UNIVERSITY OF COLORADO

SCOPE: SMALL-SAT CONNECTED OPTICAL POSITIONING ENTITY ORIENTATION DETERMINATION THROUGH LIDAR AND ITERATIVE CLOSEST POINT

Mattia Astaria
Nicholas Cenedella
Pepe Feu Vidal
Connor Kerry
Greg Kondor
Nolan Lee
Guy Margalit
Mason Markle
Jake Mitchell
Zach Schira
Alec Viets

Monday 11th December, 2017

SCOPE, or Small-sat Connected Optical Positioning Entity, is a proof of concept sensor package whose purpose is to return the position, velocity, orientation, and rotation rate state information of a target satellite. This sensor package is currently being constructed by a team of undergraduate aerospace engineering students at the University of Colorado, Boulder. The real-time state information of a target satellite is critical for developing autonomous solutions for docking and other close-proximity operations. SCOPE's objective is to provide the necessary state information for these close-proximity operations to be executed without ground station communication (autonomously). The focus of this paper is to explore the methods by which the target satellite's orientation and rotation rate will be determined. This will be accomplished by using an iterative closest point algorithm called FLOOD to process point cloud data from a flash LiDAR sensor. So far in the project, the algorithm has been designed and tested through a Monte Carlo simulation using the simulation software blender. In simulation, 70 percent of orientation at time steps from 1 to 10 meters could be determined within one degree of accuracy. Currently, the FLOOD algorithm is being used to return the static orientation of a target with an actual LiDAR. Preliminary tests showed that FLOOD was not as successful in a test scenario as in simulation. Orientation was only determined within one percent accuracy in 20 percent of the trails. However, since these are only preliminary results, the performance of the LiDAR/FLOOD system in determining orientation has shown promising trends, as 90 percent of points were determined within 10 degrees of actual orientation. Further development will include the altering of the TARGET satellite to be more asymmetrical, and using a different testing environment to reduce the presence of ambient light and foreign objects.

Contents

List of Figures

List of Tables

Acronyms

LiDar	Light Detection and Ranging
FLOOD	Flash Lidar Object Orientation Determination
SCOPE	Small-sat Connected Optical Positioning Entity
ICP	Iterative Closest Point
MSE	Mean Squared Error
POSE	Position and Orientation

Nomenclature

$\hat{v}, \hat{u}, \hat{w}$	SCOPE body coordinate frame
$\hat{x}, \hat{y}, \hat{z}$	TARGET body coordinate frame
$\hat{X}, \hat{Y}, \hat{Z}$	SCOPE base (inertial) coordinate frame
Q	Transformation Matrix
q	Quaternion
ϕ	Roll angle
θ	Pitch angle
ψ	Yaw angle

1. Introduction

SCOPE, Small-sat Connected Optical Positioning Entity, is a proof-of-concept sensor package tasked with returning state information on a target satellite. This state information consists of the relative position, velocity, orientation, and rotation rate of the target. The sensor package is being developed by a University of Colorado senior projects group of 11 undergraduate aerospace engineering students. The purpose of the determination of relative state information is to assist with close proximity autonomous operations. These operations (such as docking procedures) require maneuvers in order to orientate two satellites. In the past, close-proximity operations have been mainly accomplished through human executed maneuvers via ground station communications. The relative orientation and rotation rate between the two satellites are therefore crucial in order to create closed loop feedback. This paper will be focusing on the determination of the rotation rate and orientation via the SCOPE sensor package. Orientation information shall be determined within 1 degree of actual and rotation rate shall be determined within 1 percent of actual. All information will be returned at a rate of 2 Hz, with the target satellite moving at translational velocities of 0.1 m/s at rotational rates of 1 to 5 deg/s about a single axis. Physical constraints of a 1U cube sat must also be met. Orientation and roll rate data will be determined using an iterative closest point algorithm called FLOOD, which process point cloud files (.pcd) created by a flash LiDAR sensor. Below in figure 1, the SCOPE sensor package as well as the TARGET satellite can be seen. This scene depicts the testing environment.

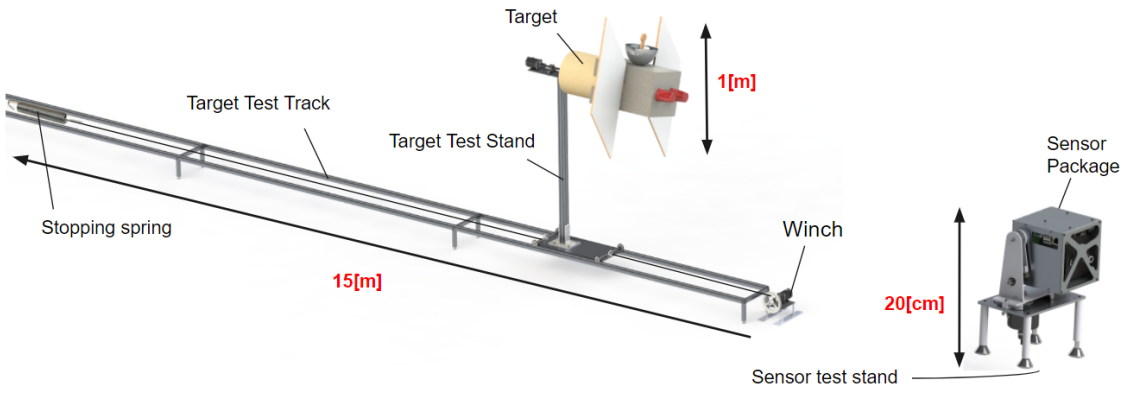

Figure 1. SCOPE test setup

It is important to note the coordinate frames present on the SCOPE sensor package as well as on the target satellite (figure 2). The three frames are as follows: The SCOPE body frame which has its origin centered at the optical camera of the SCOPE sensor package. Next, the target satellite body frame which shows that rotation will occur about the w axis which points towards the senor package. Lastly, the SCOPE base (inertial) frame is fixed with the z axis always pointing at the target satellite. The orientation and rotation rate of the target will be determined in relation to the SCOPE base frame.

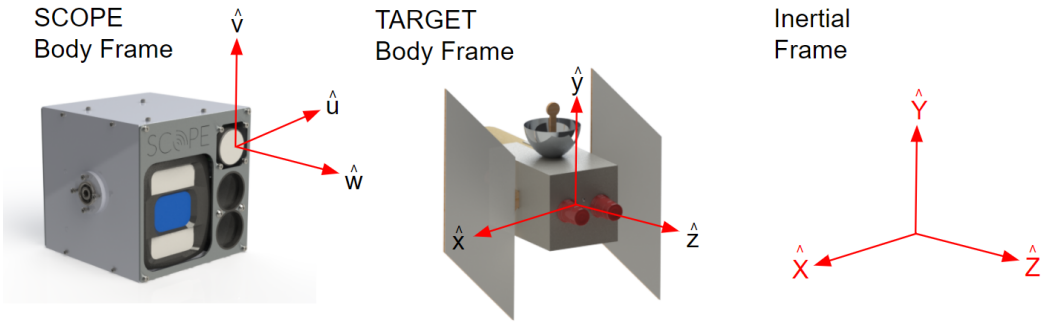

Figure 2. Test setup coordinate frames

In the following sections, the process for the development and verification of the FLOOD algorithm will be discussed. This process includes the design and optimization of the FLOOD algorithm, preliminary testing through Monte Carlo simulations utilizing blender to create [.pcd] files, and testing on a physical target satellite (as seen in Figure 1).

2. FLOOD

The FLOOD algorithm is used to determine orientation and position (POSE) within the final 10m of the proposed mission. This algorithm is based on the commonly used iterative closest point method to compare point clouds to a 3-d model of a the TARGET satellite in order to determine the translation and orientation of the TARGET. The algorithm has two main sections: search and calculation. The search section looks for the closest point in the model to each point from in the point cloud. The calculation section then calculates an ideal rotation and translation to minimize the mean squared distance between the points and the model.

The search is the most computationally expensive part of the algorithm, so it must be performed as efficiently as possible. To do this, the 3-d model is initially parsed, and the faces of the model are placed in a k-d tree data structure. Rather than storing the individual vertices from the model as the nodes of the k-d tree, each face is stored in a bin at one of the leaf nodes. This allows the distance calculated to be the exact minimum distance to the model instead of using the minimum distance to a vertex, which is much less precise. When actually performing the search, the algorithm traverses down the tree to a leaf node. It then calculates the minimum distance from the point to each triangular face in the bin. After finding the minimum distance within that bin, it must traverse back up the tree and check if it is possible for a face in a neighboring bin to be closer than the minimum distance. If it finds there is a bin that may contain a closer point then it must traverse back down the sub tree and check the other bin.

Once all of the minimum distances have been calculated the optimization can be performed. This optimization is set up as a standard least squares minimization. Formally, given the two sets of points (the points produced by the LiDAR and the closest points in the model) there is a point:

$$c_i = arg\min_{c_k \in M} \|(Rd_i + t) - c_k\| \tag{1}$$

Where D is the set of points produced by the LiDAR, M is the set of points from the model, and R and t are a rotation matrix and translation vector respectively. The error function that is then used to actually perform the minimization can be written as:

$$\epsilon = \frac{1}{m} \min_{t,R} \sum_{i=1}^{m} \|(Rd_i + t) - c_i\|^2 \tag{2}$$

This of course is simply the mean squared error of the distance between the points in each set after applying a rotation and translation to the LiDAR point cloud to attempt to align it with the model. This error

function can easily be written in the form a general least squares problem, and Singular Value Decomposition can then be used to solve the least squares problem. This process is done iteratively until the POSE converges to a solution. This is not guaranteed to be a global minimum, so to avoid converging to a local minimum solution the algorithm will test with several different initial orientations, and use the the one that results in the lowest error.

Static testing has been done with the FLOOD algorithm fully integrated with the Flash LiDAR sensor on the target model. This testing (which is discussed in greater detail later) demonstrated the the FLOOD is capable of determining the POSE of the TARGET satellite. The following series of images show the actual orientation of the TARGET, the point cloud collected by the LiDAR sensor, and the predicted orientation.

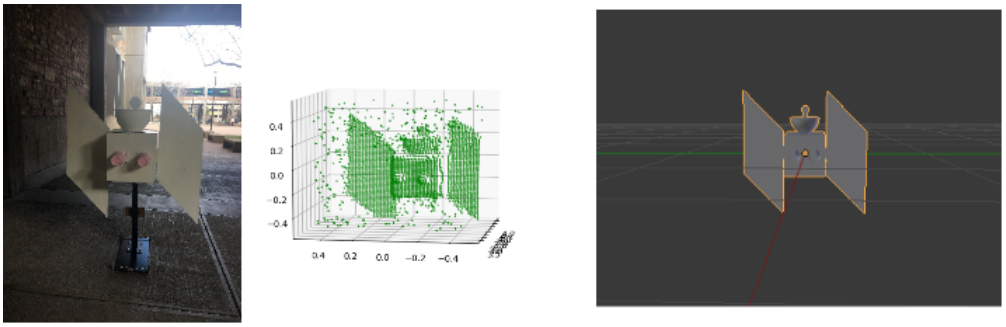

Figure 3. Actual Orientation, LiDAR Point Cloud, and Predicted Orientation

3. Simulation

As a means to conduct a preliminary study of the feasibility of the FLOOD algorithm in determining the relative orientation and rotation rate of a target satellite, the program Blender was used in order to develop point cloud files (.pcd) files without the use of actual LiDAR in a controlled environment. Blender is a 3D graphics modeling software that can simulate motion of objects in three dimensional space. For the purpose of the SCOPE senior project team, Blender was used for its ability to simulate both translational and rotational motion a target satellite. Specifically, Blender's BlenSor package was used to create .pcd outputs from a simulated LiDAR sensor. With BlenSor, a LiDAR sensor is placed in a simulated environment, and data can be retrieved with specified parameters that will be discussed below. By using BlenSor's simulated environment, the algorithms developed were then tested to ensure that an object would be recognized within the requirements provided. Moreover, a Monte Carlo simulation with 100 iterations was created through Blender's python interface, where the parameters were varied according to a uniform distribution across the 100 simulations. Trends were then studied to quantify the initial performance of the FLOOD algorithm. It is important to note that one of the parameters entered into BlenSor was the error of the individual beams of the LiDar at $\pm 2cm$. This was done to ensure that trends in the errors would provide indications on FLOOD's performance and not the LiDAR's performance.

The Blender Interface can be seen below in Figure 4.

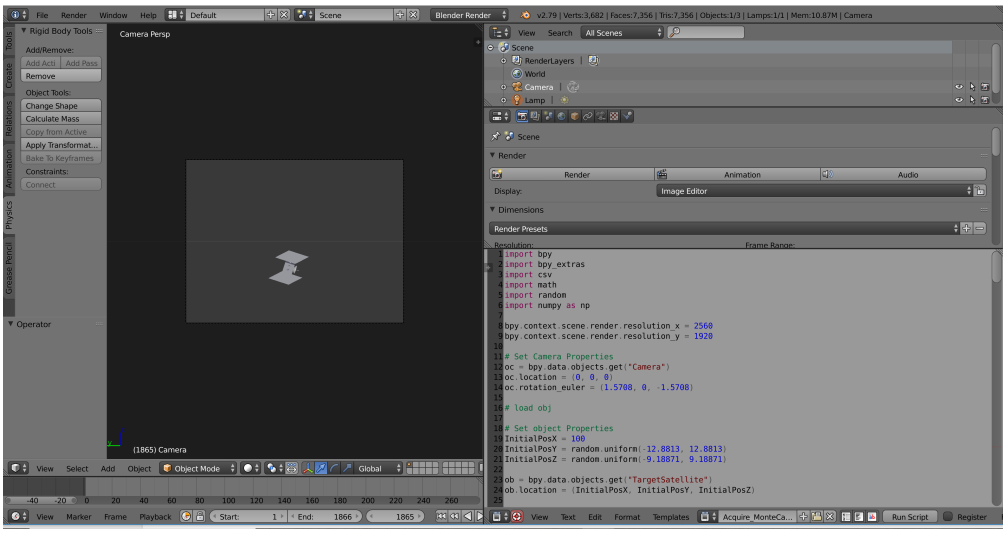

Figure 4. Blender GUI with python interface

The parameters of the LiDAR that were entered into the Blender program to be used in the simulation are listed below.

Table 1. LiDar Specs

Range	Accuracy	FOV	Data Rate	Power
>10m	$\pm 2cm$	14 by 10 degrees	2 Hz	10W

The path and properties of the object were defined as such:

- Velocity = 0.1 m/s
- Distance = 1 to 10 m
- Rotation Rate = 1 to 5 deg/s in all DOF
- Initial position in FOV
- Initial rotation
- Size 0.2 to 1 m in width
- Path, considering only straight trajectories

Data from 1 to 10 m was then analyzed to indicate FLOOD's performance in determining orientation and rotation rate. Below is the error in orientation determination (in degrees) from 1 to 10 m. The requirement is for measurements to have less than 1 degree error, so error is plotted in degrees.

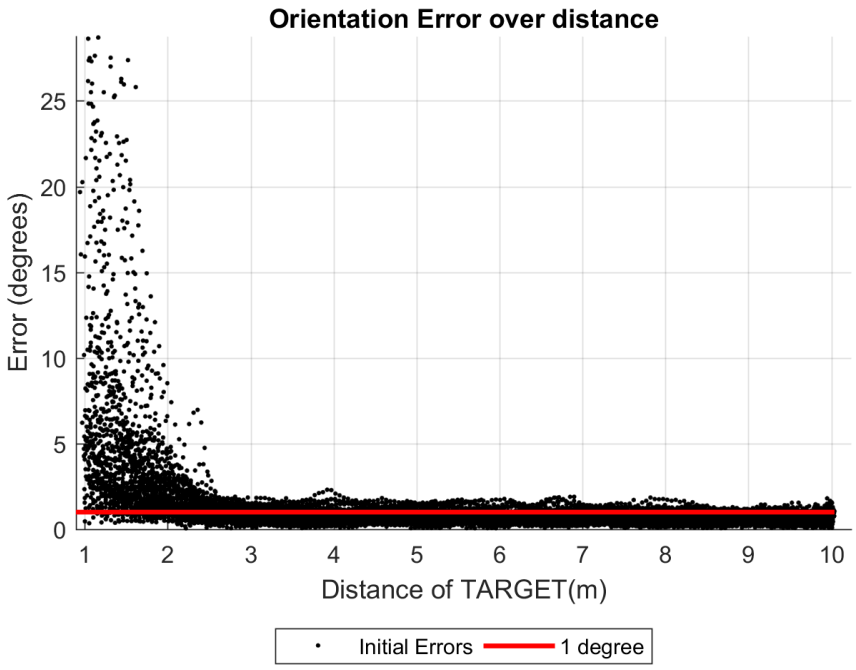

Figure 5. Orientation Simulation Results

In the plot above, each dot represents a instance in time where state estimation was executed. It can be seen that the highest amount of error occurs under 2 meters. This is most likely due to the target being larger than the LiDAR field of view so the edges of the target satellite are lost. There is also plenty of error above 1 degree over 2 m. Overall, 70 percent of points reside below the 1 deg limit. This is not acceptable, and must be addressed moving forward with the project. Rotation error was also analyzed. The percent error was analyzed from 1 to 10 m and the plotted results can be seen below.

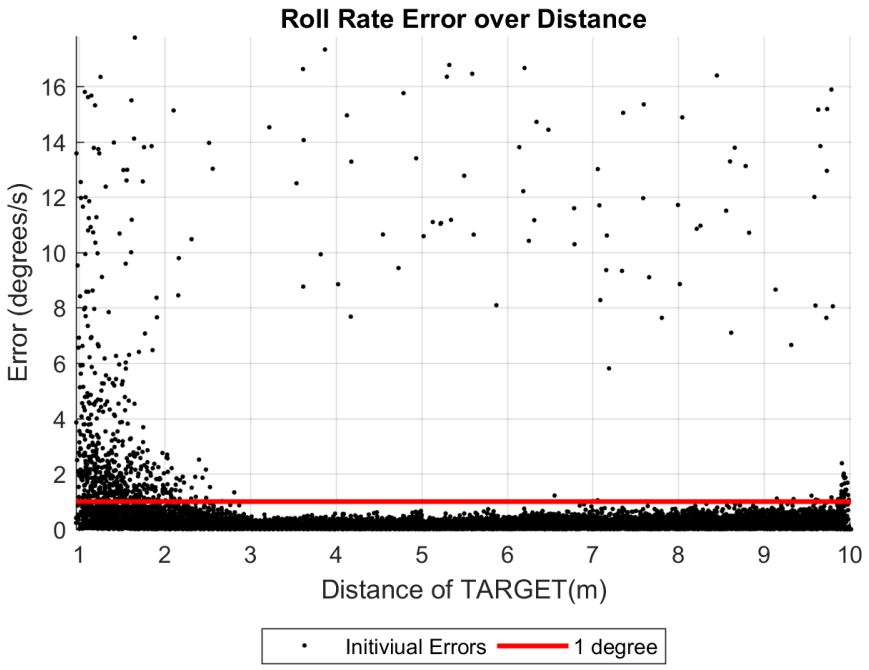

Figure 6. Rotation Rate Simulation Results

The rotation rate was determined more accurately than orientation, where 94.8 percent of the points where below the threshold value of 1 percent. The greatest concentration of points that are over 1 percent occur under 2 meters. Again, this is most likely due to field of view restrictions. Moving forward it is important to consider options that may improve the state estimation for distances under 2 meters.

4. Results

In order to test the accuracy of FLOOD, the test set up depicted in Figure 1 was built. Besides SCOPE and its test stand, the main components of this set up are the target, its test stand, and a 15 meter long track. The track allows for translational motion of the target, which is obtained by attaching fishing wire to the bottom plate, and pulling it with a winch [along the track]. The target stand connects the target to the track and provides rotational motion through its top plate. To do so, the top plate consists of a NEMA24 stepper motor that has an encoder in the back. Using a driver provided by the motor manufacturer and controlled through an Arduino-Uno, the target is spun between 1 and 5 [deg/s]. In order to validate that the system returns orientation and rotation rates within 1 [deg] and 1 [deg/s] accuracy respectively, an encoder is attached to the back of the motor on the top plate. The encoder returns orientation data with one order of magnitude greater than that of SCOPE's LiDAR.

As a preliminary test, the ability of SCOPE to determine orientation from 1 to 10 meters was tested using the materials described above. The data collected will then be compared to simulation results. To make this comparison, it is important to note that there are differences between actual testing and the simulations that will cause results to differ. In simulation, the TARGET rotates in all degrees of freedom where as in actual testing only one degree of freedom is considered. Also, foreign objects such as the ground and structures that are present in actual testing and not in simulation will introduce another source of error.

The target satellite was placed at one meter intervals away from the sensor package. At each distance, the TARGET was rotated about its axis. The yaw angles tested at each distance were 0, 90, 180, and -90 degrees. Using the FLOOD/LiDAR system, orientation was determined ten times at each distance/angle. It should be noted that the center of the face of the LiDAR was the same height as the center of the target and

the track ran perpendicular to the LiDAR's face, so all other Euler angles were close to or at zero.

Below is the average error in the yaw angle determination by FLOOD. The following plot (figure 7), shows this error as the TARGET is placed farther and farther away from SCOPE. Average error at different orientations further quantifies FLOOD's performance. Error is typically greatest when the TARGET is very close to SCOPE at 1 meter. As the TARGET is placed further away at 2 meters, error decreases and then begins to steadily increase as the TARGET moves further and further away. This behavior of high error at close distance was seen in simulation, and was expected.

An interesting aspect of this plot is the local maxima occurring at 7 meters. Error increases for every orientation until 7 meters where it decreases. Another important aspect to note is that only 20% of measurements are with in the desired error of 1 degree.

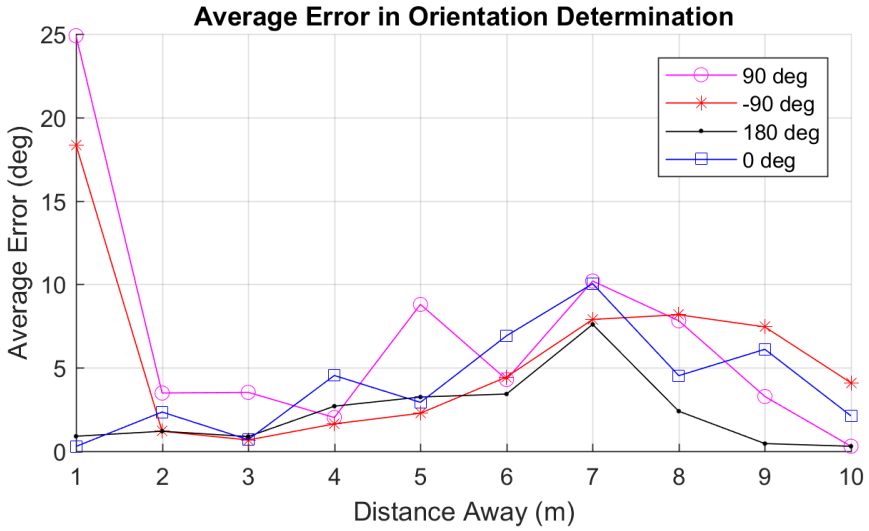

Figure 7. Error in yaw determination at given orientations and various distances

Comparing these results to simulation results, the most obvious take away is the disparity of error under 1 degree. FLOOD was able to more accurately determine orientation in the simulated scenarios when compared to the actual test results (70 percent in simulation compared to 20 percent in the actual test). Determining the source for this disparity is important so the performance of the FLOOD algorithm can be improved and requirements can be better fulfilled.

5. Project Conclusions

The preliminary results of the orientation abilities of SCOPE show that there is definite room for improvement. Although error in orientation somewhat resembled what was expected from simulations, only approximately 20 percent of points fell below 1 percent error. This is a drastic decrease from the 70 percent of points seen in simulation. The reason for FLOOD having worse performance in an actual testing scenario is the presence of ambient light, foreign structures (such as walls or the ground), and differences between the manufactured TARGET from the 3D model of the TARGET. These differences should be used as lessons to improve the performance of the FLOOD algorithm in the future. Specifically, changing the testing environment and shape of the TARGET for orientation tests. This could reduce noise and readings of foreign objects from the LiDAR and thus increase the ability of FLOOD to accurately determine the TARGET's orientation. Testing at night in an open space such as a field has proven to reduce noise due to ambient light. Also the asymmetry of the TARGET could be changed so that different orientations wouldn't given similar error when attempting to align the same point cloud.

One hindrance that limited testing, and therefore limited our ability to acquire comprehensive data was the damaging of the encoder on the back of the test stand. Without the encoder, rotation rate could not

be tested, which is a significant portion of the goals to be accomplished in the orientation phase of the project. The FLOOD algorithm uses a history of orientations in order to iterate from and find the next possible orientation that reduces the error between the 3D model and point cloud within a specified threshold. Without the ability to determine the instantaneous rotation and rotation rate, a digital level had to be used to determine the TARGET's orientation. In simulation, the TARGET was always dynamic, so that once FLOOD has made its initial guess, it only had to update the orientation as the TARGET moved. In the results presented in this study, the TARGET was always static which tested FLOOD's ability to develop an initial guess and not its ability to update the TARGET's orientation instantaneously. If the encoder was not broken and the tests were able to be run again, results from actual tests and simulations could be compared without caveats.

6. Bibliography

[1] M. Attia Y. Slama M. A. Kamoun "On performance evaluation of registration algorithms for 3d point clouds" Computer Graphics Imaging and Visualization (CGiV) 2016 13th International Conference pp. 45-50 2016.

[2] "Monte Carlo Simulation." Monte Carlo Simulation: What Is It and How Does It Work? - Palisade, "O3D301 - 3D Camera - Eclass: 27310205 / 27-31-02-05." Ifm.com. N. p., 2017. Web. 16 Oct. 2017. www.palisade.com/risk/monte_carlo_simulation.asp

[3] Rusinkiewicz, Szymon; Marc Levoy (2001). Efficient Variants of the ICP Algorithm. Proceedings Third International Conference on 3-D Digital Imaging and Modeling. Quebec City, Quebec, Canada. pp. 145–152.

[4] "Tutorials." Blender.org https://www.blender.org/support/tutorials/

7. Appendix

7.1. Code

7.1.1. Blender Python interface

```
import bpy
import blensor
import math
import random
import numpy as np
from pyquaternion import Quaternion

bpy.context.scene.render.resolution_x = 2560
bpy.context.scene.render.resolution_y = 1920

scene = bpy.data.scenes["Scene"]

# Set Camera Properties
oc = bpy.data.objects.get("Camera")
oc.location = (0, 0, 10)
oc.rotation_euler = (0, 0, 0)
oc.scan_type = 'tof'
oc.tof_max_dist = 20
oc.tof_xres = 176
oc.tof_yres = 132

# load obj

# Set object Properties
InitialPosZ = 0
InitialPosY = random.uniform(-1.2881, 1.2881)
InitialPosX = random.uniform(-0.9189, 0.9189)

ob = bpy.data.objects.get("Testing")
ob.location = (InitialPosX, InitialPosY, InitialPosZ)
ob.rotation_mode = 'XYZ'

Initial_ang_Pos_X = random.uniform(0, 2 * math.pi)
Initial_ang_Pos_Y = random.uniform(0, 2 * math.pi)
Initial_ang_Pos_Z = random.uniform(0, 2 * math.pi)
ob.rotation_euler = (Initial_ang_Pos_X, Initial_ang_Pos_Y, Initial_ang_Pos_Z)

psi_rate = random.uniform(1 * math.pi / 180, 5 * math.pi / 180)
theta_rate = random.uniform(1 * math.pi / 180, 5 * math.pi / 180)
phi_rate = random.uniform(1 * math.pi / 180, 5 * math.pi / 180)

#scale = random.uniform(1, 2)
#ob.scale = (scale, scale, scale)

Velocity = 0.1

R = 9

endpoint = [0, 0, 9]

fps = 2

Length = int(fps*R/Velocity)
```

```
X = np.linspace(InitialPosX, endpoint[0], num=Length)
Y = np.linspace(InitialPosY, endpoint[1], num=Length)
Z = np.linspace(InitialPosZ, endpoint[2], num=Length)

bpy.context.scene.frame_end = len(X)

noise = 0
bpy.context.object.save_scan = True
direct = "D:\Documents(HDD)\SCOPE\Orientation_pcd\Orient"
posFile = open("position.txt", "w+")
rotFile = open("rotation.txt", "w+")

for n in range(1, Length):
    # Update position/rotation
    scene.frame_set(n)
    x = X[n]
    y = Y[n]
    z = Z[n]
    ob.location = (x, y, z)
    ob.keyframe_insert('location', frame=n)
    rotZ = Initial_ang_Pos_Z + psi_rate / fps * n
    rotY = Initial_ang_Pos_Y + theta_rate / fps * n
    rotX = Initial_ang_Pos_X + phi_rate / fps * n

    deg2rad = math.pi/180

    R2 = np.array([[math.cos(rotY*deg2rad), 0, -math.sin(rotY*deg2rad)], [0, 1, 0], [math.sin(rotY*de
    R3 = np.array([[math.cos(rotZ*deg2rad), -math.sin(rotZ*deg2rad), 0], [math.sin(rotZ*deg2rad), mat
    R1 = np.array([[1, 0, 0],[0, math.cos(rotX*deg2rad), -math.sin(rotX*deg2rad)], [0, math.sin(rotX*

    Q=np.matmul(np.matmul(R3,R2),R1)

    qw = math.sqrt(1.0 + Q[0][0] + Q[1][1] + Q[2][2]) / 2.0
    w4 = (4.0*qw)
    qx = (Q[2][1] - Q[1][2])/w4
    qy = (Q[0][2] - Q[2][0])/w4
    qz = (Q[1][0] - Q[0][1])/w4


    ob.rotation_euler = (rotX, rotY, rotZ)
    ob.keyframe_insert('rotation_euler', frame=n)
    posFile.write(str(x) + "  " + str(y) + "  " + str(z) + "\n")
   rotFile.write(str(qw) + "  " + str(qx) + "  " + str(qy) + "  " + str(qz) + "\n")

    pos = (x, y, z)

    dist = np.linalg.norm(pos)

    # Take scan
    if dist <= 3:
        noise = 0.007
    elif dist > 3 and dist <= 5:
        noise = .01
    elif dist > 5 and dist <= 7:
        noise = .015
    else:
        noise = .02
    fname = direct + str(n) + ".pcd"
    oc.tof_noise_sigma = noise
    blensor.dispatch_scan(oc, fname)
```

7.2. Group Member Contributions

Group Member	Contributions
Mattia Astaria	Testing
Nicholas Cenedella	Abstract, Introduction, Simulations, Testing, Testing section, Conclusions
Pepe Fue Vidal	Testing
Greg Kondor	Manufacturing
Nolan Lee	Testing
Guy Margalit	Simulation section proof reading
Mason Markle	Testing
Jake Mitchell	Testing
Zach Schira	FLOOD section, Testing
Alec Viets	Testing