ERDC/GRL TN-20-2
May 2020

An Assessment of an Inexpensive COTS 360-degree Camera for Mapping and Localization in a GPS-denied Area

By Garry P. Glaspell

PURPOSE: The efficacy of using a low Size, Weight, Power, and Cost (SWAP-C) Commercial off the Shelf (COTS) 360-degree camera was evaluated for localization and positioning in a GPS-denied environment. Specifically, OpenVSLAM was utilized to generate point clouds with negligible drift using a RICOH THETA S 360-degree field of view camera without the aid of an Inertial Measurement Unit (IMU). OpenVSLAM is also demonstrated to show that it can localize and position with a pre-generated map.

BACKGROUND: There are a variety of Open Source SLAM modules that are available on GitHub, including, but not limited to, GMapping,[1] ORB-SLAM2,[2] LSD-SLAM,[3] ElasticFusion,[4] and OpenVSLAM.[5] While most of these support monocular, stereo, and Red, Blue, Green, and Depth (RGBD) cameras to some extent, OpenVSLAM is set apart by its ability to utilize perspective, fisheye, and equirectangular cameras. The equirectangular camera that was used for this technical note (TN) is the RICOH THETA S.[6] The RICOH THETA S is a 360-degree camera that records video in full HD (1920×1080) resolution at 30 fps. It weighs ~125g and has the following dimensions: 44mm (W) x 130mm (H) x 22.9mm (D). Even though the RICOH THETA S utilizes two fisheye cameras simultaneously, the field of view from the cameras does not overlap, in contrast to a stereo camera. Consequently, this setup is considered a monocular SLAM approach, and as a result, the generated point clouds are not to scale when compared to real-world values. However, the underlying assumption is the 360-degree field of view provided by the THETA S is capable of maintaining localization where a camera with a smaller field of view would fail. This is especially true when mapping building interiors where blank walls do not provide landmarks for visual SLAM techniques. In addition, to minimize the drift that typically occurs over time, OpenVSLAM provides loop closure. Similar to ORB-SLAM2, OpenVSLAM creates a sparse three-dimensional (3-D) reconstruction of the mapped area. OpenVSLAM has two modes of operation, "mapping" and "localization." The mapping mode will generate a point cloud of an unexplored area, while localization allows the camera to maintain its position on a pre-loaded map.

[1] https://github.com/OpenSLAM-org/openslam_gmapping
[2] https://github.com/raulmur/ORB_SLAM2
[3] https://github.com/tum-vision/lsd_slam
[4] https://github.com/mp3guy/ElasticFusion
[5] https://github.com/xdspacelab/openvslam
[6] https://theta360.com/en/

For this TN, the focus was primarily on the mapping mode. OpenVSLAM also provides Robot Operating System (ROS) drivers for incorporation on a robot platform.

OBJECTIVES: This TN specifically addresses the Army Warfighting Challenges 2015 and the Army Robotic and Autonomous System (RAS) near-term objective of "Enhancing stand-off from threats and improve situation awareness" and their mid-term objective to "Enhance Platoon, Squad, and Soldier situation awareness." At a theatre level (CNA 17-21), the Army Critical Capability Gap(s) specifically list the following items:

1. Gap #500459: "The Army units at Theater level lacks the capacity to integrate host-nation or third nation security forces into friendly sustainment area and base security operations during decisive action 100% of the time"

2. Gap #500915: "The Army at theater level lacks the capability under decisive operations to detect personnel, material and facilities that are subsurface down to 100 meters"

3. Gap #500917: "The Army at theater level lacks the capability under decisive operations to maintain situational awareness of subsurface structure down to 100 meters"

4. Gap #500918: "The Army at theater level lacks the capability under decisive operations to Map a subsurface structure as specified in the mission with Standard Shareable Geospatial

APPROACH: Due to the Army's need to obtain and maintain situational understanding in Dense Urban Areas (DUA), listed in Section 3 of the FY15 Force 2025 Maneuvers Annual Report, the efficacy of using a COTS 360-degree camera was investigated to fill this niche. The main payoff is the ability to map and localize underground structures and building interiors with a low SWAP-C camera integrated on an Unmanned Ground Vehicle (UGV) platform. In a typical scenario, one or multiple UGVs, equipped with a 360-degree camera, would explore an unknown area. The primary UGV would operate in "mapping" mode. Localization is possible by wirelessly transferring the generated map to other mission-specific robots, also equipped with 360-degree cameras. This approach dramatically simplifies the complexity and cost of the payload required to coordinate the actions of multiple robots during a mission.

SOFTWARE INSTALLATION: The first piece of software needed is the RICOH THETA computer application (Windows or MAC) for converting the recorded video from "fisheye" to "equirectangular." Unfortunately, at this time, Linux version 3.12.0 of the software cannot be downloaded.[1] While the RICOH THETA S has a "live stream" option for viewing the video feed, the live steam video of the THETA S is "fisheye" and not the required "equirectangular" perspective. Thus, for this TN, the video was recorded on the camera, transferred it to the computer with the RICOH THETA computer application installed and used the software to convert the video to the equirectangular format. It is significant to note that the THETA S is also equipped with Wi-Fi and it may be possible to use the Wi-Fi transfer mode to get the video feed directly into Linux, but this has not been confirmed possible, although online comments support this. Since the live feed is in the "fisheye" format, the video transferred over Wi-Fi is expected to be "fisheye" as well.

[1] https://support.theta360.com/en/download/pcmac/

Figure 1 shows the raw video feed from the THETA S and after stitching with the RICOH THETA computer application.

Figure 1. Screenshot of raw video collected with the RICOH THETA S (left), after stitching with the RICOH THETA computer application (center), and when viewed in a video player (right).

The second piece of software that was needed is OpenVSLAM.[1] The documentation for installing OpenVSLAM is quite extensive, so this TN focuses only on key issues.[2] The OpenVSLAM documentation walks you through installing Eigen, OpenCV, DBoW2 and g2o from source. For OpenCV, version 3.4.9 was installed and used the cmake option `-DBUILD_WITH_MARCH_NATIVE=OFF`. Installing OpenCV version 4.X was avoided since ROS only currently supports 3.X at the time of writing. While not the focus of this TN, integrating OpenVSLAM into ROS is something that would be interesting to pursue and could be the topic of a future TN. There are presently two ways OpenVSLAM can be installed. One approach utilizes the "PangolinViewer" and the other option is "SocketViewer." If you are installing OpenVSLAM to a bare metal Linux computer, it is recommended using the PangolinViewer option. Note the PangolinViewer viewer option requires compiling Pangolin from source as well. However, to keep from having to swap files between multiple computers the "SocketViewer" option was installed in the Windows Subsystem for Linux (WSL). This allows connection from the RICOH THETA S camera to thelaptop, copy, convert the video in windows, and then process the converted video with OpenVSLAM using WSL. The SocketViewer option requires compiling socket.io-client-cpp from source as well as installing Protobuf (which can be installed from apt if using Ubuntu 18.04 or later). When compiling OpenVSLAM with cmake `-DBUILD_WITH_MARCH_NATIVE=OFF` was also set similar to how OpenCV was installed. Next, `npm install` was used to setup the environment for the server. Finally, as per the setup instructions, the vocabulary file for DBoW2 was downloaded.[3] OpenVSLAM also provides sample equirectangular and fisheye datasets for evaluation.[4] Figure 2 shows the processed equirectangular datasets using SocketViewer.

[1] https://github.com/xdspacelab/openvslam

[2] https://support.theta360.com/en/download/pcmac/

[3] https://drive.google.com/open?id=1wUPb328th8bUqhOk-i8xllt5mgRW4n84

[4] https://openvslam.readthedocs.io/en/master/simple_tutorial.html#sample-datasets

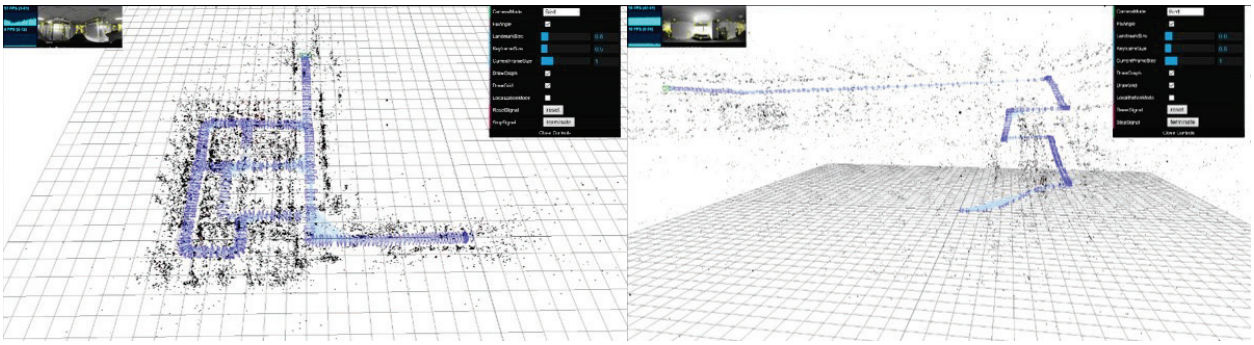

Figure 2. Equirectangular datasets of a single floor (left) and multiple floors (right) after processing the video with OpenVSLAM. The black points are landmarks detected by OpenVSlam and the blue line shows the camera's path through the building.

If using a camera other than the RICOH THETA S with OpenVSLAM, calibration is required. Specifically, OpenVSLAM requires a YAML Ain't Markup Language (YAML) file of the camera configuration. One way to create the YAML file is with OpenCV,[1] which has already been installed in the setup instructions. Provided below is an example of the YAML file for the RICOH THETA S.

```
# Equirectangular model for RICOH THETA S
Camera.name: "RICOH THETA S 960"
Camera.setup: "monocular"
Camera.model: "equirectangular"
Camera.fps: 30.0
Camera.cols: 1920
Camera.rows: 960
Camera.color_order: "RGB"
Feature.max_num_keypoints: 2000
Feature.scale_factor: 1.2
Feature.num_levels: 8
Feature.ini_fast_threshold: 20
Feature.min_fast_threshold: 7
Feature.mask_rectangles:
  - [0.0, 1.0, 0.0, 0.1]
  - [0.0, 1.0, 0.84, 1.0]
```

OpenVSLAM. OpenVSLAM can process data from multiple inputs including a USB video class (UVC) camera, a sequence of static images, as well as video files. For the duration of this TN, the focus will be placed primarily on working with video files. It is significant to note the examples of using standard benchmarking datasets are provided; specifically the KITTI Odometry and the EuRoC MAV datasets. However, since neither of these datasets utilizes a 360-degree camera, they are not the focus of this TN. Shown below is an example of creating a map database from a video file.

[1] https://docs.opencv.org/3.4.9/dc/dbb/tutorial_py_calibration.html

```
./run_video_slam \
    -v /path/to/orb_vocab/orb_vocab.dbow2 \
    -c /path/to/config.yaml \
    -m /path/to/video.mp4 \
    -p /path/to/map.msg
```

The "-v" flag is the path to the DBow2 vocabulary file, the "-c" flag is the camera calibration YAML file, the "-m" flag is the path to the video file, and the "-p" flag is where the map database file will be created. It is believed that the resulting map.msg file is a postgres database compressed by MessagePack. Using the command above, videos were processed that had been collected inside a building. Multiple videos were recorded with more complex movements than the one previously recorded. To stress-test the SLAM module, the last set of videos had multiple 360-degree turns and repeated path traversal. As evidenced in Figure 3, neither of these actions - separate or combined - resulted in any observable drift.

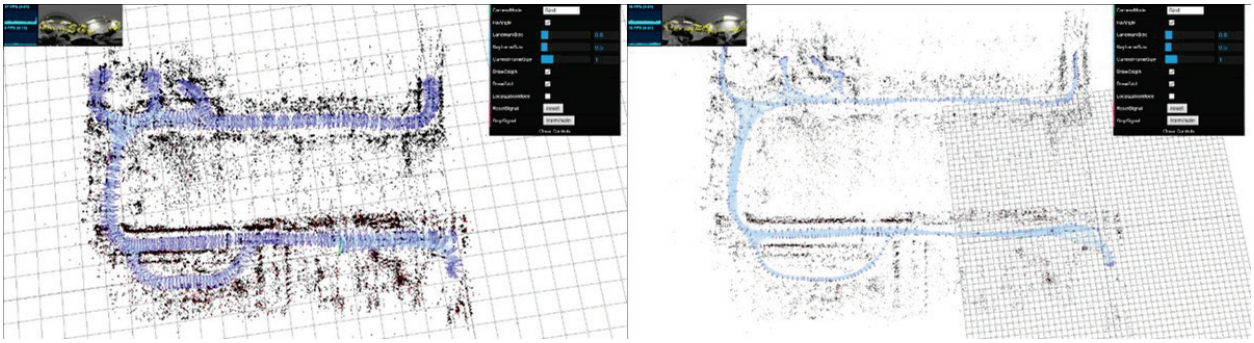

Figure 3. Processed video with multiple 360-degree turns (left) and repeated path traversal (right). The black points are landmarks detected by OpenVSlam and the blue lines are indicative of the camera's path.

While the sample size is small, it appears that the 360-degree field of view from the two fisheye cameras on the RICOH THETA S minimizes the scale drift that is typically observed with a monocular SLAM approach. To extract the resulting point cloud from the map.msg database, MikiyaShibuya posted a python script on GitHub which is listed below.[1]

```
"""
Copyright (c) 2019,
National Institute of Advanced Industrial Science and Technology (AIST),
All rights reserved.

This work is licensed under the terms of the 2-Clause BSD license.
For a copy, see <https://github.com/xdspacelab/openvslam/blob/master/LICENSE>
.
"""

import sys
import msgpack
import json
```

[1] https://github.com/xdspacelab/openvslam/issues/87#issuecomment-515683888

```
def main(bin_fn, dest_fn):

    # Read file as binary and unpack data using MessagePack library
    with open(bin_fn, "rb") as f:
        data = msgpack.unpackb(f.read(), use_list=False, raw=False)

    # The point data is tagged "landmarks"
    landmarks = data["landmarks"]
    print("Point cloud has {} points.".format(len(landmarks)))

    # Write point coordinates into file, one point for one line
    with open(dest_fn, "w") as f:
        for id, point in landmarks.items():
            pos = point["pos_w"]
            f.write("{}, {}, {}\n".format(pos[0], pos[1], pos[2]))
    print("Finished")


if __name__ == "__main__":
    argv = sys.argv

    if len(argv) < 3:
        print("Unpack all landmarks in the map file and dump into a csv file"
)
        print("Each line represents position of landmark like \"x, y, z\"")
        print("Usage: ")
        print("    python pointcloud_unpacker.py [map file] [csv destination]
")

    else:
        bin_fn = argv[1]
        dest_fn = argv[2]
        main(bin_fn, dest_fn)
```

The output of the above script is a Comma Separated Values (CSV) file that can then be loaded into a point cloud processing program such as CloudCompare.[1] The resulting point cloud is shown in Figure 4. While the resulting point cloud is 3-D in nature, it is too sparse to identify features. However, the top-down perspective is feature-rich enough to function as a 2-D map. The sparseness is actually by design and a result of culling detected landmarks in the SLAM module. One of the main advantages of this approach is it is very effective in a dynamic environment, which is the intended use case. In addition, the file size is significantly smaller, allowing for larger maps, when compared to other SLAM modules, before memory and or file size constraints limit the map growth. This approach also improves accuracy by reducing the re-projection of identified landmarks. The re-projection of identified landmarks ultimately results in generating double walls

[1] https://cloudcompare.org/

and drift when rotating the camera. This approach is more independent of the amount of ambient light and the field of view of the camera when compared to other SLAM modules, which will significantly aid in localization.

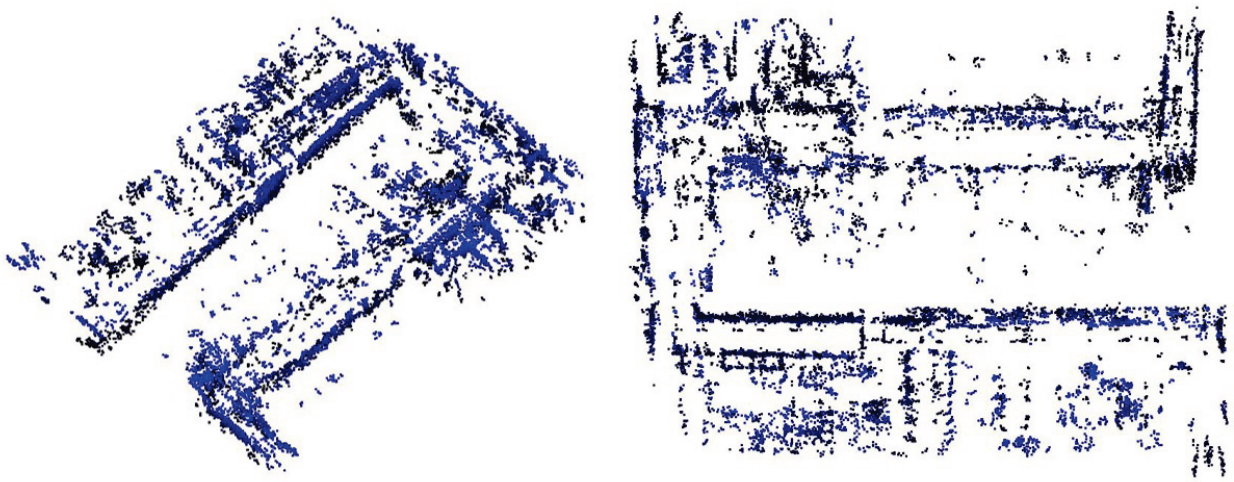

Figure 4. The resulting sparse 3-D point cloud (left) was loaded into CloudCompare. A top-down perspective (right) is also shown. The blue points are points near the ceiling and the black points are closer to the floor.

Once a map.msg database is created, it is possible to localize your position against the created map. To test this, a new video was recorded with the RICOH THETA S and ran the following command.

```
./run_video_localization \
    -v /path/to/orb_vocab/orb_vocab.dbow2 \
    -c /path/to/config.yaml \
    -m /path/to/new_video.mp4 \
    -p /path/to/map.msg
```

Similar to "mapping" mode, the -v, -c, and -m flags mean the same thing. However, in "localization" mode, -p flag refers to the path where the map.msg database file is located. Using the aforementioned command, the video localization was tested using a video and a map database that was recorded. Note that the video used for localization was different from the video used to create the map database. Figure 5 is a screenshot showing the current position of the camera (the green triangle) in relation to the pre-generated map.

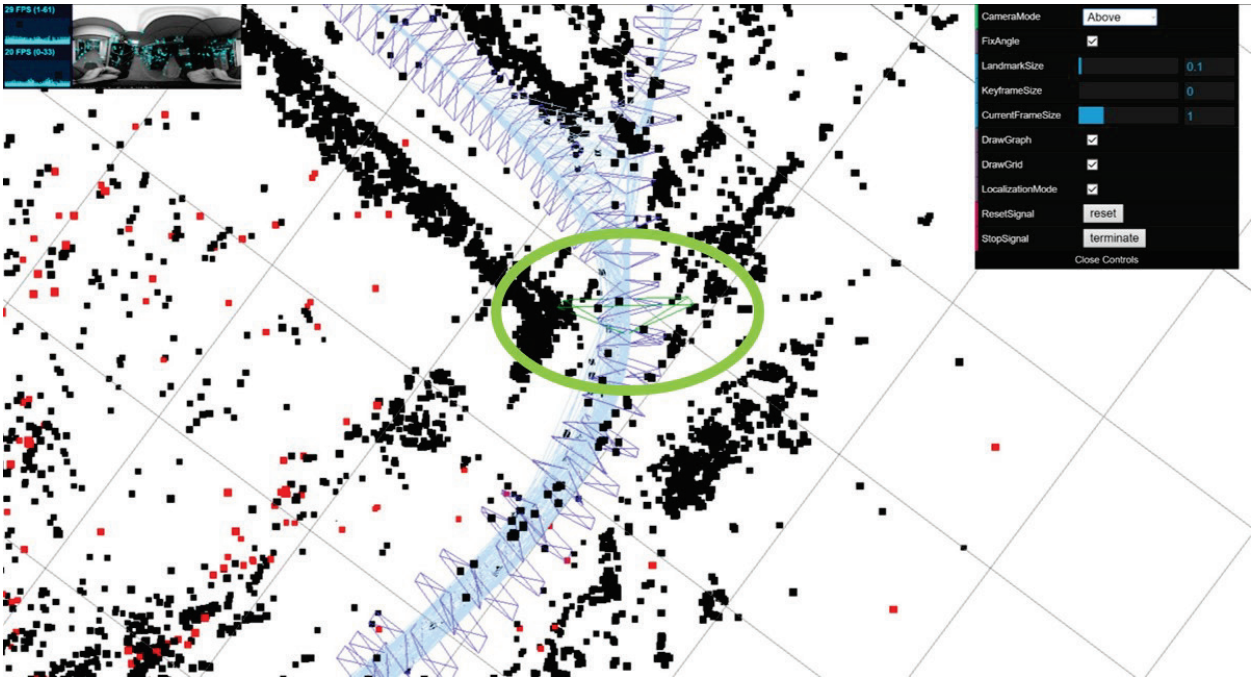

Figure 5. Screenshot of video localization in progress. The green circle shows the current camera position in relation to the pre-generated map.

To localize a camera in real-time, instead of using a pre-recorded video, the command below would be used.

```
./run_camera_localization \
    -v /path/to/orb_vocab/orb_vocab.dbow2 \
    -c /path/to/config.yaml \
    -p /path/to/map.msg
    --mapping
```

The main difference between this command and the previous localization command is the lack of the -m flag, since the camera feed is being used rather than a pre-recorded video, and the addition of the "--mapping" flag. The advantage of using "--mapping" is the pre-mapped areas can be localized as well as continue to map in unexplored areas.

CONCLUSIONS AND RECOMMENDATIONS: In short, the efficacy of using a 360-degree camera for mapping and localization has been demonstrated. While the resulting 3-D map is too sparse for feature extraction, it does provide enough density for an accurate 2-D map. It is also significant to note that the map does not indicate any noticeable drift, either scale or re-projection, even after executing multiple 360 turns or repeated path traversal. However, the real advantage of this approach is in localization. It was demonstrated that with a single low SWAP-C COTS camera, effectively localizing and positioning with a pre-generated map can be done. Also, the "--mapping" flag allows for continuation to map when the boundaries of the pre-generated map are exceeded. Ultimately, this would allow coordinated movements with other mission specific vehicles or robots in a GPS-denied environment cheaply and effectively.

It would be preferable to have both the RICOH THETA camera and OpenVSLAM working under one operating system when moving this concept into fruition. Since the RICOH THETA computer application can be installed on Windows 10, it would seem straightforward to compile OpenVSLAM in Windows. However, when attempting to do this, a link error with g2o was received. The author of OpenVSLAM indicated that a Windows version is currently in development, but not presently released. An alternative approach would be getting the RICOH THETA S working in Ubuntu. Presently, it is unclear if RICOH plans to offer a Linux version of their computer application; an often requested feature. Getting the RICOH THETA S to work in Linux is feasible since the S model uses UVC 1.1, currently supported in Linux. In contrast, the newer RICOH THETA V and z1 uses UVC 1.5, not currently supported in Linux. However, there is some antidotal evidence that it is possible to get the RICOH THETA V and z1 working in Ubuntu 18.04 by leveraging the Wi-Fi transfer mode on the camera. It is also significant to note that the V and z1 models have a resolution of 4k. It is unclear at this time if the higher resolution offered by the V and z1 will improve accuracy compared to the S model, but might be worth investigating. The simplest path forward would be compiling OpenVSLAM on MacOSX, which coincidentally also supports the RICOH THETA computer application as well as ROS. ROS support would allow OpenVSLAM to be incorporated with the ROS packages Move_Base and Frontier_Exploration providing some degree of autonomy on a robotics platform. It would also be interesting to run two RICOH THETA cameras simultaneously. This would minimize scale drift and result in the point clouds having real world values. An alternative approach is the incorporation of an IMU. The author has created a branch on GitHub of OpenVSLAM with IMU support, but not merged into the main branch at this time. Finally, the RICOH camera series is not the only fisheye camera that has ROS drivers; the Kodak PIXPRO 4kVR360 camera is another option.