# Visual mini-robot identification, tracking and control

Ken Tossell, Andy Hammond, Eduardo Arvelo, Nuno C. Martins

***Abstract*—In order to facilitate motion control for a swarm of inch-scale robots that have limited sensing, computation and communication capabilities, we have designed a software tool that locates and tracks robots using overhead imagery. The software follows robots as they move about a planar surface, providing position information suitable for use in centralized or cooperative motion planning.**

***Index Terms*—Image motion analysis, mobile robot motion-planning, robot vision systems.**

## I. Introduction

ADVANCEMENTS in small-scale fabrication technology have made possible a new class of ultra-miniature robots. These microscale robots show promise in applications such as search and rescue, infrastructure monitoring, and medicine. Microrobots are particularly useful components in search applications because of their ability to gain access to denied environments inconspicuously. We anticipate the emergence of microscale medical robots that would be capable of minimal-risk microsurgery and medication delivery.

Our work is part of a larger-scale testbed for research into microrobot control and coordination algorithms. We have designed a tracking system and low-level control software for inch-scale robots (“bristle-bots”[1]); it is the central component of the testbed, which will be used to evaluate algorithms that are intended for eventual implementation on microscale legged robots.

Our group is investigating robot-control systems of various types, including both autonomous and centrally controlled microrobots, with environmental and positional information coming from a central, fixed-position observer (e.g., camera-based methods, medical imaging, or GPS and other fixed-reference-point geolocation techniques), from the robots themselves (e.g., intraswarm radio distance sensing based on time difference of arrival or auditory short-range robot-robot identification), or using a combination of single-source and agent-supplied data.

### A. Motivation

The inspiration to design a visual tracking system for these robots arose from a need to be able to locate and follow microrobots without relying greatly on the robots’ assistance in gathering motion and position data. Once the robots were located and identified, visual tracking data would be used in control algorithms to coordinate a fleet of robots. The tracking system would provide position information with respect to a fixed reference frame; before sending commands and neighbor-position data to a robot, the software would have the option of translating all coordinates into a robot-centric coordinate system, simplifying onboard computation for the robots. This visual system, unlike some currently employed tracking mechanisms [1], would not require active participation by the robots to a large extent. This would serve to reduce power consumption and requirements for computational power, which could be channeled to other activities that the robot would have to carry out. The software presented in this paper meets the aforementioned specifications by locating robots in a video stream, determining each robot’s unique identity using physical or behavioral observation, and tracking individual robots’ motion while maintaining an accurate estimate of each robot’s position and orientation.

### B. Testbed

The bristle-bot testbed consists of a 5 $m^2$ platform, an overhead camera, and a central computer, which is connected to each of the robots wirelessly. The platform’s flat, glossy surface allows the robots to move about freely, with a minimal number of disruptive physical and visual artifacts. The camera captures 38 1-megapixel color images per second, with a resolution of 6 pixels per cm. The main computer is a Core i7-based machine with an NVIDIA GeForce graphics processing unit and 12 GiB of system memory.

During initial vision development, we have used toy “robots” that move in a fashion similar to bristle-bots but offer no control capabilities. The vision subsystem is tuned to follow these devices as they move randomly on the platform. We expect to find that motion models developed for these toy robots will be applicable to controllable bristle-bots.

### C. System Overview

The centralized portion of the system has been separated into three primary components: vision, tracking, and control (Fig. 1). Each component subscribes to and publishes one primary data stream; additionally, each component maintains a cyclic data sharing relationship with the succeeding part of the system.

The software has been developed using ROS[2], which provides drivers for cameras and other devices, support for

Received August 6, 2010. This material is based upon work supported by the National Science Foundation under grants 0931878 and 0755224.

Ken Tossell is a senior computer science student at the University of Maryland (e-mail: ktossell@umd.edu).

Andy Hammond is a senior electrical engineering student at the University of Maryland (e-mail: ahammond@umd.edu).

[1]Bristle-bots are small robots made using toothbrush heads and vibrating motors. When a motor vibrates, it shifts the robot downward, and the rear-angled bristles bend, forcing the robot away from the bristle-ground intersection and causing the body of the robot to move forward. Using independent left-side and right-side motors, each attached to its own toothbrush head, the bristle-bots are able to move in any forward direction.

[2]ROS (Robot Operating System) is an open-source meta-operating system for mobile robotics: http://www.ros.org/

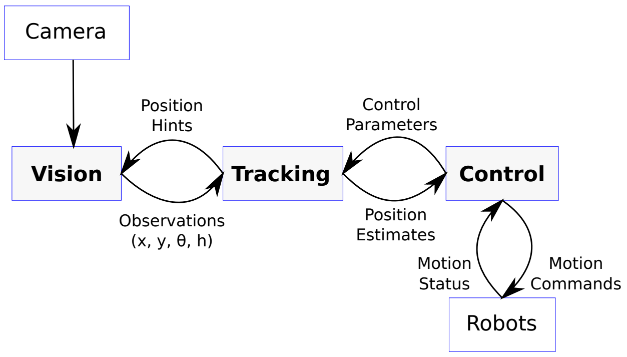


Fig. 1. Main System Components

interprocess communication, and tools for image processing [2].

## II. Vision Subsystem

We use a two-part vision process, first performing a full-image search for regions of likely robot activity by finding moving objects. Once this list of regions has been compiled, we combine it with a list of regions where robots are known to have existed recently. The second vision step – local inspection within each of the regions – finds individual robots, delivering precise position, orientation and color information to the tracking subsystem.

### A. Global Search

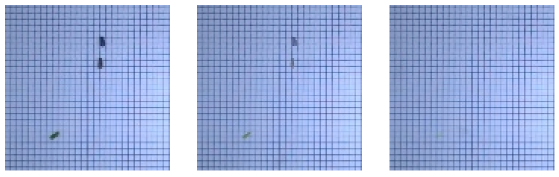
A first-level processor identifies regions of interest using a combination of short- and long-term motion detection [3]. The software maintains a slow-adapting visual model of the platform; this model is used to find regions whose appearances are historically unusual. The system takes its collected knowledge of robot positions into account when it updates the model, so robots that have come to a stop will not be reabsorbed into the background model. The software creates a map of the pixels that meet a threshold value for their color difference from the background model and finds connected components, which represent moving objects. Components that are significantly too large or too small to be robots are ignored in order to avoid triggering the small-area image processor on camera noise or observed operator intervention. The long-term background model and foreground separation appear in Fig. 2 and are described by the following equations:

$$B_t^c(x,y) = \begin{cases} B_{t-1}^c(x,y)+1 & \text{if } B_t^c(x,y) > B_{t-1}^c(x,y) \\ B_{t-1}^c(x,y) & \text{if } B_t^c(x,y) = B_{t-1}^c(x,y) \\ B_{t-1}^c(x,y)-1 & \text{if } B_t^c(x,y) < B_{t-1}^c(x,y) \end{cases} \tag{1}$$

$$\forall \; c \in \{\text{red}, \text{green}, \text{blue}\}; 0 <= B_t^c(x,y) <= 255.$$

$$slowForeground(x,y) = \begin{cases} 1 & \text{if } \sum_{c\in\{r,g,b\}} |B_t^c(x,y)| \geq k \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $B$ is the three-color slow-adapting background model, $k$ is a constant threshold, and $(x, y)$ is the location of a pixel in the image.



(a) Initial image (b) Foreground fades (c) Fully separated

Fig. 2. Updating the long-term background model, beginning at $t = 0$

The image is next processed with a short-term motion detector, finding image positions whose color values have shown significant changes since the previous frame. Any long-term foreground blob (an area of pixels that do not match the slow-adapting background model) that does not contain enough short-term foreground pixels will be ignored. This keeps the system from searching repeatedly in a region that neither matches the long-term background nor contains any robots, such as a region before the introduction, or after the removal, of an obstacle or an operator's arm.

The software creates a list of square blob-bounding regions big enough to ensure that each region contains an entire robot. This list – the list of observed regions – is combined with the list of regions enclosing the estimated locations of known robots – the list of predicted regions – to form a set of areas that need further testing. The system combines regions with significant overlap in order to avoid having a robot lie along a region boundary, where it is likely to be misdetected.

### B. Local Search

The second-level processor receives the list of regions; it scans each region individually, searching for individual robots. The early-stage robots appear essentially as dark near-rectangular shapes, in contrast with the white background. In order to avoid building a recognition system that is overly adapted to these temporary robots, we have used a simple pixel intensity filter. For each region, we calculate the mean $\mu$ and variance $\sigma^2$ of grayscale intensity (including non-robot pixels), and we take the pixels that have intensity at most $\mu - 1.5\sigma$ as positive pixels (Fig. 3). This threshold is a heuristic value that has been found to be effective. To remove stray dark pixels, we perform an erosion operation on the resulting bitmap. A dilation follows, restoring the original bitmap less stray pixels.

Another round of connected component analysis separates the robots when more than one robot appears within a region. The software finds each blob's average color in RGB color space and stores the hue of that color. It continues by determining the convex hull for each labeled component and finding minimal bounding rectangles for those enclosures 4.

Any rectangle that fits within a predefined range for proportion and area is accepted as a robot. The vision system collects the robots and passes them to the tracking and identification system in the form $(x, y, \theta, h)$, where $x$ and $y$ represent the center point of the robot, $\theta$ is an angle parallel to the longer

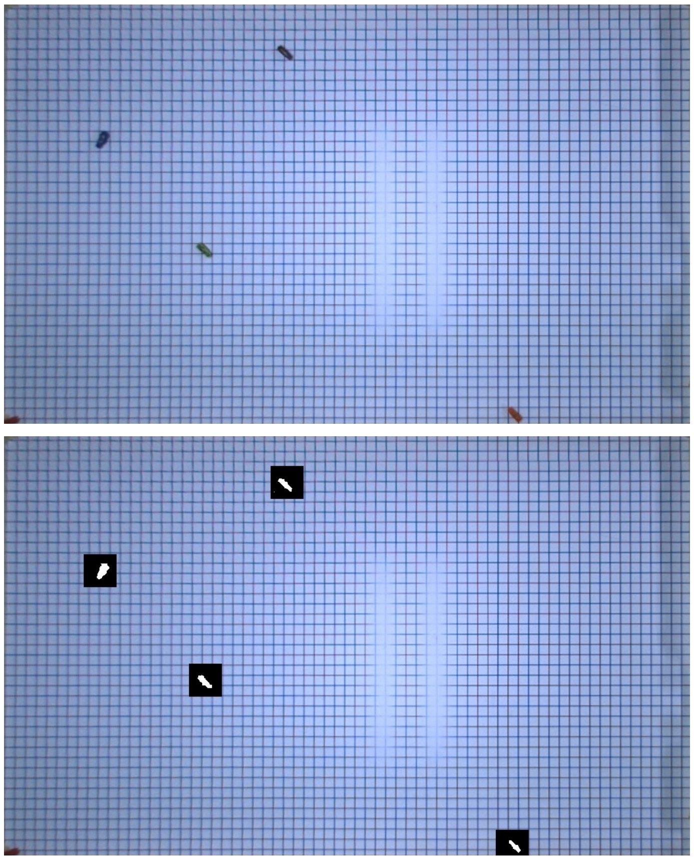


Fig. 3. Intensity-based local search.

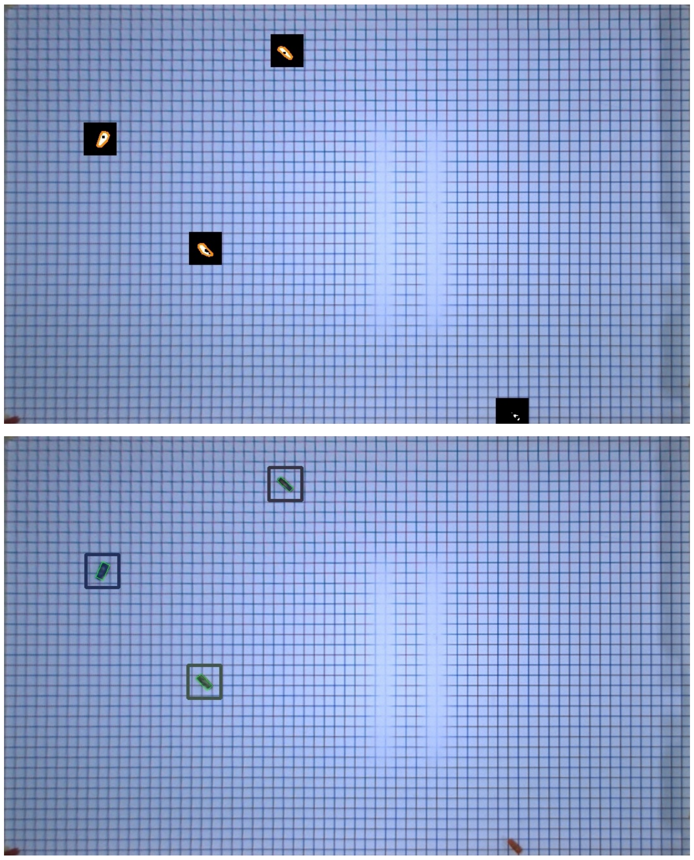


Fig. 4. Convex hull and bounding rectangle based on bitmap (Fig. 3).

axis (either forward or backward), and $h$ is the average hue in radians.

## III. Tracking Subsystem

The tracking subsystem uses information from the vision system together with internal mathematical models to answer two questions: Given a robot $(x, y, \theta, h)$, how has it moved since the previous frame (or further back in the recent past)? And, knowing what path that robot has followed, what is the robot's identity?

We split this problem into two parts. First, we maintain a position estimate for each robot. At present, we estimate a robot's position using an $(x, y, \theta)$-tracking Kalman filter. Once we learn more about the robot's motion characteristics, we will expand this so that the system might accurately reflect the nonlinear motion that the bristle-bots are known to employ. This future estimator will track roughly the values listed in Fig. 5, along with their first and second derivatives.

### A. Matching From Frame To Frame

Our most basic matching task is the job of determining a mapping from the set of robots observed in one video frame to the corresponding robots in the next frame. For this, we use an algorithm (Algorithm 1) that weighs Euclidean distance and hue variation to construct the desired mapping while marking new robots (such as those robots that are moving for the first time) and missing robots (those that were removed from the

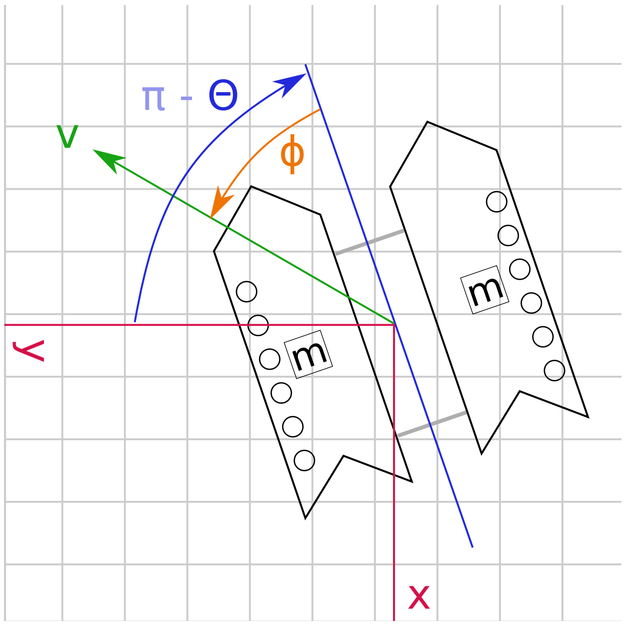


Fig. 5. Possible model of bristle-bot state variables

test area and any robots missed due to failure of the vision subsystem).

Initial tests have shown that this matching procedure yields correct results in conditions ranging from ideal configurations (large distances between robots) to collisions. In the latter case (and in non-colliding close passes), the Kalman filter's resistance to abrupt change, combined with color differentiation, most often enables the tracking subsystem to follow multiple robots through a collision without any identity swapping.

### *B. Matching Virtual Robots To Real-World Robots*

Robots may appear or vanish at any time through operator intervention or due to vision system error, but the true set of robots – the robots that are radio-connected and known to the controller – does not change. Our software must handle any situation in which there is a subset of robots that do not have a bidirectional virtual-to-physical mapping. Since the tracking subsystem typically knows nothing about the state of a robot when that robot first appears, it must use a method, such as one of the procedures described below, to determine with certainty which modeled robot corresponds to a particular real-world robot.

*1) Static Matching:* In the static approach, the tracking system cooperates with the vision system once again, using visual cues to uniquely identify unmapped robots. The average color of a robot is one such cue; other markings and patterns would assist this process. The major advantage of this type of matching procedure is the possibility of running it on every frame, constantly confirming or revising the virtual-to-physical map. The most basic implementations would severely limit the number of usable robots, however, due to the limitation in easily distinguishable colors and patterns. Additionally, this requires a predefined list of robots; new agents cannot as easily join the swarm. We have implemented a basic form of color-based identification using a five-entry robot list and have encountered only infrequent mismatch, lasting no longer than a few frames.

*2) Dynamic Matching:* For dynamic matching, future versions of the tracking and control subsystems will coordinate to identify any unmapped robots through a process of elimination. The software would instruct each of the robots to go in a particular direction – either right (group R) or left (group L) – for a period of time. After visually observing which unmapped robots had moved in which direction, it would have half of the left-commanded robots go left (LL), with the other half going right (LR). The right-commanded robots would undergo the same procedure. The software would continue to divide the groups of unmapped controllable robots until it had reduced each group to a single robot (LRRL $= 0110_2 =$ subject #6) and become certain of which internal robot model would map to which physical robot. This would normally complete an initial identification procedure for a field of $n$ robots with $t$-second observation periods within $t * log_2(n)$ seconds.

We will also investigate two other types of dynamic matching. In one, the robots themselves would choose a direction of motion, sending the chosen angles back to the tracker and controller, which would narrow the set of possible mappings as above. In another, the software would stop all robots on the field and then instruct one robot to move at a time.

## IV. Control Subsystem

We are in the process of developing on-robot software for low-level motion control. We plan to use the visual tracking system to evaluate the robots' motion across the full range of motor speeds; we will use the resulting information to find an accurate transformation between motion goals $(v, \phi)$ (see Fig. 5) and motor speeds, controlled as duty cycles $(D_L, D_R)$.

The bristle-bots will communicate with the control machine using a short-range wireless link, sending status information and receiving messages such as direct motion commands $(v, \phi)$, robot location updates (position of a particular robot or of all robots $\{(x, y, \theta)\}$), navigation goals $(x, y)$ (to be met using on-robot processing of centrally collected self- and neighbor-position information) and paths for the robots to follow (e.g., lists of spline curves and velocity functions).

Future work in the area of controls will include the development of swarm motion planning software using receding horizon control.

## References

[1] Youngjoon Han, Moonyong Han, Hyungtae Cha, Mincheol Hong, and Hernsoo Hahn, "Tracking of a moving object using ultrasonic sensors based on a virtual ultrasonic image," *Robotics and Autonomous Systems*, vol. 36, no. 1, 31 July 2001, pp. 11-19.

[2] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *International Conference on Robotics and Automation,* ser. Open-Source Software workshop, 2009.

[3] S.-C. S. Cheung, C. Kamath, "Robust background subtraction with foreground validation for urban traf.c video," *EURASIP Journal on Applied Signal Processing*, 2005:14, 2330.2340.

## Appendix

**Algorithm 1** Robot-robot mapping

```
New $\leftarrow \varnothing$
Unmatched $\leftarrow$ Observed
**while** Unmatched $\neq \varnothing$ **do**
  $obs \leftarrow$ remove_from Unmatched {randomized}
  $best\_match \leftarrow \Diamond$
  $best\_distance \leftarrow 100^2$ {at most 100px away}
  **for** $i = 0$ to len(Known) $-1$ **do**
    $candidate \leftarrow$ Known$[i]$
    $est \leftarrow$ posn_est $candidate$ {Kalman filter}
    $distance \leftarrow (obs.x - est.x)^2 + (obs.y - est.y)^2$
    {probability that the candidate would have this color, constructed using difference from running average or using histogram under KDE:}
    $distance \leftarrow distance/($prob_color $candidate\ obs.hue)$
    **if** $distance < best\_distance$ **then**
      **if** $candidate.best\_obs == \Diamond$ or $candidate.best\_dist \geq distance$ **then**
        $best\_match \leftarrow candidate$
        $best\_distance \leftarrow distance$
      **end if**
    **end if**
  **end for**
  **if** $best\_match \neq \Diamond$ **then**
    **if** $best\_match.best\_obs \neq \Diamond$ **then**
      add_to Unmatched $best\_match.best\_obs$ {send suboptimal match back}
    **end if**
    $best\_match.best\_obs \leftarrow obs$
    $best\_match.best\_dist \leftarrow best\_distance$
  **else**
    add_to New $obs$ {$obs$ is a new robot}
  **end if**
**end while**
```