Examining the Performance of Scale Optimized Visual Odometry in Simulated Underwater Environments

Jack Perisich

Advisor: Junaed Sattar
Dept. of Computer Science and Engineering
University of Minnesota

May 2020

Scale Optimized Visual Odometry is the process of using a stereo camera to update the scale of points found using a traditional monocular visual odometry algorithm in order to achieve greater accuracy whilst not encurring the negative performance impact associated with stereo pattern matching. This paper examines the performance of this method on a robot in a simulated underwater enviroment. While the scale optimized method gives accurate poses in some environments, it inherits many of the shortcomings of the monocular odometry system that it is based on. Some of these shortcomings, such as trouble maintaining accuracy in texture-depleted environments and inability to make sharp turns without a sufficient number of surrounding objects in all directions, are likely to be encountered when operating in underwater domains. Solutions will need to be developed to handle these cases before scale optimized visual odometry can be used robustly underwater.

Introduction

For many autonomous robotic systems, being able to localize the robot with respect to objects in the surroundings is a vital part of being able to operate successfully. Many of these systems perform this task using some form of stereo matching. While stereo matching works well for many of these systems, it does not work so well for others. One of the major downsides of stereo matching is its computational complexity. In many autonomous systems, it is desirable to have the system be able to operate robustly for long periods of time. Due to the high computational cost of stereo matching, it also has an associated high power cost, greatly reducing the time the system can operate without being recharged [*1*]. Another common issue that arises with stereo matching systems is that they tend to fail when viewing a repetitive texture. With many points in the environment looking similar to one another, it is hard to choose which point is the best stereo match [*2*]. In environments where repetitive textures are common, stereo matching will likely have poor results.

The main alternatives to stereo matching odometry algorithms are monocular odometry algorithms. These algorithms rely on a single camera and use the motion of points in the camera frame to determine the location of the robot with respect to these points [*3*]. While this process is much faster than stereo matching and incurs much less of a computational cost, it cannot fully estimate the position of points by itself without also knowing the motion of the camera. Thus, the depth of points and the position of the robot have to be estimated to some unknown scale.

Scale optimized visual odometry uses a stereo camera system to determine points, but does so without stereo matching. Instead, it uses an existing monocular odometry algorithm operating on one of the cameras while the other camera is used to determine the unknown scale, thereby allowing the system to estimate both the depth of the points and the pose of the robot [*1*]. By avoiding stereo pattern matching, the computational cost of the algorithm is much smaller than that of traditional stereo systems. It also maintains the benefit of being more robust on repetitive textures.

For this paper, a version of scale optimized visual odometry called SO-DSO [*1*] was tested on a simulated robot in underwater environments. SO-DSO uses direct sparse odometry (DSO) [*3*] to perform the underlying monocular odometry. The goal of these tests was to analyze the effectiveness and accuracy of the method overall, as well as determining the primary sources of error and to discuss the ways in which this error could be minimized. Such an odometry algorithm is appealing for underwater robots, since they may have to be deployed for long peroids of time without recharging, and also because an underwater robot is likely to encounter repetitive textures with a high frequency. Most underwater environments have significant regions where there are no unique features to make stereo matching easy, so being able to determine pose using a repetitive texture, such as the ocean floor, is desirable. The underwater domain is also sensor-sparse, meaning that other technologies such as GPS are not feasible. This further increases the necessity for a robust visual odometry system.

Background

As discussed previously, many visual odometry systems rely on stereo pattern matching in order to estimate robot pose. Stereo matching is a process that finds distinctive features in one camera image and then tries to find the location of those same features in another image captured by a second camera a known distance away from the first. Once the locations in both camera frames are established, the distance between the cameras can be used to determine the location of the point in the world coordinate system as shown in Figure 1. While this process can be fairly accurate, it requires the testing of a single point in one camera image with many points in the other (more specifically, all points along the epipolar line) in order to find the one most similar. This requires a significant amount of computational power dedicated to finding

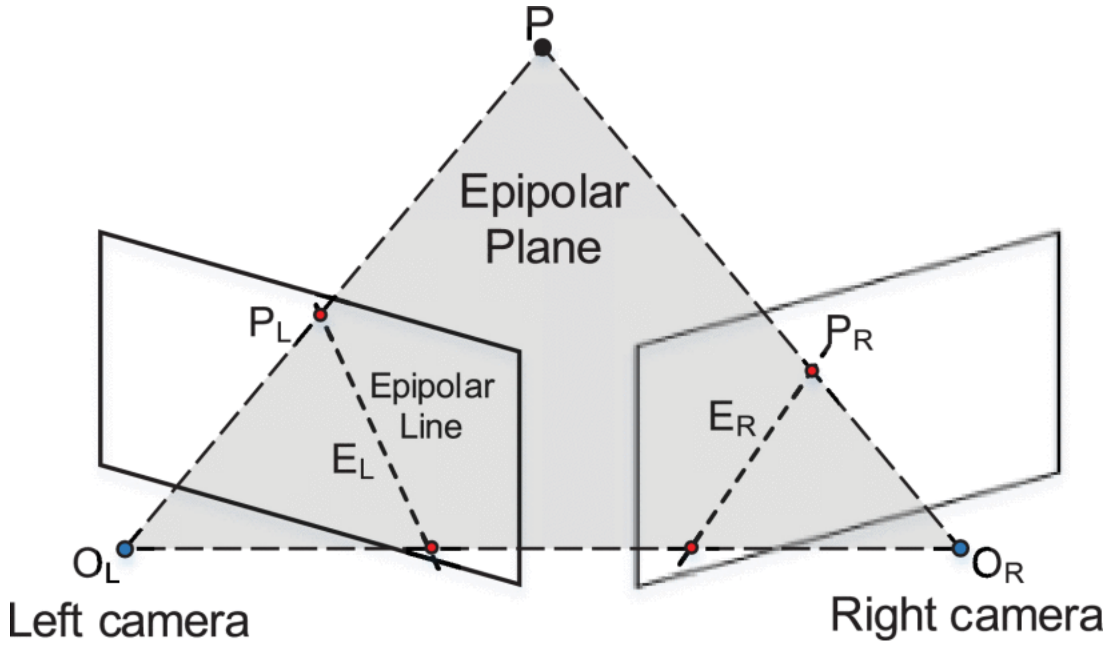


Figure 1: Illustration of how two points in two stereo cameras can be used to determine the location of the point in the world coordinate space. Image sourced from [*5*].

matching pairs in the stereo images. It also suffers when trying to match points in environments with repetitive textures. Since many different points in repetitive textures look similar, when trying to find a point in one image that matches a point in the other, there will be many possible candidates with high similarity and no obvious way to determine which one is more likely to be the actual matching point [*2*]. Most stereo algorithms will also struggle with texture-depleted environments which are common in the underwater domain. In these situations, there are no good points for the stereo algorithm to choose which often leads to large amounts of error since the points chosen have few identifying features [*4*].

In contrast to this, monocular visual odometry systems operate without the need for a second camera. Since depth information is lost when using a single image, monocular odometry systems require that the camera be in motion in order to estimate the locations of points in the environment as well as the robot pose. By looking a how points move between image frames over a period of time, an estimation can be made for location of these points in the world co-

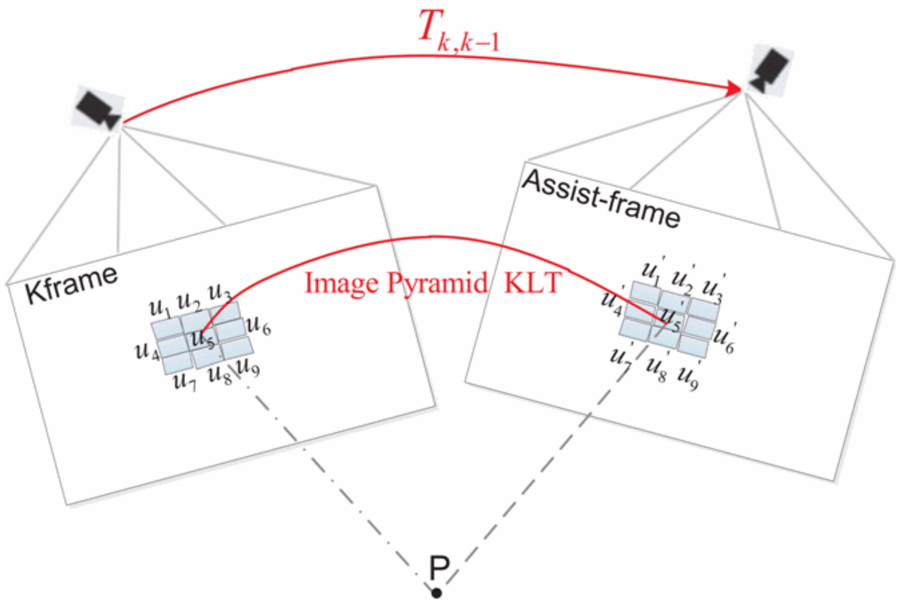

$T_{k,k-1}$
Kframe
Assist-frame
Image Pyramid KLT
u_1 u_2 u_3 u_4 u_5 u_6 u_7 u_8 u_9
u_1' u_2' u_3' u_4' u_5' u_6' u_7' u_8' u_9'
P

Figure 2: Illustrates the process of monocular odometry. Points in the first frame (called a 'keyframe') are compared to the same point in a future frame (called an 'assist frame'). The movement of the point in the camera frame can be used to estimate the point's location and the change in pose of the camera up to a certain scale. Most monocular odometry algorithms will use numerous assist frames. Illustration sourced from [*8*].

ordinate system [*3*]. This process is illustrated in Figure 2. However, all these estimations are only accurate up to an unknown scale. The monocular vision system cannot differentiate between a small change in the camera position with nearby points, and a large change in camera position with points that are further away. Therefore, a monocular vision system needs to be augmented by providing it with information about the camera movement in order to determine this unknown scale. This has primarily been tried by using an onboard inertial measurement unit (IMU), such as in [*6*]. However, this makes the odometry system sensitive to noise in the IMU which can accumulate over time to cause significant errors in the pose estimates (referred to as "IMU drift"). Other systems have been designed that use stereo matching at certain frames to correct this scale. While accurate, this again incurs the undesirable cost of stereo matching [*7*].

Scale optimized visual odometry aims to determine this unknown scale using a stereo cam-

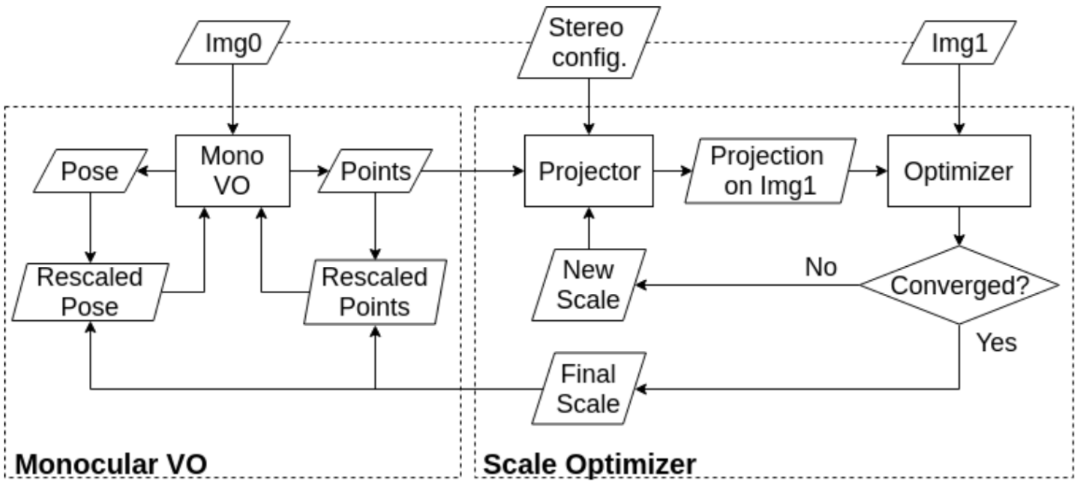


Figure 3: A chart showing the flow of execution for the scale optimized method. 3D points and the stereo image are sent to the scale optimizer which iterates until it converges on a value for the scale. This final scale is returned to the monocular odometry algorithm. Image sourced from [*1*].

era but without relying on stereo matching. The scale optimization method takes as inputs the 3D points found by the monocular odometry algorithm, the stereo image, and the initial scale. The scale optimization method then aims to adjust the scale so that the points and pose calculated are accurate. It does this by using Gauss-Newton optimization [*9*] to minimize an error function. This error function is derived by projecting the 3D points onto the stereo image and summing the differences between the 3D points and the pixels they are projected onto. This method is repeated until the value for the scale converges, at which point the 3D points and pose have been determined and the optimized scale is sent back to the underlying monocular vision algorithm [*1*]. This process is shown in Figure 3. In practice, usually around ten iterations are needed in order for the scale to converge. Using this method, the pose of the robot can be found accurately with a much lower computational overhead than if stereo matching had been used. This method has been tested extensively on the KITTI Visual Odometry dataset [*10*] and the EuRoC MAV dataset [*11*] and shows promising results on both.

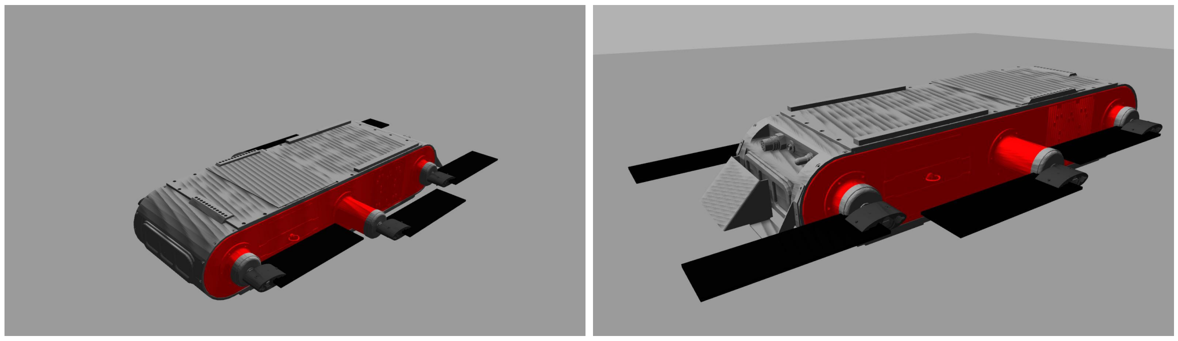
Results

To test the effectiveness of the scale optimized method, the current implementation of SO-DSO was applied to a simulated robot moving through three virtual scenes using Gazebo. The robot used in the simulation is the Aqua underwater robot [*12*]. The model of Aqua used in the simulation is pictured in Figure 4. The pose estimates generated by the algorithm were then compared to the ground truth poses to determine both the effectiveness of the algorithm, as well as to investigate the causes of significant error.

Experiments

In the first environment, the robot was moved through a maze of traffic cones as shown in Figure 5. While such a scenario will never actually exist underwater, this case serves as a good baseline to indicate the conditions necessary for SO-DSO to perform with a high degree of accuracy. The robot was directed to move from the entrance at the bottom of the maze to the exit at the top of the maze. As can be seen by comparing the estimated poses with the ground truth poses, the algorithm generally performed well in this scenario. While there was significant variation at places where the robot changed directions, this is expected behavior as the scale optimized odometry algorithm currently determines pose at the location of the primary camera (in this case, the left camera) and ground truth poses were taken from the center of the robot. Because of this, curvature on left turns will be diminished and curvature on right turns will be exaggerated. While these smaller parts may differ, the overall trajectory matches with the estimation showing the robot moving in the correct directions.

The second environment proved more challenging for SO-DSO to handle. The second environment featured a collapsed house which the robot was asked to approach and then move along the wall until reaching the edge. After reaching the edge, the robot should attempt to turn around the corner of the building. While a collapsed house is not likely to be found underwa-



(a) Front view. (b) Back view.

Figure 4: The model of Aqua used for the simulated experiments.

ter, the model was chosen as a stand-in for a more applicable object (such as a shipwreck or underwater structure) since no textured models of these objects were available. As can be seen in Figure 6, SO-DSO was unable to capture the full amount of the turn when approaching the house. Because of this, the trajectory never “flattened out” as can be seen when looking at the ground truth. SO-DSO also failed when approaching the corner of the house. It was unable to successfully make the turn around the corner of the building, at which point it lost track of its position and could not produce any more poses. The potential reasons for these failures will be discussed in the next section.

The final environment tested was an apartment building (this model was also chosen as a stand-in for a more applicable model). Similar to the collapsed house, the robot was instructed to follow the closest wall to the edge. The algorithm performed noticably better at following the apartment building than the collapsed house, as can be seen in Figure 7. The reasons for this are discussed in the following section.

Evaluation

It can be seen from the three trials conducted that the ablility of SO-DSO to accurately determine pose is heavily dependent on the environment. However, many of these limitations originate

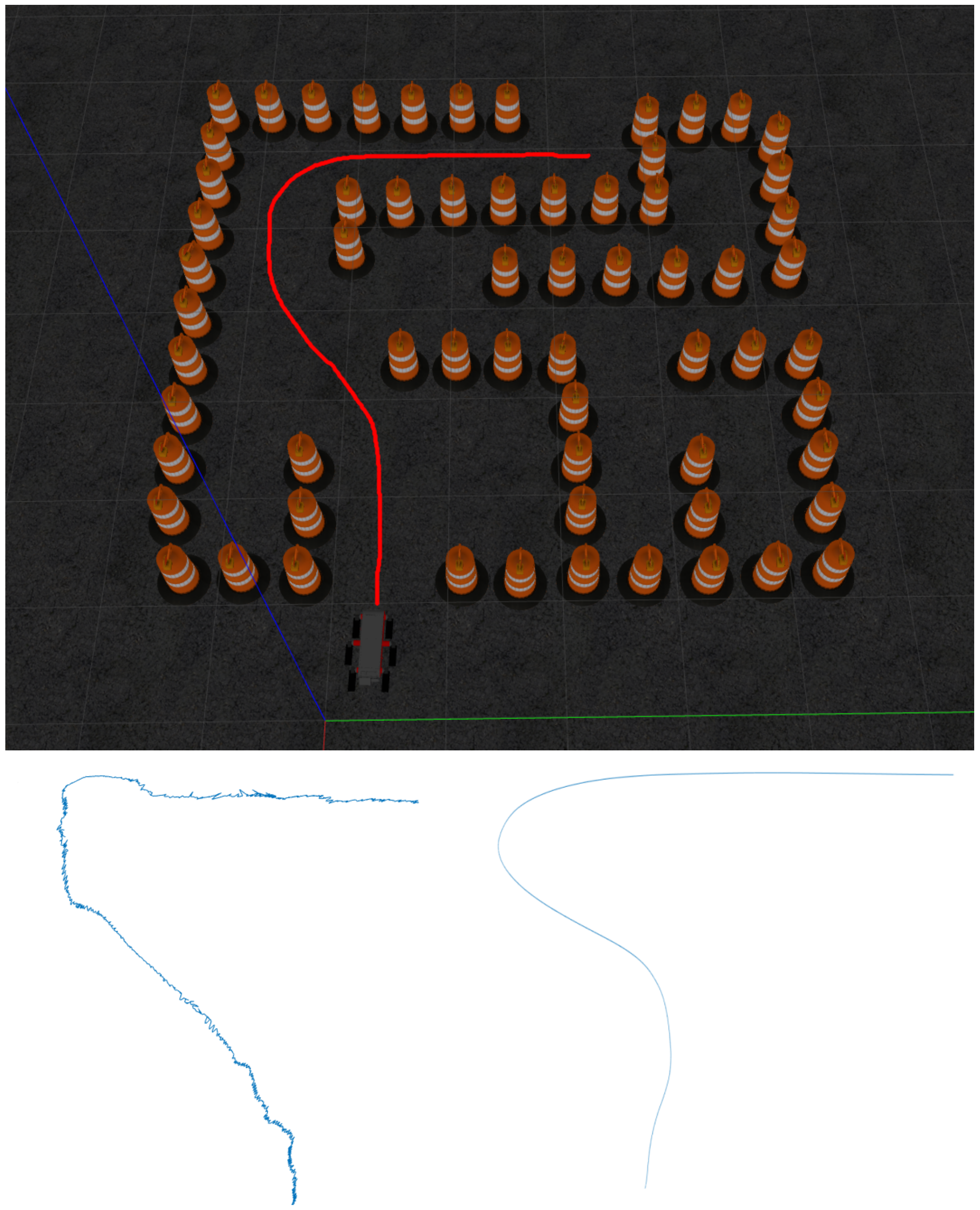


Figure 5: Estimated poses (bottom left) and ground truth poses (bottom right) obtained when directing the robot through a maze of traffic cones (top). The red line indicates the trajectory of the robot with respect to the environment. Plots are all in the horizontal plane.

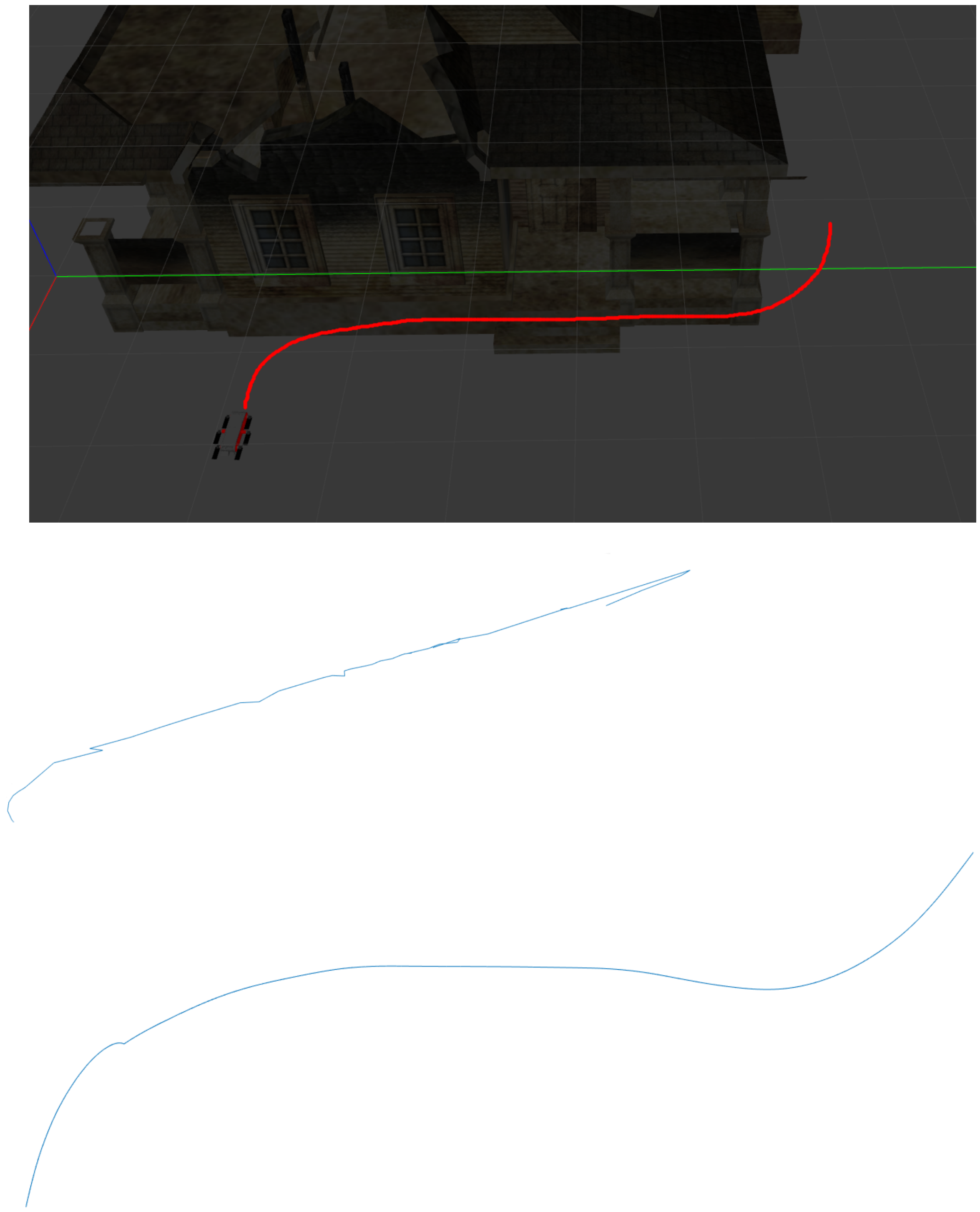


Figure 6: Estimated poses (middle) and ground truth poses (bottom) obtained when directing the robot to follow along the wall of a collapsed house (top). The red line indicates the trajectory of the robot with respect to the environment. Plots are all in the horizontal plane.

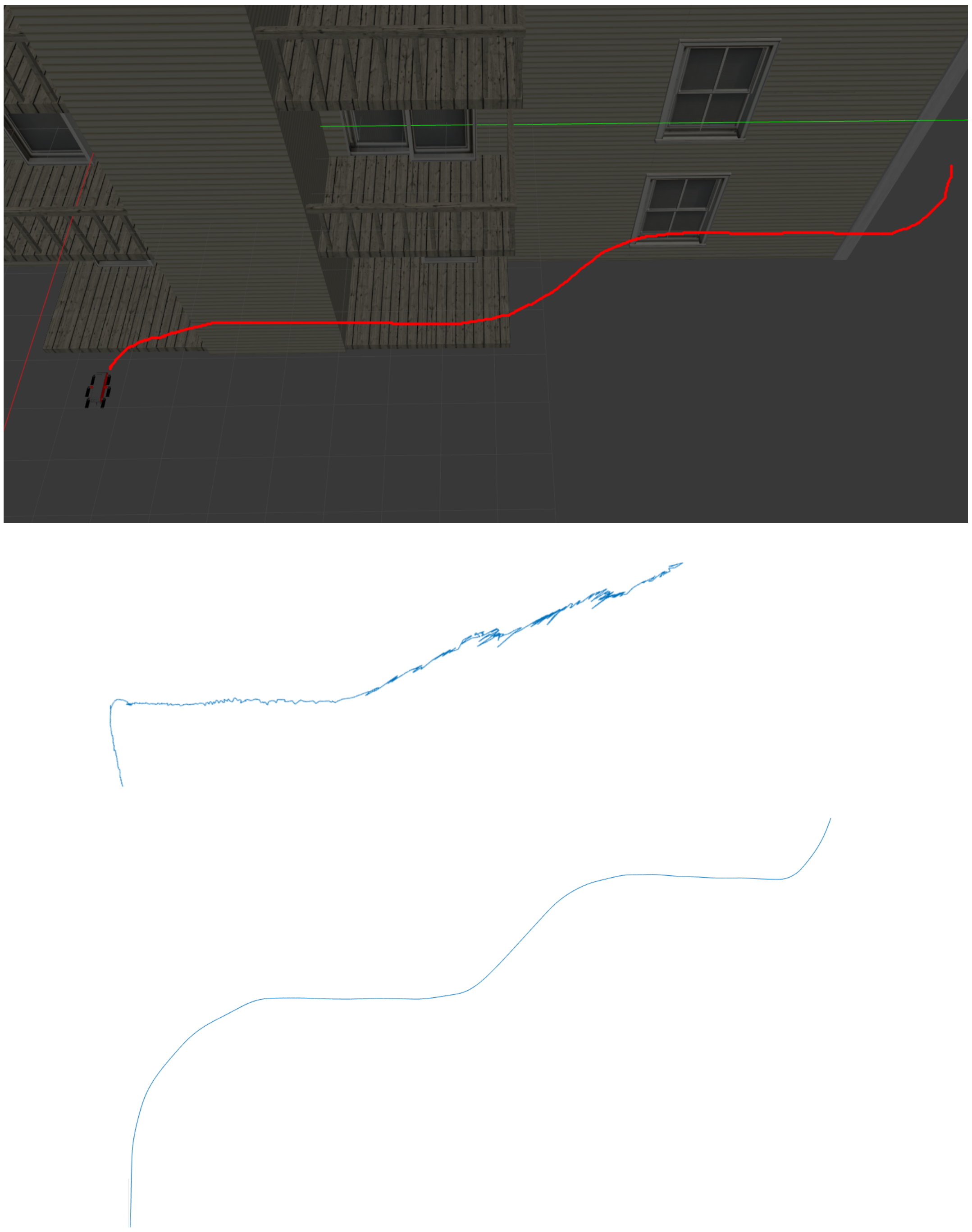


Figure 7: Estimated poses (middle) and ground truth poses (bottom) obtained when directing the robot to follow along the wall of an apartment building (top). The red line indicates the trajectory of the robot with respect to the environment. Plots are all in the horizontal plane.

with the underlying monocular vision algorithm. While SO-DSO aims to correct the pose of monocular vision, it does not manage to correct the situations in which monocular vision fails in the first place.

Circling Objects

The most noticable problem in the above trials is the difficulty of circling objects. Most monocular vision systems will fail when asked to do a turn with no translation, including DSO, on which SO-DSO is based. Without a translation in the robot's movement, there is no paralax in the objects from which the monocular odometry algorithm can deduce pose, so it simply fails. Therefore SO-DSO, which does not correct this problem, also fails in these situations, as in Figure 6 and Figure 7.

From the first trial depicted in Figure 5, we can see that tight turns are not impossible for the system to handle; very little translation is actually needed in certain environments. However, when circling objects, the points of interest that monocular vision is trying to track become more difficult to find due to the camera being mostly pointed away from the object. While this could potentially be counteracted by moving the robot farther from the wall, thereby allowing it to keep more of the object in the camera frame, this is not a feasible solution since all trials showed that the estimated poses had much greater error when all the objects were far away. Circling objects then become difficult if there are not other suitable objects surrounding it for SO-DSO to use to maintain its tracking. This is potentially a major problem in the underwater domain specifically, as there are very few stationary objects, so this assumption will only hold in rare circumstances.

Texture-Depleted Environments

The other main issue is that the algorithm works significantly worse in texture depleted environments. The traffic cones in the first trial proved especially easy to track due to their coloring;

the lines between contrasting colors give the underlying monocular odometry algorithm plenty of points that are good to track. In the case of the collapsed house, the entire house is a very dark color. This leaves fewer suitable candidate points for the underlying monocular odometry algorithm to choose from. Therefore, since the performance of SO-DSO is heavily dependent on the performance of this algorithm, these limitations are passed on to SO-DSO as well. This is another problem which poses a significant challenge for working in underwater domains. The contrast between different colors is diminished in many underwater environments, so it would potentially be necessary to preprocess images with a color-correction algorithm (such as [*13*] or [*14*]) before using them to compute pose.

Accuracy of Points

Another noticable issue is in the assignment of depth to the points. Going back to the maze trial, it can be seen that the positions of the points from the robot's perspective are much like what is expected; the different objects in the environment can be made out by looking at the point cloud (Figure 8). This suggests that the directions of these points from the robot are highly accurate. However, when viewed from the top, the objects can barely be made out by looking at the point cloud (Figure 9). Taking these two views into account, it can be concluded that there is significant error in the depths of the points being computed. While this is seemingly not having a large effect on the actual pose calculation, it poses another problem if future work tries to use these points to accomplish some other task (such as checking for loop closure or trying to recompute pose after all trackable objects have moved out of frame for a short time). A similar observation can be made when looking at the points from the collapsed house trial (Figure 10) and the apartment building trial (Figure 11).

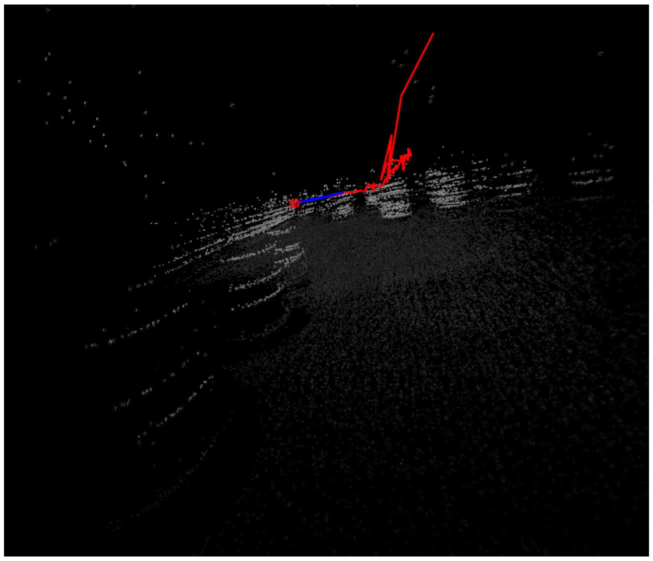


Figure 8: Points in the traffic cone maze as seen from the robot's prospective. The traffic cones are easily distinguishable to the human eye, indicating very little error in the direction of the points from the robot

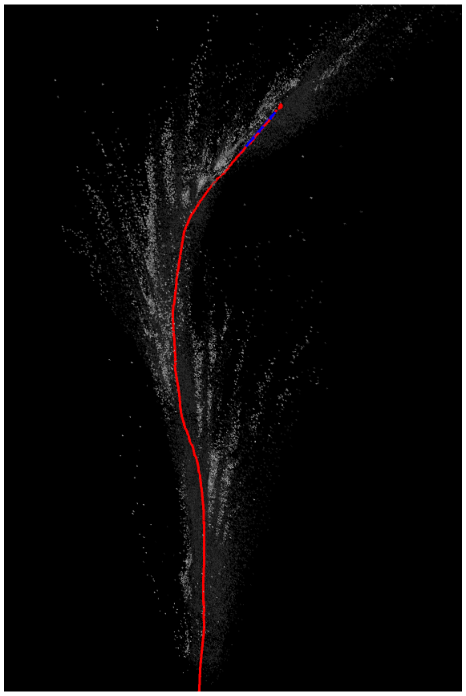


Figure 9: Points in the traffic cone maze as seen from above. The traffic cones are not easily visible to the human eye which, in conjuction with Figure 8, demonstrates significant error in the depths of these points

Miscellaneous

Some smaller issues also exist that may create difficulty in certain situations. One of these is the tendency of the monocular odometry algorithm to track points that may not be useful, such as points on the horizon. Because of the sharp contrast in color on the horizon, it will often be chosen as a good point for the monocular odometry algorithm to track. An example of this happening can be seen in Figure 13. This has two negative effects. First, it adds a considerable number of points to the point cloud that are not associated with any objects in the environment, thereby compounding the negative effects discussed in the previous paragraph. Examples of these points can be seen in Figure 12. Second, in areas where not many interesting points are available, the horizon points have a significant impact on the pose calculation. This will be another major challenge for operating in underwater environments. While a "horizon" in the traditional sense will likely not be visible, a similar effect will likely be seen with points on the water's surface if the robot is operating in shallow water. Because the environment underwater is often texture-depleted, these points will have a significant impact on the pose calculation.

Finally, we can see that SO-DSO does handle repetitive textures well. Figure 14a shows SO-DSO tracking along the side of the apartment wall. The underlying monocular vision algorithm is able to track using the points shown in the figure, and this does not cause large amounts of error in the estimated trajectory. Many stereo odometry algorithms would struggle in this scenerio as any point translated horizontally along the wall will look similar the starting point. This means that a stereo algorithm will not be able to match points in both cameras accurately. This same result can be seen when the algorithm is applied using datasets from the real world. Figure 14b shows the algorithm being applied to stereo camera data from an underwater robot moving through a shallow pool. SO-DSO is able to track the points on the pool floor despite the texture of the pool floor being highly repetitive.

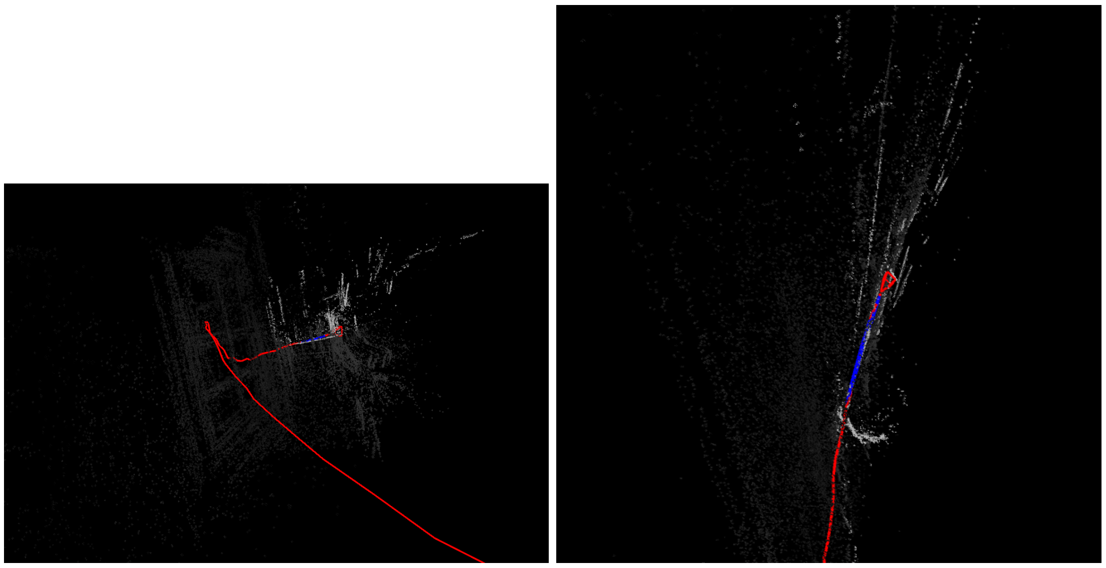


Figure 10: Points in the collapsed house trial as viewed from the robot (left) and from above (right). The window of the house is visible when viewing from the robot's perspective, but little of the building's structure is discernable when viewing from above.

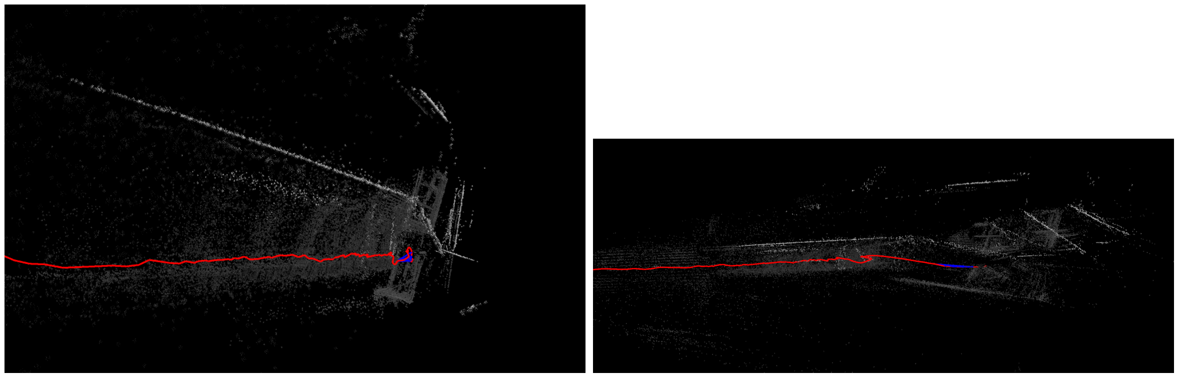


Figure 11: Points in the apartment trial as viewed from the robot (left) and viewing the apartment from the front perpendicular to the motion of the robot (right). The banister and window in the distance are both clearly visible from the robot's perspective, but very few features can be seen when looking from the other perspective.

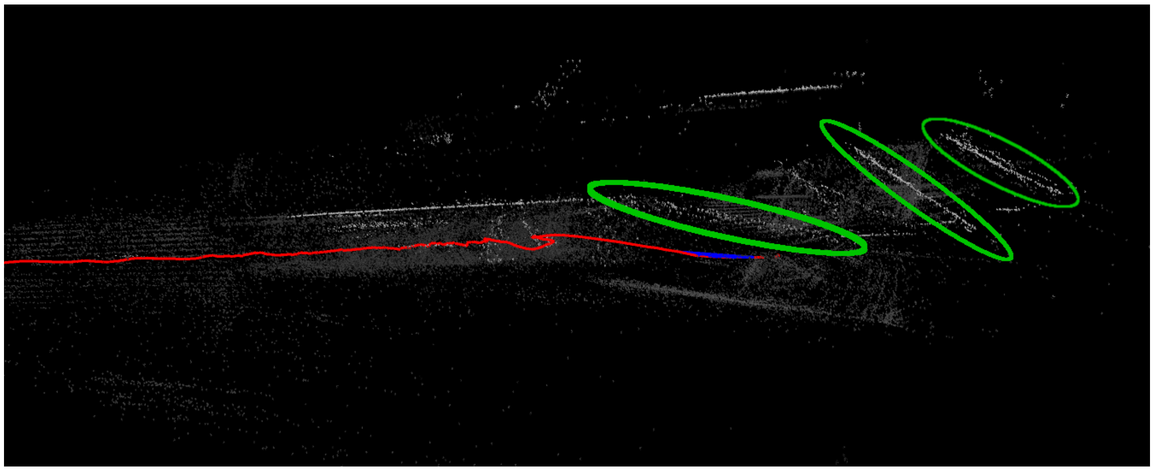


Figure 12: Points in the apartment trial viewed from the side. Examples of fake points generated by the horizon have been circled in green.

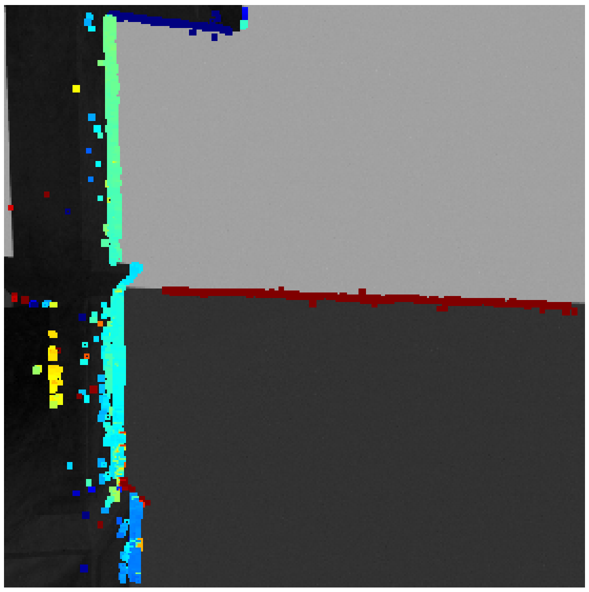


Figure 13: Points on the horizon also being selected when objects are mostly out of frame. These points are red, meaning that DSO believes they are closeby.

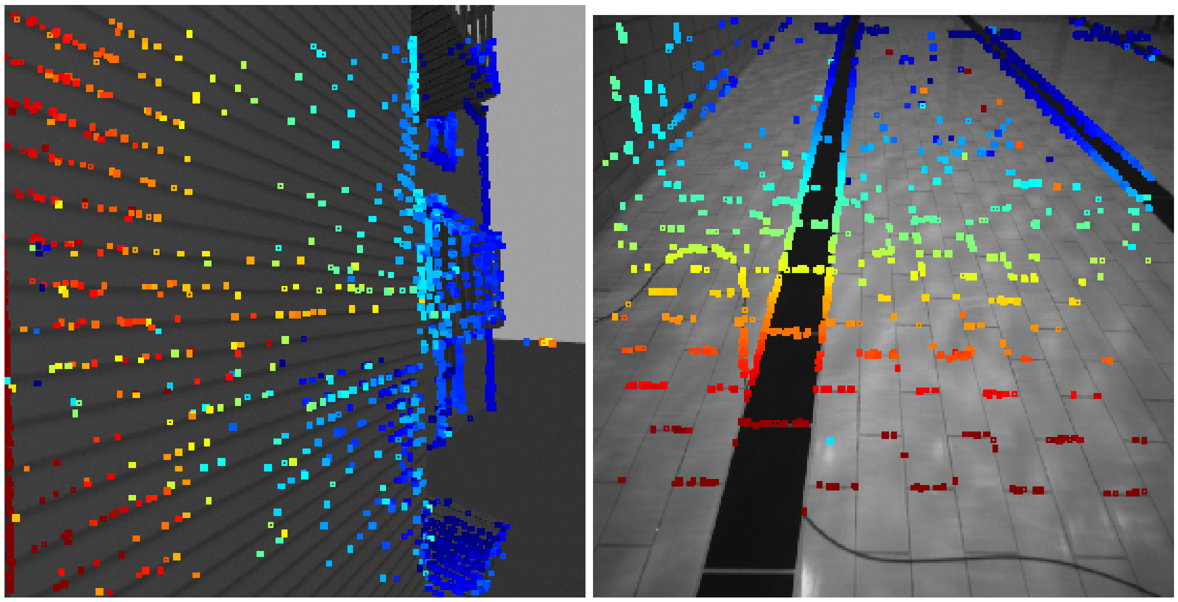

(a) Points in the apartment simulation. (b) Points on a pool floor from a real-life dataset.

Figure 14: SO-DSO is able to track points in environments with repetitive textures.

Conclusions

While SO-DSO shows promising results in many environments, there is still much improvment that needs to be made before it can be used reliably in difficult environments, such as underwater. The algorithm heavily relies on a large availability of candidate points for the underlying monocular odometry algorithm to track, which makes working in spare, texture-depleted environments very difficult. Other changes must also be made to increase robustness in general, such as being able to handle rotations without any translation.

However, it should be noted that these downsides are all primarily caused by the underlying monocular odometry algorithm. SO-DSO acts as it claims to: it provides increased accuracy in pose estimation while not encurring the large computational penalty of stereo matching. It does not fix the problems inherent in the underlying methodology; it merely inherits them. In environments such as the underwater domain, it is unclear whether the added performance boost is enough to justify the many problems that become exposed when working in these difficult environments. Any future work aimed at making SO-DSO a more robust tool in these domains should be focused on improving the underlying monocular vision. Further work is also needed to examine the benefits and downsides of using SO-DSO over more traditional, stereo matching methods to determine whether the added computational cost of stereo matching methods is justifiable.

References

1. J. Mo and J. Sattar, “Extending monocular visual odometry to stereo camera systems by scale optimization,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 11 2019, pp. 6921–6927.

2. G. Saygili, L. van der Maaten, and E. Hendriks, “Improving segment based stereo matching using SURF key points,” in *Proceedings of the 19th IEEE International Conference on Image Processing*, 09 2012, pp. 2973–2976.

3. J. Engel, V. Koltun, and D. Cremers, “Direct sparse odometry,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, 2018.

4. R. Gomez and J. Gonzlez-Jimnez, “Robust stereo visual odometry through a probabilistic combination of points and line segments,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 05 2016, pp. 2521–2526.

5. S. A. S. Mohamed, M.-H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila, “A survey on odometry for autonomous navigation systems,” *IEEE Access*, vol. 7, pp. 97 466–97 486, 2019.

6. S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, “Keyframe-based visual-inertial odometry using nonlinear optimization,” *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 03 2015.

7. R. Wang, M. Schworer, and D. Cremers, “Stereo DSO: Large-scale direct sparse visual odometry with stereo cameras,” in *Proceedings of the IEEE International Conference on Computer Vision*, 10 2017, pp. 3903–3911.

8. Z. Fu, Y. Quo, Z. Lin, and W. An, “FSVO: Semi-direct monocular visual odometry using fixed maps,” in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 2553–2557.

9. A. Ruszczyski, *Nonlinear Optimization*. Princeton University Press, 2006, vol. 13.

10. A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: the KITTI dataset,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 09 2013.

11. M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart, “The EuRoC micro aerial vehicle datasets,” *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 01 2016.

12. G. Dudek, P. Gigure, C. Prahacs, S. Saunderson, J. Sattar, L. A. Torres-Mndez, M. Jenkin, A. German, A. Hogue, A. Ripsman, J. Zacher, E. Milios, H. Liu, P. Zhang, M. Buehler, and C. Georgiades, “AQUA: An amphibious autonomous robot,” *Computer*, vol. 40, no. 1, pp. 46–53, 02 2007.

13. C. Fabbri, M. J. Islam, and J. Sattar, “Enhancing underwater imagery using generative adversarial networks,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 05 2018, pp. 7159–7165.

14. M. J. Islam, Y. Xia, and J. Sattar, “Fast underwater image enhancement for improved visual perception,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3227–3234, 2020.