Simultaneous Localization And Modelling:
SLAM for Mobile 3D Printing

Jinbo Li

A Thesis

In

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science (Electrical and Computer Engineering) at

Concordia University

Montreal, Quebec, Canada

August 2018

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Jinbo Li

Entitled: Simultaneous localization and modelling: SLAM for mobile 3D printing

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

Dongyu Qiu Chair

Chair's name

Luis Rodrigues Examiner

Examiner's name

Tsz Ho Kwok Examiner

Examiner's name

Krzysztof Skonieczny Supervisor

Supervisor's name

Approved by _____

Chair of Department or Graduate Program Director

_____ 2018 _____

Dean of Faculty

ABSTRACT

Simultaneous localization and modelling: SLAM for mobile 3D printing

Jinbo Li

Traditional additive manufacturing is constrained by the workspace of the printer, i.e. printers can only print objects within the printer's boundary. Mobile 3D printing is developed here to fabricate large-scale objects that extend beyond a printer's workspace. Mobile 3D printing uses a small-size robotic system to build large objects by connecting multiple small segments. A possible example application for this is additive construction on extraterrestrial surfaces, using locally sourced material, to minimize the overall need for equipment and materials launched from Earth.

The system is equipped with both a laser total station (range and bearing sensor) and 3D scanner; measurements from these two sensors are fused to overcome the deficiency of each individual sensor. An Extended Kalman Filter (EKF) based Simultaneous Localization And Mapping (SLAM) algorithm is implemented in order to align neighboring segments. A representation for planar patches of the model being printed, with each patch represented by 2 angles for the normal vector plus a 3D point on the patch, is proposed and shown to be particularly suited for this type of task.

The system achieves sub-millimeter geometric accuracy and avoids the SLAM inconsistency problem for well beyond the bounds of odometry error that could be expected to be encountered in practice.

ACKNOWLEDGEMENTS

It would not have been possible to write this thesis without the help and support of the kind people around me.

Above all, I would like to thank my supervisor Krzysztof Skonieczny for his patient guidance and encouragement. I am extremely lucky to have a supervisor who cared so much about my work, and who responded to my questions and queries so patiently.

I also am grateful to my team members. Amir Nassiraei helped me on most of the mechanical work in this project, like drilling, wiring and sensor installing. Parna is always willing to answer my questions of the project. She has provided many useful suggestions. Ty helped me a lot in my thesis writing and word choice.

Contents

List of Figures

List of Tables

Abbreviations

CAD	**C**omputer-**A**ided **D**esign
CC	**C**ontour **C**rafting
DCP	**D**igital **C**onstruction **P**latform
EKF	**E**xtended **K**alman **F**ilter
FDM	**F**used **D**eposition **M**odeling
FSPF	**F**ast **S**ampling **P**lane **F**iltering
ICP	**I**terative **C**losest **P**oint
ISRU	**I**n-**s**itu **R**esource **Utilazation**
MSAC	**M**-estimator **SA**mple **C**onsensus
RANSAC	**RAN**dom **SA**mple **C**onsensus
SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
SLAMMOT	**S**imultaneous **L**ocalization **A**nd **M**apping and **M**oving **O**bject **T**racking
SPmodel	**S**ymmetries and **P**erturbations model
STL	**S**tandard **T**riangle **L**anguage

Symbols

$b_1^i \ b_2^i \ b_3^i \ b_4^i$	Planes parameters for plane i in general form
$e_1 \ e_2$	Eigenvectors of printer position part of covariance matrix
h^i	Jacobian matrix
m^i	Measurement
m_φ^i	The φ element of measurement vector m^i
$\hat{m}^i$	Prediction of measurement based on the predicted states
$r^i \ \beta^i$	range and bearing of landmark in polar coordinate system
u	The line used to cut error ellipse
$x_0^i \ y_0^i \ z_0^i$	Centroid of planar patch i
$x_{l,i} \ y_{l,i}$	Landmark position
$x_p \ y_p \ \theta_p$	Printer pose
$x_v^i \ y_v^i \ z_v^i$	vertex of a triangulated planar patch i
${}^G x$	Parameters in Global frame
${}^P x$	Parameters in Printer frame
${}^S x$	Parameters in 3D scanner frame
${}^T x$	Parameters in total station frame
$\alpha_1 \ \alpha_2$	Coefficients of eigenvectors
$\theta^i \ \varphi^i$	Planar patch direction in spherical coordinate system for planar patch i
μ_t	state vector at procedure step t
μ_{x_p}	Corresponding values in the state vector
$\bar{\mu}$	Intermediate state vector
δ	Printing error

$F_{x,i}$	Selection matrix
H^i	Selected Jacobian matrix
K^i	Kalman gain
L	Segmenting length
N	The number of planes in the model of the 3D object being printed
Q_s	3D scanner measurement error
Q_T	Total station measurement error
R	Odometry uncertainty
R_z	Rotation matrix of a rotation around z axis
$\bar{\Sigma}$	Intermediate covariance
$\bar{\Sigma}_{x_p,y_p}$	Printer position part of the updated covariance
Σ_l	Total station landmark covariance
Σ_p^i	Planar patch covariance for patch i
Σ_t	covariance at procedure step t

Chapter 1

INTRODUCTION

Expanding space exploration will generate a need for larger and more permanent bases and structures on extraterrestrial surfaces [11, 12]. NASA has shown great interest in a practice called in-situ resource utilization (ISRU) for over 40 years, and two of the major areas of this practice are related to mobile 3D printing technology: The technique could support extraterrestrial surface construction, as well as manufacturing and repairing parts of various structures. A mobile 3D printing technique is developed in this research which uses a small-size robotic system to build large objects with sub-millimeter precision.

1.1 Large scale printers

There is related work in the literature on large volume construction by 3D printer. Cesaretti et al [13] present a 3D printing technology called D-shape (Figure 1.1), aiming to build lunar soil habitats. This method holds sand together by spraying a binding-liquid on the desired part of each layer. After, the part of the sand that has not been sprayed by the binding-liquid is removed, and the remaining sand is binded into a model. This method works well on Earth. However, since D-shape has to use a machine larger than the model to be printed and sending such large volume equipment to space is extremely expensive, it is not ideal for extraterrestrial construction.

FIGURE 1.1: D-shape 3D printer [1]

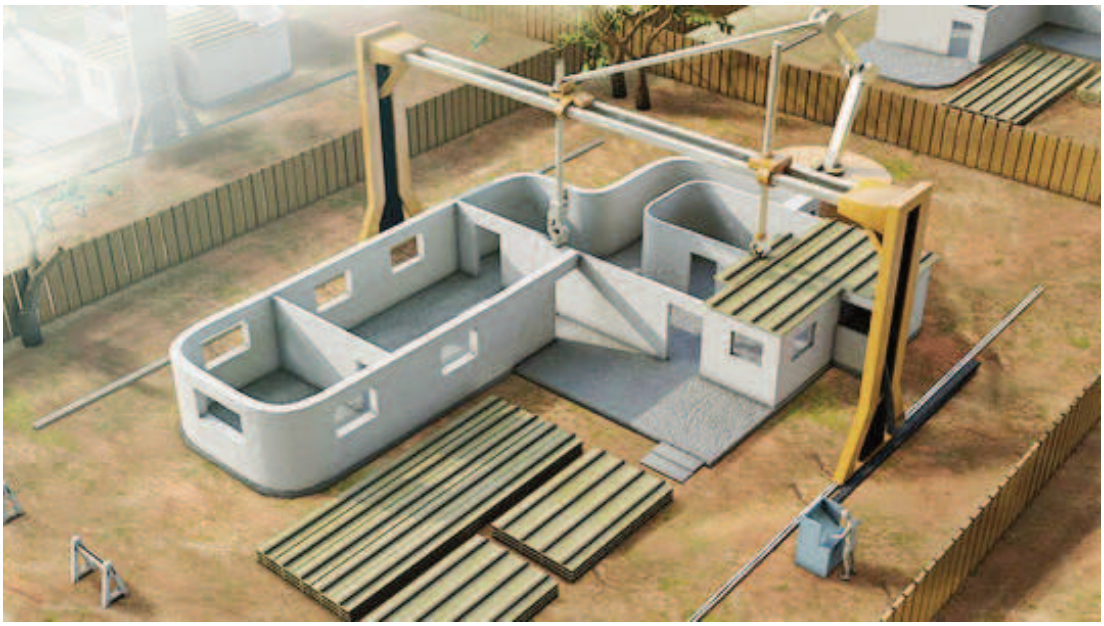

FIGURE 1.2: 3D concrete printer with contour crafting [2]

FIGURE 1.3: Digital Construction Platform (DCP) is printing a dome structure [3]

An additive fabrication technology called Contour Crafting is presented by Khoshnevis et al [14, 15] (Figure 1.2). Their approach also increases the size of the automated additive fabrication in order to construct structures. Additionally, they mention their future plan to use a mobile gantry robot building lunar bases [16]. The mobile gantry has two rover platforms connected by a crossbeam. On the crossbeam there is a nozzle which moves along the crossbeam and extrudes building materials. Bosscher et al [17] simplifies the Contour Crafting technology by replacing the gantry robot system with the cable-suspended robot system, so that the system becomes more portable and inexpensive.

In order to improve the mobility of automated construction system, Keating et al [18] present the Digital Construction Platform (DCP) which consists of a compound arm system carried on a tracked mobile platform (Figure 1.3). This platform is able to build on-site with a radius of 10.1m, and a maximum printable volume of $2786m^3$. "Printing while driving" and "print from a stationary position" two strategies are mentioned, but only the later strategy is conducted as a case study in their work.

FIGURE 1.4: MX3D company is using 3D printer to build bridge [4]

A company called MX3D built a bridge in mid-air by printing steel structure segment by segment separately (Figure 1.4). However, no detail is given on what the process is and how the localization is done after moving the 3D printer.

1.2 Mobile printing

The biggest challenge for the additive fabrication technique is the size of the machine used for construction [19, 20]. NASA proposes to mount the Contour Crafting system on the ATHLETE robot [21] or other mobile robots to achieve construction for large scale models (Figure 1.5). Their study mentions that for small structures, the positioning precision can be a few millimeters, but for larger structures, the positioning accuracy cannot achieve sub-centimeter, therefore 3D visual feedback is needed in order to achieve higher accuracy. However, no further detail is mentioned about how to implement that, and no other sensor besides stereo camera is mentioned.

Wilkinson et al [22] present preliminary findings from a workshop using a multi-robot system to address the problem of large-scale additive construction. They monitor topographical changes to the build site using a single overhead Kinect 3D scanner, and mention in passing the use of each robot's local information about where material has been deposited, but provide no further detail and do not achieve practical construction.

FIGURE 1.5: ATHLETE with printer head [5]

De Sá Bonelli et al [6] developed a mobile 3D printer called the 3&Dbot which can achieve printing while moving. However, no sensor is applied in this system and the robot is localized only by the odometry. Therefore, the accuracy it can achieve is low and the error will continuously accumulate as the printer moves.

As opposed to printing by a mobile robot on the ground, Hunt et al [23] combine 3D printing technique with aerial robots to build an aerial 3D printer. They demonstrate the feasibility of aerial 3D printing by bridging gaps in terrain and repairing damaged structures. However, they also mention that the major limitations of it is the small payload of flying robots and the flight stability. Dams et al [24] demonstrate the feasibility of aerial building manufacturing system using extruded polymers as building materials. They conclude that Reprocell 500 high-density foam has sufficient rheology and shear strength to be the aerial building material.

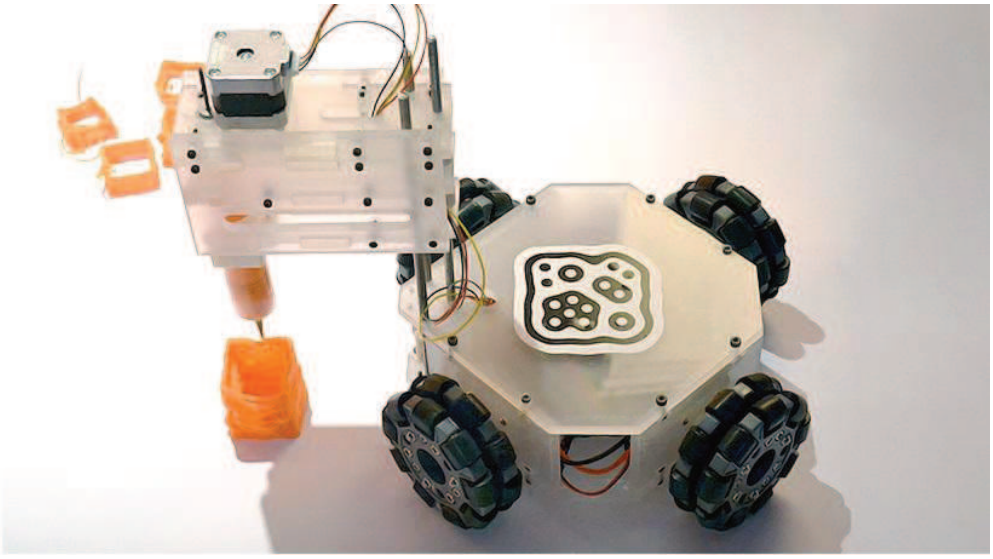

FIGURE 1.6: 3&Dbot Mobile 3D Printer is able to achieve printing while moving [6]

Labonnote et al [25] makes a summary of the state-of-the-art additive construction in terms of large scale construction, systematic mapping studies, building materials and building designs. There is still a general lack of published papers on the implementation of mobile 3D printing and the study of its potential challenges, so our study has important implications for future studies in this area.

1.3 Materials and printing methods

Research of the lunar material processing aspect of ISRU additive construction also shows some important progress. D. Shrunk et al [26] mention that the lunar regolith can be sintered by using microwaves in order to produce construction materials. Khoshnevis et al [16] demonstrate the feasibility of Contour Crafting (CC) technology combined with sulfur concrete and regolith sintering in the lunar environment. In-situ materials of regolith and sulfur are easy to find on the Moon, could save energy and are recyclable. Regolith can be sintered into either blocks or molten building material for CC, and it has been proven to be strong enough for building structures. Along with the research of D-shape printing technology, Cesaretti et al [13] develop a suitable lunar regolith simulant and prove that simulant can satisfy the required structural features in a vacuum environment. Krishna

Balla et al [27] demonstrate that raw lunar regolith can be melted by laser under a low energy level, and additive fabrication is successfully tested by using lunar regolith simulant. Mueller, R. P. et al [28] demonstrate the feasibility of using basalt regolith fines, which largely exist in planetary regolith, as construction materials. Flexural strength testing on samples made by lunar regolith simulant indicates that the strength of lunar regolith material is even better than residential concrete.

This research thus presumes the feasibility of using in-situ extraterrestrial materials for extended (e.g. fused doposition modeling (FDM)) approaches to additive manufacturing, and focuses on the robotic challenges. The thermo-plastic polylactide (PLA) is used going forward.

1.4 Printing on existing complex surfaces

Mobile 3D printing requires each new segment of a compound part to be printed onto existing other segments in a way that joins them together. In order to demonstrate printing onto pre-existing segments, Bulger et al [7] performs a tension test comparing monolithic to compound parts (Figure 1.7). The result shows directly printing material onto pre-existing segments of the structure being built is workable. Tension testing indicates the compound part interface is at least stronger than the stress concentration introduced at the gripping interface of the tension testing machine. Choi et al [8] modifies a fused deposition modeling (FDM) system by reversing the z stage, which normally displaces a mobile print bed up and down, and attaching the printer nozzle to the bottom of the z stage, which enables the system to print on any surface within the limit of building chamber. The modified system is used to print models on a horizontal 3D plate, a vertical wall, and a curved surface in order to demonstrate the feasibility of 3D printing on pre-existing surfaces (Figure 1.8). Espalin et al [29] develop a Multi-Material, Multi-Technology FDM System. This system has a pneumatic slide connecting two FDM machines. The printed model is transported between two FDM machines through pneumatic slide, controllable with high precision, in order to accomplish multiple materials printing during the same build.

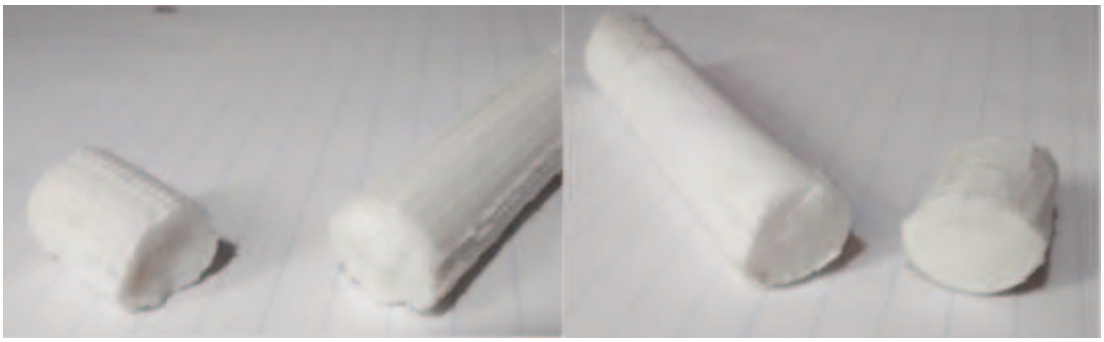

FIGURE 1.7: Result of tension test, the figure shows the failure surfaces of the monolithic (left) and compound (right) test specimens. The breakage did not happen at the intersection between two pieces but happened at the clip position [7].

FIGURE 1.8: ABS P400 cylinder built on black PC/ABS plate in the flexible FDM system [8]

1.5 General introduction to Kalman Filtering for Simultaneous Localization and Mapping

Mobile 3D printing requires precise robot localization to enable placing the nozzle on the interface with a prior segment. Extended Kalman filter (EKF) based simultaneous localization and mapping (SLAM), is used in robot localization when the system is nonlinear. Kalman filter is an algorithm for linear systems. It uses multiple measurements, statistical noise (i.e. Gaussian noise), and odometry prediction, combined with probabilistic calculation, to give a more precise robot localization result than a single sensor measurement. EKF is an extension of the Kalman filter, it linearizes a nonlinear function by using partial derivatives i.e. Jacobian matrix. A SLAM algorithm keeps track of the robot position and orientation, while also constructing and updating a map. In our study, the "map" is a parameterization of the 3D-printed model (i.e. the object being built) and SLAM is introduced in order to get a more accurate absolute and relative position between printer and model. It improves the alignment of the entire model and decreases drift.

The EKF is a common approach for the SLAM algorithm, and typically it is feature based. By measuring the change of feature positions, the algorithm makes an estimation of where the robot (or the object equipped with sensor) is. Meanwhile, many features can be used as landmarks, such as points, lines, planes and features in images. Williams et al [30] propose an approach based on the EKF-SLAM algorithm to navigate an undersea vehicle by using scanning sonar. Sonar targets are deployed at the field test site, acting as point features. As the underwater vehicle operates, it identifies sonar targets, as well as the reef wall or a rocky outcropping, then the system builds a feature based underwater map. Similarly, in our study, prisms are used as point features to help localize where the printer robot is in global frame. Garulli et al [31] and Smith et al [32] present algorithms extracting line features from range scans, while simultaneously updating the robot pose and the linear features on the map. Clemente et al [33] develop a single hand-held camera, which identifies features from images. Having the measurements from camera, the system can build outdoor closed-loop maps using EKF-SLAM. Additionally, Leonard et al [34] develop an EKF based localization

algorithm using points, lines and arcs as landmarks. The robot is equipped with multiple servo-mounted sonar sensors and the EKF algorithm provides vehicle position updates.

EKF-SLAM has been applied with much success, but it also has some disadvantages. The EKF-SLAM is very sensitive to outliers in landmark detection. If the correspondence is not determined correctly, a large error will be introduced to the system. Moreover, the EKF suffers from a $O(K^2)$ complexity where K being the number of landmarks [35, 36]. In addition, consistency is one of the most important criteria to measure if an estimator is reliable or not; it reflects whether the result of estimation converges to the true value as the number of data points used increases indefinitely. Unfortunately, EKF-SLAM always has inconsistency issue when the model is non-linear [30, 37].

There are some other SLAM methods developed to overcome disadvantages of EKF-SLAM. Sasiadek et al [35] present a comparison between the EKF-SLAM and FastSLAM. The result shows that FastSLAM gives better performance for non-linear and non-Gaussian conditions. Sim et al [38] introduce a vision-based SLAM using the Rao-Blackwellised Particle Filter. This method has a better performance of handling outliers, and meanwhile improves the particle filter which scales poorly with respect to the dimensionality of the state. Grisettiyz et al [39] present an improved method to reduce the number of particles in the Rao-Blackwellized Particle Filter; it also decreases the uncertainty of robot pose in the prediction step. Thrun et al [40] introduce GraphSLAM, which is a unifying algorithm for the offline SLAM problem. This method can handle a great number of features, so it is especially suitable for large-scale mapping problems.

The reason why we choose EKF-SLAM over other SLAM algorithms is that it is easy and sufficient to use at this stage of our work. Currently, the system is implemented for CAD models with simple geometry, so that the number of states in the state vector is small (typically less than 100). Moreover, the total station provides an accurate measurement of the system's yaw angle orientation, so it is easy for us to find correspondence by measuring angles of planes. Even if there are plane patches that lie on the same plane, the algorithm can distinguish them by their positions. Furthermore, the consistency does not seem to be

an issue in our work, because even if a large odometry error is introduced into the system, the printing result still maintains good quality, as will be shown in Section 4.4.

1.6 Plane patches in SLAM

When a stereo camera or a laser scanner is used as a sensor, detected planes are often extracted from point clouds and work as the feature landmarks, since they have a relatively large area and can be easily spotted. Several papers propose methods using planar patches as features in the SLAM algorithm. Zureiki et al [41] present a method using planar patches as landmarks in SLAM. While exploring the environment, the algorithm localizes the robot pose and simultaneously updates the plane model. The SLAM algorithm generates a 3D map showing both the robot track and the planes in the environment. This study represents planes in the general form. Each plane is given by its normal vector and its distance to the origin. Meanwhile, Bolle et al [42] represent complex 3D objects by triplets $S = (v, p, P)$ where v is an orientation vector, p is a location vector, and P is a size scale. For 3D planar patches, $S = (v, p, 0)$ where v is the normal vector of the plane and p is the vector from origin point to the center of mass on the plane. Biswas et al [43] present the Fast Sampling Plane Filtering (FSPF) algorithm, which extracts planes and points corresponding to planes from 3D point cloud. Then the algorithm down-projects the 3D planes into 2D and finds the correspondence of them and the points, in order to build a 2D map. Other than that, Viejo et al [44] mention an Iterative Closest Point (ICP)-like method combined with SLAM. This method uses the automatic seeded selection algorithm to extract planar patches from 3D point cloud. Having multiple planar patch data from different scanning positions, a robot can obtain the movement it performed by pose registration. However, only the pose of the robot is estimated from pose registration, in order to estimate the planes on the model as well, SLAM algorithm is required. Furthermore, Gee et al [45] describe a real-time SLAM algorithm discovering planes and lines captured by a hand-held camera in an indoor environment. Their work represents planes in nine parameters, which include plane origin, and two orthonormal basis vectors laying on the plane. Weingarten et al [46] develop a SLAM

algorithm based on planes, namely the Symmetries and Perturbations Model (SPmodel), which is used to represent planar features. [47] SPmodel is a representation of uncertain geometric models. It is worth noting that unlike classic probabilistic models which use different parameters to represent different geometric elements, SPmodel is able to represent any type of geometric elements and their respective uncertainties.

The comparison of the previously mentioned SLAM algorithms that use planar patches as landmarks are listed in Table 1.1. As shown in the table, on the one hand, most classical planar patch representations use normal vectors to indicate the directions of the planes, while the representations of plane position vary. On the other hand, SPmodel is different from all of those classical representations; it uses probability theory to represent the imprecision of any geometric element and symmetries theory to represent the partiality (degrees of freedom of each individual element and position relationship between different elements) due to the characteristics of the geometric element.

A novel plane parameterization particularly suited for mobile 3D printing will be introduced in Section 2.2

1.7 Predictive SLAM

Chang et al [48] present a SLAM algorithm with Environmental-Structure Prediction. This algorithm predicts the structure inside an unexplored region before the robot actually measures it. The prediction is based on the surrounding of that unexplored region and compares it with the explored region in the map. A correct prediction can reduce the processing time of SLAM. Ström et al [49] propose a similar approach making prediction about the unexplored surrounding area based on the previously explored area. Chung, S. Y. et al [50] extend the Simultaneous Localization and Mapping and Moving Object Tracking (SLAMMOT) problem to simultaneous map prediction and robot trajectory prediction. The robot can at the

TABLE 1.1: Comparison of SLAM algorithms that use planar patches as landmarks

	point cloud	plane normal	distance to origin	center of mass/ centroid on the plane	orthonormal basis vectors on the plane	three Cartesian coordinates and three roll-pitch-yaw angles
General form [41]		✓	✓			
Triplets $S = [v \quad p \quad 0]$ [42]		✓		✓		
FSPF [43]	✓	✓				
automatic seeded selection algorithm [44]		✓		✓		
planar structure formed by 3D points [45]		✓			✓	
SPmodel [46]						✓

same time passively execute SLAMMOT and actively predict the unexplored map. Predictive SLAM is relevant to our formulation of mobile 3D printing, because it is very useful to predict the "map" (i.e. parameterization) of the model we are printing based on its CAD.

1.8 Sensor fusion

In order to apply multiple sensors to the system at the same time and overcome the deficiency of any individual sensor, sensor fusion is a necessary technique to use. Ahn et al [51] propose a practical approach for EKF-SLAM in indoor environment by fusing ultrasonic

sensors and stereo camera. It has the advantage that "it can resolve the false data association and divergence problem of an ultrasonic sensor-only algorithm and overcome both the low frequency of SLAM update caused by the computational burden and the weakness to illumination changes of a vision sensor-only algorithm". Nützi et al [52] fuse IMU and vision for absolute scale estimation in monocular SLAM. This fusion helps the system to get an estimation of the vehicles absolute position and velocity without drift. Zhang et al [53] present a sensor fusion strategy for monocular camera and laser rangefinder applied for SLAM in dynamic environment, which "eliminates any pseudo segments that appear from any momentary pause of dynamic objects in laser data". As will be presented in the next chapter, this work fuses a 3D scanner and total station laser range-finder.

1.9 Outline of remainder of document

This thesis starts by this introduction as Chapter 1, then covers the overview of the mobile 3D printing system in Chapter 2, which introduces the geometry and state vector of the system. Later in Chapter 3, the procedure of mobile 3D printing is demonstrated by a flowchart, then introduced block by block in each section. Chapter 4 shows the experimental results of mobile 3D printing and demonstrates the importance of each individual implementation. After that, Chapter 5 gives the conclusion and future work. We also present the measurement of printing error and sensitivity of the algorithm to the odometry error in Appendix A and B. The code is attached in Appendix C.

1.10 Contributions

The main contribution of this work is a procedure named **mobile 3D printing**, which can fabricate large-scale objects that extend beyond a printer's workspace. This technique uses a small-size robotic system to build large objects (Figure 3.1).

Aside from the novel procedure itself, key insights that were identified to enable the main contribution include:

- A determination of the sensors required to achieve mobile 3D printing. Our system is equipped with total station and 3D scanner as sensors, and by integrating sensor fusion, the deficiency of each individual sensor is overcome, in order to achieve sub-millimeter alignment precision. Meanwhile, an EKF based SLAM algorithm is used to accurately localize the printer pose and printed model, so that the next segment can be printed at an appropriate position and have a solid connection with the previous segment. Since we fuse two sensors in the system, there are also two separate parts of the SLAM algorithm for both total station and 3D scanner. Among them, the total station SLAM is a standard SLAM which updates landmarks and printer pose simultaneously, while the 3D scanner SLAM is a predictive SLAM which predicts the shape of model after printing, before we actually measure it.

- Furthermore, a proposed plane representation that avoids problems related to normal vector re-normalization and general form plane representations.

Chapter 2

SYSTEM OVERVIEW

2.1 Geometry of the system

Mobile 3D printing utilizes a robot consisting of a 3D printer equipped with a global landmark sensor (i.e. total station) and local 3D geometry sensor (i.e. 3D scanner). Figure 2.1 shows the layout of the Zego delta 3D printer equipped with the Leica TS16 total station and the GOM Core 3D scanner. The base plate of the 3D printer is removed to enable printing directly onto the surface below the printer. In addition, it is also assumed that at least three global landmarks have been placed (e.g. on the perimeter of the construction site).

For the system (Figure 2.2), there is a printer coordinate frame, a scanner coordinate frame, a total station coordinate frame and a global coordinate frame. The global coordinate frame is defined with the same origin as the initial printer frame, rotated by $-\frac{\pi}{2}$, and it is fixed once defined. Transformations from the total station coordinate frame and the scanner coordinate frame to the printer coordinate frame are required in order to convert measurement from these two sensors into the printer coordinate frame. Therefore, two calibrations should be done to the total station and the 3D scanner with respect to the printer, in order to calculate the relative position between coordinate frames precisely. These will be presented in section 2.2.3.

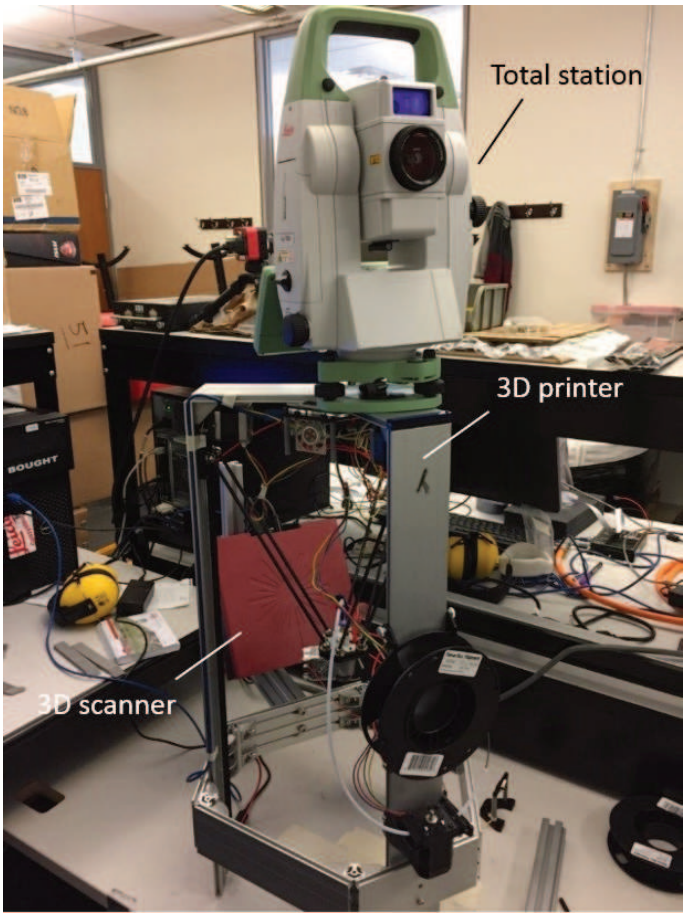

FIGURE 2.1: Configuration of the mobile 3D printer

2.1.1 Model of object to be printed

The model of the object to be printed is designed in CAD (Computer-Aided Design) and saved in STL format (Standard Triangle Language, a format that stores planes in triangulated surfaces, Figure 2.3). Notice that in this thesis, 'model' refers to the whole CAD model and 'segment' refers to the segmented model. An STL file can directly be used for printing software, and can also be read and converted into vertices and normal vectors of the triangulated surfaces on models. Those parameters are all defined in an STL coordinate frame. In order to simplify our work, we define the STL coordinate frame equal to the global coordinate frame, so no further transformation is required.

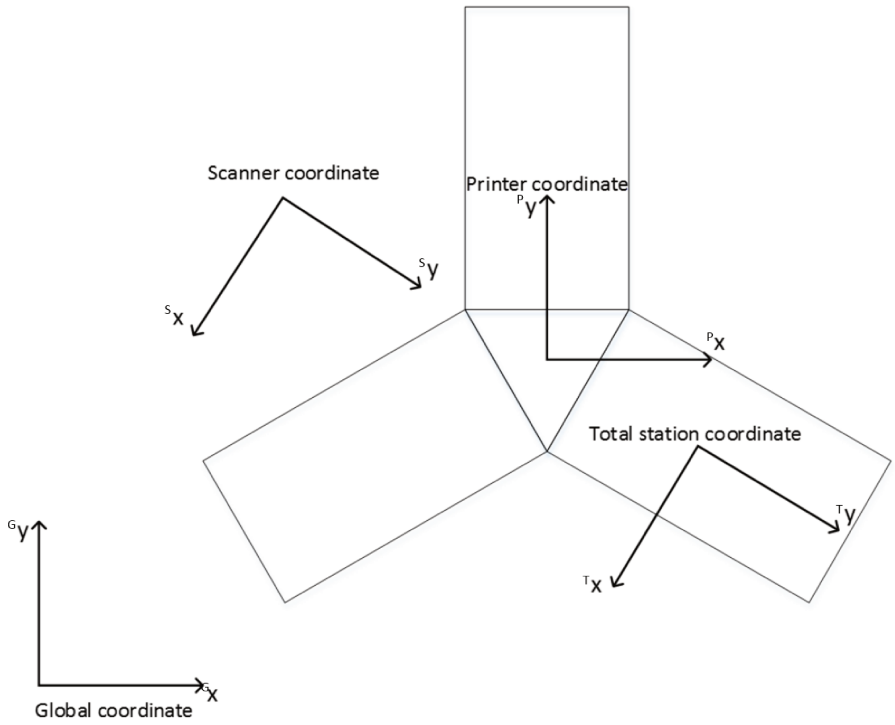

FIGURE 2.2: Coordinate frame definitions of the mobile 3D printing system

2.2 State vector

2.2.1 Plane parameters

The flowchart in Figure 2.4 shows the whole printing process. It consists of repeatedly printing segments and moving the printer, interspersed with sensor measurements to achieve state estimation. Each of the blocks will be explained in detail in chapter 3. In short, during the total station blocks, the total station does a set of measurements to the prisms (point landmarks), and the algorithm updates printer pose and landmark positions. During the 3D scanner blocks, the 3D scanner does a scanning of the printed model, and the algorithm updates the printer pose and plane parameters of the printed model.

The first step of implementing EKF SLAM is to choose an appropriate state vector, μ. Estimates of the state vector, μ_t, are updated at each procedural step t, for $t = 0, \cdots, 6$ as shown in Figure 2.4. Since the objects of study are the printer, landmarks and model printed, the state vector of the SLAM algorithm should have three partitions: the printer pose

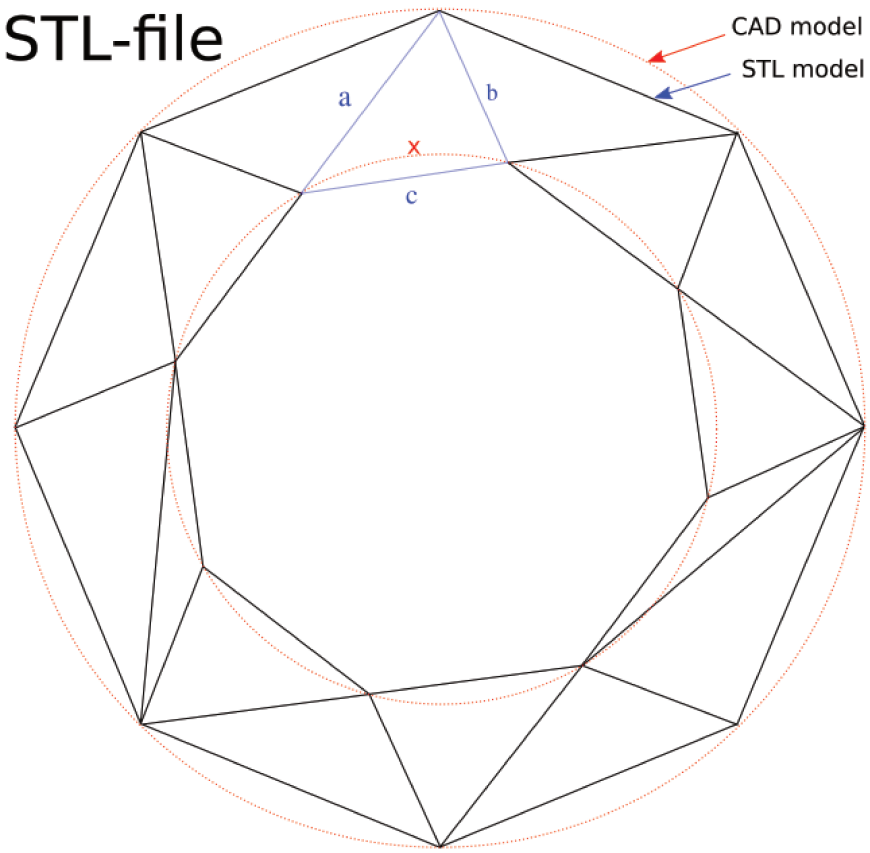

FIGURE 2.3: STL format saves models in triangulated surfaces, if the model has curves, it will be converted to the approximation of the same shape (Figure cited from wikipedia [9]: The differences between CAD and STL Models)

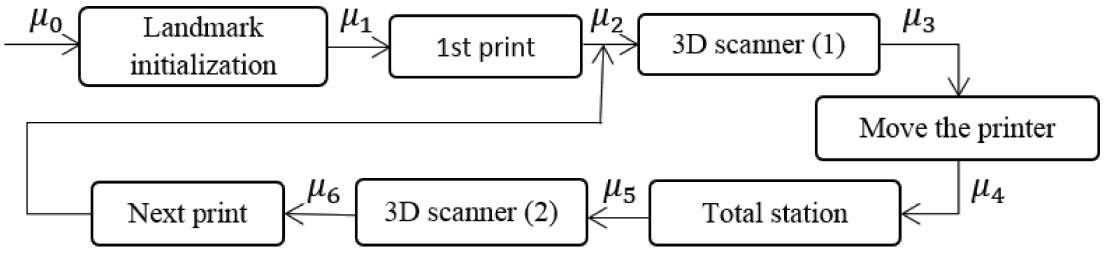

FIGURE 2.4: Flowchart of the mobile 3D printing procedure

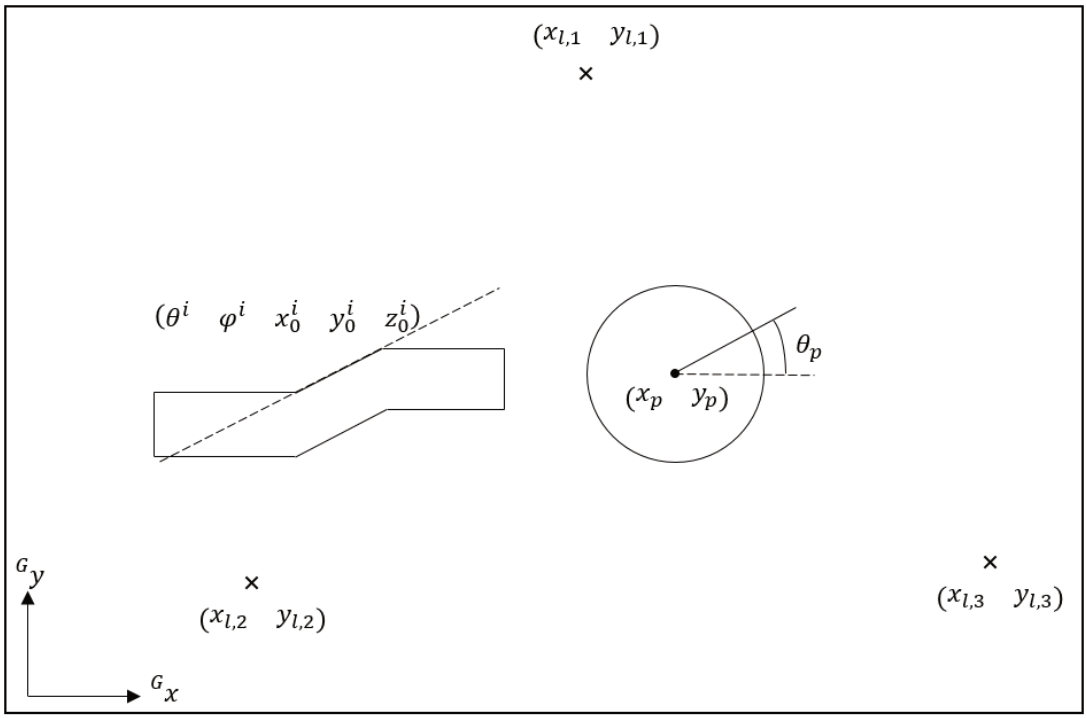

FIGURE 2.5: Schematic diagram of mobile 3D printing system. All variables expressed in global frame

$\begin{bmatrix} x_p & y_p & \theta_p \end{bmatrix}^T$, landmark positions $\begin{bmatrix} x_{l,1} & y_{l,1} & x_{l,2} & y_{l,2} & x_{l,3} & y_{l,3} \end{bmatrix}^T$, and plane parameters $\begin{bmatrix} \theta^i & \varphi^i & x_0^i & y_0^i & z_0^i \end{bmatrix}^T$ for i={1, ..., N} where N is the number of planes in the model of the 3D object being printed. All the state variables are defined in the global frame (Figure 2.5).

The representations chosen for the printer pose and landmark positions are commonly used angle and position in Cartesian coordinates, but the representation for planes needs to be discussed further.

The most straightforward way to represent a plane is the general form, which uses normal vector $\begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}^T$ and distance from plane to origin b_4 to represent a plane, i.e. the commonly used plane representation:

$$b_1 x + b_2 y + b_3 z + b_4 = 0$$

However, this representation should not be used in SLAM for mobile 3D printing. There are two reasons, discussed below.

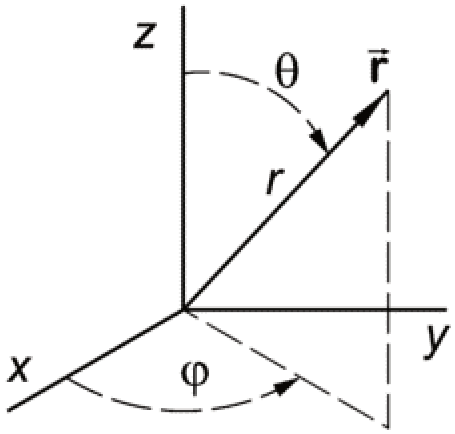

FIGURE 2.6: Spherical coordinate system for representing plane normals (Figure cited from citizendium [10]: spherical polar coordinates)

First, if the general form is used to represent planes, the normal vector $\begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix}^T$ has an implicit condition that these three parameters are normalized. Some steps of the procedure (e.g. output from a plane fitting function) always give three normalized parameters to indicate the plane direction. However, in the SLAM algorithm those parameters are likely to be denormalized because their estimates are treated independently. Note that this is also a problem for all other parameterizations that use a 3D Cartisian normal vector, listed in Table 1.1. By using extra constraints, we could ensure the unit length of the plane normal, or to avoid the need for extra constraints, plane directions can instead be represented in a spherical coordinate system (Figure 2.6).

Let θ be the angle between the normal and z-axis (polar angle), and let φ be the angle between x-axis and the projection of normal vector on XY plane (azimuthal angle), the normal can then be represented as:

$$\begin{bmatrix} \theta & \varphi \end{bmatrix}^T$$

Having the equation set below for normal vector representation, 'r' is always 1, so normal vectors are always normalized.

$$\begin{cases} b_1 = rsin\theta cos\varphi \\ b_2 = rsin\theta sin\varphi \\ b_3 = rcos\theta \end{cases}$$

Notice that the spherical coordinate system has a singularity at $\theta = 0$ or π, (e.g. top or bottom of a cube) where φ can be undefined and the system can suddenly experience an instantaneous change in φ value. For this reason, horizontal planes, while still added into the state vector, are not presently updated in the 3D scanner SLAM algorithm since their normal vectors are too close to the $\theta = 0$ sigularity.

Second, the commonly used general form of planes has an assumption that for an infinite plane, all the parameters are independent. However, in our work, since we are looking at only part of the plane, uncertainties added into those parameters make them become dependent. For example, around a measurement point far from the origin errors in b_1 could not be considered independent of errors in b_4; small differences in plane direction could in fact require large differences in b_4 to match the measurement.

Therefore, the centroid of the point cloud $\begin{bmatrix} x_0 & y_0 & z_0 \end{bmatrix}^T$ is introduced to provide information about both the position of the plane the along normal vector and the region of the planar patch in the infinite size plane.

Given $\begin{bmatrix} \theta^i & \varphi^i \end{bmatrix}^T$ provides plane direction and $\begin{bmatrix} x_0^i & y_0^i & z_0^i \end{bmatrix}^T$ provides plane position for planar patch i, the state variables for a planar patch can be defined as:

$$\begin{bmatrix} \theta^i & \varphi^i & x_0^i & y_0^i & z_0^i \end{bmatrix}^T$$

As the mobile 3D printing procedure is adding to the overall model segment by segment, the number of planes on the printed model is incrementing. The number of planes saved in the plane parameter partition of state vector, N, is increasing as well.

The state vector used throughout the whole algorithm is shown in figure 2.7. Among these states, the total station part of the SLAM updates printer pose and landmark position, values in the left bracket. Meanwhile, the 3D scanner part of the SLAM updates printer pose and plane parameters, the values in the right bracket.

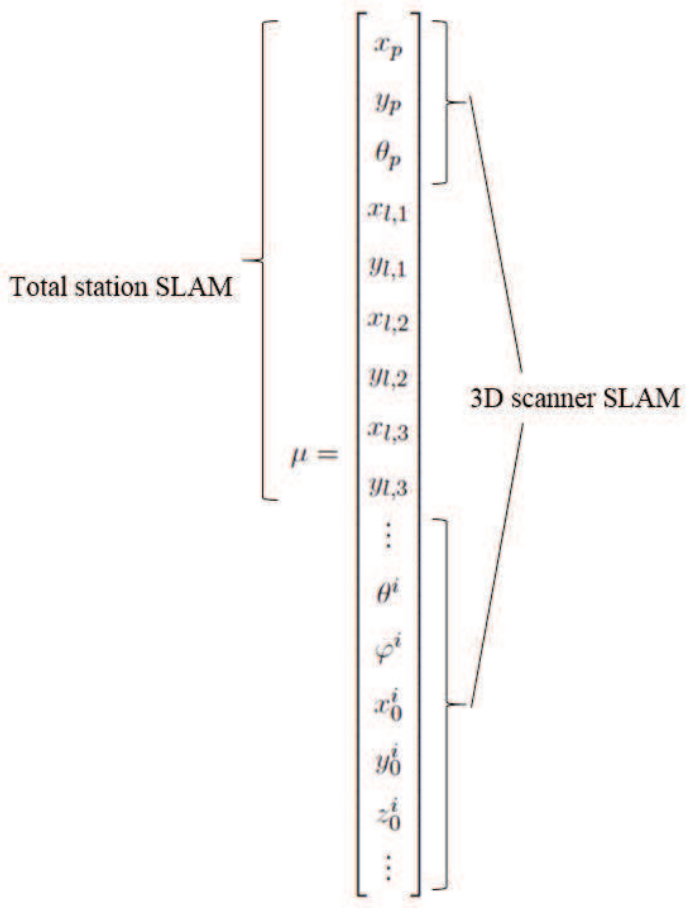

FIGURE 2.7: State vector of mobile 3D printing system. i={1,$\cdots$,N} where N is the number of planes in the model of the 3D object being printed

2.2.2 STL and plane parameters conversion

In the printing process, a conversion between STL file and state vector plane parameters is required. The conversion from planes to STL is used to generate an STL file for the segment to be printed next, and the conversion from STL to planes provides prediction of plane parameters to the SLAM algorithm in order to update the state vector after a new segement is printed.

Thus, it is required to calculate all the planes on a segment, given the original model and a segmenting plane. On the other hand, we must be able to get the STL file of the segment to be printed next.

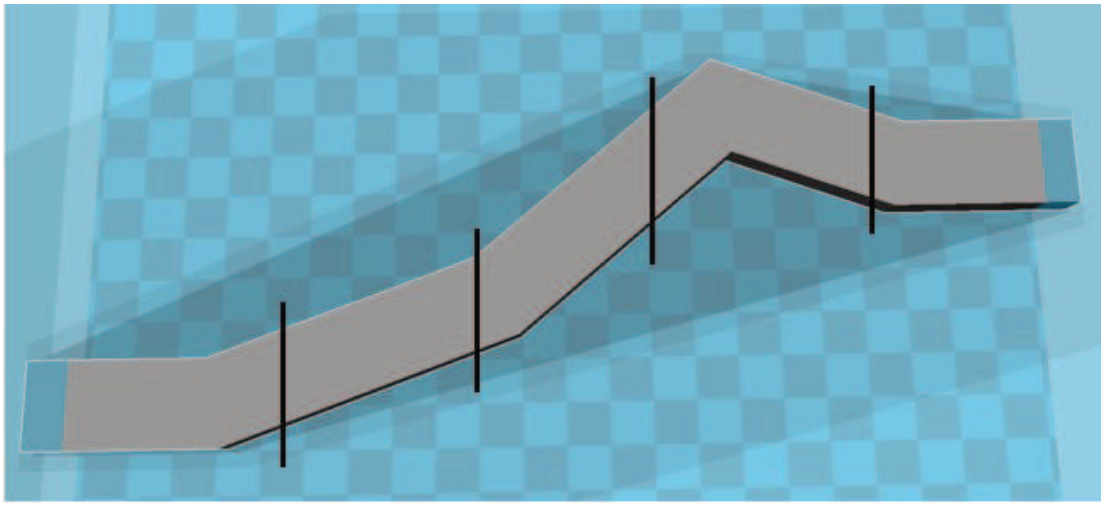

FIGURE 2.8: one dimensional model segmenting

We are currently segmenting the model in only one dimension (figure 2.8), along ${}^{G}x$-axis, since the model we design is short in ${}^{G}y$ direction and long in ${}^{G}x$ direction. The segmenting angle is 20 degrees to the ${}^{G}x$-axis in the ${}^{G}x$-^{G}z plane, which makes the segmenting surface to be a ramp. This angle is implemented to avoid the nozzle of the printer hitting the previous segment while printing the subsequent segment. An example segment, with segmenting plane visible, is shown in Figure 2.11.

2.2.2.1 STL to planes

A program is needed to compute all the planes on the model. Those planes are going to be used to represent the prediction of the model in the state vector for the SLAM algorithm. Since STL file is a format which saves the model in triangulated surfaces, we can read and convert it into vertices and normal vectors of those surfaces. The conversion from STL to infinite size planes is straightforward because we can define any plane in 3D space by having a normal vector and a point that lies on the plane.

However, throughout this work, the program is actually based on integral planar patches. Multiple contiguous triangles may be on the same planar patch (e.g. one rectangle planar patch is made of two triangles, one polygonal planar patch is made of multiple triangles). Having multiple triangles representing one single planar patch is redundant, so contiguous

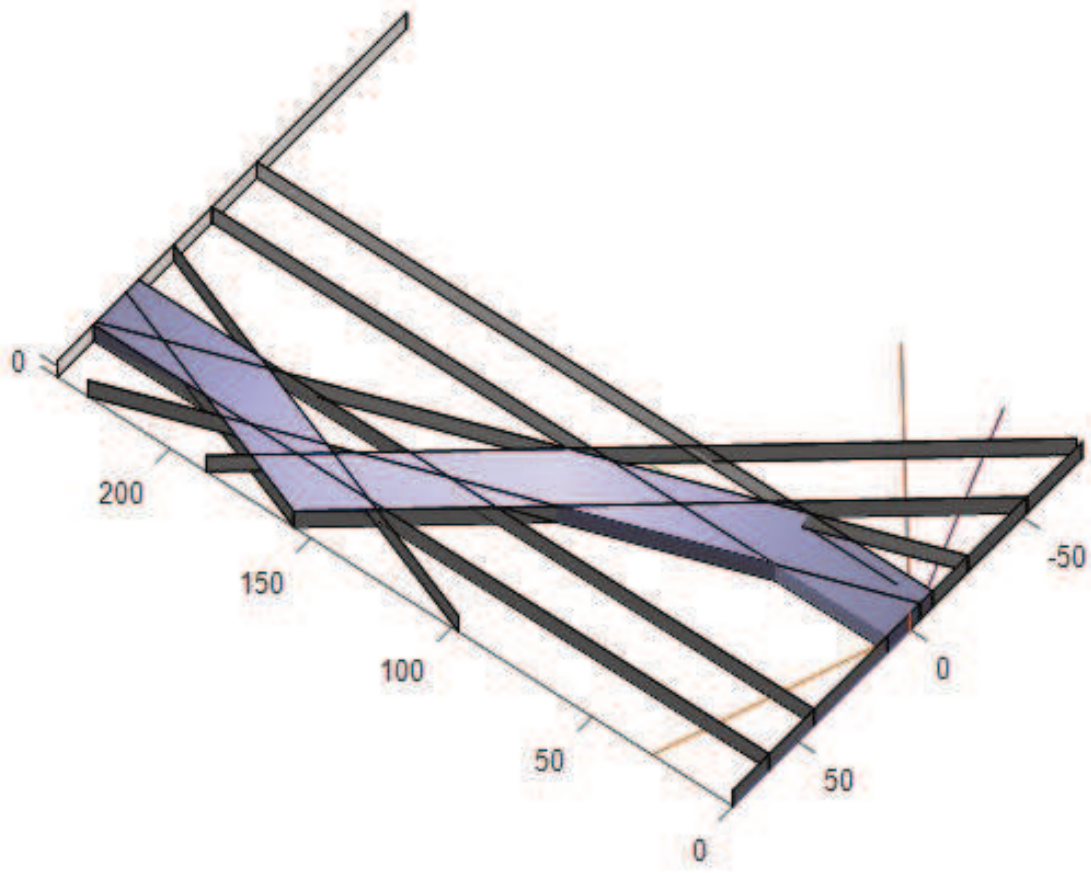

FIGURE 2.9: planes on the STL model

triangles on one planar patch (i.e. lie on the same plane, and share one or more vertices) are flagged for elimination until only one remains.

This conversion generates an $N \times 7$ matrix (N stands for the number of planes on the model, while 7 includes four plane parameters in general form and a vertex that lies on the plane in Cartesian coordinates).

Moreover, only knowing the plane parameters and a vertex on this plane is still not enough, since they only define an infinite plane without boundary (Figure 2.9), the limits of these planar patches must be known to determine which of them should be kept after segmenting. Therefore, the previously mentioned $N \times 7$ matrix should be further extended into a $N \times 9$ matrix, in which the column 5 to 6 store the left and right limits of the plane on ${}^{G}x$-axis (see figure 2.10). This information can be extracted from vertices saved in STL files.

8x10 double

	1	2	3	4	5	6	7	8	9
1	0	0	1	0	0	60.0000	0	-5	0
2	0	0	1	-7.3000	0	39.9421	0	5	7.3000
3	-1	0	0	0	0	0	0	5	0
4	-0.3420	0.9397	0	8.9823	40	60.0000	100	26.8382	0
5	0	-1	0	-5	0	41.7633	0	-5	0
6	0	1	0	-5	0	40	40	5	0
7	0.3420	-0.9397	0	-18.9823	41.7633	60.0000	41.7633	-5	0
8	0.3420	0	0.9397	-20.5200	39.9421	60.0000	60.0000	12.2781	0

FIGURE 2.10: the matrix saves plane parameters (column 1-4), boundaries of the planes (column 5-6) and centroids of the planar patches (column 7-9)

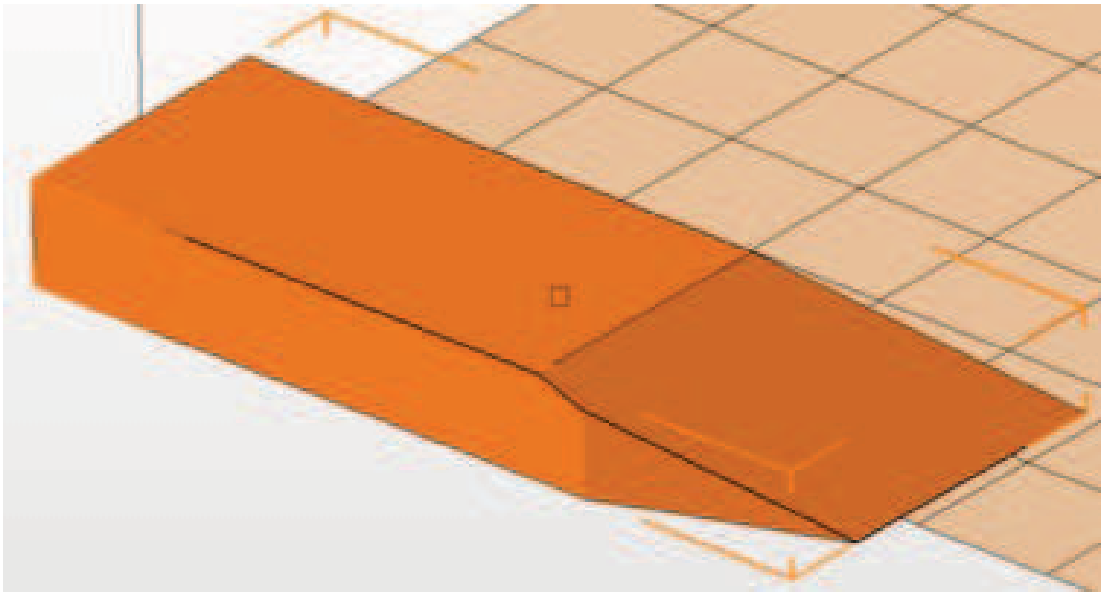

FIGURE 2.11: The model is segmented by a plane 20 degrees to the x-axis, the result is a 60mm length segment with a ramp

By interfacing the segmenting plane with the top and bottom planes, the boundaries of the segment can be computed. Then, boundaries of all the planar patches in the model are checked. If the planar patch overlaps with the boundaries of a segment, it will be kept and the boundaries will be changed based on the intersection, otherwise it will be discarded. In this way, all the planes on the segment will be computed for initialization of the state vector.

Take the model in Figure 2.11 for example, it has 8 planes in total, so the output of the STL to planes function is a 8×9 matrix (Figure 2.10). In this matrix, column 1 to 4 save the general form plane parameters, column 5 to 6 save the left and right boundaries of planar patches on ^{G}x-axis, and column 7 to 9 save one arbitrary vertex on each planar patch.

As mentioned previously in Section 2.2, directions of planes are saved in spherical coordinate system in the state vector, so if we want to save those converted planes into the state vector, the output plane direction $\begin{bmatrix} b_1^i & b_2^i & b_3^i \end{bmatrix}$ (column 1-3 of row i) must be transformed into spherical coordinate at first.

$$\begin{cases} \theta^i = \arccos \frac{b_3^i}{\sqrt{(b_1^i)^2+(b_2^i)^2+(b_3^i)^2}} \\ \varphi^i = \arctan \frac{b_2^i}{b_1^i} \end{cases}$$

Having the directions of planar patches in spherical coordinate system, combined with vertices on the planes, plane parameters $\begin{bmatrix} \theta^i & \varphi^i & x_0^i & y_0^i & z_0^i \end{bmatrix}^T$ can be saved into the state vector.

2.2.2.2 Planes to STL

Having a program that converts from STL files to planes is not enough. It is also required to convert inversely, knowing all the plane parameters on the new segment, to generate the STL file for new printing. This step is done right after the SLAM algorithm gets an updated estimate of the ramp plane for segmenting, and right before the printing procedure.

This program always reads an STL file of the full CAD model and segments on that STL file by the current segmenting plane (i.e. the ramp plane). A new STL file of the next segment to be printed is then generated. The reason for starting with an STL file of the full CAD model is because a STL file of full CAD model is always able to compensate for previous printing errors, while using an STL file of remaining CAD model from last segmenting could run into a problem where part of the useful information has already been discarded.

When writing the STL file, planes are at first saved into a 'struct' type then converted into STL file. The 'struct' contains 2 fields: vertices and faces. The vertices provide information about all the triangulated surfaces of the model, and the faces indicate the order of the vertices on the triangulated surfaces.

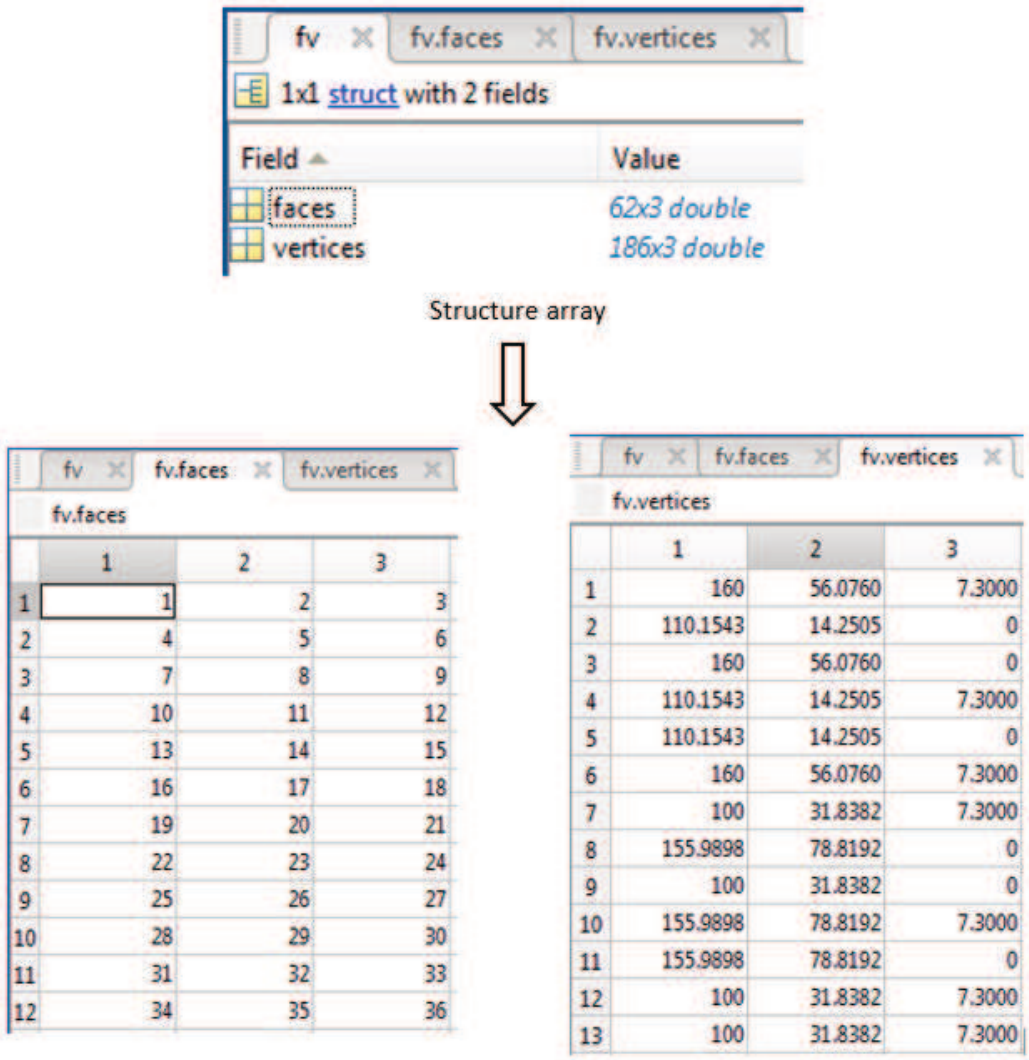

FIGURE 2.12: planes to STL

In the process of segmenting, the edges of each polygon in the CAD should be checked to see if they are to be included in the next segment. Suppose the equation of the segmenting plane is

$$b_1 x + b_2 y + b_3 z + b_4 = 0$$

and the position of a vertex is

$$\begin{bmatrix} x_v^i & y_v^i & z_v^i \end{bmatrix}$$

A pair of vertices comprise an edge.

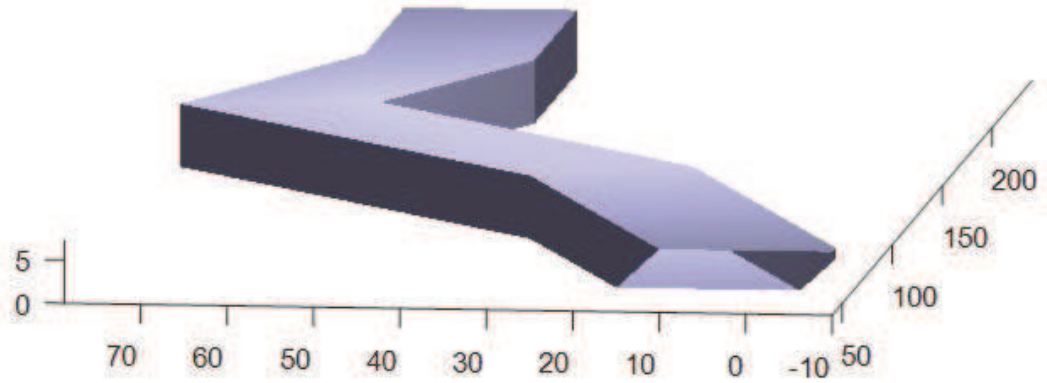

FIGURE 2.13: generated model without closing the segmenting plane

Then if both vertices of the edge satisfy $b_1x_v^i + b_2y_v^i + b_3z_v^i + b_4 < 0$, this edge is under the segmenting plane so it is part of a prior segment and should be discarded.

If both vertices satisfy $b_1x_v^i + b_2y_v^i + b_3z_v^i + b_4 > 0$, this edge is above the segmenting plane so it is part of a new segment and should be saved.

If one vertex satisfies $b_1x_v^i + b_2y_v^i + b_3z_v^i + b_4 < 0$ while the other vertex satisfies $b_1x_v^i + b_2y_v^i + b_3z_v^i + b_4 > 0$, then the edge is crossing the segmenting plane so a new vertex at the intersection should be generated.

So far, an STL file with an open face along the segmenting plane has be generated (Figure 2.13), the next step is to close this face.

It is easy to find that the outline of the face is actually composed of all the newly generated vertices from the previous step. If we sequentially connect all those vertices to get a polygon, we can use this polygon to close the model.

The polygon used to close the open part of the model after segmenting can have two different types of shapes, either convex or concave (Figure 2.14).

If it is convex, it is much easier to deal with, because by using convex hull function we can get the order of vertices on the outline easily. However, if it is concave, convex hull function will not give us the order of the point in its interior, so triangulating a concave shape is not trivial.

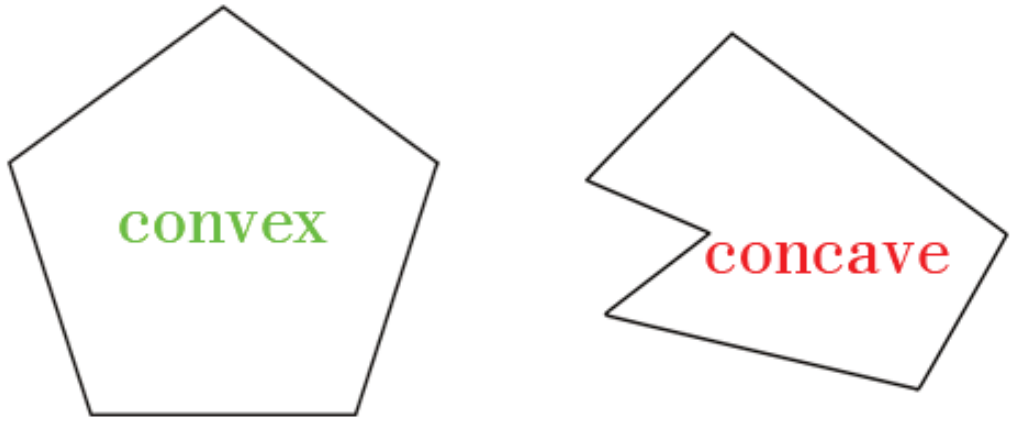

FIGURE 2.14: concave and convex polygons

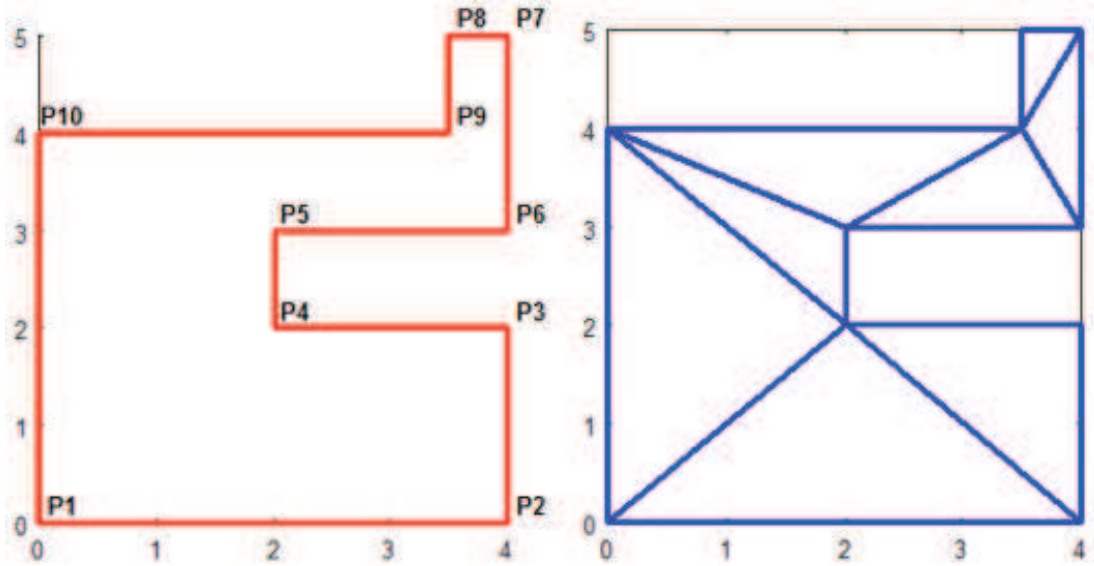

FIGURE 2.15: Delaunay triangulation

In order to solve this problem, Delaunay triangulation is used to triangulate complex concave cases given all the vertices in order. Figure 2.15 shows one example of using Delaunay triangulation on a concave polygon.

Figure 2.16 shows how the concave surface is generated for the new ramp plane after segmenting.

Delaunay triangulation can triangulate a concave polygon easily, but it still requires the sequential order of the vertices on the outline of polygon as input. For a concave polygon,

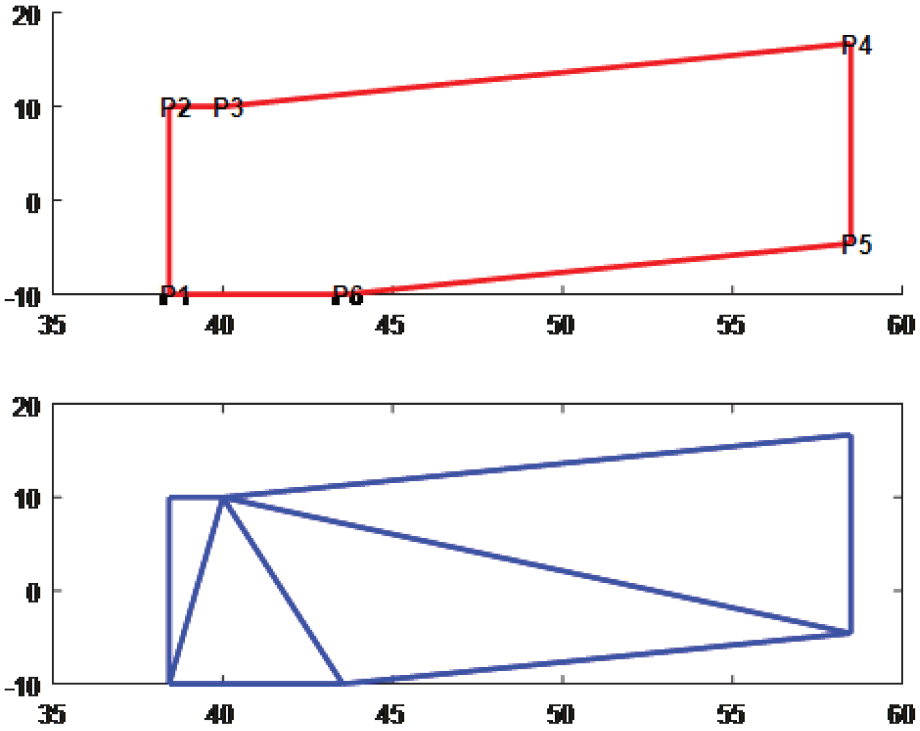

FIGURE 2.16: concave polygon is generated for closing the model

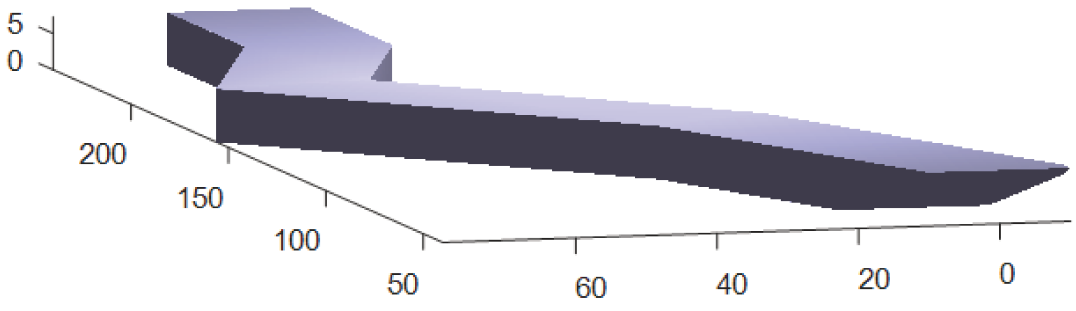

FIGURE 2.17: closed model

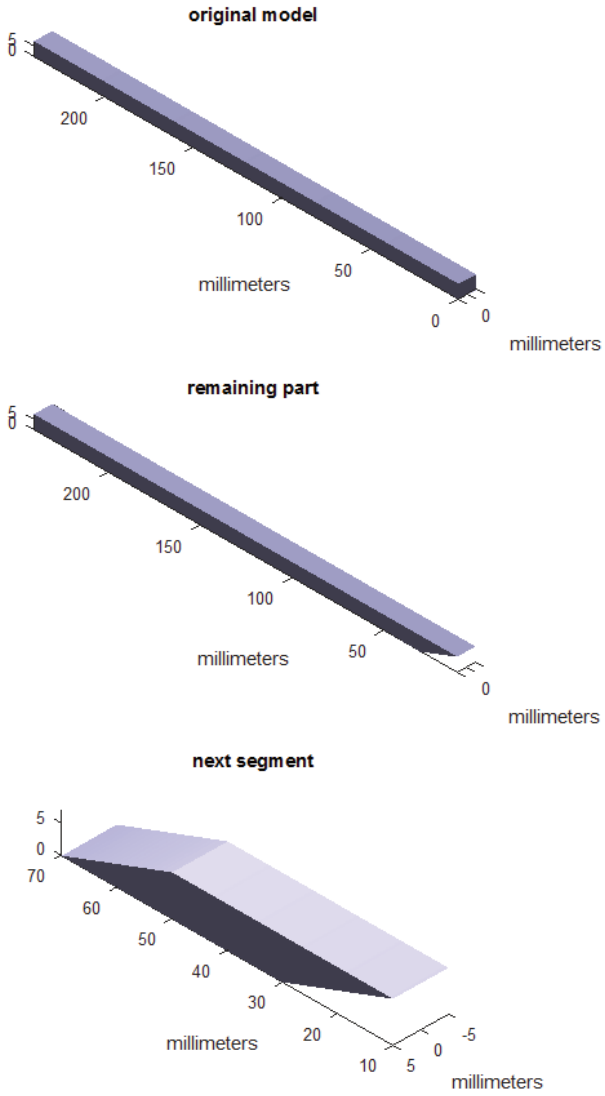

FIGURE 2.18: The original CAD model (top), the remaining portion of the model to be built (portion above current segmenting plane) (center), the next segment to be printed (cut with another segmenting plane) (bottom)

there are vertices that lie within, rather than on, the convex hull. Those vertices must be connected to the other vertices using information encoded in the STL file.

The open STL model can be closed by the generated polygon (Figure 2.17). The other end of the model should be segmented in a similar way, but using a parallel plane L(mm) away on the x-axis, where L is the segmenting length we set. Moreover conversely, planes below the segmenting plane are kept and above the segmenting plane are discarded. Finally an STL file of the next segment for printing is generated (figure 2.18).

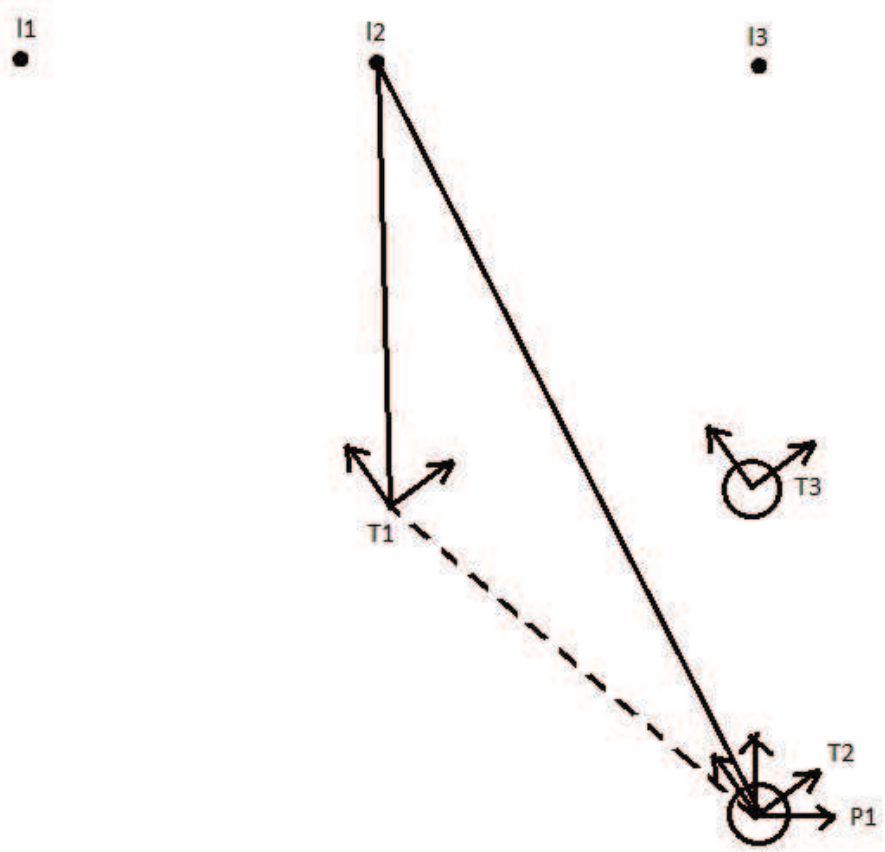

FIGURE 2.19: Geometry of total station calibration procedure. Circles represent lens target holders. P1 is the printer frame at position1. Ti are the total station frames at position i

2.2.3 Sensor calibration

2.2.3.1 Total station calibration

In order to get total station position and orientation in the printer frame, we use a screw with through-hole to fix the total station on top of the printer, enabling the leveling laser light of the total station to project onto the printing surface below. Moreover, a lens with a target in the center is used, which helps the 3D scanner locate where the laser beam is, since the laser beam on the printing surface cannot be captured directly by the 3D scanner.

The calibration steps are as following:

1. Put the printer at one pose, defined as P1. Print two lens holders, at ${}^{P1}(0,0)$ and ${}^{P1}(0,Y)$

2. While the printer is at P1, the total station pose is defined as T1. Use the total station to do a set of measurements to landmarks defined as ${}^{T1}l_i$.

3. Then, move the printer to a new pose such that the laser beam is projected on ${}^{P1}(0,0)$. Do a set of total station measurements to landmarks defined as${}^{T2}l_i$.

4. After, move the printer to another new pose such that the laser beam is projected on ${}^{P1}(0,Y)$. Again, do a set of total station measurements to landmarks defined as ${}^{T3}l_i$.

There are three unknowns relating T1 to P1 that need to be calculated: offset between total station and printer position on x-axis ${}^{P1}x_{T1}$, on y-axis ${}^{P1}y_{T1}$ and angle offset ${}^{P1}\theta_{T1}$.

A geometric relationship can be represented:

$$\begin{cases} {}^{T2}l_i = \begin{bmatrix} {}^{T2}x_{T3} \\ {}^{T2}y_{T3} \end{bmatrix} + \begin{bmatrix} \cos{}^{T2}\theta_{T3} & -\sin{}^{T2}\theta_{T3} \\ \sin{}^{T2}\theta_{T3} & \cos{}^{T2}\theta_{T3} \end{bmatrix} {}^{T3}l_i & \text{(2.1a)} \\ \begin{bmatrix} {}^{T2}x_{T3} \\ {}^{T2}y_{T3} \end{bmatrix} = \begin{bmatrix} \cos{}^{T2}\theta_{P1} & -\sin{}^{T2}\theta_{P1} \\ \sin{}^{T2}\theta_{P1} & \cos{}^{T2}\theta_{P1} \end{bmatrix} \begin{bmatrix} {}^{P1}x_{T3} \\ {}^{P1}y_{T3} \end{bmatrix} & \text{(2.1b)} \\ {}^{T1}l_i = \begin{bmatrix} {}^{T1}x_{T2} \\ {}^{T1}y_{T2} \end{bmatrix} + \begin{bmatrix} \cos{}^{T1}\theta_{T2} & -\sin{}^{T1}\theta_{T2} \\ \sin{}^{T1}\theta_{T2} & \cos{}^{T1}\theta_{T2} \end{bmatrix} {}^{T2}l_i & \text{(2.1c)} \end{cases}$$

Substitute equation (b) into equation (a), ${}^{T2}l_i$ and ${}^{T3}l_i$ are respectively the total station measurement to the three landmarks at position T2 and T3, so they are known. ${}^{P1}x_{T3}$ and ${}^{P1}y_{T3}$ are the position offset between two lens holders, we define this offset to be (0, Y) by ourselves, so it is also known. Therefore, there are only 2 unknowns remain (${}^{T2}\theta_{P1}$ and ${}^{T2}\theta_{T3}$) with 6 equations (2 equations for x and y of each set of landmark measurements), so we can solve all the unknowns.

ANGULAR MEASUREMENT		
Accuracy [1] Hz and V	Absolute, continuous, diametrical	1" (0.3 mgon), 2" (0.6 mgon), 3" (1 mgon), 5" (1.5 mgon)
DISTANCE MEASUREMENT		
Range [2]	Prism (GPR1, GPH1P) [3] Non-Prism / Any surface [4]	1.5m to 3500m R500: 1.5m to >500m, R1000: 1.5m to >1000m
Accuracy / Measurement time	Single (prism) [2,5] Single (any surface) [2,4,5,6]	1mm + 1.5ppm / typically 2.4s 2mm + 2ppm / typically 3s

FIGURE 2.20: bearing and angle error of total station in the datasheet

Similarly, equation (c) has 6 equations with 3 unknowns (${}^{T1}x_{T2}$, ${}^{T1}y_{T2}$ and ${}^{T1}\theta_{T2}$), so all the unknowns can be solved.

Since P1 and T2 are at the same position, there is

$$ {}^{T1}x_{P1} = {}^{T1}\ x_{T2} $$

$$ {}^{T1}y_{P1} = {}^{T1}\ y_{T2} $$

In addition, there is

$$ {}^{T1}\theta_{P1} = {}^{T1}\ \theta_{T2} + {}^{T2}\ \theta_{P1} $$

Finally, the transformation matrix between total station and printer is:

$$ {}^{T1}T_{P1} = \begin{bmatrix} \cos {}^{T1}\theta_{P1} & -\sin {}^{T1}\theta_{P1} & {}^{T1}x_{P1} \\ \sin {}^{T1}\theta_{P1} & \cos {}^{T1}\theta_{P1} & {}^{T1}y_{P1} \\ 0 & 0 & 1 \end{bmatrix} $$

The laser total station has a very high accuracy if measured landmarks are in a leveled condition. Figure 2.20 shows the errors of angular and distance measurement in the total station datasheet. However, in this work, the total station is mounted on top of the 3D printer in order to achieve automation and cannot be calibrated after each printer motion. Due to a non-level printing surface, the total station is always working in an uncalibrated condition. This uncalibrated condition increases the total station measurement error.

For example, if the angle offset due to tilt is 1 degree, since the height of the 3D printer is about 70 cm, the offset of relative position would be

$$\sin 1^\circ \times 700mm = 12.2mm$$

This is much larger compare to the accuracy of total station itself.

Therefore, the total station measurement error (specifically for the range error) should be set to a number much larger than the measurement error given by the datasheet. Here we suppose the total station measurement error covariance matrix is:

$$Q_T = \begin{bmatrix} 10 & 0 \\ 0 & 4.7124e-06 \end{bmatrix}$$

I.e. 10mm range error and 4.7124e-06 rad bearing error.

2.2.3.2 3D scanner calibration

The first step of 3D scanner calibration is to print a small cuboid oriented 45 degrees around z-axis relative to the printer at the center of the print bed. Next, the 3D scanner gets a 3D point cloud of the cuboid. The scanning result is saved in a PLY file. (PLY file is a polygon file format used to store 3D data from the 3D scanner)

The Computer Vision System Toolbox in Matlab is used to process the point cloud. An ROI region is set and the point cloud outside this region is discarded, as the only interesting part to us is the cuboid. The point cloud result after cropping is shown in figure 2.21. Then the point cloud is downsampled in order to reduce the processing time. By using a plane fitting function (based on M-estimator SAC (MSAC) algorithm, a variant of the Random sample consensus (RANSAC) algorithm) three outputs are obtained, the plane parameters, the linear indices of inlier points and the linear indices of outlier points.

Running this plane fitting function repeatedly, both the top (figure 2.23) and side plane (figure 2.24) of the cuboid can be isolated.

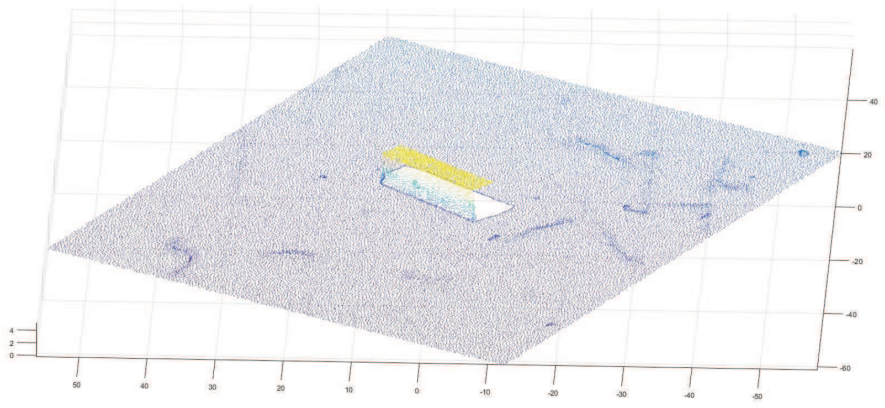

FIGURE 2.21: point cloud of the segment used for calibration

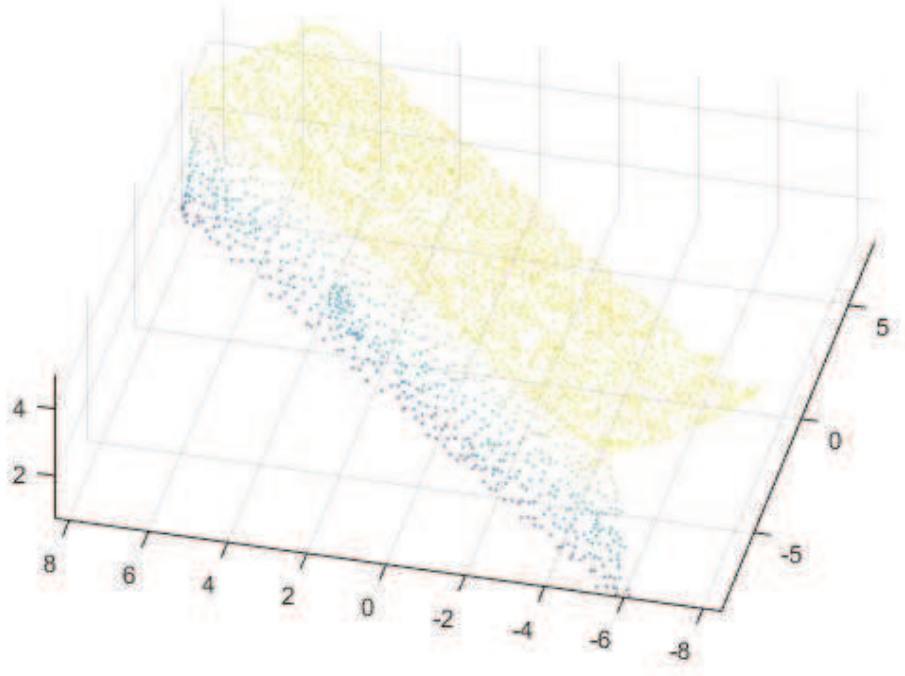

FIGURE 2.22: the point cloud without bottom plane

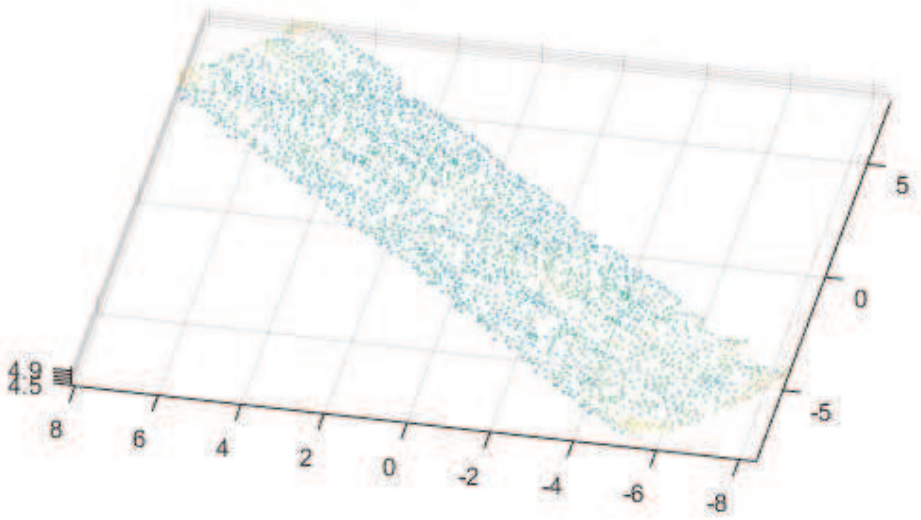

FIGURE 2.23: the point cloud of the top plane

The average of all the points in the top planar patch estimates the centroid of the rectangle itself, and then the coordinate of bottom center can be calculated by projecting the top center along normal vector to intersect with base plane. The bottom center is the origin of 3D printer in the 3D scanner frame.

Finally, the direction of the 3D scanner needs to be determined. The intersection of side plane and bottom plane can indicate the direction of the segment. The structure of the 3D printer is the delta shape equilateral (i.e. triangle), and the 3D scanner is installed at the center of one delta edge, so the angle between the scanner and printer should be 120 degree as designed. However, due to possible angular error between 3D scanner and 3D printer, there is an offset. Instead, the measured direction of the side plane patch is used to cancel out any angular error of the mounted 3D scanner.

From previous steps, both the origin and the direction can be calculated and used to build the transformation matrix, which is able to transform scan results from scanner frame to

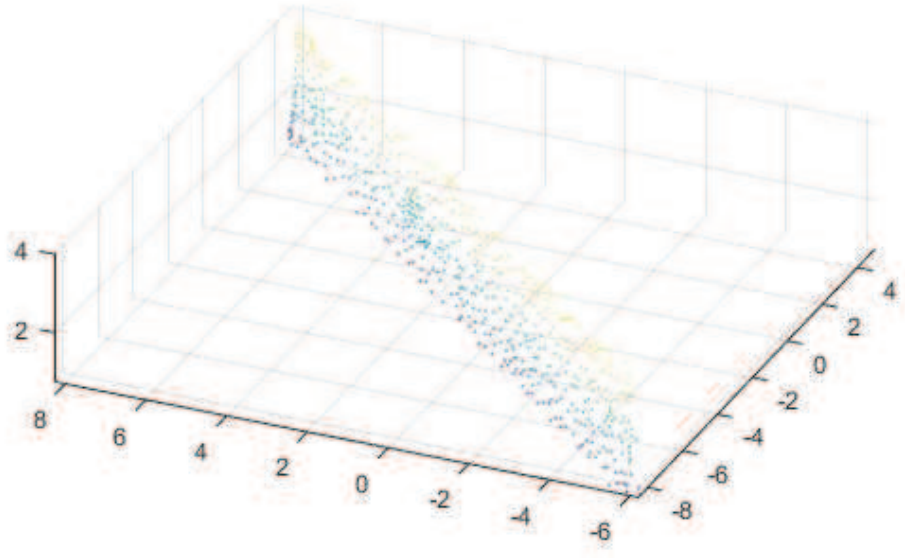

FIGURE 2.24: the point cloud of the side plane

printer frame.

Chapter 3

MOBILE 3D PRINTING PROCEDURE

3.1 Procedure introduction

In this section, the procedure of the whole printing process will be introduced briefly with pseudo code (Algorithm 1) and a printing procedure flowchart (Figure 3.2). Further detail will be explained in the following sections for each individual step.

Algorithm 1 Mobile 3D printing

```
Landmark initialization
1st print
while printing is not finished do
    3D scanner (1)
    Move the printer
    Total station
    3D scanner (2)
    Next print
end while
return
```

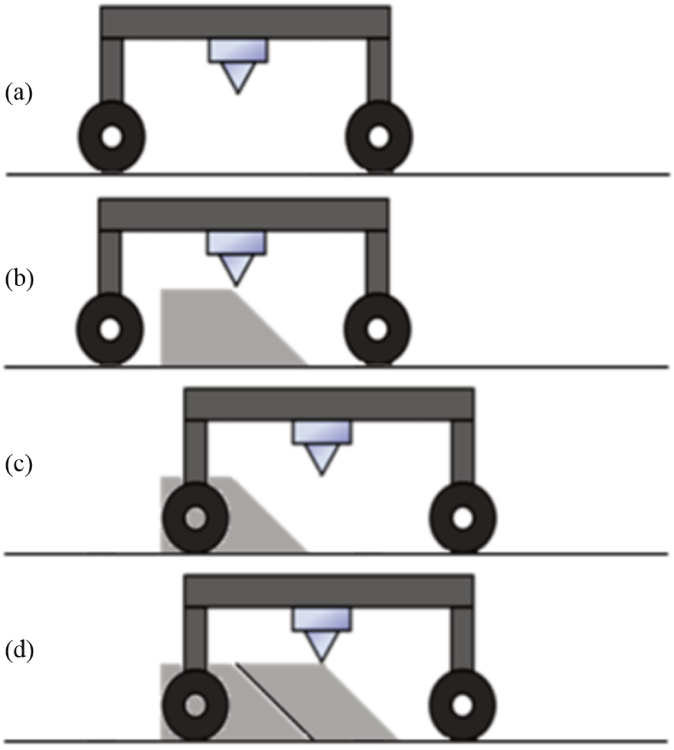

FIGURE 3.1: Illustration of mobile 3D printing process. (a) t=0, 1; (b) t=2, 3; (c) t=4, 5, 6; (d) t=2, 3.

The global frame can be defined anywhere, but to make geometry easy, we define it with the same origin as the initial printer frame, rotated by $-\frac{\pi}{2}$.

At first, a set of total station measurements is taken relative to the landmarks. Then, if the model exceeds the printing area, it needs to be segmented in order to obtain a first small segment to print. During the segmenting process, the algorithm not only generates a new STL file for that small segment (planes to STL, section 2.2.2.2), but also computes the plane parameters on that small segment (STL to planes, section 2.2.2.1) as the prediction of planes on the printed model. After that, the 3D scanner scans the segment just printed to correct the previous predictions and update plane patch location estimates.

Next, the printer is moved to a new position close to the center of the next segment and with an appropriate view for scanning. The odometry data gives an estimation of what the new pose of the printer is. Then, once again, the total station measurement is run followed by a 3D scanner measurement. The EKF-SLAM algorithms of these two sensors can get an

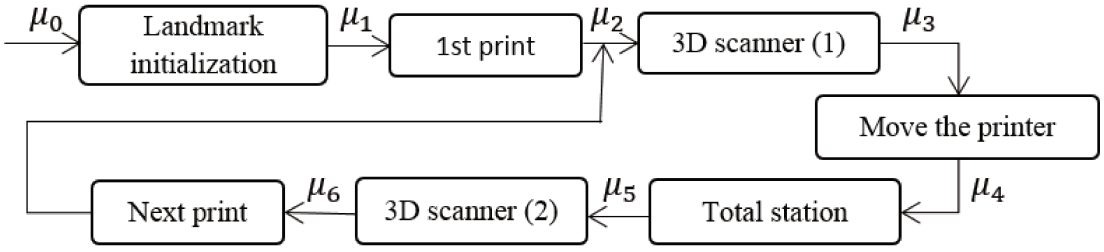

FIGURE 3.2: mobile 3D printing procedure

updated estimation of printer pose, landmark positions and model parameters, which helps the next generated segment properly connect with the previous segment. After printing, the 3D scanner scans the printed model again, as before. The previous steps repeat until the whole model is finished printing.

The mobile 3D printing procedure can be divided into smaller blocks, as shown in Figure 3.2. The changes to the state vector and its covariance at each block of the procedure flowchart are described below:

1. Before printing, the initial pose of the printer is defined as rotated $\frac{\pi}{2}$ to the origin of the global frame. At this moment, the state vector only contains information about the printer pose (position and orientation).

$$\mu_0 = \begin{bmatrix} x_p \\ y_p \\ \theta_p \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{\pi}{2} \end{bmatrix}$$

   The covariance matrix is a matrix of all zeros.

$$\Sigma_0 = \mathbf{0}$$

2. Landmark initialization

   After the initial printer pose, the total station is used to do a set of measurements to three landmarks. Then, the landmark positions, a 6×1 vector, are added into the state vector. With this first total station measurement, printer pose still cannot be

updated since two sets of total station measurements are required for the total station SLAM. Therefore, it only adds the landmark positions in global frame into the state vector.

$$\mu_1 = \begin{bmatrix} x_p \\ y_p \\ \theta_p \\ x_{l,1} \\ y_{l,1} \\ x_{l,2} \\ y_{l,2} \\ x_{l,3} \\ y_{l,3} \end{bmatrix}$$

The covariance matrix is expanded as well. The landmark part of the covariance, Σ_l, is initialized with a small error

$$\Sigma_l = 0.1 I_{6\times 6}$$

(0.1mm for both x and y). In future, it is recommended that the landmark covariance be initialized based on the total station range and bearing errors.

3. 1st print

   Next, the printer prints the first segment of the model. After printing, predictions of all the planes on the first segment (STL to planes in Section 2.2.2.1) are added into the state vector. The size of the state vector is expanded to $(9 + 5 \times N)$, where N is

the number of planes currently on the model.

$$\mu_2 = \begin{bmatrix} x_p \\ y_p \\ \theta_p \\ x_{l,1} \\ y_{l,1} \\ x_{l,2} \\ y_{l,2} \\ x_{l,3} \\ y_{l,3} \\ \vdots \\ \beta^i \\ \varphi^i \\ x_0^i \\ y_0^i \\ z_0^i \\ \vdots \end{bmatrix}$$

The covariance matrix is expanded as well to $(9 + 5 \times N) \times (9 + 5 \times N)$. Each plane parameter part of the covariance matrix Σ_p^i is initialized by the printing error. The error of the printer can be estimated by printing one model with simple geometry and then using the 3D scanner and its inspection software to check the deviation value (Appendix A). The printing error, and thus planar patch covariance, is:

$$\Sigma_p^i = \delta = \begin{bmatrix} 0.015 & 0 & 0 & 0 & 0 \\ 0 & 0.015 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0.2 \end{bmatrix}$$

I.e. 0.015 rad (0.8594 degree) for angular error and 0.2 mm for position error. At this moment, the covariance matrix is:

$$\Sigma_2 = \left[\begin{array}{c:c:ccc} \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & \cdots \\ \hdashline \mathbf{0} & \Sigma_l & \cdots & \mathbf{0} & \cdots \\ \hdashline \vdots & \vdots & \ddots & & \\ \mathbf{0} & \mathbf{0} & & \Sigma_p^i & \\ \vdots & \vdots & & & \ddots \end{array}\right] \tag{3.1}$$

4. 3D scanner (1)

   In the first 3D scanning step, 3D scanner SLAM is run, so planes on the model, the printer pose and the corresponding covariance are updated simultaneously.

5. Move the printer

   The printer is relocated to a position convenient to print the next segment. Printer pose estimate is updated with odometry data, and the printer pose estimation covariance is updated as well.

6. Total station

   The total station is run again to get another set of measurements. Total station SLAM is run, so printer pose, landmark positions and the corresponding covariance is updated simultaneously.

7. 3D scanner (2)

   3D scanner SLAM is run again. Planes on the model, printer pose and the corresponding covariance is updated.

8. Next print

   The result of the previous two SLAM algorithms provides an accurate estimate of the printer pose and plane parameters of the model. Therefore, the next segment can be printed, connected to the previous one. Right after printing, a prediction of planes on the model is made before the measurement.

Repeat step 4 to step 8 until the whole model is printed.

3.2 Landmark initialization

The total station (1) step only does measurement and initialization of the landmark positions in the global frame. The printer pose stays the same as it has not been moved.

The transformation matrix between the total station frame and the 3D printer frame is already known (mentioned in the total station calibration step 2.2.3.1), so the measured landmark positions can be converted into the printer frame, then converted into the global frame. The landmarks in the global frame are added into the state vector, so the state vector is updated from μ_0 to μ_1.

Figure 3.3 shows a plot of landmarks and printer pose in the global frame. The blue dots represent landmark positions in the global frame, and the green circle represents the printer pose, which has not been moved yet.

3.3 1st print

A large model is segmented into a smaller segment to be printed. The segmenting plane is selected such that the size of the segment is a fixed length L, (in our case 60mm) along the ${}^{G}x$-direction. This step is a plane to STL conversion as described in section 2.2.2.2, knowing the CAD model and segmenting plane, the algorithm will generate a new STL file for the first print, and at the same time, planes on the printed segment is added into the state vector. The state vector is updated from μ_1 to μ_2.

3.4 3D scanner (1)

The 3D scanner is used to perform feature-based SLAM, with planar patches as the features.

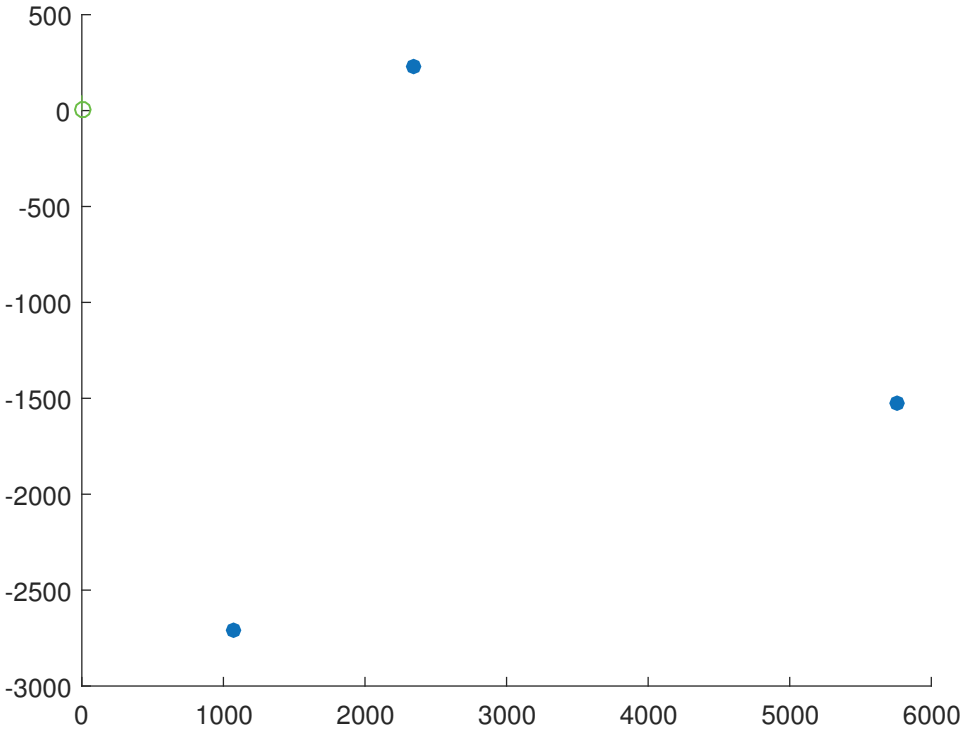

FIGURE 3.3: original printer pose (green circle) and landmark positions (blue dots) from the first set of total station measurements (in mm)

3.4.1 Correspondence

A crucial part of the feature-based SLAM is to find the correspondence between predicted features and observed features. In our work, since planes are used as features in the 3D scanner part of SLAM, the correspondence between predicted planes in the global frame (G represents parameters in the global frame) ${}^{G}\bar{\mu}_{b^i}$ and measured planes in the printer local frame (P represents parameters in the printer frame) ${}^{P}b^j$ should be determined.

Measured planes are first transformed from local frame to global frame in order to compare with the predicted planes within the same coordinate system.

Note that for this step, general form is used for the plane patches because this is the form that is output by the plane fitting function and the disadvantages of general form are not applicable if the parameters are not being modified, as is the case in this step.

The transformation matrix of planes from global to local is discussed, since it is more straightforward, then the inverse of this matrix is the transformation from local to global. This transformation matrix is combined from two parts. One part is a rotation of the normal vector $\left[{}^{G}b_1^j, {}^{G}b_2^j, {}^{G}b_3^j\right]^T$ to printer local frame. The rotation matrix is shown as below:

$$\begin{bmatrix} {}^{P}b_1^j \\ {}^{P}b_2^j \\ {}^{P}b_3^j \end{bmatrix} = \begin{bmatrix} \cos\theta_p & -\sin\theta_p & 0 \\ \sin\theta_p & \cos\theta_p & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{G}b_1^j \\ {}^{G}b_2^j \\ {}^{G}b_3^j \end{bmatrix}$$

The other part is a translation with regard to the position of printer, the equation is

$${}^{P}b_4^j = {}^{G}\, b_1^j x_p + {}^{G}\, b_2^j y_p + {}^{G}\, b_4^j$$

Combining these two equations and inverting, the transformation matrix for planes from local to global is:

$$\begin{bmatrix} {}^{G}b_1^j \\ {}^{G}b_2^j \\ {}^{G}b_3^j \\ {}^{G}b_4^j \end{bmatrix} = inv \left(\begin{bmatrix} \cos\theta_p & -\sin\theta_p & 0 & 0 \\ \sin\theta_p & \cos\theta_p & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_p & y_p & 0 & 1 \end{bmatrix} \right) \begin{bmatrix} {}^{P}b_1^j \\ {}^{P}b_2^j \\ {}^{P}b_3^j \\ {}^{P}b_4^j \end{bmatrix}$$

${}^{G}\bar{\mu}_{b^i}$ are simply each plane i from the state vector converted to general form. Now, both the prediction ${}^{G}\bar{\mu}_{b^i}$ and the measurement ${}^{G}b^j$ are in the global frame. Next, correspondence that gives a value of i for each j (i.e. a planar patch in the state vector corresponding to each measurement) is determined by the directions and the centroid positions of planar patches. At first, the algorithm searches a closest match in terms of plane direction of ${}^{G}\bar{\mu}_{b^i}$ for each ${}^{G}b^j$ (the number of measured planes should always be less than predicted planes because some parts of the model are occluded). However, for the case that multiple predicted planes ${}^{G}\bar{\mu}_{b^i}$ have very similar directions, there can be confusion while matching, so their centroid positions are also compared to determine the correspondence.

Meanwhile, if there are multiple measured planes ${}^{G}b^j$ having very similar directions, it is

possibly due to printing error. It is possible that at the interface of two segments, one side plane is detected to be two separated patches. In this case, a ${}^{G}\bar{\mu}_{b^i}$ will be assigned correspondence to multiple ${}^{G}b^j$. If this happens, the tolerance of the plane fitting function for the measurement is relaxed and the plane fitting function is run again, until the number of planes detected decreases and this correspondence error is eliminated.

Additionally, the plane fitting function assigns a direction of the normal vector (i.e. inside or outside) randomly to the detected planes from point cloud. It gives either inside or outside randomly. Meanwhile, the inside and outside direction of predicted planes are specified in STL file, meaning normal vectors that actually correspond could be pointed in opposite directions.

In order to determine the directions of measured planes, the algorithm unifies the direction of top, bottom and ramp planes to positive direction of z-axis. And for the side planes, the SLAM algorithm checks the relative position of side planes and top planes in the y direction, so that whether they are facing towards positive or negative ^{G}y direction can be determined.

3.4.2 3D scanner SLAM algorithm

After determining the correspondence, the next step is to run the 3D scanner SLAM in order to further correct the printer pose and meanwhile update the planes on the printed model, based on the 3D scanner measurement (Algorithm 2). Each m^i refers to a planar patch isolated from the 3D scanner measurement point cloud, with correspondence established to a particular planar patch i in the state vector.

At the start of this step, the state vector μ_2 contains the original printer pose, the landmark positions measured in "total station (1)". In addition, we have the predicted model planes as described in "1st print". Due to the inclusion of these newly predicted elements, we label the intermediate state vector at this stage $\bar{\mu}$, and the covariance $\bar{\Sigma}$.

Algorithm 2 3D scanner SLAM algorithm (μ_2, Σ_2, m^i)

$\bar{\mu} = \mu_2$

$\bar{\Sigma} = \Sigma_2$

for all m^i **do**

$$\hat{m}_i = \begin{bmatrix} \hat{\theta}^i \\ \hat{\varphi}^i \\ \hat{x}_0^i \\ \hat{y}_0^i \\ \hat{z}_0^i \end{bmatrix} = \begin{bmatrix} \bar{\mu}_{\theta^i} \\ \bar{\mu}_{\varphi^i} - \bar{\mu}_{\theta_p} \\ \sin\bar{\mu}_{\theta_p}\left(\bar{\mu}_{x_0^i} - \bar{\mu}_{x_p}\right) - \cos\bar{\mu}_{\theta_p}\left(\bar{\mu}_{y_0^i} - \bar{\mu}_{y_p}\right) \\ \cos\bar{\mu}_{\theta_p}\left(\bar{\mu}_{x_0^i} - \bar{\mu}_{x_p}\right) + \sin\bar{\mu}_{\theta_p}\left(\bar{\mu}_{y_0^i} - \bar{\mu}_{y_p}\right) \\ \bar{\mu}_{z_0^i} \end{bmatrix}$$

$$F_{x,i} = \begin{bmatrix} 1 & 0 & 0 & \underbrace{0 & \cdots & 0}_{6} & \underbrace{0 & \cdots & 0}_{5i-5} & 0 & 0 & 0 & 0 & 0 & \underbrace{0 & \cdots & 0}_{5N-5i} \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

$H^i = h^i F_{x,i} =$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ -\sin\bar{\mu}_{\theta_p} & \cos\bar{\mu}_{\theta_p} & \cos\bar{\mu}_{\theta_p}\left(\bar{\mu}_{x_0^i} - \bar{\mu}_{x_p}\right) + \sin\bar{\mu}_{\theta_p}\left(\bar{\mu}_{y_0^i} - \bar{\mu}_{y_p}\right) \\ -\cos\bar{\mu}_{\theta_p} & -\sin\bar{\mu}_{\theta_p} & -\sin\bar{\mu}_{\theta_p}\left(\bar{\mu}_{x_0^i} - \bar{\mu}_{x_p}\right) + \cos\bar{\mu}_{\theta_p}\left(\bar{\mu}_{y_0^i} - \bar{\mu}_{y_p}\right) \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \sin\bar{\mu}_{\theta_p} & -\cos\bar{\mu}_{\theta_p} & 0 \\ 0 & 0 & \cos\bar{\mu}_{\theta_p} & \sin\bar{\mu}_{\theta_p} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$\begin{bmatrix} e_1 & e_2 \end{bmatrix} = eigenvector(\bar{\Sigma}_{x_p,y_p})$

$R_z = \begin{bmatrix} cos(m_\varphi^i - \frac{\pi}{2}) & -sin(m_\varphi^i - \frac{\pi}{2}) \\ sin(m_\varphi^i - \frac{\pi}{2}) & cos(m_\varphi^i - \frac{\pi}{2}) \end{bmatrix}$

$u = \alpha_1 e_1 + \alpha_2 e_2$

▷ Use eigenvectors of the error ellipse to represent the intersection line between the plane landmark and the xy plane

$\sigma_{cut} = u^T \bar{\Sigma}_{x_p,y_p} u$

$Q'_{x_0,y_0} = \begin{bmatrix} \sigma_{cut} & 0 \\ 0 & 0.1 \end{bmatrix}$

$Q_{x_0,y_0} = R_z Q'_{x_0,y_0} R_z^T$

$$Q_s = \begin{bmatrix} Q_\theta & 0 & \mathbf{0} & 0 \\ 0 & Q_\varphi & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{0} & Q_{x_0,y_0} & \mathbf{0} \\ 0 & 0 & \mathbf{0} & Q_{z_0} \end{bmatrix}$$

$K^i = \bar{\Sigma} {H^i}^T \left(H^i \bar{\Sigma} {H^i}^T + Q_s\right)^{-1}$

$\bar{\mu} = \bar{\mu} + K^i \left(m^i - \hat{m}^i\right)$

$\bar{\Sigma} = \left(I - K^i H^i\right) \bar{\Sigma}$

end for

$\mu_3 = \bar{\mu}$

$\Sigma_3 = \bar{\Sigma}$

return

The predicted plane parameters in the printer frame for each detected plane i are:

$$\hat{m}^i = \begin{bmatrix} \hat{\theta}^i & \hat{\varphi}^i & \hat{x}_0^i & \hat{y}_0^i & \hat{z}_0^i \end{bmatrix}^T$$

and calculated by the current estimated states: plane parameters in the global frame, as well as the printer pose in global frame. The equations are as follows:

$$\begin{bmatrix} \hat{\theta}^i \\ \hat{\varphi}^i \\ \hat{x}_0^i \\ \hat{y}_0^i \\ \hat{z}_0^i \end{bmatrix} = \begin{bmatrix} \bar{\mu}_{\theta^i} \\ \bar{\mu}_{\varphi^i} - \bar{\mu}_{\theta_p} \\ \sin\bar{\mu}_{\theta_p}\left(\bar{\mu}_{x_0^i} - \bar{\mu}_{x_p}\right) - \cos\bar{\mu}_{\theta_p}\left(\bar{\mu}_{y_0^i} - \bar{\mu}_{y_p}\right) \\ \cos\bar{\mu}_{\theta_p}\left(\bar{\mu}_{x_0^i} - \bar{\mu}_{x_p}\right) + \sin\bar{\mu}_{\theta_p}\left(\bar{\mu}_{y_0^i} - \bar{\mu}_{y_p}\right) \\ \bar{\mu}_{z_0^i} \end{bmatrix}$$

For updating the printer pose, the Jacobian is a matrix of partial differentials of predicted plane parameters $\hat{m}^i$ with respect to printer pose $\begin{bmatrix} x_p & y_p & \theta_p \end{bmatrix}$ (i.e. $\begin{bmatrix} \bar{\mu}_{x_p} & \bar{\mu}_{y_p} & \bar{\mu}_{\theta_p} \end{bmatrix}$).

$$\begin{bmatrix} \frac{\partial\hat{\theta}^i}{\partial x_p} & \frac{\partial\hat{\theta}^i}{\partial y_p} & \frac{\partial\hat{\theta}^i}{\partial\theta_p} \\ \frac{\partial\hat{\varphi}^i}{\partial x_p} & \frac{\partial\hat{\varphi}^i}{\partial y_p} & \frac{\partial\hat{\varphi}^i}{\partial\theta_p} \\ \frac{\partial\hat{x}_0^i}{\partial x_p} & \frac{\partial\hat{x}_0^i}{\partial y_p} & \frac{\partial\hat{x}_0^i}{\partial\theta_p} \\ \frac{\partial\hat{y}_0^i}{\partial x_p} & \frac{\partial\hat{y}_0^i}{\partial y_p} & \frac{\partial\hat{y}_0^i}{\partial\theta_p} \\ \frac{\partial\hat{z}_0^i}{\partial x_p} & \frac{\partial\hat{z}_0^i}{\partial y_p} & \frac{\partial\hat{z}_0^i}{\partial\theta_p} \end{bmatrix}$$

For updating the plane parameters, Jacobian is a partial differential of state vector with respect to plane parameters $\begin{bmatrix} \theta & \varphi & x_0 & y_0 & z_0 \end{bmatrix}$ (i.e. $\begin{bmatrix} \bar{\mu}_{\theta^i} & \bar{\mu}_{\varphi^i} & \bar{\mu}_{x_0^i} & \bar{\mu}_{y_0^i} & \bar{\mu}_{z_0^i} \end{bmatrix}$).

$$\begin{bmatrix} \frac{\partial\hat{\theta}^i}{\partial\theta} & \frac{\partial\hat{\theta}^i}{\partial\varphi} & \frac{\partial\hat{\theta}^i}{\partial x_0} & \frac{\partial\hat{\theta}^i}{\partial y_0} & \frac{\partial\hat{\theta}^i}{\partial z_0} \\ \frac{\partial\hat{\varphi}^i}{\partial\theta} & \frac{\partial\hat{\varphi}^i}{\partial\varphi} & \frac{\partial\hat{\varphi}^i}{\partial x_0} & \frac{\partial\hat{\varphi}^i}{\partial y_0} & \frac{\partial\hat{\varphi}^i}{\partial z_0} \\ \frac{\partial\hat{x}_0^i}{\partial\theta} & \frac{\partial\hat{x}_0^i}{\partial\varphi} & \frac{\partial\hat{x}_0^i}{\partial x_0} & \frac{\partial\hat{x}_0^i}{\partial y_0} & \frac{\partial\hat{x}_0^i}{\partial z_0} \\ \frac{\partial\hat{y}_0^i}{\partial\theta} & \frac{\partial\hat{y}_0^i}{\partial\varphi} & \frac{\partial\hat{y}_0^i}{\partial x_0} & \frac{\partial\hat{y}_0^i}{\partial y_0} & \frac{\partial\hat{y}_0^i}{\partial z_0} \\ \frac{\partial\hat{z}_0^i}{\partial\theta} & \frac{\partial\hat{z}_0^i}{\partial\varphi} & \frac{\partial\hat{z}_0^i}{\partial x_0} & \frac{\partial\hat{z}_0^i}{\partial y_0} & \frac{\partial\hat{z}_0^i}{\partial z_0} \end{bmatrix}$$

The full Jacobian matrix h^i is a horizontal concatenation of the two parts, and is computed as:

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ -\sin\bar{\mu}_{\theta_p} & \cos\bar{\mu}_{\theta_p} & \cos\bar{\mu}_{\theta_p}\left(\bar{\mu}_{x_0^i}-\bar{\mu}_{x_p}\right)+\sin\bar{\mu}_{\theta_p}\left(\bar{\mu}_{y_0^i}-\bar{\mu}_{y_p}\right) & 0 & 0 & \sin\bar{\mu}_{\theta_p} & -\cos\bar{\mu}_{\theta_p} & 0 \\ -\cos\bar{\mu}_{\theta_p} & -\sin\bar{\mu}_{\theta_p} & -\sin\bar{\mu}_{\theta_p}\left(\bar{\mu}_{x_0^i}-\bar{\mu}_{x_p}\right)+\cos\bar{\mu}_{\theta_p}\left(\bar{\mu}_{y_0^i}-\bar{\mu}_{y_p}\right) & 0 & 0 & \cos\bar{\mu}_{\theta_p} & \sin\bar{\mu}_{\theta_p} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Moreover, a binding matrix $F_{x,i}$ is defined to select which plane among all those planes in state vector is going to be updated during each individual loop of SLAM algorithm. The appropriate set of plane parameters is already identified in the correspondence step, as described in section 3.4.1.

$$F_{x,i} = \begin{bmatrix} 1 & 0 & 0 & \underbrace{0 \cdots 0}_{6} & \underbrace{0 \cdots 0}_{5i-5} & 0 & 0 & 0 & 0 & 0 & \underbrace{0 \cdots 0}_{5} & \underbrace{0 \cdots 0}_{5N-5i} \\ 0 & 1 & 0 & 0 \cdots 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 \\ 0 & 0 & 1 & 0 \cdots 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 & 1 & 0 & 0 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 & 0 & 1 & 0 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 & 0 & 0 & 1 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 & 0 & 0 & 0 & 1 & 0 & 0 \cdots 0 & 0 \cdots 0 \\ 0 & 0 & 0 & 0 \cdots 0 & 0 \cdots 0 & 0 & 0 & 0 & 0 & 1 & 0 \cdots 0 & 0 \cdots 0 \end{bmatrix}$$

H^i is the Jacobian matrix selected by $F_{x,i}$. It is the final Jacobian matrix used in the SLAM algorithm.

$$H^i = h^i F_{x,i} = \begin{bmatrix} \frac{\partial\hat{\theta}^i}{\partial x_p} & \frac{\partial\hat{\theta}^i}{\partial y_p} & \frac{\partial\hat{\theta}^i}{\partial\theta_p} & \frac{\partial\hat{\theta}^i}{\partial\theta} & \frac{\partial\hat{\theta}^i}{\partial\varphi} & \frac{\partial\hat{\theta}^i}{\partial x_0} & \frac{\partial\hat{\theta}^i}{\partial y_0} & \frac{\partial\hat{\theta}^i}{\partial z_0} \\ \frac{\partial\hat{\varphi}^i}{\partial x_p} & \frac{\partial\hat{\varphi}^i}{\partial y_p} & \frac{\partial\hat{\varphi}^i}{\partial\theta_p} & \frac{\partial\hat{\varphi}^i}{\partial\theta} & \frac{\partial\hat{\varphi}^i}{\partial\varphi} & \frac{\partial\hat{\varphi}^i}{\partial x_0} & \frac{\partial\hat{\varphi}^i}{\partial y_0} & \frac{\partial\hat{\varphi}^i}{\partial z_0} \\ \frac{\partial\hat{x}_0^i}{\partial x_p} & \frac{\partial\hat{x}_0^i}{\partial y_p} & \frac{\partial\hat{x}_0^i}{\partial\theta_p} & \frac{\partial\hat{x}_0^i}{\partial\theta} & \frac{\partial\hat{x}_0^i}{\partial\varphi} & \frac{\partial\hat{x}_0^i}{\partial x_0} & \frac{\partial\hat{x}_0^i}{\partial y_0} & \frac{\partial\hat{x}_0^i}{\partial z_0} \\ \frac{\partial\hat{y}_0^i}{\partial x_p} & \frac{\partial\hat{y}_0^i}{\partial y_p} & \frac{\partial\hat{y}_0^i}{\partial\theta_p} & \frac{\partial\hat{y}_0^i}{\partial\theta} & \frac{\partial\hat{y}_0^i}{\partial\varphi} & \frac{\partial\hat{y}_0^i}{\partial x_0} & \frac{\partial\hat{y}_0^i}{\partial y_0} & \frac{\partial\hat{y}_0^i}{\partial z_0} \\ \frac{\partial\hat{z}_0^i}{\partial x_p} & \frac{\partial\hat{z}_0^i}{\partial y_p} & \frac{\partial\hat{z}_0^i}{\partial\theta_p} & \frac{\partial\hat{z}_0^i}{\partial\theta} & \frac{\partial\hat{z}_0^i}{\partial\varphi} & \frac{\partial\hat{z}_0^i}{\partial x_0} & \frac{\partial\hat{z}_0^i}{\partial y_0} & \frac{\partial\hat{z}_0^i}{\partial z_0} \end{bmatrix} F_{x,i}$$

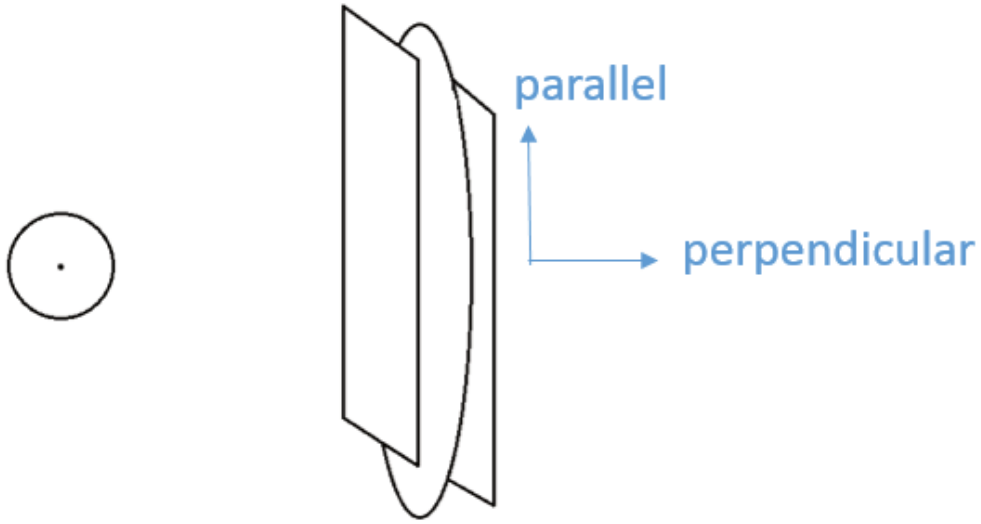

FIGURE 3.4: error ellipse of point landmark (left) and plane landmark (left)

Q_s is the 3D scanner measurement error on planes. In fact, the point measurement error is given in the datasheet of 3D scanner, but using planes as landmarks is different from using point landmarks. For instance, in the direction parallel to the normal vector of a plane, the measurement uncertainty is as small as the 3D scanner point measurement error, while in the direction perpendicular to the normal vector of a plane, the measurement uncertainty is much larger, as even a large amount of movement in that direction would not have any impact on the representation of that plane. (Figure 3.4)

Therefore, the uncertainty of a plane landmark is an ellipse with major axis along the plane and minor axis perpendicular to the plane. In most cases, that ellipse is not axis-aligned. The direction of that ellipse is related to the direction of the plane in local frame. Therefore, we at first define an x-axis aligned ellipse Q' as:

$$Q' = \begin{bmatrix} \sigma_{cut} & 0 \\ 0 & 0.1 \end{bmatrix}$$

σ_{cut} is determined by a cut of the printer position error ellipse with a line defined as:

$$u = \alpha_1 e_1 + \alpha_2 e_2$$

Where α_1 and α_2 are the coefficients, e_1 and e_2 are the eigenvectors of the printer position covariance matrix $\bar{\Sigma}_{x_p,y_p}$. The length of the intersection is calculated as:

$$\sigma_{cut} = u^T \bar{\Sigma}_{x_p,y_p} u$$

Then the final ellipse $Q_{x0,y0}$ can be generated by rotating $Q'_{x0,y0}$ by $\left(\bar{\mu}_{\varphi^i} - \frac{\pi}{2}\right)$ radians, the rotation matrix is as follows.

$$Q_{x0,y0} = R_z Q'_{x0,y0} R_z^T$$

Finally, the covariance of planar patch can be defined as follows.

$$Q_s = \begin{bmatrix} Q_\theta & 0 & \mathbf{0} & 0 \\ 0 & Q_\varphi & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{0} & Q_{x_0,y_0} & \mathbf{0} \\ 0 & 0 & \mathbf{0} & Q_{z_0} \end{bmatrix}$$

m^i is the actual measurement of 3D scanner. It is mentioned in the section 3.4.1 that m^i saves planes with their general form and the centroids of the planar patches. In order to match the format of state vector, the directions of planes need to be converted from Cartesian coordinate normal vector $\begin{bmatrix} b_1^i & b_2^i & b_3^i \end{bmatrix}$ into spherical coordinate angles $\begin{bmatrix} \theta^i & \varphi^i \end{bmatrix}$. The equations are as follows:

$$\theta^i = \arccos\left(\frac{b_3^i}{(b_1^i)^2 + (b_2^i)^2 + (b_3^i)^2}\right)$$

$$\varphi = \arctan\left(\frac{b_2^i}{b_1^i}\right)$$

Recall that b_i here are in the printer frame.

Finally, Kalman gain K^i, updated state and its covariance, μ and Σ respectively, can be calculated. The equations are as follows:

$$K^i = \bar{\Sigma} H^{iT} \left(H^i \bar{\Sigma} H^{iT} + Q_s\right)^{-1}$$

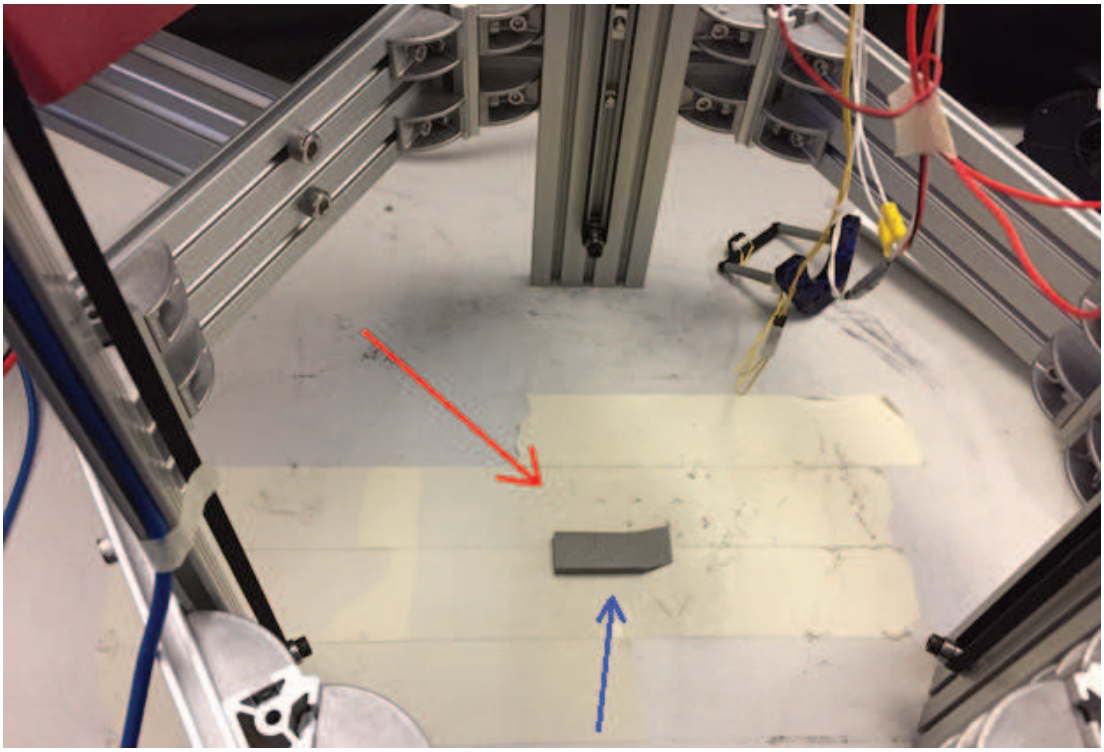

FIGURE 3.5: scan direction

$$\mu_3 = \bar{\mu} + K^i \left(m^i - \hat{m}^i\right)$$

$$\Sigma_3 = \left(I - K^i H^i\right) \bar{\Sigma}$$

3.5 Move the printer

In this step, the printer is moved to an appropriate pose for scanning side planes and ramp planes, the state vector is updated from μ_3 to μ_4. Note that the planned motion may need to take into account the suitability of the 3D scanner orentation; automation of this may be fruitful future work. For example, as shown in Figure 3.5, the red arrow shows the current scanning direction, and the blue arrow shows the optimal scan direction. Scanning from the current direction will result in a bad scan quality on the side planes. Therefore, the printer should be rotated to get a better scan result.

After motion, the printer pose $\begin{bmatrix} x_p & y_p & \theta_p \end{bmatrix}$ in the state vector is updated by odometry, giving μ_4. The covariance of the printer pose is updated as well, according to the motion model of EKF algorithm:

$$\Sigma_{4:x_p,y_p,\theta_p} = G\Sigma_{t-1}G^T + R$$

G is the Jacobian for motion model. Since the printer is not yet integrated with a mobile robot, the printer is actually moved by hand. Therefore, G does not exist in our case. On the other hand, R is a 3×3 matrix representing odometry error, but here it actually represents the position and angle uncertainty of putting the printer at desired pose by hand. This matrix of uncertainty is defined as:

$$R = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 3^\circ \end{bmatrix}$$

I.e. 10mm for position error and 0.0524 rad (3 degrees) for angular error.

3.6 Total station (2)

The total station (2) step is used to estimate the pose of the printer in global frame by total station right after it has been moved. A commonly implemented EKF-SLAM algorithm using point landmarks (in our case, optical prisms) is applied in this process (Algorithm 3). At the start of this step, the state vector $\bar{\mu}$ contains the predicted printer pose after moving, the landmark position estimates from the prior "total station" step, and the estimated planes from the prior "3D scanner" step. With new total station measurements, the EKF-SLAM algorithm can update the printer pose and the landmark positions simultaneously.

A polar coordinate system is used in this SLAM, since we have different confidence for bearing and range measurements from total station. Therefore, a conversion from Cartesian

Algorithm 3 Total station SLAM algorithm (μ_4, σ_4, m^i)

$\bar{\mu} = \mu_4$

$\bar{\Sigma} = \Sigma_4$

for all observed landmarks m^i **do**

$$\hat{m}^i = \begin{bmatrix} \hat{r}^i \\ \hat{\beta}^i \end{bmatrix} = \begin{bmatrix} \sqrt{(\bar{\mu}_{i,x} - \bar{\mu}_x)^2 + (\bar{\mu}_{i,y} - \bar{\mu}_y)^2} \\ \text{atan2}\left(\bar{\mu}_{i,y} - \bar{\mu}_y, \bar{\mu}_{i,x} - \bar{\mu}_x\right) - \bar{\mu}_\theta \end{bmatrix}$$

$$F_{x,i} = \begin{bmatrix} 1 & 0 & 0 & \underbrace{0 \quad \dots \quad 0}_{2i-2} & 0 & 0 & \underbrace{0 \quad 0}_{2} & 0 & \dots & 0 \\ 0 & 1 & 0 & & 0 & 0 & & 0 & \dots & 0 \\ 0 & 0 & 1 & & 0 & 0 & & 0 & \dots & 0 \\ 0 & 0 & 0 & & 0 & 0 & & 0 & \dots & 0 \\ 0 & 0 & 0 & & 0 & 0 & & \underbrace{0 \quad 0}_{2N-2i} & \dots & 0 \end{bmatrix}$$

$q = (\bar{\mu}_{i,x} - \bar{\mu}_x)^2 + (\bar{\mu}_{i,y} - \bar{\mu}_y)^2$

$\delta_x = \bar{\mu}_{i,x} - \bar{\mu}_x$

$\delta_y = \bar{\mu}_{i,y} - \bar{\mu}_y$

$$H^i = \frac{1}{q} h^i F_{x,i} = \frac{1}{q} \begin{bmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & \sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{bmatrix} F_{x,i}$$

$$K^i = \bar{\Sigma} {H^i}^T \left(H^i \bar{\Sigma} {H^i}^T + Q_T \right)^{-1}$$

$\bar{\mu} = \bar{\mu} + K^i \left(m^i - \hat{m}^i \right)$

$\bar{\Sigma} = \left(I - K^i H^i \right) \bar{\Sigma}$

end for

$\mu_5 = \bar{\mu}$

$\Sigma_5 = \bar{\Sigma}$

return

coordinate system to polar coordinate system is needed, as shown below.

$$r^i = \sqrt{\left(x^i\right)^2 + \left(y^i\right)^2}$$

$$\beta^i = \tan^{-1} \frac{y^i}{x^i}$$

Then, the observed features from the total station, given as $\begin{bmatrix} x^i & y^i \end{bmatrix}$ can be represented by:

$$m^i = \begin{bmatrix} r^i & \beta^i \end{bmatrix}^T$$

According to the predicted landmark positions in global frame $\begin{bmatrix} \bar{\mu}_{i,x} & \bar{\mu}_{i,y} \end{bmatrix}$ and predicted printer pose in global frame $\begin{bmatrix} \bar{\mu}_x & \bar{\mu}_y & \bar{\mu}_\theta \end{bmatrix}$, landmarks in total station frame can be estimated

as $\hat{m}^i$

$$\hat{m}^i = \begin{bmatrix} \hat{r}^i \\ \hat{\beta}^i \end{bmatrix} = \begin{bmatrix} \sqrt{(\bar{\mu}_{i,x} - \bar{\mu}_x)^2 + (\bar{\mu}_{i,y} - \bar{\mu}_y)^2} \\ \text{atan2}\left(\bar{\mu}_{i,y} - \bar{\mu}_y, \bar{\mu}_{i,x} - \bar{\mu}_x\right) - \bar{\mu}_\theta \end{bmatrix}$$

h^i is the Jacobian matrix for estimated measurements $\hat{m}^i$, i.e. the partial differential of $\hat{m}^i$ with regards to the printer pose and landmark positions.

$$h^i = \begin{bmatrix} \frac{\partial \hat{r}^i}{\partial x_p} & \frac{\partial \hat{r}^i}{\partial y_p} & \frac{\partial \hat{r}^i}{\partial \theta_p} & \frac{\partial \hat{r}^i}{\partial x^i} & \frac{\partial \hat{r}^i}{\partial y^i} \\ \frac{\partial \hat{\beta}^i}{\partial x_p} & \frac{\partial \hat{\beta}^i}{\partial y_p} & \frac{\partial \hat{\beta}^i}{\partial \theta_p} & \frac{\partial \hat{\beta}^i}{\partial x^i} & \frac{\partial \hat{\beta}^i}{\partial y^i} \end{bmatrix}$$

The matrix $F_{x,i}$ is introduced to map the low-dimentional matrix h^i into a matrix H^i with dimension of $2N + 3$, N is the number of landmarks, which is 3 in our work.

$$F_{x,i} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \underbrace{0 \quad \dots \quad 0}_{2i-2} & & & \underbrace{0 \quad 1}_{2} & & \underbrace{0 \quad \dots \quad 0}_{2N-2i} & & \end{bmatrix}$$

$$H^i = \frac{1}{q} h^i F_{x,i} = \frac{1}{q} \begin{bmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & \sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{bmatrix} F_{x,i}$$

Where

$$q = (\bar{\mu}_{i,x} - \bar{\mu}_x)^2 + (\bar{\mu}_{i,y} - \bar{\mu}_y)^2$$

$$\delta_x = \bar{\mu}_{i,x} - \bar{\mu}_x$$

$$\delta_y = \bar{\mu}_{i,y} - \bar{\mu}_y$$

Q_T is the total station measurement error. It is mentioned in Section 2.2.3 that due to a potentially unleveled condition the position measurement error (with respect to its calibration to the printer frame) is much larger than the number in the total station datasheet, but the

angle measurement is not affected, so Q_T is defined as:

$$Q_T = \begin{bmatrix} 10 & 0 \\ 0 & 4.7124e-06 \end{bmatrix}$$

K^i is the gain of EKF algorithm.

$$K^i = \bar{\Sigma} {H^i}^T \left(H^i \bar{\Sigma} {H^i}^T + Q_T \right)^{-1}$$

$$\mu_5 = \bar{\mu} + K^i \left(m^i - \hat{m}^i \right)$$

$$\Sigma_5 = \left(I - K^i H^i \right) \bar{\Sigma}$$

This algorithm updates both printer pose and landmark positions. The updated printer pose and its zoomed-in view are shown in Figure 3.6. The blue circle represents the original printer pose, the red circle represents the predicted printer pose (from odometry), the green circle represents the updated printer pose, and the green dots show the updated landmark positions.

3.7 3D scanner (2)

Printer position estimation is not accurate enough from total station SLAM alone, and needs a further correction from the 3D scanner.

The "3D scanner (2)" step is similar to the previously described "3D scanner (1)" step, so the algorithm will not be repeated. The only difference is at "3D scanner (2)" step, the input is μ_5 and the output is μ_6.

Error ellipse is introduced to visualize the confidence interval of a 2D Gaussian distribution, which in our case is the printer position (x_p and y_p) part of the covariance matrix (2×2). The confidence of the error ellipse is set to be 95%, which defines the region that contains 95% of all samples that can be drawn from the Gaussian distribution $\mathcal{N}(\mu_{x_p,y_p}, \Sigma_{x_p,y_p})$.

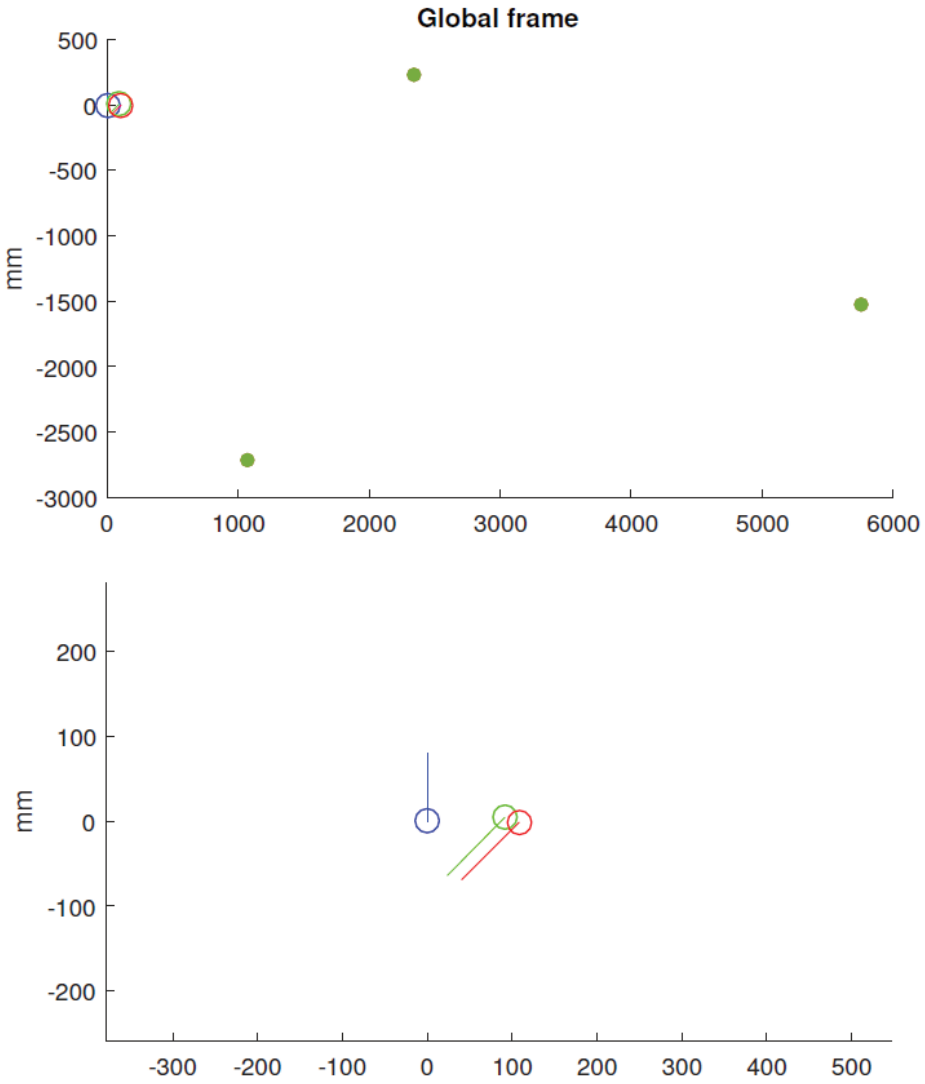

FIGURE 3.6: The figure at the top shows the map of global frame, filled dots represent landmarks and circles represent printer poses. A zoomed in image is shown at the bottom, where blue circle represents the original printer pose, red circle represents the odometry printer pose, and the green circle represents the printer pose result of total station SLAM

The top part of Figure 3.7 shows the changes of error ellipse following 3D scanner SLAM steps, and the bottom of the figure shows a zoomed in view of the same. By running the algorithm on the first plane (in this case, the ramp plane), the error ellipse gets narrow in the x direction but not much different in the y direction, since the plane does not help decrease the uncertainty perpendicular to the normal vector. By running the algorithm on a side plane perpendicular to the ramp plane, the error ellipse gets narrow in the y direction. Eventually, the overall error ellipse gets very close to the anticipated printer position which

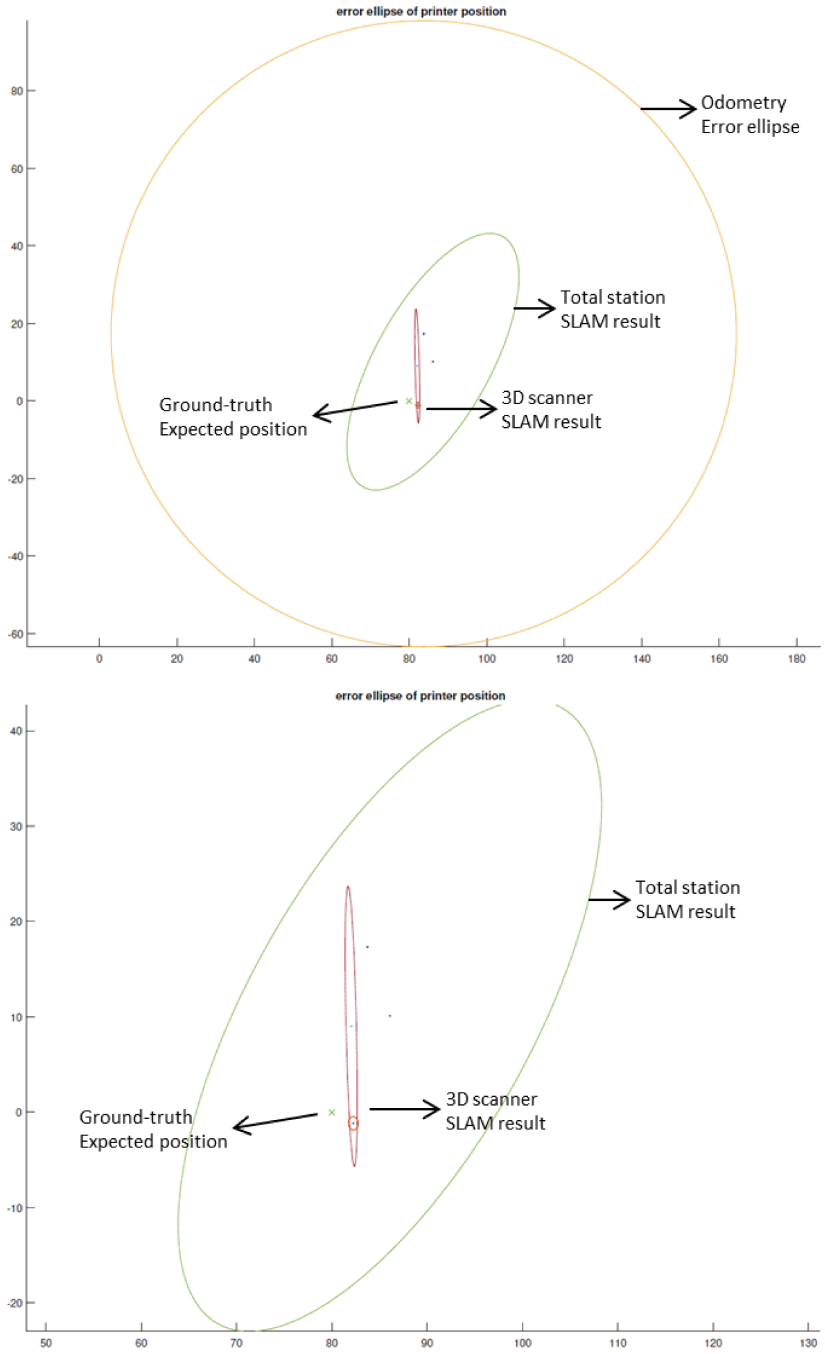

FIGURE 3.7: The figure at the top shows the odometry error ellipse (yellow), error ellipse after total station SLAM (green), and error ellipse after 3D scanner SLAM (circles inside the green ellipse). A zoomed in image is shown at the bottom, after running total station and 3D scanner SLAM sequentially, error ellipse gets closer to the ground-truth expected position (green cross)

is shown by the green cross.

3.8 Next print

According to the updated state vector from the last step μ_6, the printer pose is given and an STL file for the next model can also be generated based on the current updated estimate of the ramp plane. The maximum and minimum limits of the segment in x and y directions in global frame can be easily acquired from the STL model, so the center of the segment in global frame can be calculated as:

$$(\frac{x_{min} + x_{max}}{2}, \frac{y_{min} + y_{max}}{2})$$

Next, the relative position and angle between the next segment and the printer pose are computed. This relative position and orientation comprise the segment pose in printer frame. This pose is entered into the 3D printing software, and the next segment is printed.

After that, the algorithm makes a prediction of plane parameters of the current printed model. The prediction is made based on the ramp plane in the current state vector μ_6 and the segmenting length we set. Since it is made before measurement, the next SLAM, in a way, is actually a predictive SLAM.

The process of updating planes on the model based on the prediction is as follows:

1. At first, an STL to planes function based on the post-printing ramp plane (i.e. predicted ramp plane) is run, so we get a list of planes p1.

2. Then, another STL to planes function based on the pre-printing ramp plane (i.e. current ramp plane in the current state vector) is run, so we get another list of planes p2.

3. The result of the STL to planes function contains directions, positions and boundaries of planar patches on the segmented CAD model, it is easy for us to compare those

characters and obtain the intersection and difference between the results of two STL to planes function (p1 and p2). Those planes that do not exist in p2 but exist in p1 or those planes whose boundaries have prolonged from p1 to p2 are regarded as updated planes, the other planes are regarded as non-updated planes.

4. After that, the current plane parameter part of the state vector is saved as p3 for later use and then totally replaced by p1 and covariance of this part is given by a large number as

$$\Sigma_p^i = \delta' = \begin{bmatrix} 0.015 & 0 & 0 & 0 & 0 \\ 0 & 0.05 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0.2 \end{bmatrix}$$

   I.e. 0.015 rad for θ angular error, 0.05 rad for φ angular error, 1 mm for x direction position error, 1 mm for y direction position error, 0.2 mm for z direction position error.

   Since both the printing error and the uncertainty of printer pose is introduced in the newly printed planes.

5. Finally, the state and covariance are replaced by those non-updated planes in p3, so updated planes are initialized with large uncertainty and non-updated planes are kept the same.

The output of this step is again the updated state vector μ_2 and the previous process is repeated until the whole model is printed.

Chapter 4

RESULTS AND DISCUSSION

To evaluate and compare print quality results, the procedure for deviation analysis is introduced here. Observations are then presented regarding the effects of sensor fusion, planar patch representation, and odometry error on print quality.

4.1 Deviation analysis

Deviation analysis compares the final geometry of the part achieved using mobile 3D printing to the originally planned CAD model. This gives us a measure of the final print quality. The Deviation analysis can be done in the GOM inspection software. The steps are as follows:

- Import both the nominal element (CAD model, blue) and actual element (3D scanning result, grey) into the inspection software and clear the unwanted parts (e.g. table surface) in the scanning result (Figure 4.1).
- Align the imported two models in order to compare (Figure 4.2).
- Do the surface comparison on the actual element (Figure 4.3 4.4). Deviation is defined as the distance from a point on the printed model to the closest point on the CAD model surface.

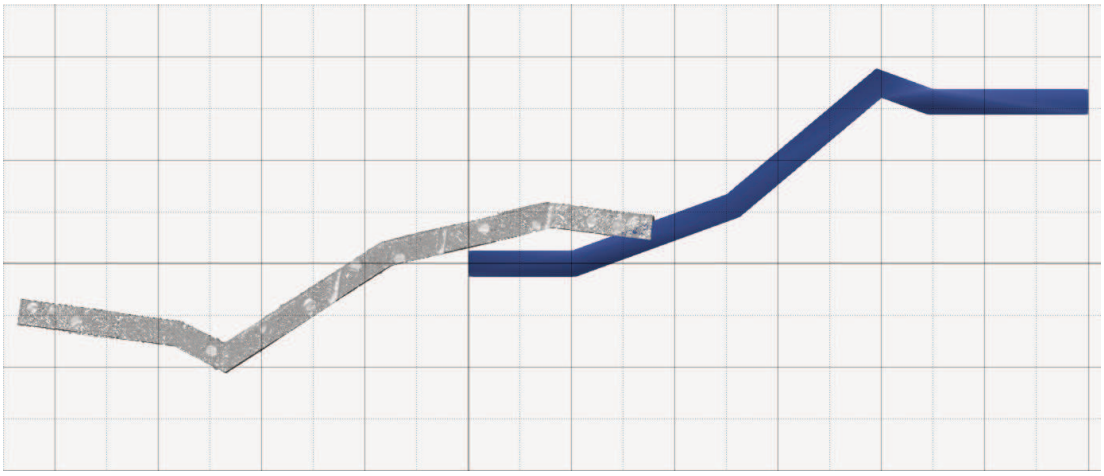

FIGURE 4.1: imported nominal element (CAD model, blue) and actual element (3D scanning result, grey)

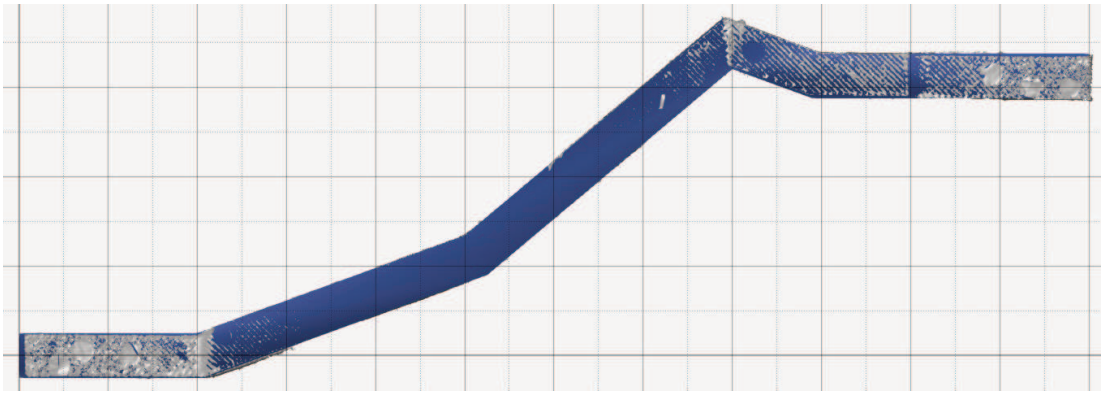

FIGURE 4.2: Alignment result of the nominal element (blue) and the actual element (grey)

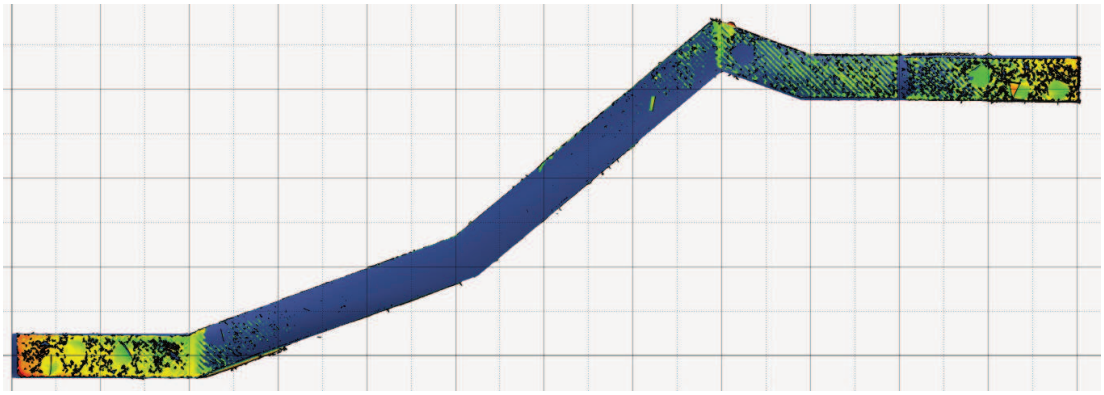

FIGURE 4.3: Surface comparison 1: green parts indicate low deviation parts, red and blue parts indicate high deviation parts

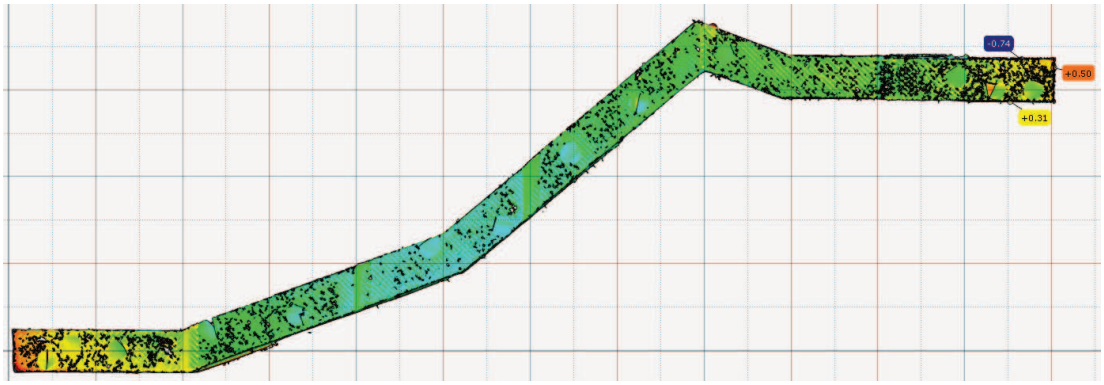

FIGURE 4.4: Surface comparison 2: labels can be placed on any point on the model. The highest deviation in this example is 0.74mm

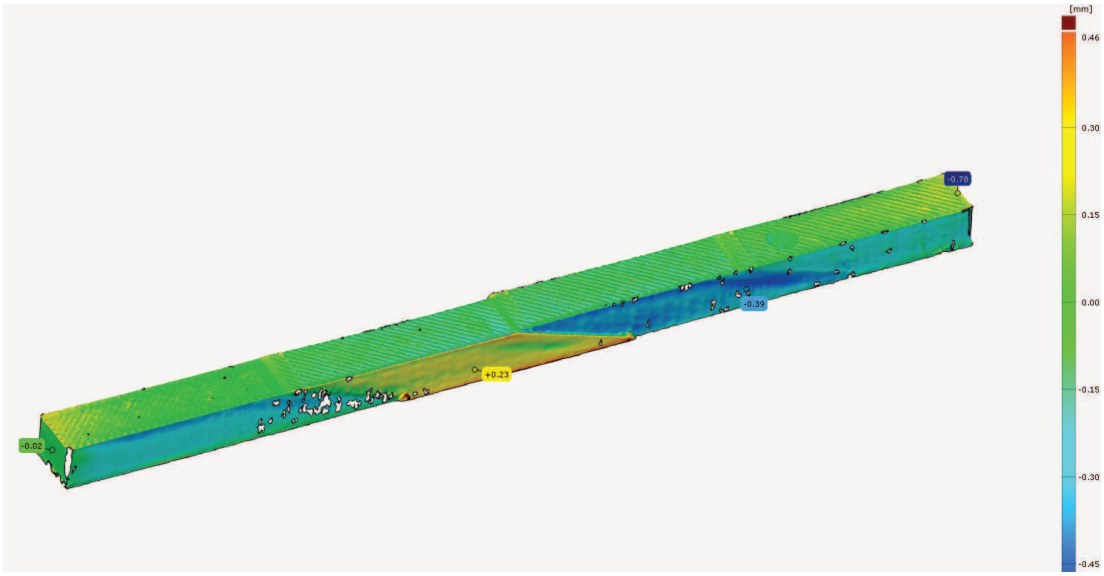

FIGURE 4.5: Surface comparison of a rectangle model.

An example rectangular model is printed and analyzed by the GOM software (Figure 4.5). The largest deviation between actual printed model and CAD model is 0.78mm, whereas the total geometric extent of the full model is 240mm.

4.2 Importance of fusing global and local sensors

It is important to evaluate if both global (total station) and local (3D scanner) sensors are required to achieve satisfactory printing. It turns out that the 3D scanner is required to

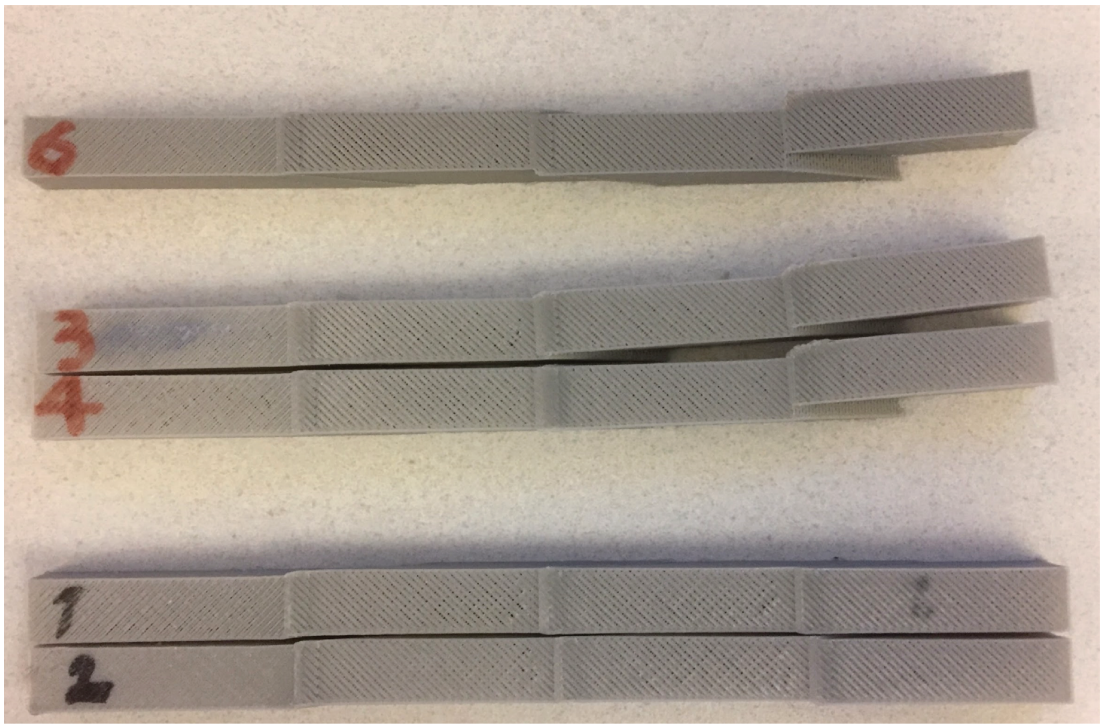

FIGURE 4.6: Printing result comparison

achieve good connection between segments while the total station is required to avoid drift relative to the intended geometry.

In Figure 4.6, model number 6 shows the printing result if the total station measurement is skipped and only the 3D scanner is used as the sensor. The model eventually drifts as the printer goes far from the original position, which demonstrates that the 3D scanner is not able to provide as accurate global angular measurements as the total station does.

The reasons for drift are as follows:

1. The planes used as landmarks are affected by potential print errors.
2. When dealing with the 3D scanner measurement, including the 3D scanner calibration, the algorithm uses a plane fitting function to extract planes from a point cloud. This plane fitting function has uncertainty since there is tolerance when fitting planes.
3. It is very likely that the 3D scanner detects a side plane made up of more than one segment. If there is even minor drift between those different segments, the plane fitting

function takes an average plane at a middle position between them. This means there is no full recovery from any drift.

Models number 3 and 4 show the printing results when setting the total station angular measurement error to be an incorrectly large number. In this case, we replaced the angular measurement error 0.3 mgon (0.0000047 rad) in the total station datasheet with 3° (0.052 rad). The print results show that the models also drift when the printer gets away from origin position, which demonstrates the necessity of trusting total station angular measurements.

The reasons for setting small angular measurement error for total station are as follows:

1. The total station has very accurate angular measurements. If we don't use these measurements to make significant adjustments to θ_p (because we incorrectly don't trust the measurement) there is little added value to using the total station. Compare these results to the result of model 6, discussed above.
2. Any (relatively small) angular error due to an unleveled condition (e.g. from tilt of table) has minimal, if any, affect the yaw of total station angular measurements.
3. At each new individual position, the total station repeatedly measures against landmarks that are stationary in the global frame, so that the error will not accumulate like they do when relative measurements against surfaces that could be drifting due to print errors.

Models number 1 and 2 show the printing result when using both the total station and 3D scanner, with all the correct error parameters set. The results have much better overall quality.

In addition, printing without the 3D scanner is also tested in our study, but since the total station gives a large position measurement error, the next printed segment has an obvious drift from the very beginning, such that the first two segments often do not even connect (Figure 4.7).

FIGURE 4.7: The printing result if 3D scanner measurement is skipped. The drift of the next segment is too large to even connect with the first segment.

4.3 3D plane patch representation

The reason for using 5 parameters (2 angles in spherical coordinates to indicate the direction of planes and 1 point in 3D Cartesian coordinates to indicate the centroid of planar patches) for plane representation instead of 4 parameters (general form: 1 vector in 3D Cartesian coordinates to indicate the direction of planes and 1 distance from plane to origin indicates position of infinite planes) is explained in detail in Section 2.2.

The following experiment demonstrates that if the odometry error is relatively large (in this example, the odometry error is set to be 30 mm) using general form for plane representation results in a bad position estimation. As shown in Figure 4.8, the error ellipse of the printer position gets smaller through each SLAM algorithm loop, but the final result (the smallest ellipse) is still about 4 mm away from the expected position (i.e. ground truth green cross). This offset is too large; printing the next segment with this estimation will result in an overlap or gap between two segments. Compare Figure 4.9, showing print results using general form,

to Figure 4.13, showing results with the same odometry error using the proposed 5-parameter 3D plane patch representation. There is 0.75mm deviation using general form (Figure 4.10), compared to 0.31mm deviation using the proposed 5-parameter form (Figure 4.14), after segmenting a 100mm model into just 2 segments.

4.4 Sensitivity to odometry error

With the sensor fusion mentioned in the section 4.2 and the 5-parameter representation mentioned in the section 2.2, the algorithm can reach millimeter localization accuracy. Even if we introduce large odometry error, the SLAM algorithm can still adjust the printer to the correct position.

The results of error ellipses with large odometry error are shown in the plots 4.11 and 4.12. Plot 4.11 shows the result when the odometry error is 30mm, and plot 4.12 is the same result after zooming in. The largest yellow circle and its center represents the odometry error ellipse and its mean position, the green ellipse and its center represents the error ellipse and its mean position after total station SLAM, and the smaller ellipses and their centers represent error ellipses and their mean positions after running 3D scanner SLAM for each plane captured (top and bottom planes don't participate in the SLAM algorithm). The final position estimate is very close to the expected (ground truth) position (green cross).

Figure 4.13 shows the print result of introducing 30mm odometry error. There is no drift or gap between two segments, the connection between them is very solid.

Another experiment is conducted to test as the odometry error gets larger, at what point the algorithm starts to diverge. To test this, the printer is moved to a known new position, $\begin{bmatrix} x_{true} & y_{true} \end{bmatrix}^T$. Then, position estimate updates with increasing odometry error are made to see if the algorithm can correct the estimate back to the true position. For each of 9 odometry errors, R, ranging from 15mm to 1500mm, the odometry estimates are sampled from a normal distribution with mean $\begin{bmatrix} x_{true} & y_{true} \end{bmatrix}^T$ and covariance matrix $\begin{bmatrix} R & 0 \\ 0 & R \end{bmatrix}$. Ten

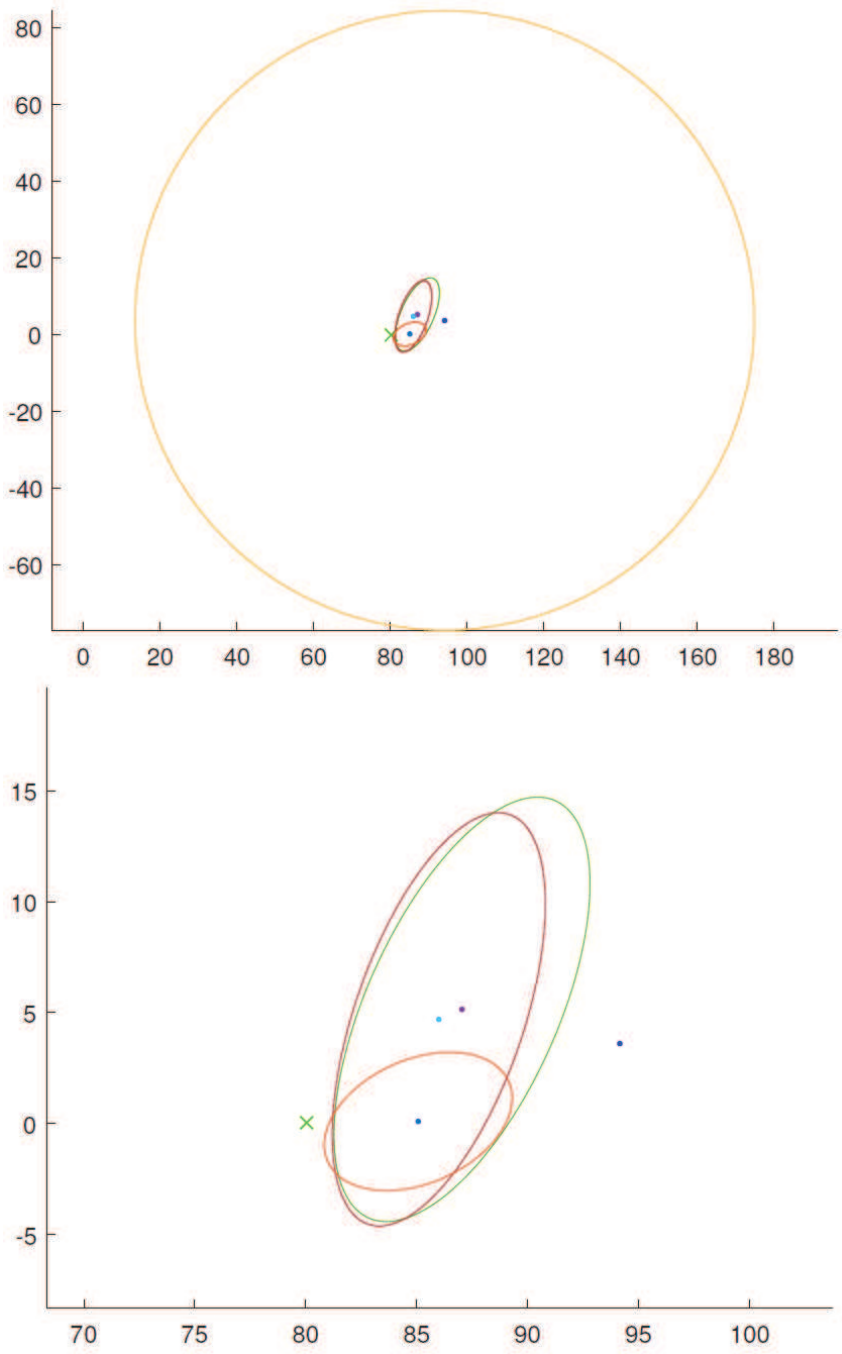

FIGURE 4.8: The figure at the top shows result of error ellipses if general form is used as plane representation. The figure at the bottom shows the zoomed in image, the SLAM algorithm is not able to adjust the printer close enough to the expected position

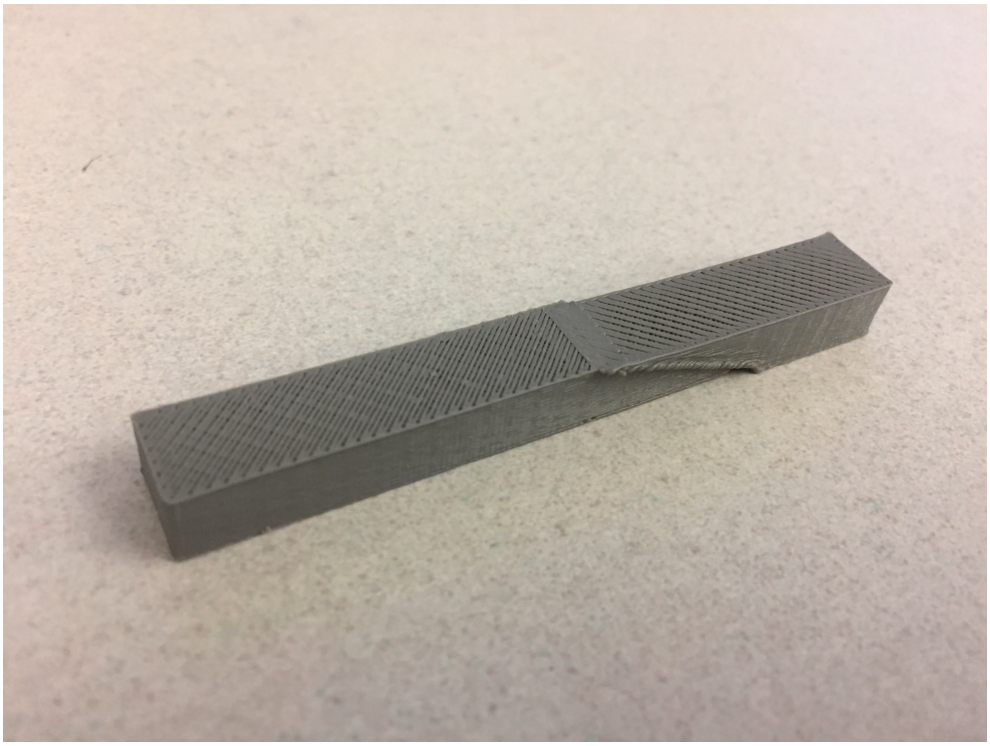

FIGURE 4.9: The print result of using general form as plane representation with 30mm odometry error

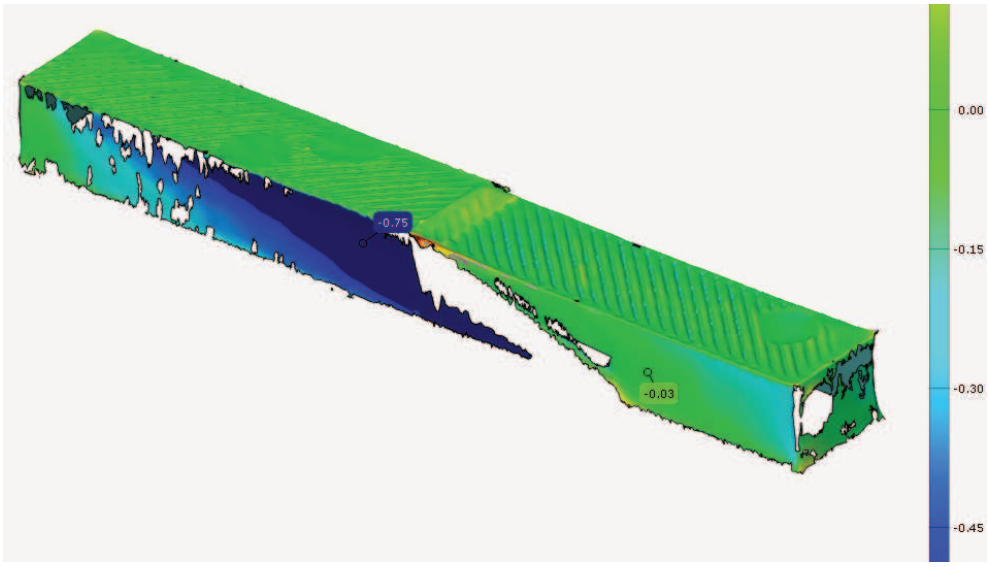

FIGURE 4.10: The deviation analysis of printing result using 4-parameter (general form) as plane representation

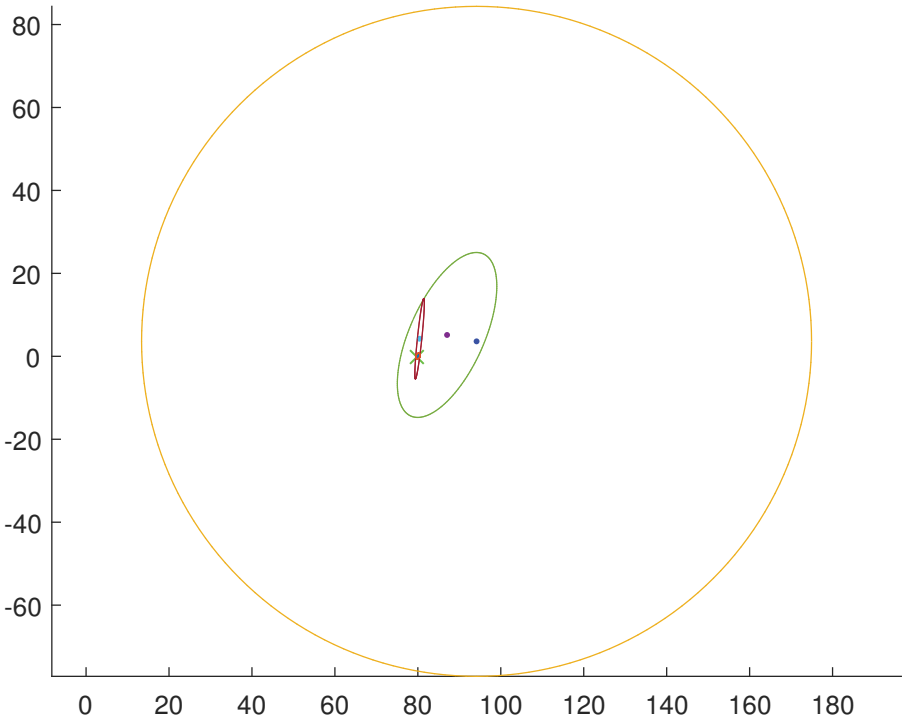

FIGURE 4.11: The print result of 30mm odometry error

samples are taken for each R, and the total station SLAM and 3D printer SLAM are run on each odometry sample. The distances between the final estimated printer positions (i.e. post-SLAM) and the true printer location (measured by hand), d, are calculated for each sample, provided in Appendix B.

Table 4.1 shows the average $\bar{d}$ and standard deviation σ, versus each odometry error, R. When the odometry error is within 100mm, $\bar{d}$ is kept smaller than 2mm, but if the odometry error continues getting larger, $\bar{d}$ starts to diverge.

The result of this experiment is satisfying. Considering the size of each segment we are trying to print is about 60mm, from a practical perspective, our algorithm converges when the odometry error is within a reasonable range. When moving a rover 60mm to position for the next print, an odometry error over 100mm would never be reasonably expected.

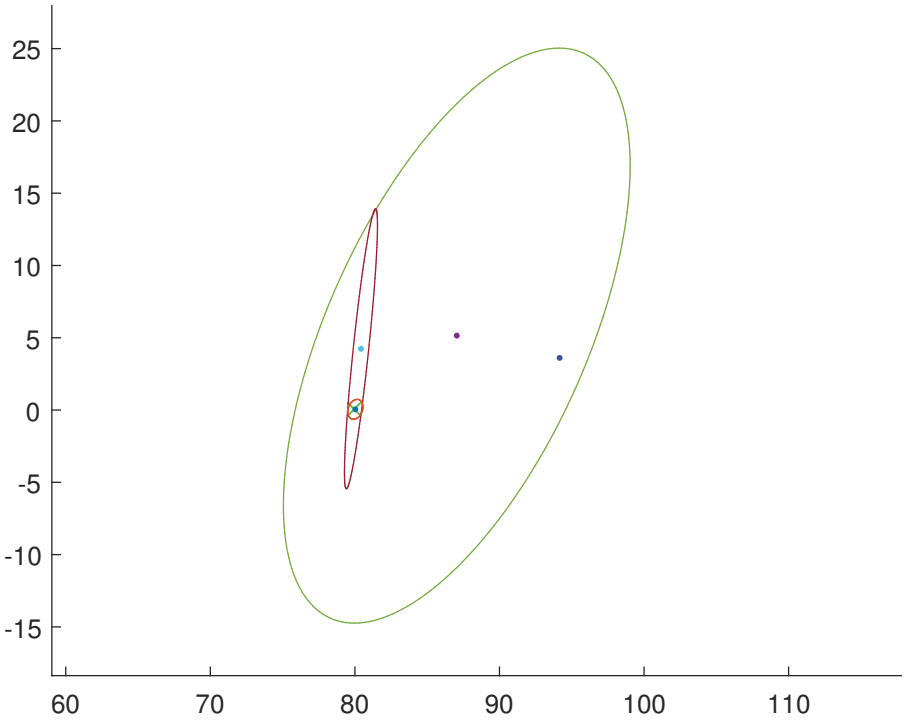

FIGURE 4.12: zoomed in plot of odometry error set to be 30mm

TABLE 4.1: Table of distance between final estimate and groud-truth expected position (mm) vs. odometry error (mm)

Odometry error R	15	30	45	60	100	200	500	1000	1500
Average $\bar{d}$	1.8388	1.74455	1.57119	1.7866	1.85843	3.17622	5.51937	12.66657	12.62124
Standard deviation σ	0.3195	0.4023	0.6667	0.3300	0.3536	2.7432	3.4512	11.0354	10.3417

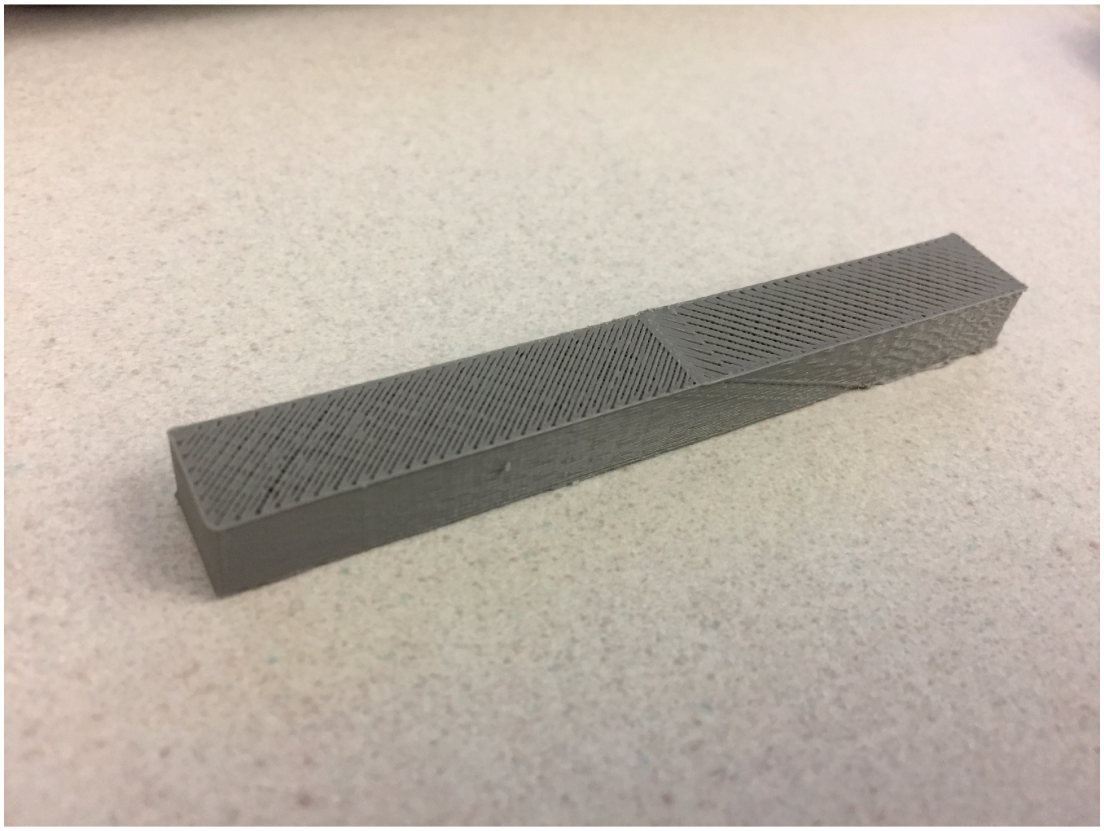

FIGURE 4.13: print result of 5-parameter planar patch representation with 30mm odometry error

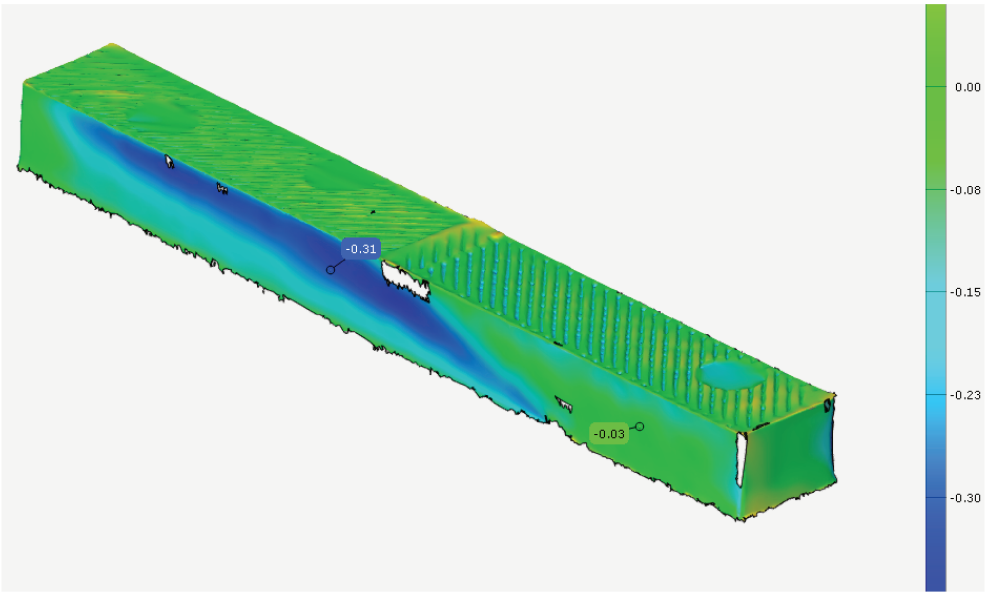

FIGURE 4.14: The deviation analysis of printing result using 5-parameter (2 angles in spherical coordinate and centroid position of planar patches) plane representation

Chapter 5

CONCLUSIONS AND FUTURE WORK

The major conclusions of this work are summarized as follows:

EKF-SLAM algorithm for mobile 3D printing

In contrast to commonly used forms of Simultaneous Localization And Mapping (SLAM), we propose a novel algorithm for predictive SLAM where the printer is building the physical features that are then mapped/modeled. Since we have knowledge of what we are going to build from a CAD model, we can directly add the predicted new planes into the state vector right after printing, before we measure them. On the other hand, it is not only complicated but also unnecessary to measure all the planes newly added to the model, since it needs a 3D scanner to scan model from multiple directions. Therefore, using predictive SLAM in simultaneous localization and modelling algorithm is a feasible approach. Furthermore, according to the experiment testing the sensitivity of the algorithm to the odometry error, the algorithm starts to diverge only if $\sigma_x^2 = \sigma_y^2$ is around 100mm, while the length of each segment is only 60mm. This experiment demonstrates the algorithm has the ability to adjust printer position correctly for odometry uncertainty that is within what could reasonably be expected.

Sensor fusion

Through the printing process, the algorithm updates printer pose, landmark position and planes on the model as the states. We use both total station and 3D scanner as our sensors. These two sensors are fused to overcome the deficiency of each individual sensor. The total station helps to get the printer pose (especially its angle) in the global frame when the printer is moved to a new location. The total station has a large range and a very accurate angle measurement. However, since the total station is mounted on top of the 3D printer, it requires a transformation from total station to printer. Due to the height of the printer and thus distance between these two frames, the position measurement accuracy is affected by any tilted angle of the print bed. On the other hand, the 3D scanner has a much smaller measurement area and its angle measurements are less accurate than the total station, but it has better position measurements in the local frame. Therefore, we can get an accurate angle estimation but rough position estimation from the total station, and the 3D scanner is able to further correct the position of the printer. The experiments in Chapter 4 demonstrated the necessity of this implementation. Using only a 3D scanner or using a 3D scanner with a total station with large angle measurement errors will both generate obvious drift as the printer moves away from the origin position, while the fused implementation shows higher quality print results.

Novel plane parameterization

We compared approaches to represent planar patches, and finally decided to present a representation of 3D planes that is particularly suited for updating planar patches located far from the origin. General form is a commonly used plane representation, but it is not feasible for this study. The general form represents infinite planes by their normal vectors and the distances of planes to the origin. However, if the plane on the model is far away from the origin, even a little bit of angle uncertainty will give rise to greatly varying distance between plane and origin [42]. This will result in the error of planes increasing as the printer moves far away from the origin. In order to solve this issue, we represent planes by their normal vectors in a spherical coordinate system (polar angle, and azimuthal angle) and the centroids of the planar patches. This representation is similar to [45], but representing the

normal vectors in spherical coordinate system helps to keep the normal vectors normalized during the algorithm. So instead of dealing with infinite planes, by this way, we only deal with planar patches on the planes. The experiment shows the new approach to represent planes gives improved results.

Future work

Since our current method only prints models extending along one direction, our future work involves developing a method that can deal with mobile 3D printing all three x-y-z three directions. In addition, it would be interesting to compare our current plane representation to the SPmodel in the future. With SPmodel, the algorithm can use other symmetric geometric shapes as landmarks instead of planes only. And for the moment, we are still moving the printer by hand and all the measurements are still done manually. Eventually, the 3D printer should be mounted on top of a mobile robot to achieve automatic mobile 3D printing. Moreover, our current method for find correspondence between scanning measurement and states is not using any probabilistic algorithm. In the future, finding correspondence should be in a probabilistic framework.

Bibliography

[1] Eddie Krassenstein. D-shape looks to 3d print bridges, a military bunker, and concrete/metal mixture, 2014. URL https://3dprint.com/27229/d-shape-3d-printed-military/.

[2] contour crafting cooperation. URL http://contourcrafting.com/. [Online; accessed 7-May-2018].

[3] Mit develops solar-powered rolling robot that can 3d print entire buildings. URL https://www.designboom.com/technology/mit-dcp-3d-print-buildings-04-28-2017/. [Online; accessed 12-June-2018].

[4] Mx3d bridge. URL http://mx3d.com/projects/bridge-2/.

[5] Lloyd Alter. 3d printed buildings proposed for the moon. URL https://www.treehugger.com/green-architecture/3d-printed-buildings-proposed-moon.html. [Online; accessed 7-May-2018].

[6] Heidi Milkert. 3&dbot mobile 3d printer has no print volume limitations. URL https://3dprint.com/15508/3dbot-mobile-3d-printer/. [Online; accessed 12-June-2018].

[7] Shawn Bulger and Krzysztof Skonieczny. Towards mobile 3d printing for planetary construction. *Earth and Space*, page 324, 2016.

[8] Jae-Won Choi, Francisco Medina, Chiyen Kim, David Espalin, David Rodriguez, Brent Stucker, and Ryan Wicker. Development of a mobile fused deposition modeling system with enhanced manufacturing flexibility. *Journal of materials processing technology*, 211 (3):424–432, 2011.

[9] Wikipedia contributors. Stl (file format) — Wikipedia, the free encyclopedia, 2018. URL https://en.wikipedia.org/w/index.php?title=STL_(file_format)&oldid=830641658. [Online; accessed 7-May-2018].

[10] Paul Wormer. spherical polar coordinates, 2008. URL http://en.citizendium.org/wiki/File:Spherical_polar.png. [Online; accessed 7-May-2018].

[11] Gerald B Sanders and William E Larson. Progress made in lunar in situ resource utilization under nasa's exploration technology and development program. In *Earth and Space 2012: Engineering, Science, Construction, and Operations in Challenging Environments*, pages 457–478. 2012.

[12] Robert P Mueller, Scott Howe, Dennis Kochmann, Hisham Ali, Christian Andersen, Hayden Burgoyne, Wesley Chambers, Raymond Clinton, Xavier De Kestellier, Keye Ebelt, et al. Automated additive construction (aac) for earth and space using in-situ resources. In *Proceedings of the Fifteenth Biennial ASCE Aerospace Division International Conference on Engineering, Science, Construction, and Operations in Challenging Environments (Earth & Space 2016)*. American Society of Civil Engineers, 2016.

[13] Giovanni Cesaretti, Enrico Dini, Xavier De Kestelier, Valentina Colla, and Laurent Pambaguian. Building components for an outpost on the lunar soil by means of a novel 3d printing technology. *Acta Astronautica*, 93:430–450, 2014.

[14] Behrokh Khoshnevis, Dooil Hwang, Ke-Thia Yao, and Zhenghao Yeh. Mega-scale fabrication by contour crafting. *International Journal of Industrial and Systems Engineering*, 1(3):301–320, 2006.

[15] Jing Zhang and Behrokh Khoshnevis. Optimal machine operation planning for construction by contour crafting. *Automation in Construction*, 29:50–67, 2013.

[16] Behrokh Khoshnevis, Melanie Bodiford, Kevin Burks, Ed Ethridge, Dennis Tucker, Won Kim, Houssam Toutanji, and Michael Fiske. Lunar contour crafting-a novel technique for isru-based habitat development. In *43rd AIAA Aerospace Sciences Meeting and Exhibit*, page 538, 2005.

[17] Paul Bosscher, Robert L Williams II, L Sebastian Bryson, and Daniel Castro-Lacouture. Cable-suspended robotic contour crafting system. *Automation in construction*, 17(1): 45–55, 2007.

[18] Steven J Keating, Julian C Leland, Levi Cai, and Neri Oxman. Toward site-specific and self-sufficient robotic fabrication on architectural scales. *Science Robotics*, 2(5): eaam8986, 2017.

[19] Neil Leach, Anders Carlson, Behrokh Khoshnevis, and Madhu Thangavelu. Robotic construction by contour crafting: The case of lunar construction. *International Journal of Architectural Computing*, 10(3):423–438, 2012.

[20] Scott A Howe, Brian H Wilcox, Christopher McQuin, Julie Townsend, Richard R Rieber, Martin Barmatz, and John Leichty. Faxing structures to the moon: Freeform additive construction system (facs). In *AIAA SPACE 2013 Conference and Exposition*, page 5437, 2013.

[21] A Scott Howe, Brian Wilcox, Martin Barmatz, and Gerald Voecks. Athlete as a mobile isru and regolith construction platform. 2016.

[22] Samuel Wilkinson, Josef Musil, Jan Dierckx, Richard Maddock, Xiaoming Yang, Miriam Dall'Igna, Octavian Gheorghiu, and Xavier De Kestelier. Iac-16. d3. 1.10 x35852 preliminary findings from a multi-robot system for large-scale extra-planetary additive construction. 2016.

[23] Graham Hunt, Faidon Mitzalis, Talib Alhinai, Paul A Hooper, and Mirko Kovac. 3d printing with flying robots. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 4493–4499. IEEE, 2014.

[24] Barrie Dams, Sina Sareh, Ketao Zhang, Paul Shepherd, Mirko Kovac, and Richard Ball. Aerial additive building manufacturing: three-dimensional printing of polymer structures using drones. *Proceedings of the Institution of Civil Engineers: Construction Materials*, 2017.

[25] Nathalie Labonnote, Anders Rønnquist, Bendik Manum, and Petra Rüther. Additive construction: State-of-the-art, challenges and opportunities. *Automation in Construction*, 72:347–366, 2016.

[26] D Schrunk, B Sharpe, B Cooper, and M Thangevelu. Book review: The moon: resources, future development and colonization/wiley, 1999. *Journal of the British Astronomical Association*, 109:290, 1999.

[27] Vamsi Krishna Balla, Luke B Roberson, Gregory W O'Connor, Steven Trigwell, Susmita Bose, and Amit Bandyopadhyay. First demonstration on direct laser fabrication of lunar regolith parts. *Rapid Prototyping Journal*, 18(6):451–457, 2012.

[28] Robert P Mueller, Laurent Sibille, Paul E Hintze, Thomas C Lippitt, James G Mantovani, Matthew W Nugent, and Ivan I Townsend. Additive construction using basalt regolith fines. In *Earth and Space 2014*, pages 394–403. 2014.

[29] David Espalin, Jorge Alberto Ramirez, Francisco Medina, and Ryan Wicker. Multi-material, multi-technology fdm: exploring build process variations. *Rapid Prototyping Journal*, 20(3):236–244, 2014.

[30] Stefan Williams, Gamini Dissanayake, and Hugh Durrant-Whyte. Towards terrain-aided navigation for underwater robotics. *Advanced Robotics*, 15(5):533–549, 2001.

[31] Andrea Garulli, Antonio Giannitrapani, Andrea Rossi, and Antonio Vicino. Mobile robot slam for line-based environment representation. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 2041–2046. IEEE, 2005.

[32] Paul Smith, Ian D Reid, and Andrew J Davison. Real-time monocular slam with straight lines. 2006.

[33] Laura A Clemente, Andrew J Davison, Ian D Reid, José Neira, and Juan D Tardós. Mapping large loops with a single hand-held camera. In *Robotics: Science and Systems*, volume 2, 2007.

[34] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Intelligent Robots and Systems' 91.'Intelligence for Mechanical Systems, Proceedings IROS'91. IEEE/RSJ International Workshop on*, pages 1442–1447. Ieee, 1991.

[35] JZ Sasiadek, A Monjazeb, and D Necsulescu. Navigation of an autonomous mobile robot using ekf-slam and fastslam. In *Control and Automation, 2008 16th Mediterranean Conference on*, pages 517–522. IEEE, 2008.

[36] Michael Calonder. Ekf slam vs. fastslam–a comparison. Technical report, 2006.

[37] Joan Sola. Consistency of the monocular ekf-slam algorithm for three different landmark parametrizations. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3513–3518. IEEE, 2010.

[38] Robert Sim, Pantelis Elinas, Matt Griffin, James J Little, et al. Vision-based slam using the rao-blackwellised particle filter. In *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, volume 14, pages 9–16, 2005.

[39] Giorgio Grisettiyz, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2432–2437. IEEE, 2005.

[40] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *The International Journal of Robotics Research*, 25(5-6):403–429, 2006.

[41] Ayman Zureiki and Michel Devy. Slam and data fusion from visual landmarks and 3d planes. *IFAC Proceedings Volumes*, 41(2):14651–14656, 2008.

[42] Ruud M Bolle and David B Cooper. On optimally combining pieces of information, with application to estimating 3-d complex-object position from range data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (5):619–638, 1986.

[43] Joydeep Biswas and Manuela Veloso. Depth camera based localization and navigation for indoor mobile robots. In *RGB-D Workshop at RSS*, volume 2011, page 21, 2011.

[44] Diego Viejo and Miguel Cazorla. 3d plane-based egomotion for slam on semi-structured environment. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2761–2766. IEEE, 2007.

[45] Andrew P Gee, Denis Chekhlov, Andrew Calway, and Walterio Mayol-Cuevas. Discovering higher level structure in visual slam. *IEEE Transactions on Robotics*, 24(5): 980–990, 2008.

[46] Jan Weingarten and Roland Siegwart. 3d slam using planar segments. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3062–3067. IEEE, 2006.

[47] Jose A Castellanos and Juan D Tardos. *Mobile robot localization and map building: A multisensor fusion approach*. Springer Science & Business Media, 2012.

[48] H Jacky Chang, CS George Lee, Yung-Hsiang Lu, and Y Charlie Hu. P-slam: Simultaneous localization and mapping with environmental-structure prediction. *IEEE Transactions on Robotics*, 23(2):281–293, 2007.

[49] Daniel Perea Ström, Fabrizio Nenci, and Cyrill Stachniss. Predictive exploration considering previously mapped environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2761–2766. IEEE, 2015.

[50] Shu Yun Chung and Han Pang Huang. Simultaneous topological map prediction and moving object trajectory prediction in unknown environments. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1594–1599. IEEE, 2008.

[51] SungHwan Ahn, Jinwoo Choi, Nakju Lett Doh, and Wan Kyun Chung. A practical approach for ekf-slam in an indoor environment: fusing ultrasonic sensors and stereo camera. *Autonomous robots*, 24(3):315–335, 2008.

[52] Gabriel Nützi, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Fusion of imu and vision for absolute scale estimation in monocular slam. *Journal of intelligent & robotic systems*, 61(1-4):287–299, 2011.

[53] Xinzheng Zhang, Ahmad B Rad, and Yiu-Kwong Wong. Sensor fusion of monocular cameras and laser rangefinders for line-based simultaneous localization and mapping (slam) tasks in autonomous mobile robots. *Sensors*, 12(1):429–452, 2012.

Appendix A

Printing error

Printing error is used to initialize the plane parameters on the printed segment. This value is measured by the GOM inspection software. A cube with 15mm length edge is at first printed. Then, it is scanned by the 3D scanner and compared with the nominal (CAD) model. After that, both position deviation (Figure A.1) and angle deviation (Figure A.2) can be measured by the GOM software. Finally, the printing error is determined to be:

$$\delta = \begin{bmatrix} 0.015 & 0 & 0 & 0 & 0 \\ 0 & 0.015 & 0 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 0.2 \end{bmatrix}$$

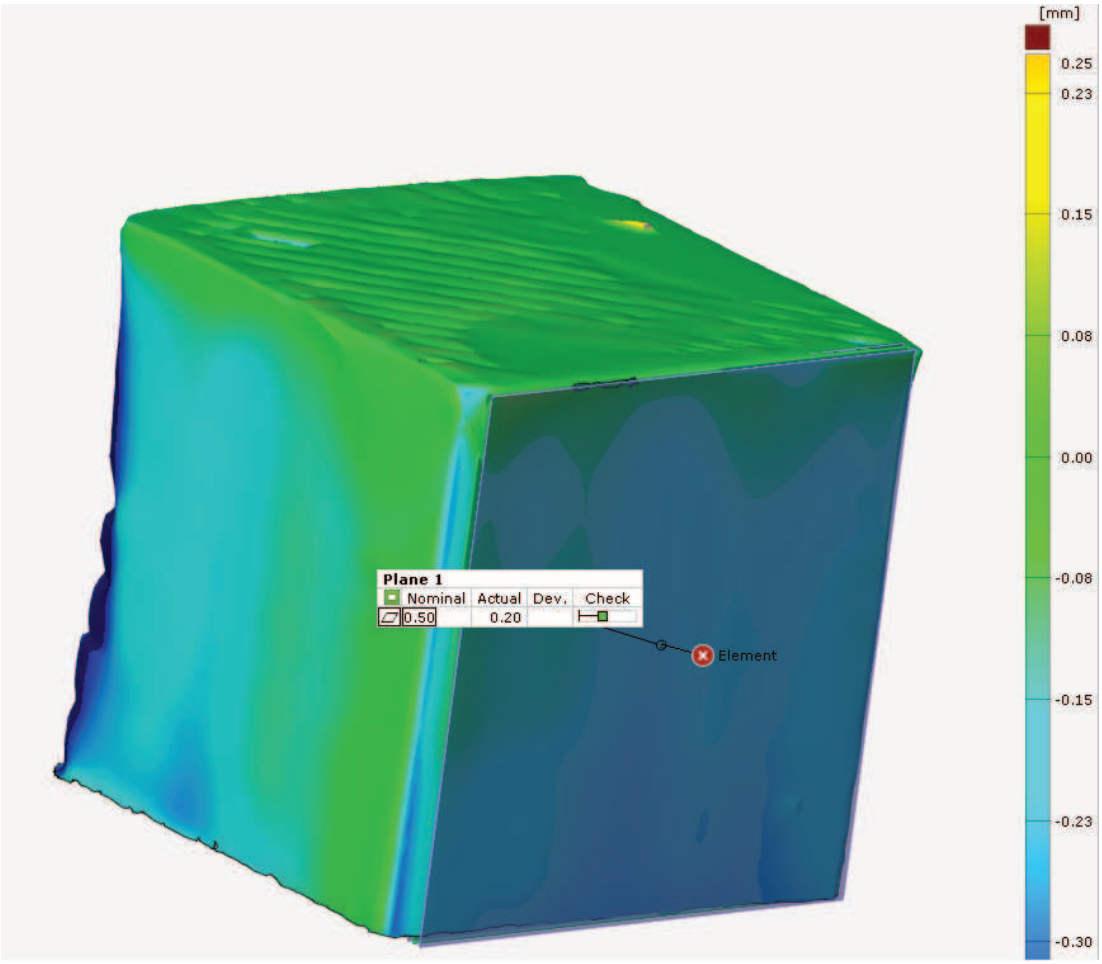

FIGURE A.1: The measurement of plane flatness gives plane position deviation as 0.2mm

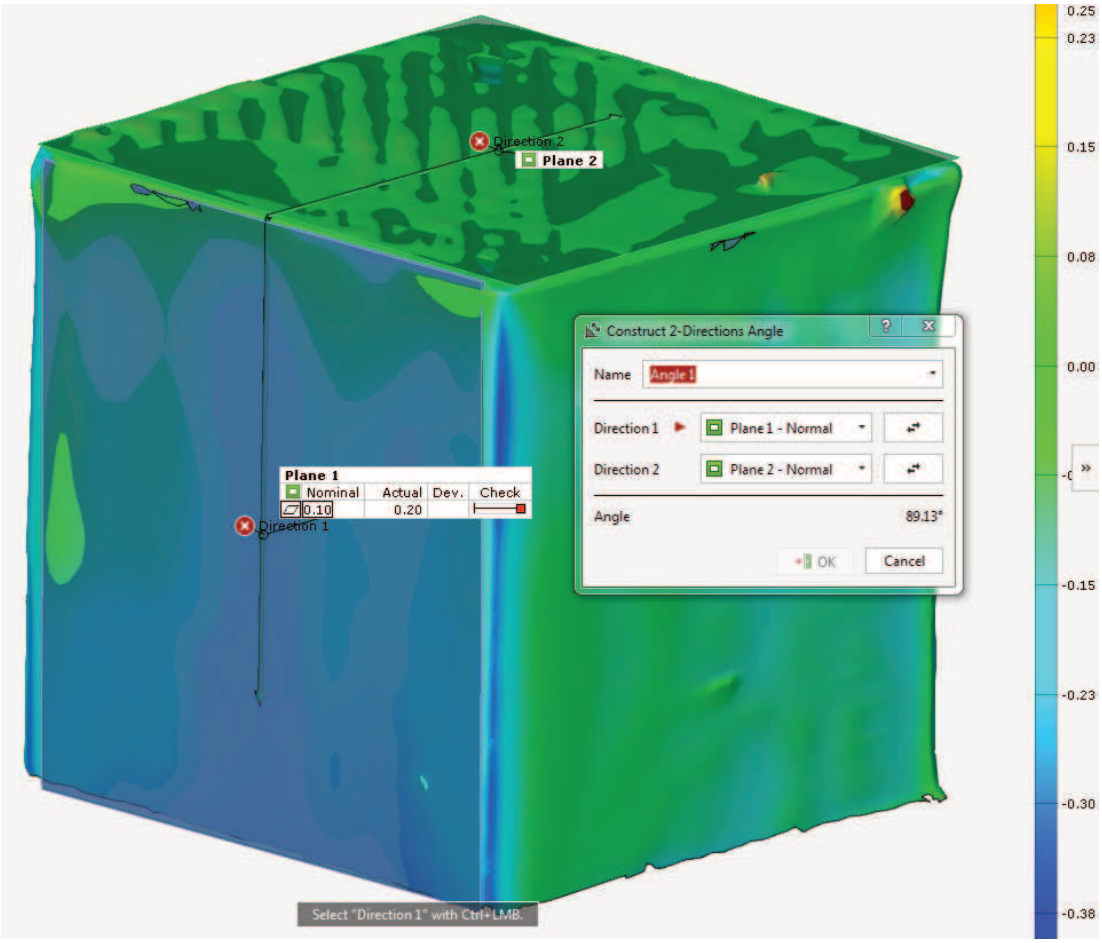

FIGURE A.2: The measurement of plane angle gives plane angle deviation as 0.015rad (0.87°)

Appendix B

Sensitivity of the algorithm to the odometry error

TABLE B.1: Table of distance between final estimate and groud-truth expected position (mm) vs. odometry error (mm)

$\sigma_x^2 = \sigma_y^2$ / Test number	15	30	45	60	100	200	500	1000	1500
1	2.1076	1.9528	1.6733	1.8024	2.1078	2.0881	0.3095	11.5363	8.5024
2	1.8313	2.2715	2.6824	2.0016	1.3589	1.6204	8.6602	1.8049	2.2065
3	1.694	2.3785	1.5156	1.2833	1.7678	2.4269	1.0542	2.3246	8.137
4	1.863	1.7256	2.5495	2.1075	2.3465	8.8335	3.235	27.2307	1.0348
5	1.3241	1.6492	0.8819	1.5791	1.2523	8.8118	1.7489	9.7706	15.9704
6	2.5255	1.9046	0.4134	1.695	2.3309	2.2317	8.7373	8.903	12.5067
7	1.991	1.3223	0.7387	1.6112	1.4458	1.9762	9.5555	8.3795	11.4375
8	1.8452	1.0723	1.9758	1.6913	1.9523	1.5396	3.8255	39.8142	13.3116
9	1.3217	1.9972	1.662	1.5144	1.9871	0.2936	9.4922	1.8257	10.8146
10	1.8846	1.1715	1.6193	2.5802	2.0349	1.9404	8.5754	15.0762	42.2909
Average $\bar{d}$	1.8388	1.74455	1.57119	1.7866	1.85843	3.17622	5.51937	12.66657	12.62124
Standard deviation	0.3195	0.4023	0.6667	0.3300	0.3536	2.7432	3.4512	11.0354	10.3417

Appendix C

Code

C.1 Main

```
clc
clear
close all
%%%%%%%%%%%%%%%define the involved file names%%%%%%%%%%%%%%%
%STL file name
name_stl='thin_curve';
%total station measurement file name
name_data='landmark test';
%3D scanner measurement file name
ReadName='p6_1.ply';
%min point number in isolated planes (isolated planes with less point will be regarded as redundency and be
    eliminated)
cloudpointnum=500;
%result of total station calibration (calculated from totalstationcalibration() program)
x_t1_p1=80.4517;
y_t1_p1=-37.9460;
R_t1_p1=-2.0834;
%total station to printer transformation matrix
trans_total_printer=[cos(R_t1_p1),-sin(R_t1_p1),x_t1_p1;
                     sin(R_t1_p1),cos(R_t1_p1),y_t1_p1;0,0,1];
%introduced odometry error (for the experiment testing sensitivity of the algorithm to odometry error)
R=10;
%odometry: printer pose (x_p(mm), y_p(mm), theta_p(rad) measured by hand)
bot_pos=[
0     0     degtorad(90);
```

```
80    20     degtorad(225);
80    20     degtorad(225);
120   30     degtorad(225);
120   30     degtorad(225);
160   50     degtorad(225);
160   50     degtorad(225);
200   50     degtorad(225);
200   50     degtorad(225);
240   65     degtorad(225);
% 240   60     degtorad(225);
% 280   0     degtorad(225);
% 280   0     degtorad(225);
% 320   0     degtorad(225);
% 320   0     degtorad(225);
];
%introduce the odometry error only after moving the printer
if size(bot_pos,1)>1
    figure(5)
    hold on
    scatter(bot_pos(end,1),bot_pos(end,2),'x','g');
    hold off
    bot_pos(end,1:2)=bot_pos(end,1:2)+[normrnd(0,R),normrnd(0,R)];
end
%check if the filename of 3D scanning matches the odometry line number
pos_num=str2double(ReadName(2));
test_num=str2double(ReadName(end-4));
if pos_num==1
    row_num=1;
else
    row_num=pos_num*2+test_num-3;
end
if row_num~=size(bot_pos,1)
    error('error:odometry and point cloud filename doesnt match');
end
%read total station measurement
[ ts_mea_2,ts_mea_1 ] = total_station_2(name_data);
%define the name of txt file for reading mu and sigma from last loop
[ mu_read_name, sigma_read_name ] = reading_file_name( bot_pos );
%define the initial sigma for printer pose, landmark position and plane parameters
sigma_planes_p1=diag([0.1,0.1,degtorad(0.1),0.1,0.1,0.1,0.1,0.1,0.1,0.1]);
sigma_planes_mov=diag([3+R,3+R,degtorad(3),0.1,0.1,0.1,0.1,0.1,0.1]);
printing_error=[0.015,0.015,0.2,0.2,0.2];
printing_error_ramp=[0.015,0.015,0.5,0.2,0.2];
sz=100;
%p1_n: initial printer pose
```

```
if pos_num==1
    %STL to planes
    ramp_plane=[0.3420,0,0.9397,-20.5200]';
    [ cad_segment_post_print ] = stltoplanes(name_stl,ramp_plane');
    cad_num=size(cad_segment_post_print,1);
    %add landmark positions into mu
    printer_mea_1=trans_total_printer*ts_mea_1;
    mu=[bot_pos(end,:)';printer_mea_1(1:2,1);printer_mea_1(1:2,2);printer_mea_1(1:2,3)];
    mu=[mu;zeros(cad_num*5,1)];
    %add measured planes into mu
    for i=1:cad_num
        mu(5*i+5:5*i+9,1) = [acos(cad_segment_post_print(i,3))/sqrt(cad_segment_post_print(i,1)^2+
     cad_segment_post_print(i,2)^2+cad_segment_post_print(i,3)^2),atan2(cad_segment_post_print(i,2),
     cad_segment_post_print(i,1)),cad_segment_post_print(i,7:9)]';
        if i==cad_num
            mu(5*i+7)=mu(5*i+7);
        end
    end
    %initialize sigma for each planes
    error_plane=diag(printing_error);
    ACell = repmat({error_plane}, 1, cad_num);
    ACell{1,end}=diag(printing_error_ramp);
    error_plane_small = blkdiag(ACell{:});
    %expand sigma
    sigma=[sigma_planes_p1,zeros(size(sigma_planes_p1,1),cad_num*5);
           zeros(cad_num*5,size(sigma_planes_p1,1)),error_plane_small];
    length=sqrt(abs(tan(mu(3)))^2+0.6^2);
    %plot printer pose and landmark positions in global frame
    figure(1)
    hold on
    if wrapToPi(mu(3))<pi/2&&wrapToPi(mu(3))>-pi/2
        plot([mu(1),(mu(1)+80/length)],[mu(2),(mu(2)+80/length*tan(mu(3)))],'g');
    elseif wrapToPi(mu(3))==pi/2||wrapToPi(mu(3))==-pi/2
        plot([mu(1),mu(1)],[mu(2),(mu(2)+80)],'g');
    else
        plot([mu(1),(mu(1)-80/length)],[mu(2),(mu(2)-80/length*tan(mu(3)))],'g');
    end
    scatter(mu(1),mu(2),sz,'g');
    scatter([mu(4),mu(6),mu(8)],[mu(5),mu(7),mu(9)],'filled');
    hold off
%pn_1: right after moving the priner
elseif test_num==1
    %the first scan right after moving printer
    mu=dlmread(mu_read_name);
    mu(1:3)=bot_pos(end,:)';
```

```
sigma=dlmread(sigma_read_name);
sigma(1:3,1:3)=sigma_planes_mov(1:3,1:3);
%initialize sigma
ramp_plane=[sin(mu(end-4))*cos(mu(end-3));sin(mu(end-4))*sin(mu(end-3));cos(mu(end-4));abcxyz2abcd( sin(mu
 (end-4))*cos(mu(end-3)), sin(mu(end-4))*sin(mu(end-3)), cos(mu(end-4)), mu(end-2), mu(end-1), mu(end) )
 '];
[ cad_segment_post_print ] = stltoplanes(name_stl,ramp_plane');
%plot the error ellipse of printer position in the global frame
figure(5)
axis equal
hold on
scatter(mu(1),mu(2),'.','b');
error_ellipse(sigma(1:2,1:2)^2,mu(1:2),'conf',0.95);
hold off
%ts_mea_1 is the first set of total station measurements
%ts_mea_2 is the last set of total station measurements
%both of them are required to transform into the printer frame
printer_mea_1=trans_total_printer*ts_mea_1;
printer_mea_2=trans_total_printer*ts_mea_2;
%plot robot position and landmark positions in the global frame
figure(2)
title('Global frame')
xlabel('mm')
ylabel('mm')
hold on
tmpAspect=daspect();
%keep x&y axes in same scale
daspect(tmpAspect([1 2 2]))
scatter(bot_pos(1,1),bot_pos(1,2),sz,'b');
%plot direction of printer
length=sqrt(abs(tan(bot_pos(1,3)))^2+0.6^2);
if bot_pos(1,3)<pi/2
    plot([bot_pos(1,1),(bot_pos(1,1)+80/length)],[bot_pos(1,2),(bot_pos(1,2)+80/length*tan(bot_pos(1,3)))
 ],'b');
elseif bot_pos(1,3)>pi/2
    plot([bot_pos(1,1),(bot_pos(1,1)-80/length)],[bot_pos(1,2),(bot_pos(1,2)-80/length*tan(bot_pos(1,3)))
 ],'b');
elseif bot_pos(1,3)==pi/2||bot_pos(1,3)==-pi/2
    plot([bot_pos(1,1),bot_pos(1,1)],[bot_pos(1,2),(bot_pos(1,2)+80)],'b');
end
scatter(printer_mea_1(1,1:3),printer_mea_1(2,1:3),'filled');

%define total station measurement error
m_error=diag([20,4.7124e-06]);
```

```
    %SLAM1: total station SLAM
    [ mu, sigma ] = SLAM1( printer_mea_2, mu, sigma, m_error );
    sigma=abs(sigma);

    %plot printer pose after SLAM1
    length=sqrt(abs(tan(mu(3)))^2+0.6^2);
    if wrapToPi(mu(3))<pi/2&&wrapToPi(mu(3))>-pi/2
        plot([mu(1),(mu(1)+80/length)],[mu(2),(mu(2)+80/length*tan(mu(3)))],'g');
    elseif wrapToPi(mu(3))==pi/2||wrapToPi(mu(3))==-pi/2
        plot([mu(1),mu(1)],[mu(2),(mu(2)+80)],'g');
    else
        plot([mu(1),(mu(1)-80/length)],[mu(2),(mu(2)-80/length*tan(mu(3)))],'g');
    end
    scatter(mu(1),mu(2),sz,'g');

    %plot printer pose from odometry
    if bot_pos(end,3)<pi/2&&wrapToPi(bot_pos(end,3))>-pi/2
        plot([bot_pos(end,1),(bot_pos(end,1)+80/length)],[bot_pos(end,2),(bot_pos(end,2)+80/length*tan(bot_pos
     (end,3)))],'r');
    elseif wrapToPi(bot_pos(2,3))==pi/2||wrapToPi(bot_pos(2,3))==-pi/2
        plot([bot_pos(end,1),bot_pos(end,1)],[bot_pos(end,2),(bot_pos(end,2)+80)],'r');
    else
        plot([bot_pos(end,1),(bot_pos(end,1)-80/length)],[bot_pos(end,2),(bot_pos(end,2)-80/length*tan(bot_pos
     (end,3)))],'r');
    end
    scatter(bot_pos(end,1),bot_pos(end,2),sz,'r');

    %plot updated landmarks
    scatter([mu(4),mu(6),mu(8)],[mu(5),mu(7),mu(9)],'filled');



    disp('SLAM1 robot pose (mm)')
    disp([mu(1:3)'])
%pn_2: right after printing
else
    %the scan right after printing
    mu_old=dlmread(mu_read_name);
    mu_old_matrix=[];
    for i=1:(size(mu_old,1)-9)/5
        %previous plane parameters
        mu_old_matrix=[mu_old_matrix;[sin(mu_old(5*i+5))*cos(mu_old(5*i+6));sin(mu_old(5*i+5))*sin(mu_old(5*i
     +6));cos(mu_old(5*i+5));abcxyz2abcd( sin(mu_old(5*i+5))*cos(mu_old(5*i+6)), sin(mu_old(5*i+5))*sin(mu_old
     (5*i+6)), cos(mu_old(5*i+5)), mu_old(5*i+7), mu_old(5*i+8), mu_old(5*i+9) )]'];
    end
    sigma_old=dlmread(sigma_read_name);
```

```
ramp_plane=[sin(mu_old(end-4))*cos(mu_old(end-3));sin(mu_old(end-4))*sin(mu_old(end-3));cos(mu_old(end-4))
 ;abcxyz2abcd( sin(mu_old(end-4))*cos(mu_old(end-3)), sin(mu_old(end-4))*sin(mu_old(end-3)), cos(mu_old(
 end-4)), mu_old(end-2), mu_old(end-1), mu_old(end) )'];
%new mu is new cad_segment + previous cad_segment
[ cad_segment_pre_print ] = stltoplanes(name_stl,ramp_plane');
%predicted plane parameters
[ cad_segment_post_print ] = stltoplanes(name_stl,[ramp_plane(1:3);ramp_plane(4)-40*ramp_plane(1)]');
%distinguish out updated planes and non-updated planes
[C2,ia2,ib2] = intersect(round(cad_segment_post_print(:,1:6),5),round(cad_segment_pre_print(:,1:6),5),'
 rows','stable');
all_planes=1:size(cad_segment_post_print,1);
%'1'=changed planes, '0'=unchanged planes
changed_planes=~ismember(all_planes,ia2);
mu=mu_old(1:9);
cad_num=size(cad_segment_post_print,1);
error_large=[];
for i=1:cad_num
    if changed_planes(i)==1;
        error_large = [error_large,0.015,0.05,1,1,0.2];
    else
        error_large = [error_large,[0.005,sigma_old(3,3),sigma_old(1,1),sigma_old(2,2),0]+printing_error];
    end
end
error_diag=diag(error_large);
sigma=[abs(sigma_old(1:9,1:9)),zeros(9,cad_num*5);
       zeros(cad_num*5,9),error_diag];
%combinations of all not updated planes
c_a = combnk(ia2,2);
c_b = combnk(ib2,2);
j=1;
if isempty(c_a)||isempty(c_b)
    sigma(5+5*ia2:9+5*ia2,5+5*ia2:9+5*ia2)=sigma_old(5+5*ib2:9+5*ib2,5+5*ib2:9+5*ib2);
    sigma(1:5,5+5*ia2:9+5*ia2)=sigma_old(1:5,5+5*ib2:9+5*ib2);
    sigma(5+5*ia2:9+5*ia2,1:5)=sigma_old(5+5*ib2:9+5*ib2,1:5);
else
    for i=1:size(c_a,1)
        sigma(5+5*c_a(i,1):9+5*c_a(i,1),5+5*c_a(i,2):9+5*c_a(i,2))=sigma_old(5+5*c_b(i,1):9+5*c_b(i,1)
 ,5+5*c_b(i,2):9+5*c_b(i,2));
        sigma(1:5,5+5*c_a(i,1):9+5*c_a(i,1))=sigma_old(1:5,5+5*c_b(i,2):9+5*c_b(i,2));
        sigma(5+5*c_a(i,2):9+5*c_a(i,2),1:5)=sigma_old(5+5*c_b(i,1):9+5*c_b(i,1),1:5);
    end
end
for i=1:size(cad_segment_post_print,1)
    %changed planes
    if changed_planes(i)==1
```

```
            mu=[mu;abcd2abcxyz(cad_segment_post_print(i,:))];
        %unchanged planes
        else
            mu=[mu;mu_old(5*ib2(j)+5:5*ib2(j)+9)];
            j=j+1;
        end
    end
end
%%%%%%%%%%%%%%%point cloud process%%%%%%%%%%%%%
check_error=1;
same_plane_error=0;
%if there are multiple side planes with simiar angle
while check_error==1
    %clear plot
    clf(figure(3))
    clf(figure(4))
    %pcfitplane tolerance is increased
    same_plane_error=same_plane_error+0.05;
    if same_plane_error>0.2
        warning('with large pcfitplane tolerance, there are still planes with similar directions');
        break;
    end
    %fit planes from the 3D scanner measurement, and the output is measured planes in printer frame
    [ para ] = point_cloud_process( ReadName, mu, cloudpointnum,same_plane_error);
    %find correspondence between measured planes and corresponding planes in the state vector
    [order_para, check_error]=plane_correspondence( mu, bot_pos, para );
end
% #test program# calculate distance between planes and orgin point as a reference
if size(bot_pos,1)~=1
    for i=1:size(order_para,1)
        if round(abs(order_para(i,3)),2)<0.97&&round(abs(order_para(i,3)),2)>0.3
            [I1,~]=plane_line_intersect(order_para(i,1:3),[-order_para(i,4)/order_para(i,1)
     ,0,0],[0,0,0],[-1,-1,0]);
        elseif round(abs(order_para(i,3)),2)<0.5
            [I2,~]=plane_line_intersect(order_para(i,1:3),[-order_para(i,4)/order_para(i,1)
     ,0,0],[0,0,0],[-1,1,0]);
        end
    end
    sqrt(I1(1)^2+I1(2)^2)
    sqrt(I2(1)^2+I2(2)^2)
    figure(4)
    hold on
    plot3([0,50],[0,50],[0,0],'r');
    scatter3(I1(1),I1(2),I1(3),'filled','k');
    hold off
```

```
end
%plot error ellipse of printer position before running SLAM algorithm
figure(5)
title('Error ellipses in global frame')
axis equal
hold on
scatter(mu(1),mu(2),'.','b');
error_ellipse(sigma(1:2,1:2)^2,mu(1:2),'conf',0.95);
hold off

%SLAM2
plane_num=(size(mu,1)-9)/5;
[ mu, sigma ] = SLAM2_thetaphi2( plane_num, order_para, mu, sigma, bot_pos);

%Pn_1 right after moving the printer
if test_num==1
    %generate STL file for printing the next piece based on the updated ramp plane
    %convert mu to mu2 in order to represent it in the previous way
    %convert ramp plane into general form
    i=(size(mu,1)-9)/5;
    ramp_para=[sin(mu(5*i+5))*cos(mu(5*i+6));sin(mu(5*i+5))*sin(mu(5*i+6));cos(mu(5*i+5));abcxyz2abcd( sin(mu
     (5*i+5))*cos(mu(5*i+6)), sin(mu(5*i+5))*sin(mu(5*i+6)), cos(mu(5*i+5)), mu(5*i+7), mu(5*i+8), mu(5*i+9) )
     '];
    %ramp plane for segmenting the whole CAD model
    center_piece=stl_generate_polygon(ramp_para,name_stl);
    %plot the center of next piece to be printed
    figure(2)
    hold on
    sz=10;
    scatter(center_piece(1),center_piece(2),sz,'b','filled')
    hold off
    %calculate the pose of the next piece to be printed in the local frame
    %bearing
    rho=sqrt((center_piece(1)-mu(1))^2+(center_piece(2)-mu(2))^2);
    %range
    theta=mu(3)+atan2(center_piece(2)-mu(2),center_piece(1)-mu(1));
    %position of next segment in robot frame
    [x_print,y_print] = pol2cart(theta,rho);
    %direction of next segment in robot frame
    theta_print=-mu(3)-3*pi/2;
    %the number that need to be imported into printing software
    print_pose=[x_print,y_print,theta_print,radtodeg(theta_print)+360];
    disp('    x(mm)     y(mm)     theta    degree')
    disp(print_pose)
end
```

```
%save mu and sigma value for next mobile 3D printing block
s1='mu_file_';
C=strsplit(ReadName,'.');
s2=C(1);
s3='.txt';
name_mu=char(strcat(s1,s2,s3));
s4='sigma_file_';
name_sigma=char(strcat(s4,s2,s3));
dlmwrite(name_mu,mu,'delimiter','\t','precision',10);
dlmwrite(name_sigma,sigma,'delimiter','\t','precision',10);
```

C.2 Point cloud process

```
function [ para ] = point_cloud_process( ReadName, mu, cloudpointnum, same_plane_error)
%POINT_CLOUD_PROCESS Summary of this function goes here
%   Detailed explanation goes here
strPath = 'C:\Users\georgeli\Desktop\project';
strFull = fullfile(strPath,ReadName);
cap = pcread(strFull);

%plot pre-processing point cloud
figure(3)
hold on
subplot(3,3,9)
xlabel('X')
ylabel('Y')
zlabel('Z')
pcshow(cap)

%down sample the point cloud
gridStep = 0.2;
downsampled = pcdownsample(cap,'gridAverage',gridStep);

[model1,inlierIndices,outlierIndices] = pcfitplane(downsampled,0.6);
bot_plane = select(downsampled,inlierIndices); %select bottom pc
rem1 = select(downsampled,outlierIndices);
rem_bot = select(downsampled,inlierIndices);

[model2,inlierIndices,outlierIndices] = pcfitplane(rem1,0.4);
top_plane = select(rem1,inlierIndices); %select top pc
rem2 = select(rem1,outlierIndices);
```

```
[model3,inlierIndices,outlierIndices] = pcfitplane(rem2,0.2);
ramp_plane = select(rem2,inlierIndices); %select ramp pc

%ROI is defined by the boundary of top pc
roi=[top_plane.XLimits(1)-10,top_plane.XLimits(2)+20;
     top_plane.YLimits(1)-10,top_plane.YLimits(2)+10;
     -inf,inf];
%crop an ROI from the current pc
indices = findPointsInROI(cap, roi);
downsampled = select(cap,indices);

if model1.Normal(3)<0
    normaltobottom = -model1.Normal;
else
    normaltobottom = model1.Normal;
end

%%%%%%%%%%%%translate%%%%%%%%%%%%
%3D scanner calibration function, gives position offset and angle offset of the 3D scanner as output
[center_printer,angle_error] = center_evaluate_function;
disp('position offset');
disp(center_printer);
disp('angle offset');
disp(angle_error);
if abs(angle_error)>2
    error('Error. Angle error from 3D scanner calibration too large')
end
angle_error=120+angle_error;
tranz=[1 0 0 0;0 1 0 0; 0 0 1 0; -center_printer(1) -center_printer(2) -center_printer(3) 1];
rottranf = affine3d(tranz);
ptCloudTranslated = pctransform(downsampled,rottranf);
%%%%%%%%%%rotate around x and y axis%%%%%%%%%%
xaxis = [1 0 0];
yaxis = [0 1 0];

a = atan2(norm(cross(normaltobottom,xaxis)), dot(normaltobottom,xaxis));
b = atan2(norm(cross(normaltobottom,yaxis)), dot(normaltobottom,yaxis));

tranx = [
    1   0   0   0;
    0   cos(b-degtorad(90))  -sin(b-degtorad(90))  0;
    0   sin(b-degtorad(90))  cos(b-degtorad(90))  0;
    0   0   0   1;
    ];% matrix to rotate about x axis
```

```
trany = [
    cos(a-degtorad(90))  0   -sin(a-degtorad(90))  0;
    0   1   0   0;
    sin(a-degtorad(90)) 0   cos(a-degtorad(90))  0;
    0   0   0   1;
    ];% matrix to rotate about y axis

angtransf = affine3d(tranx);
ptCloudTformed_1 = pctransform(ptCloudTranslated,angtransf);
angtransf = affine3d(trany);
ptCloudTformed = pctransform(ptCloudTformed_1,angtransf);
%%%%%%%%%%%rotate around z axis%%%
theta = deg2rad(angle_error);  %%camera angle isn't exact 120 degree as designed

tranz = [
cos(theta)  -sin(theta) 0   0;
sin(theta)  cos(theta)  0   0;
0   0   1   0
0   0   0   1;
];% matrix to transform about z axis
rottranf = affine3d(tranz);
ptCloudTformed = pctransform(ptCloudTformed,rottranf);

%final adjust, move bottom plane to z=0 plane
%there is confusion between top and bottom, so we need to make sure we
%adjust z based on bottom plane
[model1,inlierIndices,outlierIndices] = pcfitplane(ptCloudTformed,0.4,[0,0,1]);
in1 = select(ptCloudTformed,inlierIndices); %select TOP ground pc
rem = select(ptCloudTformed,outlierIndices); %select TOP ground pc
[model2,inlierIndices,outlierIndices] = pcfitplane(rem,0.4,[0,0,1]);
in2 = select(rem,inlierIndices); %select TOP ground pc
if abs(model1.Parameters(4))>abs(model2.Parameters(4))
    model1=model2; %always adjust by bottom plane
end

model1.Parameters %pre-adjustment bottom plane
tranz=[1 0 0 0;0 1 0 0;0 0 1 0; 0 0 -abs(model1.Parameters(4)) 1];
rottranf = affine3d(tranz);
ptCloudTformed2 = pctransform(ptCloudTformed,rottranf);
[model1,inlierIndices,outlierIndices] = pcfitplane(ptCloudTformed2,0.4);
model1.Parameters %post-adjustment bottom plane

%%%%%%%%%%%%%%%%%%isolate planes%%%%%%%%%%%%%%%%%
r_angle=radtodeg(mu(3));
x_dist=mu(1);
```

```
y_dist=mu(2);
%%%%%%%%%%%%%%
i=1;
lock_top=0;
lock_ramp=0;
para=zeros(10,7);
while(1)
    [model,inlierIndices,outlierIndices,meanError] = pcfitplane(ptCloudTformed2,0.4);
    %if the remaining pc number is less than cloudpointnum, break from the loop
    if size(inlierIndices,1)<cloudpointnum
        break;
    end
    remain_pc = select(ptCloudTformed2,outlierIndices);
    choice = select(ptCloudTformed2,inlierIndices);
    ptCloudTformed2=remain_pc;
    A = [cosd(r_angle-90) sind(r_angle-90) 0 0; ...
     -sind(r_angle-90) cosd(r_angle-90) 0 0; ...
     0 0 1 0; ...
     x_dist y_dist 0 1];
    tform = affine3d(A);
    ptCloudA = pctransform(choice,tform);
    ptCloudB = pcdenoise(choice,'Threshold',1);
    %if plane is too small or get bottom plane once again, ignore and stop searching
    para(i,1:4)=model.Parameters;
    figure(3)
    hold on
    subplot(3,3,i)
    pcshow(choice)
    centroid_allp=[mean(ptCloudB.Location),mean(ptCloudB.Location),mean(ptCloudB.Location)];
    centroid_pc=[mean(ptCloudB.XLimits),mean(ptCloudB.YLimits),mean(ptCloudB.ZLimits)];
    centroid_adj=plane_line_intersect([model.Parameters(1),model.Parameters(2),model.Parameters(3)],[0,0,-
     model.Parameters(4)/model.Parameters(3)],centroid_pc,centroid_pc+[model.Parameters(1),model.Parameters(2)
     ,0]);
    %plot each planar patches and their centroids
    if round(abs(para(i,3)),2)<0.98&&round(abs(para(i,3)),2)>0.3
        figure(4)
        hold on
        title('ramp analyse')
        pcshow(choice)
        y=10:.1:30;
        z=0:.1:7.3;
        [Y,Z] = meshgrid(y,z);
        a1=model.Parameters(1); b1=model.Parameters(2); c1=model.Parameters(3); d1=-model.Parameters(4);
        X=(d1- b1 * Y - c1 * Z)/a1;
        shading flat
```

```
        xlabel('x'); ylabel('y'); zlabel('z')
        scatter3(centroid_allp(1),centroid_allp(2),centroid_allp(3),'filled','m');
    end
    %subplot planar patches in different
    figure(3)
    hold on
    title(['plane ' num2str(i)])
    hold off
    para_pos_g(i,1:2)=ptCloudA.XLimits;
    para(i,5:7)=centroid_adj;
    i=i+1;
    if abs(round(model.Parameters(3),3))>0.95&&abs(round(model.Parameters(4),3))>0.5&&lock_top==0
        save_top=choice;
        lock_top=1;
    elseif i-1>2&&abs(round(model.Parameters(3),3))<1&&abs(round(model.Parameters(3),3))>0.5&&lock_ramp==0
        save_ramp=choice;
        lock_ramp=1;
    end
end

%transform top point cloud into global
A = [cosd(r_angle-90) sind(r_angle-90) 0 0; ...
 -sind(r_angle-90) cosd(r_angle-90) 0 0; ...
 0 0 1 0; ...
 x_dist y_dist 0 1];
tform = affine3d(A);
save_top_g = pctransform(save_top,tform);
save_ramp_g = pctransform(save_ramp,tform);

element_segment=find(ismember(para(:,1:4), [0,0,0,0], 'rows'),1);
para(element_segment:end,:)=[];

%transform plane from local to global
para_g=[cosd(r_angle-90) -sind(r_angle-90) 0 0; ...
sind(r_angle-90) cosd(r_angle-90) 0 0; ...
0 0 1 0; ...
x_dist y_dist 0 1]*para(:,1:4)';
para_g=para_g';

%if measured planes have very close angle, then one of them will be removed
flag=0;
before_para_size=size(para,1);
for i=4:size(para,1)-1
    for j=1:size(para,1)-i
        abs(para(i,1)-para(i+j,1))
```

```
        if abs(para(i,1)-para(i+j,1))<same_plane_error
            if para(i,1)<para(i+j,1)
                para(i,:)=[];
                flag=1;
                break
            else
                para(i+j,:)=[];
                flag=1;
                break
            end
        end
    end
    if flag==1
        break
    end
end
after_para_size=size(para,1);
if after_para_size<before_para_size
    warning('two similar planes have been combined');
else
    warning('no similar planes have been combined');
end
%%%%%%%%%%%%%%%%%%unify plane direction%%%%%%%%%%%%%%%%%%%%%%
%top, bottom and ramp planes normal vector z>0
%side planes normal vector points outside of model
if exist('save_top','var')==0
   error('Error. No top planes detected. height error too big')
end
save_area=[];
save_top_ds = pcdownsample(save_top_g,'random',0.01);
save_ramp_ds = pcdownsample(save_ramp_g,'random',0.01);
for i=1:1:size(para,1)
    p_direction=0;
    if i>3
        if abs(abs(para(i,1))-1)<0.01
            if para(i,1)>0
                para(i,1:4)=-para(i,1:4);
            end
        else
            %check most of the points in point cloud is above plane or under plane
            if (para_pos_g(i,1)+para_pos_g(i,2))/2<save_ramp_ds.XLimits(1)
                %compare top
                for j=1:1:size(save_top_ds.Location,1)
                    if abs(save_top_ds.Location(j,1)-(para_pos_g(i,1)+para_pos_g(i,2))/2)<1
                        save_area=[save_area;save_top_ds.Location(j,1:2)];
```

```
                    end
                end
            else
                %compare ramp
                for j=1:1:size(save_ramp_ds.Location,1)
                    if abs(save_ramp_ds.Location(j,1)-(para_pos_g(i,1)+para_pos_g(i,2))/2)<1
                        save_area=[save_area;save_ramp_ds.Location(j,1:2)];
                    end
                end
            end
            for j=1:size(save_area,1)
                if save_area(j,2)>(-para_g(i,1)/para_g(i,2))*save_area(j,1)-para_g(i,4)/para_g(i,2)
                    p_direction=p_direction+1;
                else
                    p_direction=p_direction-1;
                end
            end
        end
    elseif para(i,3)<0
        para(i,1:4)=-para(i,1:4);
    end
    if abs(para(i,3))>0.5
        if para(i,3)<0
            para(i,1:4)=-para(i,1:4);
        end
    elseif (p_direction<0&&para_g(i,2)<0)||(p_direction>0&&para_g(i,2)>0)
        para(i,1:4)=-para(i,1:4);
    end
end

end
```

C.3 STL to planes

```
function [ save_segment ] = stltoplanes(name_stl,slice_plane)
%STLTOPLANES_F Summary of this function goes here
    filename=strcat('C:\Users\georgeli\Desktop\project\',name_stl,'.stl');
    [F,V,N] = stlread(filename);

    num=size(N,1);
    save_plane=zeros(num,10);
    for i=1:1:num
```

```
    d=-N(i,1)*V(i*3,1)-N(i,2)*V(i*3,2)-N(i,3)*V(i*3,3);
    plane_para=[round(N(i,1),4),round(N(i,2),4),round(N(i,3),4),round(d,4),V(3*i-2,:)];
    plane_limit=[min([V(i*3-2,1),V(i*3-1,1),V(i*3,1)]),max([V(i*3-2,1),V(i*3-1,1),V(i*3,1)])];
    save_plane(i,1:4)=plane_para(1:4);
    save_plane(i,5:6)=plane_limit;
    save_plane(i,7:9)=plane_para(5:7);
    if N(i,2)>0
        save_plane(i,10)=1;
    elseif N(i,2)<0
        save_plane(i,10)=-1;
    end
end
[~,ia,~]=unique(save_plane(:,1:4),'rows');
save_plane=save_plane(ia,:);

%%%%%slice%%%%
model_length=max(V(:,1));
left=0;
p1=planeModel(slice_plane');
p2=planeModel([0,0,1,0]);
p4=planeModel([0,0,1,-7.3]);
%calculate intersection on positive direction side and negative direction side
for i=1:size(save_plane,1)
    p3=planeModel(save_plane(i,1:4));
    if save_plane(i,end)==-1
        intersect_r_neg=three_plane_intersect_y(p1,p2,p3);
        if intersect_r_neg(1)<=save_plane(i,6)&&intersect_r_neg(1)>=save_plane(i,5)
            x_r_neg=intersect_r_neg;
        end
        intersect_l_neg=three_plane_intersect_y(p1,p4,p3);
        if intersect_l_neg(1)<=save_plane(i,6)&&intersect_l_neg(1)>=save_plane(i,5)
            x_l_neg=intersect_l_neg;
        end
    elseif save_plane(i,end)==1
        intersect_r_pos=three_plane_intersect_y(p1,p2,p3);
        if intersect_r_pos(1)<=save_plane(i,6)&&intersect_r_pos(1)>=save_plane(i,5)
            x_r_pos=intersect_r_pos;
        end
        intersect_l_pos=three_plane_intersect_y(p1,p4,p3);
        if intersect_l_pos(1)<=save_plane(i,6)&&intersect_l_pos(1)>=save_plane(i,5)
            x_l_pos=intersect_l_pos;
        end
    end
end
save_segment=zeros(num,10);
```

```
unit=zeros(1,6);
%while right<model_length+segment_length
n=1;
%%%%%%segment bottom and top planes
save_segment(n,:)=-save_plane(find(ismember(save_plane(:,1:3), [0,0,-1], 'rows'),1),:); %'-'in order to
 match scan
save_segment(n,5:6)=[0,max([x_r_neg,x_r_neg])]; %'-'in order to match scan
n=n+1;
save_segment(n,:)=save_plane(find(ismember(save_plane(:,1:3), [0,0,1], 'rows'),1),:);
save_segment(n,5:6)=[0,max([x_l_neg,x_l_neg])];
n=n+1;
%%%%%%segment side planes
for i=1:1:size(save_plane,1)
    if  save_plane(i,3)==0
        if save_plane(i,end)<=0&&(save_plane(i,5)<x_r_neg(1)||save_plane(i,6)<x_r_neg(1))
            save_segment(n,:)=save_plane(i,:);
            if save_segment(n,6)>x_r_neg(1)
                save_segment(n,5:6)=[save_plane(i,5),x_r_neg(1)];
            end
            n=n+1;
        elseif save_plane(i,end)>=0&&(save_plane(i,5)<x_r_pos(1)||save_plane(i,6)<x_r_pos(1))
            save_segment(n,:)=save_plane(i,:);
            if save_segment(n,6)>x_r_pos(1)
                save_segment(n,5:6)=[save_plane(i,5),x_r_pos(1)];
            end
            n=n+1;
        end
    end
end
%%%%%%segment right side plane
if x_r_pos(1)>=model_length||x_r_neg(1)>=model_length
    save_segment(n,:)=save_plane(find(ismember(save_plane(:,1:3), [1,0,0], 'rows'),1),:);
    n=n+1;
else
    %find side cross right end
    [M,I]=max([zeros(2,1);save_segment(3:end,6)]);
    p1=planeModel(save_segment(I,1:4));
    %find top or bottom plane
    for i=1:1:find(ismember(save_segment(:,1:4), [0,0,0,0], 'rows'),1)-1
        if save_segment(i,1:3)==[0,0,1]
            p2=planeModel(save_segment(i,1:4));
            break
        end
    end
    %find ramp plane
```

```
        p3=planeModel(slice_plane);
        if p1.Parameters(1)~=0
            I=three_plane_intersect_x(p3,p2,p1);
        elseif p1.Parameters(3)~=0
            I=three_plane_intersect_z(p3,p2,p1);
        elseif p1.Parameters(2)~=0
            I=three_plane_intersect_y(p3,p2,p1);
        end
        save_segment(n,:)=[slice_plane',min([x_l_neg(1),x_l_pos(1)]),max([x_r_neg(1),x_r_pos(1)]),I,0];
    end
    save_segment(find(ismember(save_segment(:,1:4), [0,0,0,0], 'rows'),1):end,:)=[];
end
```

C.4 Planes to STL

```
function [center_print] = stl_generate_polygon( ramp_para, name_stl )
%consider surface as polygon and use Delaunay instead of using triangle
slice_normal=[ramp_para(1),ramp_para(2),ramp_para(3)];
length_set=60;
right_1=[];
left_1=[];

% fv = stlread('C:\Users\georgeli\Desktop\project\George-01.stl');
filename=strcat('C:\Users\georgeli\Desktop\project\',name_stl,'.stl');
fv = stlread(filename);
right_end=max(fv.vertices(:,1));
figure(4)
subplot(3,1,1)
title('original model')
xlabel('millimeters')
ylabel('millimeters')
hold on
patch(fv,'FaceColor',       [0.8 0.8 1.0], ...
         'EdgeColor',       'none',        ...
         'FaceLighting',    'gouraud',     ...
         'AmbientStrength', 0.15);

% Add a camera light, and tone down the specular highlighting
camlight('headlight');
material('dull');

% Fix the axes scaling, and set a nice view angle
```

```
axis('image');
view([-135 35]);

fvsave=[];
for i=1:6:size(fv.vertices,1)
    combine_tri=unique(fv.vertices(i:i+5,:),'rows');
    fvsave=[fvsave;0,0,0;combine_tri];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%left side slicing%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
check_end=0;
for loop=1:2
    %saved last outline surface
    if exist('IO1','var')==1
        last_p_all=p_all;
    end
    %for the second loop para and end should be changed
    if loop==2
        if abs((ramp_para(4)-length_set*ramp_para(1))/sqrt(1-ramp_para(3)^2))>right_end
            ramp_para=[ramp_para(1),0,sqrt(1-ramp_para(1)^2),ramp_para(4)];
            check_end=1;
        end
        ramp_para=[ramp_para(1),0,sqrt(1-ramp_para(1)^2),ramp_para(4)-length_set*ramp_para(1)];
    else
        ramp_para=[ramp_para(1),0,sqrt(1-ramp_para(1)^2),ramp_para(4)];
    end
    if check_end==0
    save_size=size(fv.vertices,1);
    bottom_i=[];
    top_i=[];
    if loop==1;
        side_i=[];
    end
    %distinguish top bot and side
    for i=1:3:save_size
        %bottom
        if unique(fv.vertices(i:i+2,3),'rows')==0
            bottom_i=[bottom_i,i];
        %top
        elseif unique(fv.vertices(i:i+2,3),'rows')==max(fv.vertices(:,3))
            top_i=[top_i,i];
        %side
        else
            if loop==1
                side_i=[side_i,i];
```

```
        end
    end
end
%update bottom order
picked=1;
i=2;
all_triangles=fv.vertices(bottom_i(picked):bottom_i(picked)+2,:);
while size(picked,2)~=size(bottom_i,2)
    check1=any(ismember(sum(ismember(all_triangles(:,1:2),fv.vertices(bottom_i(i),1:2)),2),2));
    check2=any(ismember(sum(ismember(all_triangles(:,1:2),fv.vertices(bottom_i(i)+1,1:2)),2),2));
    check3=any(ismember(sum(ismember(all_triangles(:,1:2),fv.vertices(bottom_i(i)+2,1:2)),2),2));
    if (check1+check2+check3>1)&&sum(ismember(picked,i))==0
        all_triangles=[all_triangles;fv.vertices(bottom_i(i):bottom_i(i)+2,:)];
        picked=[picked,i];
    end
    i=i+1;
    if i>size(bottom_i,2)
        i=2;
    end
end
fv.vertices(bottom_i(1):bottom_i(1)+size(all_triangles,1)-1,:)=all_triangles;
%update top order
picked=1;
i=2;
all_triangles=fv.vertices(top_i(picked):top_i(picked)+2,:);
while size(picked,2)~=size(top_i,2)
    check1=any(ismember(sum(ismember(all_triangles(:,1:2),fv.vertices(top_i(i),1:2)),2),2));
    check2=any(ismember(sum(ismember(all_triangles(:,1:2),fv.vertices(top_i(i)+1,1:2)),2),2));
    check3=any(ismember(sum(ismember(all_triangles(:,1:2),fv.vertices(top_i(i)+2,1:2)),2),2));
    if (check1+check2+check3>1)&&sum(ismember(picked,i))==0
        all_triangles=[all_triangles;fv.vertices(top_i(i):top_i(i)+2,:)];
        picked=[picked,i];
    end
    i=i+1;
    if i>size(top_i,2)
        i=2;
    end
end
fv.vertices(top_i(1):top_i(1)+size(all_triangles,1)-1,:)=all_triangles;
%%%%%%%%%%%%
new_fv=[];
p_all=[];
%generate triangles for bottom
[~,~,poly_vertex]=combine_triangle(fv.vertices(min(bottom_i):max(bottom_i)+2,:));
if loop==1
```

```
    [new_poly,IO,connect_order,p1,p2]=polygon_slice(poly_vertex,ramp_para,1);
else
    [new_poly,IO,connect_order,p1,p2]=polygon_slice(poly_vertex,ramp_para,2);
end
if isempty(p1)==0&&isempty(p2)==0
    new_poly(find(ismember(new_poly,p1,'rows'),1),:)=round(new_poly(find(ismember(new_poly,p1,'rows'),1)
 ,:),5);
    new_poly(find(ismember(new_poly,p1,'rows'),2),:)=round(new_poly(find(ismember(new_poly,p1,'rows'),2)
 ,:),5);
end
p_all=[p_all;p1;p2];
for i=1:size(IO,1)
    if IO(i)==1
        new_fv=[new_fv;new_poly(connect_order(i,1),:);new_poly(connect_order(i,2),:);new_poly(
 connect_order(i,3),:)];
    end
end
%generate triangles for top
[~,~,poly_vertex]=combine_triangle(fv.vertices(min(top_i):max(top_i)+2,:));
if loop==1
    [new_poly,IO,connect_order,p1,p2]=polygon_slice(poly_vertex,ramp_para,1);
else
    [new_poly,IO,connect_order,p1,p2]=polygon_slice(poly_vertex,ramp_para,2);
end
if isempty(p1)==0&&isempty(p2)==0
    new_poly(find(ismember(new_poly,p1,'rows'),1),:)=round(new_poly(find(ismember(new_poly,p1,'rows'),1)
 ,:),5);
    new_poly(find(ismember(new_poly,p1,'rows'),2),:)=round(new_poly(find(ismember(new_poly,p1,'rows'),2)
 ,:),5);
end
p_all=[p_all;p1;p2];
for i=1:size(IO,1)
    if IO(i)==1
        new_fv=[new_fv;[new_poly(connect_order(i,1),:);new_poly(connect_order(i,2),:);new_poly(
 connect_order(i,3),:)]];
    end
end
%generate triangles for side
new_side_i=[size(new_fv,1)+1];
previous_size=size(new_fv,1)+1;
if loop==1
    side_save=[];
    for i=1:size(side_i,2)
        if mod(side_i(i),2)==1
            side_save=[side_save,side_i(i)];
```

```
        end
    end
    side_i=side_save;
    side_i=[side_i,side_i(end)+6];
end
for i=1:size(side_i,2)-1
    [~,~,poly_vertex]=combine_triangle(fv.vertices(side_i(i):side_i(i+1)-1,:));
    if loop==1
        [new_poly,IO,connect_order,p1,p2]=polygon_slice(poly_vertex,ramp_para,1);
    else
        [new_poly,IO,connect_order,p1,p2]=polygon_slice(poly_vertex,ramp_para,2);
    end
    if isempty(p1)==0&&isempty(p2)==0
        new_poly(find(ismember(new_poly,p1,'rows'),1),:)=round(new_poly(find(ismember(new_poly,p1,'rows')
,1),:),5);
        new_poly(find(ismember(new_poly,p1,'rows'),2),:)=round(new_poly(find(ismember(new_poly,p1,'rows')
,2),:),5);
    end
    p_all=[p_all;p1;p2];
    new_side_i=[new_side_i,previous_size+nnz(IO)*3];
    previous_size=previous_size+nnz(IO)*3;
    for j=1:size(IO,1)
        if IO(j)==1
            new_fv=[new_fv;new_poly(connect_order(j,1),:);new_poly(connect_order(j,2),:);new_poly(
connect_order(j,3),:)];
        end
    end
end
new_side_i=[new_side_i,previous_size+nnz(IO)*3];
side_i=unique(new_side_i(1:end-1));
%slicing surface
p_all=unique(round(p_all,5),'rows');
if isempty(p_all)==1
     p_all=[p_all;];
else
    if size(unique(p_all(:,1:2),'rows'),1)==size(p_all,1)
        k=convhull(p_all(:,1),p_all(:,2));
    else
        k=convhull(p_all(:,2),p_all(:,3));
    end
    save_insert=[];
    %if slicing surface is concave
    if size(k,1)-1~=size(p_all,1)
        for i=1:size(p_all,1)
            if sum(ismember(k,i))==0
```

```
                concave_p=i;
                insert_row=p_all(concave_p,:);
            end
        end
        concave_num=find(ismember(round(new_fv,4),round(p_all(concave_p,:),4),'rows'));
        close_p=[];
        for i=1:size(concave_num,1)
            for j=1:size(side_i,2)-1
                if concave_num(i)>=side_i(j)&&concave_num(i)<side_i(j+1)
                    if concave_num(i)-1>=side_i(j)&&concave_num(i)-1<side_i(j+1)
                        close_p=[close_p;concave_num(i)-1];
                    else
                        close_p=[close_p;side_i(j+1)-1];
                    end
                    if concave_num(i)+1>=side_i(j)&&concave_num(i)+1<side_i(j+1)
                        close_p=[close_p;concave_num(i)+1];
                    else
                        close_p=[close_p;side_i(j)];
                    end
                end
            end
        end
        p_all=p_all(k(1:end-1),:);
        p_all=[p_all;p_all(1,:)];
        between_p=unique(new_fv(close_p,:),'rows');
        for i=1:size(between_p,1)
            save_insert=[save_insert;find(ismember(round(p_all,4),round(between_p(i,:),4),'rows'))];
        end
        save_insert=[save_insert;save_insert(1)];
        for i=1:size(save_insert,1)-1
            if abs(save_insert(i)-save_insert(i+1))==1
                insert_position=min([save_insert(i),save_insert(i+1)]);
            end
        end
        p_all=[p_all(1:insert_position,:);insert_row;p_all(insert_position+1:end-1,:)];
    else
        %is convex
        p_all=p_all(k(1:end-1),:);
    end
end
%saved last outline surface
if exist('IO1','var')==1
    last_connect_order=connect_order1;
    last_IO=IO1;
end
```

```
if size(unique(p_all(:,1:2),'rows'),1)==size(p_all,1)
    [IO1,connect_order1]=delaunay_slice(p_all(:,[1,2]));
else
    [IO1,connect_order1]=delaunay_slice(p_all(:,[2,3]));
end
%close the open outline
if loop==1
    for i=1:size(IO1,1)
        if IO1(i)==1
            new_fv=[new_fv;[p_all(connect_order1(i,3),:);p_all(connect_order1(i,2),:);p_all(connect_order1
 (i,1),:)]];
        end
    end
else
    for i=1:size(IO1,1)
        if IO1(i)==1
            new_fv=[new_fv;[p_all(connect_order1(i,1),:);p_all(connect_order1(i,2),:);p_all(connect_order1
 (i,3),:)]];
        end
    end
    for i=1:size(last_IO,1)
        if last_IO(i)==1
            new_fv=[new_fv;[last_p_all(last_connect_order(i,3),:);last_p_all(last_connect_order(i,2),:);
 last_p_all(last_connect_order(i,1),:)]];
        end
    end
end
%replace
fv.vertices=new_fv;
nn=1;
%edit faces
fv.faces=[];
for i=1:1:size(fv.vertices,1)/3
    for j=1:1:3
        fv.faces(i,j)=nn;
        nn=nn+1;
    end
end
end
if loop==1
    %save remaining
    stlwrite('C:\Users\georgeli\Desktop\project\remaining.stl',fv) % Save to binary .stl
    fv = stlread('C:\Users\georgeli\Desktop\project\remaining.stl');
    figure(4)
    if loop==1
```

```
        subplot(3,1,2)
    else
        subplot(3,1,3)
    end
    title('remaining part')
    xlabel('millimeters')
    ylabel('millimeters')
    patch(fv,'FaceColor',       [0.8 0.8 1.0], ...
             'EdgeColor',       'none',        ...
             'FaceLighting',    'gouraud',     ...
             'AmbientStrength', 0.15);

    % Add a camera light, and tone down the specular highlighting
    camlight('headlight');
    material('dull');

    % Fix the axes scaling, and set a nice view angle
    axis('image');
    view([-135 35]);
else
    %calculate the center of the piece
    biggest_x=-inf;
    smallest_x=inf;
    biggest_y=-inf;
    smallest_y=inf;
    for i=1:size(fv.vertices,1)
        if fv.vertices(i,1)<smallest_x
            smallest_x=fv.vertices(i,1);
        end
        if fv.vertices(i,1)>biggest_x
            biggest_x=fv.vertices(i,1);
        end
        if fv.vertices(i,2)<smallest_y
            smallest_y=fv.vertices(i,2);
        end
        if fv.vertices(i,2)>biggest_y
            biggest_y=fv.vertices(i,2);
        end
    end
    center_print=[(smallest_x+biggest_x)/2,(smallest_y+biggest_y)/2];
    %save next_piece
    stlwrite('C:\Users\georgeli\Desktop\project\next_piece.stl',fv) % Save to binary .stl
    fv = stlread('C:\Users\georgeli\Desktop\project\next_piece.stl');
    figure(4)
    subplot(3,1,3)
```

```
        title('next segment')
        xlabel('millimeters')
        ylabel('millimeters')
        patch(fv,'FaceColor',       [0.8 0.8 1.0], ...
                 'EdgeColor',       'none',        ...
                 'FaceLighting',    'gouraud',     ...
                 'AmbientStrength', 0.15);

        % Add a camera light, and tone down the specular highlighting
        camlight('headlight');
        material('dull');

        % Fix the axes scaling, and set a nice view angle
        axis('image');
        view([-135 35]);
    end
end
end
```

C.5 Polygon slcing

```
function [new_poly_vertex,IO,connect_order,p1,p2] = polygon_slice( poly_vertex,ramp_para,check )
%POLYGON_SLICE Summary of this function goes here
%   Detailed explanation goes here
save_i=[];
slice_normal=[ramp_para(1),ramp_para(2),ramp_para(3)];
slice_point=[0,0,-ramp_para(4)/ramp_para(3)];
poly_vertex=[poly_vertex;poly_vertex(1,:)];
%slice the polygon, if there is intersection put into save_i
for i=1:size(poly_vertex,1)-1
    v1=ramp_para(1)*poly_vertex(i,1)+ramp_para(2)*poly_vertex(i,2)+ramp_para(3)*poly_vertex(i,3)+ramp_para(4);
    v2=ramp_para(1)*poly_vertex(i+1,1)+ramp_para(2)*poly_vertex(i+1,2)+ramp_para(3)*poly_vertex(i+1,3)+
     ramp_para(4);
    if (v1<0&&v2>0)||(v1>0&&v2<0)
        save_i=[save_i,i];
    end
    if i==1
        if check==1
            if v1>0
                direction_i=1;
            else
                direction_i=0;
```

```
            end
        else
            if v1>0
                direction_i=0;
            else
                direction_i=1;
            end
        end
    end
end
%if there is intersection update the new shape
if isempty(save_i)==0
    p1=plane_line_intersect(slice_normal,slice_point,poly_vertex(save_i(1),:),poly_vertex(save_i(1)+1,:));
    p2=plane_line_intersect(slice_normal,slice_point,poly_vertex(save_i(2),:),poly_vertex(save_i(2)+1,:));
    if direction_i==0
        new_poly_vertex=[p1;poly_vertex(save_i(1)+1:save_i(2),:);p2];
    else
        new_poly_vertex=[poly_vertex(1:save_i(1),:);p1;p2;poly_vertex(save_i(2)+1:end-1,:)];
    end
%distinguish both are valid or invalid
elseif (check==1&&v1<0)||(check==2&&v1>0)
    p1=[];
    p2=[];
    new_poly_vertex=[];
    IO=[];
    connect_order=[];
    return
else
    poly_p=poly_vertex(1:end-1,:);
    %poly_p has to be rounded, sometimes instead of 0 there is a very small
    %number
    p1_n=find(ismember(round(poly_p(:,1),5),0));
    if size(p1_n,1)==2
        p1=poly_p(p1_n(1),:);
        p2=poly_p(p1_n(2),:);
    else
        p1=[];
        p2=[];
    end
    new_poly_vertex=poly_vertex(1:end-1,:);
end
%use delaunay triangulate the polygon
if size(unique(new_poly_vertex(:,3)),1)==1
    [IO,connect_order]=delaunay_slice(new_poly_vertex(:,[1,2]));
    for i=1:size(connect_order,2)-1
```

```
        for j=1:size(connect_order,1)
            if IO(j)==1&&abs(connect_order(j,i)-connect_order(j,i+1))==1&&(connect_order(j,i)>connect_order(j,
     i+1))
                connect_order=fliplr(connect_order);
                return
            end
        end
    end
elseif size(unique(new_poly_vertex(:,1)),1)==1
    [IO,connect_order]=delaunay_slice(new_poly_vertex(:,[2,3]));
    for i=1:size(connect_order,2)-1
        for j=1:size(connect_order,1)
            if IO(j)==1&&abs(connect_order(j,i)-connect_order(j,i+1))==1&&(connect_order(j,i)>connect_order(j,
     i+1))
                connect_order=fliplr(connect_order);
                return
            end
        end
    end
else
    [IO,connect_order]=delaunay_slice(new_poly_vertex(:,[1,3]));
    for i=1:size(connect_order,2)-1
        for j=1:size(connect_order,1)
            if IO(j)==1&&abs(connect_order(j,i)-connect_order(j,i+1))==1&&(connect_order(j,i)>connect_order(j,
     i+1))
                connect_order=fliplr(connect_order);
                return
            end
        end
    end
end
end
```

C.6 SLAM1: total station SLAM

```
function [ mu, sigma ] = SLAM1( printer_mea_2, mu, sigma, Q )
%SLAM1 Summary of this function goes here
%   Detailed explanation goes here
for i=1:3
    %convert landmark measurement in local frame from Cartesian to polar
    [theta,rho] = cart2pol(printer_mea_2(1,i),printer_mea_2(2,i));
    %z_m is measurement in total station part of SLAM.First row is
```

```
    %distance, second row is angle
    z_m(:,i) = [rho,wrapTo2Pi(theta)]';
    %z_hat is predicted measurement from odometry data. First row is
    %distance, second row is angle
    delta = mu(2*i+2:2*i+3)-mu(1:2);
    q = delta'*delta;
    z_hat(:,i) = [sqrt(q);wrapTo2Pi(wrapTo2Pi(atan2(delta(2),delta(1)))-(wrapTo2Pi(mu(3))-pi/2))];
    %F is used to extend matrix dimension
    F = [eye(3),zeros(3,2*i-2),zeros(3,2),zeros(3,2*3-2*i),zeros(3,size(mu,1)-9);
        zeros(2,3),zeros(2,2*i-2),eye(2),zeros(2,2*3-2*i),zeros(2,size(mu,1)-9)];
    %Jacobian matrix
    H = 1/q*[-sqrt(q)*delta(1),-sqrt(q)*delta(2),0,sqrt(q)*delta(1),sqrt(q)*delta(2);
            delta(2),-delta(1),-q,-delta(2),delta(1)]*F;
    %Kalman gain
    K = sigma*H'*inv(H*sigma*H'+Q);
    %accumulated mu and sigma change
    mu=mu+K*(z_m(:,i)-z_hat(:,i));
    sigma=(eye(size(mu,1))-K*H)*sigma;
end

end
```

C.7 SLAM2: 3D scanner SLAM

```
function [ mu, sigma ] = SLAM2_thetaphi2( plane_num, order_para, mu, sigma, bot_pos)
%SLAM2 3D scanner SLAM algorithm
%This algorithm updates robot pose and plane parameters
state_num=plane_num*5+9;
%save plane parameter part of the state vector into matrix
for i=1:(size(mu,1)-9)/5
    mu_matrix(i,:)=[sin(mu(5+5*i))*cos(mu(6+5*i)),sin(mu(5+5*i))*sin(mu(6+5*i)),cos(mu(5+5*i)),-sin(mu(5+5*i))
     *cos(mu(6+5*i))*mu(7+5*i)-sin(mu(5+5*i))*sin(mu(6+5*i))*mu(8+5*i)-cos(mu(5+5*i))*mu(9+5*i),mu(7+5*i:9+5*i
     )'];
end
syms x_0 y_0 z_0 x_r y_r theta_r phi_g theta_g
for i=3:size(order_para,1)
    %plane parameters from 3D scanner (has been transformed into robot frame) is used as measurement
    order_para_g=[cos(mu(3)-pi/2),-sin(mu(3)-pi/2),0;sin(mu(3)-pi/2),cos(mu(3)-pi/2),0;0,0,1]*order_para(i
     ,1:3)';
    [z_m_theta_g,z_m_phi_g]=abc2tp(order_para_g(1)/norm(order_para_g),order_para_g(2)/norm(order_para_g),
     order_para_g(3)/norm(order_para_g));
    [z_m_theta,z_m_phi]=abc2tp(order_para(i,1),order_para(i,2),order_para(i,3));
```

```
%plane parameters from CAD model (still in global frame) need to be transformed into robot frame is used
 as prediction
[mu_theta,mu_phi]=abc2tp(mu_matrix(order_para(i,end),1),mu_matrix(order_para(i,end),2),mu_matrix(
 order_para(i,end),3));
z_hat_1_4=[cos(mu(3)-pi/2) sin(mu(3)-pi/2) 0 0;-sin(mu(3)-pi/2) cos(mu(3)-pi/2) 0 0;0 0 1 0;mu(1) mu(2) 0
 1]*mu_matrix(order_para(i,end),1:4)';
z_hat_local(:,i)=[z_hat_1_4(1:3)',sin(mu(3))*(mu_matrix(order_para(i,end),5)-mu(1))-cos(mu(3))*(mu_matrix(
 order_para(i,end),6)-mu(2)),cos(mu(3))*(mu_matrix(order_para(i,end),5)-mu(1))+sin(mu(3))*(mu_matrix(
 order_para(i,end),6)-mu(2)),mu_matrix(order_para(i,end),7)]';
%F is used to extend matrix
F = [eye(3),zeros(3,6),zeros(3,5*order_para(i,end)-5),zeros(3,5),zeros(3,5*plane_num-5*order_para(i,end));
    zeros(5,3),zeros(5,6),zeros(5,5*order_para(i,end)-5),eye(5),zeros(5,5*plane_num-5*order_para(i,end))];
%point cloud center
x0=order_para(i,7);
y0=order_para(i,8);
z0=order_para(i,9);
b=[x0,y0,z0,1]';
% closest point from centroid to the plane in state vector
[I2,check]=plane_line_intersect([z_hat_local(1,i),z_hat_local(2,i),z_hat_local(3,i)],z_hat_local(4:6,i)',b
 (1:3)',b(1:3)'+[z_hat_local(1,i),z_hat_local(2,i),0]);
% replace b with another point closest to origin
[I1,check]=plane_line_intersect([z_hat_local(1,i),z_hat_local(2,i),z_hat_local(3,i)],b(1:3)',[0,0,b(3)
 ],[0,0,b(3)]+[z_hat_local(1,i),z_hat_local(2,i),0]);
z_m(:,i) = [z_m_theta,z_m_phi,I1(1:3)]';
% intersection of reference point and plane in state vector
[I,check]=plane_line_intersect([z_hat_local(1,i),z_hat_local(2,i),z_hat_local(3,i)],z_hat_local(4:6,i)',I1
 (1:3),I1(1:3)+[z_hat_local(1,i),z_hat_local(2,i),0]);
z_hat(:,i)=[acos(z_hat_local(3,i))/sqrt(z_hat_local(1,i)^2+z_hat_local(2,i)^2+z_hat_local(3,i)^2);
atan2(z_hat_local(2,i),z_hat_local(1,i));
I(1);
I(2);
I(3)];
% plot planes, measured and estimated centroid of planes
figure(6)
hold on
axis equal
y=b(2)-10:.1:b(2)+10;
z=0:.1:7.3;
[Y,Z] = meshgrid(y,z);
a1=z_hat_1_4(1); b1=z_hat_1_4(2); c1=z_hat_1_4(3); d1=-z_hat_1_4(4);
X=(d1- b1 * Y - c1 * Z)/a1;
surf(X,Y,Z)
shading flat
xlabel('x'); ylabel('y'); zlabel('z')
pts1=[I1;I];
```

```
plot3(pts1(:,1), pts1(:,2), pts1(:,3));
pts2=[b(1:3)';I2];
plot3(pts2(:,1), pts2(:,2), pts2(:,3));
scatter3(b(1),b(2),b(3),'filled','r','LineWidth',3)
scatter3(I(1),I(2),I(3),'filled','k','LineWidth',3)
scatter3(I1(1),I1(2),I1(3),'filled','k','LineWidth',3)
scatter3(I2(1),I2(2),I2(3),'filled','r','LineWidth',3)
scatter3(z_hat_local(4,i),z_hat_local(5,i),z_hat_local(6,i),'filled','b','LineWidth',3)

I_g=[cos(mu(3)-pi/2),-sin(mu(3)-pi/2),0,mu(1);
     sin(mu(3)-pi/2),cos(mu(3)-pi/2),0,mu(2);
     0,0,1,0;
     0,0,0,1]*[I,1]';
ja1=[0,0,0;
     0,0,-1;
     -sin(theta_r),cos(theta_r),cos(theta_r)*(x_0-x_r)+sin(theta_r)*(y_0-y_r);
     -cos(theta_r),-sin(theta_r),-sin(theta_r)*(x_0-x_r)+cos(theta_r)*(y_0-y_r);
     0,0,0];
tform_xyr=double(subs(ja1, [theta_g,phi_g,x_0,y_0,z_0,x_r,y_r,theta_r], [mu_theta,mu_phi,I_g(1),I_g(2),I_g
 (3),mu(1),mu(2),mu(3)]));

ja2=[1,0,0,0,0;
     0,1,0,0,0;
     0,0,sin(theta_r),-cos(theta_r),0;
     0,0,cos(theta_r),sin(theta_r),0;
     0,0,0,0,1];
tform_b=double(subs(ja2, [theta_g,phi_g,x_0,y_0,z_0,x_r,y_r,theta_r], [mu_theta,mu_phi,I_g(1),I_g(2),I_g
 (3),mu(1),mu(2),mu(3)]));
% H is combined with tform_xyr and tform_b since this part of SLAM is updating both robot pose and plane
 parameters simultaneously
H = [tform_xyr,tform_b]*F;

error_measurement=diag([0.001,0.001,1,1,0.01]);
r=wrapTo2Pi(z_m(2,i))-wrapTo2Pi(mu(3));
% eigenvector of x_p y_p part of covariance matrix
[V,D]=eig(sigma(1:2,1:2));
syms a b
eqns=[V(1,1)*a+V(2,1)*b==cos(mu(6+5*order_para(i,end))+pi/2),V(1,2)*a+V(2,2)*b==sin(mu(6+5*order_para(i,
 end))+pi/2)];
S=solve(eqns,[a b]);
alpha1=round(double(S.a),3);
alpha2=round(double(S.b),3);
%plot
figure(9)
title('eigenvector (red) and cutting plane (blue)')
```

```
hold on
axis equal
error_ellipse(sigma(1:2,1:2)^2,[0;0],'conf',0.68);
line([0,V(1,1)*D(1,1)],[0,V(2,1)*D(1,1)],'color','r')
line([0,V(1,2)*D(2,2)],[0,V(2,2)*D(2,2)],'color','r')
line([0,S.a*V(1,1)+S.b*V(2,1)],[0,S.a*V(1,2)+S.b*V(2,2)],'color','b')
hold off
%
u=alpha1*[V(1,1);V(2,1)]+alpha2*[V(1,2);V(2,2)];
sigma_cut=u'*sigma(1:2,1:2)*u;
R=[cos(r-pi/4),-sin(r-pi/4);sin(r-pi/4),cos(r-pi/4)];
xy_error=R*diag([sigma_cut,0.1])*R';
error_measurement(3:4,3:4)=xy_error;
% Kalman gain
K = sigma*H'*inv(H*sigma*H'+error_measurement);
% accumulated mu and sigma change
z_diff=z_m(:,i)-z_hat(:,i);
if abs(z_diff(1))>pi
    if z_diff(1)>0
        z_diff(1)=z_diff(1)-2*pi;
    else
        z_diff(1)=z_diff(1)+2*pi;
    end
elseif abs(z_diff(2))>pi
    if z_diff(2)>0
        z_diff(2)=z_diff(2)-2*pi;
    else
        z_diff(2)=z_diff(2)+2*pi;
    end
end
check_add=K*z_diff;
print_adj=check_add(1:3)
side_adj=check_add(25:29)
ramp_adj=check_add(35:39)
if i~=1&&i~=2
    sqrt(z_diff(3)^2+z_diff(4)^2)
    mu=mu+K*z_diff;
    sigma=(eye(state_num)-K*H)*sigma;
end
%plot
figure(5)
hold on
axis equal
%adjust x&y in same scale
tmpAspect=daspect();
```

```
    daspect(tmpAspect([1 2 2]))
    scatter(mu(1),mu(2),'.');
    %plot error ellipse
    error_ellipse(sigma(1:2,1:2)^2,mu(1:2),'conf',0.95);

end
end
```