Implementing Visual Odometry on the Raspbot

Landon Doyle, Max Edwards, Jared Hansen

The purpose of this report is to document the implementation of visual odometry on a robot and discuss the challenges encountered during the setup and operation of the visual odometry algorithm as well as our methodology and our outcomes. Our team aimed to deploy the visual odometry algorithm available at the GitHub repository: https://github.com/alishobeiri/Monocular-Video-Odometery. We decided to focus on getting the visual odometry algorithm implemented first, as we figured that would be the most time-consuming facet of the assignment, and also because we needed it for the midterm. After that we decided we would modify our existing teleoperation code from assignment 1 to capture a video we could process on a device external to the robot and compare our estimated trajectory and the output from the odometry algorithm.

Difficulties in Configuring and Running Visual Odometry Algorithm

Installation and Dependencies: The initial challenge and the part of the assignment we spent the most time on was deploying the visual odometry algorithm. We had already been struggling with this as part of the midterm, but we were finally able to get the algorithm working after capturing a sample video, creating some sample pose data, and using a much older version of opencv-python, 3.4.17 to be precise.

After getting the algorithm working, we took some time to look through the two files, test.py and monovideoodometery.py to understand how they were working and what arguments were expected.

After many attempts to get the algorithm working we were finally able to configure it to process and provide odometry on a short sample video we had recorded on a mobile phone, we then moved on to recording a video with our raspbot.

Reconfiguring Teleoperation Code

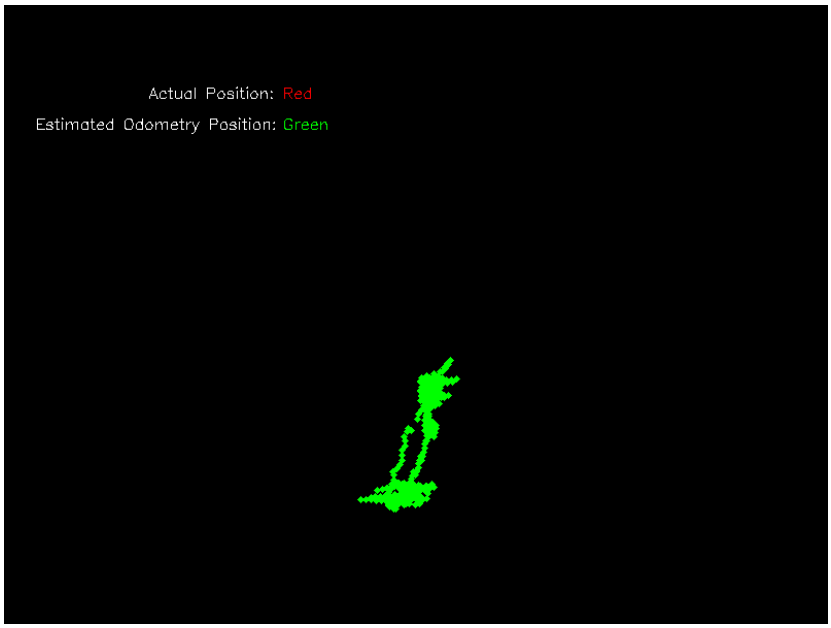
To capture and save live video from the robot, we modified our existing teleoperation code from the first assignment. This was fairly simple but there were complications with configuration of libraries to capture video off the robot. We also dealt with framerate issues as the robot processed teleoperation commands and recorded video at the same time. We were eventually able to capture a relatively smooth video as moving the robot around the circuit.

After recording the video on the robot we transferred it over to a laptop and used https://mconverter.eu/convert/avi/png/ to split our recorded .avi file into discrete .png frames. We then used this directory of roughly 900 frames to perform the visual odometry. During this process we ran into issues with some of the frames and ended up writing a script that would determine which frames the algorithm was breaking on and removing them from the directory. This enabled the algorithm to work as expected and left us with a relatively smooth video.

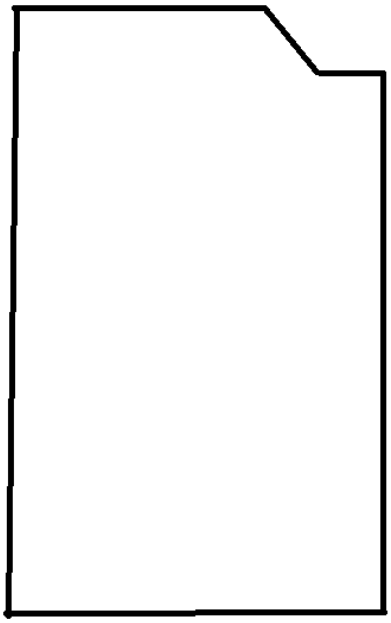
Discrepancies in Estimated Trajectory

After processing the recorded video through the visual odometry algorithm, we encountered a substantial difference between our estimated trajectory and the observed trajectory output by the algorithm.

Visual odometry trajectory



Predicted Trajectory



Several factors potentially contributed to this difference:

Calibration Accuracy: We did not calibrate the camera on the raspbot before hand and this potentially led to some issues.

Environmental Factors: The room we recorded the circuit in was not especially bright even with all the lights on, it also had a very patterned carpet that may have effected the estimated trajectory

Algorithm Limitations: The monocular visual odometry may not be as good of an odometry algorithm as others we could have chosen, but as we needed to use it for the midterm we decided to use it anyway to gain more practice

Control Methods: The places where the visual odometry algorithm is least accurate in its trajectory is during the turns, we could try different methods and speeds of turning the robot in the future to see if we could get clearer trajectory output.

Conclusion

Implementing visual odometry on the raspbot taught us several things, firstly it was a great accomplishment to finally get the visual odometry algorithm deployed, after continued technical and configuration challenges. This assignment also gave us further experience with the raspbot, as well as an opportunity to use its camera which will help us with the final project we plan to do. Even if the visual odometry output was not very close to our predicted output, it still roughly represents the circuit and we have some concrete ideas of how we could get better performance in the future. The final lesson we learned in doing this assignment is some of the challenges that come from the limited processing power of

the raspberry pi on the raspbot, these lessons have convinced us that we need to figure out a way to perform live offboard processing if we are going to be able to accomplish the goals of our final project.