Realtime Structure from Motion with QR Detection

EMMANUEL GALLEGOS* and CARL GUO*, University of Illinois at Urbana-Champaign, USA

This work successfully generates extremely fast (under 1 second) 3D affine structure from motion using dual live-feeds from cameras. By leveraging fast QR code detectors, an approximate 3d point cloud can be generated for scenes to use in realtime structure from motion applications, such as in the field of robotics for live collision detection by measuring distances in the scene.

CCS Concepts: • **Artificial Intelligence → Computer Vision**; Robotics.

Additional Key Words and Phrases: vanishing point detection, ransac, 3d scene reconstruction, classical computer vision

ACM Reference Format:
Emmanuel Gallegos and Carl Guo. 2022. Realtime Structure from Motion with QR Detection. *J. ACM* 1, 1, Article 1 (December 2022), 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

The problem of obtaining structure from motion has been extensively studied, and several pipelines exist that allow for some kind of 3D construction using multiple views of a given scene. Some of these pipelines seek to create as robust of 3D models as possible, including the creation of meshes, potentially crowdsourcing from hundreds or even thousands of images from the web of given scenes[1].

In the realm of robotics, however, speed is often of more importance than robustness. For example, in an environment with multiple agents coordinating to perform tasks, algorithms from the field of task and motion planning attempt to generate collision free paths for robots. However, in actual execution, robotic motion can be susceptible to hardware noise, faulty controls, and general environmental instability. For these types of environments, it is important to have systems in place that can detect impending collisions in real-time before they happen.

To do this, however, requires knowing the positions of both robots in 3D space at any given time. Thus, our system seeks to leverage live feeds from multiple cameras in a scene to perform real-time 3D scene reconstruction in order to measure the distance between robots in a scene, and create an alert if the robots appear to be on a collision course. We do this by leveraging fast detection models that are trained to detect QR codes, and placing these codes in a scene to allow for extremely fast structure from motion.

*Both authors contributed equally to this research.

Authors' address: Emmanuel Gallegos, eg11@illinois.edu; Carl Guo, carlguo2@illinois.edu, University of Illinois at Urbana-Champaign , 201 N. Goodwin Ave, Urbana, Illinois, USA, 61801.

0004-5411/2022/12-ART1 $15.00
https://doi.org/XXXXXXX.XXXXXXX

2 SYSTEM OVERVIEW

2.1 Hardware

All computations were performed on personal computers, from basic laptops, Macbooks, and a modern gaming computer. We used the built in webcam in our laptops as well as an external USB webcam to create live streams of image data of the same scene.

2.2 Physical Set-Up

We tested our system in two locations. One was in a living room setting, using static images that were taken of a carefully constructed scene as shown in the following figure:

Fig. 1. Living Room Setup

We used this system as a proof-of-concept, in order to show that our calculations were correct in measuring real-world 3D distances. Images were taken of the scene from multiple angles and used to generate affine 3D structure from motion, which was used to calculate relevant distances in the scene.

Fig. 2. Office Setup

Next, we moved into an office environment, as shown in **Figure 2**, and used the multiple camera setup to generate a live dual feed of the same scene from two static cameras. For this experiment, one camera was placed on the office chair and the other was the Macbook built-in webcam.

We placed the QR codes in the scene and measured the ground truth distances between the two QR codes with a ruler, as well as the dimensions of the QR codes. The dimensions of the QR codes, as well as the live image feeds are the only inputs into our algorithm, which then outputs a live feed showing the realtime distance between the two QR codes.

In a real world scenario, these QR codes could be taped or otherwise mounted to robots, and high-resolution cameras could be mounted to the ceilings or walls.

2.3 Image Processing Pipeline

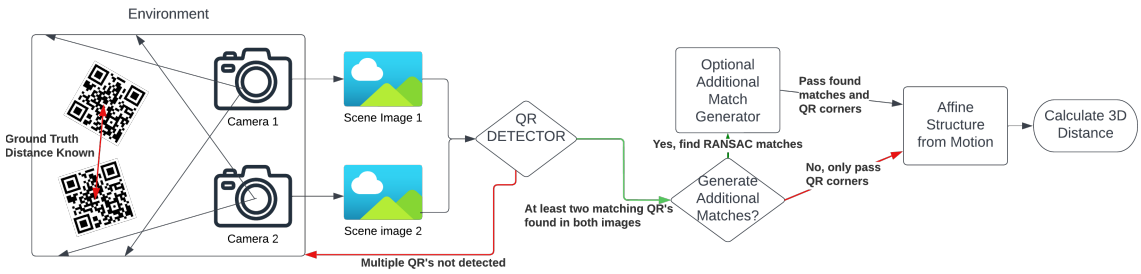

Fig. 3. System Overview

The basic pipeline of the code can be seen in Figure 3 is as follows:

(1) A live feed from two webcams is continuously generated by our program using OpenCV.
(2) If both frames are successfully captured, they are both converted to grayscale and passed through a QR code detector using OpenCV. The output of the QR code detector is the list of any QR detections, the image coordinates of the four corners of the code, as well as some other information.
(3) If the QR codes corresponding to both robots are found in both frames, then the coordinates of the 16 total QR corners, as well as both original image frames, are passed into a module that performs structure from motion.
(4) We have an optional pipeline that generates other matches between the two scenes in addition to the QR code corner correspondences. In this pipeline, candidate matches are generated using ORB descriptors, and passed into a homography estimation module that uses RANSAC to fine-tune the subset of candidate matches we wish to keep. The methods used to find these matches are described in [2]. The rationale is that if the matches generate a homography between the two images with many inliers and low residuals, then the inliers are probably good matches for other uses, like for use in structure from motion. If this optional module is used, the matches generated with the homography estimation are used as additional input along with the QR code corner correspondences, which are known to be reliable, as the QR detector model is quite robust. These inlier matches can help give additional 3D info in the scene if the two QR codes are coplanar or otherwise give little 3D scene information, at the cost of time and potential noise if poor matches are found.
(5) The data matrix which consists of our QR code corner matches and, optionally, the matches generated from homography estimation, are then used to generate 3d coordinates for the scenes with affine structure from motion, attempting to eliminate the affine ambiguity using a method based on the method described in [3].
(6) We then use the known dimensions of the QR codes, and the calculated 3d coordinates of the corners to estimate a ratio of calculated distance in our 3D model to real-world 3D distance. We then calculate the distance between the two QR code centers in the 3D model, and divide by the model-to-real-world ratio to get an estimate of the real-world 3D distance between the two QR code centers in centimeters.

(7) If this distance appears too small, or if the distance appears to be shrinking, an alarm could theoretically be set-off, or a signal sent to disable the robots, etc. We did not implement this step for our setup, but in a real environment this addition would be easy to integrate.

3 RESULTS

3.1 Still-Image SFM

After running the ORB descriptor on the still images, we notice a potential issue due to a large majority of the putative matches being concentrated on the QR code regions. We were concerned that this could possibly make our 3D reconstruction inaccurate. The descriptor matches can be seen in the figure below:

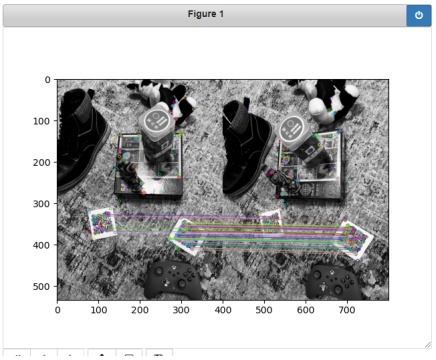

Fig. 4. ORB Descriptor Home Matches

However, despite this concern, we were able to get a fairly accurate distance measurement between the centers of the QR codes from the 3D reconstruction. Our ground truth distance was **27.4 cm**, and the distance we recovered from our 3D reconstruction was **25.97 cm**. This measurement results in an accuracy of **94.8%**.

To validate our result, we re-projected the RANSAC inlier matched points back onto the image. We measure the total re-projection residuals for these points to be **14.46 pixels** using a sum of squared distance metric. In **Figure 5**, on the left is the 3D reconstruction of the RANSAC inliers with the estimated distance. The QR corners are represented as orange dots and green triangles, and the red dots connected by the line are our estimated QR code centers; the rest of the blue dots are the other inliers. On the right side, we show the re-projected inliers along with the original coordinates of the image overlaid on top of the still image in grayscale. The original points are represented as blue dots and the re-projected points as orange triangles.

3.2 Live-Feed SFM

We ran our structure from motion pipeline on ten frames from both cameras using the setup shown above in **Figure 2**. To measure the accuracy of our 3D reconstruction, similar to our still-image experiment, we estimated the distance between the center of our two QR codes for each pair of frames. The ground truth distance between the QR codes was set to be 30 cm. We ran this experiment twice, once excluding the RANSAC step to find inlier matches and another including it. Throughout the process, we also record the frames from both cameras and a 3d reconstruction of the matches from both images. An example of these outputs can be seen in **Figure 6**. In the figure, it can be seen that we use the two frames we generate from our cameras to capture the same scene

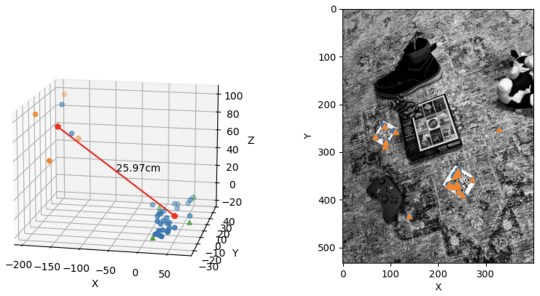

Fig. 5. Still Image Distance Results and Residuals

but from different angles and distances. Using the known QR corner coordinates and keypoint matches, we can estimate the distance between the QR code centers fairly accurately.

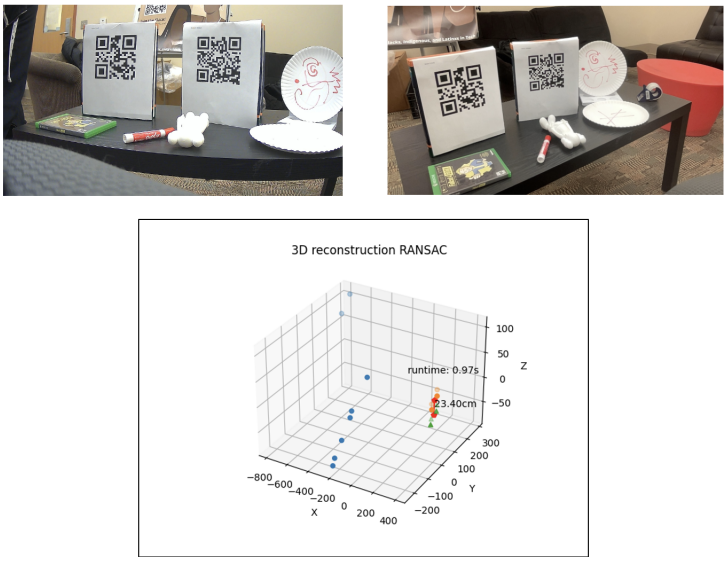

Fig. 6. Top Left: external camera frame, Top right: Macbook camera frame, Bottom: 3D reconstruction with estimated distance between QR codes (23.4cm) and runtime (0.97s)

We display both the estimated distance between the two QR codes and the runtime of the pipeline in **Table 1** and **Table 2**. In the results of our live feed experiment, we saw a noticeable difference in distance estimation if we leveraged RANSAC to fine tune candidate matches. For distance estimations without the RANSAC step, the recorded standard deviation was **5.17**, the average was **17.64**, and the total residual using Sum of Squared Differences (SSD) was **42.07**. If the optional fine tuning was included, we observed the data to have a standard deviation of **2.94**, mean of **23.5**, and total residual of **24.47**. From these metrics, we can see that using RANSAC and homography estimation to fine tune keypoint matches resulted in more precise and accurate distance estimations. For both tests, the average runtime to calculate 3D coordinates and derive the distance between the QR codes was a little under 1 second.

Estimated Distance (cm)	Runtime (s)
16.42	1.0
23.56	0.95
14.4	0.95
13.96	0.96
26.86	0.95
15.52	0.96
13.44	0.96
24.33	0.96
13.36	0.96
14.51	0.96

Table 1. Live Feed Non-RANSAC Results

Estimated Distance (cm)	Runtime (s)
21.78	1.01
22.26	0.97
23.98	0.97
22.29	0.97
20.9	0.97
23.4	0.97
18.72	0.97
26.39	0.97
27.19	0.98
27.96	0.97

Table 2. Live Feed RANSAC Results

4 DISCUSSION + CONCLUSION

Our proposal for fine-tuning candidate matches by applying RANSAC with homography estimations resulted in more consistent and accurate distance estimations (and by extension) 3D projections. However, the distance estimation still is not perfect, which could be due to the concern we mentioned earlier about most of the putative matches being from regions on the QR code. Furthermore, in some of our live feed tests, we encountered some occasions where, when trying to recover a structure from motion Q matrix using Cholesky decomposition, the calculation failed due to our estimated matrix not being positive definite. To solve the issue of matches being biased towards QR codes, one possible solution could be to apply a mask to the ORB detector, to tell it to avoid looking for matches in the QR region. To deal with the issue of positive definite matrices, we simply ignored the frames if a proper Cholesky decomposition could not be calculated, and adjusted the angle or position of the QR codes. This would be analogous to simply waiting for one second and hoping that on the next set of image frames, either the RANSAC loop will generate better matches or the robots will move in such a way to make the QR codes more visible. Another sensible solution could be to prompt the cameras to make small angle changes to recorrect the frame if too many consecutive iterations fail to generate a valid 3D scene.

In the above work, we show that our method of using QR code detectors to generate approximate 3D clouds through performing affine structure from motion runs extremely quickly and was able to function properly with multiple real-time live feed data streams. These results can be applied to quickly measure distance between objects in a scene, which can be leveraged for live collision detection or other realtime applications.

5 STATEMENT OF INDIVIDUAL CONTRIBUTION

Emmanuel was the primary developer, though much of the final code used was adapted from code from two assignments from this semester, namely Assignment 3 Part 1 (Homography estimation for image alignment), which drew methods from [2] and Assignment 5 Part 1 (Affine structure from motion), which drew methods from [3].

Carl was the primary experiment coordinator, setting up the experiments, and adding additional code to time and document the results of the experiments. Both of us ultimately worked on this project with overall approximately equal contributions.

REFERENCES

[1] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M. Seitz. 2007. Multi-View Stereo for Community Photo Collections. In *2007 IEEE 11th International Conference on Computer Vision*. 1–8. https://doi.org/10.1109/ICCV.2007.4408933

[2] Richard Hartley and Andrew Zisserman. 2003. *Multiple View Geometry in Computer Vision* (2 ed.). Cambridge University Press, New York, NY, USA.

[3] Carlo Tomasi and Takeo Kanade. 1992. Shape and Motion from Image Streams under Orthography: A Factorization Method. *Int. J. Comput. Vision* 9, 2 (nov 1992), 137–154. https://doi.org/10.1007/BF00129684

Received 9 December 2022