3D Scene Reconstruction with a Mobile Camera

Robert Carrera and Rohan Khanna

Stanford University: CS 231A

1 Introduction

Autonomous supernumerary arms, or "third arms", while still unconventional, hold promise for use as prostheses, as workplace assistants, or as assistants for the disabled.[7] Currently, wheelchair users use static or adjustable arms to hold computers and books and simple powered arms can even play drums. However, much work has to be done for supernumerary arms to be able to cooperatively and intelligently assist a user.

A prerequisite for these tasks is the ability to locate and move to a target in the environment without collisions. To do so, the arm must be able to perform simultaneous localization and mapping (SLAM) to map its environment and situate itself within the environment. Ideally, the supernumerary arm would be low-cost, low-weight and low-power in order to increase battery life, increase user comfort (in the case of a wearable arm), and increase affordability. To this end, we propose a monocular SLAM that creates a sparse map of the environment using robot position information to reduce the computational demand of standard monocular structure from motion (SFM). Monocular SLAM requires less equipment than stereo or RGB-D methods, reducing the weight and power consumption of the arm. The scale-ambiguity of monocular SLAM is a source of drift and error, but it also allows for use in a wide range of scene depths, as opposed to stereo or RGB-D methods. We desire this versatility from an arm that would be used daily by users in a variety of situations. We compare the results of vision-only SFM to the results of our visual-inertial SFM.

2 Previous Work

A variety of monocular SLAM approaches exist, which vary most fundamentally in their approaches to camera pose estimation. Feature-based methods use keypoints in each frame and correspondences between frames to triangulate a 3D point map. Within feature-based methods, both filtering-based and keyframe-based approaches can be used. As the camera pose changes, collects a new frame, and triangulates new 3D points, an

SFM algorithm would ideally minimize the global reprojection error to all past frames. However, a global optimization using all collected frames would quickly grow to be too computationally demanding for practical use.

In the filtering-based approach, 3D point locations are updated to be related to the most recent pose, and all explicit information of past pose location is discarded. While this approach does not store more pose information over time, relating all the 3D points in the global map to the most recent frame grows in computational expense as the number of 3D points grows. The monocular SLAM algorithm of Eade and Drummond provides one example of a filtering-based approach.

In the keyframe-based approach, a global optimization becomes feasible by removing all of the past poses except for a relatively sparse set of keyframes. The keyframe based approach has been shown to yield more accurate results except in the most computationally limited situations.[1] Parallel tracking and mapping (PTAM) was one of the first keyframe-based algorithms to allow real-time SLAM on a mobile monocular camera, and at the time of its publication was able to achieve state-of-the-art results.[3]

Direct approaches, instead of using features to find pose change between frames, find a transformation that minimizes the difference in pixel intensity between frames. By using pixel intensity instead of features, direct methods use all of the information available in the image. Direct methods typically produce more accurate results in scenes with little texture or with motion blur or camera defocus, but they are also more difficult to implement.[2] One notable example is large-scale direct SLAM (LSD-SLAM).[6]

Hybrid approaches, such as the efficient semi-direct visual odometry (SVO), have also been shown to work well. SVO uses pixel intensity matching to get an initial pose estimate and then refines both the pose and 3D point estimates by keyframe-based optimization. SVO can run in real time even on embedded systems, but it does not build a global map. Instead, it discards map information from scenery the camera has “passed”, since this method was developed for odometry during autonomous drone flight.

A great deal of work exists that examines visual-inertial SLAM, the incorporation of velocity and acceleration data from sensors into the SLAM algorithm. Often, these methods use the inertial measurement unit (IMU) to obtain an initial estimate of the relative camera pose between frames and then use keypoints for pose refinement and triangulation. However, we have found few works discussing the use of robot position information and visual information to construct a 3D map. The lack of published work on this topic may be due to the fact that the position information found in high-end non-mobile industrial robots is accurate enough to use without vision-based refinement, rendering the camera pose estimation step of SFM trivial. However, we are interested in seeing how and if lower quality position information from a low-cost non-mobile arm can augment vision-based camera pose estimation.

2.1 Proposed Method (Key Contributions)

We choose a feature-based SFM with keyframes because this method has been shown to achieve reasonably accurate results and is also manageable to implement. While this method is not novel, its integration with robot-position information obtained from a low-cost robot and a comparison with visual-based methods alone has to our knowledge never been attempted. The value of this study is therefore in studying whether or not lower-quality position information can be used in affordable vision-capable supernumerary arms.

3 Technical Solution

3.1 Technical Summary

We implement a vision-only SFM and position-augmented SFM pipeline that automatically detects features and correspondences, computes and refines the fundamental matrix F, and carries out triangulation and bundle adjustment on a captured set of frames. Position information is injected at the nonlinear refinement by triangulation step of the pipeline, replacing the RANSAC step and cutting down on computation time.

3.2 Methods

3.2.1 Feature extraction and correspondence detection

Our implementation uses SIFT features because of their invariant properties and because a FAST feature detector and a corner detector both returned too few features for tracking in our scene. While FAST features have been used successfully in monocular SLAM algorithms such as PTAM or a localization algorithm like SVO, these features may have failed in our implementation due to the choice of our scene objects. For our SIFT features, feature correspondences were found using a brute-force method.

3.2.2 Calculating F

After obtaining feature correspondences, we calculate the fundamental matrix F using the normalized 8-point algorithm and RANSAC to minimize the epipolar constraint Sampson distance error. The 8-point algorithm uses the epipolar constraint

$$x_i'^T F x_i = 0 \qquad \text{(Eqn. 1)}$$

in order to construct a linear system of equations

$$Af = 0 \qquad \text{(Eqn. 2)}$$

where x_i', x_i are a pair of image correspondences and f is a vector composed of the elements of F. The above equation can be solved for F using singular value decomposition (SVD). The set of correspondences are normalized to have a centroid at the origin and a root-mean square distance of $\sqrt{2}$ from the origin. During each iteration of RANSAC, eight correspondences are normalized again and used to compute F. Then first order Sampson distance approximations are made to the geometric error of the correspondences, computed from Eqn. 1, where the Sampson distance cost for each correspondence is

$$\frac{\left(x_i'^T F x_i\right)^2}{(Fx_i)_1^2+(Fx_i)_2^2+\left(F^T x_i'\right)_1^2+\left(F^T x_i'\right)_2^2} \qquad \text{(Eqn. 3)}$$

We choose the F with the largest set of inliers, as defined by a threshold of 0.005 pixels Sampson distance. After triangulation is carried out via the method of 3.2.4., F is further refined using the Gold-Standard iterative nonlinear minimization of the geometric reprojection error, using the Levenberg-Marquardt algorithm.

3.2.3 Triangulation and global map building

After refining F with the Levenberg-Marquardt algorithm, we can again perform triangulation. To do so, we first compute the essential matrix E from F, using the relation

$$K^T F K = E \qquad \text{(Eqn. 4)}$$

From E we compute four possible choices of the normalized camera projection matrix from the singular value decomposition of E. For each choice, we begin with a linear estimate of the triangulation derived from the relation of each 3D point to its projection

$$x_i = P_i X_i \qquad \text{(Eqn. 5)}$$

We can then refine this linear estimate with a non-linear minimization of the reprojection error, using ten iterations of the Newton-step with the linear estimate as an initial estimate.

3.2.4 Bundle adjustment of keyframes

To perform an optimization of the global map and to eliminate scale drift, bundle adjustment is carried out on the set of keyframes. Due to time constraints, we use the bundle adjustment made available that we utilized in class.

While a full implementation of the pipeline would carry out bundle adjustment on a set of keyframes, we make a simplifying assumption that each captured image from our relatively small set of images is a keyframe. While this would be untenable in real-time, it is trivial to dispose of non-keyframe maps before carrying out the bundle adjustment step.

3.2.5 Injecting Robot Position Information

To incorporate robot position information, we can skip directly to the non-linear refinement of F using Levenberg-Marquardt, as we can obtain the rotation and translation using forward kinematics. Given a robot arm length of 17.7 cm from the base servo to the camera, and a base height of 7.62 cm, from the ground to the base servo, and the servo angular position q_{A^1} for the turntable driving servo and q_B for the arm driving servo, we can find the rigid body rotation from frame to frame as

$$\begin{matrix} f_1 \\ f_2 \end{matrix} R = \begin{matrix} f_1 \\ \text{base} \end{matrix} R \begin{matrix} \text{base} \\ f_2 \end{matrix} R$$

and the rigid body translation from frame to frame as

$${}^{f_2}T_{f_1,f_2} = \begin{matrix} f_2 \\ \text{base} \end{matrix} R\left({}^{\text{base}}T_{\text{base},f_2} - {}^{\text{base}}T_{\text{base},f_1}\right)$$

Where ${}^{\text{base}}T_{\text{base},f_1}$ is the rigid body transformation from the base to the end effector in frame 1 and using forward kinematics of our robot can be found to be:

$${}^{\text{base}}T_{\text{base},f_1} = \begin{bmatrix} \cos q_A \cos q_B & \sin q_B & -\sin q_A \cos q_B & L_B \cos q_A \cos q_B \\ -\cos q_A \sin q_B & \cos q_B & \sin q_A \sin q_B & L_B \sin q_B + L_A \\ \sin q_A & 0 & \cos q_A & -L_B \sin q_A \cos q_B \end{bmatrix}$$

Where L_A is the base height, and L_B is the arm length.
We use this rigid body rotation and translation between frames to calculate E as

$$E = [t]_x R$$

And subsequently find F using

$$K^{-T} E K^{-1} = F$$

Then this F is refined directly using the reprojection error and Levenberg-Marquardt, after an initial triangulation. Feature matching between frames is still carried out via a brute force method.

4 Experiments and Results

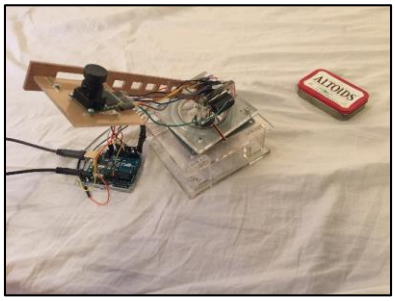

Figure 1. Experiment setup with manipulator, camera, microcontroller, and obstacle

We built a simple, low-cost robot out of lasercut acrylic, an Arduino Uno, a small steel turntable, and two servo motors capable of 180° rotation. An ArduCam ((OV5642) model type) was mounted to the end-effector. Camera position information in coordinates relative to the robot center was retrieved from the servo motor position command using forward kinematics. The robot captured photographs of the scene and recorded the current servo position command at each frame. Our experimental setup is pictured in Figure 1. Programming the Arduino Uno was done through the Arduino IDE. The ArduCam sent image data to the Arduino Uno via SPI and I2C was used to check the internal ArduCam fifo register. This data was displayed to the user using a program supplied by the developers of ArduCam and then was saved to disk to be processed by MATLAB. We were forced to manually take the picture at each configuration, as real time operation was much too slow for automatic image collection to be feasible. The servo angles for the arm to sweep through were hard coded into the program and in MATLAB.

To calibrate our camera, we used the MATLAB Camera Calibration Toolbox. The camera was held fixed while the checkerboard pattern was varied as can be seen in Figure 2. The distance of the patterns from the camera was chosen to be about the distance the camera would be from obstacles. The average reprojection error was ~0.535 pixels.

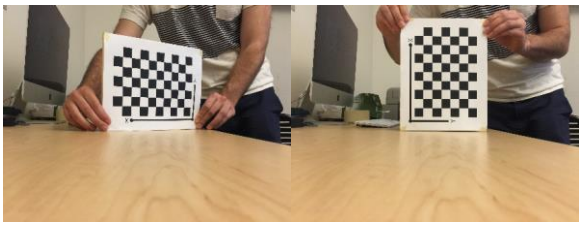

Figure 2. Images used in camera calibration

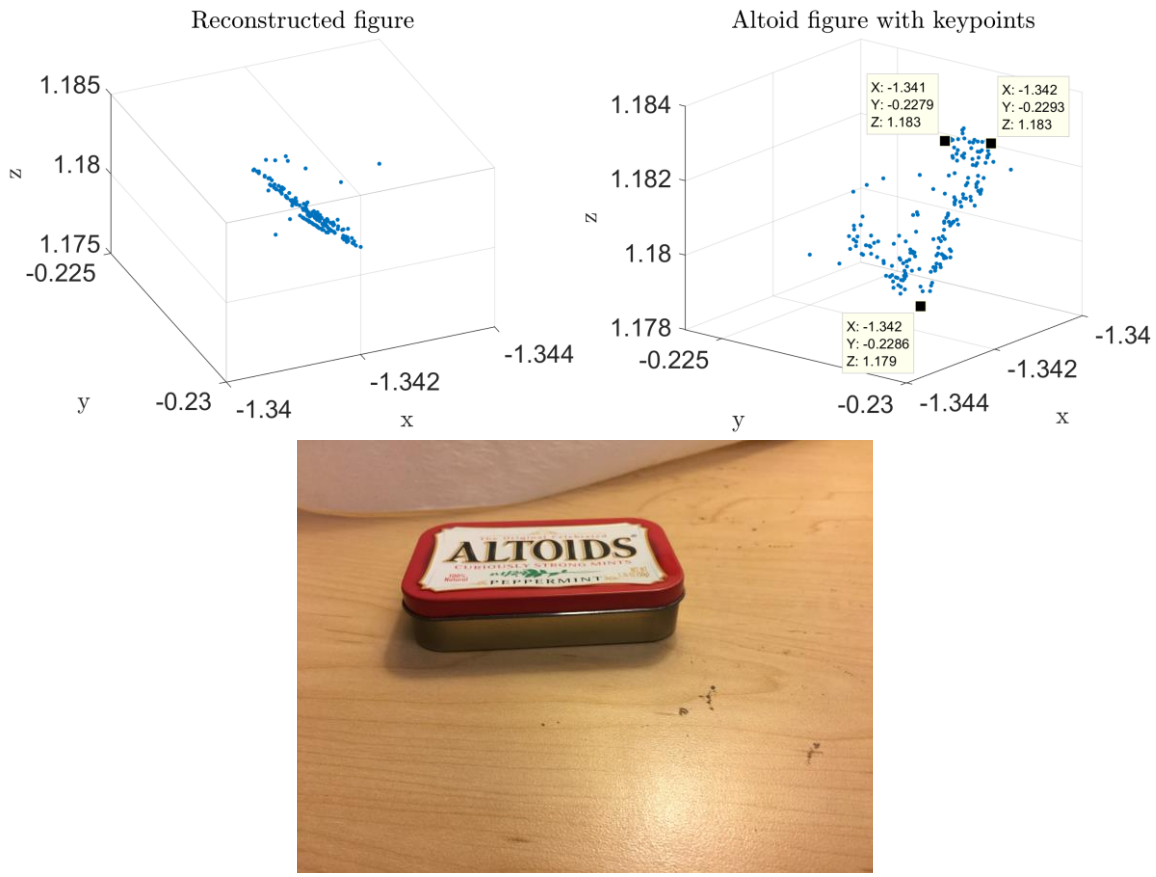

Figure 3. (left) The reconstruction, seen from an angle that illustrates the planarity of the reconstructed surface. (center) A representative frame from the frame set used for reconstruction. (right) The reconstruction, seen from an angle that shows a corner of the tin (bottom) and the relative dimensions of the tin.

The Arduino program, and the ArduCam program handled data collection, while the MATLAB SFM pipeline handled data processing. All images in the configuration were collected and saved to disk and then processed by the pipeline. The pipeline consisted of automatic SIFT feature detection, using RANSAC to estimate F and determine inliers in the feature set, then using Levenberg-Marquardt with the geometric error to refine F. Then we determined the transformation between camera frames using a nonlinear Newton step to determine the true transformation. All of the above code, asides from the ArduCam program, we wrote entirely ourselves. We slightly modified bundle adjustment and merge all frames as provided in class to decrease run-time. Additionally, we used vlfeat to get SIFT features and correspondences between SIFT features.

We ran our SFM pipeline with the images obtained by the robot with vision-only SFM and with position-augmented SFM. We show the results of the vision-only reconstruction on an Altoids tin in Figure 3. We also show the results of the vision-only pipeline on the statue image dataset from class.

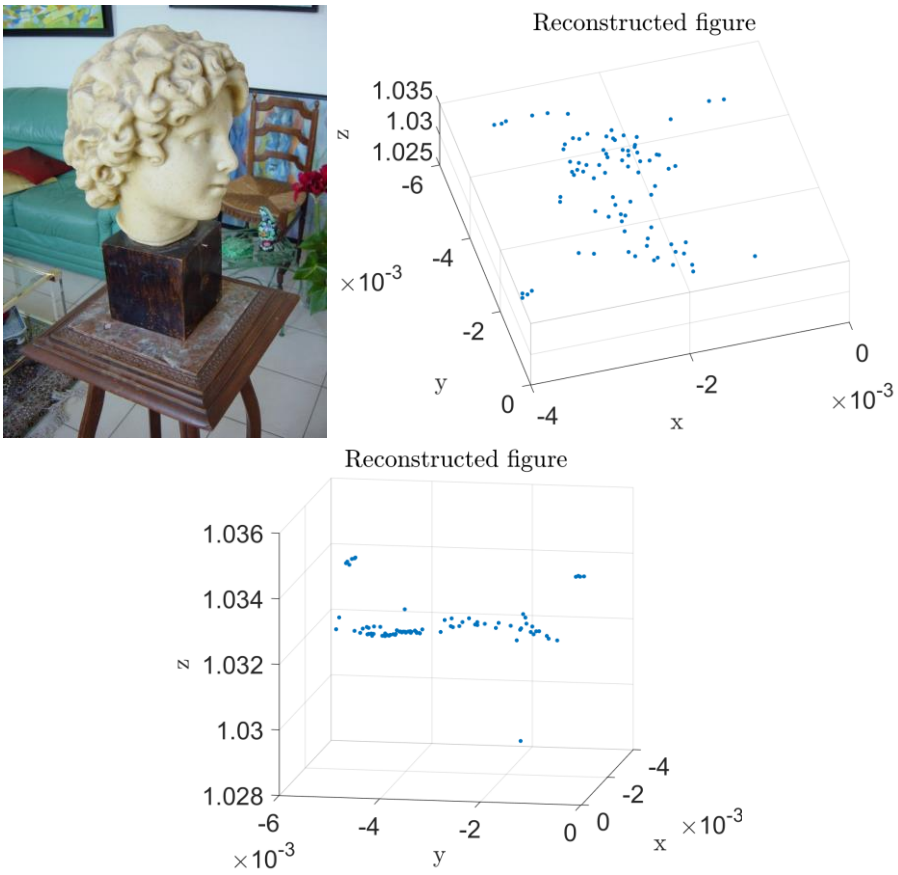

Figure 4. (left) A representative frame from the sculpture reconstruction dataset. (right) The reconstructed statue as seen from the front. A narrower neck, thicker head, and even hair is visible. (bottom) The sculpture as seen from the side, nose facing down and top of the head facing left. A facial cusp, or protrusion out from the neck, is visible.

We were also interested in decreasing runtime so we initially considered a faster feature detection such as corner detection. Using MATLAB's implementation of corner detection via minEigenFeatures we obtained the reconstruction in Figure 5. This is markedly worse than the reconstruction in Figure 3 and is not suitable for our pipeline. Although it is terminates in under 2 minutes, it produces a poor quality reconstruction. For this reason we were forced to use SIFT, which runs for a longer time, but produces a higher quality reconstruction.

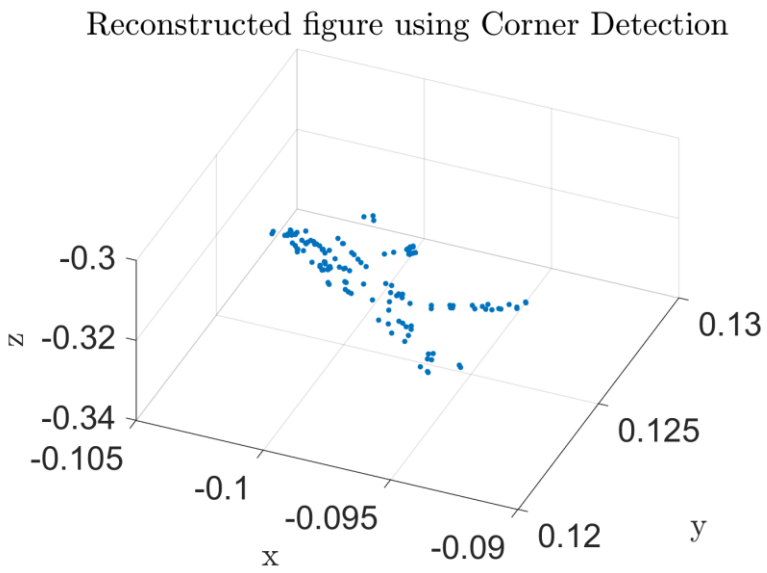

Figure 5. Reconstruction of the Altoid using minEigenFeatures—a corner detector.

No meaningful results were obtained from the robot position data. The inlier set obtained from the Sampson error threshold was simply too sparse to obtain meaningful results. Possible fixes are discussed in future work. Additionally, each run through the SFM pipeline took between 10 – 20 minutes. Implications of these results will be discussed in the next section.

5 Discussion

5.1 Conclusions

Our results indicate that our visual-only SFM pipeline reconstructs a given scene with reasonable accuracy. While we did not obtain useful data on how robot position data influences the reconstruction results relative to the vision-only pipeline, we believe that doing so would contribute to the general knowledge of how position information from lower-cost robots can be utilized in the SLAM problem. We believe that the robotic testbed we developed is an effective system for conducting further study.

5.2 Future Work

Much future work remains to fully implement the pipeline and to better quantify the vision-only SFM results and the position-augmented vision SFM. The addition of

automatic keyframe generation based on a heuristic, such as a total translation from the last keyframe greater than the average scene depth, would be a necessary part of a real-time SFM pipeline. We would also have to modify our camera pose estimation such that, in each new frame, we search for features that correspond to features in the preceding keyframe that also correspond to points in the global map, and minimize the corresponding reprojection error. Currently, the pipeline performs bundle adjustment on all of images in the set, which creates a consistent global map but which bears too high of a computational cost for real-time SFM.

We did not obtain useful results for quantifying the performance of the position-augmented SFM pipeline. One possible reason for this is that we immediately try to fit correspondences to the epipolar constraints of the F obtained from the robot position. We might have obtained better results if we had used a set of inliers obtained from searching along epipolar lines as a correspondence set for RANSAC and performed repeated rounds of RANSAC and guided matching to attain more feature correspondences, in order to refine F. Our current injection point for the robot data might cause the algorithm to be too dependent on the robot position information. This modified algorithm would still save time computing feature correspondences, as we could use epipolar geometry to constrain our search, instead of using brute force.

We could also create a figure showing a ground truth trajectory plotted along with the pose estimates from the vision-only algorithm. The results would be an indicator of the quality of our pose estimation in the vision-only pipeline. Time constraints did not permit the creation of this figure.

Our code is available on Github at <https://github.com/rkhanna24/CS231A>.

6 References

1. Strasdat, Hauke, J. M M Montiel, and Andrew J. Davison. "Real-time Monocular SLAM: Why Filter?" *2010 IEEE International Conference on Robotics and Automation* (2010). Web.
2. Forster, Christian, Matia Pizzoli, and Davide Scaramuzza. "SVO: Fast Semi-direct Monocular Visual Odometry." *2014 IEEE International Conference on Robotics and Automation (ICRA)* (2014). Web.
3. Klein, Georg, and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality* (2007). Web.
4. Eade, Ethan, and Tom Drummond. Monocular SLAM as a Graph of Coalesced Observations. *2007 IEEE 11th International Conference on Computer Vision* (2007). Web.
5. Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge, UK: Cambridge UP, 2003. Print.
6. Engel, Jakob, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. *Computer Vision – ECCV 2014 Lecture Notes in Computer Science* (2014): 834-49. Web.