JPL D-28523

Mars Science Laboratory (MSL) Focused Technology Program

2-D Target Tracking Technology Validation Report

Revision 1

Prepared By:
Won S. Kim
Robert C. Steinke
Robert D. Steele

Document Custodian:
Won S. Kim

Paper copies of this document may not be current and should not be relied on for official purposes. The current version is in the MSL Project Library at http://msl-lib.jpl.nasa.gov, in the Rover Validation Test Reports folder.

April 2004

Jet Propulsion Laboratory
4800 Oak Grove Drive
Pasadena, CA 91109-8099

MSL Rover Technology Validation Report: 2-D Target Tracking

Signature Sheet

Approval

Won S. Kim, Task Manager, Instrument Placement Valdiation — Date 4/26/04
Jet Propulsion Laboratory

Larry H. Matthies, Supervisor, Machine Vision Group (3482) — Date 5/3/04
Jet Propulsion Laboratory

Issa A. Nesnas, Task Manager, Visual Tracking & CLARAty Test Beds — Date 5/3/04
Jet Propulsion Laboratory

Stephen F. Peters, MSL Mission Data System (MDS) — Date 5/6/04
Jet Propulsion Laboratory

Richard A. Volpe, Mars Technology Office — Date 5/6/04
Jet Propulsion Laboratory

Revision History

Revision	Date	Description	Author
Initial Release	12/2003	Initial release	Kim, Steinke, Steele
Rev. 1	4/2004	1. Incorporated numerous comments of the reviewers for the affine tracker 2. Added SDOG tracker experiments 3. Added an overview of target approach system	Kim, Steinke, Steele

Related Documents

Document Name	Date	Author
CLARAty Delivery for 2-D Visual Tracking	4/2003	Nesnas, Bajracharya, Bandari, Bualat
Ames Delivery for SDOG Visual Tracking	2/2004	Bualat, Kunz
Camera Calibration and Stereo Vision Technology Validation Report	1/2004	Kim, Steinke, Steele, Ansar

Scope

The scope of this document is to report the test and validation of two visual target tracking algorithms: affine and SDOG (sign of difference of Gaussian) 2-D trackers. This algorithm validation process supports technology selection process before flight qualification of the software selected. The test validation matrix for the 2-D tracker software is shown below.

Test Validation Matrix for the Affine and SDOG 2-D Trackers

Test item	Affine 2-D Tracker		SDOG 2-D Tracker	
	Software control parameters	Tracking reliability and error	Software control parameters	Tracking reliability and error
Transform	X	X		
Tracking window size	X	X		
Number of pyramid levels	X	X		
Template update	X	X		
Frame confidence threshold			X	X
Template confidence threshold			X	X
Different camera motions		X		X
Different image-size rocks		X		X
Lighting variation		X		X
Maximum image displacement		X		X

Acknowledgment

This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. We would like to acknowledge that Issa Nesnas and Max Bajracharya of JPL and Esfandiar Bandari and Maria Bualat of NASA Ames provided the CLARAty affine 2-D tracker software, and Maria Bualat and Clay Kunz of NASA Ames provided the CLARAty SDOG 2-D tracker software package. We would like to thank the technology providers for various technical discussions.

Contents

Signature Sheet…… 1
Revision History…… 2
Related Documents…… 2
Scope…… 3
Acknowledgment……4

Summary……7

1 Introduction…… 10

2 An Overview of the Target Approach System…….. 12
2.1 Theoretical Computations of Target Approach Accuracy…….. 12
2.2 Target Approach System with 2-D/3-D Tracker and Camera View Handoff……13
2.3 Hypothetical Calculations of Target Tracking Percentage and Placement Error……....14

3 Test Plan and Experimental Setup…… 17
3.1 Test Plan…… 17
3.2 Experimental Control Variables……19
3.3 Experimental Setup…… 19

4 Affine Tracker Software…… 21
4.1 Functional Description…… 21
4.2 Reconfiguration of the Affine Tracker…… 24
4.3 Tracking Performance Metrics…….. 24
4.4 Adding Supporting Capabilities……..... 26
4.5 Software Bug Findings and Fixes……28

5 Affine Tracker Experimental Procedure…….. 30

6 Affine Tracker Performances…… 33
6.1 Transforms…… 33
6.2 Tracking Configuration…… 37
6.3 Window Size and Pyramid Level…….. 40
6.4 Template Window Update…… 43
6.5 Maximum Image Displacement for Active Camera Control…… 47
6.6 Lighting Changes…… 50
6.7 Tracking with Oblique and Sideway Camera Motion…… 54
6.8 Tracking with Actual Rocky8 Mast Camera Images…… 56
6.9 Average Tracking Performances from Forward, Roll, and Yaw Camera Motions.... 57
6.10 Tracking with Small-Image-Size Rocks…… 60
6.11 Forward-Motion Tracking with Large-Image-Size Rocks…… 62
6.12 Roll-Motion Tracking with Large-Image-Size Rocks…… 66
6.13 Yaw-Motion Tracking with Large-Image-Size Rocks……69
6.14 Alternate Template Update Method…… 75

7 SDOG Tracker Performances…… 77
7.1 Software Functional Description…… 77

7.2 Forward-Motion Tracking……. 77
7.3 Roll and Yaw Motion Tracking…… 84
7.4 Lighting Changes…… 86
7.5 Maximum Image Displacement for Active Camera Control……..…… 86
7.6 Execution Timing……88

8 Improving Tracking Performance…… 89

9 Conclusion……..……. 90

References…….. 91

Summary

As part of the Instrument Placement Technology Validation Task, this report details experimental results of 2-D visual target tracking. Two visual target tracking software packages used in our test and validation were the pyramidal Lucas-Kanade affine tracker and the SDOG (sign of difference of Gaussians) tracker. Both were tested within the CLARAty (Coupled Layer Architecture for Robotic Autonomy) software environment. Since the 2-D tracking software did not support active camera control, forward motion tests were limited to 4 m. The full 10-m forward motion tests will be performed when the 2-D/3-D tracking software with active camera control is available.

Performance evaluation of the pyramidal Lucas-Kanade affine tracker:

1. **Pure translation tracked more reliably, while affine transform tracked more accurately**. Pure translation matching tracked more reliably than affine transform matching, but drifted more and was less accurate. Affine transform matching tracked more accurately when it tracked, but lost targets more easily than pure translation.
2. **A combined configuration of pure translation followed by affine transform performed best**. Based on the above observation, the software was re-configured to allow a combined tracking configuration of pure translation followed by affine monitoring/correction. This combined tracking configuration demonstrated best tracking performance, and thus we conducted the test and validation of the "affine" tracker using this combined configuration.
3. **Low-texture targets needed larger window sizes**. The ideal window size was affected by two sources of error. If the window is too small, there may not be enough texture to track well. If the window is too large, the terrain may violate the planarity assumption of the affine tracker and not track well. Experiments indicated that typically 15×15 pixels appeared to be a good size for small rocks, and 29×29 pixels for large rocks.
4. **Increasing the number of pyramid levels helped to track larger image displacements between images**. Large image displacements require a large effective window size to prevent a target feature from leaving the tracking window entirely. As the number of pyramid levels increases by one, the effective window size doubles. For example, a 15×15 window size with 4 pyramid levels has an effective window size of $15\times2^3 = 120$ pixels at the top pyramid level. Likewise, 29×29 window size with 3 pyramid levels has the similar effective window size. Experimental results showed that combinations of widow size and pyramid levels having similar window size yielded similar tracking performance in terms of handling large image displacements between images. In terms of texture, however, the 15×15 window performed better for high-texture targets, while the 29×29 window was better for low-texture targets on large-image-size rocks.
5. **The best template update was every 5% to 10% change in distance to target or every 5º to 10º change in roll and yaw motions**. Upper bounds for the template window update interval were about 2 m for straightforward motion at about 10-m target distance (20% change in distance to target) and 20º for roll and yaw motions. However, the optimal update interval depends upon targets and other factors. Typically, every 50 cm change at 10-m target distance (5% change in distance to target) for forward motion and every 5º to 10 º change in roll or yaw motion appeared to be good template update intervals. Beyond monitoring the distance and orientation changes, it seems to be a good idea to monitor the affine tracker ssd (sum of squared differences) threshold to trigger the template update.
6. **The affine tracker using the 15×15 window with 3 pyramid levels tracked up to about 30 pixels as the maximum image displacement.** Tracking experiments with various image skips to change image displacement sizes showed that up to about 30 pixels image displacements were tracked for a 15×15 window with 3 pyramid levels. The

maximum displacement dictates the tracking range of the active camera control. We will investigate the maximum image displacement more carefully when we test the 2-D/3-D tracker supporting active camera control, since it is an important factor that affects the tracking reliability and determines the maximum rover motion between images.

7. **Tracking performance was fair with unfavorable sunlight directions, while it was poor with dramatic sunlight changes.** The 9-AM images made dark shadow on the rock surfaces facing the camera. Even though the lighting and shadow condition for 9 AM was significantly poorer than those of 2 PM and 4 PM, the tracking performance for 9 AM was pretty good, demonstrating fair tracking performance even with unfavorable sunlight directions. In the next experiment, there were dramatic changes in ambient sunlight when patches of opaque clouds moved across the sun. For 8-mm images, automatic gain control (AGC) was helpful to tracking by maintaining the average intensity of the camera image relatively constant, resulting in 44% tracking percentage. For 4-mm images, AGC was not helpful due to "bright" sky in view, resulting in 0% tracking. Incidentally, this dramatic lighting change due to dark clouds would not happen in Martian environment.
8. **Tracking tests with Rocky8 mast cameras demonstrated the necessity of active camera control.** Targets tracked well for the first 1.8 m until the rover went over a rock, which caused large image displacements (sometimes over 100 pixels). By 2.4 m, all targets were lost. This clearly indicates that active camera control with 2-D/3-D tracking is essential.
9. **Average tracking performances were above 80% up to 100% with the 15×15 window for all forward, roll, and yaw camera motions.** Targets on small rocks tracked 100%, while several targets on large rocks were lost with the 15×15 window size. Forward-motion tests were limited to 4 m, since full 10-m forward-motion tests require active camera control.
10. **Increasing the window size to 29×29 with 4 pyramid levels resulted in near 100% tracking for low-texture targets on large rocks, and main causes of tracking failures were identified.** Targets on large rocks often did not have enough texture to track well with small windows. When the window size was increased to 29×29, the tracking performance improved to or near 100%. Increasing the number of pyramid levels to 4 from 3 also helped tracking some targets that often had large image displacements between images. Tracking failures occurred when there were problems with occlusions during the course of the tracking, significant background changes beyond occluding boundaries of the target rock, shadow changes, large image displacements, and target windows containing quite non-planar objects such as two or more portions of separate rocks.
11. **Potential improvements for tracking reliability were suggested**. Items include: 1) active camera control with 2-D/3-D tracking, 2) brute-force translation matching based on normalized-cross-correlation or DOG (difference of Gaussian) filter, 3) reliable target loss detection, 4) template update triggered by some thresholds, 5) selective-region target window where the target may be off-centered.

Performance evaluation of the SDOG tracker and comparison with the affine tracker:

1. **Peak tracking percentage resulted with 60% to 70% for the frame confidence threshold and 80% for the template confidence threshold**. For forward motion 70% frame confidence and 80% template thresholds were best, while 60% frame confidence and 80% template thresholds were best for roll and yaw motions.
2. **For forward motion, SDOG tracker performed slightly better than the affine tracker.** For forward motions with 8-mm and 16-mm lenses, SDOG tracker yielded 96% to 100% tracking depending upon confidence threshold values, while the affine tracker

yielded 89% to 98% tracking depending upon window size and number of pyramid levels.

3. **For the roll and yaw motions, the affine tracker performed slightly better than SDOG tracker.** For 8-mm-lens roll motion, SDOG tracker yielded 56% to 94%, while the affine tracker yielded 100%. For 8-mm-lens yaw motion, both SDOG and affine trackers yielded 100% tracking.
4. **SDOG tracker was robust for dramatic sunlight changes.** The SDOG tracker yielded 100% tracking for both 4-mm and 8-mm lenses with dramatic sunlight changes, while the affine tracker yielded 0% tracking for the 4-mm lens and 44% for the 8-mm lens.
5. **The maximum image displacement of the SDOG tracker was rather limited**. The maximum image displacement of the SDOG tracker was about 20 pixels, worse than 30 pixels for the affine tracker of the window size 15 with 3 pyramid levels. This result was surprising, considering the SDOG tracker does brute-force search. More investigation is planned in the next 2D/3D tracker test and validation.
6. **The affine matching took about 4 times longer than SDOG matching.** The execution time for the combined pure translation matching followed by affine matching was 20 ms, while the execution time for the SDOG matching was 5 ms in spite of the "brute-force" search as opposed to "iterative" search. This demonstrates the software implementation of the SDOG matching algorithm is computationally very efficient.

Software bug findings:
Tracking performance with the initial software delivery of the affine tracker was poor. Through collaborative efforts with technology providers, several software bugs were identified. The final software delivery after three critical bug fixes showed dramatic performance improvements. Five major bugs identified including the first three critical ones were listed here.

1. **Partial derivatives inverse matrices at different pyramid levels**. Partial derivatives inverse matrices were not retrieved correctly at lower pyramid levels. This messed up feature matching at lower pyramid levels.
2. **Delta translation updates between pyramid levels.** The delta translation updates were not halved when the pyramid level for matching went down, and not doubled when it went up. This also messed up feature matching at lower pyramid levels.
3. **Interchange of width and height.** In two places, image width and height were swapped. This prevented tracking beyond the 768-pixel column of the 1024-column image.
4. **Crash after long runs with a large number of targets.** There are memory leaks in Feature_Window and Image_Pyramid classes. Objects created with new() during class constructions could not be deleted upon class destructions. This bug not fixed yet.
5. **Crash for track windows at the image boundary.** The program crashes when target windows are at the image boundary. This bug not fixed yet.

In this report, experiments results were obtained by using the final affine tracker software delivery after the fixes of the first three critical bugs. The last two bugs remained to be fixed, but they should not affect the affine tracker experimental results.

1. Introduction

At present, the baseline operation of the Mars Exploration Rover (MER'03) flight mission depicts the state of art technology for instrument placement on Mars. As illustrated in Figure 1, if a target rock is about 10 m away from the rover, the MER baseline operation requires 3 sols to place an instrument on the target position of the rock. After surveying the panoramic images received from the rover, scientists select a target rock to explore that might be 10 m away from the rover. Scientists then decide a waypoint that is, for example, about 2 m away from the target, and send a go-to-waypoint command to the rover. The rover moves 8 m, takes images, and sends them to Earth. Using these images, scientists and ground operators determine the rover base or anchor position where the target rock is within arm's reach, and send the corresponding go-to-waypoint command. The rover then travels 2 m reaching the target, takes close-up target images, and sends them to Earth. Scientists and ground operators determine the manipulator motion commands and send them to the rover. The manipulator moves according to the motion commands, placing an instrument on the target position of the rock.

In Mars Science Laboratory (MSL'09) enhanced operation, it is desirable to achieve the entire target approach and instrument placement from 10 m away in a single sol, as shown in Figure 2. Reducing the 3-sol baseline operation to a single-sol operation increases Mars science return significantly. According to an MSL baseline document [Steltzner 2003], 8 to 10 sols are expected to be spent per rock on the average, assuming 3-sol baseline instrument placement. With the enhanced single-sol instrument placement capability, 8 to 10 sols/rock will be reduced to 6 to 8 sols/rock, resulting in 20% to 25% increase in science return.

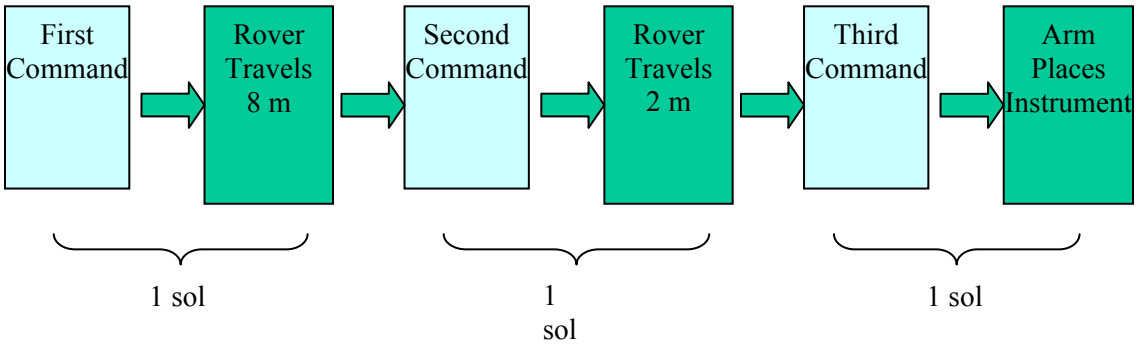


Figure 1. Mars Exploration Rover (MER'03) baseline operation requires 3 sols to place an instrument on a rock from 10 m away

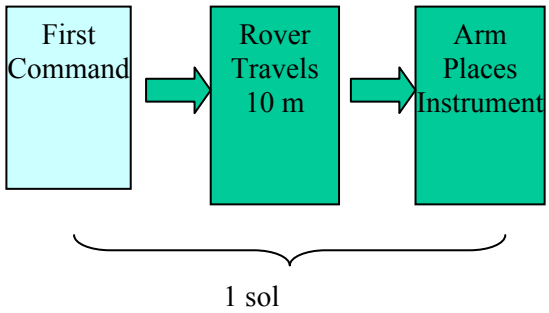


Figure 2. Mars Science Laboratory (MSL'09) enhanced operation requires a single sol to place an instrument on a rock from 10 m away.

The single-sol target approach and instrument placement is technically challenging. In particular, the operation must be fail-safe and reliable. Target approach and instrument placement technologies were demonstrated earlier for some experimental conditions. However, fail-safe, reliable operations have not been demonstrated yet. Extensive experiments are necessary to produce fail-safe, reliable operations for target approach and instrument placement.

Three main technologies to achieve single-sol target approach and instrument placement operations (Figure 3) are 1) visual target tracking for approach, 2) stereo vision, and 2) manipulation to place an instrument. Previously, we tested and validated the JPL stereo vision software [Kim, Steinke, Steele, Ansar, 2003]. In this report we describe the test and validation of the 2-D target tracking software, which is a critical portion of the 2-D/3-D target tracking software. Two 2-D target tracking software packages were evaluated: affine [Bajracharya et. al., 2003] and SDOG [Bualat, et. al., 2004] trackers. In the next report we will test and validate the full 2-D/3-D tracking software [Nesnas et. al., 2004] supporting active camera control.

Following the MSL technology infusion process guideline, we tested the 2-D target tracking software packages within the CLARAty (Coupled Layer Architecture for Robotic Autonomy) test bed. CLARAty [Volpe et. al., 2000 & 2001; Nesnas, et. al., 2000; Estlin et. al., 2001] provides a common software environment that enables implementing comprehensive control for planetary rovers and robotic systems.

Section 2 describes an overview of the target approach system, and Section 3 presents the test plan and experimental setup. Sections 4, 5, and 6 describe the affine tracker software, experimental procedure, and tracking performance results, respectively. Thereafter, Section 7 describes SDOG tracker software and tracking performance results in comparison with the affine tracker. Section 8 lists suggestions for potential improvements of tracking performance. Finally, Section 9 is the conclusion.

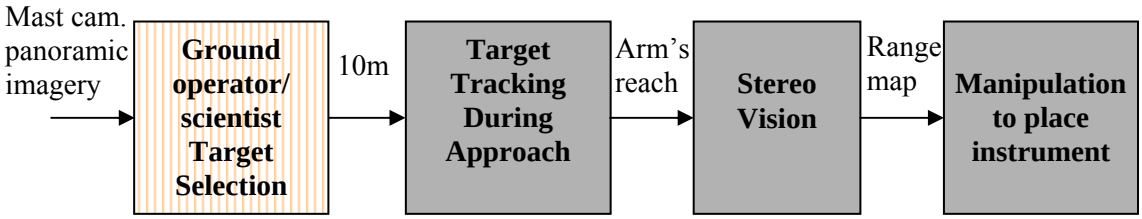


Figure 3. Three technology components to achieve single-sol target approach and instrument placement operations

2. An Overview of the Target Approach System

2.1 Theoretical Computations of Target Approach Accuracy

First we performed theoretical computations of the target approach accuracy with and without a visual target tracker. If no target tracker is used, two main factors contributing to the target positioning error after 10-m travel are the stereo range "sensing" error and the rover navigation error. The stereo range error ΔR is computed by

$$\Delta R = \frac{R^2}{f_p B} \Delta d \ ,$$

where B is the stereo baseline, R is the range, and f_s is the camera's focal length in pixels. The camera's focal length *f* can be converted from *mm* to pixels by

$$f_p = \frac{f}{pixel\ size} \ .$$

In our experiments, we used the Point Grey Research's Dragonfly high-resolution digital camera equipped with the Sony ICX204AL 1/3-inch CCD image sensor of 1024×768 pixels resolution. Since the image size of the 1/3-inch format CCD is approximately 4.8 mm × 3.6 mm, we can estimate the pixel size by dividing the horizontal CCD size (4.8 mm) by the horizontal image resolution (1024 pixels), resulting in 4.69 μm. According to the manufacture's specification which is more accurate than the above estimate, its pixel size is 4.65 μm × 4.65 μm. Thus, in computing the stereo range error for Table 1, we used the focal length conversion from *mm* to pixels by $f_p = f / 0.00465$. Further, the stereo disparity error Δd is assumed to be 1 pixel for the 3σ stereo range error [Kim et al, 2003]. Based on the above relations, stereo range errors for different cameras at 10-m distance are computed and listed in the fourth column of Table 1. The stereo range error $\Delta R_{stereo,10m}$ at a 10-m distance is 10.4 cm for 16-mm Pancam and 202.2 cm for 2.3-mm Hazcam.

If no visual target tracking is used, the target positioning error after 10-m travel to the target is root-sum-square (RSS) of the stereo range "sensing" error at 10 m and the rover navigation estimation error for 10 m travel.

$$\Delta R_{no_tracking,10m} = \sqrt{\Delta R_{stereo,10m}{}^2 + \Delta R_{nav,10m}{}^2} \ .$$

If we assume the rover navigation error based on the wheel and visual odometers is roughly 2% of the rover travel distance, the navigation error $\Delta R_{nav,10m}$ for 10 m travel is 20 cm. The computation results are listed in the fifth column of Table 1. The target positioning errors without visual tracking are all more than 20 cm.

Focal length (1/3" CCD camera)	Field of view angles	Stereo baseline	Stereo range error (3σ) at 10 m distance	Target approach error (3σ) with 2% navigation error	Target approach error (3σ) with ideal visual tracking and camera handoff
16 mm	17° × 13°	28 cm	10.4 cm	22.5 cm	1.5 cm
6 mm	49° × 37°	20 cm	38.8 cm	43.7 cm	3.9 cm
4 mm	65° × 49°	20 cm	58.1 cm	61.4 cm	5.8 cm
2.3 mm	113° × 86°	10 cm	202.2 cm	203.2 cm	10.1 cm

Table 1. Target approach error with and without visual target tracker for different camera lenses.

When a visual target tracker is employed, the target positioning error can be greatly reduced. Assuming the stereo-based manipulation is performed at 1-m distance, the positioning error of the target which is being tracked over camera images is approximately determined by the stereo range error at 1 m distance. Since a 1-pixel (±0.5-pixel) image expands to a 10-pixel (±5-pixel) image as the camera approaches the target from 10 m away to 1 m away, a 1-pixel target position accuracy at 10 m away can yield a ±5-pixel accuracy in 1 m away. Since the target position cannot estimated more accurately than ±5 pixels at 1 m distance, the 3σ error stereo disparity error Δd = 5 pixels at 1 m distance if there was no camera handoff. However, assuming stereo-based manipulation uses Hazcam (2.3 mm), camera handoff from Pancam (16 mm) or Navcam (6 mm or 4 mm) to Hazcam (2.3 mm) is generally required. When the camera focal length is reduced from f_{s1} to f_{s2}, the corresponding stereo disparity error is also reduced by a factor of f_{s2}/f_{s1}. Therefore, the stereo range error is given by

$$\Delta R = \frac{R^2}{f_{s2}\, B_2}\left(\Delta d\, \frac{f_{s2}}{f_{s1}}\right) = \frac{R^2}{f_{s1}\, B_2}\Delta d \ ,$$

where f_{s1} is the focal length of the initial camera used at 10 m away, B_2 is the baseline of the final camera after handoff, and Δd = 5 pixels for 3σ error value. This formula provides the theoretical target approach positioning error assuming ideal visual tracking with perfect camera handoff. The results are shown in sixth column of Table 1. The theoretical 3-σ error of the target positioning is 1.5 cm with ideal visual tracking starting with Pancam and handing off to Navcam and then finally to Hazcam.

2.2 Target Approach System with 2-D/3-D Tracker and Camera View Handoff

A functional diagram of the target approach system consisting of the 2-D/3-D tracking and camera handoff software is shown in Figure 4. The 2-D tracking module is a feature image matcher, and does not require stereo camera views. The 2-D/3-D tracker is essentially the 2-D tracker with active camera control using stereo camera views. Active camera control points the camera to the target each time when the rover moves to a new position, so that the target is positioned at the center in subsequent camera images. Active camera control prevents the target moving out of the camera view, and greatly reduces the search area for the 2-D tracker feature image matching. Pointing the camera to the target requires the knowledge of the target position in 3-D space relative to the rover position. The rover pose estimator such as the visual odometer provides the rover pose estimate, while the triangulation of the target image points in stereo camera views provides the target position estimate in 3-D space.

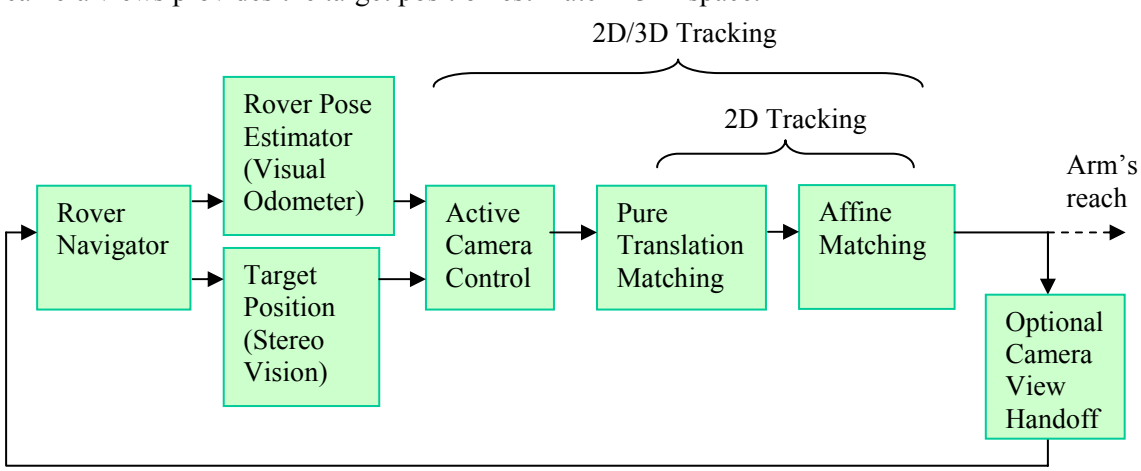


Figure 4. Target approach system with 2-D/3-D tracking and camera view handoff

2.3 Hypothetical Calculations of Target Tracking Percentage and Placement Error

For the given target approach system in Figure 4, we made an attempt to hypothetically calculate the overall target tracking percentage and placement error of the system. Table 2 depicts the system component and its associated errors affecting the tracking performance. As the rover navigator moves the rover, the resulting camera motion changes the target image size and orientation, which affect the tracking performance of the 2-D tracker. Large changes in target image size and orientation lower the tracking percentage and increase the error.

The active camera control adjusts the fixed-mast pan/tilt angles for the Pancam and Navcam to point to the target. It uses the Visual Odometer (VO) based rover pose estimator and stereo vision based target position estimator to compute the desired pan and tilt angles. The rover pose and target position estimation errors together with pan/tilt mechanical errors determine how much the target image is off from the center of the next image. The target image displacement affects the tracking performance of the 2-D tracker. Large image displacements lower the tracking percentage and increase the error.

Finally, the tracking performance of the 2-D tracker is described by the tracking percentage and error. It depends upon various experimental conditions, software algorithms, and software parameter settings, including the target image size and orientation change and the target image displacement as mentioned above. The camera view handoff error is also needed to compute the overall target tracking performance of the system.

In our hypothetical calculations, the following baseline operational scenario is assumed.

1. Pancam for 4 m (from 10 m to 6 m)
2. Handoff from Pancam to Navcam
3. Navcam for 4m (from 6 m to 2 m)
4. Handoff from Navcam to Hazcam
5. Hazcam for 1m (from 2 m to 1 m)
6. Anchor rover and place instrument

Further, since target tracking performance depends upon experimental conditions, important experimental test variables are listed here.

- Straight flat, rocky, or winding path
- Image collection interval
- High-texture or low-texture targets
- Lighting conditions
- Software algorithms and configuration
- Software parameter settings

Table 3 is a hypothetical example of a simple error budget chart, comparing three experimental conditions: 1) straight forward motion on a flat-surface terrain, 2) straightforward motion on a terrain with small rocks that the rover can go over, and 3) winding or zigzag motion to avoid large-rock obstacles. All assumed to collect images every 20 cm. The tracking performance numbers in Table 3 are simply very rough approximate guesses, and actual experimental numbers may be quite different. However, this hypothetical calculation exercise helps to understand the overall system performance and experimental designs.

Software system component Baseline operational scenario: 1. Pancam for 4 m (from 10 m to 6 m) 2. Handoff from Pancam to Navcam 3. Navcam for 4m (from 6 m to 2 m) 4. Handoff from Navcam to Hazcam 5. Hazcam for 1m (from 2 m to 1 m) 6. Anchor rover and place instrument	Error budget for target approach Experimental test variables: • Image collection/process interval • Straight flat, rocky, or winding path • High-texture or low-texture targets • Lighting conditions • Software algorithms and configuration • Software parameter settings
Rover navigator	Rover motion changes the target image, affecting the matching performance: • Target image size change • Target image roll, pitch, yaw changes
Rover pose estimator using visual odometer (VO)	VO estimation error affect active camera control: • Rover pose distance error • Rover pose orientation error
Target position estimation using stereo vision	Stereo vision triangulation error affects active camera control: • Target position error on image plane
Active camera control to point the fixed-mast to the target (for Pancam and Navcam only; use active gaze control for Hazcam)	Fixed-mast pointing errors: • pan/tilt encoder resolution • pan/tilt backlash
2-D target tracking using a combined configuration of pure translation matching followed by affine matching	The above active camera control with VO and stereo vision determines the target image displacement, which affects the tracking performance: • Tracking percentage • Tracking error Effects of various experimental test variables on 2-D tracker tracking performance are described in this report.
Camera handoff	• Handoff success percentage • Handoff error

Table 2. Target approach software system components and their error contributions to tracking performance for error budget analysis.

Terrain	Flat	Small rocks	Large rocks
Approach path	straight	straight	winding
Image frame interval	20 cm	20 cm	20 cm or 10º
Rover navigator ▪ Size change per frame ▪ Pitch change ▪ Yaw change	 2% at 10m 10% at 2 m – –	 2% at 10 m 10% at 2 m 10º –	 2% at 10m 10% at 2m 10º 10º
VO rover pose ▪ Distance error (2%) ▪ Orientation error (2%)	 0.4 cm 0.1º	 0.4 cm 0.2º	 0.4 cm 0.3º
Target position error on image plane	1 pixel	1 pixel	1 pixel
Pan/tilt (540:1, 16 CPR) ▪ encoder resolution ▪ backlash	 0.04º 0.03º	 0.04º 0.03º	 0.04º 0.03º
Overall orientation error for active camera control	0.1º	0.2º	0.3º
Target image displacement between frames ▪ Pancam (17º FOV) ▪ Navcam (45º FOV) ▪ Hazcam (100º FOV) with active gaze	 6 pixels 2.3 pixels 1 pixel	 12 pixels 4.6 pixels 2 pixels	 18 pixels 9.2 pixels 3 pixels
2-D target tracking and camera handoff (tracking percentage and error each step) 1. Pancam for 4 m (from 10 m to 6 m) 2. Handoff from Pancam to Navcam 3. Navcam for 4m (from 6 m to 2 m) 4. Handoff from Navcam to Hazcam 5. Hazcam for 1m (from 2 m to 1 m) 6. Anchor rover and place instrument	 95%; 2 pixels 1 pixel 95%; 3 pixels 1.5 pixels 90%; 2 pixels 1 pixel	 90%; 3 pixels 1 pixel 90%; 4 pixels 1.5 pixels 90%; 2.5 pixels 1 pixel	 85%; 4 pixels 1 pixel 85%; 5 pixels 1.5 pixels 85%; 3 pixels 1 pixel
Overall single-sol target approach and instrument placement (tracking percentage, pixel error, and placement error)	81%; 3.0 pixels 1σ = 2.0 cm 3σ = 6.1 cm	73%; 3.5 pixels 1σ = 2.4 cm 3σ =7.1 cm	61%; 4.0 pixels 1σ = 2.7 cm 3σ = 8.1 cm

Table 3. A hypothetical example of an error budget analysis of the overall target approach system for three different test conditions. The numbers are very rough guesses and could be quite different in actual experiments.

3. Test Plan and Experimental Setup

3.1 Test Plan

Day 1: Collect camera images using a linear motion stage on a sunny day

- Three different sunlight directions
  - Morning (9 am)
  - Early afternoon (2 pm)
  - Late afternoon (4 pm)
- Three different directions of linear motion (see Figure 5)
  - Straightforward (parallel to the camera axes) for all
  - Oblique (30 degrees to the camera axes) for morning only
  - Sideways (perpendicular to the camera axes) for morning only
- Four lenses mounted on the camera head
  - 16 mm, 17° × 13° [e.g.: linearMotionStage/forward-9am/16mm/*.pgm]
  - 8 mm, 33° × 25° [e.g.: linearMotionStage/oblique-9am/8mm/*.pgm]
  - 4 mm, 65° × 49° [e.g.: linearMotionStage/sideways-9am/4mm/*.pgm]
  - 2.3 mm, 113° × 86° [e.g.: linearMotionStage/forward-2pm/2.3mm/*.pgm]
- Collect hi-resolution camera images every 1 cm over 4 m for all above conditions
  - High resolution CCD images: 1024 pixels × 768 pixels
  - File examples: Hi-A-40.pgm to Hi-A-440.pgm (401 images)

Day 2: Collect camera images using a linear motion stage on a cloudy day

- Clouds moving across the sun, causing significant change in lighting condition
  - Shadow contrast changes as clouds move across the sun
  - Late afternoon (5 pm, cloudy)
- Straight forward direction only
- Four lenses mounted on the camera head
  - 16 mm, 17° × 13° [e.g.: linearMotionStage/forward-5pmCloudy/16mm/*.pgm]
  - 8 mm, 33° × 25° [e.g.: linearMotionStage/forward-5pmCloudy/8mm/*.pgm]
  - 4 mm, 65° × 49° [e.g.: linearMotionStage/forward-5pmCloudy/4mm/*.pgm]
  - 2.3 mm, 113° × 86° [e.g.: linearMotionStage/forward-5pmCloudy/2.3mm/*.pgm]
- Collect hi-resolution camera images every 1 cm over 4 m

Day 3: Collect camera images for different roll and yaw orientations on a sunny day

- One sunlight direction: Early afternoon (2 pm)
- For roll (see Figure 6):
  - Collect images for every 1 degree roll of the tripod from –45 to +45 degrees
  - File example: linearMotionStage/roll-11am/16mm/*[0-90].pgm
- For yaw (see Figure 6):
  - Collect images for every 2 cm on a linear motion stage placed sideways over 4 m
  - Re-orient the camera head to point to a target located 5 m straight in front
  - File example: linearMotionStage/yaw-3pm/8mm/*[0-200].pgm

Day 4: Collect camera images using the mast on Rocky8 on a sunny day

- One sunlight direction: Early afternoon (2 pm)
- Straight forward direction only
- Pancam and Navcam pairs mounted on the mast
  - 16 mm, 17° × 13° [e.g.: rocky8Live/forward-2pm/16mm/*.pgm]
  - 4 mm, 65° × 49° [e.g.: rocky8Live/forward-2pm/4mm/*.pgm]
- Collect hi-resolution camera images every 1 cm over 10 m

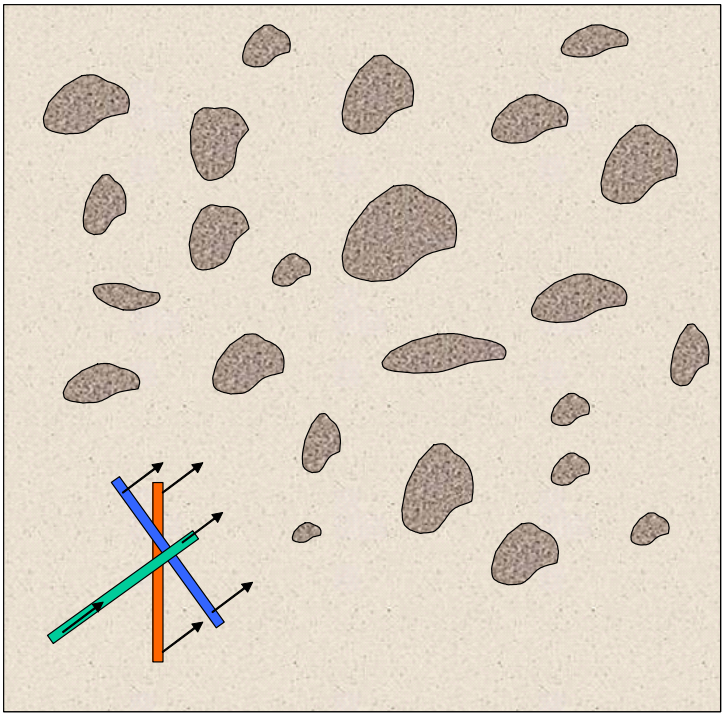


Figure 5. Translational camera motion in straightforward (green), sideways (blue), and oblique (red) directions. The camera tripod platform can move up to 4 m along the wooden linear motion stage. Camera orientation (black arrows) is fixed. The Mars yard is approximately 20×20 m.

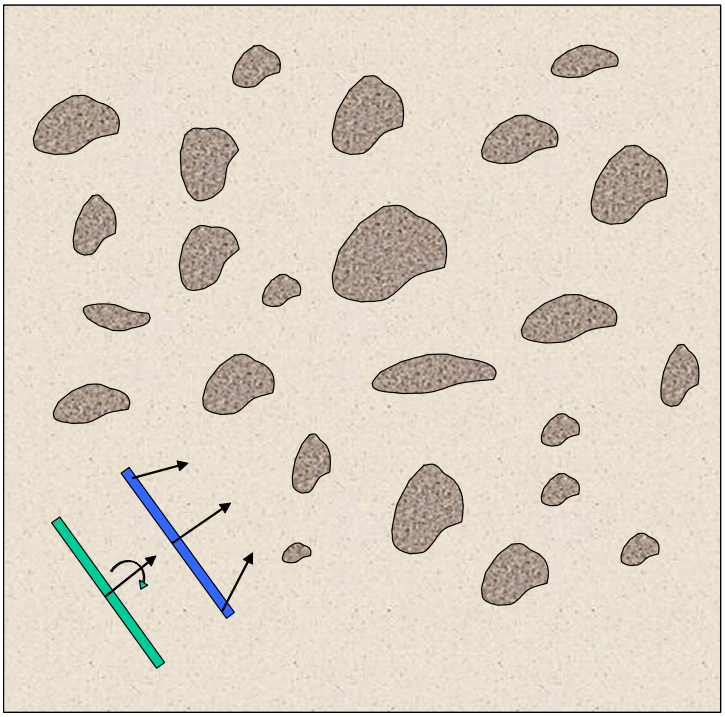


Figure 6. In camera roll motion (green), the camera head position is fixed while its roll angle is incrementally varied. In camera yaw motion (blue), the camera head is translated and at the same time its yaw angle is incrementally varied so that the camera head sees the same target location.

3.2 Experimental control variables

- Window size
  - 7, 15, 29, 59, 75, 119
- Number of pyramid levels
  - 1, 2, 3, 4, 5
- Template window update interval
  - 1, 10, 20, 100
- Transformation
  - 2-D translation, scale, z_rotation, affine transform
- Image skip
  - 0, 1, 2, 3, 5, 10, 20
- Camera motion
  - Straightforward, oblique at 30 degrees, and sideways
  - Roll
  - Yaw
- Sunlight direction
  - The sun in front of, above, or behind the camera
- Lighting variation
  - Abrupt changes in sunlight as an opaque clouds pass across the sun
- Target selection to track
  - Different texture
  - Rock image sizes
  - Occluding boundaries

3.3 Experimental Setup

Four high-resolution firewire Dragonfly cameras manufactured by Point Grey Research were used for the experiments. The resolution was 1024 pixels × 768 pixels. Four kinds of lenses were used in the experiment: 2.3 mm, 4 mm, 8 mm, and 16 mm lenses for a 1/3-in CCD image format. Table 4 summarizes their field of view angles for the lenses used.

Manufacturer and Focal length	Horizontal FOV × Vertical FOV
Computar 2.3 mm	113° × 86°
Fujinon 4 mm	65° × 49°
Fujinon 8 mm	33° × 25°
Fujinon 16 mm	17° × 13°

Table 4. Lenses used for the 2-D target tracking experiments

The images were collected using a laptop computer with an OrangeLink firewire card-bus PC card. The Point Gray's Dragonfly image capture software had been modified to support the consecutive acquisition of camera images with new filenames. A firewire hub was used to allow up to 5 cameras to be connected to the computer simultaneously.

We designed a wooden linear motion stage (Figure 7), which was necessary to move the camera head by a small increment over a long distance, avoiding large image displacements between consecutive images. As an example, we collected camera images every 1 cm over 4 m using this linear motion stage. The use of the linear motion stage was essential due to the lack of active camera control of the 2-D tracker.

Figure 7. A wooden linear motion stage was used in the Mars Yard to move the camera head linearly by small increments and collect image sequences. The camera head was mounted on the tripod.

4. Affine Tracker Software

The Visual Tracking Task led by JPL provided the affine 2-D visual target tracker within the CLARAty software environment for test and validation. A document "CLARAty Delivery for 2-D Visual Tracking" [Bajracharya, et. al., 2003] describes basic software functionalities of the affine 2-D tracker. Main components are feature matcher, template update, and image display.

4.1 Functional Description

A functional diagram of the initial delivery of the affine 2-D tracker is illustrated in Figure 8. When the user designates a desired target position on a rover camera image received, the affine tracker creates a template image window centered at the designated target position with a user-specified window size. As the rover moves a small distance and takes a new camera image, the pyramidal feature matcher in Figure 8 determines the new target image location in the new camera image by the Newton-Raphson style iterative method to minimize the difference between the template image and the new target image. The image motion parameters including the target position are updated for the next image matching. The template image needs to be updated occasionally to improve tracking reliability. Although threshold-triggered template update is desirable in practical applications, at present the only option available in the current version of the affine tracker software is to specify the template update interval *N* which updates the template image every *N*-th image. This template update interval parameter is thus used in our experiments.

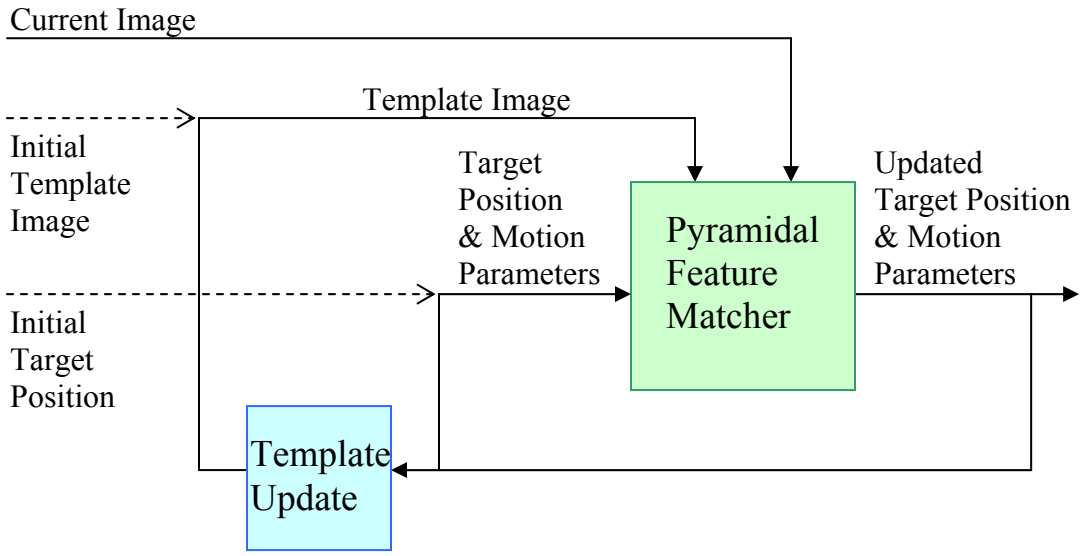


Figure 8. Functional diagram of the initial delivery of the affine 2-D tracker

Figure 9 illustrated how pyramidal images are used for feature matching. First, pyramidal images are obtained by reducing the camera image repeatedly. At pyramid level 0, the image is at full resolution or 1024×768 pixels in our tests. At level 1, the image is a half-size (in both dimensions) or 512×384 pixels. At level 2, the image is a quarter-size or 256×192 pixels. At level 3, the image is a one-eighth-size or 128×96 pixels. The pyramidal feature matching starts with low-resolution coarse matching at the highest pyramid level first, goes down to lower pyramid levels, and finishes with full-resolution fine matching at pyramid level 0. The pyramidal feature matcher uses the same target window size for all pyramid levels regardless of the image reduction. Increasing the pyramid level by one reduces the image size by half, doubling the effective window size. In general, if the window size = W×W pixels and the number of pyramid

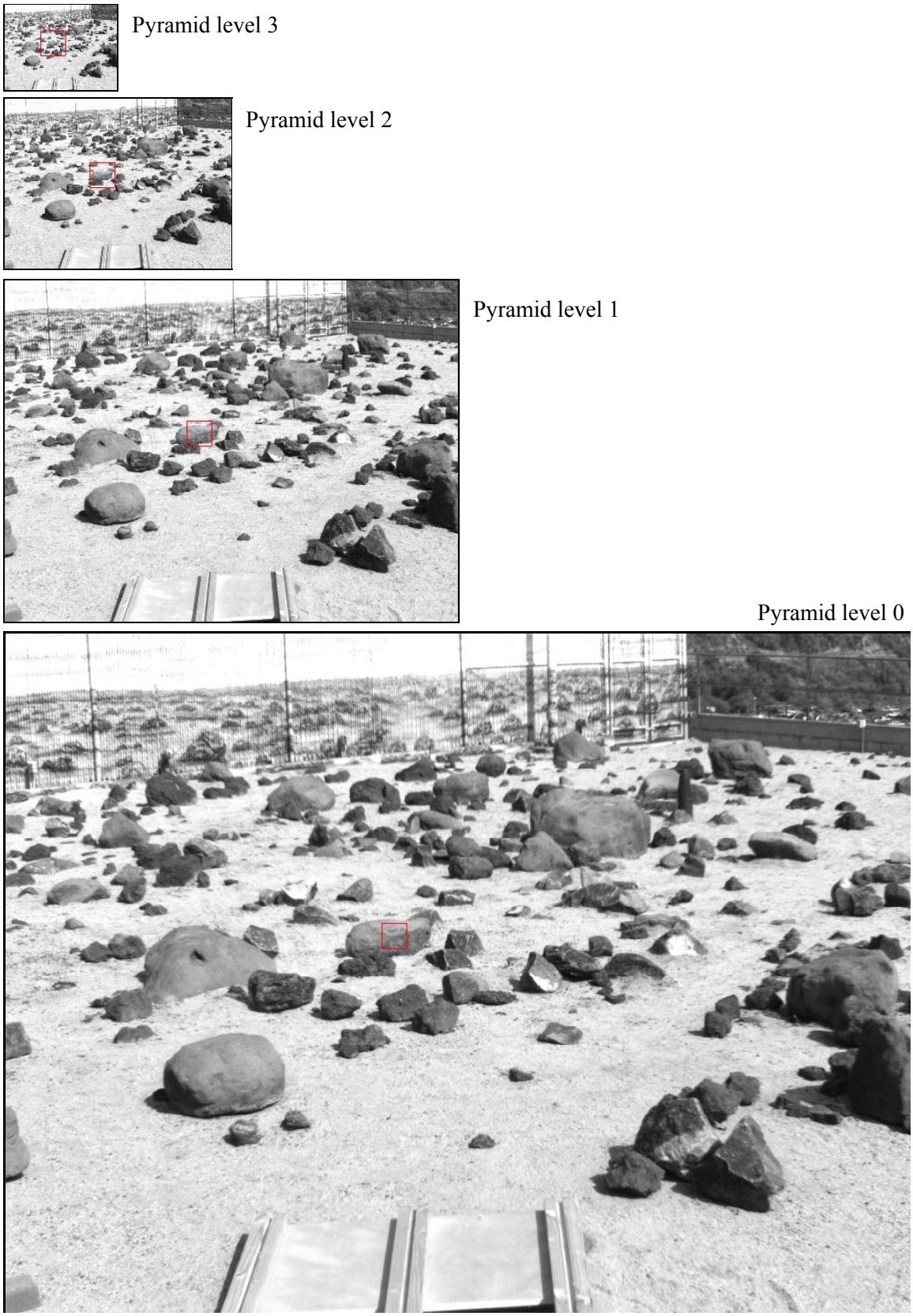


Figure 9. Pyramidal feature matching starts with the highest (level 3) and finishes with level 0.

levels = L, the effective window size = $(W \cdot 2^{L-1}) \times (W \cdot 2^{L-1})$ pixels at the highest pyramid level of L−1. In Figure 9, the target window size is 29×29 pixels. The effective window size is 232×232 pixels at pyramid level 3, 116×116 pixels at level 2, 58×58 pixels at level 1, and 29×29 pixels at level 0. Note that the target window at pyramid level 0 covers only a small portion of the big rock image, while the same 29×29 target window at pyramid level 3 covers several rocks including the big rock. The window size is always adjusted to odd numbers so that the window is centered at the center of a pixel not the edge.

The affine visual target tracker supports four options of transforms to match the template image of a feature to the current image: pure translation, z-rotation, scale, and affine transforms. As illustrated in Table 5, the pure translation uses 2 parameters, tx and ty, that shifts the feature in x and y coordinates to find the matching location of the feature in the new image. The scale transform uses a scale parameter in addition to 2 translation parameters. The scale parameter resizes the feature template image window for matching. The z-rotation transform uses a z-rotation or roll parameter in addition to 2 translation parameters. The z-rotation parameter rotates the feature template image window in roll orientation for matching. The affine transform uses 6 parameters: 4 deformation parameters in addition to 2 translation parameters. The four deformation parameters are the elements of a 2×2 affine deformation matrix A that allows scaling, rotation, stretch, and shearing of the 2-D planar template image window, assuming the feature can be approximated to be a planar surface.

Image Motion Transform	Parameters	Image Motion Transform Model
Pure Translation	2 parameters: t_x t_y	$\begin{bmatrix} x_j \\ y_j \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$
Translation and Scale	3 parameters: t_x t_y s	$\begin{bmatrix} x_j \\ y_j \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$
Translation and In-Plane Rotation	3 parameters: t_x t_y ω_z	$\begin{bmatrix} x_j \\ y_j \end{bmatrix} = \begin{bmatrix} 1 & -\omega_z \\ \omega_z & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$
Affine Transform	6 parameters: t_x t_y a_{11} a_{12} a_{21} a_{22}	$\begin{bmatrix} x_j \\ y_j \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

Table 5. Four image motion models are supported by the affine 2-D tracker for feature matching between images *I* and *J*. The image motion model describes the geometric relation between the target image point (x_i y_i) in image *I* and the matching target image point (x_j y_j) in image *J*.

The 2-D tracker that we test does not support active camera control. Namely, the camera is not actively re-oriented to eliminate the target image motion between consecutive images. Since no active camera control is involved in our experiments, we can run the 2-D tracker to track multiple targets for experimental efficiency. Instead of running the 2-D tracker 10 times with a single target each time, we run the 2-D tracker one time with 10 targets. It should be noted, however, that the user designates a single target and the tracker tracks the designated single target with active camera control in the actual single-sol target approach and instrument placement scenario.

4.2 Reconfiguration of the Affine Tracker

Our experimental comparison of pure translation and affine matching, which is described in Section 4.2 in detail, indicated that pure translation matching has best reliability but is less accurate, while affine matching has best accuracy but is more susceptible to lose tracking. This experimental finding prompted us to re-configure the initial delivery of the affine 2-D tracker to explore a sequential combination of pure translation and affine matching. As shown in Figure 10, we first use the pure translation matching to determine an approximate feature match. This (tx1, ty1) position is then fed into the affine matching to get a more accurate match if possible. The ssd (sum of squared differences) error is a good indicator of whether the affine matcher was successful. Therefore, the affine position result is used only when its ssd error is less than the translation ssd error. Otherwise the translation position is used. As described in Section 4.3, this combined tracking configuration resulted in best tracking performance. Therefore, we decided to use this combined tracking configuration throughout our test and validation experiments reported here. The combined tracking configuration was used for all experiments presented in this report except for those described in Sections 4.2 and 4.3.

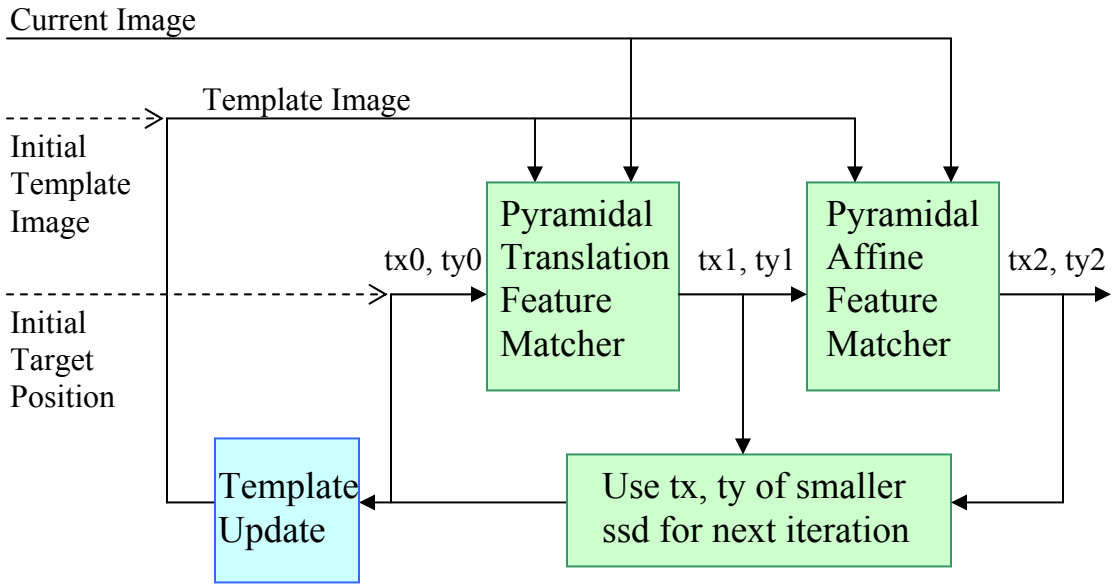


Figure 10. A combined tracking configuration: pure translation matching followed by affine matching is used as the "affine" tracker in our experiments.

4.3 Tracking Performance Metrics

Two tracking performance metrics used in this report are tracking percentage and drift error. The tracking percentage is calculated as the percentage of the number of tracked targets over the total number of initially selected targets. In this report, the tracking percentage calculation is from the first image to the latest image. A normalized tracking percentage such as tracking percentage per meter or degree could be considered, but we have not conducted statistical experiments to show an exponential decaying of 2-D tracking percentage over tracking distance or orientation change. The tracking percentage normalized per iteration or image-to-image does not make much sense, because we are interested in overall tracking performance. Further, the relation between the number of images used and the tracking percentage is not that simple.

To examine the tracking drift error, three different methods are used in this report. In the first method, the final target position of the 2-D tracker output is compared with the actual human measurements of the target position from the last image. In the second method, initial target positions are selected in a grid array pattern. Then, final target positions are visually examined by observing the relative positions in the grid. In the third method, affine transformed template windows (Figure 11) are superimposed.

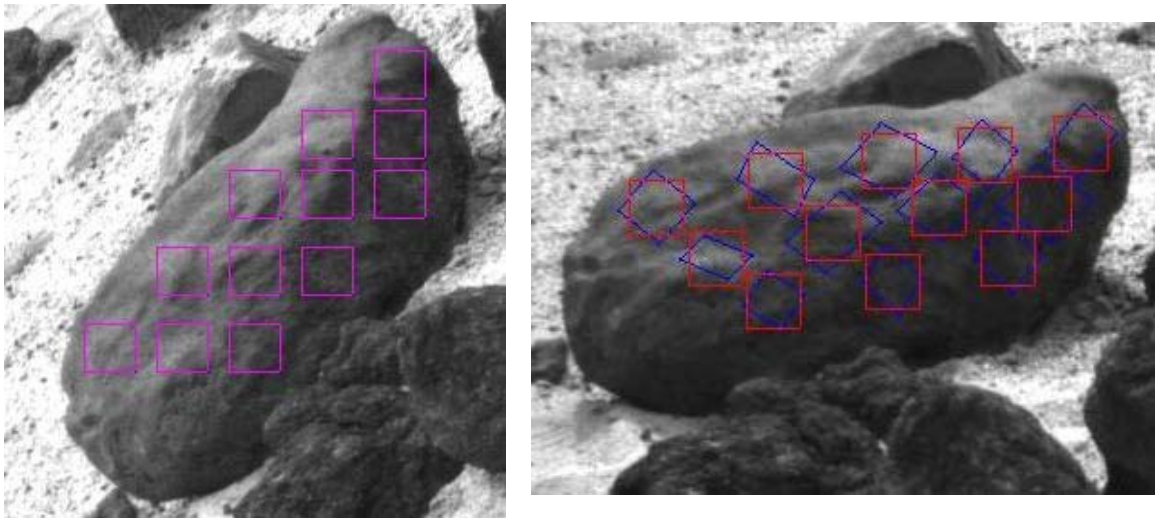


Figure 11. Close-up views of beginning (left) and end (right) images after 40º roll (top right) using 29×29 window with 3 pyramid levels: affine transformed template windows (blue) in the end image depict the matching quadrilaterals corresponding to the square template windows of initial targets (red) in the beginning image. All affine transformed quadrilaterals are rotated about 40º relative to the initial template square windows, illustrating good affine tracking performance. Badly deformed quadrilaterals imply poor tracking.

4.4 Adding Supporting Capabilities

For our test and validation of the affine 2-D tracker, we added the following several "helper" routines, which are briefly described here.

1. **Input file features_to_add.txt**
   The user can specify initial x and y positions of all targets in the input file named features_to_add.txt. The 2-D tracker software reads the file and recognizes them as initial target positions to track.

2. **Output file posout.txt**
   The 2-D tracker generates the output file named posout.txt, which prints out the target ID, x and y positions for each new image used for tracking.

3. **A combined 2-D translation and affine transform**
   This enables feature matching by a sequential combination of 2-D translation followed by affine transform.

4. **Affine transformed template window**
   We implemented an excellent way to observe how well affine tracking works by displaying the affine transformed windows (quadrilaterals) which are affine transformations of initial template windows (squares). The affine transform does translation, scale, stretch, and shear of the initial template image of a square window in order to match with the current target image. This capability enabled us to visually examine affine tracking behavior.

5. **Optional printouts of feature match parameters**
   The 2-D tracker can optionally print out 2-D translation parameters after each feature match. It can also print out transform parameters for each pyramid level to examine the computational behavior of iterative match in more detail.

6. **Optional drawing of thick lines for tracking window**
   The tracking window can optionally be drawn with three-pixel-wide lines on camera images instead of single-pixel-wide lines. Thick lines are needed to allow image reduction for use in reports, since thin lines are hard to see with image reduction.

7. **Mouse-click user interface for target selection**
   Target positions can also be specified interactively by using a mouse in addition to the text-based entry using features_to_add.txt described above. The user can create and delete targets directly on the displayed image.

Details of the mouse-click interface are described here. The 2-D tracker software is supposed to be used for a single-target tracking. However, for the test and validation purpose, the initial delivery of the affine 2-D tracker software provided the multiple-targets tracking interface, where the computer automatically selected targets to track based on corner detector scores. Since we needed to designate targets anywhere we wanted for the test and validation of the affine 2-D tracker, we added a user interface that allows the user to enter the target positions by mouse click. The following was the mouse-click user interface design specification we came up with. Upon running the affine 2-D tracking program, the initial image will be displayed for the user to select or specify targets via the user interface.

The two modes of the user interface are:

- Point-click mode [User-specified target creation/selection mode] and
- Region selection mode [Computer-generated target creation/selection mode].

The steps for the user-specified target creation/selection mode are:

- Double-click the left mouse button outside of any existing targets to specify the target position. One click will request the user to click one more time. If the two click positions are farther apart than the 2-D track window size, the user interface goes to the region selection or computer-generated target creation/selection mode described below.
- The goodness value for this target computed by the corner detector will be displayed.
- To reject a target position, click the target with the right mouse button.
- To accept this target, press the left mouse button.
- To select an existing target, click the target with the left mouse button. The goodness values of the target will be displayed. To accept the target, click the left mouse button again to confirm it. To reject the target, click the right mouse button.
- To reject an existing target without selecting it, click the target with the right mouse button.
- To exit this target designation user interface, click the right mouse button outside any exiting targets. A list of goodness values for all targets will be displayed. To confirm the exit, click the right mouse button again. To cancel the exit, click the left mouse button.

The steps for the computer-generated target creation/selection mode are:

- Click the two diagonal endpoints of a region with the left mouse button to indicate the region. If the region is smaller than the 2-D track window size, the user interface goes to the point-click or user-specified target creation/selection mode described above.
- The affine 2-D tracker software will find targets in this region, and these computer-generated targets will be displayed to the user.
- To select a target, click the target with a left mouse button. The goodness values of the target will be displayed. To accept the target position, click the left mouse button again to confirm it. To reject a target position, click the target with the right mouse button.
- To reject a target position without selecting it, click the target with the right mouse button.

To exit the region selection mode and go back to the user-clicked target creation/selection mode, click the mouse outside the region.

Although we implemented the above user interface, we ended up mostly not using it because computer-selected targets were not that useful at present, generating too many undesirable targets. This is because the corner detector of the affine 2-D tracker computes the goodness value per pixel, not per window, by considering only a very small local area independent of the window size. Instead, we implemented a much simpler text-based target entry using features_to_add.txt, which turned out to be very convenient in our experiments because it remembers the initial target positions without requiring the repeated entry of the same initial target positions. In our test and validation experiments, the text-based entry was much easier to enter exactly same initial target positions than the mouse-click entry.

4.5 Software Bug Findings and Fixes

As shown in Figure 12, tracking performance with the initial delivery of the affine 2-D tracker software was poor (left-side two plots). The final delivery software after first three bug fixes (right-side two plots) showed dramatic performance improvements.

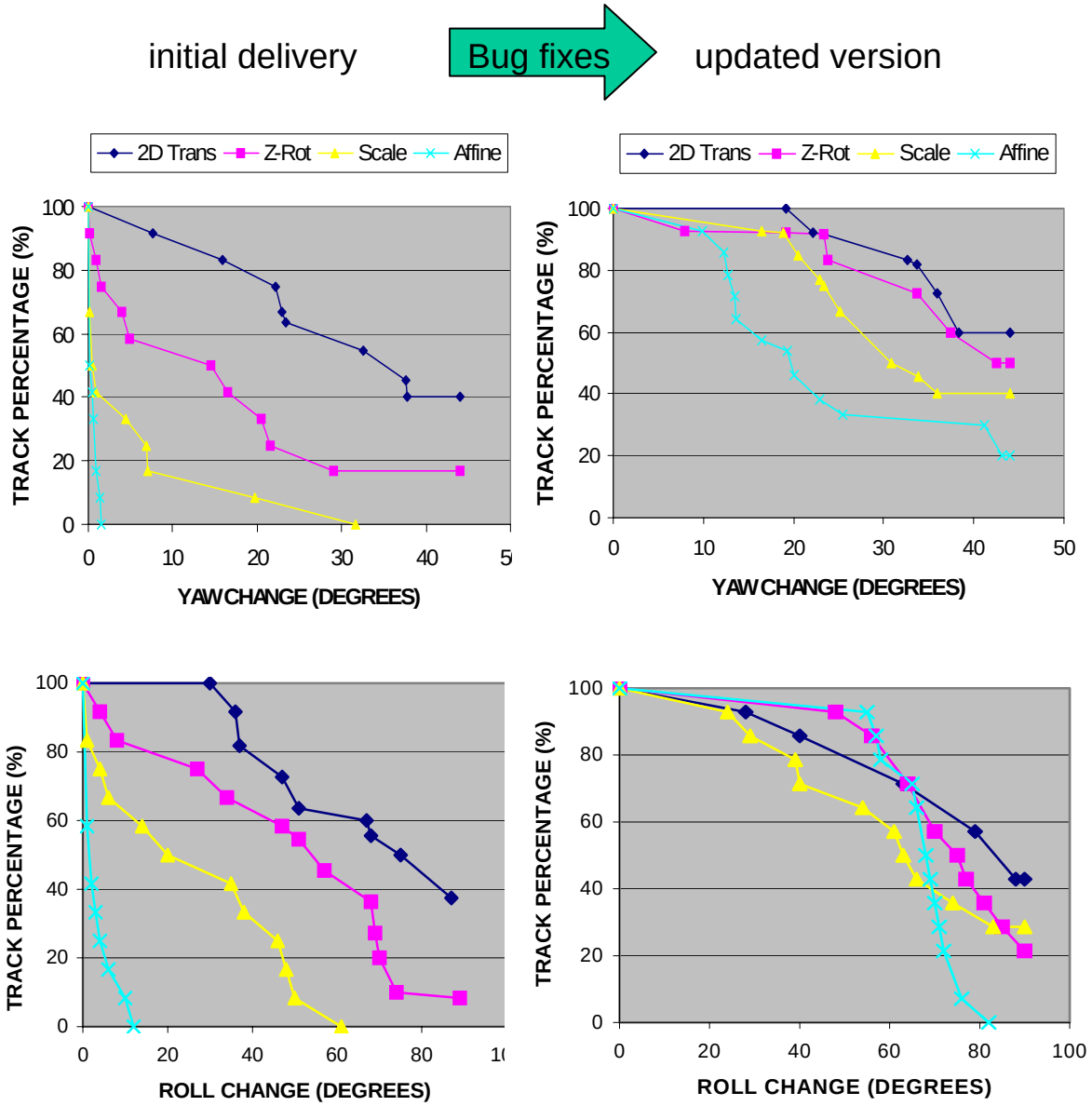


Figure 12. Tracking performance with different transforms for yaw (top row) and roll (bottom row) camera motions: before (left column) and after bug fixes (right column). The final delivery of the affine 2-D tracker after bug fixes showed impressive performance improvements.

In collaboration with technology providers, we identified the following five software bugs.

1. **Partial derivatives inverse matrices at different pyramid levels.** Partial derivatives inverse matrices were not retrieved correctly at lower pyramid levels. This messed up feature matching at lower pyramid levels.
2. **Delta translation updates between pyramid levels.** The delta translation updates were not halved when the pyramid level for matching went down, and not doubled when it went up. This also messed up feature matching at lower pyramid levels.
3. **Interchange of width and height**. In two places, image width and height were swapped. This prevented tracking beyond the 768-pixel column of the 1024-column image.
4. **Crash after long runs with a large number of targets.** There are memory leaks in Feature_Window and Image_Pyramid classes. Objects created with new() during class constructions could not be deleted upon class destructions. This bug is not fixed yet.
5. **Crash for track windows at the image boundary**. The program crashes when target windows are at the image boundary. This bug is not fixed yet.

Among these five bugs identified, the first three were fixed in the latest delivery of the 2-D tracker software. The last two bugs do not directly affect the tracking performance. However, sometimes we could not track several targets in a single run when the number of targets, window size, and/or the number of pyramid levels is large. In this case, we had to divide the targets and execute 2-D tracker in several runs.

5. Affine Tracker Experimental Procedure

Here is the procedure on how to download, compile, and run the run_feature_tracker, which is the affine 2-D tracker main program.

To check out the CLARAty stereo vision package from the CLARAty repository,

```
tcsh                    // csh causes an error
klog user_name          // enable access of afs files
start_claraty           // set environmental variables
yam setup -nolink -nobuild -d dir_name
```

When YAM.config comes up, edit it as shown below, or just copy and paste
The content of WORK_MODULES, LINK_MODULES, and BRANCH* from
/home/marstech/IPvalidation/2dtracking/claraty/*/YAM.config.

```
WORK_MODULES =    corner_detect_op \
                  feature_tracker \
                  image_displayer

LINK_MODULES =    SiteDefs/SiteDefs-R1-10l \
                  arrays/arrays-R1-08b \
                  data_io/data_io-R1-00d \
                  draw_ops/draw_ops-R1-02 \
                  image/image-R1-04c \
                  image_io/image_io-R1-03b-Build01 \
                  image_io_pnm/image_io_pnm-R1-01 \
                  image_ops/image_ops-R1-04-Build02 \
                  image_pyramid/image_pyramid-R1-01 \
                  image_rgb/image_rgb-R1-01-Build02 \
                  matrices/matrices-R1-09b \
                  points/points-R1-08a \
                  share/share-R1-09a \
                  string_io/string_io-R1-02c

BRANCH_corner_detect_op   =   corner_detect_op-R1-01  steele
BRANCH_feature_tracker    =   feature_tracker-R1-01   steele
BRANCH_image_displayer    =   image_displayer-R1-03   steele
```

To compile:

```
cd <sandbox>    // <sandbox> is dir_name specified in the "yam setup" command above
gmake all
cd src/feature_tracker
ymk all         // compile feature_tracker source codes to create run_feature_tracker
```

A couple of source codes, feature_tracker_app.cc and feature_matcher_lk.h, can be compiled with different compiler options by either commenting or un-commenting define statements.

In feature_tracker_app.cc:
#define NewTwoClickInterface This option allows mouse click entry of target

	positions.
#define noCombinedConfiguration	If not defined, a combined 2-D translation and affine transform is used in matching
#define AFFINE_POINTS	Computes affine transformed template windows and superimposes the computed quadrilaterals on the image
#define noAffineTemplateUpdate	Affine template is not updated while 2-D translation template is updated
#define DRAW_THICK_LINES	Draw three-pixel-wide thick lines for tracking window; thick lines are needed to see well after image reduction for reports.

In feature_matcher_lk.h:

#define REJECT_LARGE_DELTA	This option allows rejecting target positions if the iterative algorithm generates unexpected large displacements for next target positions
#define PRINT_DELTA_T	Prints the translation parameters tx and ty after the completion of feature matches over all pyramid levels
#define PRINT_V	Prints the transform parameter vector v for each pyramid level

To run run_feature_tracker using Drun:

~/<sandbox>/bin/Drun run_feature_tracker params.txt

Drun allows the user to use correct shared libraries by modifying LD_LIBRARY_PATH to point at <sandbox>/lib/$YAM_TARGET and PATH to point at <sandbox>/bin/$YAM_TARGET.

To run run_feature_tracker without using Drun:

First set LD_LIBRARY_PATH to point at <sandbox>/lib/$YAM_TARGET.

run_feature_tracker params.txt

The input file params.txt specifies the input parameters for tracking. An example file is shown here.

params.txt:

```
{
      feature_window_size = 29;     // 29×29 pixels window
      feature_distance_threshold = 29;
      use_fixed_detect_threshold = 1;
      detector_threshold = 4000;
      detector_percentage_of_max = 75;
      detector_window_size = 3;
      detector_derivative_size = 7;
      use_fixed_match_threshold = 1;
      matcher_max_num_iters = 20;
      matcher_ssd_diff_threshold = 0.1;
      num_pyramid_levels = 2;       // matching in level 1 then 0
      base_filename = "/home/marstech/IPvalidation/
            2dtracking/linearMotionStage/
```

```
            roll-11am-2003-08-21/16mm/Hi-C-%03d.pgm";
        window_update_rate = 100;     // update template interval
        max_num_of_images = 100;
        min_num_of_features = 1;
        debug_sleep = 0.1;
        image_num_begin = 0;
        image_num_end = 91;
        image_num_skip = 0;           // no image skip
    }
```

In addition to params.txt, run_feature_tracker checks to see if the following input file exists.

features_to_add.txt: This file specifies x and y image coordinates of each target position per line. This input file has the same effect of the user entering all the listed target points with a mouse.

After the run, run_feature_tracker produces the following outputs.

posout.txt: each line prints out the image number followed by all features containing feature number, feature position x, and feature position y

6. Affine Tracker Performances

6.1 Transforms

As described in Section 4.1 and Table 5, the affine 2-D visual target tracker supports four options of transforms to match the template image of a feature to the current image: pure translation, z-rotation, scale, and affine transform. These four transforms were applied to three sets of the image sequences collected earlier with 3 different camera motions of 4-m straightforward, 90º roll, and 45º yaw. Only high-texture targets with salient corner features were selected. No template updates were used in these tests. Namely, the feature windows selected at the beginning images were used as template images all the way to the end images.

For the forward camera motion (Figure 13), the tracking percentage was calculated as the ratio of the number of tracked targets to the total 14 initial targets during the course of tracking. After 2.15-m forward motion, pure translation lost one target, reducing its tracking success rate to 93% (13 out of 14 targets). After the full 4-m run, pure translation tracked 9 out of 14 initial targets resulting in 64% tracking percentage. Tracking percentages for other transforms were similarly calculated and plotted in Figure 13. The plot shows that pure translation performed best, while affine transform did worst.

For the yaw camera motion (Figure 14), 4 targets moved out of camera view during the course of tracking. The successful tracking percentage was calculated as the ratio of the number of tracked targets to the total number of targets remaining in the camera view. After the full 22° yaw motion, pure translation lost one target and another target was out of view, resulting in 92% (12 out of 13 in-view targets). After full 44° run, pure translation tracked 6 targets out of 10 remaining targets in camera view, resulting in 60% tracking percentage. Tracking percentages for other transforms were similarly calculated and plotted at the top of Figure 14. Just like the forward motion, pure translation performed best, while affine transform did worst.

The tacking percentages of four transforms for the roll camera motion are shown in Figure 15. Unlike the forward and yaw camera motions described above, z-rotation and affine transforms performed best, while scale transform did worst. Camera roll motion had the same effect of simply rotating the 2-D images, which is essentially 2-D planar transformation regardless of the 3-D shapes of objects in the scene. Ideally, this allows perfect matches for both z-rotation and affine transforms. For this reason, z-rotation and affine transforms performed better than pure translation for the roll motion. Unlike the roll camera motion, forward and yaw camera motions generally involve 3-D non-planar transformation, and in these conditions pure translation resulted in more reliable tracking. Affine transform is more accurate when it tracks, but is more sensitive to discrepancies and easier to lose targets. This is because affine transform has 6 degrees of freedom compared to 2 degrees of freedom for pure translation.

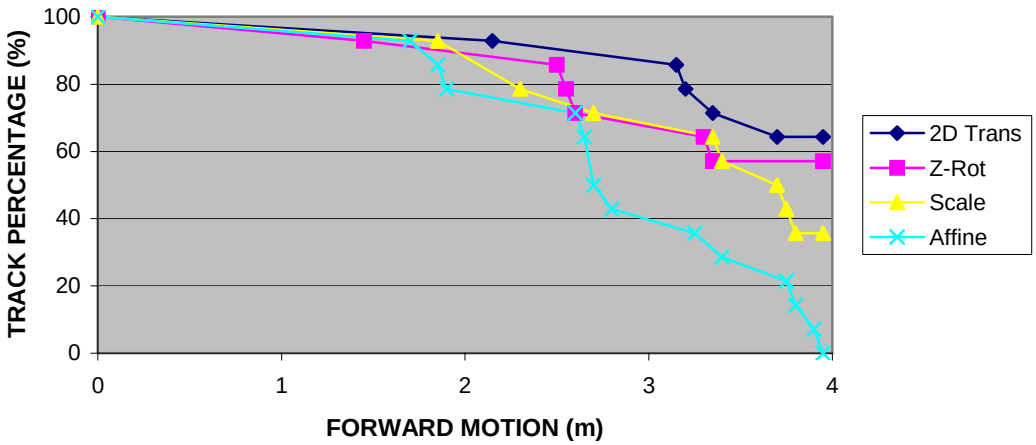


Figure 13. Tracking performance of different transforms over 4-m forward motion with no template update: pure translation performed best, while affine transform performed worst.

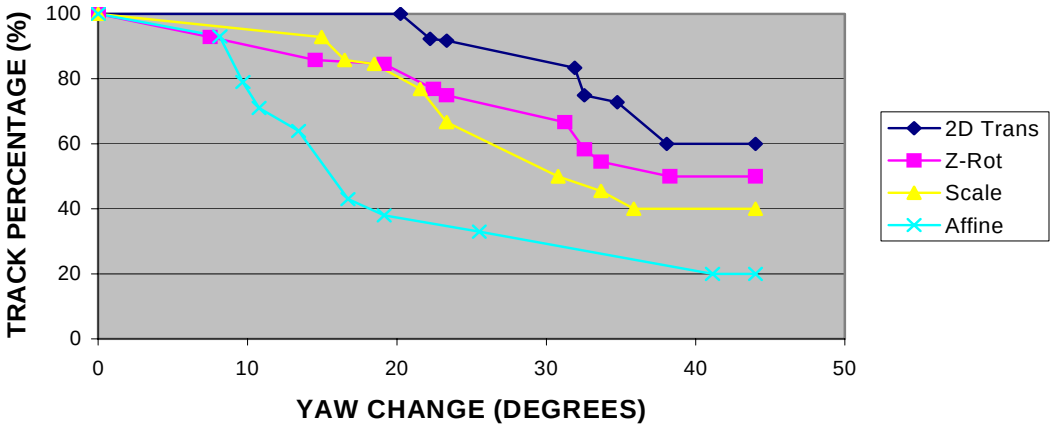


Figure 14. Tracking performance of different transforms over 45° yaw motion with no template update: pure translation performed best, while affine transform performed worst.

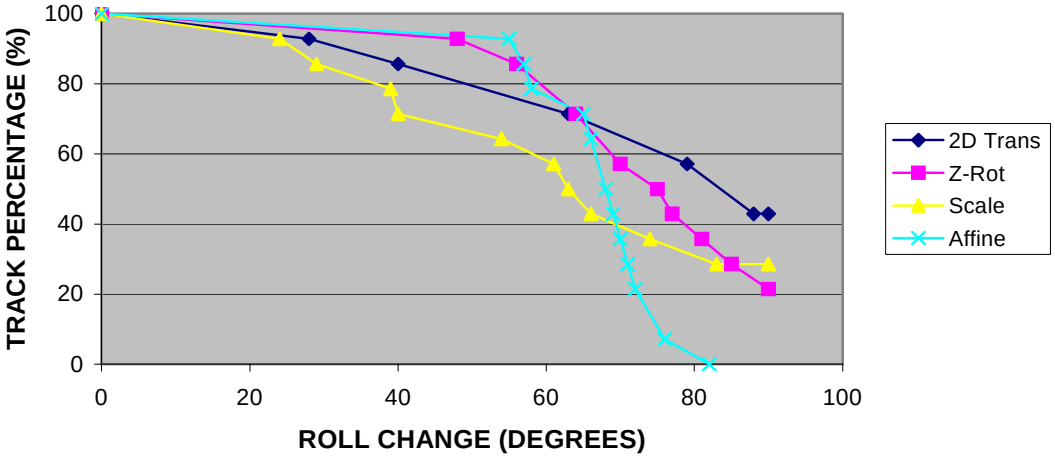


Figure 15. Tracking performance of different transforms over 90° roll camera motion with no template update: z-rotation and affine performed best, while scale transform performed worst.

Figure 16 shows the rms (root-mean-square) intensity matching errors of pure translation and affine transform during the course of 2-D tracking of a target over the 3-m forward camera motion. The rms intensity error is the square root of the per-pixel normalized ssd (sum of squared differences) between the template window image and the matching current image, where 255 is the maximum intensity error. When the template image was not updated (Figure 16 top), the rms intensity error increased with the image number for both pure translation and affine transform. The affine transform yielded slightly lower rms intensity error when it tracked. However, it lost tracking at image number 350, while pure translation never lost the target. When the template image was updated (Figure 16 bottom) for every 100 images, the rms intensity error dropped sharply at each template update. With the template update, affine transform as well as pure translation did not lose the target. During this tracking, the affine transform yielded slightly lower rms intensity error than pure translation.

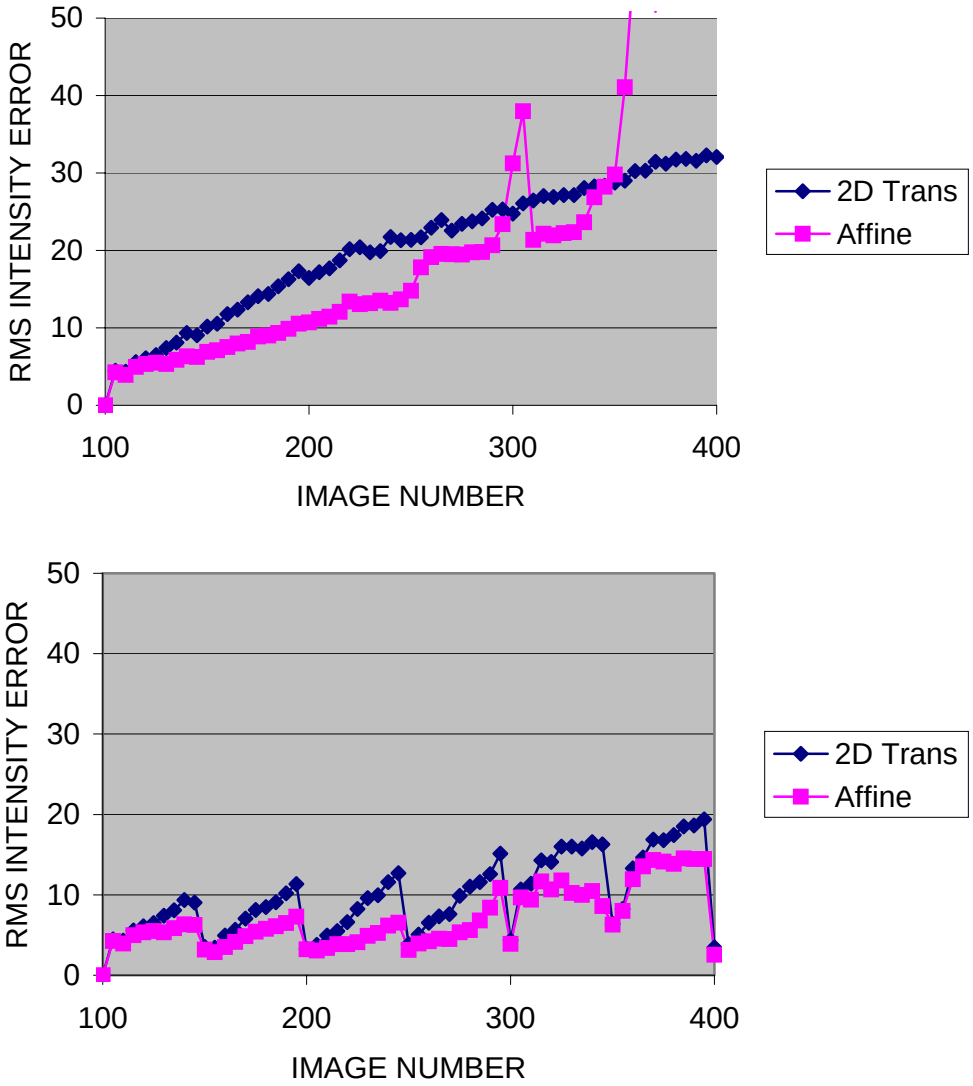


Figure 16. The rms intensity errors of 2-D translation and affine transform matching over 3-m straightforward camera motion from image number 100 to 400. No template update (top) and template update of every 100 images or 1 m (bottom).

Similarly, Figure 17 shows the rms intensity matching errors of pure translation and affine transform during the course of 2-D tracking of a target over the 90º-roll camera motion. When the template image was not updated (Figure 17 top), the affine transform yielded significantly lower rms intensity error when it tracked. This is because pure translation does not handle roll motion, while affine transform does. However, the affine transform lost tracking at image number 70, while pure translation never lost the target. *This example clearly illustrates that affine transform is more accurate when it tracks, but less reliable.* When the template image was updated (Figure 17 bottom) for every 20 images or 20º roll change, the rms error dropped sharply at each template update. With the template update, both pure translation and affine transform maintained low rms intensity error and did not lose the target.

Our experiments suggested that affine transform performed best with smaller tracking errors when it tracked, but was not as reliable as pure translation, losing targets more easily. More comparative experiments are presented in the next Section.

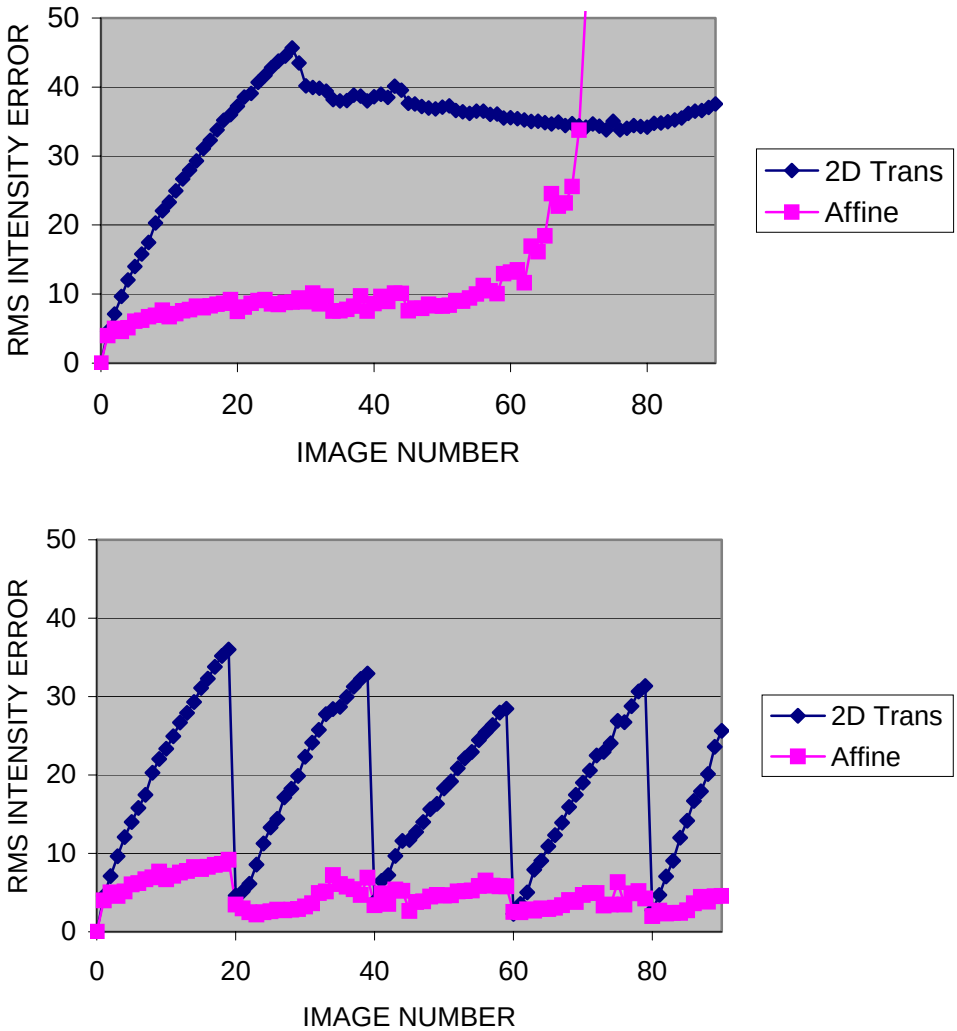


Figure 17. The rms intensity errors of 2-D translation and affine transform over 90° roll camera motion from image number 0 to 90; no template update (top) and template update of every 20 images or 20° (bottom).

6.2 Tracking Configuration

Experimental results in section 4.2 indicate that 2-D translation has the best reliability while affine transform has the best accuracy. In order to maximize the tracking performance, we decided to investigate a combined configuration of the 2-D translation matching and affine matching as shown in Figure 10 of Section 4.2. In order to compare the three methods of pure translation, affine transform, and the combined configuration, we performed 2-D tracking experiments on an image sequence (no image skip) of straightforward camera motion with an 8-mm lens. The template update interval was set to 10 or every 10-th image. The tracking results are shown in Table 6. Affine transform alone was less reliable: 49% tracking for 15×15 window with 3 pyramid levels and 89% for 29×29 window with 2 pyramid levels. By contrast, the pure translation alone and the combined configuration were more reliable: 98% to 100%.

	15×15 window with 3 pyramid levels	29×29 window with 2 pyramid levels
Translation alone	44/45 (98%)	44/45 (98%)
Affine transform alone	22/45 (49%)	40/45 (89%)
Combined configuration	44/45 (98%)	45/45 (100%)

Table 6. Tracking success rates of pure translation, affine, and combined configuration for straightforward camera motion with 8-mm lens: pure translation and combined configuration were more reliable than affine matching alone.

Tracking behaviors of pure translation, affine, and combined configuration for two targets are compared in Figure 18. For clearer comparison, no template image was updated, and two targets were chosen that were not tracked all the way by the affine tracker. The rms intensity error (normalized ssd per pixel) of the affine matching started lower, but eventually the affine matching lost the target. On the other hand, pure translation tracking had higher rms intensity error, but did not lose the target. Combined tracking basically chooses the better of the two matching results. This method effectively combines the strengths of both transforms. Initially, the rms error of the affine transform stays low providing an accurate position estimate, and when the affine tracker loses the target, the pure translation matching maintains the approximate position of the target to bring the affine tracker back on target.

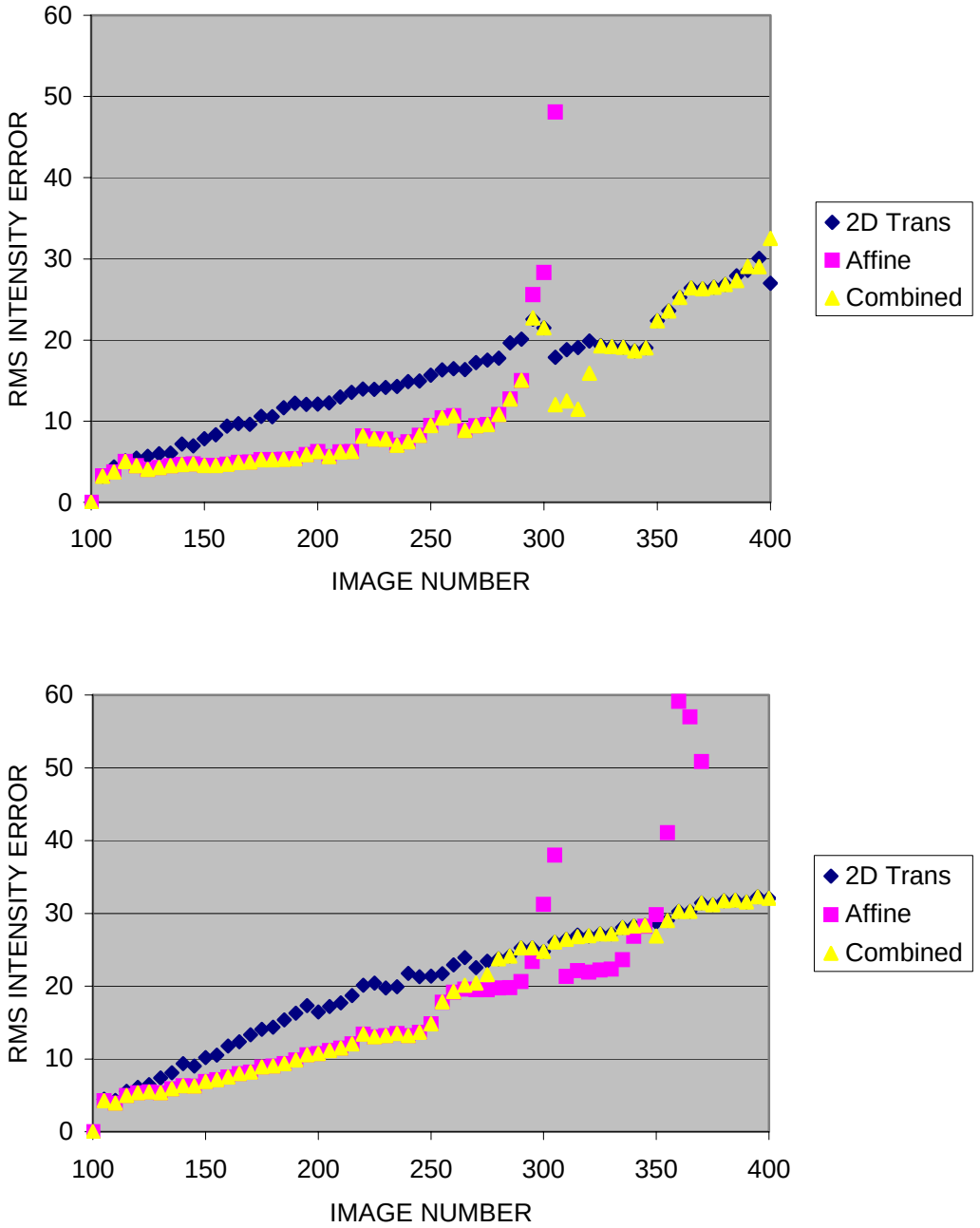


Figure 18. Matching intensity errors as a function of the image number for pure translation, affine, and combined configuration for two different targets (top and bottom) without template update: the combined configuration chooses the better of the translation and affine matching, combining the strengths of both methods.

Figures 19, 20, and 21 compare tracking results of three different transform methods. When pure translation matching without template update is used, target location shifted from the left corner to the right corner of the bigger rock at the end of the track (Figure 19). When the template is updated every 20º, the target location still shifted (Figure 20), but not as much as no template update of Figure 19. When the combined 2-D translation and affine transform was used, the target tracked very well (Figure 21) with lowest drift errors.

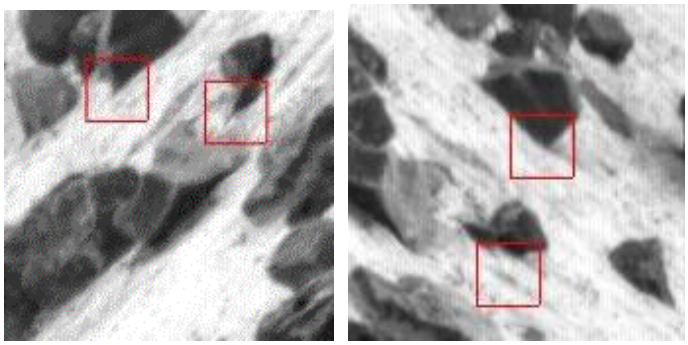


Figure 19. Pure translation matching of two targets without template update over 90º roll motion yielded the tracking error of 30 and 39 pixels, respectively.

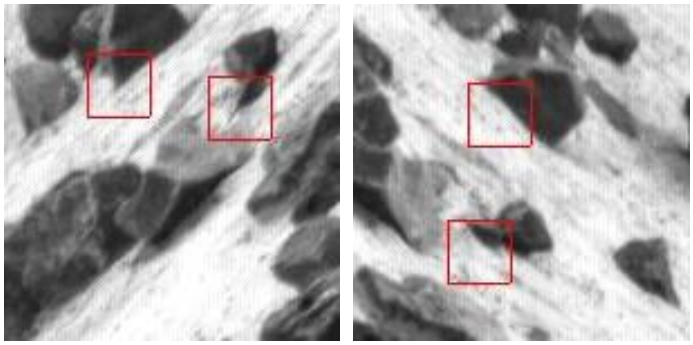


Figure 20. Pure translation matching of two targets with template update of every 20º roll yielded the tracking error of 13 and 24 pixels, respectively.

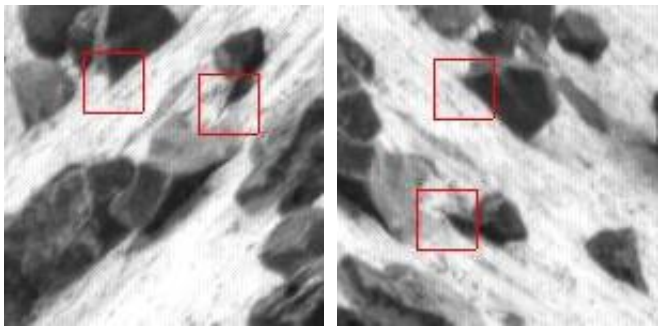


Figure 21. Combined configuration of pure translation with affine correction with template update every 20º yielded the tracking error of 1.7 and 1.0 pixels, respectively.

6.3 Window Size and Pyramid Level

We investigated the effect of window size and pyramid level on 2-D tracking performance. Figure 22 shows the beginning and end images of the image sequence collected for a 4-m straightforward traverse with a 16 mm lens. The image sequence consisted of total 400 images with a 1-cm straightforward camera motion between consecutive images. We selected 11 targets in the beginning image based on corner detector values. These targets were high-texture targets, and all the targets tracked 100% over a wide range of the window sizes and pyramid levels. In this initial test, no images were skipped, resulting in a 1-cm camera motion between images. In order to compare various window sizes and pyramid levels, we increased camera motion by skipping images until tracking performance was adequately degraded. We chose 19-cm camera motion between images by running the 2-D tracker on every 19-th image to perform multiple runs to compare various window sizes and pyramid levels. Every 19-th image (image skip of 18) was chosen simply because it hits the last or 400-th image (19 is a multiple of 399). Table 7 shows the results of those runs. Each entry shows the number of targets successfully tracked out of 11 total targets selected. It also shows the drift error in pixels of successfully tracked targets compared to accurate human measurements as ground-truth values. Accurate measurements of the final target positions were obtained from the last image by carefully comparing with the initial template target images.

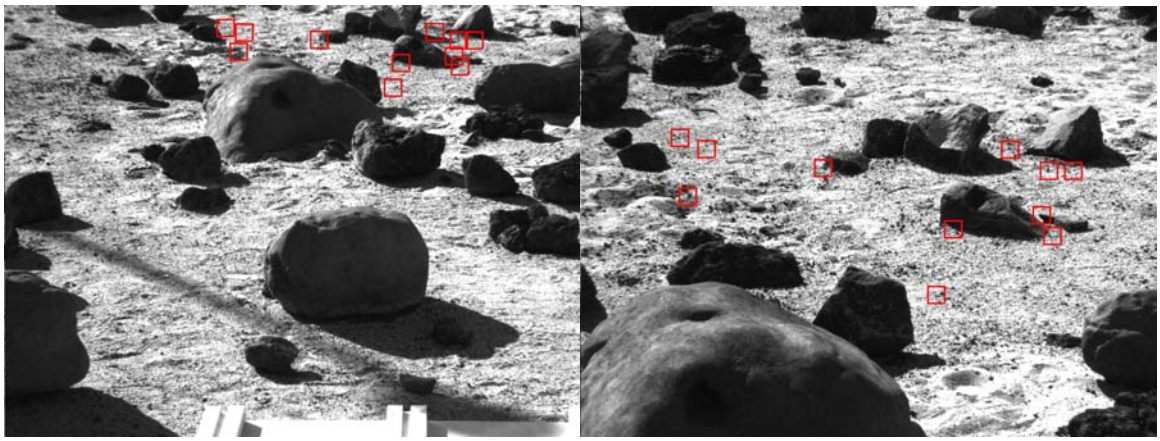


Figure 22. Beginning and end images for a 4-meter straightforward traverse with 16 mm lens, where the 2-D tracker corner detector selected high-texture targets.

Pyramid levels	Window size			
	7x7	15x15	29x29	59x59
1	0% (0/11)	18% (2/11) 11.6 pixels error	27% (3/11) 14.4 pixels error	36% (4/11) 15.1 pixels error
2	27% (3/11) 11.7 pixels error	45% (5/11) 9.6 pixels error	64% (7/11) 2.6 pixels error	100% (5/5) 4.8 pixels error
3	55% (6/11) 12.3 pixels error	91% (10/11) 2.0 pixels error	100% (5/5) 3.2 pixels error	
4	100% (11/11) 2.1 pixels error	100% (5/5) 2.2 pixels error		

Table 7. 2-D tracking success rate and drift error results for relatively large 19-cm camera motion between images. Diagonal entries have similar tracking performance. The 7×7 window with 4 pyramid levels performed best for high-texture targets.

As described in Section 4.1 and Figure 9, if the window size = W×W pixels and the number of pyramid levels = L, the effective window size = $(W \cdot 2^{L-1}) \times (W \cdot 2^{L-1})$ pixels. Increasing the number of pyramid levels by one doubles the effective window size. Doubling the window size while reducing the number of pyramid levels by one keeps the effective window size the same. In Table 7, doubling the tracking window size (adjusted to be an odd number) showed a similar effect of reducing the number of pyramid levels by one in terms of tracking performance. Namely, each upward diagonal in Table 7 shows similar tracking performance. The approximate effective window size is 7×7 pixels for the gray diagonal, 15×15 for purple, 30×30 for pink, 60×60 for green, and 120×120 for yellow. The tracking performance gets better as the effective window size increases. The effective window size of 30×30 (pink) or below (purple and gray) was too small to handle large image displacements caused by a relatively large 19-cm camera motion between consecutive images. The effective window size is 60×60 (green) and 120×120 (yellow) yielded good tracking performance. The effective window size is a major factor that determines the maximum image displacement that can be tracked. In Table 7, all 11 targets had good features to track. In this case, a small 7×7 window with 4 pyramid levels performed best with low drift error. On the other hand, a large 59×59 window performed poorly. A large window size tends to include adjacent occluding or occluded objects that confuse tracking because their relative positions shift in the image during the tracking. Further, they violate to a greater degree the assumption of the affine tracker that the target is a planar surface. As long as a 7×7 window has sufficient texture to track, 7×7 window with 4 pyramid levels tracks more accurately than 59×59 with 1 level while the maximum image displacement to track is about the same.

Next, we investigated tracking on low-texture targets. Figure 23 shows the beginning and end images for a 3-m straightforward traverse with 8-mm camera lens. We selected 25 targets in a grid pattern on a large rock with low texture. We then performed similar runs as the previous experiment but with 5-cm and 10-cm motions between images, using the image skip of 4 and 9, respectively. Smaller camera motions were used this time because the low-texture targets were harder to track. Tables 8 and 9 show the results of these runs. Each entry shows the successful tracking percentage and the number of targets tracked out of 25 initial targets selected. We did not compute the drift error because we did not feel human measurements of the low-texture target positions in the final image were sufficiently accurate as ground truth values. Instead, a target was considered lost if it drifted too much from its relative position in the grid. In Tables 8 and 9, window size of 29×29 performed better unlike in Table 7 where 7×7 window performed better. The results indicate that targets with low texture need to have a larger window size. The 59×59 windows tend to include adjacent occluding or occluded objects whose image positions relative to the large rock keep changing during the course of tracking. Some of the targets near the boundaries of the large rock get confused and lost during tracking.

As in Table 7, the trend of improving performance with increasing pyramid level continues in Tables 8 and 9, but it exhibits saturation or plateau beyond which there is almost no performance improvement. Raising the number of pyramid levels helps to handle larger image displacements. If the number of pyramid levels is sufficiently large to handle the large image displacements occurring, the tracking performance will approach its plateau. The green-colored areas in Tables 8 and 9 show this saturation effect. As expected, the plateau for each window size in Table 9 is reached about 1 pyramid level lower than that in Table 8, since image displacements from 5-cm camera motion are roughly half of those from 10-cm motion. The best tracking success rate was, however, only 84% for 10-cm forward motion, much lower than 96% for 5-cm forward motion. The low saturation at 84% tracking success rate indicates that some of large image displacements of low-texture targets could not be overcome by simply increasing the larger number of pyramid levels. Some of these targets had adjacent occlusion and background interference problems.

Figure 23. Beginning and end images of 3-m straightforward traverse with 8 mm lens, where a human operator selected a grid of low-texture targets from a large rock.

Pyramid levels	Window size			
	7x7	15x15	29x29	59x59
1	0% (0/25)	16% (4/25)	44% (11/25)	52% (13/25)
2	8% (2/25)	48% (12/25)	68% (17/25)	68% (17/25)
3	8% (2/25)	64% (16/25)	84% (21/25)	68% (17/25)
4	12% (3/25)	72% (18/25)	84% (21/25)	

Table 8. Tracking results for 10-cm camera motion between images (image skip of 9). Green area shows the saturation of improvement in tracking performance with higher pyramid levels.

Pyramid levels	Window size			
	7x7	15x15	29x29	59x59
1	4% (1/25)	40% (10/25)	68% (17/25)	76% (19/25)
2	16% (4/25)	72% (18/25)	88% (22/25)	76% (19/25)
3	16% (4/25)	80% (20/25)	96% (24/25)	76% (19/25)
4	24% (6/25)	80% (20/25)	96% (24/25)	

Table 9. Tracking results for 5-cm camera motion between images (image skip of 4). Green area shows the saturation of improvement in tracking performance with higher pyramid levels.

The tracking results with high-texture and low-texture targets of Tables 7, 8, and 9 indicate two important effects of window size.

- For small-image-size rocks, a small window of 7×7 or 15×15 pixels would be fine with enough texture to track well. Larger window sizes tend to increase the drift error due to non-planarity of the target.
- For medium to large-image-size rocks, the window size needs to be appropriately increased to cope with less texture. At the same time, the window size should not be too large to keep the target reasonably planar and not to include adjacent occluding or occluded objects. A 29×29 window size appears to be a good compromise.

Therefore, we recommend varying the window size depending on the planarity and texture of the target. This will require that the command generation interface allow the specification of window size at the time a feature is selected.

6.4 Template Window Update

Previous experiments in Figures 13, 14, and 15 provide rough upper bounds for the template window update: 2 m interval for 4-m straightforward motion starting from about 10 m away from the target position (20% change in distance), and 20º change for roll and yaw motions. Beyond these upper bounds, the tracker starts losing targets due to a large discrepancy between the template image and the current image. So as we reduce the template update interval, the tracking reliability increases. However, if we update the template image too often with a smaller template update interval, the tracking position error increases again due to the template drift problem.

Although threshold-triggered template update is desirable in practical applications as described in Section 2.1, at present the only option available in the current version of the 2-D tracker software is to specify the template update interval *N* to periodically update the template image window every *N*-th image. To examine the effect of the template update on 2-D tracking performance and find its optimal value, we ran 2-D tracking experiments by varying the template update interval for two different sets of initial targets (Figure 24). Figures 25 and 26 show the results from two such experiments for forward camera motion. The results indicate that different features may have somewhat different optimal values for template update interval. For instance, as illustrated in Figure 27, corner features are scale-invariant and thus require less frequent template update. An update interval of every 50 images (Figure 25) to every 100 images (Figure 26), which corresponds to 0.5 to 1 m distance change, appears to be a good value to use for straightforward motion where targets were about 10 m away from the 16-mm lens camera. Too frequent update of the template image window caused the template image to drift much and increased the rms (root-mean-square) tracking position error, while too infrequent update also increased the tracking position error due to much difference between the template and current target images. Since the percent change in target image size is roughly proportional to the percent change in target distance from the camera, the percent change in target distance appears to be a good threshold parameter for the threshold-triggered template update. In Figures 25 and 26, the average target distances were both roughly 10 m from the camera, and the optimal template update of every 0.5 to 1 m distance for the 10-m distant target corresponds to 5% to 10% change in target distance.

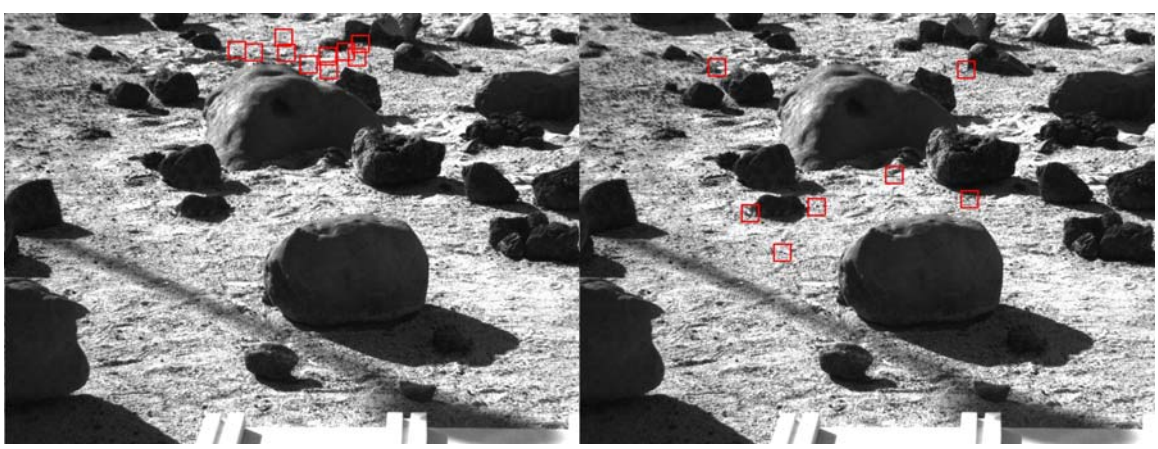


Figure 24. Two sets of initial target positions used in template update experiments for forward camera motion with 16 mm lens.

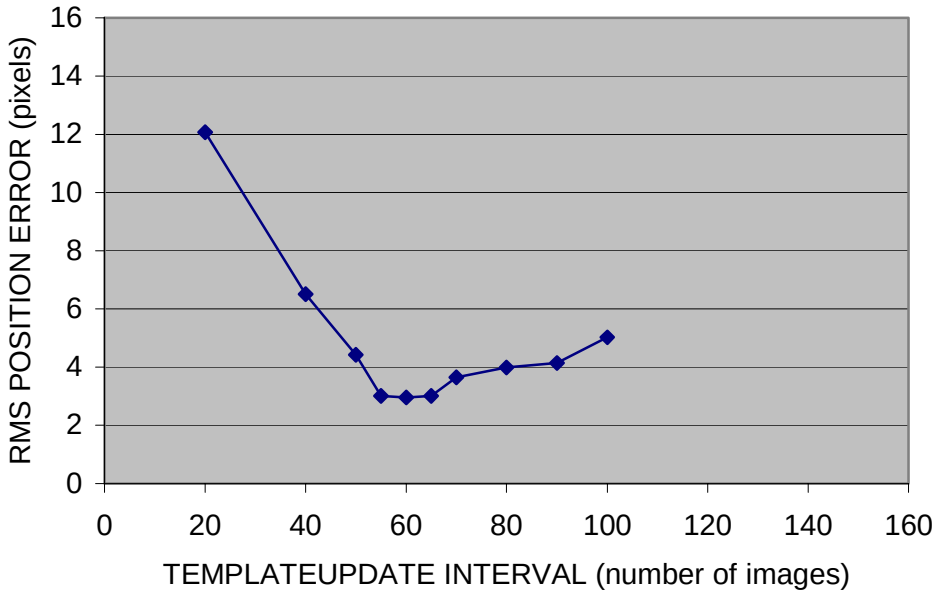


Figure 25. Tracking position error (in pixels) vs. template update interval (in number of images) from a set of 10 targets (Figure 24 left) for 4-m straightforward tracking: both too frequent (N > 100) and too infrequent (N < 50) template updates increased the tracking error. No targets were lost over all template update intervals tested.

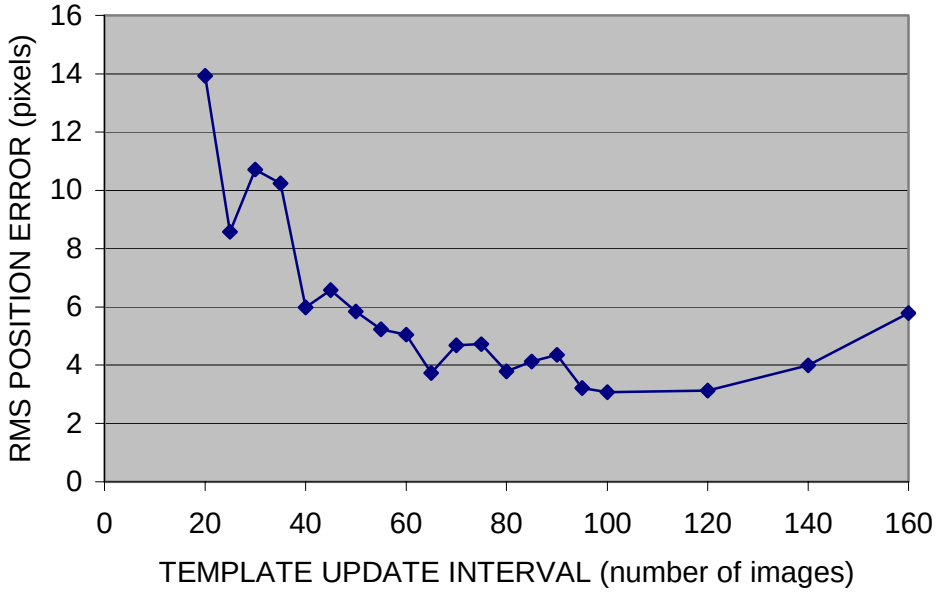


Figure 26. Tracking position error (in pixels) vs. template update interval (in number of images) for another set of 7 targets (Figure 24 right) : both too frequent (N > 150) and too infrequent (N < 50) template updates increased the tracking error. No targets were lost over all template update intervals tested.

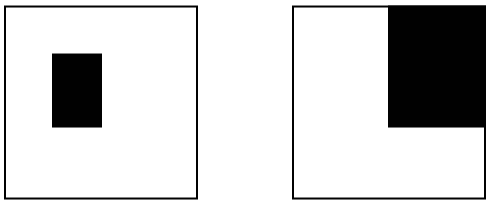


Figure 27. A black rectangle inside the window is scale-variant (left), while a rectangular corner is scale-invariant (right). Thus, corner-feature need less frequent template update for forward camera motion where the target image size changes.

Figures 28 and 29 show the effect of the template update interval on tracking error for the roll and yaw camera motions. For roll motion of Figure 28, a good template update interval is every 5 to 10 images or every 5º to 10º roll change. For yaw motion of Figure 29, a good template update interval is every 20 images or approximately every 5º yaw camera motion. A good threshold value to trigger the template update for roll and yaw camera motions appears to be around 5º to 10º change.

Although the current version of the 2-D tracker only allows the periodic template update of every n-th image or a fixed distance, it appears to be a better option to use the distance percent change to trigger template update, considering the image size change by perspective image formation. The orientation change is another parameter to monitor for template update. The ssd value or rms intensity error of the affine tracker is also a good parameter to monitor for template update. The affine tracker ssd would remain relatively flat until the tracker started to lose the target and then it would spike. The tracker would cache the last few images seen. When the affine tracker ssd crosses a threshold, the template for that target would be updated using the one with the lowest ssd value among the saved images. The template window would be updated by whichever occurs first: distance percent change, orientation change, or affine tracker ssd. Actual experiments of the above or similar ideas were not performed, because the 2-D tracker without active camera control would be extremely labor-intensive to collect data for full 10-m target approach experiments. We plan to perform more concrete experiments on template window update in the next 2-D/3-D tracking experiments with active camera control.

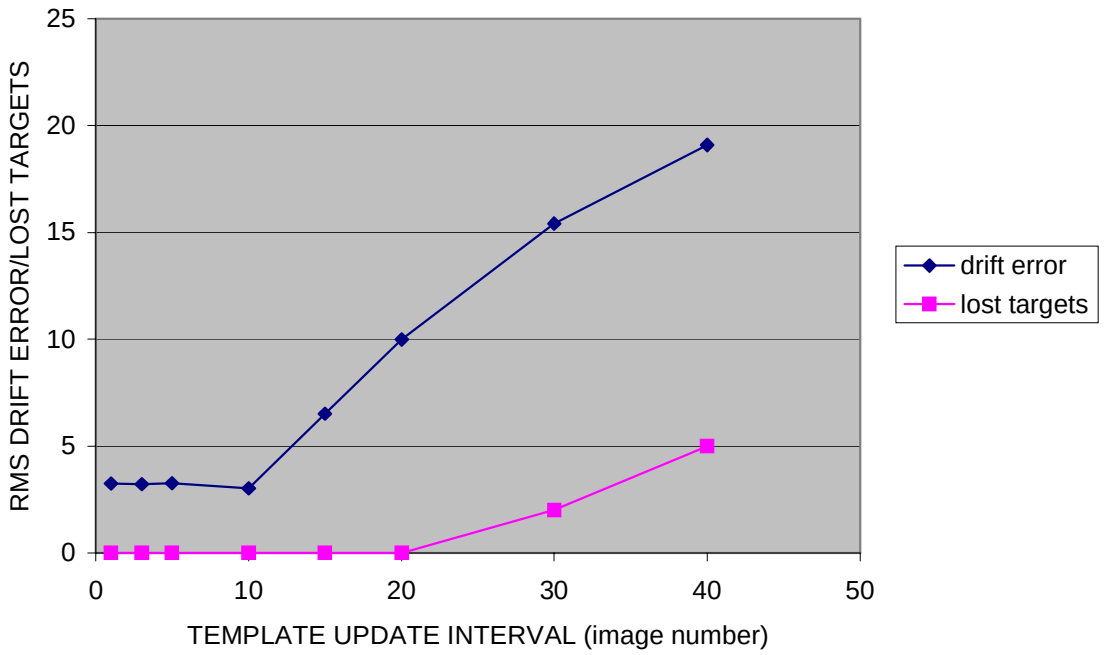


Figure 28. Tracking position error (in pixels) vs. template update interval (in number of images) for a set of 14 targets for roll camera motion: targets were lost for large template update intervals

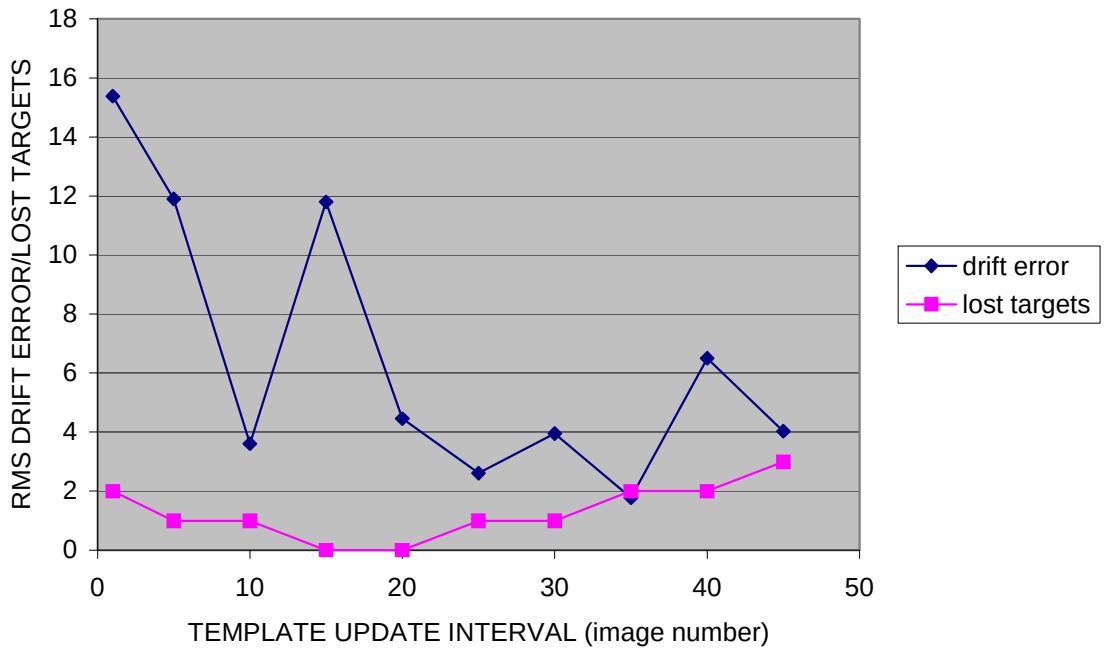


Figure 29. Tracking position error (in pixels) vs. template update interval (in number of images) for a set of 11 targets for yaw camera motion

6.5 Maximum Image Displacement for Active Camera Control

Another goal was to determine the accuracy requirements for active camera control. If the feature tracker can track large image displacements, then the active camera control subsystem does not need to be as precise. To measure the capability of the feature tracker for tracking large image displacements, we first ran the tracker on images taken by the camera moved forward every 1 cm with a set of targets. All targets tracked successfully. This provided a baseline of where each feature was located in each image which we used as an approximate ground truth. The initial 11 target positions selected and their final tracked positions after 4-m straightforward motion with the 16-mm lens are shown in Figure 30. We then increased image skip until targets were lost. By comparing the runs where the targets were lost to the baseline run where no targets were lost, we evaluated exactly when a target was lost and what image displacement occurred at that time. We also calculated the image displacement for all instances where targets successfully tracked. The tracking instances were binned by image displacement, and within each bin the fraction of instances that tracked successfully over the total number of both successful and unsuccessful instances was calculated. This provides a statistical estimate of the probability of tracking a particular image displacement. This data is plotted in Figures 31 and 32. The 15×15 window with 3 pyramid levels (Figure 31) tracked perfectly up to 12 pixels image displacement and quite well above 90% up to 30 pixels, while the 15×15 window with 2 pyramid levels (Figure 32) tracked perfectly up to 9 pixels and quite well up to 27 pixels. As a very rough approximation, if we assume the maximum trackable image displacement Δd_{max} is about 1/3 of the effective window size, we can compute it by

$$\Delta d_{max} \approx 1/3 * W * 2^{L-1},$$

where W is the window size and L is the number of pyramid levels. For window size 15, $\Delta d_{max} \approx$ 20 pixels with 3 pyramid levels, and $\Delta d_{max} \approx 10$ pixels with 2 pyramid levels. Vertical dashed lines in Figures 31 and 32 mark these values. It is, however, important to note that the tracking percentage plotted here is per frame not over the whole course of tracking. Even if the tracking percentage per frame is 90%, the overall tracking percentage after 10 frames is reduced to mere 34%. Thus, in the next experiment, we measured the overall tracking percentage for different image skips.

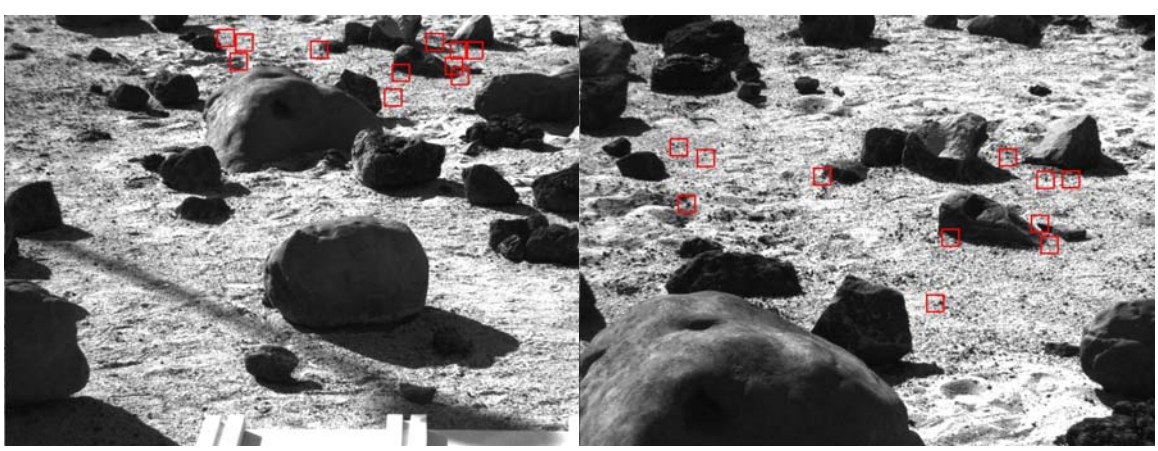


Figure 30. An example of beginning (left) and end (right) images for 4-m forward motion with 16-mm lens. Tracking data were collected with different image skips to investigate the effect of different image displacements on tracking performance.

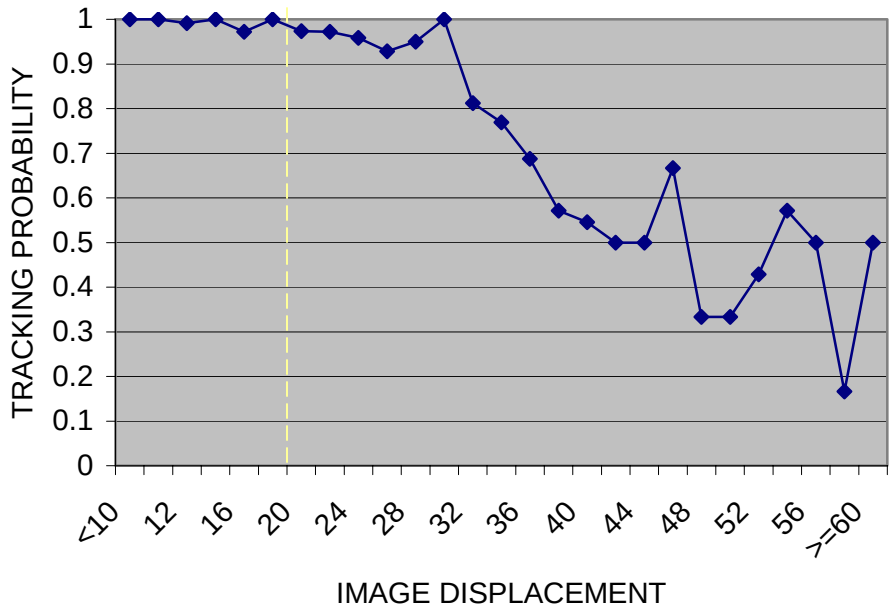


Figure 31. Tracking probability per frame vs. image displacement for the 15×15 window with 3 pyramid levels (effective window size of 60 pixels). The dashed yellow line of 20-pixel displacement represents the 2/3 overlap of the template and the matching window.

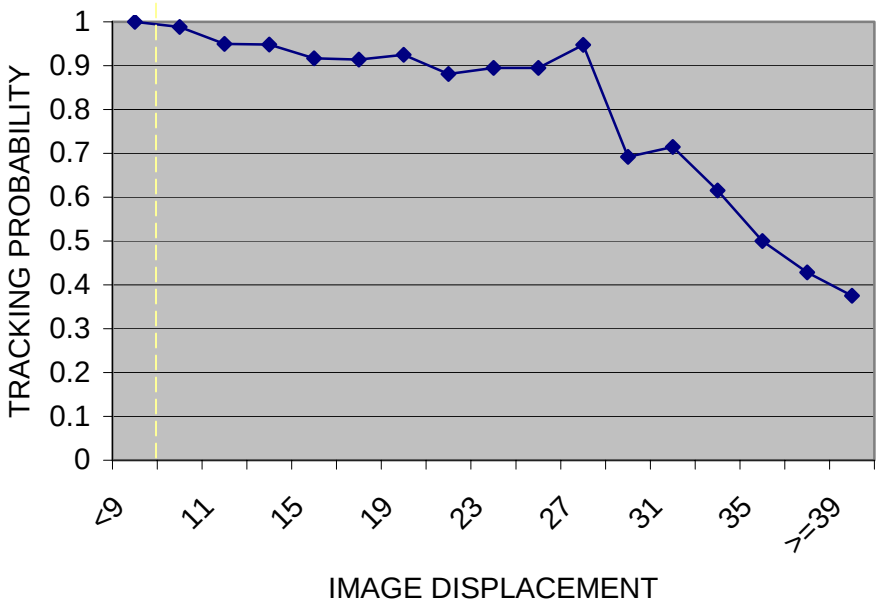
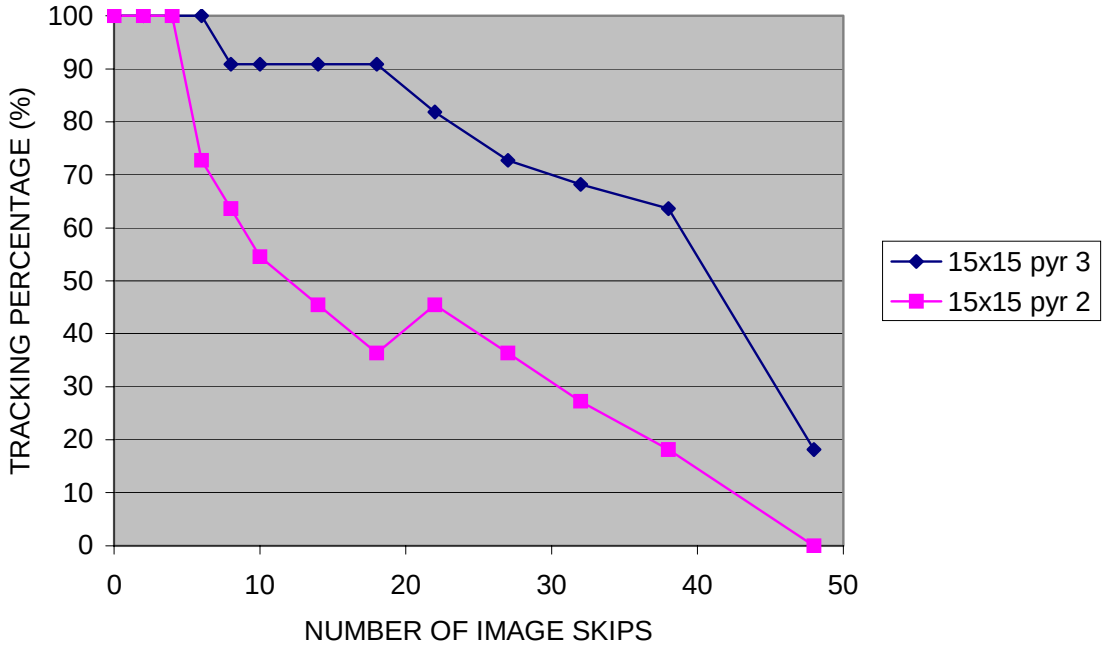


Figure 32. Tracking probability vs. image displacement for the 15×15 window with 2 pyramid levels (effective window size of 30 pixels). The dashed yellow line of 10-pixel displacement represents the 2/3 overlap of the template and the matching window.

Figure 33 shows the overall tracking percentage for different image skips. The maximum and average image displacements of the target images are also shown below the tracking percentage plot. For the window size 15 with 3 pyramid levels, the tracking percentage stayed above 90% for the image skip of up to 18. The maximum image displacement at the image skip of 18 was 35 pixels, and the average was 14 pixels. For the window size 15 with 2 pyramid levels, the tracking percentage stayed above 90% over the image skip of up to 4. The maximum image displacement at the image skip of 4 was 11 pixels, and the average was 4 pixels. This plot clearly demonstrates that higher number of pyramid levels can handle larger image displacements.



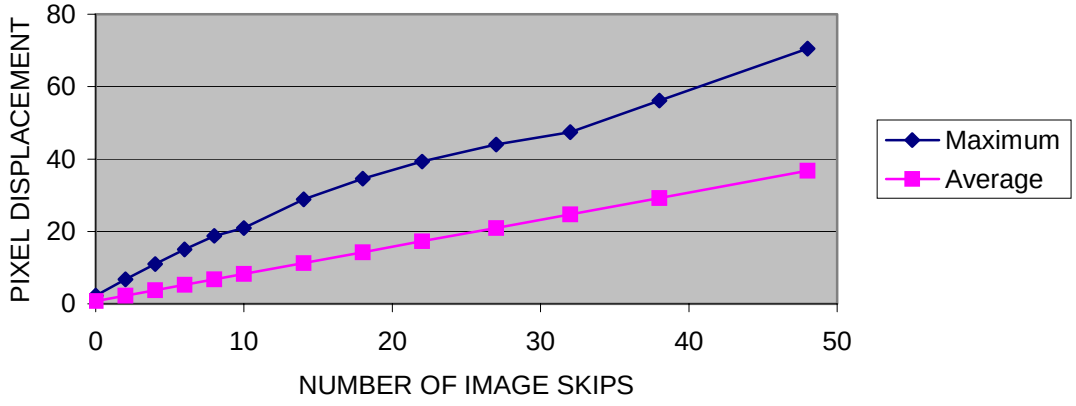


Figure 33. Overall tracking percentage with respect to the number of image skips for the window size of 15 with 2 (pink in top figure) and 3 (blue in top figure) pyramid levels. The maximum and average image displacements with respect to the number of image skips are shown right below the tracking percentage plot. The affine tracker handled larger image displacements when three pyramid levels compared to two pyramid levels.

6.6 Lighting changes

The effect of lighting was studied by performing 2-D tracking on the four sets of straightforward image sequences collected in four different time of the day: 9 AM, 2 PM, 4 PM, and 5 PM. For the first three sets, it was bright, sunny with no clouds. For the 5 pm set, it was still sunny with almost no clouds, but patches of dark opaque clouds happened to pass across the sun during the image collection. Each set has different sunlight direction, forming different shadows. 2-D tracking performances of the four sets using 15×15 window with 3 pyramid levels are summarized in Table 10. Images at the end of 3-m straightforward tracking were compared in Figure 34 for 9 am, 2 pm, and 4 pm (5-pm images were shown separately in Figure 35). Left three images are from 16 mm lens, while right three images are from 8 mm lens. Since some of the 2 pm images had data collection errors, only 3-m portions, instead of full 4-m runs, were used in this experiment.

The 2-pm images (middle pair) did not show much shadow over the rock surfaces, and resulted in 100% tracking. The 4- pm images had shadow in the left-hand side, behind rocks, but did not have much shadow on the rock surfaces facing the camera. They resulted in almost 100% tracking, where only one target for 8 mm drifted away about 10 pixels.

The 9-am images had dark shadow on the rock surfaces facing the camera. At first glance, it appeared that the tracking performance would be very poor because of dark shadow. It turned out that surface images of rocks still had enough texture to track reasonably well even with dark shadow, resulting in 80% and 100% tracking performances. Only two targets for the 8 mm lens lost tracking. Even though the lighting and shadow condition for 9 am was significantly poorer than those of 2 pm and 4 pm, the tracking performance was pretty good, demonstrating robustness of tracking relative to the sunlight direction change.

Image collection start time	F/L: 8mm	F/L: 4 mm
9 am; sun was in front, no cloud	100% (18 of 18 targets)	80% (8 of 10 targets)
2 pm; sun was above, no cloud	100% (18 of 18 targets)	100% (10 of 10 targets)
4 pm; sun was behind, no cloud	94% (17 of 18 targets)	100% (10 of 10 targets)
5 pm; a patch of cloud crossing sun	44% (8 of 18 targets)	0% (0 of 10 targets)

Table 10. Tracking performance of 3-m forward motion at three different sunlight directions (9 am, 2 pm, 4 pm) and at one condition with dramatic sunlight changes (5 pm) using 15×15 window with 3 pyramid levels.

Tracking performance at 5 pm (Table 10) with dramatic sunlight changes was very poor. For the 8 mm lens, 8 out of 18 initial targets were suddenly lost after 0.75 m tracking, and 2 more targets were lost later, resulting in 44% tracking over 3 m. For the 4 mm lens, 7 out of 10 initial targets were suddenly lost after 0.75 m tracking, and all targets were lost by 1.75 m, resulting in 0% tracking over 3 m.

Images in Figure 35 clearly explain why many targets were suddenly lost. The left three images were from 8 mm lens, and the right three were from 4 mm lens. Three distinct images during the course of the tracking were shown from top to bottom. Under bright sunlight, the shadows were stark (top pair in Figure 35). When a patch of opaque clouds partially covered the sun, the surface of the Mars Yard became darker, and shadows became dimmer (middle). When the sun was fully covered, shadows completely disappeared (bottom pair). In order to compensate for lighting

change, automatic gain control (AGC) was activated for both 8-mm and 4-mm images during our data collection. AGC tried to keep the average intensity over the entire image relatively constant regardless of ambient light changes. In the images on the left side of Figure 35, the 8-mm lens did not include the "bright" sky in view, and AGC was effective to maintain the average brightness of the Mars Yard at about the same level, even with the dramatic reduction in direct sunlight (bottom left image in Figure 35). Thus, AGC significantly helped tracking performance with the 8-mm lens during the dramatic sunlight changes. AGC, however, did not avoid shadow contrast changes: stark shadows (top left image in Figure 35) with bright sunlight and almost no shadows (bottom left image) when the sun was covered by opaque clouds. These shadow changes made 2-D tracking for the 8-mm lens less reliable, resulting in mere 44% tracking success rate.

By contrast, the 4-mm images included the "bright" sky in view, and the average intensity of the entire image stayed very high regardless of the lighting changes of the Mars Yard surface. Therefore, AGC did not compensate for sunlight changes effectively for the 4-mm lens. When the sun was covered by opaque clouds, the whole image area of the Mars Yard surface got much darker (bottom right image in Figure 35) unlike the 8-mm lens. Large changes in intensity of the image tracking area resulted in 0% tracking success rate for the 4-mm lens.

Incidentally, this dramatic lighting change due to dark clouds (opacity > 10) would not happen in Martian environment. Here is a direct email quote from Leslie Tamppari, who was Deputy Project Scientist for MSL, regarding Martian clouds and sun lighting [Tamppari, 2003].

"The clouds on Mars are very thin clouds - more like stratospheric clouds on earth. The typical visible opacity is only around 0.4 or so. They appear to form in haze layers most of the time, but there can be lee waves associated with topographic rises, but again the clouds are thin for the most part. Occasionally they are thicker, but usually associated with mountaintops and we won't be going there. Dust will also occur on Mars and could reach opacities higher than the clouds. In dust storm conditions, the opacities can get above 2. However, these are likely to be hazy as well.

Sunlight conditions will vary as a function of time of year and latitude (as well as time of day, of course). If we are at far N or S latitudes during winter, we could have the sun low in the horizon (5 degrees above for a few months)."

However, it should be noted that shadow images, for instance, behind occluding boundary of a rock can change as the rover and its cameras move. Thus, a tracking window that includes changing shadows (by adjacent rocks, mountain, clouds, or the rover) can in general cause 2-D tracking to fail.

8 mm 4 mm

Figure 34. Images at the end of 3-m forward tracking using 15×15 window with 3 pyramid levels for 8 mm lens (left) and 4 mm lens (right): at 9 AM (top pair), at 2 PM (middle pair), and at 4 PM (bottom pair)

8 mm 4 mm

Figure 35. Images at the end of 3-m forward tracking using 15×15 window with 3 pyramid levels for 8 mm lens (left column) and 4 mm lens (right column): with full sun (top pair) after 1 m forward motion, with sun partially covered (middle pair) after 1.25 m forward motion, and with sun fully covered by a patch of clouds (bottom pair) after 2.75 m forward motion.

6.7 Tracking with Oblique and Sideways Camera Motion

2-D tracking was performed for the image sequences collected with oblique and sideways camera motions as depicted Figure 5 in Section 3.1. The camera was moved on a liner motion stage, so that consecutive images did not show large image displacements. Since the camera orientation was not changed during the image collection, oblique and sideways camera motions tended to lose targets out of the camera view more quickly than forward camera motion. For this reason, oblique motion allowed 2-D tracking up to 1.6 meters with a 16 mm lens (Figure 36) and over the entire 4 meters with 4 mm lens (Figure 37). Sideways motion allowed tracking up to 1.2 meters with a 16mm lens (Figure 38) and up to 3.3 meters with an 4 mm lens (Figure 39). Oblique and sideways camera motions are similar to straightforward camera motions in terms of tracking, because they all cause translation and scale changes in images. Since oblique and sideways camera motions have the problem of targets going out of camera view quickly, we decided not to spend further time evaluating oblique and sideways camera motions. Instead, we focused on examining 2-D tracking with straightforward camera motions only for translation and scale changes in images.

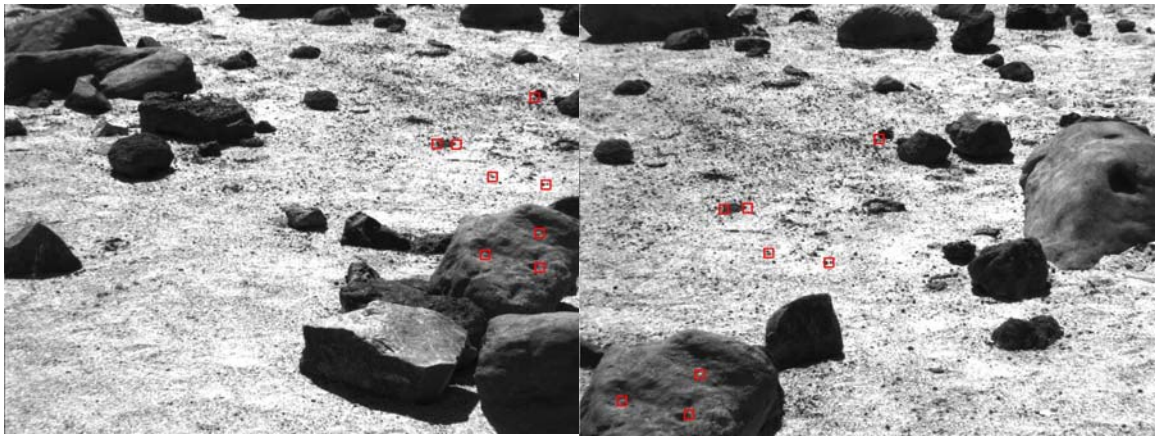


Figure 36. 2-D target tracking with oblique camera motion using a 16 mm lens: initial image (left) and end image after 1.6 m oblique move (right).

Figure 37. 2-D target tracking with oblique camera motion using a 4 mm lens: initial image (left) and end image after full 4-m oblique move (right).

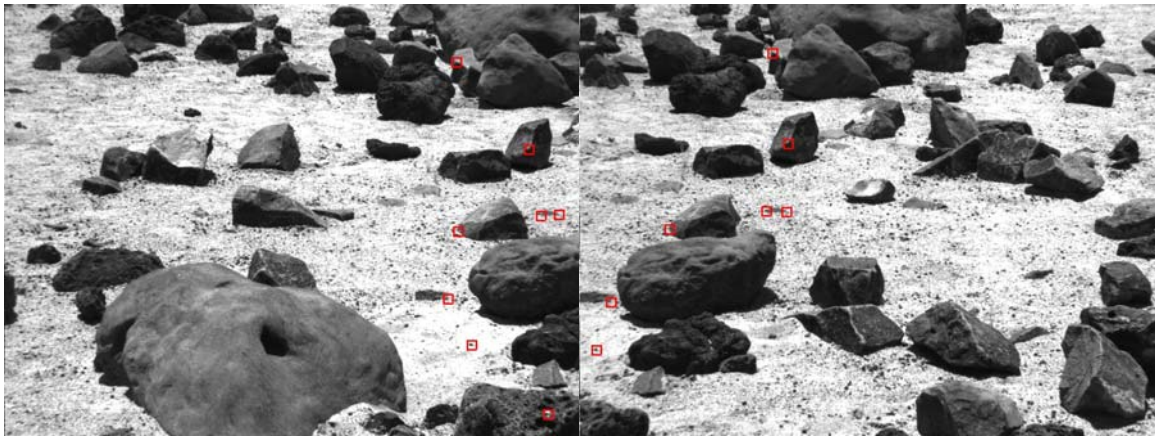


Figure 38. 2-D target tracking with sideways camera motion using a 16 mm lens: initial image (left) and the end image after 1.2 m sideways move (right).

Figure 39. 2-D target tracking with sideways camera motion using a 4 mm lens: initial image (left) and the end image after 3.3 m sideways move (right).

6.8 Tracking with Actual Rocky8 Mast Camera Images

Figure 40 shows a series of the Rocky8 image data. Rocky8 mast camera images were collected every 3 cm over a 10 m straightforward rover motion. Targets tracked well for the first 1.8 m (top right of Figure 40) until the rover went over a rock, which caused large image displacements (sometimes over 100 pixels). By 2.4 m, all targets were lost. Again, it clearly indicates that active camera control is essential. Since we will be evaluating 2-D/3-D tracking next and it provides active camera control, we decided not to spend further time evaluating actual Rocky8 mast camera images without active camera control.

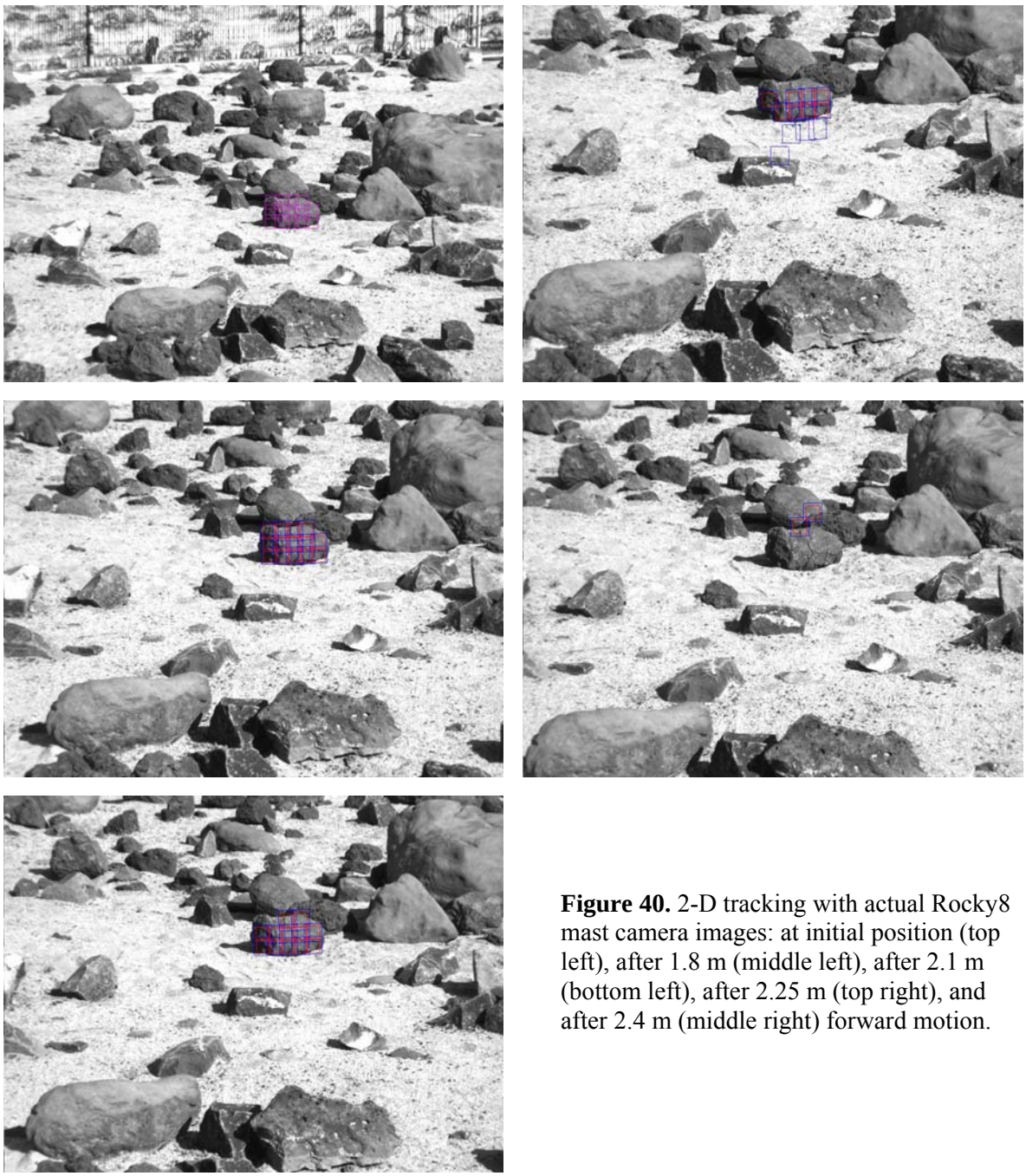


Figure 40. 2-D tracking with actual Rocky8 mast camera images: at initial position (top left), after 1.8 m (middle left), after 2.1 m (bottom left), after 2.25 m (top right), and after 2.4 m (middle right) forward motion.

6.9 Average Tracking Performances from Forward, Roll, and Yaw Camera Motions

To collect a statistical overview of 2-D tracking performance, we ran 2-D tracking on three image sequences of 4-m straightforward, 90° roll, and 45° yaw camera motions. Tracking targets were selected over several rocks of small, medium, and large image sizes. Tracking parameters were selected to be values based on experimental results described in previous Sections. The feature tracking window size was set to 15 pixels × 15 pixels in all tests. The number of pyramid levels was set to 3 for straightforward camera motion, and set to 4 for roll and yaw camera motions to accommodate larger image displacements. The image skip is set to 4 for forward motion and 0 for roll and yaw motions. The template update interval was set to 10 or every 10-th image processed for all motions. These parameter seetings makes the template update occur every 50 cm for forward motion, every 10° for roll moton, and every 2.5° for yaw camera motion. Table 11 shows the test results of 2-D tracking for forward, roll, and yaw camera motions with 16, 8, 4, and 2.3 mm lenses. The average tracking performances were 80% to 100%. The travel distance of the forward motion was full 4 m for all lenses. The total roll motion was full 90° for the 8-mm lens but 50° for the 16-mm lens due to target images going out of camera view. The total yaw motion was full 45° for both 16-mm and 8-mm lenses.

	16 mm (17ºx13º)	8 mm (33ºx25º)	4 mm (65ºx49º)	2.3 mm (113ºx86º)
Forward Motion (4 m)	65/65 (100%)	40/45 (89%)	31/31 (100%)	23/23 (100%)
Roll Motion (50°/90°)	13/16 (81%)	14/14 (100%)		
Yaw Motion (90°)	10/12 (83%)	14/16 (88%)		

Table 11. Average tracking performance with forward, roll, and yaw camera motions.

Figures 41-44 show beginning and end images of 2-D tracking of Table 11 for 4-m forward camera motions with 16, 8, 4, and 2.3 mm, respectively. For the 16 mm lens (Figure 41), all selected targets including the ones on large-image-size rocks tracked well with a 100% success rate. Each target seems to have sufficient texture to track. For the 8 mm lens (Figure 42), all 20 targets on small-image-size and medium-image-size rocks tracked 100%, while 5 out of 25 targets on large-image-size rocks failed. The failed targets do not seem to be not enough texture to track well. For the 4 mm lens (Figure 43) and 2.3 mm lens (Figure 44), all targets tracked well with 100% success.

Figures 45 and 46 show beginning and end images of 2-D tracking of Table 11 for roll camera motions with 16 and 8 mm, respectively. Three targets were lost out of 16 selected targets for the 16 mm lens (Figure 45), while all targets were tracked for the 8 mm lens (Figure 46). The 16-mm lens allowed only 40° roll motion for tracking experiments because target images could not be selected to be contained in full 90° roll motion. The problem was that the 16-mm lens happened to be at a far outer position of the camera head during the image collection with camera roll motion.

Figures 47 and 48 show beginning and end images of 2-D tracking of Table 11 for yaw camera motions with 16 and 8 mm, respectively. One target was lost out of 12 selected targets for the 16 mm lens (Figure 47), while two targets were lost out of 16 for the 8 mm lens (Figure 48). We investigated on why some targets lost tracking and on how to achieve more reliable tracking. These investigations are described in next several sections.

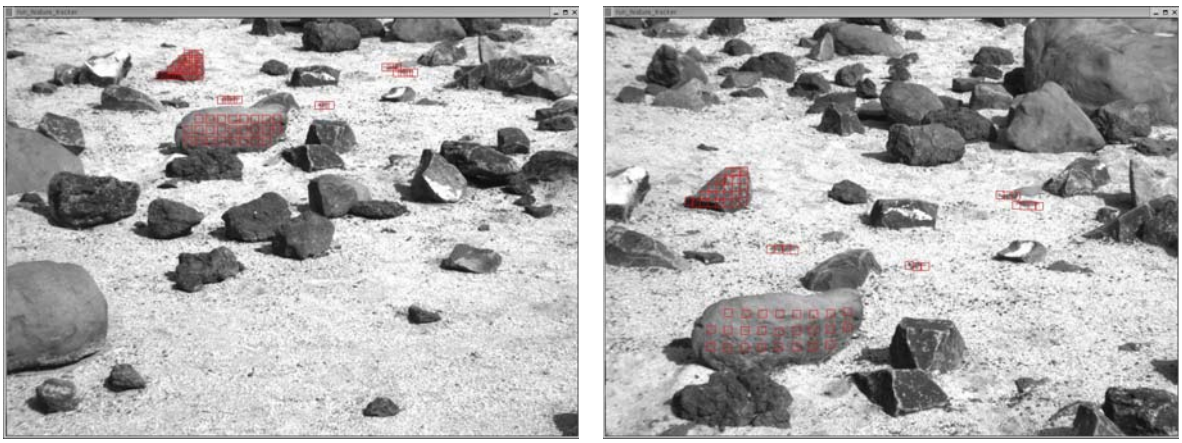


Figure 41. Beginning (left) and end (right) images of 4-m straightforward camera motion with 16 mm lens. Targets were selected on several rocks of small, medium, and large image sizes in grid patterns by using a text-based entry.

Figure 42. Beginning (left) and end (right) images for 4-m forward motion with 8 mm lens

Figure 43. Beginning (left) and end (right) images for 4-m forward motion with 4 mm lens

Figure 44. Beginning (left) and end (right) images for 4-m forward motion with a 2.3 mm lens

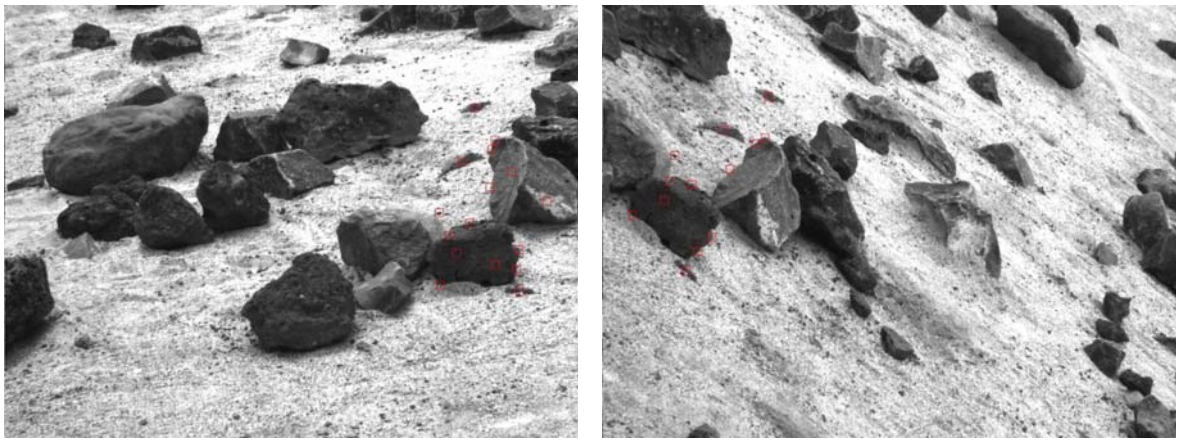


Figure 45. Beginning (left) and end (right) images for 40º roll motion with 16 mm lens. Targets were out of camera view beyond the 40º roll motion selected.

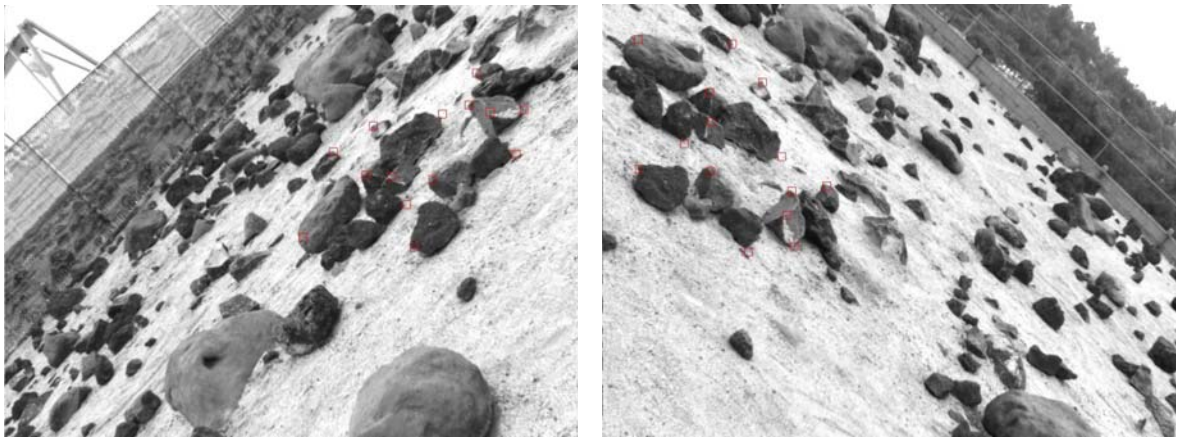


Figure 46. Beginning (left) and end (right) images for 90º roll motion with 8 mm lens.

Figure 47. Beginning (left) and end (right) images for yaw camera motion with 16 mm lens

Figure 48. Beginning (left) and end (right) images for yaw camera motion with 8 mm lens

6.10 Tracking with Small-Image-Size Rocks

Since the average tracking performance was only 80% to 100% as described in Section 4.11, we investigated further as to when tracking fails and how it can be improved. More careful examination revealed that 2-D tracking with small-image-size rocks was 100% successful for all forward, roll, and yaw camera motions, as long as target rocks were not occluded during the course of tracking. Figure 49 shows such an example of tracking targets on several small-image-size rocks with roll camera motions. Red squares superimposed on the images represent updated tracking windows, which were always reset to the upright orientation for computational efficiency. Thus, four corners of tracking windows between images do not correspond to each other; only the center position does. A good way to observe how well a combined pure translation and affine transform tracking works is to display the affine transformed quadrilateral windows corresponding to the initial tracking square windows. In Figure 49, after tracking over 70° roll camera motion, all affine transformed quadrilateral windows (blue) are about 70° rotated relative to the upright feature square windows (red) without much scale change. This clearly depicts an excellent tracking performance.

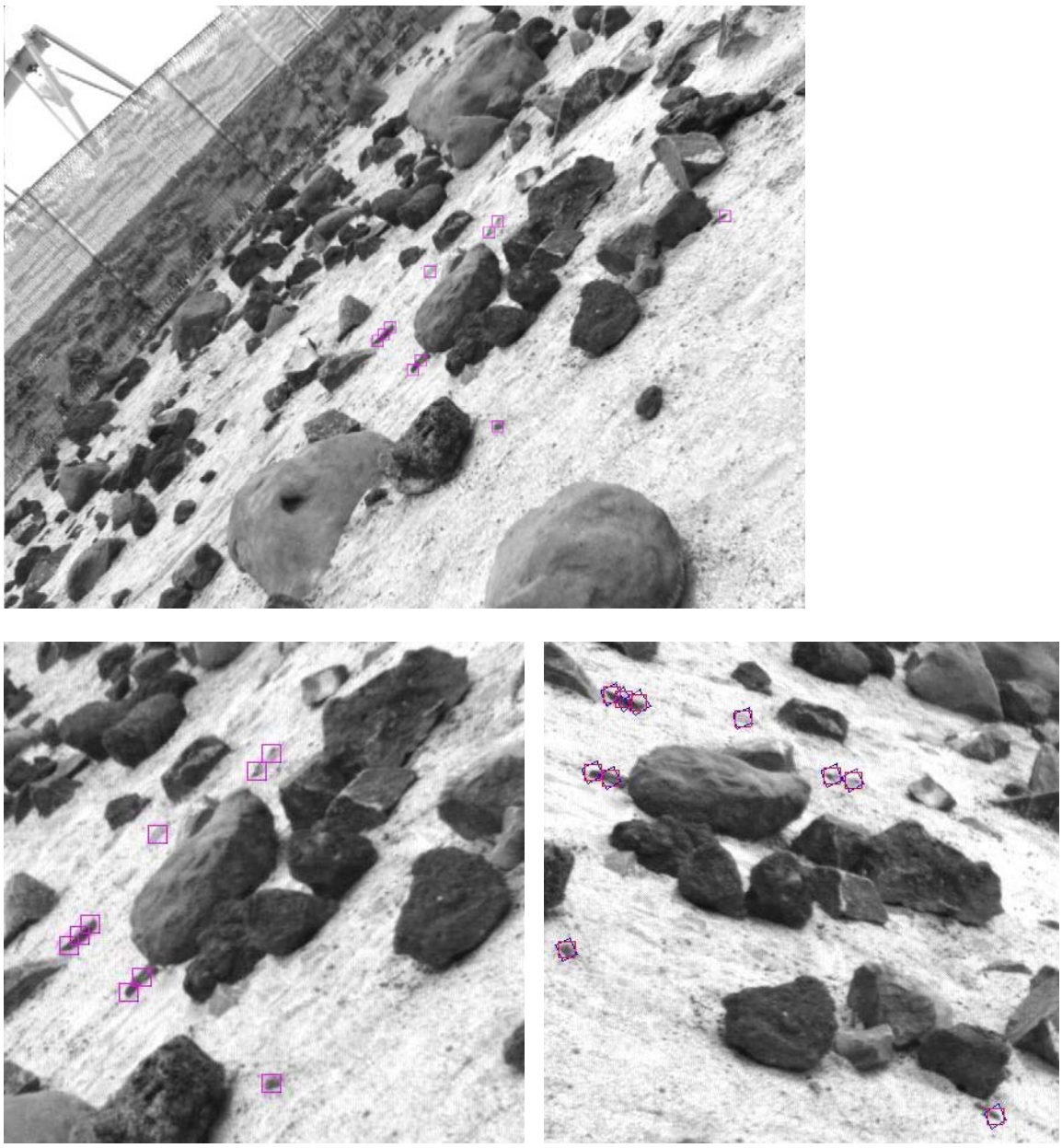


Figure 49. 2-D tracking of small-image-size rocks over 70º roll camera motion with 8 mm lens. A portion of the beginning image (top) was enlarged (bottom left), showing initial target feature windows in red squares. All targets tracked well using a 15×15 window with 3 pyramid levels. Some targets were about to move out of field of view of the camera at roll = 25° (bottom right). Affine transformed windows (blue quadrilaterals) show an excellent tracking performance.

6.11 Forward-Motion Tracking with Large-Image-Size Rocks

For forward camera motion, the biggest problem was to track targets selected on large image-size rocks with the 8mm lens. Unlike targets on small-image-size rocks, some of these targets lost tracking. Figure 50 show tracking results for 15×15 window with 4 pyramid levels. Only 7 out of 12 targets or (58%) tracked. Affine transformed windows for these lost or strayed targets were often greatly deformed. It appeared that the lost targets did not have sufficient texture within the tracking windows. So we approached this problem by increasing the feature tracking window size to include more texture. In this study, the effective window size was kept the same. Namely, when the window size is doubled, the number of pyramid levels is reduced by 1.

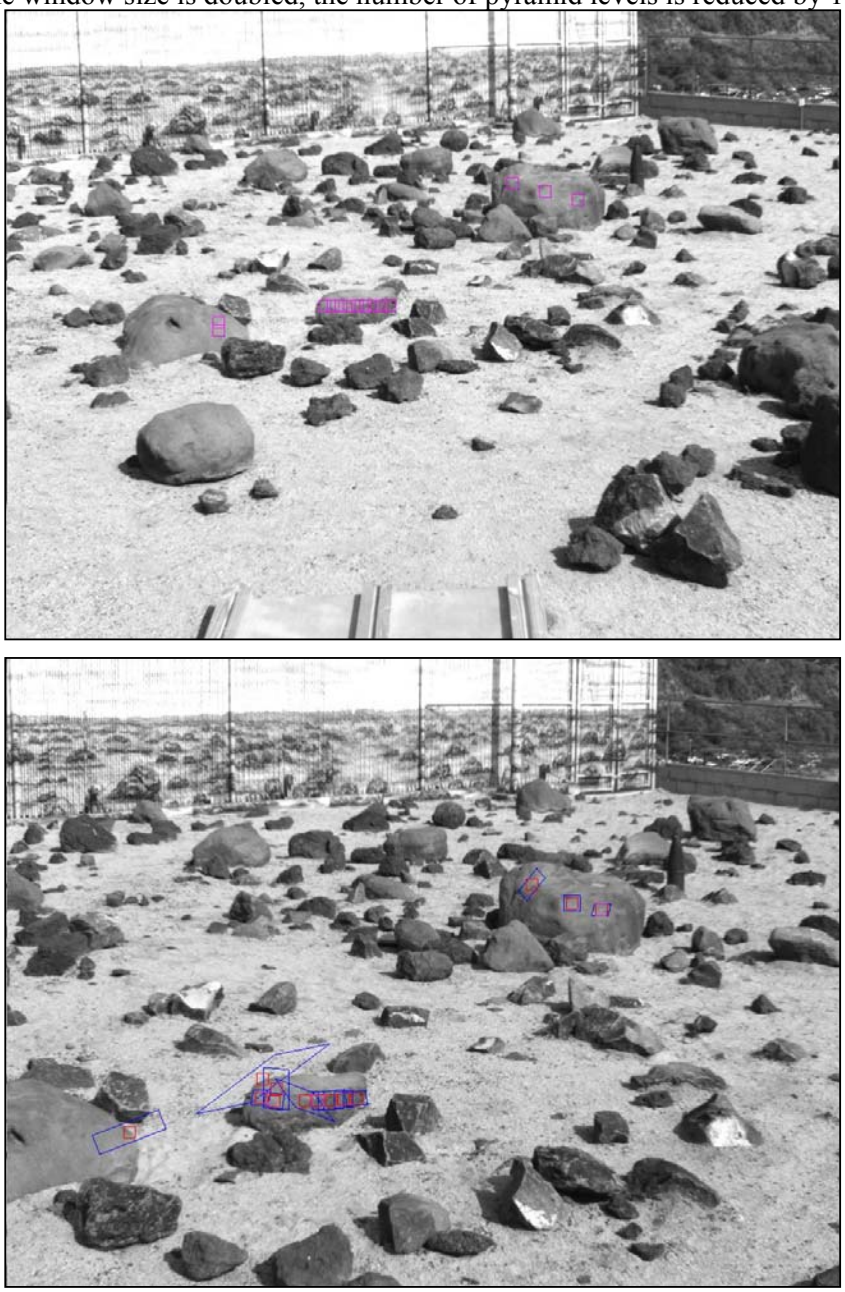


Figure 50. Initial image (top) and end image after tracking over 4-m forward motion (bottom) using 15×15 window with 4 pyramid levels.

When the window size was increased to 29×29 with 3 pyramid levels, all 12 targets tracked, clearly demonstrating that low-texture targets on large-image-size rocks need a larger window size. However, one target on the leftmost rock in Figure 51 drifted noticeably over the surface of the rock and its affine transformed window collapsed. The 6-parameter affine transform was a lot more sensitive to low-texture targets than the 2-parameter pure translation, resulting in the affine transformed window badly deformed for some "sensitive" targets. Since we used a combined configuration of pure translation and affine transform, pure translation helped not to lose these "senstive" targets.

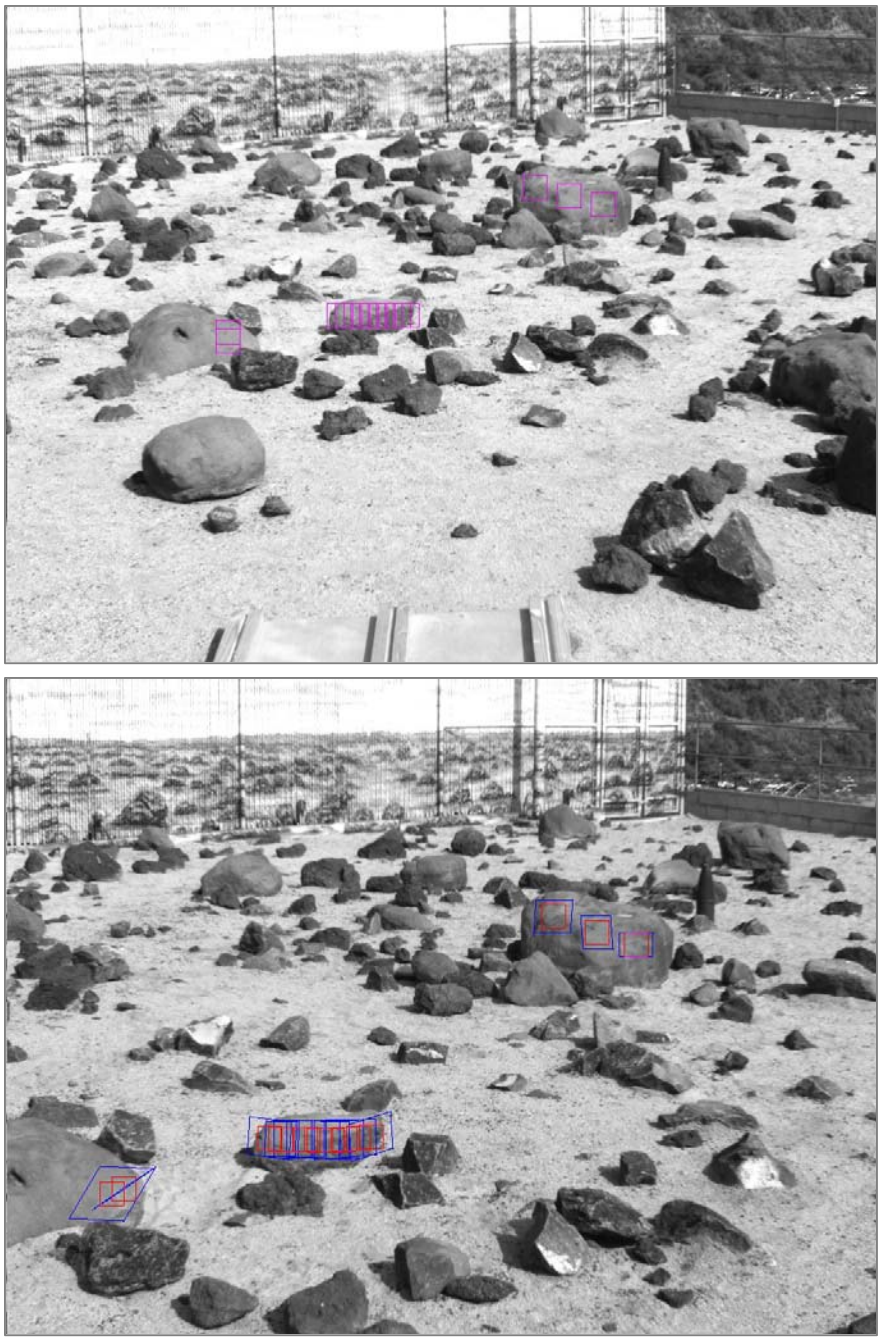


Figure 51. Initial image (top) and end image after tracking over 4-m forward motion (bottom) using 29×29 window with 3 pyramid levels.

When the feature track window size was further increased to 59×59 with 2 pyramid levels (Figures 52 and 53), again all 12 targets tracked. Since the current software could not run all 12 targets in a single run for the large-size window (due to the item 4 bug in Section 4.5), we ran in two separate runs. All 12 targets tracked well with no noticeable drifts in terms of the center positions of the targets. We noted, however, that affine transformed windows were pulled towards adjacent occluding/occluded rocks. In this example, 59×59 window with 2 pyramid levels performed slightly better than 29×29 window with 3 pyramid levels.

Figure 52. Initial image (top) and end image after tracking over 4-m forward motion (bottom) using 59×59 window with 2 pyramid levels. More targets were selected in Figure 53.

Two affine transformed quadrilaterals were oriented quite differently for the lower-left rock in the bottom image of Figure 53. One was pulled towards the occluding rock in the front, while the other was pulled towards the occluded rock behind. Several affine transformed quadrilaterals for both upper-right and lower-middle rocks were also pulled by the occluded distinct rocks behind. As the camera moves, relative rock locations in the image change. This causes the affine transform to be distorted inaccurately.

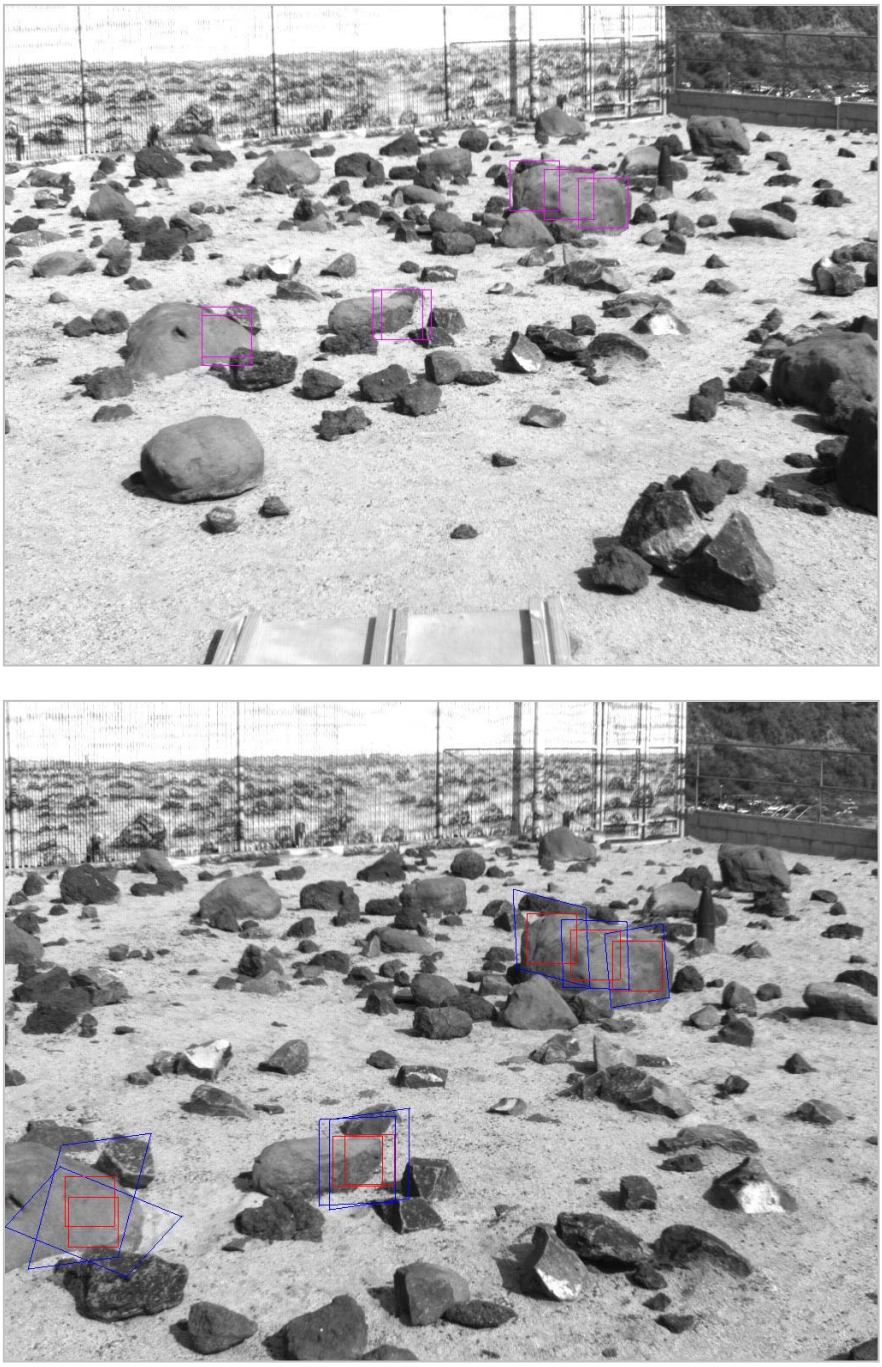


Figure 53. Initial image (top) and end image after tracking over 4-m forward motion (bottom) using 59×59 window with 2 pyramid levels. Different targets were selected here.

6.12 Roll-Motion Tracking with Large-Image-Size Rocks

Among three different tests with roll camera motions, for a couple of times all targets selected from large-image-size rocks tracked 100% successfully even with the relatively small 15×15 window. In the first example shown in Figure 54 for an 8 mm lens, all 15 targets selected from a large rock tracked well over the entire 90º roll using 15×15 window size with 4 pyramid levels.

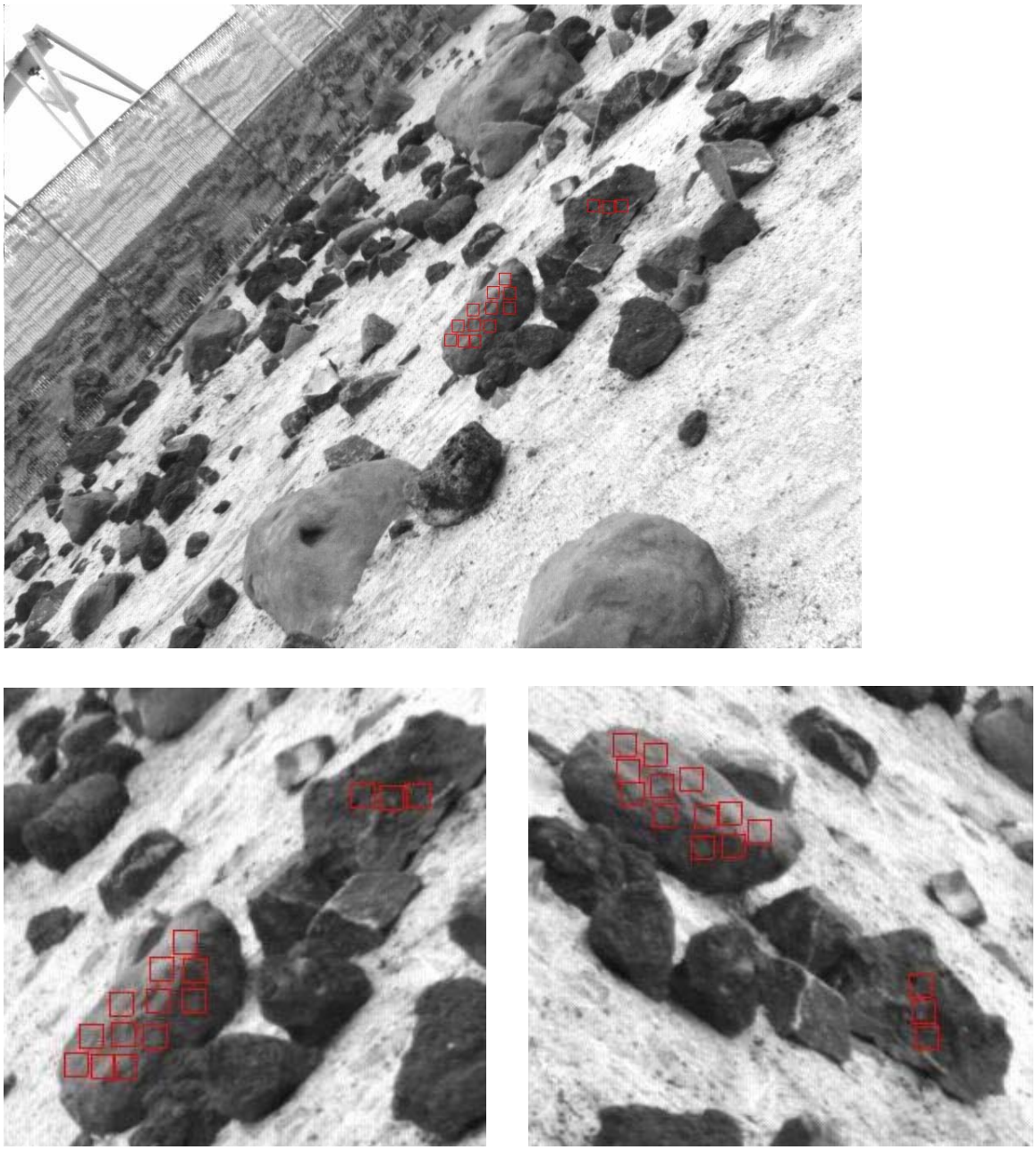


Figure 54. All 15 targets selected on two large-image-size rocks for the 16-mm lens tracked well through a 90° roll camera motion, using a window size of 15×15 and 4 pyramid levels: initial image (top), its close- up (bottom left) showing targets selected, and an enlarged portion of the end image after the 90º roll (bottom right).

A second example of good tracking is shown in Figure 55 for 16 mm lens. All 5 targets tracked well through a 60° roll using 15×15 window with 3 pyramid levels.

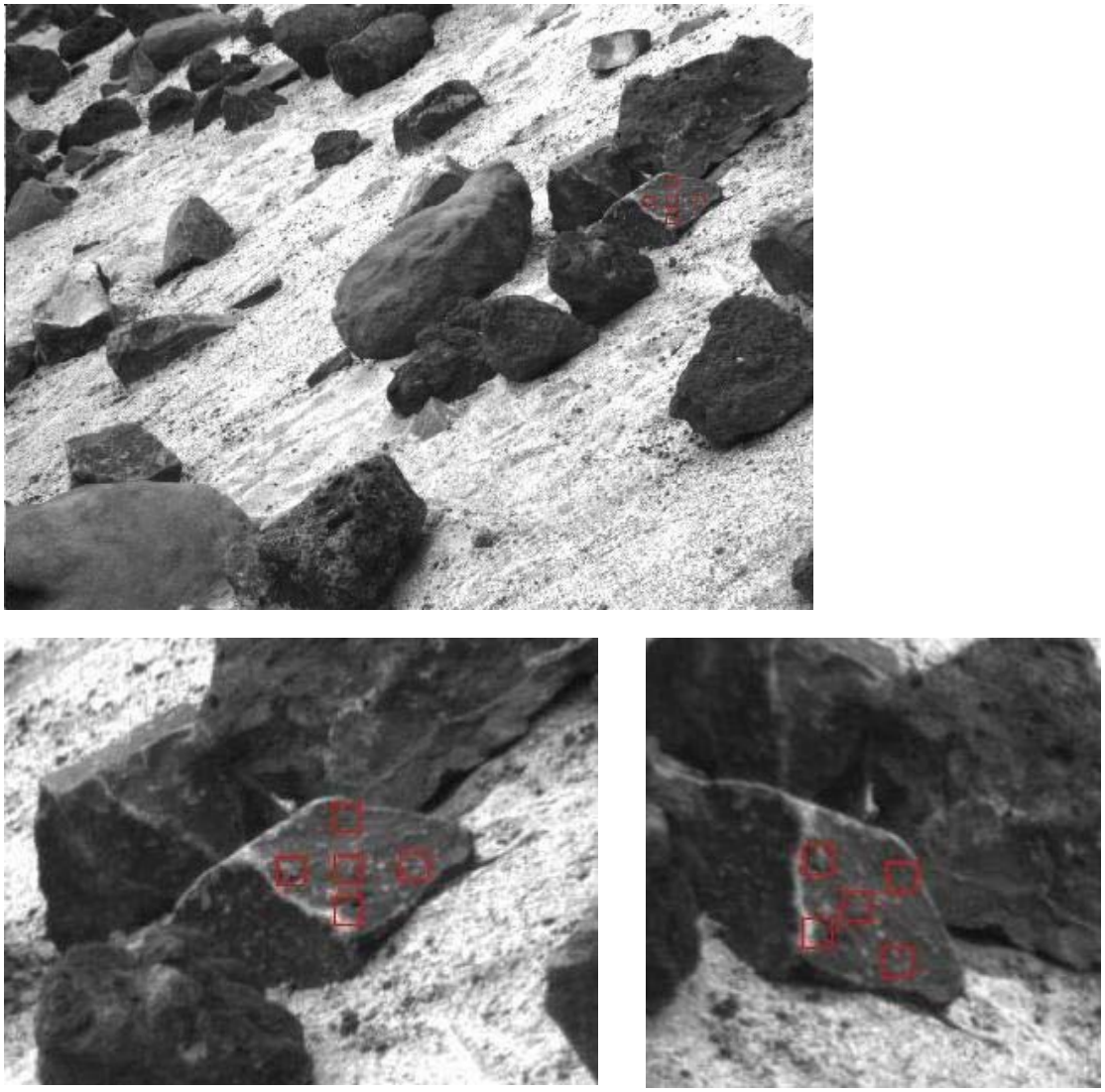


Figure 55. All 5 targets selected on the face of another large-image-size rock also tracked well using a window size of 15×15 and 4 pyramid levels: initial image (top), its close-up (bottom left) showing targets selected, and an enlarged portion of the end image after 90º roll (bottom right).

In the third example, however, 15×15 window with 3 pyramid levels did not yield 100% tracking for the 16 mm lens. As shown in Figure 56, one target out of 12 on a large-image-size rock lost tracking over 40° roll change. Further, several of affine transformed windows were quite poor. The tracking was performed only up to 40°, because the rock went out of camera view afterwards. Increasing the pyramid level alone did not help; 15×15 window with 4 pyramid levels yielded almost identical tracking performance. When the window size was increased to 29×29 with 2 pyramid levels, the same one target still lost tracking for some unknown reason while all other 11 targets showed excellent tracking with very good affine transformed windows. However, when the 29×29 window with 3 pyramid levels (Figure 57) is used, finally all 12 targets tracked very well. In particular, affine transformed windows were also very good. This example clearly demonstrates that larger window helps tracking for low-texture targets.

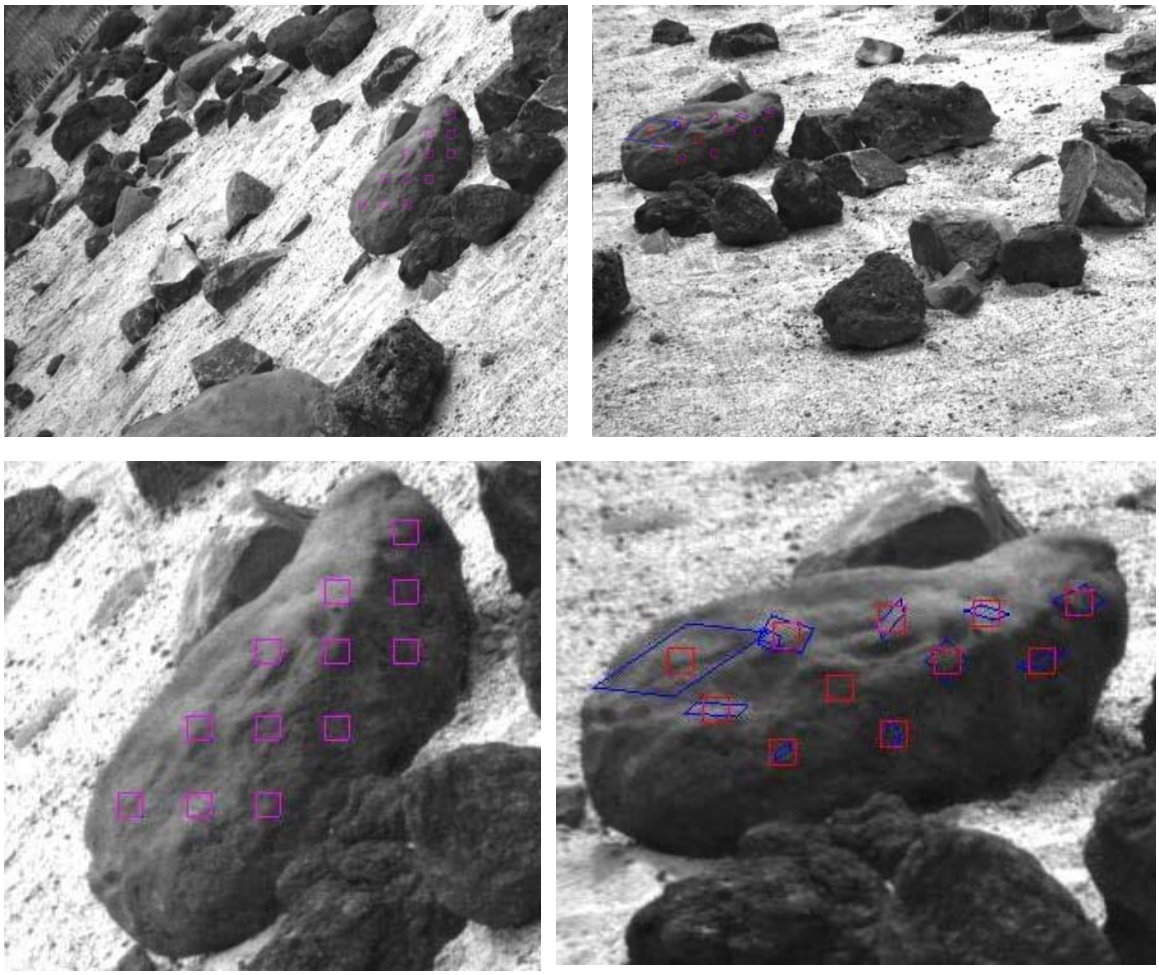


Figure 56. Beginning (top left) and end image after 40º roll (top right) using 15×15 window with 3 pyramid levels. Their close-up views are in bottom left and bottom right.

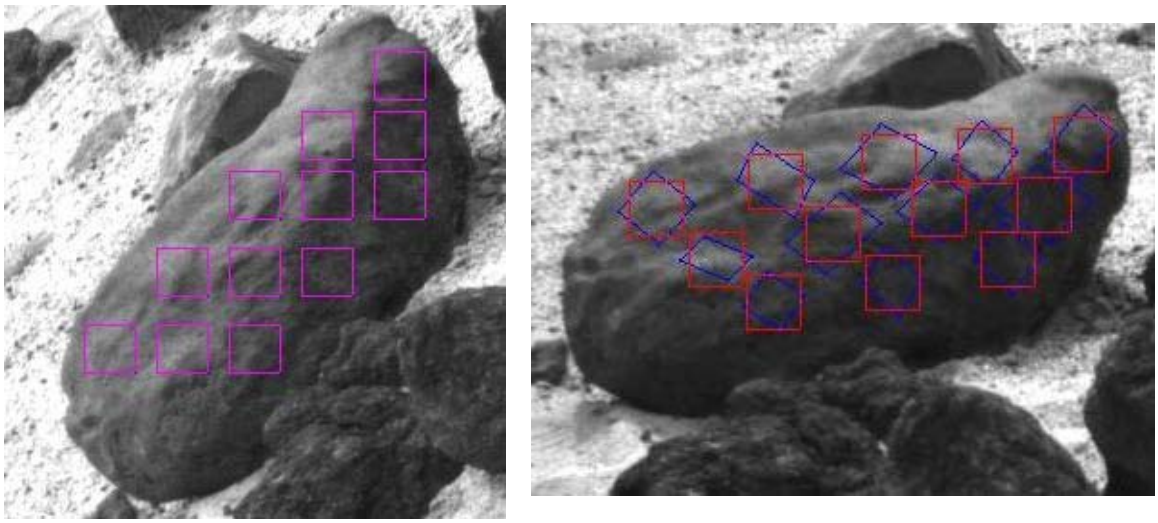


Figure 57. Beginning (top left) and end image after 40º roll (top right) using 29×29 window with 3 pyramid levels. Their close-up views are in bottom left and bottom right.

6.13 Yaw-Motion Tracking with Large-Image-Size Rocks

The yaw camera motion in this report actually involves both camera translation and yaw motion as illustrated in Figure 58. As the camera head travels 4 m along the linear motion stage, its yaw orientation was changed to point to a same target located at about 5-m, and the total yaw change was about 44° from -22° to +22°. Since the pointing adjustment of the camera head was done visually, the collected image sequence introduced large image displacements, in particular, for the 16-mm lens. In order to track a target with large image displacements between images, 2-D tracking with the yaw camera motion often needed a larger effective window size or one higher number of pyramid levels compared to forward or roll motion tracking. In the example shown in Figure 58, the 15×15 window with 3 pyramid levels was good enough for the 8-mm lens. All 16 targets on large-image-size rocks tracked over the entire 45º yaw.

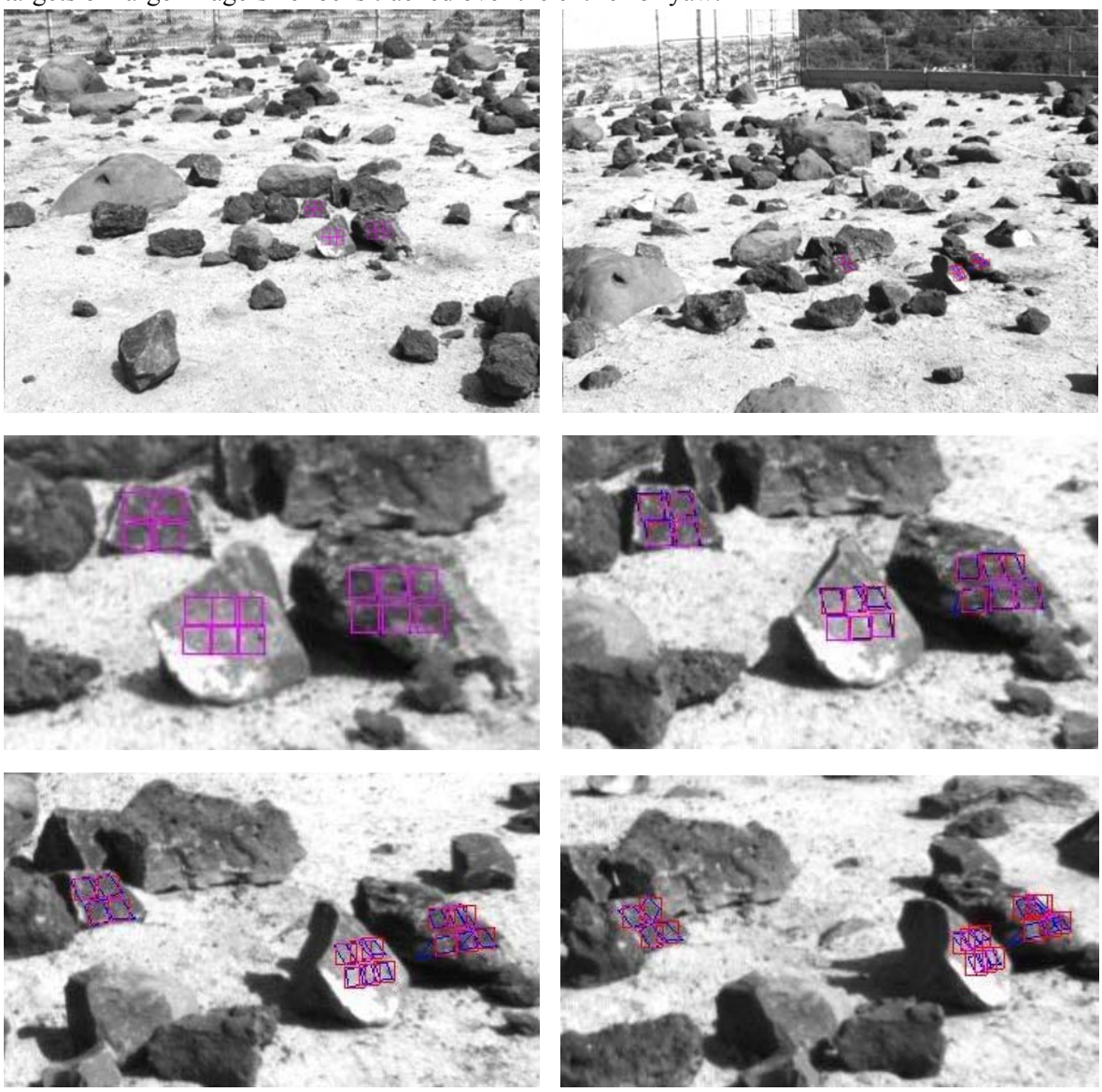


Figure 58. Beginning (top left), end (top right) images for 45º yaw camera motion with 8 mm lens. Enlarged close-up views show target tracking windows at yaw of 0º (middle left), 15º (middle right), 30º (bottom left) and 45º (bottom right).

As in forward and roll camera motions, 15×15 window size was often not large enough to track low-texture targets on large-image-size rocks for yaw camera motion. Further, the yaw camera motion with the 16-mm lens required higher number of pyramid levels due to large image displacements introduced during the image collection. In an example shown in Figure 59, the 29×29 window with 3 pyramid levels did not provide good tracking performance. Out of total 12 targets selected, two targets were lost and two other targets drifted much. There were quite a few image displacements over 30 pixels up to 38 pixels. To handle these large image displacements, the number of pyramid levels was raised to 4. Tracking performance using 29×29 window with 4 pyramid levels (Figure 60) provided good tracking performance except two affine transform windows were a little off. It was interesting to observe that an effective window size of about 120 pixels was not enough to handle 30-pixel or larger displacements, while an effective window size of 240 pixels was able to handle them reliably. We also ran 59×59 and 75×75 windows with 3 pyramid levels, both of which showed good tracking performance with reasonable affine transformed windows (Figures 61 and 62). Since the affine transform assumes the planar target, larger window size helps tracking only if the target surface is approximately planar within the target tracking window. Also doubling the window size (in each dimension) increases the computational time by a factor of 4.

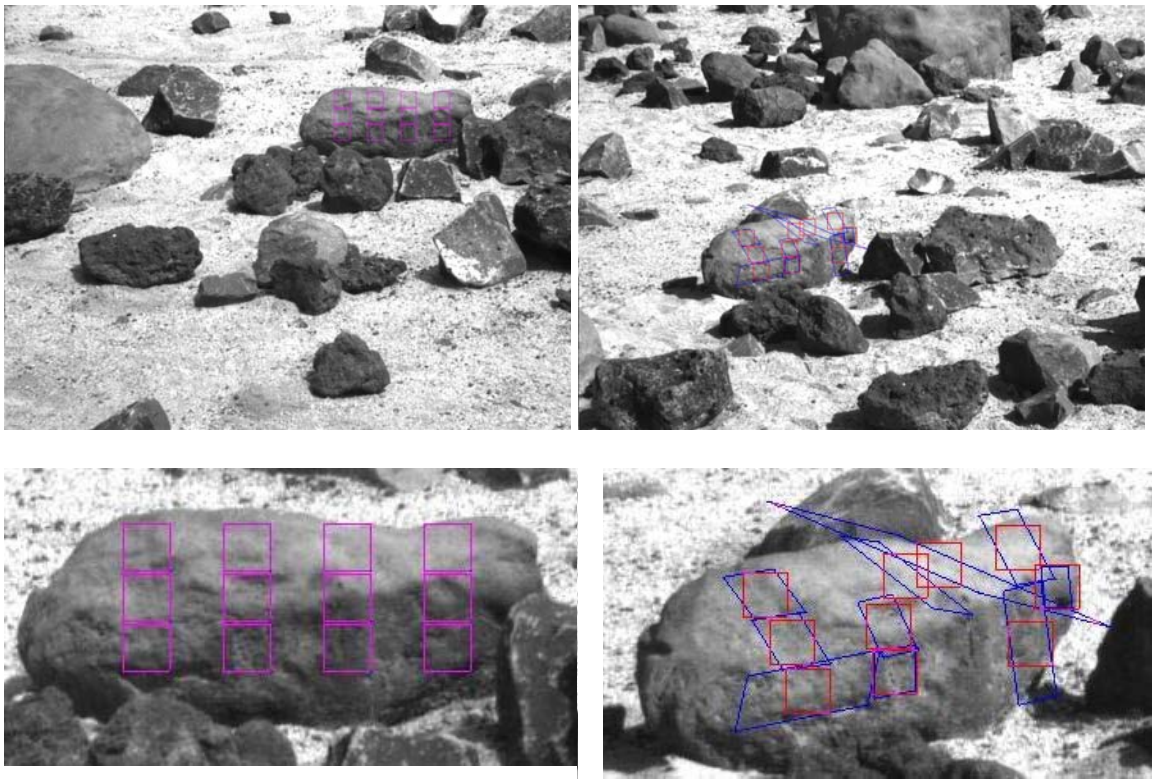


Figure 59. Begin/end images after 45º yaw motion using 29×29 window with 3 pyramid levels: full images (top) and enlarged views of 12 targets (bottom).

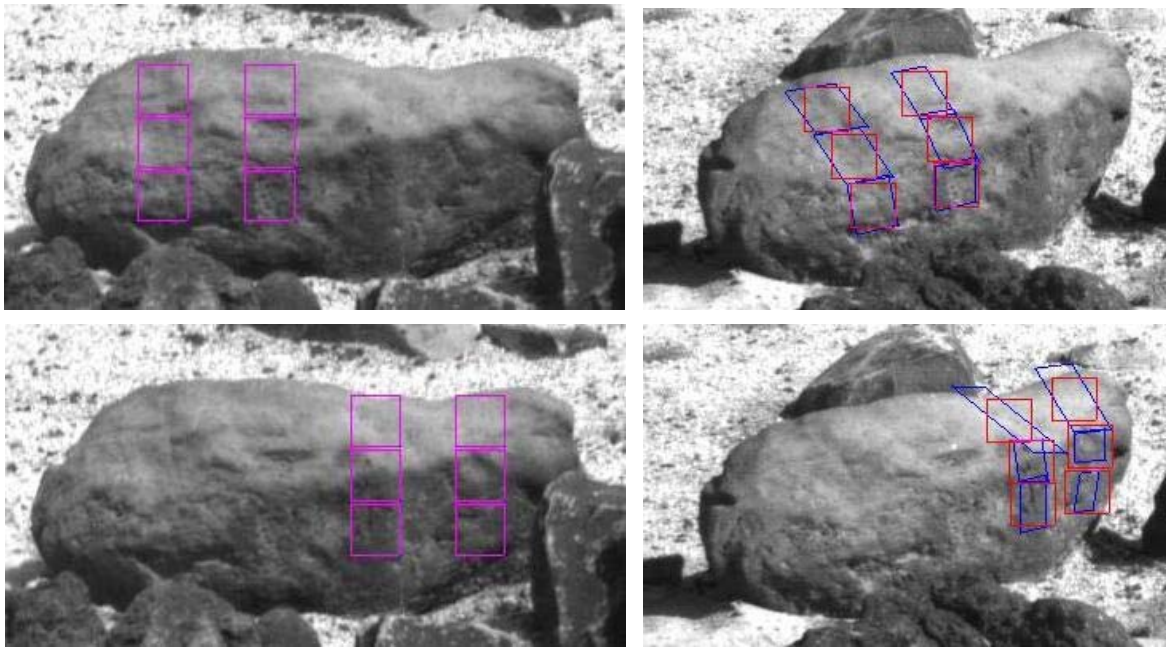


Figure 60. Begin/end images for 45º yaw motion using 29×29 window with 4 pyramid levels**:** enlarged views for six targets (middle) and for six others (bottom)

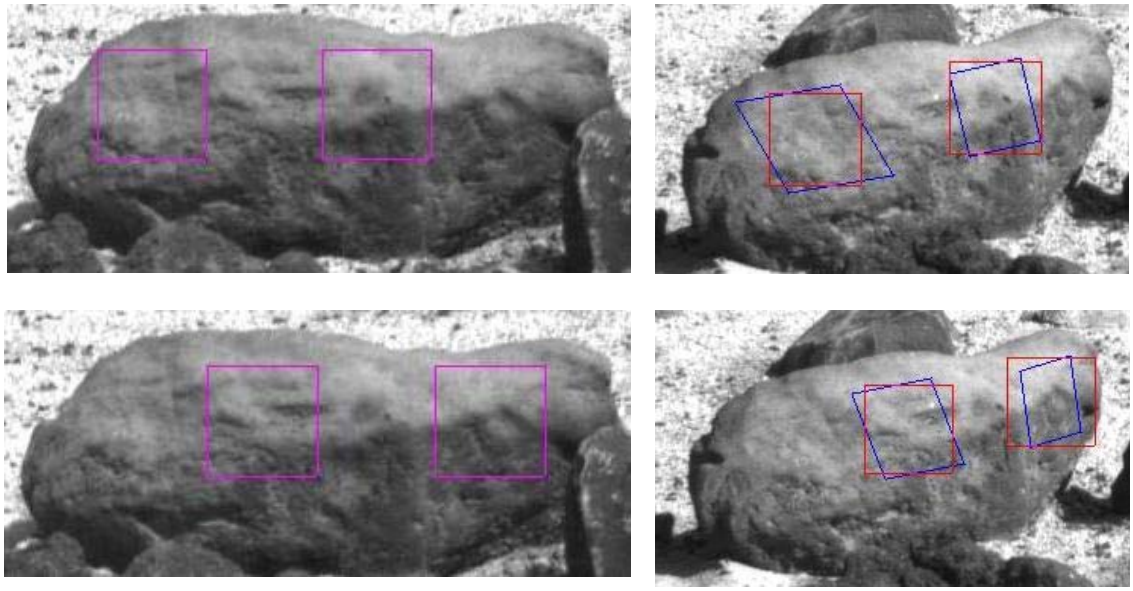


Figure 61. Begin/end images for 45º yaw motion using 59×59 window with 3 pyramid levels for two targets (top) and for two others (bottom)

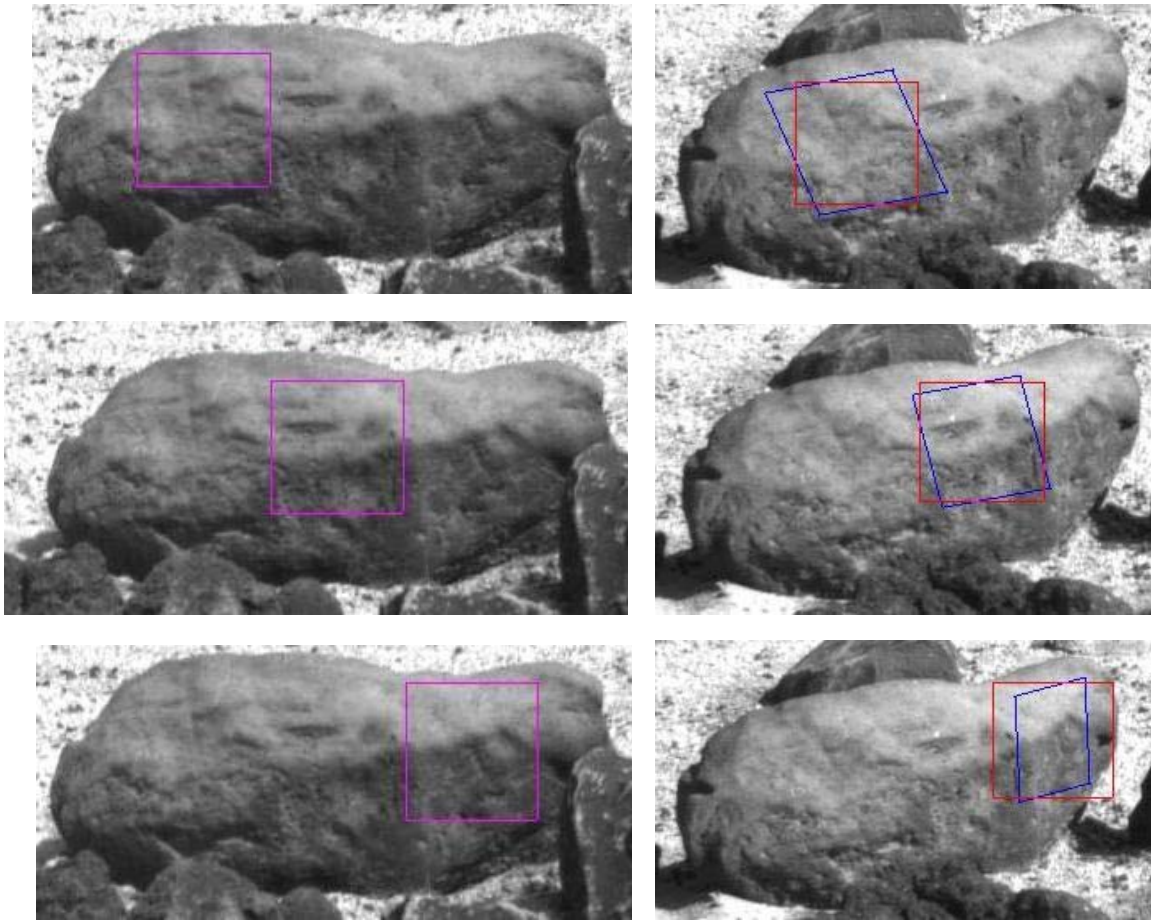


Figure 62. Begin/end images for 45º yaw motion using 75×75 window with 3 pyramid levels for three different targets (top, middle, and bottom)

Tracking performance was further tested with another large-image-size rock for 16 mm lens. As shown in Figure 63, only 6 targets out of 9 tracked for both 15×15 and 29×29 windows with 4 pyramid levels. Top three targets lost tracking, while six others tracked well. We noted that the background above the top occluding boundary of the rock changed dramatically from white to black during the course of tracking due to the shift of the dark shadow of an adjacent rock as the camera yaw changes. More cautious target selection is required in a clutered environment. It is best if the target image does not encounter dramatic shadow changes as well as occlusions.

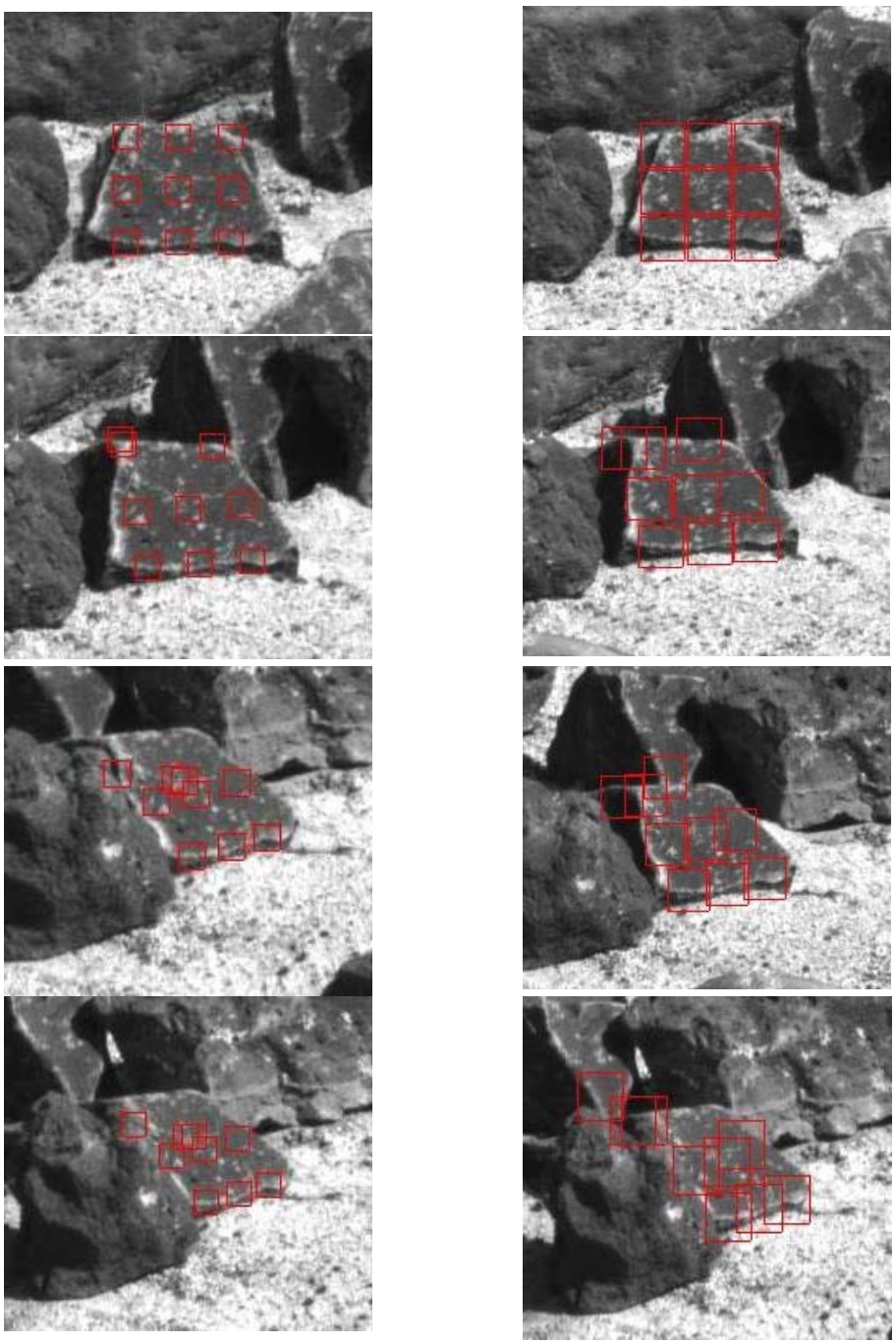


Figure 63. Enlarged close-up views showing target tracking windows at yaw of 0º (top row), 15º (second row), 30º (third row) and 45º (bottom row) for 45º yaw camera motion with 16 mm lens: 15×15 window (left column) and 29×29 window (right column) with 4 pyramid levels.

To avoid the dramatic background change, we selected the target window so that it just fits the rock using a 75×75 window as shown in Figure 64. However, both 1 and 2 pyramid levels with 75×75 window showed poor tracking performance. Only when the number of pyramid levels increased to 3, it showed good tracking performance. Poor performance with 1 or 2 pyramid levels was due to large image displacements between images up to 38 pixels. An effective window size of 75×75 or 150×150 could not overcome large image displacements reliably, while an effective size of 300×300 pixels using 3 pyramid levels could.

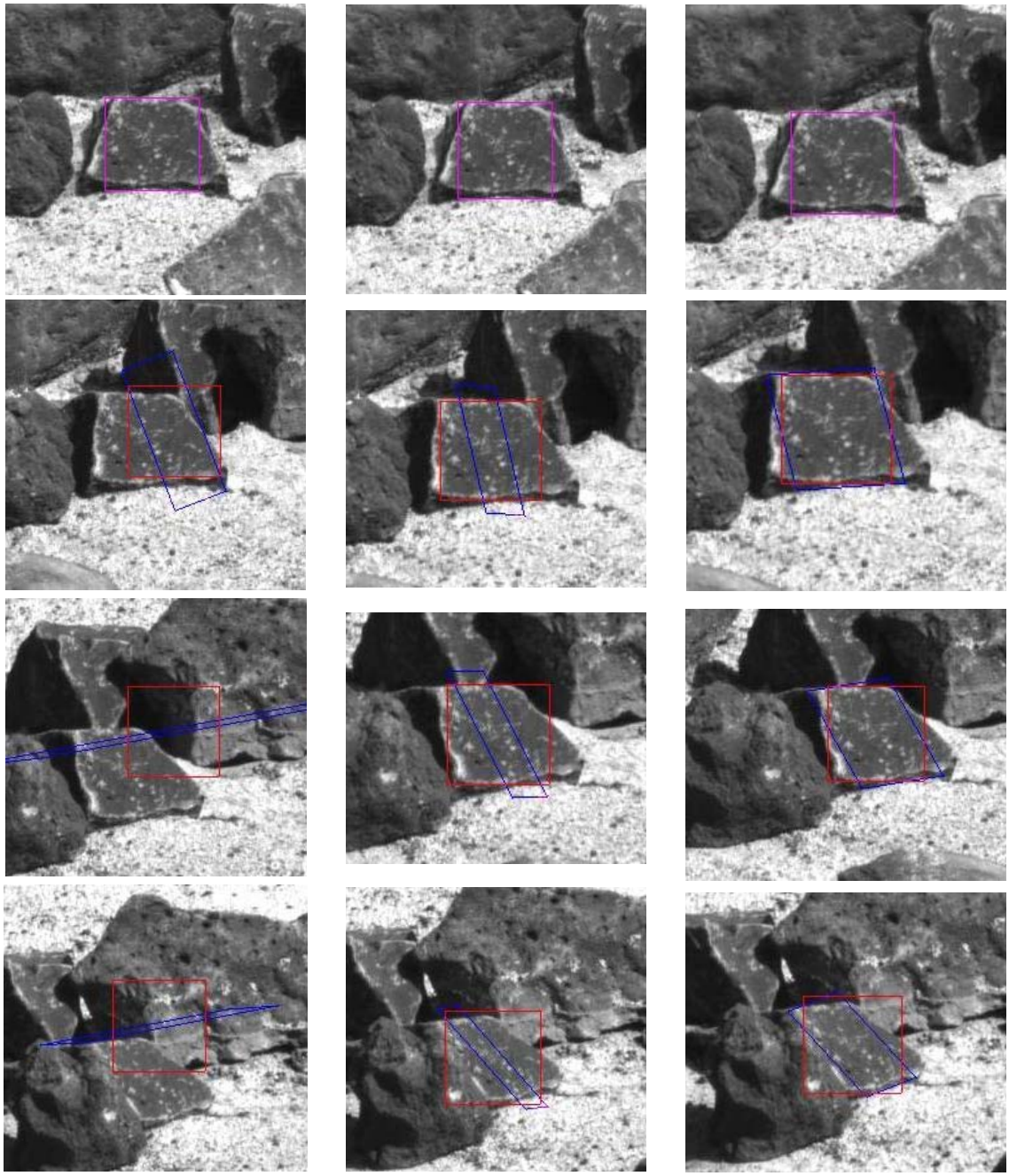


Figure 64. Enlarged close-up views showing target windows at the 0º (top row), 15º (second row), 30º (third row) and 45º (bottom row) yaw for a total 45º yaw camera motion with 16 mm lens: 75×75 window with 1 (left column), 2 (middle column), and 3 pyramid levels (right column).

6.14 Alternate Template Update Method

In the combined translation and affine transform configuration (Section 4.2) that we used in this report, the affine template was updated every time the translation template was updated. An alternate update method was tried, where only the translation template was updated while the affine template was never updated. When the end images of the two methods after 4-m forward motion were compared (Figure 65), tracking performances of these two update methods appeared to be similar at first glance.

Figure 65. Beginning image of 4-m forward-motion tracking is shown at the top panel. The end image resulted from the method updating both translation and affine templates (left) was similar to the end image resulted from the method updating translation template only (right).

In a different example run with the yaw motion (Figure 66), however, the method updating both translation and affine templates showed a better tracking performance than the alternate method. The alternate method of updating only the translation template tracked poorly at the end, although it tended to yield more accurate affine transformed window (blue) when it tracked. Careful examinations of the alternate method revealed that the ssd (square-sum-of-differences) value from affine matching kept increasing because the affine template was never updated, while the ssd value from pure translation matching reduced to near zero whenever the template was updated. Thus, except for the very beginning portion of tracking, the ssd value from affine matching was consistently higher, and affine matching was never used again later. This clearly indicates that the alternative method is not a good option.

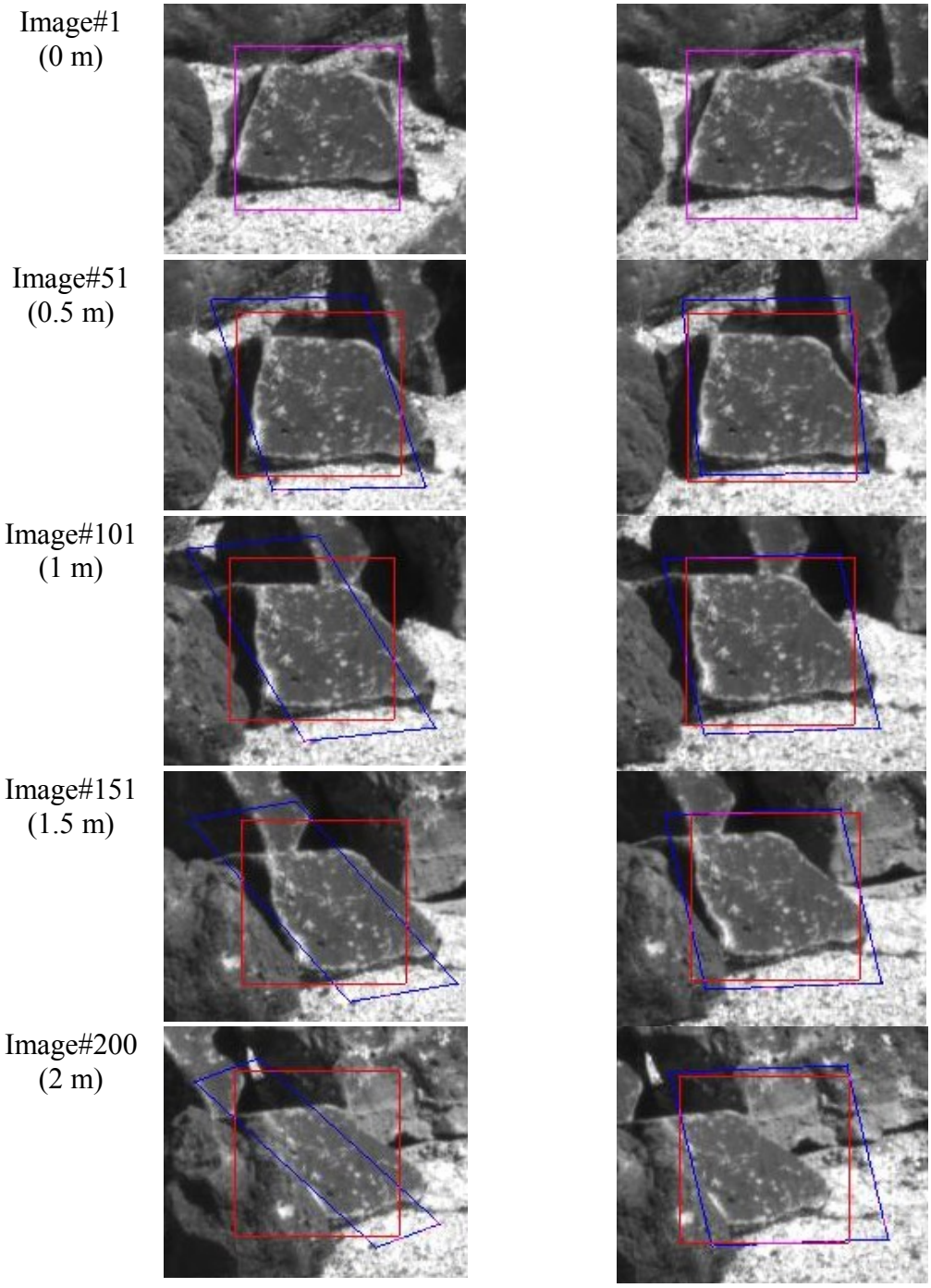


Figure 66. Comparing two update methods to track a target with yaw camera motion using 100×100 window with 3 pyramid levels: the method of updating both translation and affine templates (left column) and an alternate method of updating translation template only (right column). The alternate update method tracked poorly at the end.

7. SDOG Tracker Performances

7.1 Software Functional Description

The SDOG (Sign of difference of Gaussians) tracker tracks a target by matching the binary sign images extracted from the difference of Gaussian operations of images. The algorithm exploits the invariance of the sign of the zero-crossing in the Laplacian of the Gaussian of an image. The difference of Gaussians effectively generates the Laplacian of Gaussian, which is essentially a band-pass filter suppressing DC and low spatial frequency components while emphasizing intensity edges. The sign of difference of Gaussians builds a binary image from the gray-scale Difference of Gaussians image. The XOR (exclusive OR) logical operator, as opposed to arithmetic operator such as sum of squared differences (SSD), is used for computationally efficient binary sign correlation. In the SDOG tracker software delivered, the template binary image size is set to a 32×32 window of 1024 bits. Thus, the maximum value for 32×32 binary correlation is 1024, while the minimum is 0. The uncorrelated value is actually 512 due to the 50% random chance of matching. The search for the binary correlation peak is the brute-force method of computing the correlation values for all possible positions in the specified search area. By contrast, in the previous iterative affine tracker, the search is done by successive approximations to the solution, which can be a local minimum.

To run the SDOG tracker, three input parameters need to be specified. Two binary correlation threshold parameters are frame confidence and template confidence thresholds. The target is considered tracking if the maximum correlation value is above the frame confidence threshold. The template is updated if the maximum correlation value is below the template confidence threshold. The third input parameter is the search area window size. In our tests, the search area window size was set to 3 times the template window size, or 96×96 pixels.

7.2 Forward-Motion Tracking

The 4-m forward-motion image sequences, which were used in the affine tracker experiments earlier, were also used to test the SDOG tracker (Figure 67). The frame and template confidence thresholds were varied from 60% to 100% for every 10% step. Since it does not make sense to update the template when the tracking performance in the current image is not confident, the template confidence threshold is equal to or greater than the frame confidence threshold. The tracking experimental results with the SDOG tracker are shown in Table 12 for the 8-mmm lens and in Table 13 for the 16-mm lens. In these two tables, 70% frame confidence and 80% template confidence thresholds resulted in best tracking performance with 100% tracking for both 8-mm and 16-mm lenses, while 60% frame confidence and 80% template confidence thresholds resulted in the second best performance with 96% tracking for 8-mm lens and 100% for 16-mm lens.

In Table 14, the tracking performances of the SDOG tracker were compared with those of the affine tracker. The affine tracker was run using the same 4-m forward-motion image sequences and initial target positions. For the 8-mm lens, the affine tracker yielded 89% tracking rate with 15×15 window with 3 pyramid levels, while 98% with 31×31 window with 2 pyramid levels. The 31×31 window performed better because some low-texture targets on large rocks needed the larger window size for good tracking performance. The one lost target with the 31×31 window is shown in Figure 68. The red arrow indicates the drift direction of the lost target. To determine the cause of the tracking failure, we examined the beginning image which is essentially the same as the one shown at the top of Figure 67 for the SDOG tracker. We noted that the initial target window for the lost target actually included two separate rocks: one small rock in the middle and

a portion of another rock at the bottom side of the target window. The SDOG tracker somehow managed to track the small rock in the middle, while the affine tracker ended up drifting away the target to a position in between the two rocks. Thus, for this small rock, a smaller 15×15 window should have used to avoid including a separate rock within the target window.

For the 16-mm lens, the SDOG tracker tracked 100% (Figure 69). The affine tracker yielded 98% tracking rate with 15×15 window and 3 pyramid levels, while 91% with 15×15 window and 2 pyramid levels. Interestingly, the smaller 15×15 window performed better this time. For the 16-mm lens images tested here, targets on large rocks appeared to have enough texture to track except one lost target, which is marked with a red arrow in Figure 70. For the larger 31×31 window size, six targets were lost out of 65 total targets as shown in three groups in Figure 71 (a), (b), and (c). We identified the causes of tracking failures. The lost target in Figure 71 (a) was due to the nearby occlusion by the front rock. Among the three targets lost in Figure 71 (b), one target in the big rock was lost due to the background rock interference, and another target above the big rock was lost because its initial target window included two separate rocks (see the top image of Figure 69). The third lost target on Figure 71 (b) and two more lost targets in Figure 71 (c) are all for the same tiny rock. Since the target is so small, a smaller window size should have been used.

The comparison of the SDOG and affine trackers in Table 14 indicates that the SDOG tracker performed slightly better than the affine tracker for the forward motion. The "iterative search" affine tracker appeared to be more susceptible to poor target window selection than the "brute-force search" SDOG tracker.

Template Confidence Threshold	Frame Confidence Threshold				
	60% (614)	70% (717)	80% (819)	90% (922)	100% (1024)
60% (614)	24%				
70% (717)	62%	11%			
80% (819)	96%	100%	0%		
90% (922)	93%	98%	91%	0%	
100% (1024)	93%	96%	89%	0%	0%

Table 12. SDOG tracker tracking performance for 4-m forward motion with 8-mm lens

Template Confidence Threshold	Frame Confidence Threshold				
	60% (614)	70% (717)	80% (819)	90% (922)	100% (1024)
60% (614)	51%				
70% (717)	89%	15%			
80% (819)	100%	100%	0%		
90% (922)	95%	97%	94%	0%	
100% (1024)	94%	94%	94%	6%	0%

Table 13. SDOG tracker tracking performance for 4-m forward motion with 16-mm lens

Camera motion, Lens focal length	SDOG, 32x32 window		Affine	
	FC=60% TC=80%	FC=70% TC=80%	31x31 window 2 pyr. levels	15x15 window 3 pyr. levels
Forward, 8-mm	96% (43/45)	100% (45/45)	98% (44/45)	89% (40/45)
Forward, 16-mm	100% (65/65)	100% (65/65)	91% (59/65)	98% (64/65)

Table 14. SDOG tracker performed slightly better than affine tracker for 4-m forward motion.

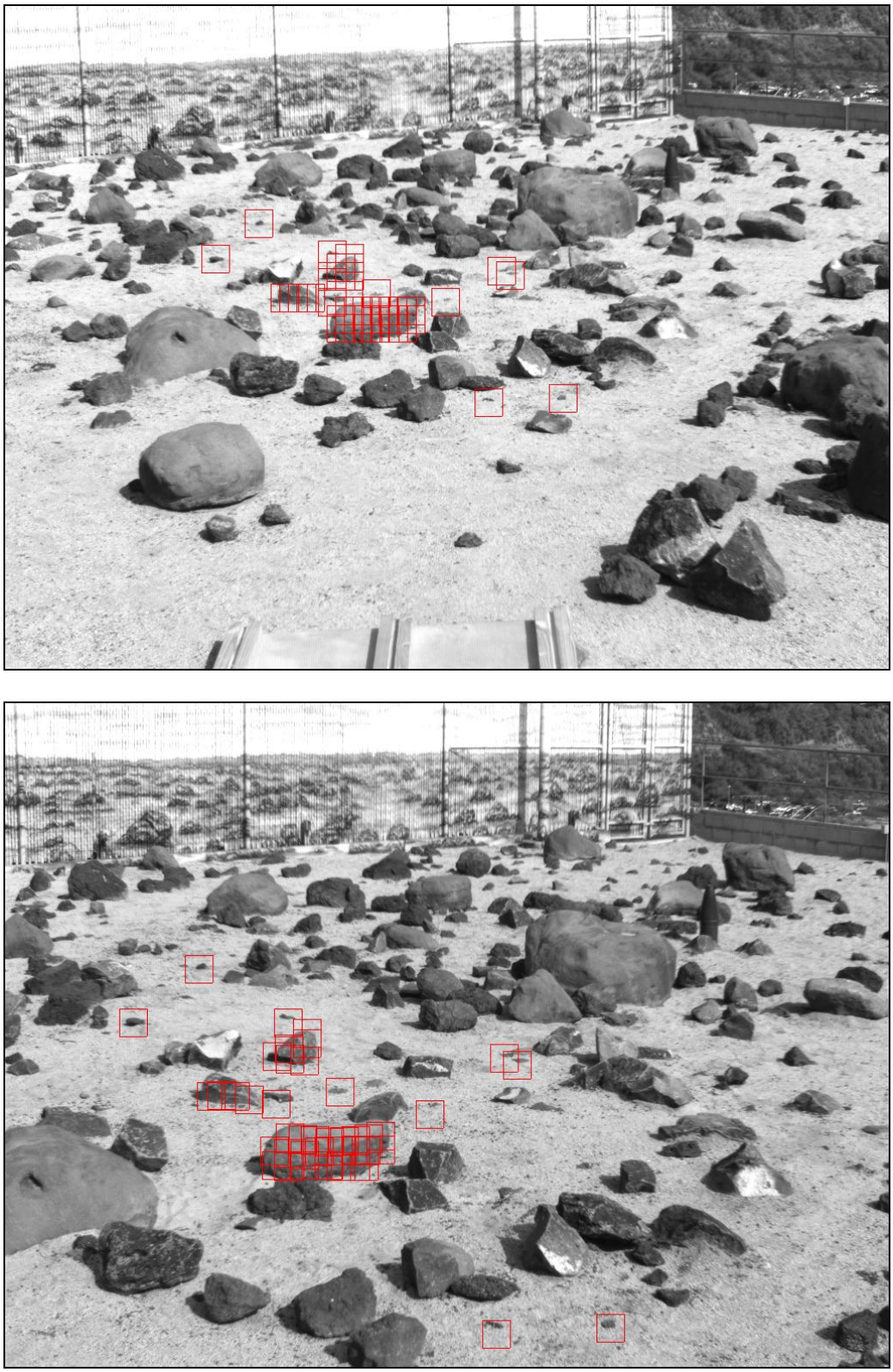
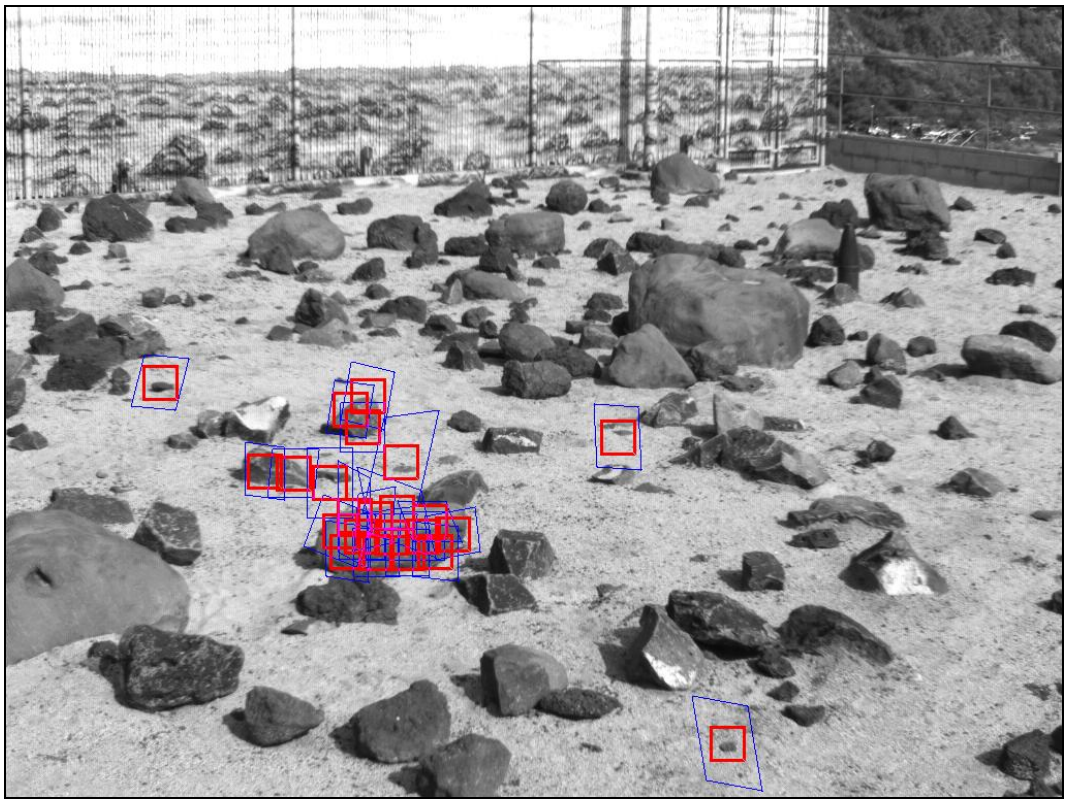
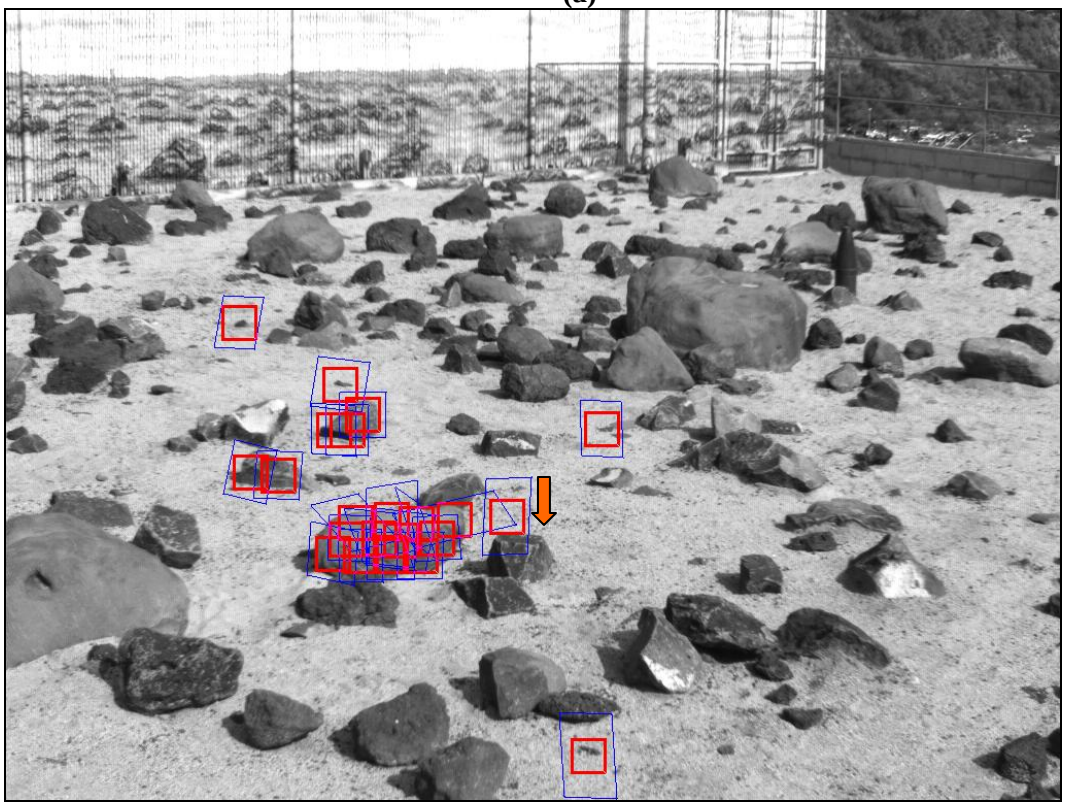


Figure 67. Beginning (top) and end (bottom) images of 45 targets for the 4-m forward motion with 8-mm lens after running SDOG tracker using 70% frame confidence and 80% template confidence thresholds: all 45 targets tracked.



(a)



(b)

Figure 68. End images of 45 targets for the 8-mm-lens forward motion after running affine tracker using 31×31 window with 2 pyramid levels: one target is lost.

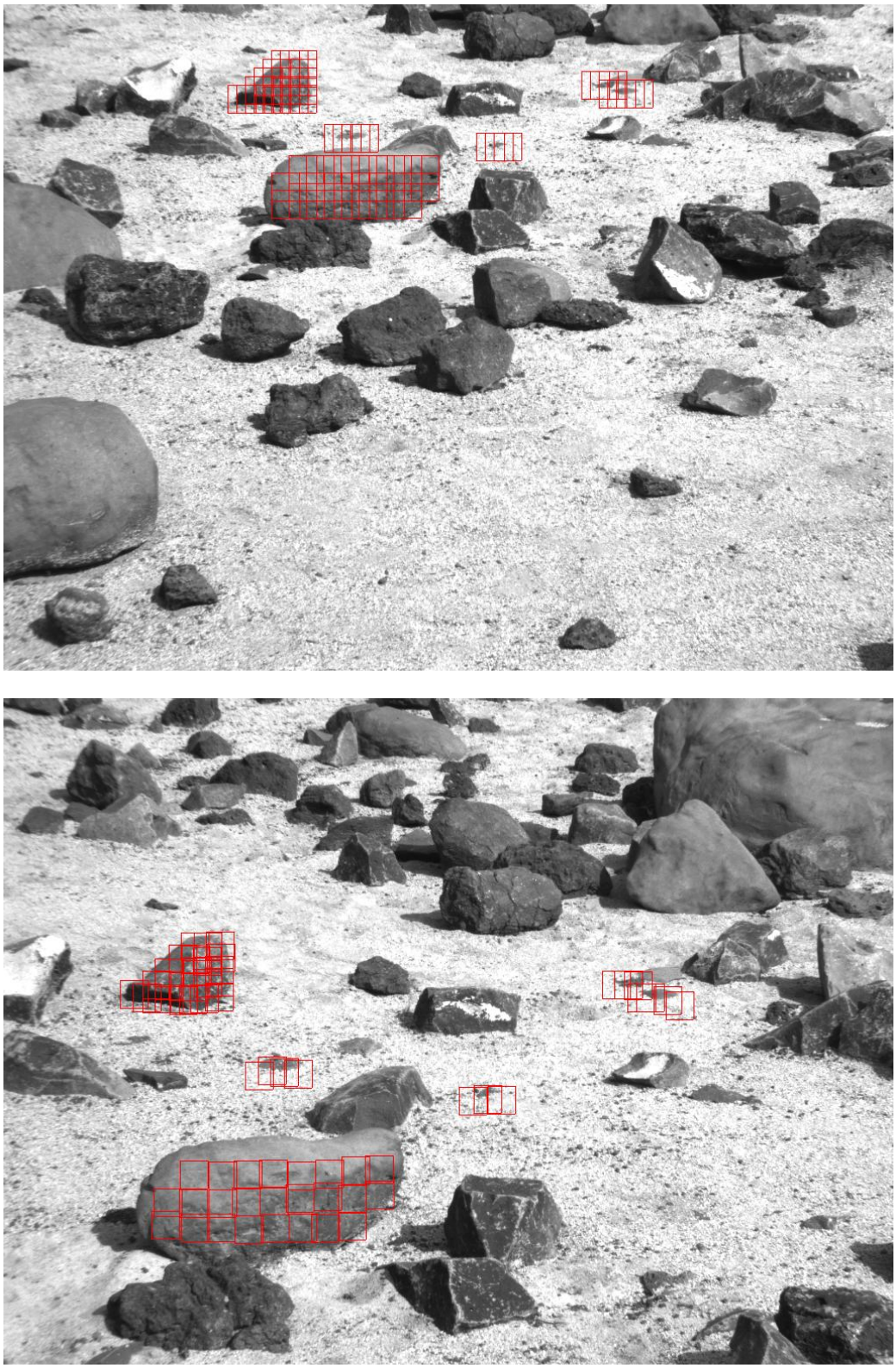


Figure 69. Beginning (top) and end (bottom) images of 65 targets for the 16-mm-lens forward motion after running SDOG tracker using 70% frame confidence and 80% template confidence thresholds: all 65 targets tracked.

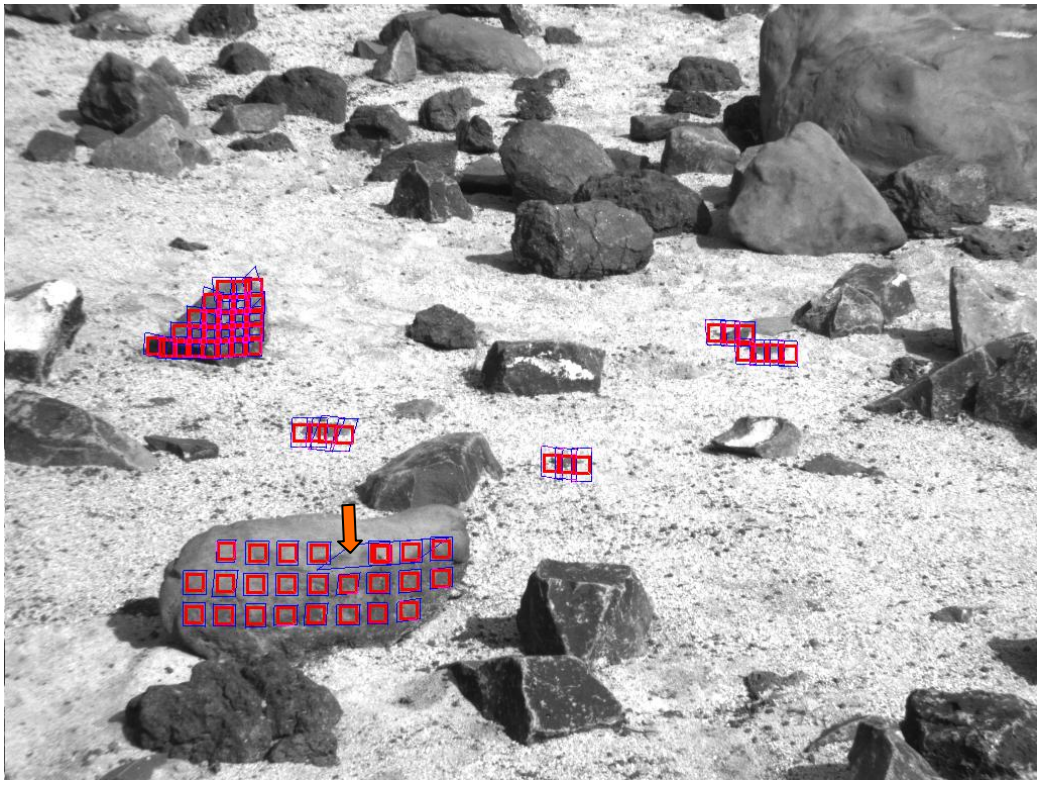
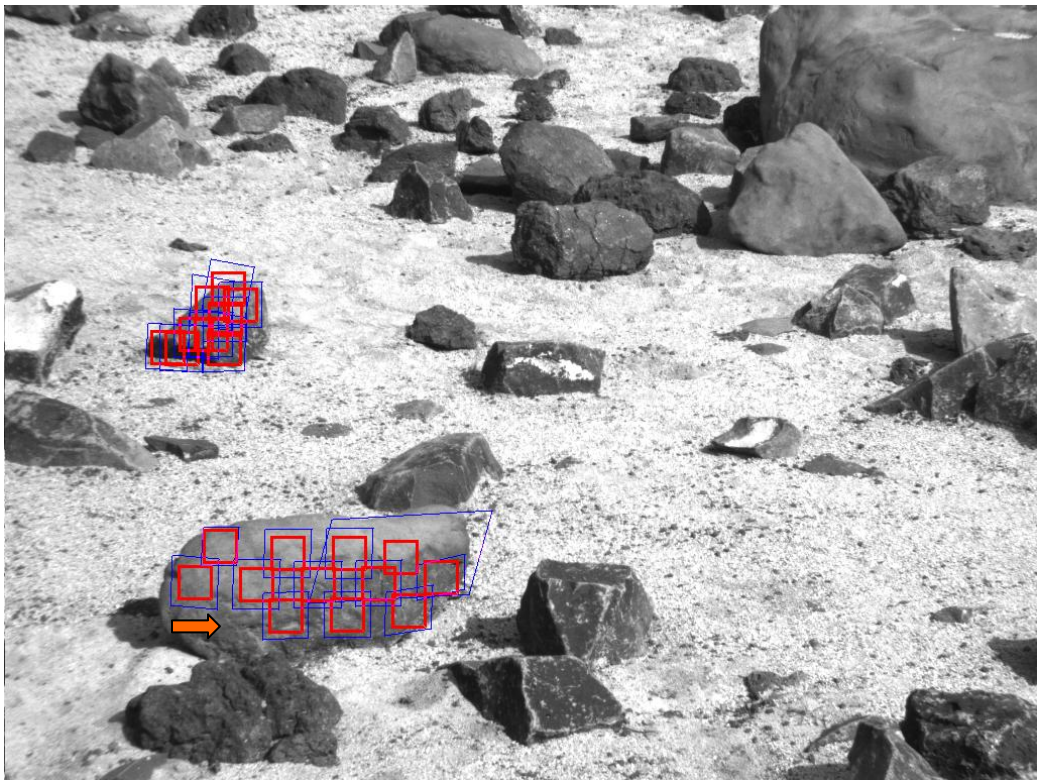
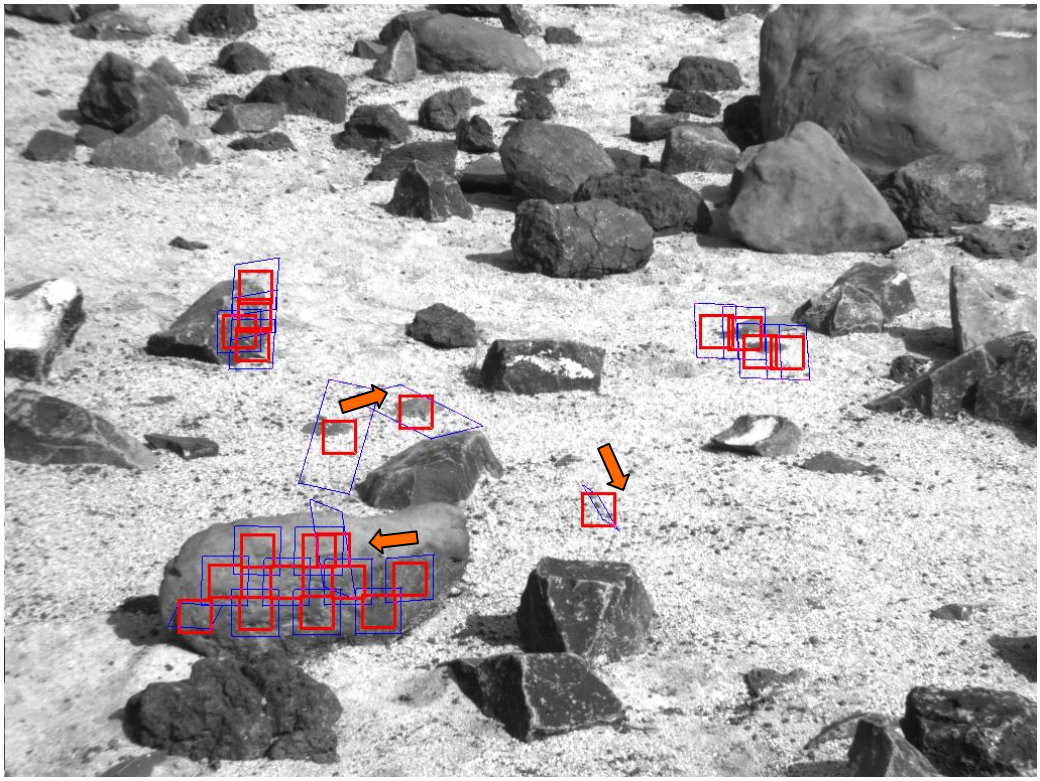


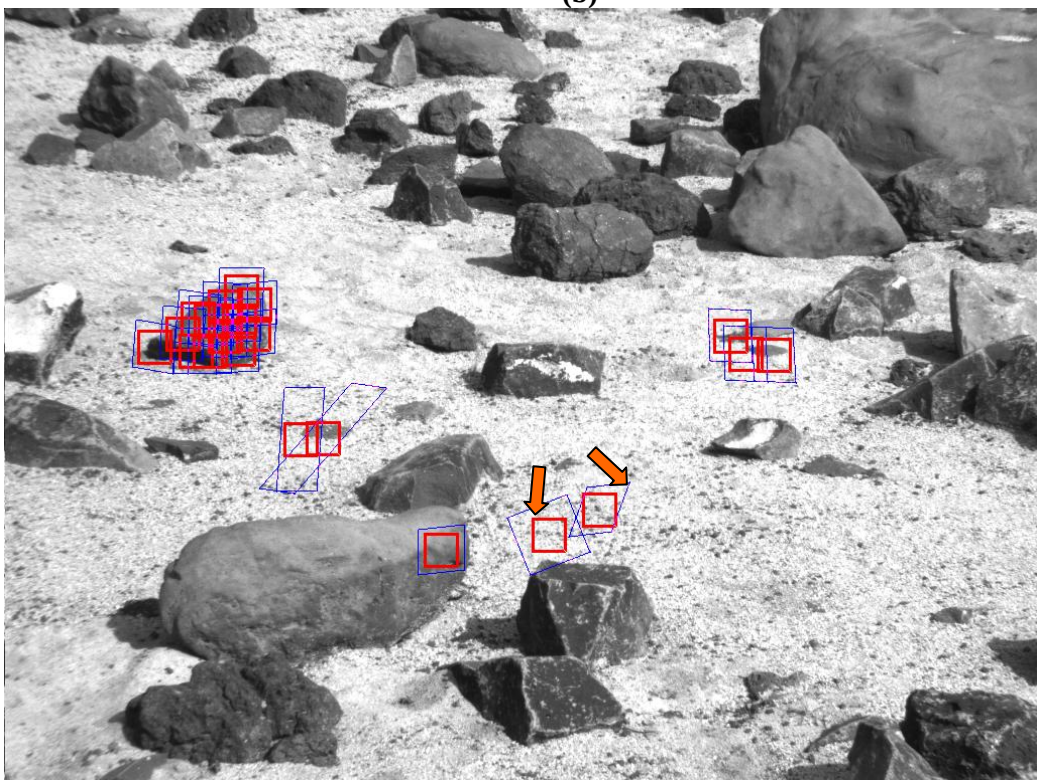
Figure 70. End images of 65 targets for 16-mm-lens forward motion after running affine tracker using 15×15 window with 3 pyramid levels: one target is lost because some low-texture targets need a larger-size window.



(a)



(b)



(c)

Figure 71. End images of 65 targets for 16-mm lens forward motion after running affine tracker using 31×31 window with 2 pyramid levels. Targets were partitioned into 3 groups for better observation: (a), (b) and (c). All 6 lost targets were caused by bad selection of target windows.

7.3 Roll and Yaw Motion Tracking

Tracking performances of the SDOG tracker were tested with roll and yaw camera motions by varying the frame and template confidence thresholds from 50% to 100% for every 10% step. The tracking results are shown in Table 15 for the 8-mm-lens roll motion and in Table 16 for the 8-mm-lens yaw motion. In these two tables, 60% frame confidence and 80% template confidence thresholds resulted in best tracking performance with 94% tracking for 8-mm-lens roll and 100% for the 8-mm-lense yaw. The 70% frame confidence and 80% template confidence thresholds, which yielded the best tracking performance for the forward motion, resulted in lower tracking performance with 56% for the 8-mm-lens roll motion and 100% for the 8-mm-lens yaw motion.

Poor tracking performance of the 8-mm-lens roll motion with the 70% frame confidence and 80% template confidence thresholds are shown in Figure 72. Five targets exhibited significant drift. This is understandable because the SDOG tracker does only pure translation matching.

The tracking performance comparison between the SDOG and affine trackers are shown in Table 17. The affine tracker yielded 100% tracking for both 8-mm-lens roll and yaw motions, demonstrating that the affine tracker performed slightly better than the SDOG tracker for the roll and yaw motions.

Template Confidence Threshold	Frame Confidence Threshold					
	50% (512)	60% (614)	70% (717)	80% (819)	90% (922)	100% (1024)
50% (512)	0%					
60% (614)	0%	0%				
70% (717)	31%	19%	0%			
80% (819)	56%	94%	56%	0%		
90% (922)	56%	88%	63%	69%	0%	
100% (1024)	63%	56%	63%	69%	0%	0%

Table 15. SDOG tracker tracking performance of 16 targets for 90º roll motion with 8-mm lens

Template Confidence Threshold	Frame Confidence Threshold					
	50% (512)	60% (614)	70% (717)	80% (819)	90% (922)	100% (1024)
50% (512)	19%					
60% (614)	19%	19%				
70% (717)	69%	69%	13%			
80% (819)	94%	100%	100%	100%		
90% (922)	100%	100%	100%	100%	0%	
100% (1024)	13%	0%	20%	0%	6%	0%

Table 16. SDOG tracker tracking performance of 16 targets for 45º yaw motion with 8-mm lens

Camera motion, Lens focal length	SDOG 32x32 FC=60%, TC=80%	SDOG 32x32 FC=70%, TC=80%	Affine 31x31 3 pyramid levels
Roll, 8-mm	94% (15/16)	56% (9/16)	100% (16/16)
Yaw, 8-mm	100% (15/16)	100%	100% (13/16)

Table 17. The affine tracker performed slightly better than SDOG tracker for the roll and yaw motions.

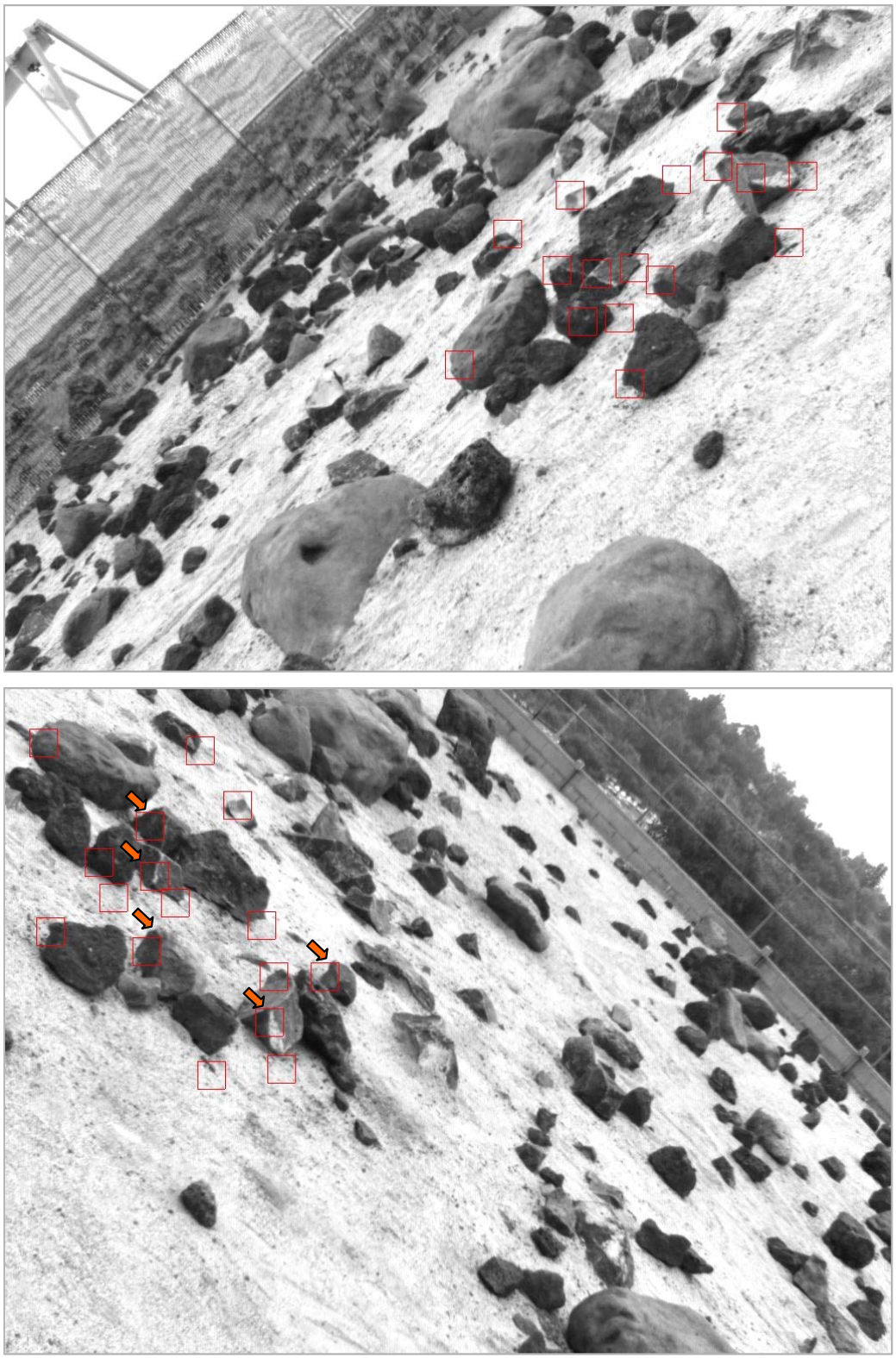


Figure 72. Beginning (top) and end (bottom) images of 16 targets for 8-mm-lens roll motion after running SDOG tracker using 70% frame confidence and 80% template confidence thresholds: 5 targets drifted a lot.

7.4 Lighting Changes

For the dramatic lighting changes, the SDOG tracker performed much better than the affine tracker as shown in Table 18. The image sequences used were the 5 PM data. During the 5 PM data collection, patches of opaque clouds happened to move across the sun, causing dramatic sunlight changes (Figure 35 of Section 6.6). Under these dramatic sunlight changes, the affine tracker performed very poorly (Table 10 of Section 6.6), yielding 0% tracking with 4-mm lens and 44% with the 8-mm lens, even with the AGC (automatic gain control) of the camera. The SDOG tracker yielded 100% tracking for both 4-mm and 8-mm lenses.

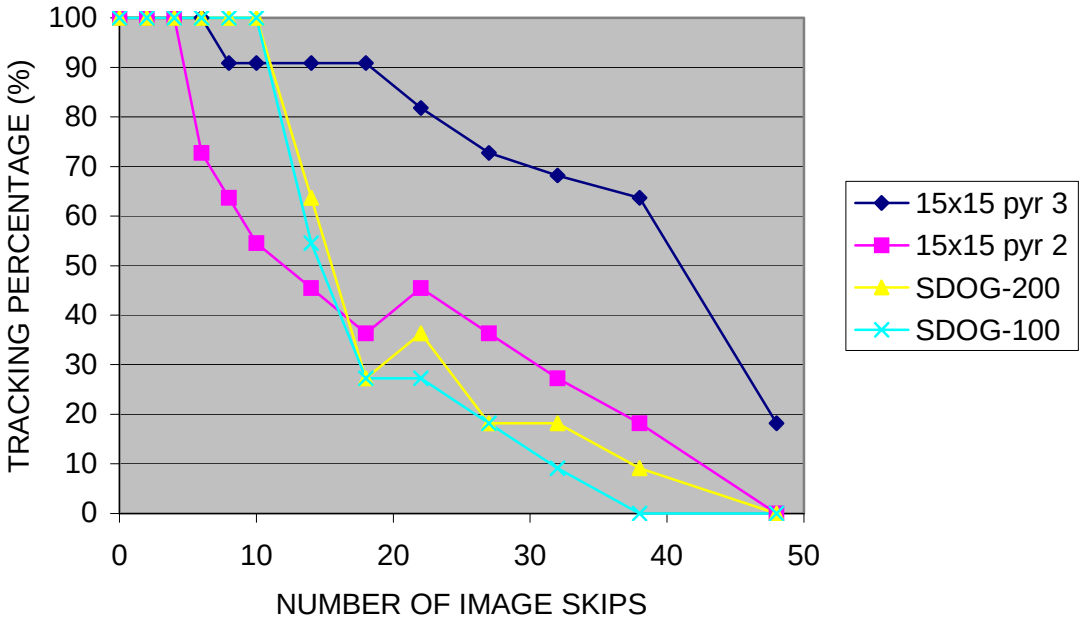
Since the SDOG tracker uses the difference of Gaussian operations which is essentially a band-pass filter suppressing DC and low spatial frequency components, it is quite robust to lighting intensity changes of DC spatial frequency components. By contrast, the current version of the affine tracker software simply relies on the camera AGC and does not compensate for the lighting changes.

Camera motion, Lens focal length	SDOG 32x32 FC=60%, TC=80%	Affine 29x29 3 pyramid levels
Forward, 4-mm; Cloudy	100% (10/10)	0% (0/10)
Forward, 8-mm; Cloudy	100% (18/18)	44% (8/18)

Table 18. SDOG tracker showed very robust tracking performance against lighting changes, while the affine tracker showed poor tracking performance.

7.5 Maximum Image Displacement for Active Camera Control

In order to determine the maximum image displacement of the SDOG tracker, we ran SDOG tracker with different image skips. Two different search window sizes of 100×100 pixels and 200×200 pixels were tested. Figure 73 shows the overall tracking percentages of the SDOG tracker for different image skips. The affine tracker results of Figure 33 in Section 6.5 were also overlaid for easy comparison. As in Figure 33, the maximum and average image displacements of the target images with respect to the number of image skips are shown right below the tracking percentage plot. For both window sizes of 100 and 200, the overall tracking percentage of the SDOG tracker stayed above 90% for the image skip of up to 10. The maximum image displacement at the image skip of 10 was 21 pixels, and the average was 8 pixels. When the number of image skips increased greater than 10, the overall tracking percentage dropped very sharply. This abrupt degradation was surprising that the SDOG tracker does brute-force search not iterative search. Two plausible causes of the abrupt degradation are: 1) use of signed binary image as opposed gray-scale image for matching and 2) high sensitivity to image size change. Three potential improvements are: 1) use of gray-scale DOG (difference of Gaussians), 2) use of down-sampled image (pyramidal image reduction), and 3) proper scaling of the matching image using the knowledge of the distance change to the target based on stereo vision triangulation. In terms of the tracking range of the image displacement, SDOG tracker was better than the affine tracker of the window size 15 with 2 pyramid levels, but worse than the affine tracker of the window size 15 with 3 pyramid levels. With improvements, we anticipate that a brute-force search method should probably be better than an iterative search, but more investigation is needed. We plan to perform further experiments in the next 2-D/3-D tracker test and validation.



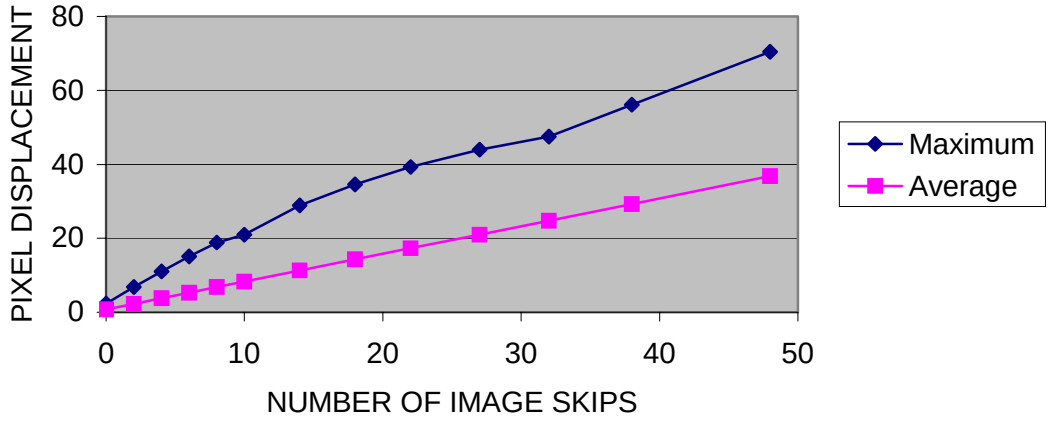


Figure 73. Overall tracking percentage with respect to the number of image skips for the SDOG tracker of search window size of 100 (cyan in top figure) and 200 (yellow in top figure) pixels as well as for the affine tracker of the window size 15 with 2 (pink in top figure) and 3 (blue in top figure) pyramid levels. The maximum and average image displacements with respect to the number of image skips are also shown right below the tracking percentage plot. The overall tracking percentage of the SDOG tracker dropped sharply when the number of image skips increased greater than 10.

7.6 Execution Timing

The matching computation times for the SDOG and affine trackers were measured using the Linux "time" command. In order to exclude the overhead time such as image read/display/write, the user CPU times were measured for two runs: one with 1 target and the other with 11 targets. Both runs tracked 50 images. The difference of the two CPU times divided by 500 (10 targets, 50 images) provided the matching computation time per target per frame. Similarly, matching computation times for the affine tracker were obtained and summarized in Table 19. The combined configuration of pure translation followed by affine matching took about 4 times longer than SDOG matching.

SDOG tracker, 32×32 window 96×96 search area	Affine tracker, 29×29 window with 3 pyramid levels		
	Pure translation	Affine alone	Combined
5 ms	5 ms	15 ms	20 ms

Table 19. Matching computation times for the SDOG and affine trackers

8. Improving Tracking Performance

Here is a list of items to be considered for potential improvement of tracking reliability

1. Restrict the selection of target windows that may cause tracking failures due to occlusions, two separate rocks within the target window, background interference, and shadow change.
   - Avoid the target window that may be occluded during the course of tracking
   - Avoid the target window that includes portions of two or more separate rocks
   - Avoid the target window that includes occluding boundaries with cluttered background
   - Avoid the target window that includes the area with large shadow change
2. Use a combined configuration of pure translation matching followed by affine matching. For pure translation matching, it is probably better to use a brute-force search not a iterative search to cover a large search space, although more investigation is needed. Examples of brute-force pure translation matching include:
   - SDOG matching
   - Normalized cross-correlation matching
   - 2-dimensional difference of Gaussian (DOG) filtered matching (2-D extension of the JPL Stereo correlation)
3. Use the difference of Gaussian (DOG) filter or the normalized cross-correlation to cope with poor lighting conditions and lighting changes. This applies to both pure translation and affine matching.
4. It is essential to have active camera control, for example, with 2-D/3-D tracking.
5. It is also essential to have camera view handoff.
6. It is critical to have reliable detection of target loss.
7. Instead of periodic template update by a fixed travel distance, update the template triggered by the thresholds of distance percent change, orientation change, and affine matching ssd (sum of squared differences).
8. Enable the affine matcher to handle the partial tracking window cropped by the image boundary. This allows higher pyramid levels with larger effective window sizes.
9. It is desirable to have an automated selection of the window size, number of pyramid levels, frame confidence and template confidence thresholds, and search window size.
10. It may be useful to have a selective region matching. This can be useful to track a low-texture target on a surface of a large-image-size rock by using a large window but avoiding nearby occluding boundaries. The target position may be off-centered in the target window. Examples of selective region matching include:
    - Quadrilateral window of general shape rather than an upright square window
    - Partially masked window to select only roughly planar portion of the target rock
    - Automatic generation of an off-centered, selective-region target window when a target is selected. The target may be low-textured and near the occluding boundary
11. Sophisticated target assessment and automatic adjustment of tracking window size, number of pyramid levels, and confidence thresholds. Minimum eigen values of a corner detector was not useful.

9. Conclusion

We evaluated the 2-D target tracking performance of two software packages: pyramidal Lucas-Kanade affine tracker and SDOG (sign of difference of Gaussian) tracker. Both were tested within the CLARAty (Coupled Layer Architecture for Robotic Autonomy) software environment. Since the 2-D tracking software did not support active camera control, we built and used a linear motion stage to collect camera images in incremental forward, roll, and yaw camera motions. Forward motion tests were limited to 4 m. We collected images with different camera focal lengths, different sunlight directions, and dramatic lighting changes.

The affine tracker experimental results indicated that pure translation matching tracked more reliably, while affine matching tracked more accurately. Not surprisingly, the combined tracking configuration of pure translation matching followed by affine matching performed best, and this combined configuration was thus used as the affine tracker in our tests. Average tracking percentages of the affine tracker were above 80% up to 100% when 15×15 window with 3 or 4 pyramid levels for 4-m straightforward, 90º roll, and 45º yaw camera motions. Most tracking failures in the above tests were related to low-texture targets on large rocks. Increasing the window size to 29×29 with 4 pyramid levels resulted in near 100% tracking, where identified tracking failures were occlusions, portions of two or more rocks within the target window, and shadow and lighting changes. Increasing the number of pyramid levels helped to track larger image displacements between images. The affine tracker tracking performance was fair with unfavorable sunlight directions, but poor with dramatic sunlight changes. The best template update was every 5% to 10% change in distance to target or every 5º to 10º change in roll and yaw motion. The maximum image displacement of the tracking range was about 30 pixels for the 15×15 window with 3 pyramid levels.

The SDOG tracker experimental results indicated that 60% to 70% for the frame confidence threshold and 80% for the template confidence threshold resulted in peak tracking percentage. SDOG tracker performed slightly better than the affine tracker for the forward motion, while the affine tracker performed slightly better than SDOG tracker for the roll and yaw motions. Unlike the affine tracker, SDOG tracker was robust for dramatic sunlight changes.

Potential improvements for tracking reliability include: 1) active camera control with 2-D/3-D tracking, 2) brute-force translation matching based on normalized-cross-correlation or DOG (difference of Gaussian) filter, 3) reliable target loss detection, 4) template update triggered by some thresholds, and 5) selective-region target window where the target may be off-centered. More conclusive experiments including full 10-m forward motion tests will be performed when the 2-D/3-D tracking software with active camera control is available.

References

M. Bajracharya, E. Bandari, M. Bualat, and I. Nesnas, "CLARAty Delivery for 2-D Visual Tracking," http://claraty.jpl.nasa.gov/vision/2d_tracking/index.html, April 2003.

M. Bualat and C. Kunz, "Ames Delivery for SDOG Visual Tracking," Feb. 2004.

T. Estlin, R. Volpe, I. Nesnas, D. Mutz, F. Fisher, B. Engelhardt, S. Chien, "Decision-Making in a Robotic Architecture for Autonomy." Proceedings of 6th International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS), Montreal Canada, June 18-21 2001.

W. Kim, R. Steinke, R. Steele, and A. Ansar, Camera Calibration and Stereo Vision Technology Validation Report, JPL D-27015, Revision1, Jan. 2004.

I. Nesnas, M. Bajracharya, R. Madison, E. Bandari, C. Kunz, M. Deans, M. Bualat, "Visual Target Tracking for Rover-based Planetary Exploration," IEEE Aerospace Conference, Big Sky, Montana, 2004.

I. Nesnas, R. Volpe, T. Estlin, H. Das, R. Petras, D. Mutz, "Toward Developing Reusable Software Components for Robotic Applications" Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Maui Hawaii, Oct. 29 - Nov. 3 2001.

A. Steltzner, "MSL Baseline Description Document: Surface System Overview," http://msl-lib/msl-lib/dscgi/ds.py/Get/File-3672/Surface_System_Overview.ppt, April 2003.

L. Tamppari, email communications, August 2003.

R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, "The CLARAty Architecture for Robotic Autonomy." Proceedings of the 2001 IEEE Aerospace Conference, Big Sky Montana, March 10-17, 2001.

R. Volpe, I.A.D. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, "CLARAty: Coupled Layer Architecture for Robotic Autonomy." JPL Technical Report D-19975, Dec 2000.

D. Wettergreen, H. Thomas, and M. Bualat, "Initial Results from Vision-based Control of the Ames Marsokhod Rover," IEEE International Conference on Intelligent Robots and Systems (IROS), Grenoble (France), pp. 1377-1382, Sep. 1997.