Information Sciences

Algorithm for Detecting a Bright Spot in an Image

Corrections for background intensity and dark current are included.

NASA's Jet Propulsion Laboratory, Pasadena, California

An algorithm processes the pixel intensities of a digitized image to detect and locate a circular bright spot, the approximate size of which is known in advance. The algorithm is used to find images of the Sun in cameras aboard the Mars Exploration Rovers. (The images are used in estimating orientations of the Rovers relative to the direction to the Sun.) The algorithm can also be adapted to tracking of circular shaped bright targets in other diverse applications.

The first step in the algorithm is to calculate a dark-current ramp — a correction necessitated by the scheme that governs the readout of pixel charges in the charge-coupled-device camera in the original Mars Exploration Rover application. In this scheme, the fraction of each frame period during which dark current is accumulated in a given pixel (and, hence, the dark-current contribution to the pixel image-intensity reading) is proportional to the pixel row number. For the purpose of the algorithm, the dark-current contribution to the intensity reading from each pixel is assumed to equal the average of intensity readings from all pixels in the same row, and the factor of proportionality is estimated on the basis of this assumption. Then the product of the row number and the factor of proportionality is subtracted from the reading from each pixel to obtain a dark-current-corrected intensity reading.

The next step in the algorithm is to determine the best location, within the overall image, for a window of $N \times N$ pixels (where N is an odd number) large enough to contain the bright spot of interest plus a small margin. (In the original application, the overall image contains 1,024 by 1,024 pixels, the image of the Sun is about 22 pixels in diameter, and N is chosen to be 29.)

The window is placed at a given position within the overall image. A weighted average of the intensities of the $4N - 4$ outer pixels of the window is taken as an estimate of background intensity and subtracted from a weighted average of the intensities of the remaining inner $(N - 2) \times (N - 2)$ pixels of the window to obtain a background-corrected weighted sum of pixel intensities for the window. The weighted averages are simply pixel-intensity averages multiplied by common denominators so as to obviate floating-point arithmetic operations and thereby accelerate computations. The window is then moved to an adjacent column position, and weighted averages for the new position are calculated from the previous weighted averages by adding the appropriate values for the new outer and inner pixels and subtracting the corresponding values for the pixels that have been left behind or changed in status between the inner and the outer. This process is repeated until the computations have been performed for all possible window positions. The position that yields the highest background-corrected weighted sum of pixel intensities is assumed to contain the bright spot of interest (the image of the Sun in the original application), and the window is then used to locate the bright spot more precisely as described next.

Within the inner $(N - 2) \times (N - 2)$ portion of the window, the position of the bright spot is determined by means of a simple centroid calculation, using the background-corrected pixel intensities. Because the window position selected as described above may not necessarily be the optimum one, the centroid calculation is performed twice in an iterative process: For the second centroid calculation, the window is re-centered on the centroid determined by the first centroid calculation.

This work was done by Carl Christian Liebe of Caltech for NASA's Jet Propulsion Laboratory.

The software used in this innovation is available for commercial licensing. Please contact Karina Edmonds of the California Institute of Technology at (626) 395-2322. Refer to NPO-41801.

Extreme Programming: Maestro Style

Modifications have been made to suit a specific development environment.

NASA's Jet Propulsion Laboratory, Pasadena, California

"Extreme Programming: Maestro Style" is the name of a computer-programming methodology that has evolved as a custom version of a methodology, called "extreme programming" that has been practiced in the software industry since the late 1990s. The name of this version reflects its origin in the work of the Maestro team at NASA's Jet Propulsion Laboratory that develops software for Mars exploration missions.

Extreme programming is oriented toward agile development of software resting on values of simplicity, communication, testing, and aggressiveness. Extreme programming involves use of methods of rapidly building and disseminating institutional knowledge among members of a computer-programming team to give all the members a shared view that matches the view of the customers for whom the software system is to be developed. Extreme programming includes frequent planning by programmers in collaboration with customers, continually examining and rewriting code in striving for the simplest workable software designs, a system metaphor (basically, an abstraction of the system that provides easy-to-remember software-naming conventions and insight into the architecture of the system), programmers working in pairs, adherence to a set of coding standards, collaboration of customers and programmers, frequent verbal communication, frequent releases of software in small increments of development, repeated testing of the developmental software by both programmers and customers, and continuous interaction between the team and the customers.

The environment in which the Maestro team works requires the team to quickly adapt to changing needs of its customers. In addition, the team cannot afford to accept unnecessary development risk. Extreme programming enables the Maestro team to remain agile and provide high-quality software and service to its customers. However, several factors in the Maestro environment have made it necessary to modify some of the conventional extreme-programming practices. The single most influential of these factors is that continuous interaction between customers and programmers is not feasible. The major resulting differences between the Maestro and conventional versions of extreme programming are the following:

- Because customers are not always available for planning sessions, members of the team act on behalf of customers during these sessions.
- In an elaboration of the frequent-planning and incremental-release concept, releases and planning meetings are synchronized with a fixed one-week iteration cycle that facilitates maintenance of focus on the development task.
- Metaphors are occasionally used as needed in specific instances, but the conventional extreme-programming concept of a system metaphor is abandoned as not being helpful.
- In a departure from the simplest-design rule, the team sometimes develops software infrastructure that affords capabilities, beyond those required in the current iteration, that may be useful later in the development process.
- In the absence of continuous involvement of customers and of frequent testing of software by customers, there is heavy reliance on automated testing.

This work was done by Jeffrey Norris, Jason Fox, Kenneth Rabe, I-Hsiang Shu, and Mark Powell of Caltech for NASA's Jet Propulsion Laboratory. Further information is contained in a TSP (see page 1).

The software used in this innovation is available for commercial licensing. Please contact Karina Edmonds of the California Institute of Technology at (626) 395-2322. Refer to NPO-41811.

Adaptive Behavior for Mobile Robots

A robotic system attempts to both preserve itself and progress toward a goal.

NASA's Jet Propulsion Laboratory, Pasadena, California

The term "System for Mobility and Access to Rough Terrain" (SMART) denotes a theoretical framework, a control architecture, and an algorithm that implements the framework and architecture, for enabling a land-mobile robot to adapt to changing conditions. SMART is intended to enable the robot to recognize adverse terrain conditions beyond its optimal operational envelope, and, in response, to intelligently reconfigure itself (e.g., adjust suspension heights or baseline distances between suspension points) or adapt its driving techniques (e.g., engage in a crabbing motion as a switchback technique for ascending steep terrain). Conceived for original application aboard Mars rovers and similar autonomous or semi-autonomous mobile robots used in exploration of remote planets, SMART could also be applied to autonomous terrestrial vehicles to be used for search, rescue, and/or exploration on rough terrain.

In SMART, controlling the motion of the robot, managing the "health" of the robot, and managing resources are considered as parts of a free-flow behavior hierarchy that autonomously adapts to changing conditions. Tasks that must be performed in the continuing development of SMART are to provide for safe, adaptive mobility on highly sloped terrain include:

- Determination of strategies for adaptive reconfiguration and driving that are nearly optimal with respect to safety and are computationally feasible for on-board implementation,
- Determination of a representation for uncertainty in sensing and prediction of the state of the robot and its environment, and
- Determination of resource-management strategies that mitigate such risks as those of the loss of battery power and/or drive motors.

SMART is based largely on a prior architecture denoted Biologically Inspired System for Map-based Autonomous Rover Control (BISMARC), which, in turn is based on a modified free-flow hierarchy. BISMARC has been used with success in a number of different simulated mission scenarios, wherein it has been demonstrated to afford capabilities for retrieving objects cached at multiple locations, fault tolerance on missions of long duration, and preparing terrain sites for habitation by humans. BISMARC includes provisions for all aspects of safety, self-maintenance, and achievement of goals, as needed to support a sustained presence on the surface of a remote planet.

BISMARC is organized as a two-level system. From stereoscopic images acquired by cameras aboard the robot, the first level generates hypotheses of motor actions. The second level processes these hypotheses, coupled with external and internal inputs, to generate control signals to drive the actuators on the robot.

The figure illustrates the free-flow action-selection hierarchy of BISMARC and SMART. The rectangular boxes represent behaviors, while the ovals represent sensory inputs (either fixed, direct, or derived). At the top are the high-level behaviors, including Don't Tip Over, Go to Goal, Avoid Obstacles, Preserve Motors, Warm Up, Get Power, and Sleep at Night. The intermediate-level behaviors (Change Center of Gravity, Avoid Obstacles, Rest, and Sleep) are designed to interact with both the short-term memory (which corresponds to perceived sensory stimuli), and the long-term memory (which encodes remembered sensory information). Control loops are prevented by use of temporal penalties, which constrain the system to repeat a given behavior no more than a predetermined number of times. The bottom-level behaviors (Tilt Arm, Change Shoulder Angles, Move, Rest, Stop, Sleep) fuse the sensory inputs and the activations of the higher-level behaviors in order to select appropriate actions for safety and achieving goals.