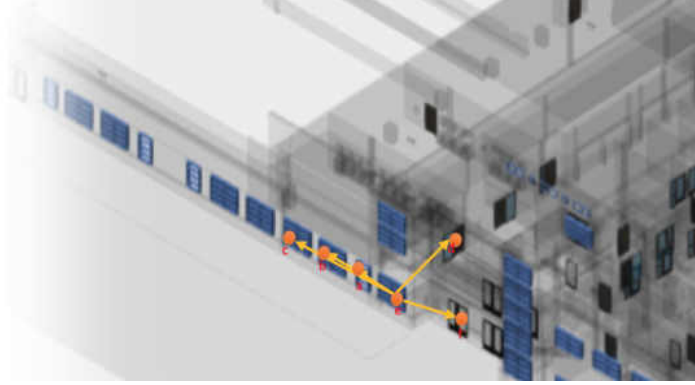
(a)
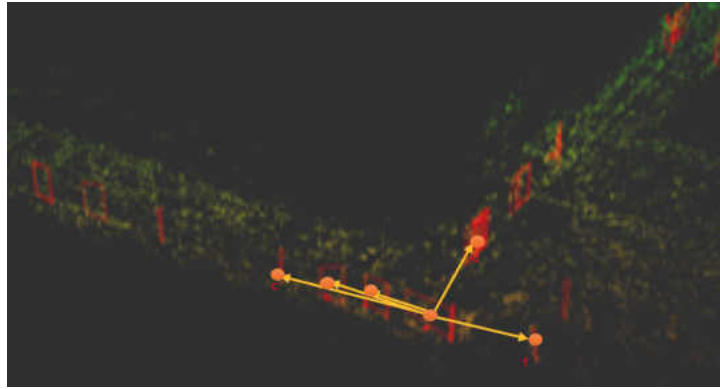
(b)

Figure 5.1. Orientation invariance of the Feature descriptor. The images show the relative distances and orientation of a few selected features, with 'e' being the feature of which the descriptor is being calculated. Image (a) shows the features in the 3D CAD model while image (b) shows the same features in the SLAM map

46

The encoding of the orientations and the distances to each of the nearest features are done by using a lookup table. The lookup table ensures that the values of distances and angles that are close to local means are assigned the same value. This binning process enables the encoding of the distances and the angles into a compact form that can then be used to form the binary descriptor. This process is outlined in algorithm 2. In addition to the distance and orientation information, the first bit in the descriptor is used to denote the type of the feature i.e. door or window.

Algorithm 2: Computation of Feature Descriptor

Result: Descriptor vector, d
Extract closest 5 features from KD tree with distances;
$v \leftarrow all\_5\_points$
$x \leftarrow current\_pt$
$vec_0 \leftarrow v_0 - x$
forall i *in* $[1, 2, 3, 4]$ **do**
    $vec_i \leftarrow v_i - x$
    $dist_i \leftarrow LUT_{dist}(|vec_i, vec_0|)$
    $\angle i \leftarrow LUT_{angle}(vec_i, vec_0)$
end
Descriptor vector,
  $d \leftarrow [feature\_type, [dist_1], [\angle 1], [dist_2], [\angle 2], [dist_3], [\angle 3], [dist_4], [\angle 4]]$

5.1.1 Lookup Table for Fast Descriptor Matching

The lookup table seen in algorithm 2 is actually computed through a binning process. There are two lookup tables created, one for angles and the other for storing distances. These tables are used to group together values that are close to local

means. The boundaries of these bins are computed using a process known as Kernel Density Estimation.

All the possible values are first computed using the CAD model. These values are then sorted and grouped into clusters. This clustering is done by first sorting all the values and then estimating the shape of the probability density function f, that provides a representation the data. This probability density function is estimated by the use of Kernel Density Estimation(KDE). KDE is a non-parametric method of estimating the probability density function. The Kernel Density Estimator, $\hat{f}_h$ is given by

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^{n} K_h\left(x - x_i\right) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right),$$
(5.2)

where K is the kernel (in our case a gaussian kernel), K_h is the scaled kernel and h is a smoothing parameter also known as bandwidth, $(x_1, x_2, \ldots, x_n)$ are a univariate, independent and identically distributed samples drawn from the distribution function, f. The $\hat{}$ is a notation to signify that the variable is an approximated value. By estimating and analyzing the shape of f, and dividing the distribution along the local minimae of this function, we can create the bins required for our look up table. This is illustrated in figure 5.2.

If we consider the underlying density of this density function to be gaussian, we can estimate the value of h using

$$h = \left(\frac{4\hat{\sigma}^5}{3n}\right)^{\frac{1}{5}} \approx 1.06\hat{\sigma}n^{-1/5},$$
(5.3)

where $\hat{\sigma}$ is the approximate standard deviation of the values. This approximation is known as Silverman's rule of thumb [51].
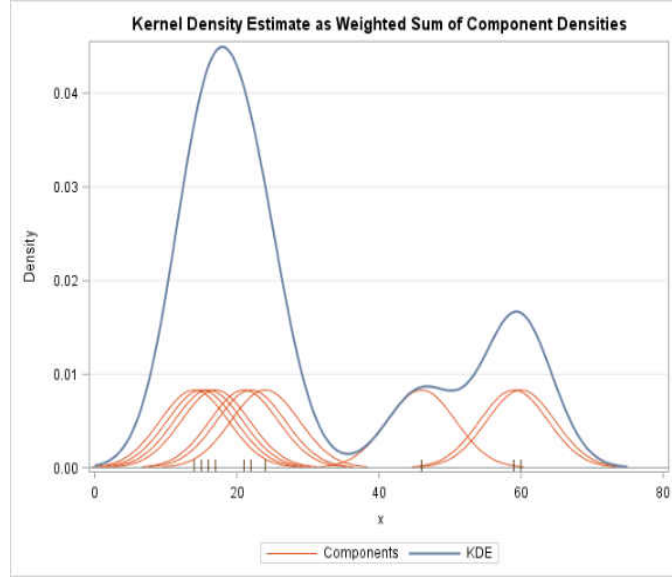
Figure 5.2. The Kernel Density Estimate. This is used to create bins for the lookup table.

Each of the bins are then assigned a number between 0 to the total number of bins, the maximum being 255 to fit in an 8-bit width in the binary representation of the number. This way each neighboring feature gives rise to two 8-bit values, one each for distance and the angle. The total size of the descriptor is thus 64 bits. The first bit in the 64-bit vector is usually zero as the number of bytes to represent the bins in the angles are less than 8 bits, in practice. So we store the type of the feature in this bit. This is done to ensure that the total size of the descriptor fits into a

49

multiple of 8, which it the number of bits that make a byte. The layout of the feature descriptor is shown in figure 5.3.

1bit	7bits	8bits	8bits	8bits	8bits	8bits	8bits	8bits
Feature Type	angle1	dist1	angle2	dist2	angle3	dist3	angle4	dist4

Figure 5.3. The 64-bit orientation invariant feature descriptor. The first bit is used to determine if the feature types match. The feature type is either 0 or 1, based on whether it is a door or a window. If the feature types match, the descriptor score is evaluated using the rest of the bits, If not, the maximum value for the distance is returned i.e. 64. The first bit in the 64-bit vector is usually zero so we store the type of the feature in this bit.

5.1.2 Descriptor Matching

Since this is a binary descriptor, matching of the descriptor is done by finding the Hamming distance between two descriptors [52]. The descriptors are matched on the basis of the lowest Hamming distance. Hamming distance computes the number of positions of two equal strings where the corresponding values are different. This is computed very easily and efficiently using the XOR operator and then summing the total number of set bits in the result. Computers with newer hardwares usually have support for counting the total number of set bits in CPU instructions. These are invoked by calling the $\_builtin\_popcount(x)$ operator of the GCC compiler. This

50

results in a very fast matching mechanism. The complexity for finding matches between the observed features and the CAD features is thus, $O(mn)$, with m being the number of observed features and n being the number of CAD features.

5.2 Initial Registration

During the initial registration phase, features are first extracted using the methods outlined in chapter 4. For each feature extracted, the centroid of the feature is stored in the KD tree and the descriptors are computed when the feature has more than a threshold number of neighbors. These descriptors and the descriptors computed from the CAD model are then matched using the methods outlined in section 5.1.

The matched features are then used in a RANSAC [50] based transformation estimation mechanism. RANSAC is a robust iterative method that is used to estimate the parameters of a mathematical model from noisy data that has outliers.

RANSAC works by selecting the minimum number of parameters needed to estimate a model hypothesis. The generated model is then used to compute the total number of inliers for the model hypothesis. This process is repeated for a total of N times to generate N hypotheses and the hypothesis with the most number of inliers is selected as the transformation. The number N is a function of the desired probability of success, p. $1 - p$ can then be considered to be the probability of failures. If w is the probability of selecting an inlier for each a point is selected, and m is the number of points needed to estimate the model, then $1 - w^m$ will be the probability of finding at least one outlier among the m points. $(1 - w^m)^N$ is thus the probability of never selecting m points that are all inliers. Thus,

$$1 - p = (1 - w^m)^N, \tag{5.4}$$

which simplifies to,

$$N = log(1 - p)/log(1 - w^m) \tag{5.5}$$

is the equation to estimate the total number of hypothesis that need to be generated for a desires probability of success, p.

N different hypotheses are generated to estimate the model. In our case, the model is the rigid transformation between two 3D point sets. Since this requires at least four point-to-point correspondences, the value of m in our case is four, which are randomly selected from the matched features. The rigid transformation is then computed using a linear least square method based on singular value decomposition(SVD). The Least Squares rigid transformation estimation method is described in section 5.2.1.

Coplanarity or collinearity of the points give rise to degenerate conditions and have to be avoided. Since collinear points are also coplanar it is sufficient to perform the check for coplanarity. The test for coplanarity can be done by computing the scalar triple product, which is given by,

$$(\vec{x_3} - \vec{x_1}) \cdot [(\vec{x_2} - \vec{x_1}) \times (\vec{x_4} - \vec{x_1})] = 0, \tag{5.6}$$

where $\vec{x_1}, \vec{x_2}, \vec{x_3}, \vec{x_4} \in \mathbb{R}^3$ are the four distinct points. The scalar triple product denotes the volume of the parallelepiped formed from the three vectors shown in equation 5.6.

This equates to zero for points on a plane. However, in practice it is better to assume a margin of error for the volume instead of zero. The equation thus becomes

$$(\vec{x_3} - \vec{x_1}) \cdot [(\vec{x_2} - \vec{x_1}) \times (\vec{x_4} - \vec{x_1})] < \epsilon, \tag{5.7}$$

where $\epsilon = 0.01$ is the margin of error. Any set of points giving rise to a value $< \epsilon$ are considered to be coplanar. If the selected points pass the coplanarity test, they are rejected and a new set of points are selected for the hypothesis. The hypothesis that generates the most number of inliers is then selected as the transformation.

5.2.1   Least Squares Rigid Transformation Estimation

The rigid transformation in $SE(3)$ between the two point sets is computed using the method outlined in [53]. If $A := \{\mathbf{a}_i | i = 1, 2, \ldots, n, \mathbf{a}_i \in \mathbb{R}^3\}$ and $B := \{\mathbf{b}_i | i = 1, 2, \ldots, n, \mathbf{b}_i \in \mathbb{R}^3\}$ are two corresponding sets of points, A being the source and B being the destination set, the rigid transformation to be computed is found by minimizing the squared error of the transformed coordinates, which is given by

$$(R, \mathbf{t}) = \underset{R \in SO(3), \mathbf{t} \in \mathbb{R}^3}{\text{argmin}} \sum_{i=1}^{n} w_i \left\| (R\mathbf{a}_i + \mathbf{t}) - \mathbf{b}_i \right\|^2 \tag{5.8}$$

where R is the rotation in $SO(3)$ and t in the translation and $w_i > 0$ are the weights assigned to the squared differences for each point pair.

We first compute the weighted centroids on both sets by,

$$\overline{\mathbf{a}} = \frac{\sum_{i=1}^{n} w_i \mathbf{a}_i}{\sum_{i=1}^{n} w_i}, \quad \overline{\mathbf{b}} = \frac{\sum_{i=1}^{n} w_i \mathbf{b}_i}{\sum_{i=1}^{n} w_i} \tag{5.9}$$

where $\overline{\mathbf{a}}$ and $\overline{\mathbf{b}}$ are the centroids of the two sets. Vectors are then computed using the point sets and the respective centroids by,

$$\mathbf{x}_i := \mathbf{a}_i - \overline{\mathbf{a}}, \quad \mathbf{y}_i := \mathbf{b}_i - \overline{\mathbf{b}}, \quad i = 1, 2, \ldots, n \tag{5.10}$$

where $\mathbf{x}_i$ and $\mathbf{x}_i$ are the corresponding vectors originating at the respective centroids. The 3×3 covariance matrix is then computed using

$$S = X W Y^\top \tag{5.11}$$

where X and Y are the vectors of dimension 3 and $W = \mathrm{diag}\,(w_1, w_2, \ldots, w_n)$, the weight matrix. The singular value decomposition of

$$S = U \Sigma V^\top \tag{5.12}$$

is computed where, U and V are orthogonal unitary matrices, and σ is a diagonal matrix with non-negative real numbers along the diagonal which are the singular values of S. The rotation matrix R is then given by

$$R = V U^\top \tag{5.13}$$

The translation can then be calculated by

$$\mathbf{t} = \overline{\mathbf{b}} - R\overline{\mathbf{a}} \tag{5.14}$$

(a)   (b)

(c)
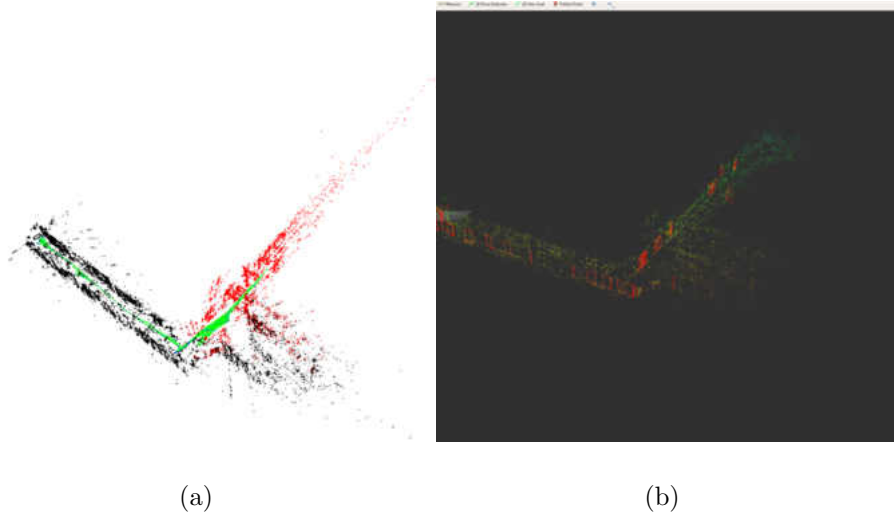
Figure 5.4. Feature Registration. The generated SLAM map is shown in
(a). Detected and reconstructed doors in the slam map are shown in (b).
The localized UAV is shown in (c)

5.3 Registration Refinement

After computing the transformation matrix using the least-squares based RANSAC method outlined above, the transformation is refined by using all the points from all the features. What this means is that we use all the four points that make up the corners of the features(doors/windows) in the 3D space and compute the least-squares rigid transformation using the method outlined in section 5.2.1. This gives rise to a better estimation of the transformation.

6. RESULTS AND DISCUSSION

The following sections evaluates the performance and accuracy of the different algorithms used in our research. Section 6.1 evaluates feature detection accuracy and performance and calculates the average precision for an Intersection over Union (IOU) over 65%. Section 6.2 evaluates the performance of the feature descriptor extraction and matching. Section 6.3 evaluates the performance of the system as a whole.

6.1   CNN Detection

The CNN used in the detection of doors and windows from images as explained in section 2.3 was tested with the testing data set aside from the augmented dataset. 20% of the augmented dataset containing 12320 images amounting to 2464 images were used to test the accuracy of the predictions from the network. Since the bounding boxes for annotation were resized to include the effects of rotating the image during augmentation, the annotations from the original images could be used in the validation.

Precision and recall are widely used to measure the performance of object classification and detection using CNNs. Precision and recall are defined by

$$Precision = \frac{TP}{TP + FP} \qquad (6.1)$$

and

$$Recall = \frac{TP}{TP + FN} \tag{6.2}$$

where TP is the number of True Positives, FP the number of False Positives, and FN the total number of False Negatives. Precision, as seen from the equation calculates what percentage of the positive detections are actually positive. Recall, on the other hand measures the percentage of correctly detected positives in the total pool of actual positives. True Positives were defined as detections that were predicted as the correct class and had an IOU over 65%. Detections with IOU less than 65% were added to the False Positives. It is worth noting that the IOU defined by

$$IOU = \frac{A_{prediction} \cap A_{groundtruth}}{A_{prediction} \cup A_{groundtruth}} \tag{6.3}$$

places a strict check on the 2D localization accuracy of th object detection. This enforces greater accuracy in the localization of the detections during the actual run of the algorithm which in turn enables better extraction and refinement of the features.

YOLOv2 outputs the confidence values for each of the detections produced. To determine an optimal threshold for confidence value, a range of values are tried out from 0 to 1 and the total number of True Positives, False Positives, True Negatives and False Negatives are counted. These vales are used to compute the precision and recall values for each of the values of the probabilities and a Precision-Recall curve is plotted. This is shown in figure 6.1.

Using the precision-recall values at each point, the best threshold for the confidence value was selected by the use of F-score. The F-score is a method of determining
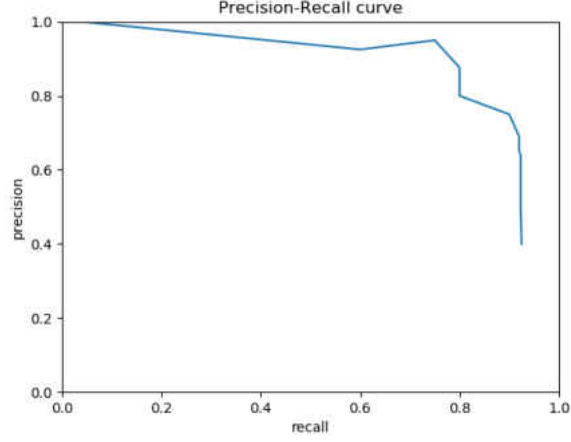
Figure 6.1. Precision-Recall curve for door/window detection

the ideal balance of precision and recall while deciding on a threshold. F-score is defined by,

$$F_\beta = \left(1 + \beta^2\right) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{ precision }) + \text{ recall}} \tag{6.4}$$

which is a measure of the effectiveness of retrieving the recall with β times more importance than precision. Since the same scene is viewed in multiple frames during the coarse of a single run of the UAV, the possibility of a feature being detected multiple times is high. We select a β value of 0.5 placing 2 times more importance to precision than to recall, to reduce the number of false positives in the detections. The highest F-score, $F_{.5} = 0.9019$ was found at the threshold value of .75 which was then selected as the threshold for the system.

Figure 6.2. Object detection and 2D feature extraction. The green rectangles are the predicted bounding boxes from running YOLOv2 on the input image. Lines are extracted from the within each bounding box and refined. The refined lines are shown in blue. The longest two lines closest to the two vertical sides of the bounding box are shown in white.

A few examples of the CNN predictions and subsequent feature extractions are shown in figure 6.2. A seen from the pictures, the algorithm is able to detect doors and windows accurately. The predicted bounding boxes fr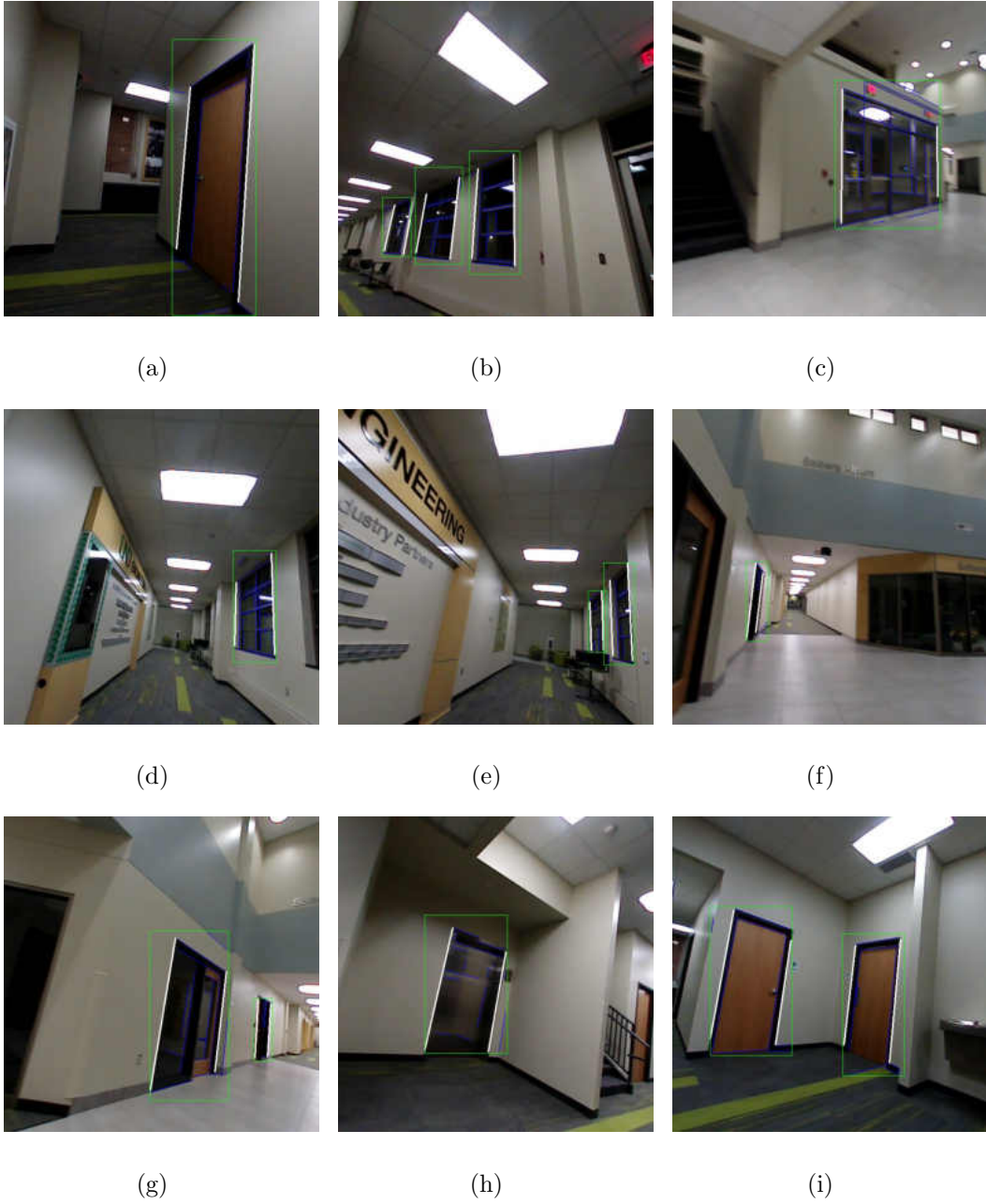om YOLOv2 are shown in green. The regions within these boxes are then used to run the image processing algorithms outlined in section 4.2. The blue lines are the line candidates detected using the LSD line detection algorithm, refined by joining broken lines that LSD failed to connect. The white lines are the two vertical edges of the feature selected from the lines detected. Figure 6.2(g), 6.2(h) show cases where smaller lines are detected that are closer to the sides of the bounding box. These are filtered out by the imposition of a threshold requirement on the length of the extracted line. Figure 6.2(d) shows a challenging case where a panel to the left side of the image would have been detected as a window using traditional image processing techniques, but is correctly filtered out from the detection.

Even though the Neural Network employed was able to successfully detect the required features, the image processing step employed to extract the vertical edges of the detected features would provide the incorrect results in a few corner cases as shown in figure 6.3. Figure 6.3(a) shows a case where the left edge is detected incorrectly due to the door being only partly visible in the image frame. Figures 6.3(b) and 6.3(c) show cases where the bounding box, even though it detects most of the feature, leaves out the edge, which results in the right edge not being detected correctly. Figure 6.3(d) shows a case where the left edge is detected correctly, but the lower portion is cut off due to the door not being fully visible. Figure 6.3(e) shows a case similar to 6.3(b) and 6.3(c) but is mostly due to the object being far from

61
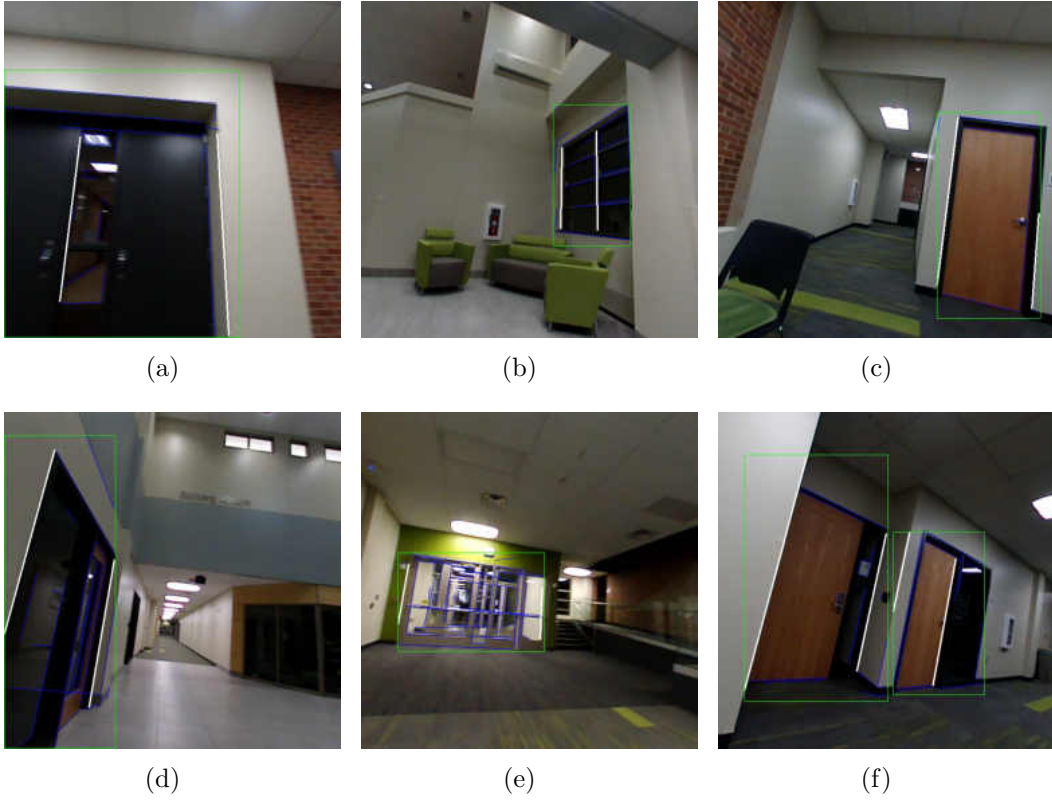
Figure 6.3. Failures in Feature Extraction. This collection of figures shows different cases that lead to bad extraction of features even after successful detection by YOLOv2

the camera and could be improved by adding a padding to the bounding box, i.e. increasing the size to get make sure the edges lie within the bounding box. Figure 6.3(f) shows two bounding boxes with incorrect edge extractions. In the left bounding box, the left edge is incorrectly detected as the edge of the wall. The right bounding box also detects the edge of the wall as the left edge of the door. This case could be improved by imposing stricter requirements on the line segments detected to be

classified as the feature edge, i.e. impose a threshold minimum length as a pergentage of the height of the bounding box for the detected lines in the bounding box.

6.2 Feature Descriptor Results and Accuracy

A naive method of selecting features based on distances to neighboring features was initially explored. In this method, lookup table of distances from each feature to every other feature was first computed from the CAD model in the preprocessing phase. Two features were then selected at random from the observed dataset and matched with distances computed from the CAD model. On finding a successful match, a new feature was then selected such that the distances to the two selected features was consistent with the distances in the CAD model. This process was repeated until a total of four correspondences were found. This method was found to be resource intensive and slow.

The histogram of the distances to the closest five features from each feature whose descriptor is to be calculated is shown in figure 6.4(a). The probability density function of the Kernel Density Estimator is shown in figure 6.4(b). As is seen from the comparison of the histogram with the KDE, the shape of the KDE closely resembles the histogram and is thus representative of the underlying data. Dividing the probability density function along the local minimae yields 24 different bins for the lookup table.

The accuracy of the matches formed on the basis of the descriptor created from this lookup table as explained in section 5.1 has been evaluated across 5 separate runs of the algorithm in different settings and has been shown in table 6.1. Each of the

matches are formed by finding the feature descriptor with the least hamming distance from the features in the CAD model.

(a)

(b)
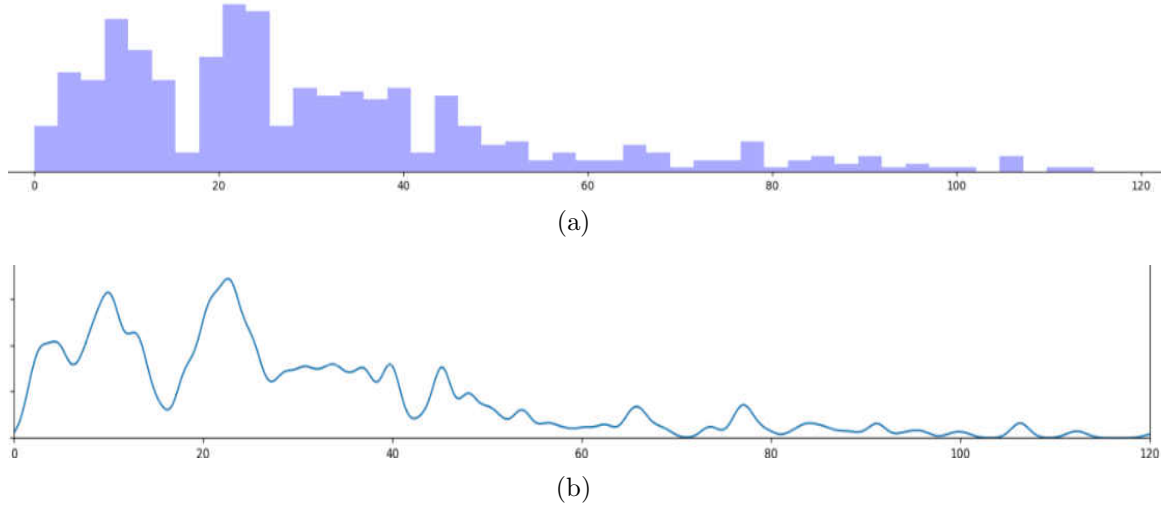
Figure 6.4. The histogram and KDE probability density function for distance. The horizontal axis represents the distance starting from 0. In figure (a), The histogram of distances between 5 closest features is shown. The vertical axis here represents the number of features at a particular distance. The KDE probability density function is shown in figure (b). Here the vertical axis represents the probability density of a particular distance.

It is seen that the matching accuracy increases with the number of features detected. This is expected, as increasing the number of features results in a more complete picture of neighboring features which are used to build the feature vector. Lower number of features detected leads to greater chance of ambiguity in the feature vectors due to multiple features having similar distances. Run 3 in one such example.

The extraction of the 5 nearest neighbors from the kd-tree has been found to take an average of approximately 0.052ms on the test system consisting of a 6th generation

core-i7 processor. The matching of descriptors using the brute-force matching technique has a complexity of $O(mn)$ and takes approximately 0.3ms for 25 descriptors in the observed dataset and 106 descriptors in the CAD dataset.

Table 6.1.
Feature Descriptor Accuracy. The total number of features matched for 5 runs is shown here along with the number of correct and incorrect runs. Accuracy is calculated by dividing the number of correct matches by the total number of matched features.

Run Id	Features detected	Correct matches	Incorrect matches	Accuracy
Run 1	21	15	6	71.4%
Run 2	25	19	6	76%
Run 3	7	4	3	57.1%
Run 4	31	24	7	77.4%
Run 5	19	14	5	73.6%

6.3 Localization within the CAD model

The descriptors computed from both the CAD model and the observed data are used to match the two sets of features and calculate the $SE(3)$ rigid transformation between the SLAM map and the CAD model. Figure 6.5 shows 5 runs of different lengths within the test setup. AprilTags [54] were placed at the goal positions for each run. AprilTags are fiducial markers that are not prone to the ambiguities inherent in other markers like Checkerboards or Circlegrids. To measure the accuracy of the

localization, the distance between the final location of the UAV and the april tag located at the goal position was measured.

The localization time was calculated based on how long it took the UAV to successfully localize itself in the CAD model from the time the system was started up. Table 6.2 shows the error between localization time and the error between the system goal position with the actual goal position for each of the runs depicted in figure 6.5.

Table 6.2.
Localization accuracy. The localization time and error between the system goal position with the actual goal position is shown.

Run Id	Localization Time	Error(m)	Error(% trajectory length)
Run 1	16	0.25	4%
Run 2	17	0.14	3%
Run 3	13	0.23	3%
Run 4	15	0.15	1%
Run 5	13	0.26	2%

The UAV was able to successfully localize itself within the CAD model approximately 15 seconds of starting on average. Run 4 had been performed with multiple goal locations to generate the complete map of the floor. This map is then used in a run to test the relocalization mode of the UAV.

(a)   (b)

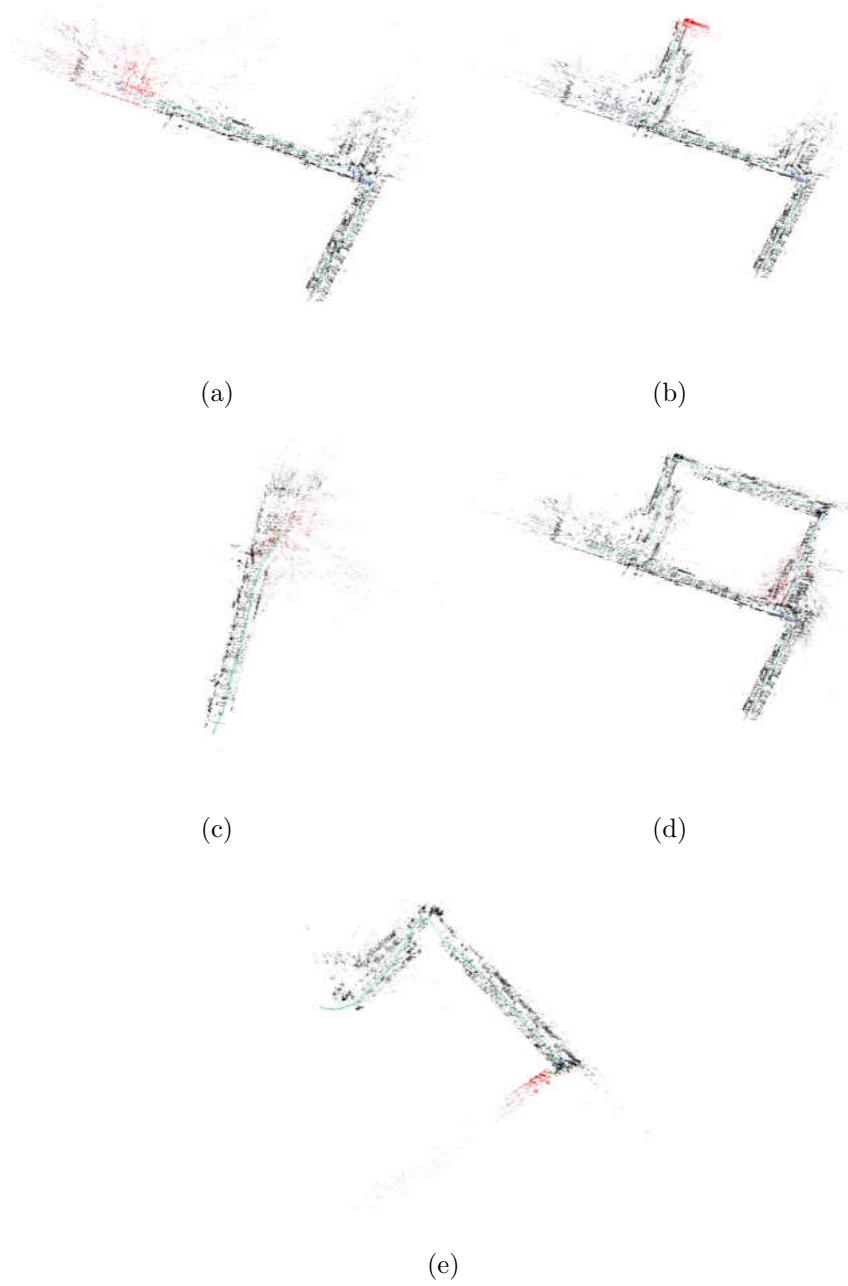
(c)   (d)

(e)

Figure 6.5. Runs of different lengths in the test setup. (a), (b), (c), (d), and (e) represent the top-down view of the generated SLAM map for Run 1, Run 2, Run 3, Run 4, and Run 5, respectively. The run in (d) has been set with multiple goal locations to generate a larger map for use in re-localization testing.

6.4 ReLocalization within the CAD model

To test the relocalilzation capability of the system, a SLAM map was generated from a previous run and its transformation with respect to the CAD model stored. These were then used when running the UAV a second time. The system was able to quickly localize the UAV within a second of the system initialization. This is possible because of ORB-SLAM2 uses a fast and efficient relocalization module. The initial SLAM map and the view from the perspective of the camera is shown in 6.6.

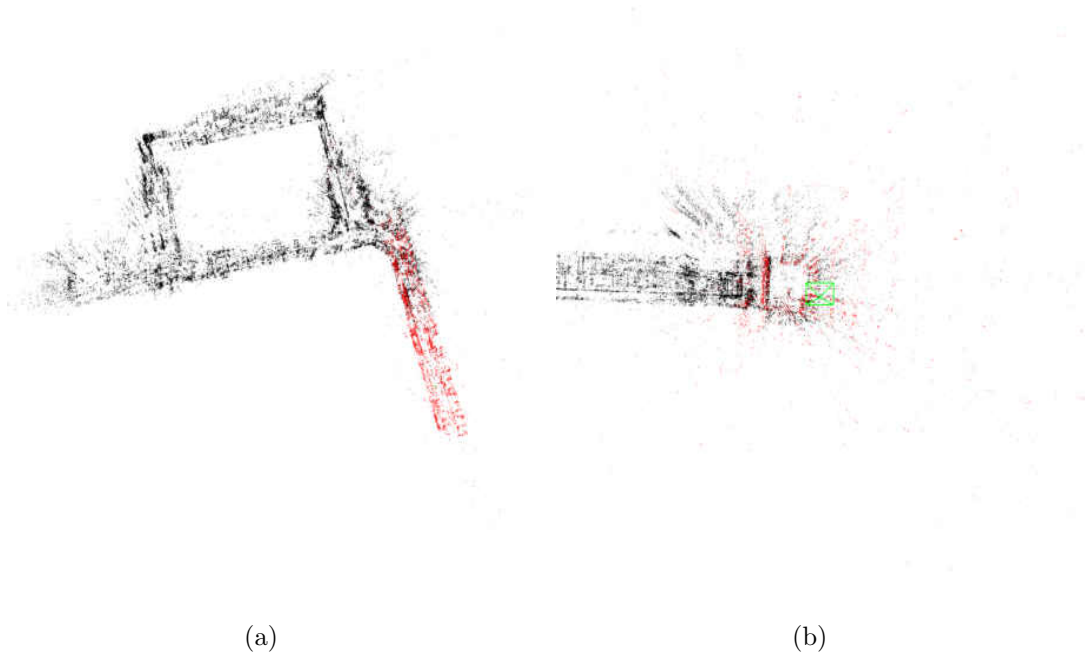(a)   (b)

Figure 6.6.   Relocalization of the UAV in previously constructed map. The previously constructed map is shown in (a). The UAV is shown localized in this map in (b).

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

Utilizing a pre-existing scale accurate SLAM system using stereo images to estimate depth, and an pre-existing real-time Convolutional Neural Network for object detection coupled with a novel and computationally efficient method of registration of the SLAM map with the CAD model, the system has been successfully tested to localize a UAV in a given 3D CAD model of a building.

The major contributions of this study has been two-fold. A novel pipeline combining a real-time CNN based object detection network, a SLAM system and a novel registration mechanism combined with image processing techniques to localize a UAV in a CAD model. A computationally efficient, orientation invariant feature descriptor to match features in the CAD model and the observed data, based solely on the spatial correlation between the features.

A major contribution of this work has been the fast and efficient computation of orientation invariant feature descriptors that were used to form correspondences between the observed features and the features in the CAD model. The use of a KD-tree enabled the quick extraction neighboring features. The use of Kernel Density Estimation to group clusters of discrete values together to reduce the time needed to match features had played a crucial part in improving the efficiency and performance of the algorithm.

Leveraging readily available 3D CAD models of buildings has been a core feature of the work and provides valuable information regarding the environment of the UAV. It provides the system with all the information needed to deduce the current location of the UAV within the CAD model, and enables the UAV to find its pose without the use of unreliable GPS in indoor environments or other costly systems that emulate the GPS mechanism indoors.

The use of a SLAM system registered to a CAD model also enables the system to leverage the relocalization feature of the SLAM to enable fast localization in previously visited locations.

To the best of our knowledge, the proposed system is the only one combining the accurate object detection mechanisms provided by modern Convolutional Neural Networks with a spatially accurate SLAM system to perform indoor localization.

7.2 Future Work

The system suffers from a few limitations that could be improved. The accuracy of the feature descriptor could be improved by applying a threshold on the computed hamming distance. This could be calculated by first saving all the computed descriptors and then studying the result of using different thresholds varying from 0 to 64 for the descriptor matches. All of these matches could then be used to generate an ROC curve, which could be used to deduce the best possible value for the threshold distance.

The system performance could also be significantly improved by the inclusion of other features like drinking fountains, exit signs, posts etc. Including more features

would enable the UAV to localize quickly due to the abundance of features and also due to the variation in the feature types which would give rise to less ambiguity in the feature descriptors. This would also alleviate problems arising from symmetric features.

The system currently relies on the connection to a ground control system(laptop) to do most of the processing. With a few optimizations made to the algorithms used in the work, it could be possible to enable all the algorithms to run on the onboard computer, thus enabling the UAV to function independently of the ground control system, using the connection to only receive the CAD model and the goal location. This would greatly increase the usability of the system as it would not have to be limited by the range of the wireless communication between the UAV and the ground control.

Some other areas that could be explored include being able to generate a temporal history of spatially correlated images from multiple visits to the same location. Leveraging the work done in this thesis, it could be possible to retrieve all images of a given location and thus monitor the progress in construction. The system, though developed with the construction industry in mind, could potentially also be used in other areas such as reconnaissance and rescue operations.

APPENDICES

A. APPENDICES

A.1 Semi-Global Block Matching

Semi-Global Block Matching [44] is the method used to compute the depth from the disparity image. It produces a better estimate of the depth as compared to naive block-matching algorithms. The algorithm minimizes the global energy function, E defined by,

$$E(D) = \sum_p \left(C\left(p, D_p\right) + \sum_q P_1 T\left[|D_p - D_q| = 1\right] + \sum_q P_2 T\left[D_p - D_q| > 1\right] \right)$$
(A.1)

with $P2 \geq P1$ where D is the disparity image, and $E(D)$ is the energy function; p, q are pixel locations, p, $C(p, D_p)$ is the pixel matching cost, P_1 is the penalty imposed for a change in disparity values of 1 between neighboring pixels N_p; P_2 is the penalty for values greater than 1. $I[.]$ is a binary function that returns zero or one based on the condition in the brackets. The function produces a smooth disparity map based on the parameters P_1 and P_2

LIST OF REFERENCES

LIST OF REFERENCES

[1] NIST GCR. Cost analysis of inadequate interoperability in the us capital facilities industry. *National Institute of Standards and Technology (NIST)*, 2004.

[2] Viorica Pătrăucean, Iro Armeni, Mohammad Nahangi, Jamie Yeung, Ioannis Brilakis, and Carl Haas. State of research in automatic as-built modelling. *Advanced Engineering Informatics*, 29(2):162–171, 2015.

[3] Pingbo Tang, Daniel Huber, Burcu Akinci, Robert Lipman, and Alan Lytle. Automatic reconstruction of as-built building information models from laser-scanned point clouds: A review of related techniques. *Automation in construction*, 19(7):829–843, 2010.

[4] Ian E Nordeng, Ahmad Hasan, Doug Olsen, and Jeremiah Neubert. Debc detection with deep learning. In *Scandinavian Conference on Image Analysis*, pages 248–259. Springer, 2017.

[5] Ahmad Hasan, Ashraf Qadir, Ian Nordeng, and Jeremiah Neubert. Construction inspection through spatial database. *arXiv preprint arXiv:1611.03566*, 2016.

[6] Francisco Agüera-Vega, Fernando Carvajal-Ramírez, and Patricio Martínez-Carricondo. Assessment of photogrammetric mapping accuracy based on variation ground control points number using unmanned aerial vehicle. *Measurement*, 98:221–227, 2017.

[7] Hesam Hamledari. Inpro: Automated indoor construction progress monitoring using unmanned aerial vehicles. *Master of Applied Science University of Toronto, Toronto, Canada*, 2016.

[8] Quentin FM Dupont, David KH Chua, Ahmad Tashrif, and Ernest LS Abbott. Potential applications of uav along the construction's value chain. *Procedia Engineering*, 182:165–173, 2017.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(6):1067–1080, 2007.

[11] Faheem Zafari, Athanasios Gkelias, and Kin Leung. A survey of indoor localization systems and technologies. *arXiv preprint arXiv:1709.01015*, 2017.

[12] Olivier Koch and Seth Teller. Wide-area egomotion estimation from known 3d structure. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[13] Lukas von Stumberg, Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. Autonomous exploration with a low-cost quadrocopter using semi-dense monocular slam. *CoRR, abs/1609.07835*, 2016.

[14] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[15] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[16] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[17] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[18] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, (6):46–57, 1989.

[19] Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and autonomous systems*, 8(1-2):47–63, 1991.

[20] Dieter Fox, Sebastian Thrun, Wolfram Burgard, and Frank Dellaert. Particle filters for mobile robot localization. In *Sequential Monte Carlo methods in practice*, pages 401–428. Springer, 2001.

[21] John J Leonard and Hugh F Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on robotics and Automation*, 7(3):376–382, 1991.

[22] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[23] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[24] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[25] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[26] Paul L Rosin. Measuring corner properties. *Computer Vision and Image Understanding*, 73(2):291–307, 1999.

[27] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[28] Tony Lindeberg. Scale invariant feature transform. 2012.

[29] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[30] Raul Mur-Artal and Juan D. Tard. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[33] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):142–158, 2016.

[34] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[35] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[36] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[37] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[38] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[39] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, July 2017.

[40] *Jetson TK1 Embedded Development kit*, 2018 (accessed March 10, 2018). `http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html`.

[41] Thomas Schulze, Alexander Gessler, Kim Kulling, David Nadlinger, Jonathan Klein, Mark Sibly, and Matthias Gubisch. Open asset import library (assimp), january 2012. *Computer Software*, 2012. `https://github.com/assimp/assimp`.

[42] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[43] Robert Sidgewick. *K-d Trees*, 2018 (accessed March 10, 2018). `https://www.coursera.org/learn/algorithms-part1/lecture/Yionu/kd-trees`.

[44] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2008.

[45] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[46] Amit Patel. *Amits A* Pages*, 2018 (accessed September 12, 2018). `http://theory.stanford.edu/~amitp/GameProgramming/`.

[47] Tzutalin. Labelimg, 2015. Git code (2015)`https://github.com/tzutalin/labelImg`.

[48] R. Grompone von Gioi, J. Jakubowicz, J. M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, April 2010.

[49] David Eberly. Least squares fitting of data. *Chapel Hill, NC: Magic Software*, 2000.

[50] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[51] Simon J Sheather. Density estimation. *Statistical science*, pages 588–597, 2004.

[52] Mohammad Norouzi, David J Fleet, and Ruslan R Salakhutdinov. Hamming distance metric learning. In *Advances in neural information processing systems*, pages 1061–1069, 2012.

[53] Olga Sorkine-Hornung and Michael Rabinovich. Least-squares rigid motion using svd. 3:1–5, 2017.

[54] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407. IEEE, 2011.